Backend Features - Banking App - FS Java Capstone

Backend Features

- Account Management - Backend Features - Banking App - FS Java Capstone
- Card Management - Backend Features - Banking App - FS Java Capstone
- User Management - Backend Features - Banking App - FS Java Capstone
- Loan Management - Backend Features - Banking App - FS Java Capstone
- Branch Management - Backend Features - Banking App - FS Java Capstone
- Transaction Management - Backend Features - Banking App - FS Java Capstone

Card Management - Backend Features - Banking App - FS Java Capstone

Card Management

Story	Description	Acceptance Criteria	Story Points
Card Creation			
Manually Create Card	As a user, I need the ability to manually create a card in the banking app's backend system, ensuring that the process is secure, reliable, and integrates seamlessly with the existing database.	1. **Functionality** ◦ API endpoint to create a new card record. ◦ Ability to associate the card with a user account. ◦ Support for different card types (e.g., debit, credit). 2. **Security** ◦ Authentication: Ensure only authorized users can access the API. ◦ Authorization: Validate user permissions for card creation. ◦ Encryption: Secure transmission of card and user data. 3. **Validation** ◦ Input Validation: Check for valid card details (e.g., card number format, expiration date). ◦ Ensure mandatory fields (like card type, user ID) are not empty. 4. **Error Handling** ◦ Implement try-catch blocks to handle unexpected errors. ◦ Provide meaningful error messages for client-side handling. ◦ Ensure the backend does not crash on encountering errors. 5. **Database Updates** ◦ Update the user record with the new card details. ◦ Ensure atomic transactions to maintain data integrity.	

		◦ Log creation date and time of the card. 6. **Output/Notification** ◦ Return a success message with the card ID upon successful creation. ◦ In case of failure, return an appropriate error message. 7. **Logging** ◦ Log all requests to create cards, including timestamp and user ID. ◦ Record successful and failed attempts for auditing. ◦ Store logs for troubleshooting and compliance purposes.	
Data Upload	As a user, I need a reliable and secure method for uploading data during the card creation process in the banking app's backend, ensuring that my personal and financial information is handled safely and accurately.	1. **Functionality** ◦ The API should accept user data relevant to card creation. ◦ Ensure correct processing and mapping of the uploaded data to the user's profile. 2. **Security** ◦ Authentication: Verify the identity of the user before allowing data upload. ◦ Authorization: Ensure the user has the necessary permissions to upload data. ◦ Encryption: Use strong encryption protocols for data in transit and at rest. 3. **Validation** ◦ Input Validation: Check for correctness, formatting, and completeness of the uploaded data. ◦ Reject any uploads that do not meet the predefined criteria. 4. **Error Handling** ◦ Implement robust error handling to prevent crashes during data upload. ◦ Provide clear error messages to the user or the calling service in case of a failure. 5. **Database Updates** ◦ Update the user's profile in the database with the new card information upon successful data upload. ◦ Ensure database transactions are atomic to maintain data integrity. 6. **Output/Notification** ◦ On successful upload, return a confirmation message or status code. ◦ Notify the user of the successful update or provide a relevant error message in case of failure. 7. **Logging**	

		◦ Log details of each API request including timestamp, user ID, and the nature of the request. ◦ Record any exceptions or errors encountered during the data upload process.	
Card Data Management			
Update Card Details	As a user, I want to be able to update my card details securely and efficiently through the backend API, so that my card information is always current and accurate.	1. **Functionality** ◦ API should allow users to update card details such as expiration date, cardholder name, and billing address. ◦ Ensure changes are reflected in real-time. 2. **Security** ◦ Authentication: Verify that only authenticated users can access the update feature. ◦ Authorization: Ensure users can only update their own card details. ◦ Encryption: Sensitive data like card details should be encrypted during transmission. 3. **Validation** ◦ Input validation: Check for the validity of the card number, expiration date, and other card details. ◦ Validate that new details adhere to standard card information formats. 4. **Error Handling** ◦ Implement error handling to prevent the application from crashing. ◦ Provide meaningful error messages to the user in case of invalid inputs or system failures. 5. **Database Updates** ◦ On successful update, the new card details should be recorded in the database. ◦ Ensure previous data integrity during the update process. 6. **Output/Notification** ◦ After a successful update, return a confirmation message. ◦ In case of failure, provide a clear error message. 7. **Logging** ◦ Log all API requests for updating card details. ◦ Include timestamps, user ID, and nature of the request for auditing and debugging purposes.	
Delete Card	As a user, I want the ability to securely delete my card information from the banking application, ensuring that all related data is permanently removed from the system without affecting other functionalities.	1. **Functionality** ◦ API endpoint to delete a specific card record. ◦ Should verify the card belongs to the user making the request. 2. **Security**	

		◦ Authentication: Verify user identity before allowing card deletion. ◦ Authorization: User can only delete cards associated with their account. ◦ Encryption: Ensure data transmission is encrypted. 3. **Validation** ◦ Input Validation: Check for valid card ID format. ◦ Ensure card ID exists in the database before attempting deletion. 4. **Error Handling** ◦ Provide meaningful error messages for invalid requests. ◦ Implement try-catch to prevent application crashes. ◦ Handle scenarios where card does not exist. 5. **Database Updates** ◦ On successful deletion, the card record should be permanently removed. ◦ Ensure no residual data remains in the database. 6. **Output/Notification** ◦ Return a confirmation message upon successful deletion. ◦ In case of failure, provide a clear error message. 7. **Logging** ◦ Log details of the deletion request such as (user ID, card ID, timestamp). ◦ Record successful and unsuccessful deletion attempts.	
Card Search			
Filter Cards	As a user, I want the ability to filter my card transactions, so that I can easily search and analyze my card usage based on specific criteria such as date, amount, or merchant type, without exposing any front-end functionality.	1. **Functionality** ◦ The API should allow filtering of card transactions based on multiple criteria (date, amount, merchant type, etc.). ◦ It must support pagination for efficient data retrieval. ◦ The API should provide a summary of transactions when requested. 2. **Security** ◦ Implement Authentication and Authorization checks to ensure only valid users can access the feature. ◦ All data transmissions must be encrypted using industry-standard encryption protocols. 3. **Validation** ◦ Input validation to ensure that all incoming data (dates, amounts, etc.) adhere to expected	

		formats and ranges. ◦ Reject and return appropriate error messages for invalid inputs. 4. **Error Handling** ◦ Gracefully handle errors to prevent the API from crashing. ◦ Provide meaningful error messages for common issues (e.g., network failures, invalid inputs). 5. **Database Updates** ◦ Ensure that queries do not affect database integrity. ◦ Optimized queries for efficient data retrieval and minimal performance impact. 6. **Output/Notification** ◦ The API should return filtered results in a structured and consistent format (e.g., JSON). ◦ Include relevant metadata (e.g., total number of results, pagination information). 7. **Logging** ◦ Log all API requests for auditing, error handling, and debugging purposes. ◦ Ensure logs contain sufficient information (timestamp, user ID, request details) while adhering to privacy standards.	
View Cards	As a user, I want to view and manage my bank cards through the Banking App's backend API, enabling me to securely access, review, and manage my card details without interacting with the front-end interface.	1. **Functionality** ◦ The API should retrieve a list of all the cards linked to the user's account. ◦ Should support query parameters for filtering and sorting the card list. 2. **Security** ◦ Authentication: Verify that the request is made by a logged-in user with valid credentials. ◦ Authorization: Ensure the user has permission to view the cards. ◦ Data Encryption: Sensitive data, such as card numbers, must be encrypted during transmission. 3. **Validation** ◦ Input Validation: Validate all input data for correctness and prevent SQL injection or other malicious attacks. 4. **Error Handling** ◦ Implement error handling to manage unexpected or invalid input without crashing. ◦ Provide meaningful error messages to the caller for known failure scenarios. 5. **Database Updates**	

		◦ No updates required in the database for this read-only feature. ◦ Ensure database queries are optimized for performance. 6. **Output/Notification** ◦ Return a structured response containing the list of cards with relevant details (excluding sensitive information like full card numbers). ◦ Include metadata in the response, such as total number of cards, pagination data if applicable. 7. **Logging** ◦ Log all API requests with necessary details for auditing and debugging purposes. ◦ Sensitive data should not be logged.	
Sort Cards	As a user, I need the ability to sort my bank cards in the banking app's back-end system, allowing me to organize and view them based on specific criteria such as card type, expiration date, and usage frequency.	1. **Functionality** ◦ The API should allow sorting of card details based on various attributes (e.g., card type, expiration date, usage frequency). ◦ Provide multiple sorting options (e.g., ascending, descending). 2. **Security** ◦ Authentication: Ensure that only authenticated users can access the sorting feature. ◦ Authorization: Verify that the user has the necessary permissions to sort cards. ◦ Encryption: Sensitive card data should be encrypted during transmission. 3. **Validation** ◦ Input validation: Validate all inputs for sorting (e.g., sorting parameters should be recognized values). 4. **Error Handling** ◦ Implement error handling to prevent the program from crashing. ◦ Provide meaningful error messages to the front-end in case of invalid inputs or system failures. 5. **Database Updates** ◦ No direct updates to the database, but the API should interact with the database to retrieve and sort card information. 6. **Output/Notification** ◦ The API should return a sorted list of cards based on the specified criteria. ◦ In case of no matching records, return an appropriate message. 7. **Logging** ◦ Log all API requests with relevant information (user ID, timestamp, type of sort requested).	

		◦ Log any errors or exceptions for auditing and debugging purposes.	

User Management - Backend Features - Banking App - FS Java Capstone

User Management

Story	Description	Acceptance Criteria	Story Points
User Authentication/Authorization			
Account Authorization	As a user, I want to ensure that my account is secure and that only I or authorized individuals have access to my personal information and functionalities within the system. I need a reliable way to prove my identity and grant permissions accordingly.	1. **Credential Verification:** ◦ The system must verify the correctness of the username and password during login. ◦ The system should lock the account for a predefined period after several consecutive failed login attempts. 2. **Role-Based Access Control (RBAC):** ◦ The system must ensure users have appropriate roles assigned to access specific features and functionalities. ◦ Unauthorized access attempts to restricted areas should be logged and alerted. 3. **Session Management:** ◦ Once authenticated, the user should be provided with a session token that expires after a period of inactivity. ◦ The system should provide a secure log-out mechanism that invalidates the user's session token. 4. **Security Features:** ◦ Passwords must be stored securely using modern hashing algorithms. ◦ The system should enforce password complexity requirements. 5. **Privacy Compliance:** ◦ Ensure that the user authentication process complies with relevant data protection regulations.	

Account Authentication	As a user, I want to be able to be authenticated as a user with an active account so that I can access secure areas of the application with confidence that my data and access are protected.	1. **Valid Credential Handling** ◦ The system must accept a combination of username and password. ◦ If incorrect credentials are entered, the user is informed with an appropriate error message. ◦ If correct credentials are entered, the user is successfully authorized and directed to their dashboard. 2. **Account Lock Mechanism** ◦ After three consecutive failed login attempts, the account should be temporarily locked for a configurable period. 3. **Password Encryption** ◦ Passwords must be encrypted using a strong encryption algorithm. ◦ No plain text passwords should be stored in the database. 4. **Session Management** ◦ Once logged in, the user must be provided with a secure token for session management. ◦ Sessions must expire after a configurable period of inactivity. 5. **Audit Logging** ◦ All login attempts, successful or not, should be logged with date, time, and location if available. ◦ The logs must be accessible only to authorized administrative personnel. 6. **Compliance with Regulations** ◦ The authentication process must be in compliance with relevant banking and privacy regulations in the jurisdiction(s) where the application is used. 7. **Accessibility** ◦ The login interface must be accessible to users with disabilities, complying with applicable accessibility standards like WCAG 2.0. 8. **Performance** ◦ The login process must complete within a reasonable time, not exceeding a defined number of seconds (e.g., 5 seconds under normal network conditions).	
User Creation			
Manually Create User	As a user I should be able to manually create User for clients e.g. from a form filled in at our branches	1. **Security:** ◦ All user input is validated to prevent SQL injection or other forms of malicious input. ◦ Passwords are hashed and securely stored, never displayed in plaintext. ◦ The system implements CAPTCHA or similar mechanism to deter automated account	

		creation. 2. **Error Handling:** ◦ Clear error messages are displayed for invalid inputs or failed creation attempts (e.g., email already in use, weak password). ◦ The system logs errors and alerts administrators of continuous failed creation attempts which might indicate abuse or system issues. 3. **Success Confirmation:** ◦ Upon successful creation, the user receives a confirmation message and, if applicable, a verification email to activate the account. ◦ The system directs the user to the next logical step, whether it be profile completion, a tutorial, or direct access to the platform. 4. **Privacy Compliance:** ◦ The system clearly outlines how the user data will be used and ensures compliance with relevant data protection regulations (e.g., GDPR, CCPA). ◦ Users have the option to read and accept the privacy policy and terms of service before the account is created. 5. **Account Verification:** ◦ Optional two-factor authentication setup is offered to enhance account security. ◦ Email verification is required to confirm the user's email address.	
Data Upload	As a user, I want to be able to upload my data securely and efficiently, so that I can manage and access it as needed in the application.	1. **Access and Authentication:** ◦ Only authenticated users should be able to access the data upload feature. ◦ The user should receive clear feedback if they're not authorized to perform this action. 2. **File Format:** ◦ The system should only accept files in `.csv` format. ◦ The system should validate the CSV format and ensure it matches the expected structure. ◦ The user should be notified if the uploaded file format is not supported or if the structure doesn't match. 3. **File Size and Performance:** ◦ There should be a maximum file size limit to ensure the system does not get overloaded (e.g., 50MB). ◦ Large files should show a progress bar or an indication to assure users that the upload is in progress.	

   - The system should handle and recover gracefully from any potential timeouts or interruptions.
4. **Data Validation:**
   - Prior to loading the data into the database, the system should validate the data for consistency, correctness, and completeness.
   - The user should be notified of any validation errors with clear instructions or feedback on how to rectify them.
5. **Duplication Check:**
   - The system should check for potential duplicate records to avoid redundancy.
   - If duplicates are found, the user should be prompted for a decision on how to proceed (overwrite, skip, or duplicate).
6. **Error Handling and Reporting:**
   - The system should log all upload attempts, successes, and failures.
   - In case of an error, the user should receive a clear error message indicating what went wrong.
   - The user should have the option to download an error report if there are issues with the uploaded data.
7. **Database Integration:**
   - Successful uploads should result in the data being correctly and securely saved into the database.
   - The system should provide a summary post-upload, detailing the number of records added, updated, or any skipped due to issues.
8. **User Feedback:**
   - Upon successful upload and integration, users should receive a success message or notification.
   - Users should have an option to view a summary or a log of their upload history.
9. **Security:**
   - Data transfers should be secured using appropriate encryption standards.
   - Sensitive data, if any, should be treated with additional layers of security and may require masking or encryption before upload.
10. **Compatibility:**
    - The upload feature should be compatible with all major browsers (e.g., Chrome, Firefox, Safari, Edge).
    - Mobile compatibility, if within the scope, should also be ensured.

Data Upload (Stretch Goal)	As a user, I should be able to upload account data from a CSV file and load it to a database. Future Implementation: To be completed using AWS Lambda, Kinesis and DynamoDB*	1. **File Format** ◦ The system must accept files in CSV (Comma-Separated Values) format. ◦ Files in any format other than CSV must be rejected with an appropriate error message. 2. **File Size** ◦ There should be a maximum file size limit, and files exceeding this limit should be rejected with a suitable error message. ◦ The user should be notified if the upload is taking longer than expected due to file size. 3. **File Content** ◦ The CSV file must have headers that match the database fields. ◦ The system should validate the content type for each field based on the corresponding database field's datatype. ◦ If there are any inconsistencies in the data, an error message should appear indicating the row number and specific issue. ◦ Empty or null values in mandatory fields must trigger an error message. 4. **Upload Process** ◦ Users must be able to browse and select a CSV file from their local system. ◦ There should be a clear indication of the upload's progress, such as a progress bar or percentage. ◦ On successful upload, users should receive a confirmation message. ◦ On unsuccessful upload, users should be informed with a relevant error message and be given the option to try again. ◦ The system should allow the user to cancel the upload process at any point. 5. **Data Loading to Database** ◦ Upon successful upload and validation, data from the CSV must be loaded into the database. ◦ Data loading should not produce any duplicates in the database. ◦ If any record from the CSV conflicts with existing data in the database, the system should provide feedback on the conflicting records and ask the user if they wish to overwrite, ignore, or take another specific action. ◦ The system must ensure that the data integrity and relations in the database are maintained during the loading process.	

		6. **Security** ◦ The data upload feature should be secure and free from vulnerabilities such as SQL injections. ◦ Uploaded files should be scanned for potential threats or malicious content before processing. ◦ Sensitive data, if present in the CSV, should be handled securely and in compliance with relevant data protection regulations. 7. **Logs and Reporting** ◦ All upload actions, whether successful or failed, must be logged with details like timestamp, username, file name, and relevant error messages if any. ◦ Users should be able to download or view a report of the upload, detailing success, failures, and actions taken on conflicts.	
User Data management			
Delete User	As a user, I would like to be able to delete User (with a delete confirmation provided to avoid accidental deletions) **Per Corporate policy: User accounts are not actually deleted. These accounts are maintained for audit purposes.**	1. **Security Verification:** ◦ Prior to deletion, the system must verify the user's identity. ◦ The user should be required to re-enter their password or undergo two-factor authentication. 2. **Data Handling:** ◦ Upon confirmation, all personal data associated with the user's account must be permanently deleted from the system. ◦ The system should also ensure that any backups or copies of this data are removed. 3. **Confirmation and Feedback:** ◦ The user should receive immediate feedback that the deletion process has begun. ◦ A confirmation email or notification should be sent once the account is successfully deleted. 4. **Error Handling:** ◦ In case of any errors during the deletion process, the user should be informed with a clear message. ◦ The system should provide guidance or options to retry or contact support. 5. **Post-deletion State:** ◦ After account deletion, the user should no longer be able to log in or recover the account. ◦ The system should anonymize or remove any public-facing data (like comments or posts) previously associated with the account. 6. **Audit Trail:** ◦ The system should maintain an audit trail of the account deletion for compliance and security	

		purposes. 7. **Compliance Adherence:** ◦ The deletion process must comply with relevant data protection and privacy laws, such as GDPR.	
Update User Details	As a user, I want to be able to update my details, such as name, email address, and password, so that I can ensure my personal and contact information is current and accurate.	1. **Authentication and Authorization** ◦ The system must validate the user's authentication token before allowing access to update user details. ◦ Only authorized users with appropriate permissions (e.g., the user themselves or an administrator) can update user details. 2. **Input Validation** ◦ All user input fields (e.g., name, email, phone number) must be validated for correct format, length, and data type according to the system's specifications. ◦ Proper error messages must be returned if any input fails validation. 3. **Updating User Details** ◦ The system must update the user's details in the database, ensuring data consistency and integrity. ◦ Only the fields provided by the user should be updated, leaving other fields unchanged. 4. **Error Handling** ◦ The system must handle any errors during the update process gracefully, providing appropriate error messages to inform the user. ◦ The system should log errors for further investigation and debugging. 5. **Concurrency Control** ◦ The system must handle concurrent updates to the same user record and prevent any data loss or corruption using appropriate concurrency control mechanisms (e.g., optimistic or pessimistic locking). 6. **Performance** ◦ The update process must be optimized for performance, ensuring a fast and seamless user experience. ◦ The system must maintain a reasonable response time even under high load or with a large number of users. 7. **Security** ◦ The system must follow best practices for secure data transmission (e.g., using HTTPS) and storage (e.g., hashing passwords). ◦ The system should be designed to prevent common security vulnerabilities such as SQL	

		injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).	
	User Search		
View User	As a user, I want to be able to view specific user details so that I can retrieve relevant information about users quickly and efficiently.	1. **Authentication and Authorization** ◦ The system must validate the user's authentication token before allowing access to update user details. ◦ Only authorized users with appropriate permissions (e.g., the user themselves or an administrator) can update user details. 2. **Input Validation** ◦ All user input fields (e.g., name, email, phone number) must be validated for correct format, length, and data type according to the system's specifications. ◦ Proper error messages must be returned if any input fails validation. 3. **Error Handling** ◦ The system must handle any errors during the update process gracefully, providing appropriate error messages to inform the user. ◦ The system should log errors for further investigation and debugging. 4. **Concurrency Control** ◦ The system must handle concurrent updates to the same user record and prevent any data loss or corruption using appropriate concurrency control mechanisms (e.g., optimistic or pessimistic locking). 5. **Performance** ◦ The update process must be optimized for performance, ensuring a fast and seamless user experience. ◦ The system must maintain a reasonable response time even under high load or with a large number of users. 6. **Security** ◦ The system must follow best practices for secure data transmission (e.g., using HTTPS) and storage (e.g., hashing passwords). ◦ The system should be designed to prevent common security vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).	
Filter User	As a user, I need the ability to filter through a list of users based on specific criteria so that I can efficiently manage and interact with users in the system.	1. **Functionality** ◦ The API endpoint must support filtering based on multiple attributes such as name, role, and status.	

		<ul><li>The API should return a list of users that match the filtering criteria.</li></ul>2. **Security**<ul><li>Implement Authentication and Authorization checks.</li><li>Only authorized users (e.g., admin roles) should be able to access this endpoint.</li><li>Ensure all sensitive data in transit is encrypted using industry-standard protocols (e.g., TLS).</li></ul>3. **Validation**<ul><li>Input validation to ensure only valid data is processed.</li><li>The API should reject requests with invalid filter parameters with an appropriate error message.</li></ul>4. **Error Handling**<ul><li>The API should handle errors gracefully and not crash upon encountering unexpected situations.</li><li>Error responses should include meaningful messages for debugging without exposing sensitive information.</li></ul>5. **Database Updates**<ul><li>No direct database updates or changes are expected as part of this user story.</li><li>The API should only read and filter existing user data from the database.</li></ul>6. **Output/Notification**<ul><li>The API response should include a structured list of filtered users.</li><li>Include relevant user details in the response, ensuring no sensitive information is exposed.</li></ul>7. **Logging**<ul><li>Log all API requests for filtering users.</li><li>Include information such as request time, user ID of the requester, filter parameters used, and any errors encountered.</li></ul>	
Sort User	As a user I would like to be able to sort the user search results that I am viewing.	1. **Functionality**<ul><li>The API should provide a feature to sort users based on specified fields (e.g., name, registration date, account type).</li><li>It must support sorting in both ascending and descending orders.</li></ul>2. **Security**<ul><li>Implement Authentication and Authorization checks.</li><li>Only authorized personnel (e.g., admins, support staff) should have access to this feature.</li></ul>	

- Ensure all sensitive data in transit is encrypted using industry-standard protocols (e.g., TLS).

3. **Validation**
    - Input validation for sorting parameters to prevent SQL injection and other malicious activities.
    - Validate that sorting parameters match the pre-defined fields available for sorting.
4. **Error Handling**
    - Gracefully handle errors with appropriate messaging.
    - Ensure the program does not crash on encountering an error (e.g., invalid sorting parameter).
5. **Database Updates**
    - No direct updates to the database are needed for this read-only operation.
    - Ensure database queries are optimized for performance.
6. **Output/Notification**
    - The API should return a sorted list of users.
    - Include relevant metadata (e.g., total number of users, number of pages).
7. **Logging**
    - Log all API requests for sorting users.
    - Include details like timestamp, user ID of the requestor, sorting parameters for auditing and debugging purposes.

Loan Management - Backend Features - Banking App - FS Java Capstone

Loan Management

Story	Description	Acceptance Criteria	Story Points
Loan Creation			
Manually Create Loan	As a user I should be able to manually create Loan for clients e.g. from a form filled in at our branches	1. **Functionality**: ◦ The system should allow authorized users to create a loan manually by inputting all necessary fields, such as loan amount, term, interest rate, and borrower details. 2. **Authorization and Access**: ◦ Only users with a designated role (for example, 'Loan Officer') should be able to create a new loan manually. Unauthorized access attempts should be logged and reported. 3. **Validation**: ◦ The system should validate all inputs. For instance, the loan amount and term should be positive numbers, and the borrower details should match with the records in the database. 4. **Loan Agreement Generation**: ◦ Upon successful creation of a loan, the system should automatically generate a loan agreement. 5. **Database Update**: ◦ The newly created loan should be correctly reflected in the database, updating the borrower's profile and the overall loan portfolio. 6. **Notification**: ◦ The system should send notifications to relevant parties (like the borrower and the loan manager) once the loan is created. 7. **Error Handling**: ◦ In case of invalid input or system error, the system should provide appropriate error	

		messages and instructions to correct the errors. 8. **Audit Trails**: ◦ All manual loan creation activities should be logged for auditing purposes.	
Data Upload	As a user, I want to upload loan data files securely and efficiently to the system so that I can process loans without manual data entry. This feature should validate the data for format and completeness, handle errors gracefully, and ensure that data is encrypted and transferred securely.	1. **Access and Authentication:** ◦ Only authenticated users should be able to access the data upload feature. ◦ The user should receive clear feedback if they're not authorized to perform this action. 2. **File Format:** ◦ The system should only accept files in `.csv` format. ◦ The system should validate the CSV format and ensure it matches the expected structure. ◦ The user should be notified if the uploaded file format is not supported or if the structure doesn't match. 3. **File Size and Performance:** ◦ There should be a maximum file size limit to ensure the system does not get overloaded (e.g., 50MB). ◦ Large files should show a progress bar or an indication to assure users that the upload is in progress. ◦ The system should handle and recover gracefully from any potential timeouts or interruptions. 4. **Data Validation:** ◦ Prior to loading the data into the database, the system should validate the data for consistency, correctness, and completeness. ◦ The user should be notified of any validation errors with clear instructions or feedback on how to rectify them. 5. **Duplication Check:** ◦ The system should check for potential duplicate records to avoid redundancy. ◦ If duplicates are found, the user should be prompted for a decision on how to proceed (overwrite, skip, or duplicate). 6. **Error Handling and Reporting:** ◦ The system should log all upload attempts, successes, and failures. ◦ In case of an error, the user should receive a clear error message indicating what went wrong. ◦ The user should have the option to download an error report if there are issues with the uploaded data.	

		7. **Database Integration:** ◦ Successful uploads should result in the data being correctly and securely saved into the database. ◦ The system should provide a summary post-upload, detailing the number of records added, updated, or any skipped due to issues. 8. **User Feedback:** ◦ Upon successful upload and integration, users should receive a success message or notification. ◦ Users should have an option to view a summary or a log of their upload history. 9. **Security:** ◦ Data transfers should be secured using appropriate encryption standards. ◦ Sensitive data, if any, should be treated with additional layers of security and may require masking or encryption before upload. 10. **Compatibility:** ◦ The upload feature should be compatible with all major browsers (e.g., Chrome, Firefox, Safari, Edge). ◦ Mobile compatibility, if within the scope, should also be ensured.	
Data Upload (Stretch Goal)	As a user, I should be able to upload account data from a CSV file and load it to a database. Future Implementation: To be completed using AWS Lambda, Kinesis and DynamoDB*	1. **File Format** ◦ The system must accept files in CSV (Comma-Separated Values) format. ◦ Files in any format other than CSV must be rejected with an appropriate error message. 2. **File Size** ◦ There should be a maximum file size limit, and files exceeding this limit should be rejected with a suitable error message. ◦ The user should be notified if the upload is taking longer than expected due to file size. 3. **File Content** ◦ The CSV file must have headers that match the database fields. ◦ The system should validate the content type for each field based on the corresponding database field's datatype. ◦ If there are any inconsistencies in the data, an error message should appear indicating the row number and specific issue. ◦ Empty or null values in mandatory fields must trigger an error message. 4. **Upload Process** ◦ Users must be able to browse and select a CSV file from their local system.	

		◦ There should be a clear indication of the upload's progress, such as a progress bar or percentage. ◦ On successful upload, users should receive a confirmation message. ◦ On unsuccessful upload, users should be informed with a relevant error message and be given the option to try again. ◦ The system should allow the user to cancel the upload process at any point. 5. **Data Loading to Database** ◦ Upon successful upload and validation, data from the CSV must be loaded into the database. ◦ Data loading should not produce any duplicates in the database. ◦ If any record from the CSV conflicts with existing data in the database, the system should provide feedback on the conflicting records and ask the user if they wish to overwrite, ignore, or take another specific action. ◦ The system must ensure that the data integrity and relations in the database are maintained during the loading process. 6. **Security** ◦ The data upload feature should be secure and free from vulnerabilities such as SQL injections. ◦ Uploaded files should be scanned for potential threats or malicious content before processing. ◦ Sensitive data, if present in the CSV, should be handled securely and in compliance with relevant data protection regulations. 7. **Logs and Reporting** ◦ All upload actions, whether successful or failed, must be logged with details like timestamp, username, file name, and relevant error messages if any. ◦ Users should be able to download or view a report of the upload, detailing success, failures, and actions taken on conflicts.	
Loan Data management			
Delete Loan	As a user, I would like to be able to delete loan (with a delete confirmation provided to avoid accidental deletions) **Per Corporate policy: Data is not actually deleted. It is maintained for audit purposes.**	1. **Access Control:** ◦ Only authorized users (e.g., banking administrators, loan officers, or individuals with specific permissions) can delete a loan. ◦ Unauthorized attempts should return an error message.	

		2. **Confirmation Prompt:** ◦ Before the loan is deleted, the system must present a confirmation prompt to the user to prevent accidental deletions. ◦ The prompt message should clearly state the action to be taken and ask for the user's confirmation. 3. **Loan Status Check:** ◦ The system should not allow deletion of loans with outstanding balance or any active status. ◦ Only loans marked as 'Closed', 'Paid off' or equivalent status should be eligible for deletion. 4. **Data Integrity:** ◦ All related data (e.g., payment history, customer details related to the loan) should either be deleted or anonymized in accordance with data protection laws and banking regulations. 5. **Log of Deletions:** ◦ A log must be maintained recording details of the deleted loans, including who deleted it, when it was deleted, and the state of the loan at the time of deletion. 6. **Error Handling:** ◦ If a deletion attempt fails, the system should provide a clear, user-friendly error message explaining why the attempt failed. 7. **Success Confirmation:** ◦ Upon successful deletion, the system should provide a success message confirming that the loan has been deleted. 8. **Non-Recoverability:** ◦ Once a loan is deleted, it cannot be recovered. The system should make this clear in the confirmation prompt before deletion.	
Update Loan Details	As a user, I would like to be able to edit and update any Loan details	1. **Authorization:** ◦ Only authorized users (loan officers, authorized bank employees) should be able to access and use the "Update Loan Details" feature. 2. **Validation:** ◦ The system should validate all loan details inputs according to business rules (e.g., loan amount, interest rate, tenure, etc.) before updating the record in the system. 3. **Auditability:** ◦ The system should maintain a complete audit trail of every update made to a loan's details, including the user who made the changes, the date and time of the change, and what was changed.	

		4. **Notification:** ◦ The system should notify the relevant parties (e.g., loan account holder, guarantor, etc.) via their preferred communication method (e.g., email, SMS) when a loan's details are updated. 5. **Error Handling:** ◦ The system should provide clear and understandable error messages when updates cannot be made, explaining why the update failed. 6. **Accessibility:** ◦ The feature should be easily accessible within the user interface, requiring no more than 2-3 clicks from the main dashboard to reach. 7. **Performance:** ◦ The system should update loan details within a reasonable timeframe (e.g., less than 5 seconds), ensuring a seamless user experience. 8. **Consistency:** ◦ The updated loan details should consistently reflect across all systems where this data might be accessed or referenced, ensuring data integrity. 9. **Rollback Mechanism:** ◦ The system should provide a mechanism to roll back changes in case of an error or unauthorized update. 10. **Data Protection:** ◦ The system should adhere to data protection and privacy standards during the update process, ensuring the confidentiality of loan details.	
Loan Search			
Filter Loan	As a user, I want to be able to filter loan options based on criteria such as amount, interest rate, loan term, and lender so that I can find loans that match my needs more efficiently.	1. **Functionality** ◦ The system allows the user to filter loans based on various parameters such as amount, interest rate, loan term, and lender. ◦ Filters can be applied individually or in combination. 2. **Back-end Implementation** ◦ All filtering logic is implemented on the back-end. ◦ No front-end logic is involved in the filtering process, ensuring a clear separation of concerns. 3. **Performance** ◦ The filter operation must complete within a reasonable timeframe (e.g., less than 2 seconds) to ensure a good user experience.	

		◦ The system efficiently handles large datasets without significant performance degradation. 4. **Security** ◦ All user inputs for filters are validated on the back-end to prevent SQL injection and other security vulnerabilities. ◦ The system adheres to data privacy standards and regulations for handling user data. 5. **Data Integrity** ◦ The system ensures that the data returned after applying filters is accurate and consistent with the database. 6. **Error Handling** ◦ In case of invalid filter inputs, the system returns a clear error message to the user. ◦ The system gracefully handles server-side errors, ensuring that they do not crash the application.	
View Loan	As a user I would like to be able to view all loans and have the ability to filter those same loans	1. **Functionality** ◦ API should retrieve specific loan details based on user's query. ◦ Support for querying by loan ID or user-specific parameters. 2. **Security** ◦ Implement authentication (AuthN) and authorization (AuthZ) mechanisms. ◦ Ensure API access is restricted to authenticated and authorized users only. 3. **Encryption** ◦ Ensure sensitive data (e.g., loan amounts, user details) is encrypted at rest. 4. **Validation** ◦ Input validation for all incoming data (e.g., loan IDs, user credentials). ◦ Prevent SQL injection and other common security vulnerabilities. 5. **Error Handling** ◦ API to provide clear, user-friendly error messages for invalid requests. ◦ Implement try-catch blocks to prevent crashes and log exceptions. 6. **Database Updates** ◦ Read-only access to loan-related tables for this API. ◦ Ensure no write operations are performed on the database. 7. **Output/Notification** ◦ Successful API calls should return detailed loan information in a structured format (e.g., JSON).	

		◦ Include loan status, amount, duration, interest rate, etc., in the response. 8. **Logging** ◦ Log all API requests with sufficient details (timestamp, user ID, request type). ◦ Include error logging for troubleshooting and auditing purposes.	
Sort Loan	As a user, I want to be able to sort loan offers or applications based on various criteria such as interest rate, loan amount, tenure, and others so that I can easily find the most suitable loan options.	1. **Sorting Feature Integration:** ◦ The backend should provide a feature to sort loan offers based on specified criteria including but not limited to interest rate, loan term, and total repayment amount. ◦ Users should be able to specify the sorting order (ascending or descending). 2. **Data Integrity:** ◦ Ensure that sorting the loan offers does not alter the underlying data or attributes of the loans. 3. **Performance:** ◦ The sorting operation must be efficient and not cause significant delays in the user interface response time. 4. **Default Sorting:** ◦ In the absence of user-specified criteria, the system should have a default sorting order (e.g., by interest rate ascending). 5. **Multiple Criteria Sorting:** ◦ Allow users to sort by multiple criteria simultaneously, where the secondary criteria are used when the primary criteria values are equal. 6. **Invalid Criteria Handling:** ◦ The system should validate the sorting criteria input and return an error message for any unrecognized or unsupported criteria. 7. **Fault Tolerance:** ◦ In case of a failure during the sorting process, the system should not crash and should provide a meaningful error message to the user. 8. **Security:** ◦ Ensure that the sorting feature does not expose any vulnerabilities that could be exploited to gain unauthorized access to data or other resources. ◦ Implement appropriate security measures to safeguard the integrity and confidentiality of the loan data during the sorting process.	

Transaction Management - Backend Features - Banking App - FS Java Capstone

Transaction Management

Story	Description	Acceptance Criteria	Story Points
Transaction Creation			
Create Transaction	As a user, I want to create a transaction, so that I can transfer funds between accounts or make payments.	1. **Transaction Details**: ◦ Users should be able to input essential transaction details such as source account, destination account or biller details, amount, and transaction description. ◦ The system should validate that the entered amount does not exceed the source account balance. 2. **Security**: ◦ Transaction initiation should require multi-factor authentication or a one-time password (OTP) to ensure the user's identity. ◦ All transaction data should be encrypted during transmission and storage. ◦ Secure logging of all transaction activities, ensuring no sensitive data like full account numbers are exposed. 3. **Error Handling**: ◦ If the transaction fails, the user should be notified with a clear error message detailing the reason for the failure. ◦ All transaction errors should be logged for audit and debugging purposes. ◦ The system should handle transaction rollbacks effectively, ensuring that funds are not deducted unless the transaction is successful. 4. **Transaction Confirmation**: ◦ Upon successful transaction completion, the user should receive a confirmation message with a transaction reference number.	

		◦ Both the source and destination accounts should reflect the updated balances post-transaction. 5. **Rate Limiting and Fraud Detection**: ◦ The system should have mechanisms to detect and prevent suspicious transaction patterns. ◦ Rate limiting should be in place to prevent excessive transactions from a single account in a short duration. 6. **Notifications**: ◦ Users should have the option to receive notifications (e.g., email, SMS) upon transaction completion. ◦ Notifications should mask sensitive data, showing only the last few digits of account numbers. 7. **Performance**: ◦ Transactions should process within a reasonable timeframe, ensuring user satisfaction. ◦ The system should be able to handle multiple concurrent transaction requests without performance degradation.	
Transaction Search			
View Transaction	As a user, I want to be able to view the full details of a specific transaction so that I can check all associated data or troubleshoot issues.	1. **Access Control:** ◦ Only users with admin privileges can access the "View Transaction" feature. ◦ Users must be authenticated and authorized to ensure secure access. 2. **Security:** ◦ All transaction details are encrypted during transmission between the server and the front-end admin portal. ◦ Any sensitive data, like account numbers, should be partially masked when displayed. E.g., "XXXX-XXXX-1234". ◦ The system should have a secure logging mechanism that records every transaction view activity by the admin, including date, time, and user ID. 3. **Error Handling:** ◦ If there's an issue retrieving transaction data from the backend, the system should show an error message like "Error retrieving transaction details. Please try again later." ◦ The system should handle and display errors gracefully without exposing any technical details or vulnerabilities. 4. **Usability:**	

		<ul><li>Each transaction detail displayed should be clearly labelled to avoid confusion.</li></ul>	
Sort Transactions	As a user, I need the ability to sort my transactions in various orders (e.g., date, amount, type) through the backend API, ensuring a tailored view of my transaction history for better financial management.	1. **Functionality**<ul><li>The API should support sorting transactions by date, amount, and transaction type.</li><li>It must provide options for ascending and descending order sorting.</li></ul>2. **Security**<ul><li>Authentication: Ensure only authenticated users can access the sort functionality.</li><li>Authorization: Verify that users can only sort transactions within their account.</li><li>Encryption: Use SSL encryption for data transmission.</li></ul>3. **Validation**<ul><li>Validate sort parameters (date, amount, type) for correct data type and format.</li><li>Ensure invalid or unsupported parameters are rejected.</li></ul>4. **Error Handling**<ul><li>Implement error handling to prevent the API from crashing on invalid requests.</li><li>Provide meaningful error messages for issues like invalid parameters or server errors.</li></ul>5. **Database Updates**<ul><li>No direct database updates needed for sorting. Sorting should be applied at the query level.</li></ul>6. **Output/Notification**<ul><li>The API should return a sorted list of transactions based on the specified parameters.</li><li>Include relevant metadata in the response (e.g., sort parameter used, total transactions).</li></ul>7. **Logging**<ul><li>Log all API requests with details like user ID, timestamp, sort parameters, and response status for auditing and debugging.</li><li>Record any exceptions or errors encountered during the API request processing.</li></ul>	
Filter Transactions	As a user, I want to be able to filter transactions based on parameters such as date range, transaction type, or customer name so that I can focus on a subset of transactions that meet specific criteria.	1. **Search Input**<ul><li>A search input box should be provided to allow the admin to enter keywords related to the transaction.</li></ul>2. **Date Range**<ul><li>Admins should have the option to filter transactions by a specific date range. Both 'start date' and 'end date' input fields should be available.</li></ul>3. **Transaction Type Filter**	

		◦ Admins should have the option to filter transactions based on their types (e.g., Deposit, Withdrawal, Transfer). 4. **Amount Range** ◦ There should be an option to filter transactions based on a specific amount range. Both minimum and maximum input fields should be available. 5. **Account Number** ◦ Allow filtering based on specific account numbers associated with the transactions. 6. **Display Results** ◦ Transactions matching the filter criteria should be displayed in a list format. ◦ Each item in the list should show transaction details such as the date, account number, transaction type, and amount. 7. **No Results** ◦ If no transactions match the filter criteria, a message indicating "No transactions found" should be displayed. 8. **Reset Filters** ◦ An option should be available to clear all active filters and view all transactions again. 9. **Security** ◦ The filtering options should adhere to the role-based access controls of the banking system. ◦ Sensitive transaction details must be protected and should not be displayed unless the admin has the appropriate permissions. ◦ All transactions and filtering actions should be logged for auditing purposes. 10. **Error Handling** ◦ If the filter functionality encounters an error, a user-friendly error message should be displayed to the admin. ◦ System errors should not reveal any technical or sensitive details to the user. ◦ Invalid date or amount range inputs should prompt a validation message informing the admin of the correct format or range. 11. **Performance** ◦ Filtering results should be returned in a timely manner, ensuring that admins do not experience unnecessary delays.	
Search Transaction	As a user, I want to have the ability to search my transactions, so that I can easily find specific transaction details based on various criteria such as transaction date, amount, type, recipient, and more. This feature should support quick	1. **Functionality** ◦ The API should allow querying of transaction data based on input parameters like transaction ID, date range, amount range, and transaction type.	

	navigation and filtering capabilities for efficient management and review of my banking transactions.	◦ It should support pagination and sorting to handle large volumes of data efficiently. ◦ The API should integrate with the existing transaction database to fetch and filter transaction data accurately. 2. **Security** ◦ Implement Authentication and Authorization checks to ensure only authenticated users can access the search functionality. ◦ Sensitive data, such as account numbers, should be encrypted in transit and at rest. ◦ Ensure compliance with data protection regulations relevant to banking and financial transactions. 3. **Validation** ◦ Input validation must be in place to verify that all incoming data is of the correct format and within acceptable ranges. ◦ Implement checks against SQL injection or other forms of injection attacks in the input fields. 4. **Error Handling** ◦ The API should return meaningful error messages for invalid queries, unauthorized access, or internal failures. ◦ Ensure the system logs the errors while not crashing or exposing sensitive information. ◦ Implement retries or fallbacks for transient errors when possible. 5. **Database Updates** ◦ Define how the transaction data is accessed and what indices might be needed to improve search performance. ◦ Ensure that search operations are read-only and do not modify transaction data. ◦ Outline any potential database optimizations necessary for handling search queries efficiently. 6. **Output/Notification** ◦ Upon a successful search, the API should return a list of transactions matching the criteria. ◦ Define the structure of the returned data, ensuring it includes essential transaction details but omits sensitive information. ◦ For no results, return a clear message indicating no transactions were found matching the criteria. 7. **Logging**	

		◦ Log all API requests with necessary details for auditing, error handling, and debugging without exposing sensitive data. ◦ Include timestamps, user information, type of request, and the outcome of the request (success or failure) in the logs. ◦ Ensure that logs are secure, properly rotated, and comply with data retention policies.	
Download Transactions	As a user, I want to be able to securely download my transaction history so that I can keep personal records, perform analysis, or use the data for reporting purposes.	1. **Functionality:** ◦ The system shall allow users to download their transaction history as a CSV, PDF, or Excel file. ◦ Users shall be able to specify a date range for the transactions they wish to download. ◦ The download feature should be accessible from the transaction search results page. 2. **Performance:** ◦ The system should generate the download file within a reasonable time, not exceeding 60 seconds. ◦ The system should handle concurrent download requests efficiently. 3. **Security:** ◦ Ensure that all data transfers are conducted over a secure channel (e.g., HTTPS). ◦ Implement role-based access controls to ensure only authorized users can download transaction data. ◦ Ensure sensitive data within the transaction history is masked or encrypted as per compliance standards. 4. **Data Integrity:** ◦ The downloaded file must accurately reflect the user's transaction history, matching the records displayed on the user interface. ◦ Ensure that the data format and structure in the downloaded file are consistent and error-free. 5. **Input Validation:** ◦ The system should validate date range inputs and ensure they are within acceptable parameters (e.g., not future-dated, not exceeding a maximum historical range). ◦ Alert the user if the selected date range returns no transactions for download. 6. **Error Handling:** ◦ In cases of failure (e.g., server error, timeout), the system should notify the user and log the incident for further investigation. ◦ Provide users with clear error messages if the download cannot be completed due to specific	

		issues (e.g., network failure, file generation error). 7. **Fallback Mechanisms:** ◦ In case of a system malfunction or heavy load, ensure there's a fallback or retry mechanism for users to attempt the download again. ◦ Inform users of the estimated wait time if the system is experiencing high traffic or delays.	

Frontend Features - Banking App - FS Java Capstone

Frontend Features

- Homepage - Frontend Features - Banking App - FS Java Capstone
- Admin Portal - Frontend Features - Banking App - FS Java Capstone
- Customer Portal - Frontend Features - Banking App - FS Java Capstone

Homepage - Frontend Features - Banking App - FS Java Capstone

Story	Description	Acceptance Criteria	Story Points
Homepage			
Admin Portal Homepage	As a User, I want to have an intuitive, secure, and informative homepage on the Admin Portal, so that I can effectively manage various banking activities, ensure optimal customer experience, and maintain system integrity.	1. **Security** ◦ The portal should require secure login credentials, following industry-standard practices for password complexity and encryption. ◦ Failed login attempts are limited to three, after which an account is temporarily locked, and an email is sent to the administrator. 2. **Usability** ◦ The homepage should display critical information (e.g., system status, recent transactions, user management, etc.) in a clear, concise manner. ◦ Navigation to other sections of the portal should be intuitive and easily accessible from the homepage. 3. **Functionality** ◦ The portal should allow the administrator to manage user accounts, including creating, editing, disabling, and deleting accounts. ◦ The portal should enable administrators to generate different types of reports (e.g., transaction reports, audit logs, etc.). 4. **Performance** ◦ The portal should load within 3 seconds under normal network conditions, with all elements rendered correctly. ◦ All interactive elements on the page (e.g., buttons, links, dropdowns) should respond to user actions within 200ms. 5. **Accessibility** ◦ The portal should comply with WCAG 2.1 AA accessibility guidelines, including features for visually impaired users like screen reader compatibility and adjustable font sizes. ◦ The portal should be responsive and compatible with different devices and screen sizes.	

User Portal Homepage	As a user, I want a user-friendly and secure portal homepage, So that I can manage and view all of my financial activities in one place.	1. **Accessibility & Compatibility:** ◦ The portal should be accessible and compatible across different devices (desktops, tablets, smartphones) and major browsers (Chrome, Firefox, Safari, Edge). ◦ The portal should comply with accessibility standards. 2. **Login Security** ◦ After a certain number of failed login attempts, the user account should be temporarily locked. 3. **Homepage Layout:** ◦ Upon successful login, the user should see a dashboard summarizing their account status, including recent transactions, current balance, and pending transactions. ◦ The dashboard should display any alerts or messages requiring the user's attention. 4. **Navigation:** ◦ A clear and intuitive navigation system should be present for users to access different banking services (e.g., account management, fund transfers, loan applications, Card management). ◦ A search function should be provided for the user to find information or services quickly. 5. **Account Management:** ◦ Users should be able to view detailed account information, update personal details, and manage their banking preferences. 6. **Performance & Reliability:** ◦ The portal should load quickly (under 2-3 seconds) even under heavy traffic. ◦ Scheduled maintenance and unexpected downtime should be minimal and communicated in advance. 7. **Privacy & Compliance:** ◦ The portal should comply with all applicable laws and regulations, including data privacy laws. ◦ The portal should display the bank's privacy policy and terms of use in a visible location.	

Admin Portal - Frontend Features - Banking App - FS Java Capstone

Admin Portal

Account Management - Admin Portal - Banking App - FS Java Capstone

Account Management

- Admin Portal: Account Creation
- Admin Portal: Account Data Management
- Admin Portal: Account Search

Story	Description	Acceptance Criteria	Story Points
Admin Portal: Account Creation			
Data Upload	As a user, I need the ability to securely upload data files to the Admin Portal for account creation purposes. This process should validate the data, handle any errors gracefully, and provide clear notifications upon completion or failure, while ensuring all activities are logged for audit and troubleshooting purposes.	1. **Functionality** ◦ The system shall allow users to upload data files in specified formats (e.g., CSV, Excel). ◦ Uploaded data shall be used exclusively for account creation within the banking application. 2. **Security** ◦ The file upload process shall comply with the bank's data protection and privacy policies. ◦ Secure transfer protocols shall be used for file uploads to prevent unauthorized access. 3. **Validation** ◦ The system shall validate file format and content, ensuring compatibility with account creation requirements. ◦ Validation checks shall include format correctness, mandatory fields presence, and data consistency. 4. **Error Handling** ◦ In case of an invalid file format or content, the system shall display a descriptive error message to the user. ◦ The system shall handle exceptions gracefully without causing system crashes or data corruption. 5. **Output/Notification** ◦ Upon successful upload and validation, the system shall display a confirmation message to the user.	

		<ul><li>In case of failure, an error notification shall be displayed with guidance for corrective actions.</li></ul>6. **Logging**<ul><li>All upload attempts, successful or failed, shall be logged with timestamps, user details, and error descriptions (if any).</li><li>Logs shall be stored securely and be accessible only to authorized personnel for monitoring and auditing purposes.</li></ul>	
Manually Create Account	As a user, I want the ability to manually create a bank account through the Admin Portal, ensuring that the process is secure, validates input correctly, handles errors gracefully, provides appropriate feedback, and logs activities for auditing purposes.	1. **Functionality**<ul><li>The feature allows the creation of a new bank account with necessary details such as account type, customer ID, and initial deposit amount.</li><li>Users can view a summary of the details entered before final submission.</li></ul>2. **Security**<ul><li>All data transmissions are encrypted using industry-standard protocols.</li><li>The feature restricts access to authorized personnel only, as defined in the user roles and permissions settings.</li></ul>3. **Validation**<ul><li>Input fields like customer ID and initial deposit amount are validated for format and value range.</li><li>Incomplete or improperly formatted inputs trigger a prompt for correction.</li></ul>4. **Error Handling**<ul><li>Clear error messages are displayed for any failed operation, such as network issues or data submission errors.</li><li>The system gracefully handles server-side errors, ensuring the application remains stable.</li></ul>5. **Output/Notification**<ul><li>Upon successful account creation, a confirmation message is displayed to the user.</li><li>Notifications for successful or unsuccessful operations are clearly distinguishable.</li></ul>6. **Logging**<ul><li>All account creation activities are logged with timestamps, user IDs, and operation details.</li><li>Error logs include error messages and relevant context for troubleshooting.</li></ul>	
Admin Portal: Account Data Management			
Update Account Details	As a user, I need the ability to manually update bank account details within the admin portal, ensuring accuracy and efficiency in account data management. This feature should be secure,	1. **Functionality**<ul><li>Ability to select a bank account from a list or by searching account number.</li></ul>	

	user-friendly, and capable of handling errors gracefully while providing necessary feedback and logging actions for auditing purposes.	◦ Fields available for update: account holder name, account type, contact details, and address. ◦ Save changes functionality with confirmation before submission. 2. **Security** ◦ Require user authentication before accessing the update feature. ◦ Implement role-based access control; only authorized personnel can update account details. ◦ Secure data transmission using encryption. 3. **Validation** ◦ Validate input fields for correct format (e.g., alphanumeric for account number, text for names). ◦ Ensure mandatory fields (e.g., account holder name, account number) are not left blank. ◦ Check for duplicate account details to prevent redundant entries. 4. **Error Handling** ◦ Display user-friendly error messages for invalid inputs or system issues. ◦ Implement timeouts for unresponsive requests. ◦ Provide an option to revert changes or retry in case of a failed update. 5. **Output/Notification** ◦ Show a success message upon successful update of account details. ◦ Notify the user of any issues during the update process. ◦ Option to send email confirmation/notification to the account holder about the update. 6. **Logging** ◦ Log all activities related to account updates including user ID, time, and nature of the update. ◦ Record unsuccessful update attempts with reason for failure. ◦ Maintain logs in a secure and compliant manner as per banking regulations.	
Delete Account	As a user with administrative privileges in the banking application, I need the ability to manually delete bank accounts from the admin portal. This feature should ensure secure, validated, and error-free operations while providing clear notifications and logging of actions taken.	1. **Functionality** ◦ The system allows deletion of bank accounts by users with administrative rights. ◦ The deletion process can be initiated from the account data management section of the admin portal. ◦ A confirmation step is required before final deletion to prevent accidental removals.	

		2. **Security** ◦ Deletion requests must be authenticated and authorized, ensuring only users with admin rights can execute this function. ◦ Secure protocols (like HTTPS) are used for data transmission during the deletion process. ◦ Sensitive data is masked or encrypted during the process. 3. **Validation** ◦ The system validates the existence of the account before attempting deletion. ◦ Input fields for account identification (like account number) have format and existence checks. 4. **Error Handling** ◦ Clear error messages are displayed if the deletion process fails. ◦ The system handles scenarios where the account is already engaged in pending transactions or locked due to compliance issues. 5. **Output/Notification** ◦ Upon successful deletion, a confirmation message is displayed to the user. ◦ In case of failure, a notification detailing the reason is shown. 6. **Logging** ◦ All deletion actions, along with timestamp and admin user details, are logged in the system. ◦ Logs include information about the success or failure of the deletion process.	
Admin Portal: Account Search			
View Accounts	As a user, I want to be able to efficiently search and view bank accounts through the Admin Portal, ensuring that the process is secure, reliable, and user-friendly. This feature should allow me to quickly find specific accounts based on various criteria without accessing or impacting the backend systems directly.	1. **Functionality** ◦ The system should provide a search function in the Admin Portal to view bank account details. ◦ Search can be conducted using account number, customer name, or other relevant identifiers. ◦ The system should display a list of accounts matching the search criteria. ◦ Each account in the list should show essential details like account number, customer name, and account balance. 2. **Security** ◦ Access to the account search feature should be restricted to authenticated users only. ◦ User roles and permissions should be checked to ensure only authorized personnel can view	

		account details. ◦ The system should employ SSL/TLS encryption for data transmission. 3. **Validation** ◦ Input fields for search criteria should validate for the correct format (e.g., numerical for account numbers). ◦ Validation messages should be displayed for incorrect or incomplete inputs. 4. **Error Handling** ◦ If the search query fails or times out, the system should display a relevant error message. ◦ Errors should be handled gracefully without exposing sensitive system information. 5. **Output/Notification** ◦ Upon a successful search, display the list of accounts in a clear, readable format. ◦ If no accounts match the search criteria, display a message indicating no results found. 6. **Logging** ◦ All search queries should be logged with timestamp, user ID, and search parameters. ◦ Failed login attempts or unauthorized access attempts should also be logged for security auditing.	
Filter Accounts	As a user, I need the ability to filter bank accounts effectively. This feature will allow me to quickly locate specific accounts based on set criteria, enhancing my efficiency and ability to manage customer accounts.	1. **Functionality** ◦ The system should provide multiple filtering options, including account number, account type, customer name, and account status. ◦ Filters should be able to be used in combination for more refined results. 2. **Security** ◦ Ensure that user authentication is verified before access to the filter feature is granted. ◦ Implement role-based access control; only authorized personnel (e.g., admin users) should have access to the filter accounts feature. 3. **Validation** ◦ The system should validate input fields to ensure that they contain appropriate data formats (e.g., numeric for account numbers). ◦ If a user enters invalid filter criteria, a prompt should inform them of the correct format. 4. **Error Handling** ◦ If the system fails to execute a filter query, it should display a user-friendly error message. ◦ Error messages should not expose sensitive system details or data.	

		5. **Output/Notification** ◦ Upon successful application of filters, the system should display the filtered results in an organized and readable format. ◦ If no accounts match the filter criteria, a message should inform the user that no results were found. 6. **Logging** ◦ All filter operations should be logged with relevant details such as user ID, timestamp, and filter criteria used. ◦ Any errors encountered during the filtering process should also be logged for troubleshooting and audit purposes.	
Sort Accounts	As a user, I want to sort bank accounts so that I can efficiently manage and view accounts based on specific criteria such as account number, account type, balance, or date created.	1. Functionality ◦ The system should allow users to sort accounts by various criteria including account number, account type, balance, and date created. ◦ The sorting should be toggleable between ascending and descending order. 2. Security ◦ Ensure that all data displayed is in compliance with data protection regulations specific to banking information. ◦ User authentication should be verified before granting access to sorting functionality. 3. Validation ◦ Validate user inputs for sorting criteria to ensure they match predefined formats (e.g., numeric for account numbers, date format for dates). ◦ If invalid inputs are detected, they should not be processed. 4. Error Handling ◦ In case of a system error during sorting, an appropriate error message should be displayed to the user. ◦ Errors should be logged with sufficient detail for debugging purposes. 5. Output/Notification ◦ Once sorting is applied, the user interface should update to reflect the new order of accounts. ◦ If no accounts meet the sorting criteria, display a message indicating no results found. 6. Logging ◦ All sorting actions performed by users should be logged with relevant details including user ID, timestamp, and sorting criteria.	

		◦ Any failed attempts or errors during sorting should also be logged for audit purposes.	

Card Management - Admin Portal - Banking App - FS Java Capstone

Card Management

Story	Description	Acceptance Criteria	Story Points
Admin Portal: Card Creation			
Manually Create Card	As a user, I need the ability to manually create cards for customers. This feature should enable me to input necessary card details and submit them for card creation, ensuring a seamless and secure process.	1. **Functionality** ◦ Ability to enter card details including card type, customer name, card limit, and expiration date. ◦ Submission button to send the card creation request. 2. **Security** ◦ Ensure secure HTTPS connections for all data transmissions. ◦ Implement session timeouts after periods of inactivity. 3. **Validation** ◦ Validate all input fields for correct data format (e.g., name, numbers). ◦ Mandatory fields should be clearly marked and validated to ensure they are not left empty. 4. **Error Handling** ◦ Display user-friendly error messages for input validation failures. ◦ Implement catch-all error handling for unforeseen errors, displaying a generic error message. 5. **Output/Notification** ◦ Upon successful submission, display a confirmation message. ◦ For unsuccessful submissions, provide clear notifications indicating the reason. 6. **Logging** ◦ Log all card creation activities with timestamps and user details.	

		◦ Implement error logging for debugging and auditing purposes.	
Data Upload	As a user, I need the ability to securely upload and manage data related to card creation. This feature should allow me to efficiently and accurately upload relevant data while ensuring data security and handling errors appropriately.	1. **Functionality** ◦ The system allows the user to upload data files for card creation. ◦ Uploaded data is used to generate new card details within the admin portal. ◦ The system provides an interface for selecting and uploading files. ◦ Only specific file formats (e.g., CSV, XLSX) are accepted for upload. 2. **Security** ◦ The system implements secure file upload protocols. ◦ Data transmission during upload is encrypted using standard encryption methods. ◦ Access to the upload feature is restricted to authorized users only. 3. **Validation** ◦ The system validates the file format and size before uploading. ◦ Data within the file is validated for format and completeness. ◦ Invalid or incomplete files are rejected with a clear message to the user. 4. **Error Handling** ◦ The system displays descriptive error messages for issues like file format incompatibility, upload failures, or data validation errors. ◦ Errors are logged with details such as timestamp, user ID, and error nature. 5. **Output/Notification** ◦ Upon successful upload and data processing, the system notifies the user with a confirmation message. ◦ In case of errors or rejections, the user receives a notification detailing the issue. 6. **Logging** ◦ All upload attempts (successful or failed) are logged. ◦ Logs contain details of the operation, user information, timestamp, and nature of the data uploaded.	
Admin Portal: Card Data Management			
Update Card Details	As a user, I want to be able to update card details through the Admin Portal's Card Data Management feature, ensuring that card	1. **Functionality** ◦ The system allows the update of card details such as cardholder name, expiry date, and	

	information is current and accurate. This process should be secure, intuitive, and provide clear feedback.	card status. ◦ The updated information is immediately reflected in the system upon submission. 2. **Security** ◦ Ensure secure HTTPS connections for all data transmissions. ◦ User access control is enforced, allowing only authorized personnel to update card details. ◦ Implement timeout for sessions to prevent unauthorized access. 3. **Validation** ◦ Input fields such as card number, expiry date, and cardholder name have format validations. ◦ The system checks for the existence of the card in the database before allowing updates. ◦ Any updates on critical fields (e.g., card number) require additional confirmation from the user. 4. **Error Handling** ◦ Display user-friendly error messages for invalid inputs or system failures. ◦ Prevent submission of incomplete or improperly formatted data. ◦ In case of system errors, ensure that the user can retry or contact support. 5. **Output/Notification** ◦ Upon successful update, a confirmation message is displayed to the user. ◦ For any update, an email notification is sent to the cardholder, if applicable. 6. **Logging** ◦ All card update activities are logged with user ID, timestamp, and details of the changes. ◦ Unauthorized access attempts are also logged and reported.	
Delete Card	As a user, I need the ability to securely delete a card from the admin portal of our banking application, ensuring that the card data is accurately removed without affecting other system functionalities.	1. **Functionality** ◦ The system allows deletion of card data by authorized users. ◦ Deletion action is accessible from the card management section of the admin portal. ◦ Confirmation prompt is displayed before final deletion to prevent accidental data loss. 2. **Security** ◦ Only users with admin privileges can access and execute the delete card functionality. ◦ Deletion request must pass through secure, authenticated API calls. ◦ Session-based user authentication is required to ensure valid user actions.	

		3. **Validation** ◦ The system validates that the card to be deleted exists in the database. ◦ Input fields for identifying the card (e.g., card number) are validated for correct format. 4. **Error Handling** ◦ If the card does not exist, display an appropriate error message. ◦ Display error messages for unauthorized access attempts. ◦ Handle network or server errors gracefully, informing the user to try again later. 5. **Output/Notification** ◦ Post successful deletion, display a confirmation message to the user. ◦ In case of failure, display a descriptive error message indicating the reason. 6. **Logging** ◦ Log all delete actions, including user ID, timestamp, and card details. ◦ Record unsuccessful deletion attempts with error specifics for auditing purposes.	
Admin Portal: Card Search			
Filter Cards	As a user, I want to filter cards, so that I can efficiently locate and manage specific bank cards based on various criteria like card number, user name, card status, etc.	1. **Functionality** ◦ Ability to filter cards based on multiple criteria: card number, user name, card type, issuance date, and status. ◦ Filters should be able to work in combination, allowing for more refined search results. ◦ Results update in real-time as filters are applied. 2. **Security** ◦ Ensure that all card data displayed is compliant with data protection regulations. ◦ Implement role-based access control: Only authorized personnel should access the card management features. ◦ Secure the communication of filter queries from the front-end to the back-end to prevent data interception. 3. **Validation** ◦ Input fields for filtering (e.g., card number, user name) should have validation to prevent invalid queries. ◦ Display a message when a user inputs an invalid search criterion (e.g., non-numeric characters in the card number field). 4. **Error Handling**	

		◦ If a query fails or times out, display a user-friendly error message. ◦ Implement a 'Retry' option for failed searches. ◦ Ensure that system stability is maintained during backend failures. 5. **Output/Notification** ◦ Display the number of cards found as a result of the applied filters. ◦ If no results are found, display a message indicating that no matching cards were found. 6. **Logging** ◦ Log all filter and search activities with timestamps for audit purposes. ◦ Any errors encountered during the filtering process should be logged with detailed information for troubleshooting.	
View Cards	As a user, I want to be able to view details of various cards. This feature should enable me to quickly find specific cards based on defined criteria, ensuring a seamless and secure user experience.	1. **Functionality** ◦ The system should allow the user to search for cards by entering specific criteria (e.g., card number, account holder's name). ◦ Search results should display relevant card details such as card number, cardholder name, card type, issue and expiry dates. ◦ The user should be able to view a detailed individual card view by selecting a card from the search results. 2. **Security** ◦ All card data should be displayed in a secure manner, complying with relevant data protection regulations. ◦ Sensitive card information like full card numbers should be masked, showing only the last four digits. ◦ User access to the card management feature should be role-based, allowing only authorized personnel. 3. **Validation** ◦ The search function should validate input criteria, ensuring they meet predefined formats (e.g., correct card number format). ◦ The system should prompt the user with appropriate messages if the input format is incorrect. 4. **Error Handling** ◦ In case of a failed search (e.g., no matches found), the system should display a relevant message to the user, like "No results found for the entered criteria." ◦ Any system errors (e.g., server issues) should be communicated to the user through a clear	

		message, such as "Service temporarily unavailable. Please try again later." 5. **Output/Notification** ◦ Upon successful search, the results should be displayed in an easy-to-read format. ◦ The system should notify the user upon successful or unsuccessful completion of actions (e.g., a message confirming the viewing of a card's details). 6. **Logging** ◦ All user actions within the card management feature should be logged for audit purposes, including search queries and viewed card details. ◦ Logs should capture necessary information such as user ID, timestamp, and the nature of the action performed.	
Sort Cards	As a user, I want to sort cards. so that I can easily manage and view them based on specific criteria like card type, status, or date of issue.	1. **Functionality** ◦ The system shall provide options to sort cards by card type, status, issue date, expiry date, and user name. ◦ The sorting feature shall allow both ascending and descending order sorting. ◦ Sorting preferences shall be retained during the session even after navigating away from the page. 2. **Security** ◦ Access to the sorting feature shall be restricted to authenticated users with the necessary permissions. ◦ The system shall ensure that sorting requests do not expose any sensitive cardholder information. 3. **Validation** ◦ The system shall validate the sorting criteria (e.g., valid card types, status codes) before applying the sort. ◦ Invalid sorting requests shall prompt an informative error message to the user. 4. **Error Handling** ◦ In case of a failure in sorting (e.g., server error, timeout), the system shall display an error message. ◦ The error message shall not contain sensitive system information. 5. **Output/Notification** ◦ Upon successful sorting, the sorted card list shall be displayed without a page reload. ◦ If no cards meet the sorting criteria, a message indicating “No cards found” shall be displayed.	

		6. **Logging** ◦ All sorting actions shall be logged with details including user ID, timestamp, and sorting criteria used. ◦ Any sorting-related errors shall also be logged for troubleshooting purposes.	

User Management - Admin Portal - Banking App - FS Java Capstone

User Management

Story	Description	Acceptance Criteria	Story Points
Admin Portal: User Authentication/Authorization			
Account Logout	As a user, I need the ability to securely log out of the portal, ensuring that my session is ended properly to maintain security and prevent unauthorized access.	1. **Functionality** ◦ The system provides a clearly visible and accessible logout option in the user interface. ◦ On selecting the logout option, the user's session is terminated. ◦ After logout, the user is redirected to the login page. 2. **Security** ◦ The logout process clears all session data associated with the user. ◦ Ensures that once logged out, the back button or session history does not allow access to the authenticated areas of the application. 3. **Validation** ◦ The system validates the request to ensure it is from an authenticated and active session before processing the logout. 4. **Error Handling** ◦ If the logout process fails, the system displays a clear and understandable error message. ◦ Provides the user with the option to retry the logout process. 5. **Output/Notification** ◦ Upon successful logout, the user receives a confirmation message or notification. ◦ If redirected to the login page, a message indicating a successful logout can be displayed for clarity. 6. **Logging** ◦ The system logs the logout action, including timestamp and user identifier, for audit and tracking purposes.	

		◦ Any errors encountered during the logout process are also logged for troubleshooting and security auditing.	
Account Login	As a user, I need the ability to securely log in to access and manage user accounts and permissions.	1. **Functionality** ◦ The login page should accept username and password inputs. ◦ A 'Log In' button should be available to submit the credentials. 2. **Security** ◦ Implement SSL encryption for data transmission. ◦ Passwords must not be visible or stored in plain text at any point. 3. **Validation** ◦ Validate that both username and password fields are not empty before submission. ◦ Check the format of the username to ensure it meets specified criteria (e.g., email format). 4. **Error Handling** ◦ If login fails, display a generic error message such as "Invalid username or password." ◦ After three consecutive failed login attempts, temporarily disable the login function for that user for a predetermined time. 5. **Output/Notification** ◦ Upon successful login, redirect the user to the admin dashboard. ◦ If login is unsuccessful, clear the password field. 6. **Logging** ◦ Log all login attempts, successful or unsuccessful, with a timestamp and user identifier. ◦ In the case of failed login attempts, log the reason for failure.	
User Authorization	As an administrator of the banking application, I need the ability to manage user authorizations within the Admin Portal, ensuring that users have the correct access rights based on their roles and responsibilities.	1. **Functionality** ◦ The API must authenticate admin users by verifying their credentials (username and password) against the database. ◦ The API should authorize users based on their roles, granting access to specific functionalities within the admin portal. ◦ The system must provide a secure session token upon successful authentication, which will be used for subsequent authorization checks. 2. **Security** ◦ Implement Authorization mechanisms to ensure only legitimate users can access the admin portal. ◦ Utilize encryption for sensitive data transmission, including passwords and session tokens, using industry-standard protocols (e.g., TLS). 3. **Validation** ◦ Perform input validation to ensure that username and password fields are not empty, meet the application's	

		requirements for length and format, and prevent SQL injection or other common security vulnerabilities. 4. **Error Handling** ◦ Gracefully handle authentication and authorization errors, such as incorrect credentials or unauthorized access attempts, without causing the program to crash. ◦ Provide meaningful error messages to the backend logs, avoiding any direct error details sent to the user that could expose system vulnerabilities. 5. **Database Updates** ◦ Update the user's login timestamp upon successful authentication. ◦ Record unsuccessful login attempts, locking the account after a predefined number of failed attempts, if applicable. 6. **Output/Notification** ◦ Return a secure session token upon successful authentication. ◦ In case of failure, return an appropriate error status code and a generic error message indicating the failure reason without exposing specific details. 7. **Logging** ◦ Log all API requests, including access timestamps, requesting user identifier, IP address, and the outcome of the request (success/failure) for auditing, error handling, and debugging purposes. ◦ Ensure that sensitive data, such as passwords, are not logged.	
Admin Portal: User Creation			
Manually Create User	As a user with administrative privileges in the banking application, I need the ability to manually create new user accounts through the admin portal. This process should be secure, validate user input, handle errors gracefully, and provide appropriate notifications and logging.	1. **Functionality** ◦ The admin portal should include a form to enter new user details (e.g., name, email, role). ◦ The system should specify the user’s role. ◦ A 'Create User' button should submit the form data for user creation. 2. **Security** ◦ Ensure all data transmission is encrypted. ◦ Implement role-based access control; only users with admin privileges can access the user creation feature. 3. **Validation** ◦ Validate all input fields in the form to ensure they meet the required format (e.g., email format, mandatory fields). ◦ Display an informative message if validation fails. 4. **Error Handling** ◦ In case of submission errors (e.g., network issues), display a clear error message. ◦ Prevent submission if mandatory fields are missing or incorrectly formatted. 5. **Output/Notification**	

		◦ Upon successful user creation, display a confirmation message. ◦ If user creation fails, provide a detailed error message explaining the reason. 6. **Logging** ◦ Log all user creation attempts, including successful and failed attempts, with timestamps. ◦ Store logs securely and make them accessible only to authorized personnel.	
Data Upload	As a user, I need the ability to upload data securely and efficiently through the Portal, ensuring that the data is validated, error-handled, and logged appropriately for future reference and audit trails.	1. **Functionality** ◦ The system shall allow the upload of specific data file formats (e.g., CSV, XLSX). ◦ The system shall provide an option to preview the data before final submission. 2. **Security** ◦ The system shall implement secure file upload practices, including file type restrictions and size limits. ◦ User authentication and authorization checks shall be performed before allowing access to the data upload feature. 3. **Validation** ◦ The system shall validate the file format and content, ensuring it adheres to predefined data structure requirements. ◦ The system shall display a message if the file does not meet the validation criteria and reject the upload. 4. **Error Handling** ◦ The system shall provide clear error messages in case of upload failures due to network issues, file corruption, or unexpected system behavior. ◦ The system shall ensure that partial or failed uploads do not corrupt existing data. 5. **Output/Notification** ◦ Upon successful upload, the system shall display a confirmation message to the user. ◦ The user shall receive notifications regarding the status of the data upload process. 6. **Logging** ◦ All data upload attempts (successful or unsuccessful) shall be logged with timestamp, user ID, and a description of the action. ◦ The system shall maintain a log of any discrepancies or errors encountered during the upload process.	
Admin Portal: User Data Management			
Update User Details	As a user, I need the ability to update user details so that I can ensure the accuracy and currency of user information in the system.	1. **Functionality** ◦ The system should allow the admin to search for users by their unique identifiers (e.g., user ID, username). ◦ The system should allow the user to update roles	

		◦ Once a user is selected, the system should display the current user details. ◦ The admin should be able to edit fields such as name, email address, and phone number. ◦ Changes should be saved only when the admin clicks the 'Save' button. 2. **Security** ◦ Ensure that only authenticated admins can access the user update functionality. ◦ Implement role-based access control to restrict editing permissions to authorized admin roles. ◦ Changes to user details should be conducted over a secure, encrypted connection. 3. **Validation** ◦ The system should validate input formats (e.g., proper email format, phone number length). ◦ Mandatory fields should be clearly marked, and the system should not allow saving until these fields are populated. ◦ Display appropriate messages for validation failures. 4. **Error Handling** ◦ In case of a system error (e.g., server not responding), display a user-friendly error message. ◦ Provide a clear error message if the update action fails (e.g., "Update failed due to network error"). ◦ Ensure that the system does not crash or become unresponsive during an error state. 5. **Output/Notification** ◦ Upon successful update, display a confirmation message (e.g., "User details updated successfully"). ◦ If the update is unsuccessful, provide a clear notification stating the reason (e.g., "Update failed: Email already in use"). 6. **Logging** ◦ All changes to user details should be logged with timestamps, admin user ID, and a summary of the changes. ◦ Attempted updates, successful or not, should also be logged for audit purposes. ◦ Maintain logs in a secure and compliant manner according to banking application standards.	
Delete User	As a user with administrative privileges in the banking application, I need the ability to delete user accounts from the admin portal. This functionality should ensure secure and accurate deletion of user data, providing feedback and logs for auditing purposes.	1. **Functionality** ◦ The admin portal must allow the deletion of user accounts. ◦ The deletion process should be reversible until confirmed. ◦ Once confirmed, the record must no longer be visible on the front-end. ◦ Because they are not deleting the data on the backend for audit purposes, but on the frontend it will no longer show the record since it is deleted 2. **Security**	

		◦ Ensure only users with administrative privileges can access the delete function. ◦ Implement a two-step verification process for account deletion (e.g., password confirmation and a CAPTCHA). 3. **Validation** ◦ Verify the existence of the user account before initiating deletion. ◦ Check for any dependencies or linked data that may be affected by the deletion. 4. **Error Handling** ◦ If deletion fails, provide a clear error message indicating the reason. ◦ Handle all exceptions gracefully and maintain system stability. 5. **Output/Notification** ◦ Upon successful deletion, notify the admin with a confirmation message. ◦ In case of failure, provide an appropriate error message to the admin. 6. **Logging** ◦ Log all account deletion attempts, successful or not. ◦ Include timestamp, admin user ID, and the target user account ID in the logs.	
Admin Portal: User Search			
View User	As a user, I want to be able to view detailed user information within the admin portal of our Banking Application, enabling me to manage user accounts efficiently.	1. **Functionality** ◦ The feature must allow the viewing of individual user profiles. ◦ It should provide comprehensive details including name, account number, contact information, and activity logs. ◦ The user interface should be intuitive, allowing easy navigation to the user profile section. 2. **Security** ◦ Ensure all user data displayed is protected and complies with relevant data protection regulations. ◦ Implement role-based access controls: only authorized personnel (e.g., admins) should access detailed user profiles. ◦ Sensitive data like account numbers must be partially masked. 3. **Validation** ◦ The system should validate that the user being viewed exists in the database. ◦ There should be checks to ensure that the user requesting the information has the necessary permissions. 4. **Error Handling** ◦ In case of an invalid user search (e.g., non-existent user ID), display a clear error message. ◦ For unauthorized access attempts, prompt a permission-denied message.	

		◦ Handle server or network-related errors gracefully and inform the user of the issue. 5. **Output/Notification** ◦ Upon successful retrieval of a user's profile, display the information in a clear, organized format. ◦ Notify the user when their view request is being processed and when it's completed. 6. **Logging** ◦ Log all attempts to view user profiles, including successful and unsuccessful attempts. ◦ Record the user ID of the person accessing the information, the time of access, and the profile viewed.	
Filter User	As a user, I need the ability to filter users within the admin portal of the banking application so that I can easily find and manage user accounts based on specific criteria.	1. **Functionality** ◦ The system shall provide options to filter users based on predefined criteria such as name, account number, email, or role. ◦ The filtering action shall display a list of users that match the criteria without refreshing the entire page. 2. **Security** ◦ All user filter operations shall be conducted over a secure connection. ◦ User data displayed as a result of the filtering process shall be encrypted to prevent unauthorized access. 3. **Validation** ◦ Input fields for filtering criteria shall validate data format (e.g., email format, numeric values for account numbers). ◦ Invalid inputs shall prompt an appropriate message and shall not trigger the search operation. 4. **Error Handling** ◦ In case of a system error during the filtering process, the system shall display a user-friendly error message. ◦ The system shall handle server timeouts or failures gracefully, informing the user to try again later. 5. **Output/Notification** ◦ Upon successful application of filters, the system shall display the number of users found. ◦ If no users match the criteria, a message stating "No users found" shall be displayed. 6. **Logging** ◦ Each filter operation shall be logged with details including user ID of the administrator, timestamp, and filter criteria used. ◦ Any errors encountered during the filtering process shall also be logged for future analysis and system improvements.	
Sort User	As as a user, I want the ability to sort users within the Portal, so that I can efficiently manage and locate user accounts based on specific	1. **Functionality** ◦ The system allows sorting of user accounts by multiple fields, including but not limited to name, account type, and creation date.	

	criteria such as name, account type, creation date, etc.	◦ Admins can toggle between ascending and descending order for each sortable field. 2. **Security** ◦ Sort functionality adheres to the existing security model of the Admin Portal. ◦ Only users with admin privileges can access and utilize the sort feature. ◦ The sorting process does not expose any sensitive user information beyond what is already permissible for the admin to view. 3. **Validation** ◦ The system validates admin actions for sorting, ensuring that only legitimate sort queries are processed. ◦ Invalid sort requests (e.g., sorting by a non-existent field) are identified and prevented. 4. **Error Handling** ◦ In case of a failure during sorting (e.g., server timeout), the system displays a relevant error message to the admin. ◦ The system ensures that failed sort actions do not affect the current state or visibility of user accounts. 5. **Output/Notification** ◦ Post successful sorting, the system updates the user list interface to reflect the new order. ◦ If no users meet the sorting criteria, a message is displayed indicating "No users found matching the sorting criteria." 6. **Logging** ◦ All admin actions related to sorting users are logged for auditing purposes. ◦ Logs include details such as the admin ID, timestamp, and the nature of the sort action performed.	

Loan Management - Admin Portal - Banking App - FS Java Capstone

Loan Management

- Admin Portal: Loan Creation
- Admin Portal: Loan Data Management
- Admin Portal: Loan Search

Story	Description	Acceptance Criteria	Story Points
Admin Portal: Loan Creation			
Manually Create Loan	As a user, I require the ability to manually create loans. This feature will enable me to input and manage loan details, ensuring that they are correctly set up in the system for further processing and management.	1. **Functionality** ◦ The system should allow the user to input all necessary loan details, including loan amount, interest rate, term, and borrower information. ◦ Ensure that the user can submit the loan information to be saved in the system. ◦ The system should confirm successful creation and display the newly created loan in the admin portal. 2. **Security** ◦ Implement role-based access control, ensuring only authorized users can create loans. ◦ Sensitive data, such as borrower information, should be encrypted and transmitted securely. ◦ Session timeouts should be enforced to prevent unauthorized access. 3. **Validation** ◦ Input fields for loan details should be validated for correct data formats and value ranges (e.g., interest rate percentages, term duration). ◦ Mandatory fields should be clearly marked, and the system should not allow submission of the loan creation form unless all mandatory fields are filled. 4. **Error Handling** ◦ If a submission fails, display a clear and specific error message to the user. ◦ Errors should be logged with enough detail for debugging, but without exposing sensitive user data.	

		◦ The system should handle server-side errors gracefully and inform the user if the problem persists. 5. **Output/Notification** ◦ Upon successful creation of a loan, display a confirmation message to the user. ◦ If applicable, send notifications to relevant stakeholders or systems about the new loan creation. 6. **Logging** ◦ All loan creation attempts (both successful and unsuccessful) should be logged with timestamps and user details. ◦ Logs should include information about the loan details entered, without logging sensitive borrower information. ◦ The system should maintain an audit trail for compliance and tracking purposes.	
Data Upload	As a user, I want to upload loan data to the admin portal so that I can manage loan applications efficiently and accurately.	1. **Functionality** ◦ Ability to upload files containing loan data. ◦ Support for multiple file formats including CSV, Excel, and XML. ◦ Real-time display of upload progress. ◦ Confirmation of successful upload. 2. **Security** ◦ File uploads must be authenticated and authorized. ◦ Implementation of secure file upload practices to prevent unauthorized access and data breaches. ◦ Files should be scanned for malware before processing. 3. **Validation** ◦ Automatic validation of file format and structure. ◦ Validation of data consistency and integrity within the file. ◦ Clear error messages for validation failures. 4. **Error Handling** ◦ Handling of common file upload errors (e.g., connection loss, file too large). ◦ Display of user-friendly error messages. ◦ Option to retry the upload process after an error. 5. **Output/Notification** ◦ Notification upon successful or unsuccessful upload. ◦ Detailed report of any data inconsistencies or errors detected during upload.	

		◦ Confirmation dialog before finalizing the upload. 6. **Logging** ◦ Logging of all file upload attempts, successes, and failures. ◦ Capture of user details, timestamp, and file metadata in logs. ◦ Secure storage of logs with restricted access.	
Admin Portal: Loan Data Management			
Delete Loan	As a user, I need the ability to securely and efficiently delete loan data from the admin portal. This feature should ensure accurate deletion of records, proper validation, and clear feedback to prevent unintended loss of data and maintain system integrity.	1. **Functionality** ◦ The feature allows the admin to select and delete specific loan records. ◦ Only loan records that are not currently linked to active transactions can be deleted. ◦ A confirmation prompt appears before final deletion to ensure intentional action. 2. **Security** ◦ Access to the delete function is restricted to users with admin privileges. ◦ Each deletion request requires re-authentication of the user for added security. 3. **Validation** ◦ The system validates the existence of the loan record before proceeding with deletion. ◦ If the loan record does not exist or is already deleted, an appropriate message is displayed. 4. **Error Handling** ◦ In the event of a system error during deletion, the operation is halted, and an error message is displayed. ◦ The system ensures that no partial deletion occurs, maintaining data integrity. 5. **Output/Notification** ◦ Upon successful deletion, a confirmation message is displayed to the user. ◦ If deletion is not possible (e.g., linked to active transactions), a notification explains the reason. 6. **Logging** ◦ All deletion actions, along with timestamp and admin user ID, are logged for audit purposes. ◦ Failed deletion attempts are also logged with relevant error details.	
Update Loan Details	As a user, I want to update loan details through the Admin Portal. This feature should enable me to modify existing loan records, ensuring that the most current and accurate information is reflected in our system	1. **Functionality** ◦ Ability to select a specific loan record from the list in the Admin Portal. ◦ Provide fields to update various loan details such as loan amount, interest rate, tenure,	

		borrower's information, etc. ◦ Include a 'Save Changes' button to confirm the updates. 2. **Security** ◦ Ensure that only authenticated users with admin privileges can access the loan update feature. ◦ Implement role-based access control to restrict unauthorized modifications. 3. **Validation** ◦ Validate input fields to accept only permissible data formats and values. ◦ Display clear error messages for invalid inputs. 4. **Error Handling** ◦ Implement graceful error handling for scenarios like network failures or server errors. ◦ Display user-friendly error messages in case of update failures. 5. **Output/Notification** ◦ Show a confirmation message/notification upon successful update of loan details. ◦ In case of update failure, provide a notification detailing the reason for failure. 6. **Logging** ◦ Log all actions performed during the update process, including successful updates and encountered errors, with timestamp and user details.	
Admin Portal: Loan Search			
Filter Loan	As a user I would like to be able to filter the loan results that I am viewing.	1. **Functionality** ◦ The feature should allow users to apply multiple filters simultaneously (e.g., amount, date range, loan status). ◦ Filters should be intuitive and easy to apply, with clear interface elements like dropdowns, checkboxes, and date pickers. ◦ The system must update the displayed loan data in real-time as filters are applied or changed. 2. **Security** ◦ Ensure that all loan data is transmitted securely using encryption. ◦ Implement role-based access control, allowing only authorized personnel to view or modify loan details. ◦ Filters should not expose any sensitive borrower information beyond the user's access level. 3. **Validation**	

		◦ Inputs for numeric fields (like loan amount) must accept only numeric values and should have upper and lower limits. ◦ Date filters should reject invalid dates and should not allow a start date that is later than the end date. ◦ The system should validate the status field against predefined loan statuses. 4. **Error Handling** ◦ If the filter query fails or returns an error, the system should display a user-friendly error message. ◦ In the event of a server or connectivity issue, the system should alert the user and suggest retrying later. ◦ Invalid filter inputs should prompt users with clear, instructive error messages. 5. **Output/Notification** ◦ Upon applying filters, the system should display the number of loans found. ◦ If no loans match the filter criteria, the system should display a message indicating "No loans found" or a similar notification. ◦ The system should provide visual feedback (like a spinner) when processing filter requests. 6. **Logging** ◦ All filter operations should be logged with relevant details (user ID, timestamp, filter criteria used). ◦ Error logs should capture any exceptions or failures in the filter functionality. ◦ Access logs must record each instance of loan data viewing or modification for audit purposes.	
View Loan	As a user, I need the ability to view details of specific loans within the admin portal so that I can manage and oversee loan-related activities effectively.	1. **Functionality** ◦ The system shall allow the admin user to search for loans using specific criteria (e.g., loan ID, customer name). ◦ Upon selecting a loan, detailed information about the loan should be displayed, including loan amount, duration, interest rate, customer details, and repayment status. 2. **Security** ◦ Access to loan information shall be restricted to authenticated and authorized admin users only. ◦ All data transmissions of loan details shall be encrypted using industry-standard encryption protocols. 3. **Validation** ◦ Input fields for searching loans shall validate for correct format and reject invalid inputs with	

		an appropriate message. ◦ The system shall validate the existence of the loan record in the database before displaying the details. 4. **Error Handling** ◦ If a loan record is not found, the system shall display a clear and friendly error message stating, "Loan record not found. Please check the entered criteria." ◦ In case of a system error or failure, the system shall display a generic error message, "An unexpected error occurred. Please try again later." 5. **Output/Notification** ◦ Upon successful retrieval of loan data, the system shall display the loan details in a clear, readable format. ◦ If no records match the search criteria, the system shall notify the user with a message, "No matching loan records found." 6. **Logging** ◦ All loan search activities by admin users shall be logged with details including user ID, timestamp, and search criteria. ◦ Any errors or exceptions encountered during the loan search process shall be logged for future auditing and troubleshooting purposes.	
Sort Loan	As a user, I want to sort loans in the portal so that I can efficiently manage and organize loan information based on specific criteria such as loan amount, date, status, or customer name.	1. **Functionality** ◦ The system shall provide options to sort loans by various fields including loan amount, date, status, and customer name. ◦ Sorting shall be available in both ascending and descending order for each field. ◦ The sorting feature shall update the display of loans in real-time as the user selects different sorting options. 2. **Security** ◦ Sorting requests shall only be processed if the user is authenticated and authorized to access loan information. ◦ The feature shall not expose any sensitive customer data during the sorting process. ◦ Data transmission during sorting operations shall be encrypted. 3. **Validation** ◦ The system shall validate the user's input for sorting criteria to ensure it matches one of the predefined sortable fields. ◦ If a user attempts to sort by an invalid or unrecognized field, the system shall display an	

		appropriate message and not perform the sorting. 4. **Error Handling** ◦ In the event of a sorting operation failure, the system shall display a generic error message without exposing underlying system details. ◦ The system shall offer the option to retry the sorting operation or return to the previous state. 5. **Output/Notification** ◦ Upon successful sorting, the user interface shall reflect the sorted loan data in the chosen order. ◦ If there are no loans matching the sorting criteria, the system shall display a message indicating 'No loans found matching the criteria'. 6. **Logging** ◦ All sorting actions performed by users shall be logged with details including user ID, timestamp, and sorting criteria used. ◦ Any failed sorting attempts shall also be logged with relevant error codes for further analysis.		

Branch Management - Admin Portal - Banking App - FS Java Capstone

Branch Management

- Admin Portal: Branch Creation
- Admin Portal: Branch Data Management
- Admin Portal: Branch Search

Story	Description	Acceptance Criteria	Story Points
Admin Portal: Branch Creation			
Manually Create Branch	As a user**,** I want to manually create a branch in the admin portal of our banking application, enabling me to add new branch details effectively and securely without backend complexities.	1. **Functionality** ◦ The user can access a form to input branch details. ◦ Inputs include branch name, address, phone number, and manager ID. ◦ The form allows submission of the entered branch details. 2. **Security** ◦ Form submission is protected against CSRF (Cross-Site Request Forgery). ◦ User input is sanitized to prevent XSS (Cross-Site Scripting) attacks. ◦ Users are authenticated and authorized to access the form. 3. **Validation** ◦ All input fields are validated for proper formatting. ◦ Mandatory fields (branch name, address) must be filled out. ◦ Phone numbers and manager IDs are checked for valid patterns. 4. **Error Handling** ◦ Clear error messages are displayed for invalid inputs or submission failures. ◦ If the server cannot be reached, the user is notified of the connection issue. ◦ Form errors do not reset previously entered, valid information. 5. **Output/Notification**	

		◦ Upon successful creation, the user receives a confirmation message. ◦ If creation fails, a descriptive error message is shown. ◦ The user is informed of any required fields left blank or filled incorrectly. 6. **Logging** ◦ All form submissions are logged with timestamps. ◦ Failed attempts (due to validation or server issues) are also logged. ◦ Logs capture user ID and action details without sensitive data.	
Data Upload	As a user, I need the ability to upload data relevant to branch management in the banking application's admin portal. This feature should facilitate the secure and efficient uploading of data without impacting backend processes.	1. **Functionality** ◦ The system should provide an interface to upload data files. ◦ Uploaded data should be strictly for branch management purposes, like branch details. ◦ The front-end should only accept specific file formats (e.g., CSV, XLSX). 2. **Security** ◦ Implement file upload size limits to prevent large file attacks. ◦ Scan all uploaded files for malware or malicious code. ◦ Ensure all data transfers are over secure, encrypted channels (e.g., HTTPS). 3. **Validation** ◦ Validate file format and reject any unsupported formats. ◦ Check the structure of the data in the file to match predefined templates. ◦ Display a message for incorrect or incomplete data files. 4. **Error Handling** ◦ Provide clear error messages for file upload failures (e.g., wrong format, size limit exceeded). ◦ Implement timeout for file upload process and alert the user in case of a timeout. ◦ Ensure the front-end remains responsive in case of upload errors. 5. **Output/Notification** ◦ Display a success message upon successful data upload. ◦ Notify the user of the progress during the upload process. ◦ Show a summary of the uploaded data, such as the number of records processed.	

		6. **Logging** ◦ Log all upload attempts, successful or not, with timestamp and user details. ◦ Record any file rejections with reasons for the rejection. ◦ Store logs in a secure and accessible manner for future audits or troubleshooting.	
Admin Portal: Branch Data Management			
Delete Branch	As a user with administrative privileges in the banking application's admin portal, I need the ability to delete branch information effectively and securely. This functionality is critical for maintaining up-to-date and accurate branch data within the system.	1. **Functionality** ◦ The system should provide an option to delete a specific branch. ◦ The delete action must only be visible and accessible to users with administrative privileges. ◦ Before deletion, the system should prompt for confirmation to prevent accidental deletions. 2. **Security** ◦ Ensure that the delete request is authenticated and authorized, verifying that the user has admin rights. ◦ Implement secure communication protocols (e.g., HTTPS) for transmitting the delete request. 3. **Validation** ◦ The system should validate that the branch exists before attempting deletion. ◦ If the branch is linked to any active transactions or customers, the system should not allow deletion and inform the user. 4. **Error Handling** ◦ In case of a failure in deletion (e.g., server error), the system should display a meaningful error message. ◦ The system should handle any exceptions gracefully, ensuring the application remains stable. 5. **Output/Notification** ◦ Upon successful deletion, the system should notify the user that the branch has been successfully removed. ◦ In case of unsuccessful deletion, the system should provide a clear notification stating the reason. 6. **Logging** ◦ All delete actions should be logged with details including user ID, timestamp, and branch ID. ◦ Any unsuccessful deletion attempts should also be logged for audit purposes.	

Update Branch Details	As a user with administrative privileges in the banking application's admin portal, I need the ability to update branch details so that I can ensure the information is accurate and current.	1. **Functionality** ◦ The feature allows updating of branch details such as name, address, contact details, and operational hours. ◦ Any changes made should be immediately reflected in the system upon submission. 2. **Security** ◦ Access to update branch details is restricted to users with administrative privileges. ◦ All data transmissions are encrypted using industry-standard protocols. ◦ The system validates user permissions before allowing access to the update feature. 3. **Validation** ◦ The system checks for the validity of all input fields (e.g., format of the phone number, email address). ◦ Mandatory fields (e.g., branch name, address) must be filled; the system should not allow submission of incomplete forms. ◦ Input length for each field should be restricted to reasonable limits to prevent data overflow. 4. **Error Handling** ◦ Clear error messages are displayed for invalid inputs or incomplete forms. ◦ In case of a system failure or connection issue, the system should inform the user and advise them to retry. 5. **Output/Notification** ◦ Upon successful update, a confirmation message is displayed to the user. ◦ If the update fails, a notification with a brief description of the issue is provided. 6. **Logging** ◦ All updates to branch details are logged with user ID, timestamp, and nature of the update. ◦ Failed update attempts are also logged for audit and troubleshooting purposes.	
Admin Portal: Branch Search			
View Branch	As a user, I need the ability to view detailed information about different bank branches, so I can manage and oversee branch operations effectively.	1. **Functionality** ◦ The system shall display a searchable list of all bank branches. ◦ Upon selecting a branch, detailed information about the branch should be displayed, including branch name, location, manager, and contact information. ◦ The interface shall refresh in real-time to reflect any updates or changes made to branch information.	

		2. **Security** ◦ Access to the branch information shall be restricted to authenticated users only. ◦ User roles and permissions shall be enforced, allowing only authorized personnel to view sensitive branch data. ◦ All data transmissions shall be encrypted using industry-standard encryption methods. 3. **Validation** ◦ The system shall validate the existence of the branch before displaying its information. ◦ If a branch is not found or is inactive, the system shall not display its information. ◦ Input fields for search parameters shall have validation checks to prevent SQL injection and other common security threats. 4. **Error Handling** ◦ In case of a system error or failure to retrieve branch data, the system shall display a user-friendly error message. ◦ The system shall handle server timeouts and display a message prompting the user to try again later. ◦ All errors shall be logged with sufficient detail for troubleshooting. 5. **Output/Notification** ◦ Upon successful retrieval of branch data, the system shall display the information in a clear, readable format. ◦ If no data is found for the selected branch, a notification shall inform the user that no data is available. ◦ Any changes in the branch data due to real-time updates shall trigger a notification to the user viewing the data. 6. **Logging** ◦ All user actions related to viewing branch information shall be logged with timestamps and user details. ◦ Any unsuccessful attempts to access branch information (e.g., due to lack of permissions) shall also be logged. ◦ System errors and exceptions encountered while viewing branch data shall be logged for audit and troubleshooting purposes.	
Filter Branch	As a user, I want to filter branches so that I can easily find and manage specific branch details based on various criteria like location, branch ID, or services offered.	1. **Functionality** ◦ The feature allows users to filter branches by predefined criteria (e.g., location, branch ID, services offered).	

		<ul><li>Filtering options are available as dropdown menus or text input fields.</li><li>The system displays the filtered results in a clear, tabular format.</li></ul>2. **Security**<ul><li>Access to branch filtering is restricted to users with admin or relevant privileges.</li><li>All data transmissions during filtering operations are encrypted.</li></ul>3. **Validation**<ul><li>Input fields for filtering criteria validate the format (e.g., numeric for branch ID, alphanumeric for location).</li><li>The system displays a message when input does not meet the expected format.</li></ul>4. **Error Handling**<ul><li>If the filtering process fails (e.g., server timeout), the system shows a user-friendly error message.</li><li>Error messages guide the user on possible next steps or suggest trying again later.</li></ul>5. **Output/Notification**<ul><li>Once filtering is complete, the system notifies the user of the number of branches found.</li><li>If no branches match the criteria, the system displays a message indicating "No results found."</li></ul>6. **Logging**<ul><li>All filtering actions are logged with details like user ID, timestamp, and filter criteria used.</li><li>Failed attempts at filtering due to errors are also logged for future analysis.</li></ul>	
Sort Branch	As a user, I need the ability to sort the list of branches in the admin portal of our Banking Application so that I can easily find and manage specific branches based on various criteria such as location, branch name, or status.	1. **Functionality**<ul><li>The system shall provide an option to sort branches by criteria such as branch name, location, and status.</li><li>Users shall be able to toggle between ascending and descending order for each sorting criterion.</li></ul>2. **Security**<ul><li>The sorting feature shall adhere to the application's existing security protocols, ensuring that user data remains protected during sorting operations.</li><li>Only authenticated users with admin privileges shall have access to use the sorting functionality.</li></ul>3. **Validation**<ul><li>The system shall validate user inputs for sorting (e.g., correct format of sorting criteria).</li></ul>	

		◦ Invalid inputs for sorting criteria shall trigger a notification to the user to correct the input. 4. **Error Handling** ◦ In case of a failure in sorting operation (e.g., server timeout), the system shall display a user-friendly error message. ◦ The system shall ensure that any error during sorting does not affect the integrity of the displayed branch data. 5. **Output/Notification** ◦ Upon successful sorting, the system shall display the branches in the specified order. ◦ The system shall provide a visual indicator (e.g., an icon) to show the current sorting criterion and order. 6. **Logging** ◦ All sorting actions performed by users shall be logged with details such as user ID, timestamp, and sorting criteria used. ◦ Error logs shall be generated for any exceptions or failures encountered during the sorting process.	

Transaction Management - Admin Portal - Banking App - FS Java Capstone

Transaction Management

Story	Description	Acceptance Criteria	Story Points
Admin Portal: Transaction Search			
View Transaction	As a user of the admin portal in the Banking Application, I need the ability to view transaction details so that I can effectively manage and review banking transactions.	1. **Functionality** ◦ The system displays transaction details such as transaction ID, date, amount, sender, receiver, and transaction type. ◦ Users can search for transactions using filters like transaction ID, date range, and amount range. ◦ The system provides a clear view option for each transaction in the search results. 2. **Security** ◦ Ensure that only authenticated users with admin rights can access transaction details. ◦ Implement role-based access control to restrict data visibility based on user roles. ◦ Sensitive data like account numbers should be partially masked. 3. **Validation** ◦ Input fields for search filters (like transaction ID, dates, amounts) must validate for correct data types and formats. ◦ Date range fields must not allow a start date that is later than the end date. ◦ The system should validate that amount fields contain only numerical values. 4. **Error Handling** ◦ If a transaction is not found, display a user-friendly message: "No transaction found with the provided criteria." ◦ For invalid search inputs, show an error message detailing the nature of the error. ◦ Handle server-side errors gracefully and display a message: "Error processing request,	

		please try again later." 5. **Output/Notification** ◦ Upon successful retrieval of transaction data, display the results clearly and concisely. ◦ If no data is found, notify the user with a message: "No matching transactions found." ◦ Provide a confirmation message or visual indicator when the search is initiated. 6. **Logging** ◦ Log all user actions related to transaction viewing, such as search queries and viewed transactions. ◦ Record any failed attempts to access the transaction details, noting the user ID and timestamp. ◦ System errors and exceptions should be logged with detailed information for troubleshooting.	
Sort Transactions	As a user, I need the ability to sort transactions efficiently. This feature will enable me to view transactions in a structured order based on various parameters like date, amount, transaction type, etc. The functionality should be intuitive, secure, and error-tolerant, ensuring that I can manage transaction data effectively.	1. **Functionality** ◦ The system shall allow the user to sort transactions by various fields such as date, amount, transaction type, customer name, and account number. ◦ Sorting options shall include ascending and descending order. 2. **Security** ◦ Sorting requests must be authenticated to ensure that only authorized users can perform sorting actions. ◦ The system shall not expose any sensitive transaction data during the sorting process. 3. **Validation** ◦ The system shall validate the user’s input for sorting criteria to ensure it corresponds to predefined fields (e.g., date format, numeric values for amount). 4. **Error Handling** ◦ In cases where sorting criteria are invalid, the system shall display a clear error message indicating the nature of the issue. ◦ If the sorting process fails due to system errors, the user shall receive a notification indicating that the sorting could not be completed. 5. **Output/Notification** ◦ Upon successful sorting, the system shall display the transactions in the sorted order as per the user’s selection. ◦ The system shall notify the user if no transactions meet the sorting criteria. 6. **Logging**	

		◦ All sorting actions performed by users shall be logged with details such as user ID, timestamp, and sorting criteria used. ◦ Any errors encountered during the sorting process shall also be logged for troubleshooting and audit purposes.	
Filter Transactions	As a user, I need the ability to filter transactions within the Admin Portal of our Banking Application, to efficiently manage and review transaction records. This functionality should allow me to apply various filters to refine the transaction list based on specific criteria such as date range, transaction amount, transaction type, account number, and customer ID.As a user, I want to filter transactions, so that I can view a subset of transactions based on specific criteria such as date range, amount range, or transaction type.	1. **Functionality** ◦ The system shall provide an interface for users to select and apply multiple filters to transaction records. ◦ Filters should include: ▪ Date range (From and To dates) ▪ Transaction amount range (Minimum and Maximum) ▪ Transaction type (e.g., Deposit, Withdrawal, Transfer) ▪ Account number ▪ Customer ID ◦ Users shall have the option to apply filters individually or in combination. ◦ The system shall display the filtered results dynamically as filters are applied. 2. **Security** ◦ Access to transaction filtering shall be restricted to authenticated users with admin privileges. ◦ Sensitive data like account numbers and customer IDs must be handled in compliance with data protection regulations. ◦ The system shall enforce secure HTTPS connections for all data transmission. 3. **Validation** ◦ Input fields for date, transaction amount, account number, and customer ID shall include format validation. ◦ The system shall prompt the user with a message if the filter criteria entered are in an invalid format. ◦ Date range inputs must ensure that the 'From' date is earlier than or equal to the 'To' date. 4. **Error Handling** ◦ In case of an error during filtering (e.g., server unavailability), the system shall display a user-friendly error message. ◦ The system shall prevent SQL injection and other common injection flaws during input processing. 5. **Output/Notification**	

		◦ Upon successful application of filters, the system shall display a list of transactions that match the filter criteria. ◦ If no transactions match the criteria, the system shall display a message indicating 'No transactions found with the applied filters.' 6. **Logging** ◦ All attempts to access the transaction filtering feature shall be logged with user ID, timestamp, and action details. ◦ Any failed attempts due to security or validation errors shall also be logged for audit purposes.	
Search Transaction	As a user, I need the ability to search for transactions within the Admin Portal of our Banking Application, to efficiently manage and review transactional data. This feature should be secure, validate inputs, handle errors gracefully, provide clear outputs/notifications, and maintain appropriate logging for audit and troubleshooting purposes.	1. **Functionality** ◦ The system shall provide a search function in the Admin Portal for transactions. ◦ Users shall be able to search by transaction ID, account number, date range, and transaction type. ◦ Search results shall display relevant transaction details like transaction ID, date, amount, account number, and transaction type. ◦ The search shall return results in a paginated format. 2. **Security** ◦ Access to the transaction search feature shall be restricted to authenticated and authorized users only. ◦ Sensitive data in the transaction details shall be masked or encrypted as per data protection guidelines. 3. **Validation** ◦ The system shall validate input fields for correct format (e.g., dates in DD/MM/YYYY format, transaction ID in the specified numeric/string format). ◦ Invalid inputs shall prompt an informative error message to the user, indicating the nature of the input error. 4. **Error Handling** ◦ In the event of a search query failing, the system shall display a generic error message indicating the failure. ◦ Error messages shall not reveal sensitive system information or technical details. 5. **Output/Notification** ◦ Upon successful search, the system shall display the results in a clear, readable format. ◦ If no results are found, the system shall display a message indicating “No transactions found for the given criteria”.	

		6. **Logging** ◦ All search queries shall be logged with necessary details such as (user ID, timestamp, search criteria) for audit purposes. ◦ Errors and exceptions during the transaction search process shall also be logged for future analysis and system improvements.	

Customer Portal - Frontend Features - Banking App - FS Java Capstone

Customer Portal

- Users - Customer Portal - Banking App - FS Java Capstone
- Accounts - Customer Portal - Banking App - FS Java Capstone
- Cards & Loans - Customer Portal - Banking App - FS Java Capstone
- Branches - Customer Portal - Banking App - FS Java Capstone

Users - Customer Portal - Banking App - FS Java Capstone

Users

Story	**Description**	**Acceptance Criteria**	**Story Points**
Customer Portal: User Authentication			
Account Login	As a user, I want to securely log into the banking application's user portal so that I can access and manage my banking information with confidence in the security and reliability of the system.	1. **Functionality** ◦ The login page shall present fields for username and password. ◦ Upon entering credentials, the system shall authenticate the user against the stored credentials. ◦ Successful authentication shall redirect the user to their dashboard. ◦ Unsuccessful authentication attempts shall not allow access. 2. **Security** ◦ All data transmission during login shall be encrypted. ◦ The system shall enforce a limit of unsuccessful login attempts. ◦ After the limit is reached, the account shall be temporarily locked. 3. **Validation** ◦ The system shall validate the format of the username (e.g., email format). ◦ Passwords must meet minimum complexity requirements. ◦ Empty username or password fields shall trigger a prompt for completion. 4. **Error Handling** ◦ Incorrect login details shall trigger a user-friendly error message.	

		◦ Error messages shall not specify whether the username or password was incorrect. ◦ System errors during login shall be handled gracefully and inform the user of the issue without exposing system details. 5. **Output/Notification** ◦ Upon successful login, a welcome message or dashboard overview shall be displayed. ◦ Upon unsuccessful login, a clear but non-specific error message shall be shown. ◦ Notification of account lockout shall be provided after the maximum attempts are exceeded. 6. **Logging** ◦ All login attempts (successful or unsuccessful) shall be logged with timestamp and user identifier. ◦ Account lockouts shall also be logged with details of the triggering event.	
Account Logout	As a user, I want to securely log out of my account in the banking application, ensuring that my session ends properly and my account information remains secure.	1. **Functionality** ◦ The logout function must be easily accessible from the user interface. ◦ Upon clicking the logout button, the user should be logged out of their session. 2. **Security** ◦ The logout process must invalidate the current session token. ◦ Post-logout, the user must be redirected to the login page, ensuring they are fully logged out. 3. **Validation** ◦ The system should verify if the user is logged in before enabling the logout functionality. ◦ If the user is not logged in, the logout option should be disabled or not visible. 4. **Error Handling** ◦ If the logout process fails, the user should receive a clear error message. ◦ The system must not log out the user if the session token invalidation process fails, ensuring the user can attempt to log out again. 5. **Output/Notification** ◦ Upon successful logout, display a confirmation message or notification to the user. ◦ The confirmation message should inform the user that they have been successfully logged out. 6. **Logging** ◦ The system should log the logout action, including timestamp and user identifier, for audit purposes.	

		◦ In case of a logout error, the error details should also be logged for troubleshooting.	
Change Password	As a user, I need the ability to reset my password through the user portal of the banking application, ensuring I can regain access to my account securely and efficiently if I forget my password or need to change it for security reasons.	1. **Functionality** ◦ The user must have an option to initiate a password reset from the login screen. ◦ Upon selecting the password reset option, the user is prompted to enter their registered email address or username. ◦ The system should verify if the entered email/username exists in the database. ◦ If the email/username is valid, an email with a password reset link is sent to the user's registered email address. ◦ The password reset link must redirect the user to a secure page to input a new password. 2. **Security** ◦ The password reset link should be time-sensitive, expiring after a predefined period (e.g., 30 minutes). ◦ All password transmissions should be encrypted. ◦ New passwords must adhere to established security policies (e.g., minimum length, complexity requirements). ◦ The user must be logged out of all other sessions when the password is reset. 3. **Validation** ◦ The system must validate the format of the email/username entered for reset. ◦ New passwords should be checked for compliance with security policies before acceptance. ◦ A confirmation must be required for the new password (user has to enter it twice). 4. **Error Handling** ◦ If the email/username does not exist in the database, display an appropriate, non-specific error message (e.g., "If an account exists with the entered information, a reset email will be sent."). ◦ Proper error messages should be displayed for network issues or system errors during the reset process. ◦ If the new password does not meet security requirements, display a clear message outlining the policy requirements. 5. **Output/Notification** ◦ Upon successful submission of an email/username for reset, notify the user that an email has been sent if the account exists.	

		◦ After successful password reset, display a confirmation message and direct the user to the login page. ◦ Send a notification email to the user upon successful password change for security purposes. 6. **Logging** ◦ Log all password reset attempts, successful or not, with timestamps. ◦ Record the IP address and device information of the user requesting the password reset. ◦ Log any errors or exceptions that occur during the password reset process.	
Customer Portal: User Authorization			
Forgot Password: Account Recovery	As a user, I want the option to reset my password via email or phone number verification so that I can regain access to my account in case I forget my login credentials. This provides a hassle-free way to recover my account and continue using the User Portal without any disruptions.	1. **Functionality** ◦ The "Forgot Password" option should be clearly visible and accessible on the login page. ◦ Clicking this option leads the user to a secure page where they can enter their registered email address or username. 2. **Security** ◦ Implement CAPTCHA or similar verification on the recovery page to prevent automated abuse. ◦ Ensure all data transmission during the password recovery process is encrypted using industry-standard protocols. 3. **Validation** ◦ The system must validate that the entered email address or username exists in the database. ◦ If the input does not match any account, prompt the user with a non-specific error message like "If your account exists, a recovery email will be sent." 4. **Error Handling** ◦ In the event of a system error or downtime during the recovery process, display a friendly error message informing the user to try again later. ◦ The error message should not disclose technical details of the failure. 5. **Output/Notification** ◦ Upon successful input validation, send an email to the registered email address with a password reset link. ◦ The reset link email should include instructions and a time limit for usage. 6. **Logging**	

		◦ Log all attempts to use the "Forgot Password" feature, including successful and failed attempts, without storing personal information. ◦ Record timestamps and IP addresses for all password recovery attempts for security audit purposes.	
User Authorization	As a user, I need the ability to access specific features and information within the Banking Application based on my role, ensuring that my interactions are secure, relevant, and aligned with my permissions	1. **Functionality** ◦ The system should provide different access levels (e.g., Admin, User, Auditor) with distinct permissions. ◦ Features and data accessible to a user must correspond to their assigned role. 2. **Security** ◦ Role assignments must be encrypted and securely stored. ◦ The system should prevent unauthorized role elevation or modification. 3. **Validation** ◦ All role assignments must be validated against an authorized list of roles. ◦ Input for role changes must be validated for format and legitimacy. 4. **Error Handling** ◦ The system should display user-friendly error messages for unauthorized access attempts. ◦ Errors in role assignment or modification should be logged and reported to the admin. 5. **Output/Notification** ◦ Users should receive confirmation notifications upon successful role assignment or change. ◦ Administrators should be notified of any unauthorized access attempts or role modifications. 6. **Logging** ◦ All role assignment changes should be logged with user ID, timestamp, and role details. ◦ Unauthorized access attempts and system errors related to role management must be logged for audit purposes.	
Customer Portal: User Registration			
Register User	As a user, I want to register for an online banking account through the User Portal so that I can access and manage my banking services online securely and efficiently.	1. **Functionality** ◦ The registration form should capture essential information such as: name, email, password, and phone number. ◦ Upon submission, the system should verify the data and create a new user account. 2. **Security** ◦ Ensure HTTPS is used for all data transmissions.	

		◦ Passwords must be encrypted and securely stored. ◦ Implement CAPTCHA to prevent automated registrations. 3. **Validation** ◦ Validate email format and uniqueness against existing users. ◦ Passwords must meet complexity requirements (minimum length, alphanumeric, and special character inclusion). ◦ Phone numbers should be validated for format and optionally for country code. 4. **Error Handling** ◦ Provide user-friendly error messages for incorrect or incomplete form submissions. ◦ Implement timeouts for registration attempts to prevent overloading the server. 5. **Output/Notification** ◦ On successful registration, display a confirmation message. ◦ Send a welcome email to the user's registered email address. 6. **Logging** ◦ Log all registration attempts, successful or unsuccessful, with timestamp etc. ◦ Record any system errors during the registration process for debugging purposes.	
User Registration Confirmation	As a user, I want to receive a confirmation after completing my registration process in the banking application's user portal, so that I can be sure my account has been successfully created and is secure.	1. **Functionality** ◦ The system should display a confirmation message upon successful completion of the user registration process. ◦ This confirmation should be visible immediately after the user submits their registration details. 2. **Security** ◦ The confirmation process must include a secure, unique link sent to the user's provided email address. ◦ This link should expire after a set period, for example, 24 hours, to prevent unauthorized access. 3. **Validation** ◦ Ensure the email entered by the user during registration is in a valid format. ◦ The system should verify that the email address is not already associated with an existing account. 4. **Error Handling** ◦ If the registration process fails, display a clear and specific error message.	

		◦ Offer the user the option to retry registration or contact support if the error persists. 5. **Output/Notification** ◦ Upon clicking the confirmation link, the user should receive a notification of successful account activation. ◦ If the link has expired or is invalid, notify the user and provide the option to resend the confirmation email. 6. **Logging** ◦ Log all successful and unsuccessful registration attempts, including time stamps and user details. ◦ Record any errors encountered during the registration and confirmation process for further analysis.	
Customer Portal: User Profile Management			
Account Deletion	As a user, I want the ability to permanently delete my banking application account, ensuring that all my personal and financial data is completely removed from the system, while maintaining the highest security and error handling standards.	1. **Functionality** ◦ The user can initiate account deletion from the User Profile section. ◦ Confirm deletion through a two-step verification process. 2. **Security** ◦ Require re-authentication (e.g., password, OTP) before accessing the deletion feature. ◦ Implement a secure method to ensure complete data erasure. 3. **Validation** ◦ Validate user identity before processing the deletion request. ◦ Check for any pending transactions or disputes before allowing account deletion. 4. **Error Handling** ◦ Display clear error messages for any failed deletion attempts. ◦ Handle exceptions, such as server unavailability, with user-friendly notifications. 5. **Output/Notification** ◦ Provide a confirmation message upon successful account deletion. ◦ Email the user a confirmation of account deletion for record-keeping. 6. **Logging** ◦ Log all account deletion activities for auditing purposes. ◦ Record user’s final actions and timestamp of account deletion.	

Application Notifications	As a user, I want to receive notifications through the User Portal in my Banking Application, so that I am promptly informed about important account activities, updates, and alerts.	1. **Functionality** ◦ Notifications are automatically generated for account activities such as transactions, balance updates, and security alerts. ◦ Users can customize notification settings, choosing which types of notifications to receive. 2. **Security** ◦ Notifications contain no sensitive personal or financial information. ◦ Secure transmission protocols are used for sending notifications to ensure data privacy. 3. **Validation** ◦ The system validates user preferences for notification types and frequency. ◦ Notifications are validated for correctness and relevance before being sent to the user. 4. **Error Handling** ◦ In case of a failure in sending notifications, the system retries automatically. ◦ Users are informed of any issues in notification delivery through an in-app message. 5. **Output/Notification** ◦ Notifications are clearly distinguishable and visible on the user interface. ◦ The system provides visual and/or auditory cues for incoming notifications based on user settings. 6. **Logging** ◦ All sent notifications are logged with timestamps and user identifiers. ◦ Failed notification attempts are also logged for system auditing and troubleshooting purposes.	
View User Information	As a user, I want to be able to retrieve my personal and banking information from the user portal of the banking application so that I can view and verify my details for accuracy and up-to-date information.	1. **Functionality** ◦ The application should allow users to retrieve their full profile information, including name, address, contact details, and account summary. ◦ Users should be able to access their recent transaction history. 2. **Security** ◦ Ensure user authentication before allowing access to personal information. ◦ Implement secure data transmission protocols (e.g., HTTPS). 3. **Validation** ◦ The system should validate user identity through multi-factor authentication before displaying personal information. ◦ Validate session tokens for each user session to prevent unauthorized data access.	

		4. **Error Handling** ◦ In case of a failure in data retrieval, display an appropriate error message. ◦ Provide a 'retry' option for users after an unsuccessful information retrieval attempt. 5. **Output/Notification** ◦ Upon successful retrieval of information, display the data clearly and concisely. ◦ Notify users of the last login date and time for security purposes. 6. **Logging** ◦ Log all user access and retrieval attempts, successful or unsuccessful, for audit and security purposes. ◦ Record any errors or exceptions encountered during the information retrieval process.	
Update Profile Information	As a user, I want to be able to update my profile information in the banking application's user portal so that my personal and contact details are always current and accurate.	1. **Functionality** ◦ The user must be able to access the profile update feature from their user portal. ◦ The system should allow editing of fields such as name, address, phone number, and email. ◦ Changes must be saved and updated immediately in the user's profile upon confirmation. 2. **Security** ◦ User authentication is required before accessing the profile update feature. ◦ The system must implement secure data transmission protocols (like HTTPS) to protect the data during the update process. ◦ Sensitive fields, like phone numbers and email, should be masked when displayed. 3. **Validation** ◦ The system must validate input formats (e.g., correct email format, valid phone number). ◦ Mandatory fields should not be left blank. ◦ The system should alert the user of invalid inputs and provide guidance for correction. 4. **Error Handling** ◦ In case of a system error or failure during the update process, the user should receive a clear error message. ◦ The system should ensure that partial updates do not occur; either all changes are saved, or none are in case of an error. 5. **Output/Notification** ◦ Upon successful update, the user should receive a confirmation message.	

		◦ If updates are not successful, the user should be notified with an appropriate error message. 6. **Logging** ◦ All profile update attempts (both successful and unsuccessful) should be logged with user identifiers, timestamp, and nature of the update. ◦ Security logs should capture any unauthorized access attempts to the profile update feature.	

Accounts - Customer Portal - Banking App - FS Java Capstone

Accounts

- Customer Portal: Transaction Management
- Customer Portal: Account Registration
- Customer Portal: User Loyalty Points (cashback, miles, points) History (Stretch)
- Customer Portal: Investing (Stretch)

Story	Description	Acceptance Criteria	Story Points
Customer Portal: Transaction Management			
View Transaction	As a user, I want to view all my banking transactions so that I can track and manage my spending, deposits, withdrawals, and transfers effectively. This function should provide a clear, comprehensive, and easily navigable display of all transactions across my various bank accounts. It should also include filters and sorting features to help me organize and review my transactions based on criteria like date, amount, transaction type, etc.	1. **Functionality** ◦ The system should display a list of all past transactions associated with the user's account. ◦ Transactions should include details such as date, amount, transaction type (debit/credit), and recipient/sender information. ◦ The user should have the option to filter transactions by date range and type. 2. **Security** ◦ Ensure that all transaction data is displayed over a secure connection. ◦ Implement role-based access control; only the account owner and authorized users can view transaction details. ◦ Sensitive data like account numbers should be partially masked. 3. **Validation** ◦ The system should validate the date range inputs for filtering transactions and prompt the user if the input is invalid. ◦ Transaction data should be validated to ensure completeness and accuracy before display. 4. **Error Handling** ◦ If transaction data fails to load, display a user-friendly error message. ◦ Provide a 'Retry' option if the initial attempt to fetch transaction data fails.	

		◦ In case of an invalid request (e.g., an unsupported filter), inform the user with a clear message. 5. **Output/Notification** ◦ Upon successful loading, the transaction data should be displayed in an easy-to-read format. ◦ If there are no transactions for the selected filters, display a message indicating 'No transactions found'. 6. **Logging** ◦ Log all user actions related to viewing transactions for auditing purposes. ◦ Any errors or exceptions should be logged with sufficient detail for troubleshooting.	
Create Transaction	As a user, I want to be able to create transactions (like money transfers, bill payments, and deposits) within my account so that I can manage my banking needs efficiently from anywhere and at any time. This feature is vital to meet the basic functionality of a modern banking application. It should allow transactions between my own accounts, as well as transactions to other bank users or institutions.	1. **Functionality** ◦ The User Portal must allow the creation of different types of transactions including transfers, bill payments, and deposits. ◦ Users should be able to select the account from which the transaction is made. ◦ The portal should provide a field for entering the transaction amount. ◦ Options for adding transaction notes or descriptions should be available. ◦ There should be a confirmation step before finalizing the transaction. 2. **Security** ◦ All transaction data must be transmitted over secure, encrypted connections. ◦ User authentication (e.g., password or biometric verification) is required before initiating a transaction. ◦ Implement timeout for session inactivity to prevent unauthorized access. 3. **Validation** ◦ Check for the validity of the account details entered. ◦ Ensure the transaction amount does not exceed the account balance. ◦ Validate transaction input fields for data type and format (e.g., numeric values for amount). 4. **Error Handling** ◦ Display user-friendly error messages for failed transactions due to network issues, insufficient funds, or invalid account details. ◦ Offer retry options in case of transaction failures. 5. **Output/Notification**	

		<ul><li>On successful transaction completion, show a confirmation message with transaction details.</li><li>Send a notification (email or SMS) to the user confirming the transaction.</li></ul>6. **Logging**<ul><li>Log all transaction activities including successful and failed attempts, with timestamps.</li><li>Ensure logs capture key details like transaction type, amount, and account involved, while maintaining user privacy.</li></ul>	
Sort Transactions	As a banking application user, I would like to be able to sort my transactions so that I can manage and view them in a sequence that best fits my needs.	1. **Functionality**<ul><li>The system should allow users to sort transactions by date, amount, and transaction type.</li><li>Each sorting option should organize transactions in both ascending and descending order.</li><li>The sorting feature should be easily accessible from the transaction management page.</li></ul>2. **Security**<ul><li>Sorting requests should be handled in the frontend without exposing sensitive transaction details to backend services unnecessarily.</li><li>The feature should comply with relevant data protection regulations for handling financial information.</li></ul>3. **Validation**<ul><li>The system must validate the user's selection for sorting (e.g., valid date range, transaction types).</li><li>Invalid sorting requests should prompt the user with an appropriate message to correct their input.</li></ul>4. **Error Handling**<ul><li>In case of a failure to sort (e.g., service unavailability), the system should display a user-friendly error message.</li><li>The system should handle unexpected sorting errors gracefully without crashing or freezing the user interface.</li></ul>5. **Output/Notification**<ul><li>Upon successful sorting, the user interface should update to display transactions in the selected order without a full page reload.</li><li>If no transactions meet the sorting criteria, a message should inform the user that there are no transactions to display.</li></ul>6. **Logging**	

		◦ All user actions related to sorting transactions should be logged for auditing purposes. ◦ Any errors or exceptions encountered during the sorting process should also be logged with sufficient detail for debugging.	
Filter Transactions	As a user, I want to be able to filter my transactions so that I can easily view and manage my account transactions based on certain criteria. This includes filtering by transaction amount, transaction type (debit/credit), transaction date, and transaction category (such as groceries, utilities, salary, etc.).	1. **Functionality** ◦ The system shall provide a filter option in the User Portal under the Transaction Management section. ◦ Users shall be able to filter transactions by date range, amount range, and transaction type (e.g., deposit, withdrawal, transfer). ◦ The filtered results shall be displayed immediately after the user applies the filter criteria. 2. **Security** ◦ All transaction data displayed must adhere to the user's permission level and privacy settings. ◦ The filter feature shall not expose any transaction details of other users. ◦ Secure coding practices must be followed to prevent injection attacks or data leaks. 3. **Validation** ◦ Date fields shall validate for correct format (e.g., MM/DD/YYYY) and logical dates (e.g., start date cannot be after end date). ◦ Amount fields shall only accept numerical values and shall validate for logical range (e.g., start amount cannot be greater than end amount). 4. **Error Handling** ◦ If no transactions match the filter criteria, display a message: "No transactions found matching your criteria." ◦ Invalid inputs in any filter fields shall prompt an error message detailing the correct format or expected value range. 5. **Output/Notification** ◦ Upon successful application of filters, the system shall display the number of transactions found. ◦ If the filter operation fails due to a system error, display a notification: "Unable to filter transactions at this moment. Please try again later." 6. **Logging** ◦ Each filter operation shall be logged with details including user ID, timestamp, and filter criteria used.	

		<ul><li>Any errors encountered during the filtering process shall be logged for further investigation by the technical team.</li></ul>	
Search Transaction	As a user, I want to have the ability to search my transactions, so that I can easily find specific transaction details based on various criteria such as transaction date, amount, type, recipient, and more. This feature should support quick navigation and filtering capabilities for efficient management and review of my banking transactions.	1. **Functionality**<ul><li>The search function should allow users to filter transactions by date, amount, transaction type (e.g., deposit, withdrawal), and transaction ID.</li><li>Users should be able to sort the search results by date, amount, and transaction type.</li><li>The system should display a summary of the transaction, including date, amount, transaction type, and a brief description etc.</li></ul>2. **Security**<ul><li>All transaction searches and results must be conducted over a secure, encrypted connection.</li><li>User authentication is required before accessing the search functionality.</li><li>Access to transaction data should be restricted based on user roles and permissions.</li></ul>3. **Validation**<ul><li>Input fields for dates and amounts should validate for correct format (e.g., mm/dd/yyyy for dates, numerical values for amounts).</li><li>The system should prompt the user with an error message if the search criteria are invalid or if no matching transactions are found.</li></ul>4. **Error Handling**<ul><li>In case of a system error or failure during the search process, the user should receive a clear and informative error message.</li><li>The system should gracefully handle network or server issues, ensuring the user can retry the search once the issue is resolved.</li></ul>5. **Output/Notification**<ul><li>Upon successful search, the transaction results should be displayed in a clear, readable format.</li><li>If no transactions match the search criteria, the system should display a message indicating "No transactions found."</li></ul>6. **Logging**<ul><li>All user search activities should be logged for audit purposes, including date, time, user ID, and search criteria used.</li><li>Error logs should be maintained, capturing details of any issues encountered during the transaction search process.</li></ul>	

Customer Portal: Account Registration			
View Account Types	As a User , I want to be able to view my account details so that I can check my registration details and verify their accuracy.	1. **Functionality** ◦ The system displays a list of all available bank account types. ◦ Each account type includes a brief description and key features. ◦ The user can view more details for each account type upon selection. 2. **Security** ◦ The display of account types does not require user login; no sensitive data is shown. ◦ All data transmissions are encrypted using industry-standard protocols. 3. **Validation** ◦ The system checks the availability of account type data from the back-end before displaying. ◦ If data is not available or outdated, the system refrains from displaying incorrect or incomplete information. 4. **Error Handling** ◦ In case of a failure to retrieve account type data, a friendly error message is displayed. ◦ The system offers options to retry or contact support in case of persistent errors. 5. **Output/Notification** ◦ Upon successful display, the user receives visual confirmation through the orderly presentation of the account types. ◦ Any updates or changes in the account types are dynamically updated without needing a page refresh. 6. **Logging** ◦ All user interactions with the account types view feature are logged for future analysis and improvement of user experience. ◦ Error occurrences are logged with details for troubleshooting and system improvement.	
Register for Account	As a User, I want to be able to register for an account so that I can utilize the bank's online services and manage my finances digitally.	1. **Functionality** ◦ User should be able to register for a bank account ◦ Users are able to select a specific bank account type ◦ User should be able to enter any addition information that is required to register for the account 2. **Security** ◦ All user data is transmitted over a secure, encrypted connection (HTTPS). ◦ Sensitive information should be encrypted	

		3. **Validation** ∘ The system validates all input fields for format and completeness. ∘ Email addresses are verified for proper formatting. ∘ Social security numbers are validated against standard formats. ∘ The system checks for duplicate usernames or email addresses. 4. **Error Handling** ∘ Users are informed of any input errors (e.g., invalid email format, incomplete fields). ∘ Users receive clear instructions on how to correct the errors. ∘ The system prevents submission until all errors are resolved. 5. **Output/Notification** ∘ Upon successful bank account registration, users receive a confirmation message on the screen. ∘ Upon unsuccessful bank account registration, users receive a confirmation message on the screen that e.g. “Account Registration unsuccessful”. 6. **Logging** ∘ All registration attempts (successful or unsuccessful) are logged with timestamps. ∘ System logs capture user input errors and the nature of the errors. ∘ Security-related events (e.g., multiple failed attempts) are flagged in the system logs for review.	
Account Registration Confirmation	As a User, I want to receive confirmation after successfully registering an account so that I can be sure that my registration was successful and I can start using the banking services.	1. **Functionality** ∘ The system shall send a confirmation message to the user’s registered email upon successful account registration if approved or unapproved. 2. **Security** ∘ All communication during the confirmation process shall be encrypted. ∘ The system shall implement measures to prevent brute force attacks on the confirmation process. 3. **Error Handling** ∘ The system shall provide clear error messages for invalid or expired confirmation links or codes. ∘ In case of system errors during the confirmation process, the system shall prompt the user to try again later.	

		4. **Output/Notification** ◦ Upon successful confirmation, the user shall receive a notification of successful account activation. ◦ If the confirmation fails, the user shall receive an appropriate notification explaining the reason. 5. **Logging** ◦ All user actions during the confirmation process shall be logged for audit purposes. ◦ System errors and failed confirmation attempts shall also be logged for further analysis.	
Customer Portal: User Loyalty Points (cashback, miles, points) History (Stretch)			
Account Points History	As a user, I want to be able to view the history of my loyalty points (cashback, miles, points) in the banking application, so that I can track my earnings and redemptions over time. This feature should provide a detailed history including dates, points earned or redeemed, and the corresponding transactions.	1. **Functionality** ◦ The system shall display the history of loyalty points earned and redeemed. ◦ The history shall include details such as date of transaction, number of points earned or redeemed, and a brief description of the transaction. ◦ The user shall be able to filter the history by date range. 2. **Security** ◦ Access to the loyalty points history shall be restricted to authenticated users only. ◦ The system shall ensure that users can only access their own points history. ◦ Secure communication protocols shall be used for data transmission. 3. **Validation** ◦ The system shall validate date ranges entered by the user for filtering history. ◦ Any invalid date range inputs shall prompt an appropriate error message. 4. **Error Handling** ◦ In case of failure to retrieve points history due to server or connectivity issues, an error message shall be displayed. ◦ The error message shall be clear and non-technical, guiding the user to try again later. 5. **Output/Notification** ◦ Upon successful retrieval, the points history shall be displayed in a clear, readable format. ◦ If no points history is found for the selected date range, a message shall indicate that no records are available. 6. **Logging** ◦ All user interactions with the loyalty points history feature shall be logged.	

		◦ Any errors encountered during the usage of this feature shall also be logged for further analysis.	
Customer Portal: Investing (Stretch)			
View Stocks	As a user, I want to view my stock portfolio within the banking application's user portal, so I can easily monitor my investment performance, check stock prices, and view key details about my investments in real-time.	1. **Functionality** ◦ The system displays a list of stocks currently owned by the user. ◦ Users can view real-time prices and performance metrics for each stock. ◦ The system provides detailed views for individual stocks, including historical performance charts. 2. **Security** ◦ Stock viewing is protected by user authentication; only the logged-in user can view their stock information. ◦ The system employs HTTPS for data transmission to ensure data privacy and integrity. ◦ Sensitive data, like stock quantities and values, are encrypted at rest. 3. **Validation** ◦ The system validates stock symbols and user ownership before displaying information. ◦ Invalid or unrecognized stock symbols result in a notification to the user without exposing system details. 4. **Error Handling** ◦ If stock data fails to load, the system shows a user-friendly error message. ◦ The system handles partial failures gracefully, showing available data with notifications about unavailable information. ◦ Errors due to network issues prompt the user to try again later. 5. **Output/Notification** ◦ Users receive a confirmation message when the stock data is successfully loaded. ◦ Any changes in the stock's status (like significant price changes) trigger notifications, if opted in by the user. 6. **Logging** ◦ The system logs all user interactions with the stock viewing feature for audit purposes. ◦ Errors and anomalies in stock data retrieval are logged for system monitoring and troubleshooting.	
Purchase Stocks	As a user, I want to be able to purchase stocks through the User Portal in the Banking	1. **Functionality**	

	Application, so that I can invest in various companies easily and manage my investments within the same platform.	◦ The feature allows users to search for stocks by company name or ticker symbol. ◦ Users can view current stock prices and relevant information before purchasing. ◦ There is a functionality for users to specify the number of shares they wish to purchase. ◦ The system calculates the total cost of the transaction based on current stock prices and the number of shares. ◦ Users can confirm or cancel the purchase before final submission. 2. **Security** ◦ All stock purchase transactions require two-factor authentication. ◦ Sensitive user data, including transaction details, are encrypted. ◦ Session timeouts are implemented to prevent unauthorized access during inactivity. 3. **Validation** ◦ Input fields for stock search, number of shares, and other relevant data have validation checks for correct format and data types. ◦ The system validates the availability of stocks and sufficient funds in the user's account before allowing the transaction to proceed. 4. **Error Handling** ◦ Clear error messages are displayed for invalid inputs or failed transactions. ◦ The system handles cases of stock unavailability or insufficient funds by prompting appropriate user actions. ◦ Error logs are maintained for failed transactions or system errors for troubleshooting. 5. **Output/Notification** ◦ Upon successful purchase, a confirmation message with transaction details is displayed. ◦ Users receive email notifications summarizing the transaction details. ◦ The user's portfolio is updated in real-time to reflect the new stock purchase. 6. **Logging** ◦ All stock purchase transactions are logged with e.g. user ID, timestamp, stock details, and transaction amount. ◦ Logs are maintained for auditing and tracking purposes, and are accessible only to authorized personnel.	
Sell Stocks	As a user, I want to be able to sell stocks through the User Portal of my Banking Application, so that	1. **Functionality**	

	I can efficiently manage my investments within the User Portal - Investing section.	◦ The system provides an option for the user to select 'Sell Stocks' under the 'User Portal - Investing' section. ◦ Allows the user to search and select the stock they wish to sell from their investment portfolio. ◦ Provides an interface to enter the number of stocks to be sold and displays the current market price per stock. ◦ Shows a preview of the transaction, including the total sale value based on the current market price and the number of stocks selected. ◦ Includes a 'Confirm Sale' button to complete the transaction. 2. **Security** ◦ Ensures all stock sale transactions require user authentication (e.g., password or biometric verification) before processing. ◦ Implements SSL/TLS encryption for the transmission of user data and transaction details. ◦ Conducts automatic logout after a period of inactivity during the stock sale process. 3. **Validation** ◦ Checks that the number of stocks the user wishes to sell does not exceed the number available in their portfolio. ◦ Validates that the entered stock symbols correspond to valid stocks in the user's portfolio. ◦ Ensures that the stock sale value is recalculated in real-time if the market price changes during the transaction process. 4. **Error Handling** ◦ Displays an error message if there is a failure in fetching the current market price of the stock. ◦ Provides a clear error notification if the user attempts to sell more stocks than available in their portfolio. ◦ Alerts the user in case of transaction failures due to network issues or backend problems. 5. **Output/Notification** ◦ Confirms successful stock sale transactions with a detailed summary, including the number of stocks sold, sale value, and transaction date. ◦ Sends a notification to the user’s registered email or phone number upon the successful completion of the stock sale.	

		◦ Updates the user's investment portfolio in real-time to reflect the sold stocks. 6. **Logging** ◦ Logs all user actions and data inputs during the stock sale process for audit purposes. ◦ Records successful and unsuccessful transaction attempts, including timestamps and user identifiers. ◦ Maintains logs of system errors and exceptions encountered during stock sale transactions.		

Cards & Loans - Customer Portal - Banking App - FS Java Capstone

Cards & Loans

Story	Description	Acceptance Criteria	Story Points
Customer Portal: Sign up for Card			
Sign Up for Card	As a potential cardholder, I want to sign up for a new card through a user-friendly and secure online application process. When I visit the card issuer's website, I should be able to easily find the "Sign Up for Card" button or link. Clicking on this button should take me to a sign-up form where I can enter my personal information, such as my name, address, phone number, and email address.	1. **Functionality** ◦ The user can access the 'Sign Up for Card' feature from the user portal. ◦ The user is presented with options for different types of cards (e.g., credit, debit, prepaid). ◦ The application allows the user to select a card type and fill in the required details (e.g., personal information, income level). ◦ Upon submission, the application sends the card request for processing. 2. **Security** ◦ All user data input is transmitted securely using encryption. ◦ User authentication is required before accessing the sign-up feature. ◦ The application should log out automatically after a period of inactivity. 3. **Validation** ◦ The application checks for the completeness and format of all the input fields. ◦ Invalid inputs (e.g., incorrect email format, missing mandatory fields) are flagged with clear error messages. ◦ The application verifies the eligibility of the user based on the provided information.	

		4. **Error Handling** ◦ The application provides informative error messages for network issues or server unavailability. ◦ The application displays a friendly message in case of system errors that prevent the completion of the sign-up process. ◦ Users are given the option to retry submission or contact support in case of persistent issues. 5. **Output/Notification** ◦ The user receives a confirmation message upon successful submission. ◦ The user is informed about the estimated time for processing the application. ◦ Notifications are sent to the user about the status of the card application (e.g., approved, pending, declined). 6. **Logging** ◦ All user interactions with the 'Sign Up for Card' feature are logged for audit purposes. ◦ Errors and exceptions during the sign-up process are logged with relevant details. ◦ System logs capture timestamps and user identifiers for each step of the sign-up process.	
View Cards on Offer	As a user, I want to be able to view various card offers available so that I can choose one that best suits my financial needs and preferences.	1. **Functionality** ◦ The system displays a variety of credit and debit card offers. ◦ Information about each card includes details such as benefits, fees, interest rates, and eligibility criteria. ◦ Users can sort and filter card offers based on type, benefits, and other relevant parameters. 2. **Security** ◦ All card offer data is presented without exposing sensitive banking or personal information. ◦ The system implements secure communication protocols (e.g., HTTPS) to protect data in transit. ◦ User authentication is required before accessing the card offers section. 3. **Validation** ◦ The system validates user inputs (e.g., filters and sorting preferences) to ensure they are within acceptable parameters. ◦ Displays an appropriate message if no cards match the user's filtering criteria. 4. **Error Handling** ◦ In case of a failure to load card offers (e.g., server issues), the system shows a user-	

		friendly error message. ◦ Provides a mechanism to retry fetching the card offers (e.g., a "Try Again" button). 5. **Output/Notification** ◦ Confirmation or informational messages are displayed upon successful loading of card offers. ◦ Visual cues (e.g., loading indicators) are provided while card offers are being fetched. 6. **Logging** ◦ User interactions with the card offers section (e.g., sort/filter selections) are logged for analytical purposes. ◦ Any errors encountered during the process are logged for troubleshooting and system improvement.	
Confirm Card Registration	As a user, I want to be able to confirm that my payment card has been successfully registered with the system. When I enter my payment card details and submit them, the system will validate the card information and save it in the system. Once my payment card is registered, I will receive a confirmation message on the screen that my payment card has been successfully registered.	1. **Functionality** ◦ The system should display a 'Confirm Registration' button after the user completes the card registration form. ◦ Upon clicking 'Confirm Registration', the system should verify the provided card details against the database. ◦ If the card details are correct and registration is successful, the user should be redirected to a confirmation page. 2. **Security** ◦ All card details entered by the user must be encrypted during transmission. ◦ The system should implement secure session management, ensuring that the confirmation process is tied to the user's current authenticated session. ◦ There should be security measures to prevent CSRF (Cross-Site Request Forgery) attacks during the confirmation process. 3. **Validation** ◦ The system must validate the format of card details (e.g., card number, expiration date) before submission. ◦ Appropriate error messages should be displayed for invalid inputs. ◦ The system should check for the existence of the card in the bank’s database. 4. **Error Handling** ◦ In case of a system error or failure during confirmation, the user should receive a clear error message. ◦ The system should handle server-side errors gracefully and not expose sensitive system	

		information to the user. 5. **Output/Notification** ◦ Upon successful confirmation, the user should receive a notification (e.g., email, SMS) confirming that the card has been registered. ◦ The confirmation page should clearly state that the card registration is complete and provide next steps or actions for the user. 6. **Logging** ◦ All user actions during the card confirmation process should be logged for audit and troubleshooting purposes. ◦ Logs should capture details like user ID, timestamp, and nature of the action without storing sensitive card details.	
Customer Portal: Sign Up for Loans			
View Loans on Offer	As a user, I want to view the various loan options available on the banking application's user portal so that I can understand the different types of loans, their terms, and eligibility requirements.	1. **Functionality** ◦ The user portal should display a list of available loan options. ◦ Each loan option should include details such as interest rate, loan amount range, repayment terms, and eligibility criteria. ◦ Users should have the ability to click on each loan for more detailed information. 2. **Security** ◦ Ensure that all loan information is transmitted over a secure, encrypted connection. ◦ Implement measures to prevent unauthorized access to loan details. 3. **Validation** ◦ Validate user input fields such as filters for loan type, amount, or duration to ensure data entered is within acceptable parameters. ◦ Display appropriate messages if the user enters invalid data. 4. **Error Handling** ◦ If the loan information fails to load, display a user-friendly error message. ◦ Provide a mechanism to retry fetching the loan information. 5. **Output/Notification** ◦ On successful loading of loan options, ensure the information is displayed in a clear, understandable format. ◦ Notify the user with a message if no loan options are available or if they don't meet the criteria for viewing certain loans. 6. **Logging**	

		◦ Log user interactions with the loan options, such as clicks and time spent on each loan detail. ◦ Record any errors or issues encountered during the loan information retrieval process.	
Sign Up for Loan	As a user, I want to be able to sign up for a loan through the banking application's user portal so that I can easily apply for financial assistance without visiting a bank branch.	1. **Functionality** ◦ The loan application form should be accessible after user authentication. ◦ Users should be able to select different types of loans (e.g., personal, home, auto) from a dropdown menu. ◦ The application form should include fields for amount, term, and purpose of the loan. ◦ A 'Submit' button should be available to send the application for processing. 2. **Security** ◦ Ensure that all data transmission is encrypted using SSL/TLS protocols. ◦ Implement CAPTCHA verification to prevent automated submissions. ◦ Session timeouts should be implemented to protect user data if left idle. 3. **Validation** ◦ Input fields should be validated for correct data format (e.g., numeric values for amount, date format for term). ◦ Mandatory fields (e.g., loan amount, loan term) must be completed before submission. ◦ Real-time validation feedback should be provided (e.g., red outline for incorrect format). 4. **Error Handling** ◦ Display user-friendly error messages for input validation failures (e.g., "Invalid amount entered"). ◦ In case of network issues or server unavailability, show an appropriate error message (e.g., "Service temporarily unavailable, please try again later"). ◦ Include a 'Retry' option for network-related errors. 5. **Output/Notification** ◦ Upon successful submission, display a confirmation message (e.g., "Your loan application has been submitted successfully"). ◦ Instruct users to expect a follow-up communication for further steps or application status. ◦ Provide a clear indication if the submission is still in progress (e.g., loading spinner).	

		6. **Logging** ◦ Log user actions related to the loan application process for auditing purposes. ◦ Errors and unsuccessful submission attempts should be logged with timestamps and user identifiers. ◦ Ensure that logs are stored securely and accessible only to authorized personnel.	
Confirm Loan Registration	As a user , I want to be able to confirm my loan registration so that I can be sure that my loan application has been successfully received and processed.	1. **Functionality** ◦ The confirmation process should clearly display the loan's terms and conditions. ◦ A visible and interactive "Confirm" button should be provided to submit the loan registration. ◦ Upon confirmation, the user should be directed to a confirmation page or dashboard indicating successful registration. 2. **Security** ◦ The confirmation process should implement HTTPS for secure data transmission. ◦ Sensitive data, like personal and financial information, must be encrypted. ◦ Implement CAPTCHA or similar mechanisms to prevent automated submissions. 3. **Validation** ◦ Ensure input fields for loan details are validated for correct format and completeness. ◦ Display clear error messages if the user enters invalid data. ◦ Confirmation button should be disabled until all required fields are correctly filled. 4. **Error Handling** ◦ Provide clear error messages for any failed submission or system errors. ◦ Implement a retry mechanism or direct user to try again later in case of system failure. ◦ Errors should be logged for further analysis but not expose sensitive information to the user. 5. **Output/Notification** ◦ Display a confirmation message or screen upon successful loan registration. ◦ Send an email or SMS notification to the user confirming the loan registration. ◦ Clearly indicate any next steps or actions required by the user. 6. **Logging** ◦ Log all user actions during the confirmation process for audit purposes.	

		◦ Errors and unsuccessful attempts should be logged with timestamp and user details. ◦ Ensure logs are secure and only accessible by authorized personnel.	
Customer Portal: Card Management			
Make Card Payments	As a user, I want to be able to make card payments for my purchases on the website. I want to be able to securely enter my card details including the card number, expiry date, CVV code, and cardholder name. The website should also provide clear instructions and feedback throughout the payment process to ensure that I understand each step and can easily complete the transaction.	1. **Functionality** ◦ The system allows users to select a credit card to make a payment. ◦ Users can enter the amount they wish to pay. ◦ The system provides an option to choose the payment account. ◦ Users can review and confirm payment details before submission. 2. **Security** ◦ The payment feature employs HTTPS for secure data transmission. ◦ User authentication is required before accessing the payment feature. ◦ The system implements anti-CSRF tokens to prevent cross-site request forgery. 3. **Validation** ◦ The system validates the entered payment amount for format (numerical) and logical errors (e.g., negative values, exceeding due amount). ◦ Payment account selection is validated for existence and fund sufficiency. 4. **Error Handling** ◦ If a payment fails, the system shows a clear error message specifying the reason (e.g., “Insufficient funds”, “Invalid amount”). ◦ In case of a system error, the user is informed without exposing any technical details. 5. **Output/Notification** ◦ Upon successful payment, the user receives a confirmation message with payment details. ◦ The system sends an email receipt to the user’s registered email address. 6. **Logging** ◦ All payment transactions are logged with necessary details (e.g., user ID, transaction amount, date/time) for auditing purposes. ◦ Failed payment attempts are logged with error codes and timestamps.	
View Card Transactions	As a cardholder, I want to view my recent card transactions so that I can keep track of my spending and detect any unauthorized transactions.	1. **Functionality** ◦ The system shall display a list of recent transactions made with the user’s cards. ◦ Transactions shall be retrievable based on specific cards linked to the user’s account.	

		2. **Security** ◦ Access to view card transactions shall require user authentication. ◦ Sensitive data, such as card numbers, shall be partially masked (e.g., last four digits visible). ◦ The system shall implement secure communication protocols to protect transaction data during transmission. 3. **Validation** ◦ The system shall validate user permissions to ensure they can only view transactions for cards they own. ◦ Date range filters applied by the user shall be validated for format and logical consistency (e.g., start date cannot be after end date). 4. **Error Handling** ◦ In case of a system error or failed transaction retrieval, the system shall display a user-friendly error message. ◦ The system shall handle network or connectivity issues gracefully, informing the user to try again later. 5. **Output/Notification** ◦ The system shall display transaction details in a clear, readable format. ◦ Notifications shall be provided for any updates or changes in the transaction status (if applicable). 6. **Logging** ◦ All user actions related to viewing card transactions shall be logged for audit purposes. ◦ System errors and failed transaction retrieval attempts shall also be logged for further analysis.	
View Card Status	As a user, I want to view the status of my cards within the banking application, so that I can easily manage and monitor my card activities and status without needing to contact customer service or visit a bank branch.	1. **Functionality** ◦ The user must be able to view the current status of all their cards (credit, debit, etc.) linked to their account. ◦ The status information should include card type, card number (last four digits), issue date, expiry date, and current status (active, blocked, expired). 2. **Security** ◦ All card status information should be displayed after successful user authentication. ◦ The application should enforce HTTPS to protect the data in transit. ◦ Sensitive card details (like full card number or CVV) should not be displayed. 3. **Validation**	

		◦ The system should validate user permissions to ensure only the account holder or authorized users can view card status. ◦ In case of multiple users (like in joint accounts), proper validation should confirm that the user has rights to view the specific card's status. 4. **Error Handling** ◦ If the card status fails to load, the system should display a user-friendly error message. ◦ The system should handle timeouts or network issues gracefully, prompting the user to try again later. 5. **Output/Notification** ◦ Upon successful loading, the card status should be displayed in a clear, readable format. ◦ Any changes in the card status (like blocking or unblocking a card) should trigger a real-time update on the user interface. 6. **Logging** ◦ All user actions related to viewing card status should be logged for audit and security purposes. ◦ Errors or system malfunctions during the view operation should also be logged for troubleshooting and improvement of the system.	
Request new Card	As a user, I want to be able to request a new card through the User Portal, so that I can easily manage my banking needs online without visiting a branch or calling customer service.	1. **Functionality** ◦ The portal allows users to initiate a request for a new card. ◦ Users can select the type of card (e.g., debit, credit) they wish to request. ◦ Users can review and confirm their card request before submission. 2. **Security** ◦ All card request processes are conducted over a secure, encrypted connection. ◦ Users are required to authenticate (e.g., password, biometric verification) before accessing the card request feature. ◦ Sensitive data (e.g., personal information, card preferences) are handled in compliance with data protection regulations. 3. **Validation** ◦ The system validates user input for correctness and completeness. ◦ Error messages are displayed if the user enters invalid or incomplete information. ◦ The system checks for existing card limits (e.g., max number of cards allowed) before allowing a new request.	

		4. **Error Handling** ◦ If the card request process encounters a technical issue, the user is informed with a clear message. ◦ Users are provided with suggestions or actions to take in case of a failed request (e.g., contact support, try again later). ◦ The system gracefully handles server-side errors without crashing or freezing the user interface. 5. **Output/Notification** ◦ Upon successful submission of a card request, users receive a confirmation message with relevant details (e.g., expected processing time). ◦ Notifications or updates regarding the card request status are sent to the user (e.g., via email, in-app notification). 6. **Logging** ◦ All user actions related to the card request are logged for audit and troubleshooting purposes. ◦ System errors or anomalies during the card request process are captured in the error logs. ◦ Privacy and security standards are maintained in the logging process, ensuring no sensitive user data is stored in plain text.	
Deactivate Card	As a user, I want the ability to stop my card through the User Portal, ensuring that I can quickly deactivate my card in case it is lost, stolen, or compromised. This feature should allow me to easily and securely suspend the card, preventing any unauthorized transactions, while providing clear feedback and maintaining a record of this action for future reference.	1. **Functionality** ◦ The 'Stop Card' feature is accessible from the Card Management section in the User Portal. ◦ Users can select the specific card they wish to stop. ◦ Upon selection, users are required to confirm the action to stop the card. ◦ Once confirmed, the card is immediately deactivated for transactions. 2. **Security** ◦ The feature includes user authentication checks before allowing access. ◦ The session is secured, ensuring data encryption during the process. ◦ The application logs out the user after a period of inactivity during the stop card process. 3. **Validation** ◦ The system validates the card's existence and status before proceeding with the stop action. ◦ Users receive a message if they attempt to stop an already deactivated or invalid card. 4. **Error Handling** ◦ In case of a system error or failure, the user is informed with a clear, understandable error	

		message. ◦ The application provides an option to retry the action or contact support if the error persists. 5. **Output/Notification** ◦ Upon successful card deactivation, users receive a confirmation notification. ◦ The confirmation includes details of the card stopped and the time of the action. 6. **Logging** ◦ All actions related to stopping a card are logged with user ID, timestamp, and card details. ◦ Failed attempts are also logged for audit and security purposes.	
Report Card as Stolen	As a user, I want to be able to report my card as stolen through the banking application's user portal so that I can ensure my financial security is not compromised.	1. **Functionality** ◦ The user can select the card to report as stolen from a list of their active cards. ◦ The application should provide an option to report the card as stolen. ◦ Upon selection, the user should be able to confirm the action. 2. **Security** ◦ The feature must require user authentication before allowing access to the card reporting option. ◦ All data transmissions related to this feature should be encrypted. ◦ The application should log out the user after a period of inactivity while using this feature. 3. **Validation** ◦ The system should validate that the selected card belongs to the authenticated user. ◦ The application must confirm the user's action to report the card as stolen before processing. 4. **Error Handling** ◦ If the card cannot be reported as stolen (e.g., due to a server error), the application should display an appropriate error message. ◦ The application should provide guidance or contact information for further assistance if the process cannot be completed. 5. **Output/Notification** ◦ After successful reporting, the user should receive a confirmation message on the application interface. ◦ The user should receive an email or SMS notification confirming the card has been reported as stolen. 6. **Logging**	

		<ul><li>The system should log all actions taken by the user during this process, including timestamp, user ID, and the card number reported.</li><li>In case of errors, the system should log the error details for troubleshooting purposes.</li></ul>	
Use cashback for Card Payments	As a user, I want to use my accumulated cashback to make payments on my card so that I can effectively utilize the rewards I have earned through my card usage.	1. **Functionality**<ul><li>The system should allow users to view their available cashback balance.</li><li>Users should be able to apply their cashback towards their card payments.</li><li>The amount of cashback applied should be deducted from the total payment amount.</li></ul>2. **Security**<ul><li>Ensure all cashback transactions are conducted over a secure connection.</li><li>Implement user authentication checks before allowing access to the cashback redemption feature.</li><li>Encrypt sensitive data related to cashback transactions.</li></ul>3. **Validation**<ul><li>Validate the availability of sufficient cashback balance before processing the payment.</li><li>Confirm that the cashback amount does not exceed the total payment due.</li><li>Verify that the card account is active and not flagged for any restrictions.</li></ul>4. **Error Handling**<ul><li>In case of insufficient cashback balance, display an appropriate error message.</li><li>Handle network or processing errors gracefully and inform the user.</li><li>Ensure errors related to card status (e.g., blocked or inactive card) are clearly communicated.</li></ul>5. **Output/Notification**<ul><li>Provide a confirmation message upon successful application of cashback to a payment.</li><li>Display the updated cashback balance and remaining payment amount after the transaction.</li><li>Notify the user of any unsuccessful transaction attempts due to errors.</li></ul>6. **Logging**<ul><li>Log all user interactions with the cashback feature for audit and troubleshooting purposes.</li><li>Record details of each cashback transaction, including date, time, amount, and card number.</li></ul>	

		◦ Maintain logs of any errors encountered during cashback transactions.	
Temporarily Lock Card	As a user, I want to be able to temporarily lock my bank card through the user portal, so that I can prevent unauthorized use while the card is not in my possession or if I suspect it is lost or stolen.	1. **Functionality** ◦ The user can locate the 'Temporarily Lock Card' feature easily within the Card Management section. ◦ On selection, the user is presented with a list of their active cards. ◦ The user can select the card they wish to lock. ◦ A confirmation step is required before the card is locked. ◦ Once confirmed, the card status is changed to 'Locked' immediately. 2. **Security** ◦ All user actions are performed over a secure connection. ◦ User authentication is required before accessing the Card Management section. ◦ Additional verification (e.g., password, security question, OTP) is required before locking the card. ◦ The feature is disabled after multiple failed attempts to prevent brute force attacks. 3. **Validation** ◦ The system validates the card's existence and active status before allowing the lock action. ◦ An appropriate message is displayed if the user attempts to lock a card that is already locked, expired, or does not exist. 4. **Error Handling** ◦ Clear error messages are displayed for network issues or server unavailability. ◦ User is informed if there is a delay in processing the lock request. ◦ An error message is displayed for any unauthorized attempt to access the feature. 5. **Output/Notification** ◦ The user receives a visual confirmation on the screen once the card is successfully locked. ◦ An email and/or SMS notification is sent to the user confirming the card lock. 6. **Logging** ◦ All actions (including successful locks, failed attempts, and errors) are logged for audit purposes. ◦ Logs contain timestamps, user identifiers, and action details without storing sensitive card information.	

Customer Portal: Loan Management			
View Loan Status	As a user, I want to be able to view the status of my loans in the banking application's user portal, so I can easily track the current status, outstanding balance, payment history, and any relevant details of my loans.	1. **Functionality** ◦ The user should be able to view all current and past loans. ◦ Each loan should display its current status (e.g., active, paid off, in arrears). ◦ Details such as loan amount, outstanding balance, interest rate, and payment history should be visible. ◦ The user interface should refresh to show the most recent data each time it's accessed. 2. **Security** ◦ Loan status and details should only be accessible after successful user authentication. ◦ Data transmission of loan information must be encrypted using industry-standard protocols. ◦ Implement timeout for inactivity to prevent unauthorized access. 3. **Validation** ◦ Ensure that the displayed loan data matches the user's account records. ◦ Validate that all the loan details are up-to-date and accurately reflect the backend database. 4. **Error Handling** ◦ If loan data fails to load, display a user-friendly error message. ◦ Provide an option to retry fetching the loan data if an error occurs. ◦ Errors should be logged for further investigation without exposing sensitive data to the user. 5. **Output/Notification** ◦ Upon successful loading, the loan status and details should be displayed in a clear, readable format. ◦ Notify the user of any changes to the loan status (e.g., payment due, payment received) through in-app notifications. 6. **Logging** ◦ Log user activities related to viewing loan status for security and auditing purposes. ◦ Any errors or anomalies in accessing loan data should be logged with sufficient detail for troubleshooting.	
Make Loan Payments	As a user, I want to be able to make loan payments through the User Portal so that I can easily manage my loan repayment schedule and amounts, ensuring that the process is secure, user-friendly, and error-free.	1. **Functionality** ◦ The portal allows the user to view their current loan balance. ◦ Users can select the loan account they wish to make a payment to.	

- There is an option to enter the payment amount.
- Users can choose the payment date, with the ability to schedule future payments.
- The system provides a confirmation screen before finalizing the payment.
- Upon confirmation, the payment is processed.

2. **Security**
   - All loan payment transactions require user authentication.
   - The system uses SSL encryption for data transmission.
   - Payment details are not stored on the user's device or browser.
   - Session timeout is implemented for inactivity.
3. **Validation**
   - The system checks for the validity of the entered payment amount (not exceeding the loan balance and not below minimum payment criteria).
   - Payment date validation ensures the chosen date is not a past date.
   - The system validates the source of payment (e.g., linked bank account) for sufficient funds.
4. **Error Handling**
   - The system displays user-friendly error messages for invalid inputs or failed transactions.
   - There are specific error messages for network issues or service unavailability.
   - Retry mechanisms are in place for temporary failures.
5. **Output/Notification**
   - Upon successful payment, the user receives a confirmation message with payment details.
   - The system sends a receipt via email or SMS as per user preference.
   - Real-time update of the loan balance after the payment is reflected in the user's account.
6. **Logging**
   - All payment transactions are logged with details including user ID, transaction ID, amount, and date.
   - Failed transactions are also logged with error details.
   - System maintains an audit trail for security and dispute resolution purposes.

Customer Portal: Cashback Program			
Earn cashback for transactions on Credit card	As a user, I want to earn cashback for transactions made with my credit card so that I can receive rewards for my spending.	1. **Functionality** ◦ The system should automatically calculate cashback based on the predefined percentage rate for each eligible credit card transaction. ◦ The cashback amount should be credited to the user's account within a specified timeframe after the transaction is completed. ◦ Users should be able to view the total cashback earned on their dashboard. 2. **Security** ◦ All credit card transactions and cashback calculations must be processed in a secure environment to protect user data. ◦ The system should have safeguards to prevent unauthorized access to cashback information and transaction details. 3. **Validation** ◦ The system should validate the eligibility of transactions for cashback. Not all transactions may qualify (e.g., balance transfers, cash advances). ◦ There should be a validation check to ensure that the cashback amount does not exceed any set maximum limits per transaction or account. 4. **Error Handling** ◦ In case of a transaction processing error, the user should be notified, and the transaction should not qualify for cashback. ◦ If there's an error in cashback calculation, the system should log the error and flag it for review by the technical team. 5. **Output/Notification** ◦ Users should receive a notification (e.g., email, in-app notification) once cashback is credited to their account. ◦ The user interface should display an updated total of cashback earned after each eligible transaction. 6. **Logging** ◦ All transactions and their corresponding cashback calculations should be logged for audit purposes. ◦ Any anomalies or errors in the cashback process should be logged with detailed information for troubleshooting.	
Customer Portal: Frequent Flyer Program			
Earn miles for transactions on	As a user, I want to earn miles for transactions made using my Frequent Flyer Card so that I can	1. **Functionality**	

Frequent Flyer Card	benefit from the Frequent Flyer Program associated with my bank's User Portal.	◦ The system should accurately track and record miles earned for each eligible transaction made using the Frequent Flyer Card. ◦ Transactions should be categorized to distinguish between those that qualify for miles and those that do not. 2. **Security** ◦ All transactions and mile accruals must be processed in a secure environment to protect user data. ◦ Implement standard encryption for data transmission. ◦ Ensure compliance with relevant financial data protection regulations. 3. **Validation** ◦ Validate transaction data to ensure only legitimate and successful transactions are considered for mile accrual. ◦ Implement checks to prevent duplication in mile accrual for the same transaction. 4. **Error Handling** ◦ In case of a transaction failure, ensure that the system does not award miles. ◦ Provide meaningful error messages to the frontend in case of failures in transaction processing or mile recording. 5. **Output/Notification** ◦ Once miles are successfully recorded, send a confirmation to the frontend for user notification. ◦ Ensure that the updated mile balance is accurately reflected in the user's account. 6. **Logging** ◦ Maintain detailed logs of all transactions and mile accrual activities for auditing and troubleshooting purposes. ◦ Log any discrepancies or errors encountered during the transaction or mile accrual process.	
Consume earned miles for new Tickets	As a user, I want to be able to consume my earned miles from the Frequent Flyer Program to purchase new tickets through the Cards & Loans feature within the User Portal of our Banking Application. This will allow me to efficiently use my reward points without the need for external processes.	1. **Functionality** ◦ The system should display the total earned miles available for the user. ◦ Allow users to select the number of miles they wish to redeem. ◦ Calculate the equivalent value of the selected miles in terms of ticket pricing. ◦ Provide an option to confirm the redemption of miles for tickets. ◦ Update the user's miles balance after successful redemption. 2. **Security**	

		◦ Implement secure session management to prevent unauthorized access. ◦ Employ encryption for the transmission of sensitive data. ◦ Require user authentication before accessing the Frequent Flyer Program features. 3. **Validation** ◦ Verify that the number of miles entered for redemption does not exceed the user's available balance. ◦ Ensure the input for miles is a valid number and not a negative value. ◦ Validate the availability of tickets against the miles to be redeemed. 4. **Error Handling** ◦ Display an error message if the redemption process fails due to network issues or server errors. ◦ Inform the user if the entered miles exceed their available balance. ◦ Provide a clear message if no tickets are available for the equivalent value of miles selected. 5. **Output/Notification** ◦ Show a confirmation message upon successful redemption of miles. ◦ Update the user interface to reflect the new balance of miles. ◦ Notify the user of the successful ticket purchase via email or in-app notification. 6. **Logging** ◦ Log all user actions related to miles redemption for auditing purposes. ◦ Record any system errors or exceptions that occur during the redemption process. ◦ Maintain a log of successful and failed redemption attempts for security and troubleshooting.	
Customer Portal: Foodie Points Program			
Earn foodie points from transactions on Foodie Card	As a User, I want to earn Foodie Points for every transaction I make using the card so that I can redeem these points later and enjoy rewards.	1. **Functionality** ◦ The portal should display the total Foodie Points earned from Foodie Card transactions. ◦ Users can view detailed transaction history to see how many points were earned per transaction. 2. **Security** ◦ Ensure all transaction data displayed is encrypted during transmission.	

		<ul><li>Implement session timeouts to prevent unauthorized access.</li></ul>3. **Validation**<ul><li>Validate that only transactions made with the Foodie Card are considered for Foodie Points.</li><li>Display an error message if the system fails to retrieve transaction data.</li></ul>4. **Error Handling**<ul><li>Provide a user-friendly error message for any failed transaction retrieval or processing.</li><li>Offer a "Retry" option in case of temporary system issues.</li></ul>5. **Output/Notification**<ul><li>Notify users via the portal when Foodie Points are successfully credited after a transaction.</li><li>Display updated Foodie Points balance immediately after each eligible transaction.</li></ul>6. **Logging**<ul><li>Log all user actions related to viewing and earning Foodie Points.</li><li>Maintain an audit trail for troubleshooting and security audits.</li></ul>	
Consume Earned points for new Orders	As a User, I want to be able to consume my earned points when placing new food orders, So that I can benefit from the rewards of the program and potentially reduce my order costs.	1. **Functionality**<ul><li>The system shall allow users to view their current Foodie Points balance.</li><li>Users can select Foodie Points as a payment option when placing new orders.</li><li>The system shall deduct the appropriate number of points from the user's balance upon order completion.</li></ul>2. **Security**<ul><li>All point transactions must be conducted over a secure connection.</li><li>User authentication is required before accessing the points redemption feature.</li></ul>3. **Validation**<ul><li>The system shall validate the sufficiency of points for the intended order.</li><li>An error message is displayed if the user's points are insufficient.</li></ul>4. **Error Handling**<ul><li>In case of a transaction failure, the system shall not deduct points from the user's account.</li><li>Clear error messages shall be displayed for different failure scenarios (e.g., network issues, server errors).</li></ul>5. **Output/Notification**<ul><li>Upon successful order placement, the user shall receive a confirmation message with order</li></ul>	

		details. ◦ The system shall update the user's Foodie Points balance in real-time and display it post-transaction. 6. **Logging** ◦ All point redemption transactions shall be logged with timestamps, user IDs, and transaction details. ◦ Any errors or exceptions during the transaction process shall also be logged for audit and troubleshooting purposes.	

Branches - Customer Portal - Banking App - FS Java Capstone

Branches

- Customer Portal: Check-In
- Customer Portal: Appointments
- Customer Portal: ATM/Branch Locator (Stretch)

Story	Description	Acceptance Criteria	Story Points
Customer Portal: Check-In			
Select Service	As a user, I want to select the service I need (e.g., deposit, withdrawal, transfer, loan consultation) when I check in through the banking app so that I can be directed to the appropriate department or representative.	1. **Functionality** ∘ The portal should display a list of available services in the branch. ∘ Users can select one service from the list. ∘ Once a service is selected, it should highlight or indicate the selection clearly. 2. **Security** ∘ Ensure that the service selection feature adheres to the application's overall security protocol. ∘ Implement session timeouts to protect user data in case of inactivity. 3. **Validation** ∘ Validate the user's selection to ensure a service is chosen before proceeding. ∘ Provide a notification or prompt if no service is selected when attempting to proceed. 4. **Error Handling** ∘ If the service list fails to load, display a user-friendly error message. ∘ Implement retry mechanisms for temporary failures in loading the service list. 5. **Output/Notification** ∘ Confirm the user's selection with a visual indication (e.g., a checkmark next to the selected service). ∘ Display a confirmation message or dialog box once the service selection is completed. 6. **Logging**	

		◦ Log the user’s actions (service selection, confirmation) for auditing and troubleshooting purposes. ◦ Ensure logs contain timestamps and user identifiers for traceability.	
View Branch	As a user, I want to view details of various bank branches within the User Portal, so that I can easily find information about branch locations, services, and hours of operation.	1. **Functionality** ◦ The system displays a list of bank branches. ◦ Users can select a branch to view detailed information including location, services provided, and hours of operation. ◦ Branch information is presented in a user-friendly format, ensuring readability. 2. **Security** ◦ All branch data displayed is strictly non-sensitive and public information. ◦ User sessions are verified for authenticity before displaying branch information. ◦ Branch data retrieval is performed over secure, encrypted channels. 3. **Validation** ◦ The system validates the availability of branch information before display. ◦ If a selected branch's details are not available, the system notifies the user accordingly. ◦ Invalid or corrupt data, if detected, is not displayed to the user. 4. **Error Handling** ◦ In case of a failure to retrieve branch information, the system displays a friendly error message. ◦ Error messages should not expose any technical details or vulnerabilities. ◦ The system provides the user with suggestions to retry or contact support if errors persist. 5. **Output/Notification** ◦ Successful loading of branch information is indicated through a visual cue. ◦ Any updates or changes in branch information are dynamically updated without needing to refresh the page. ◦ Notifications for updates or maintenance schedules related to branch information are provided when relevant. 6. **Logging** ◦ User interactions with the branch information feature are logged for audit and troubleshooting purposes. ◦ Logs include timestamps, user ID, and nature of interaction without logging any personal user	

		data. ◦ Anomalies in user behavior or system performance while accessing branch information are flagged for review.	
Get in Line	As a user, I want to join a virtual queue at a specific bank branch through the User Portal, so that I can minimize my waiting time and have a more efficient visit.	1. **Functionality** ◦ The system allows users to select a bank branch from a list. ◦ Users can view the estimated waiting time for each branch. ◦ A "Join Queue" button is available for users to enter the virtual queue. ◦ Upon joining, users receive a queue number and their position in the line. 2. **Security** ◦ User authentication is required to access the 'Get in Line' feature. ◦ The system ensures that queue data is encrypted during transmission. ◦ Session timeout is implemented to prevent unauthorized access. 3. **Validation** ◦ The system validates user selections for branch and queue. ◦ Error messages are displayed if a user attempts to join a queue without selecting a branch. ◦ The system checks for and notifies the user if the selected branch queue is full. 4. **Error Handling** ◦ In case of system errors, users receive a clear error message with steps to retry or contact support. ◦ The system handles failed queue join attempts gracefully, prompting the user to try again. 5. **Output/Notification** ◦ Once successfully in line, users receive a confirmation message with their queue number. ◦ Notifications are sent to update users on their queue status and estimated wait time. 6. **Logging** ◦ All user actions within the 'Get in Line' feature are logged for audit and troubleshooting purposes. ◦ Queue join and leave events are logged with timestamps and user identifiers.	
Customer Portal: Appointments			
Select Service	As a user, I want to be able to select the specific banking service I need when making an	1. **Functionality**	

	appointment at a branch through the User Portal, so that I can ensure my visit is efficient and my banking needs are met.	◦ The system shall present a list of available banking services to the user. ◦ Users can select one service from the list for each appointment. ◦ The system shall save the user’s selection and associate it with their appointment details. 2. **Security** ◦ All user interactions with the service selection feature shall be conducted over a secure, encrypted connection. ◦ The system shall not expose any sensitive user data during the service selection process. 3. **Validation** ◦ The system shall validate the user’s selection to ensure a service is chosen before allowing the user to proceed. ◦ Invalid or incomplete selections shall prompt the user to make a valid choice. 4. **Error Handling** ◦ In case of a system error during service selection, the user shall be presented with a friendly error message. ◦ The system shall offer the option to retry the selection process after an error. 5. **Output/Notification** ◦ Upon successful service selection, the user shall receive a confirmation on the screen. ◦ The system shall notify the user if the selected service is not available at their chosen branch or time. 6. **Logging** ◦ All user actions within the service selection feature shall be logged for auditing purposes. ◦ Error occurrences and system malfunctions within this feature shall be logged for further analysis and improvement.	
Select Banker	As a user, I want to be able to select a banker from my local branch through the User Portal of the Banking Application, so that I can schedule an appointment for in-person banking services.	1. **Functionality** ◦ The User Portal must display a list of available bankers from the user's selected branch. ◦ Users must be able to view each banker's specialties, availability, and ratings. ◦ The system should allow users to select a preferred banker for their appointment. 2. **Security** ◦ All user interactions with the banking portal must be conducted over a secure, encrypted connection. ◦ The system should implement robust authentication to ensure that only authenticated users can select a banker.	

		◦ Sensitive user data, such as user preferences and selections, must be securely stored and handled. 3. **Validation** ◦ The system must validate the availability of the selected banker before confirming the user's choice. ◦ Users should be notified if their selected banker is not available and prompted to choose another. 4. **Error Handling** ◦ In case of system errors or connectivity issues, the user should be presented with a clear error message. ◦ The system should provide suggestions or actions to resolve the issue, such as retrying the operation or contacting support. 5. **Output/Notification** ◦ Upon successful selection of a banker, the user should receive a confirmation notification. ◦ The system should inform the user of the next steps, like appointment scheduling. 6. **Logging** ◦ All user actions and system responses related to the banker selection process must be logged for audit and troubleshooting purposes. ◦ Logs should capture key information such as timestamps, user ID, and actions performed, while ensuring sensitive data is anonymized.	
View Calendar	As a user, I want to view a calendar in the User Portal of the Banking Application, which allows me to see available appointment slots at different branches, so that I can schedule my visit conveniently.	1. **Functionality** ◦ The calendar displays available appointment slots for various bank branches. ◦ Users can view the calendar on a daily, weekly, or monthly basis. ◦ Calendar integrates with the real-time availability of the branches. 2. **Security** ◦ Calendar data is only accessible to authenticated users. ◦ All calendar data transmissions are encrypted. ◦ User session times out after a period of inactivity, requiring re-authentication. 3. **Validation** ◦ The system validates the availability of slots before allowing a user to view them. ◦ Invalid date or time entries are flagged with an error message. 4. **Error Handling**	

		◦ In case of a system error or downtime, a friendly error message is displayed. ◦ The system offers a 'Retry' option after an error occurs. ◦ If the calendar fails to load, users are prompted to check their network connection or try again later. 5. **Output/Notification** ◦ After successfully viewing the calendar, a confirmation message is displayed. ◦ Notifications are sent to users if there are any changes in the availability of slots they are interested in. 6. **Logging** ◦ All user interactions with the calendar feature are logged for auditing purposes. ◦ Error logs are maintained for system failures or exceptions.	
Customer Portal: ATM/Branch Locator (Stretch)			
Search for Branches	As a user, I want to be able to search for bank branches within the user portal of the banking application, so that I can easily find the nearest branch location, operating hours, and available services.	1. **Functionality** ◦ The portal must provide a search function to find bank branches. ◦ Users should be able to enter search criteria (e.g., zip code, city). ◦ The system displays a list of branches matching the search criteria. ◦ Each branch listing includes address, operating hours, and services available. 2. **Security** ◦ Ensure that the branch search feature complies with data privacy regulations. ◦ Implement measures to prevent unauthorized access to branch information. ◦ Search queries should not expose any sensitive bank or user data. 3. **Validation** ◦ Input fields for search criteria must validate user input for format (e.g., correct zip code format). ◦ The system should prompt the user to correct invalid search inputs. 4. **Error Handling** ◦ In case of a failed search (e.g., no network connectivity), display an appropriate error message. ◦ Provide users with suggestions to resolve the error (e.g., check internet connection, try a different search term). 5. **Output/Notification**	

		◦ After a successful search, display the results in an easy-to-read format. ◦ If no branches are found, inform the user with a message like "No branches found for the entered criteria." 6. **Logging** ◦ Log all search queries for branches for auditing purposes. ◦ Ensure logs capture date, time, and user ID, but not the specific search details to maintain user privacy. ◦ Logs should be stored securely and only accessible by authorized personnel.		