# Theme Park Bites

CSE2410 Project Report

Austin Phillips, Adrian Rodriguez, Evan Thompson

12/10/2023

## Abstract

This report describes "Theme Park Bites," a web application aimed at assisting individuals with food allergies in navigating dining options at theme parks. The application's primary goal is to simplify the process of identifying allergen-free food items, enhancing the overall experience for these visitors.

Developed utilizing the Blazor Server Framework and MudBlazor for interface elements, the project employs a streamlined architecture to ensure a responsive and intuitive user experience. The design phase focused on creating a clear, easy-to-navigate data structure, while the construction phase emphasized robust security measures and efficient functionality, particularly in data management.

In the testing phase, the application underwent rigorous checks to confirm its operational integrity across various devices, with a special emphasis on mobile compatibility. The report concludes with an evaluation of the application's performance and potential areas for future enhancements. "Theme Park Bites" represents a thoughtful and technically sound solution to a common challenge faced by theme park visitors with dietary restrictions.

## Table of Contents

Title....1

Abstract....2
Table of Contents....3

Introduction....4
Methodology....4
Design....5

Construction....6
Testing....7
Conclusion....8
References....9
Appendices....9

## Introduction

The project's initial idea was to create an application that could benefit users by increasing their time efficiency in a specific use case. The next idea was to base the project around menus, but increased complexity was needed. So, the finalized idea was a user-orientated application to efficiently search food menus at theme parks focusing on attendants with allergies and diets.

The decision to base the project around theme park menus compared to any other restaurant was due to realizing a problem that could be fixed. This problem was because most menus were designed to appeal to a customer's appetite by separating food options by categories, making it difficult for users with allergies and specific diets to find items that meet their criteria. Regarding theme parks, attendants only have a little time to browse a website's menu, with theme parks tending to be overcrowded and attendants constantly being on the move, making wasted time even more significant. This led to the birth of "Theme Park Bites," which looks to create a user-friendly menu with the capability to filter items by common allergens, dietary restrictions, location, and area.

## Methodology

Multiple process models were discussed to divide the software development phases, but the prototype model was the most effective model that best fit the goal and requirements. The prototype model (Figure 1) is to repeatedly create a tangible and visual representation of the software before finalizing the product. This model proved especially useful during the implementation of the filters, allowing each filter to have its own model to test if it is working correctly. This process made testing more straightforward than having all filtering options in a single module. Since the goal was to provide a simple menu filtering experience to the user, prototyping allowed for testing each procedure with the overall filtering process to ensure that each filter linked correctly to the corresponding tags. In the end, the prototype model helped immensely by enabling the continuous refactoring of the code until the product's finalization.

A list of requirements was created that would need to be met to achieve the desired goal of a user-friendly menu filtering application. The requirements were split among stakeholders who would be invested in the project. These stakeholders included the application users, the site admins, and theme park management. The requirements for the users were to be able to filter from a multitude of options such as festival, park, area, location, allergies, and tags. The idea was that users would click park or festival and then filter for what they need based on location, tag, and allergies. In addition to filtering, the user's requirements included being available on a mobile device to provide a fast means of accessing the information they need. Other non-functional requirements that were not the main focus but would improve the user experience are the additional features of a downloadable menu, a rating system, and a light/dark background theme. Of these additional features, only two were implemented due to time constraints, leaving the rating system to be one feature that was wanted to be implemented but could not be.

The second list of requirements falls under the site admin stakeholder. The site admin would have access to alter the application's data and change what is displayed to the users. This means admins can change parks, festivals, locations, tags, and allergies. Additionally, they can change food items, food descriptions, and prices. Allowing for fast and efficient changes to be implemented into the app for rotating food items or adding items to new locations. The admin can make changes to either parks or festivals to ensure that changes made to one does not alter the other since park menus are generally unchanging, while festival-themed menus are constantly rotating items.

The last requirement is based on theme park management. They must supply official menus to incorporate into the database. Since the application focuses on user satisfaction through quick and accurate filters, it must reflect well on the theme park that chooses to incorporate it. Maintaining the theme park's image is essential, and marketing it to attract more customers is integral to having a successful product.

## Design

The design of "Theme Park Bites" implements a data class mapping approach (Figure 3), where each data class is directly linked to a corresponding model class. This one-to-one mapping simplifies the system's overall structure, enabling each module's independent development, testing, and maintenance. This approach leads to a flexible design that is easily adaptable to refactoring and adding functionality.

In choosing the Model View Controller (MVC) architecture, the application benefits from its pattern, which distinctly separates the data (Model), user interface (View), and business logic (Controller). This separation works particularly well for "Theme Park Bites" due to the need for consistent and smooth interactions between the user and the data. The MVC architecture enhances the app's efficiency and streamlines the development process, allowing for distinct phases of development focused on each aspect of the application.

The application's interface design is user-focused, emphasizing ease of navigation and optimal user experience. It organizes food offerings according to their locations within the park, enabling users to conveniently find and select their preferred dining options. In Emphasizing cross-platform compatibility, the design ensures a seamless and accessible user interface on mobile devices. This mobile-optimized approach is essential for accommodating the dynamic environment of theme parks, where users predominantly rely on mobile access. The application's user interaction flow, which is central to the design, is depicted in Figure 2, illustrating the use case diagram that guides the development of these design elements. Additionally, Figure 4 presents an initial design concept that was ultimately not used. This early design did not meet the project's requirements for mobile accessibility, leading to its exclusion in favor of a design that better accommodated mobile phone usage.

## Construction

In the construction phase of "Theme Park Bites," the focus was on implementing the design using the Blazor Server Framework, known for its robust server-side processing capabilities. This choice proved vital for the application, particularly in enhancing its performance on various devices, including those with limited processing capabilities. The app offloads significant computational tasks from the client side by leveraging the server-side processing, resulting in a smoother user experience.

Dapper ORM was crucial in this phase, especially in securing data transactions. By using this Object-Relational Mapping tool, the application ensures secure and efficient interactions with the database, thereby preventing common security vulnerabilities like SQL injection attacks. This choice was instrumental in maintaining the integrity and security of user data, a critical aspect given the sensitive nature of dietary and allergen information.

The MVC architecture significantly influenced the development of data and model classes. By clearly separating concerns, it allowed for a more organized and efficient construction process. Each data class, representing the application's data layer, was developed with a focus on accurately modeling the information structure relevant to the app's functionality. Correspondingly, model classes were designed to integrate into a single data class, ensuring seamless one-to-one mapping and interaction. This clear distinction and alignment between data and model classes enhanced the overall coherence and maintainability of the codebase. The MVC architecture immensely helped the development of the interface for performing database operations within the application. The Controller layer in the MVC structure played a crucial role in mediating between the View (user interface) and the Model (data), handling the logic for database operations. This separation allowed for a more modular approach, where database interactions could be efficiently managed and updated without impacting the user interface or the underlying data structure. While each model class has an individual data class, these classes were designed to create a hierarchal structure with the ParkModel on the top and the FoodModel on the bottom; this structure is shown in the code snippet in Figure 5. This allowed the ParkModel produced from the query in Figure 5 to be filtered by allergy, tags, location, and area.

The user interface construction was undertaken with a keen focus on responsiveness and interactivity. MudBlazor components were utilized to develop a user interface that is aesthetically pleasing and highly functional across different platforms, especially mobile devices. Critical features like interactive menu filtering were implemented using these components, which allowed for an intuitive and user-friendly experience. This feature enables users to easily filter food options based on various criteria, such as allergen content, thus enhancing the overall usability of the application.

Throughout the construction phase, the development team strongly emphasized adaptability and scalability. This approach ensured that the application could be easily updated and expanded in line with emerging user needs and technological advancements. The combination of Blazor Server Framework, Dapper ORM, and MudBlazor components resulted in a well-rounded construction of "Theme Park Bites," aligning with the initial design objectives and paving the way for an effective and secure user experience.

## Testing

### Component Testing

This stage involved rigorous examination of the application's modular components. Individual data and model classes were tested through their corresponding admin panels. By inputting, updating, and deleting data, we verified the correct functioning of these classes (Figure 3). The success of these operations was confirmed by matching the data entered in the admin panel with what was displayed in the application and stored in the database. This verification ensured that each component interacted seamlessly with the database, as demonstrated by the consistency between the data table and the database records.

The light and dark mode functionality underwent testing across multiple devices. The application's responsiveness to system settings for theme changes was validated, ensuring the user's preference for visual themes was accurately reflected. This adherence to user settings was crucial for maintaining a personalized experience.

### Integration Testing

Integration testing focused on the cohesive operation of the application's components when functioning together. The application was put through scenarios mimicking real-world mobile device usage to ensure full functionality. The proper integration of components was evident in the responsive behavior of the application on mobile platforms, confirming that the application's design was effectively translated into a practical and operational framework.

Through these testing protocols, "Theme Park Bites" was verified to meet most of its core functional requirements. However, the ability for users to rate food offerings and see those ratings alongside food items was not able to be implemented within the project's timeframe. Despite this, the application's modular design ensures that such features can be easily integrated in future updates.

## Conclusion

The creation of "Theme Park Bites" stemmed from the goal of developing an application that enables users to view food offerings at theme parks through various filters. Overall, it successfully does that while fulfilling most of the requirements laid out at the beginning of development. Though it only fulfills some requirements, the only one left to be implemented is a non-functional requirement, which can be added to the application by an update without drawing from the project's primary goal. In Conclusion, this application would be a fantastic addition to any theme park and significantly decrease time spent by users searching for food items compatible with their dietary restrictions.

## References

https://disneyworld.disney.go.com/dining/animal-kingdom/flame-tree-barbecue/menus

## Appendices

Figure 1

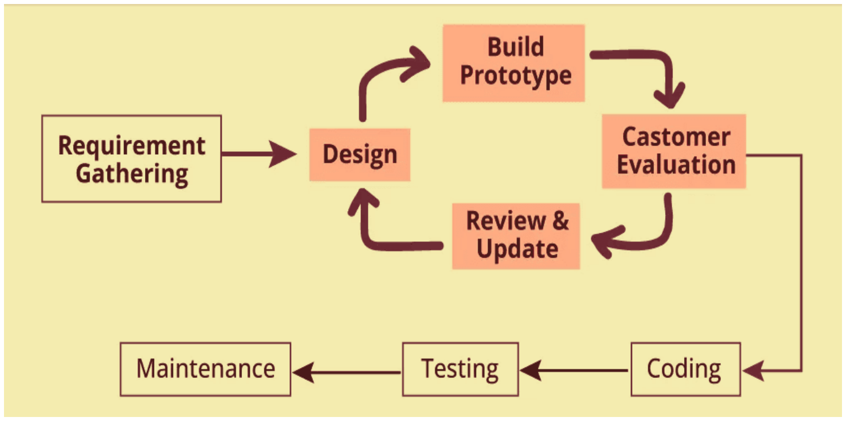


Figure 2

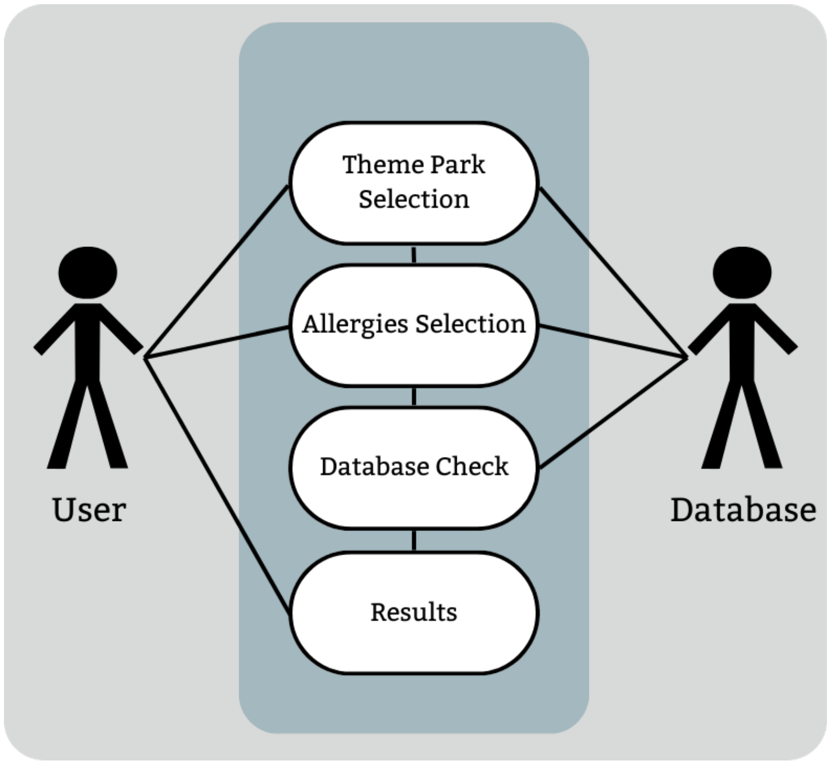


Figure 3

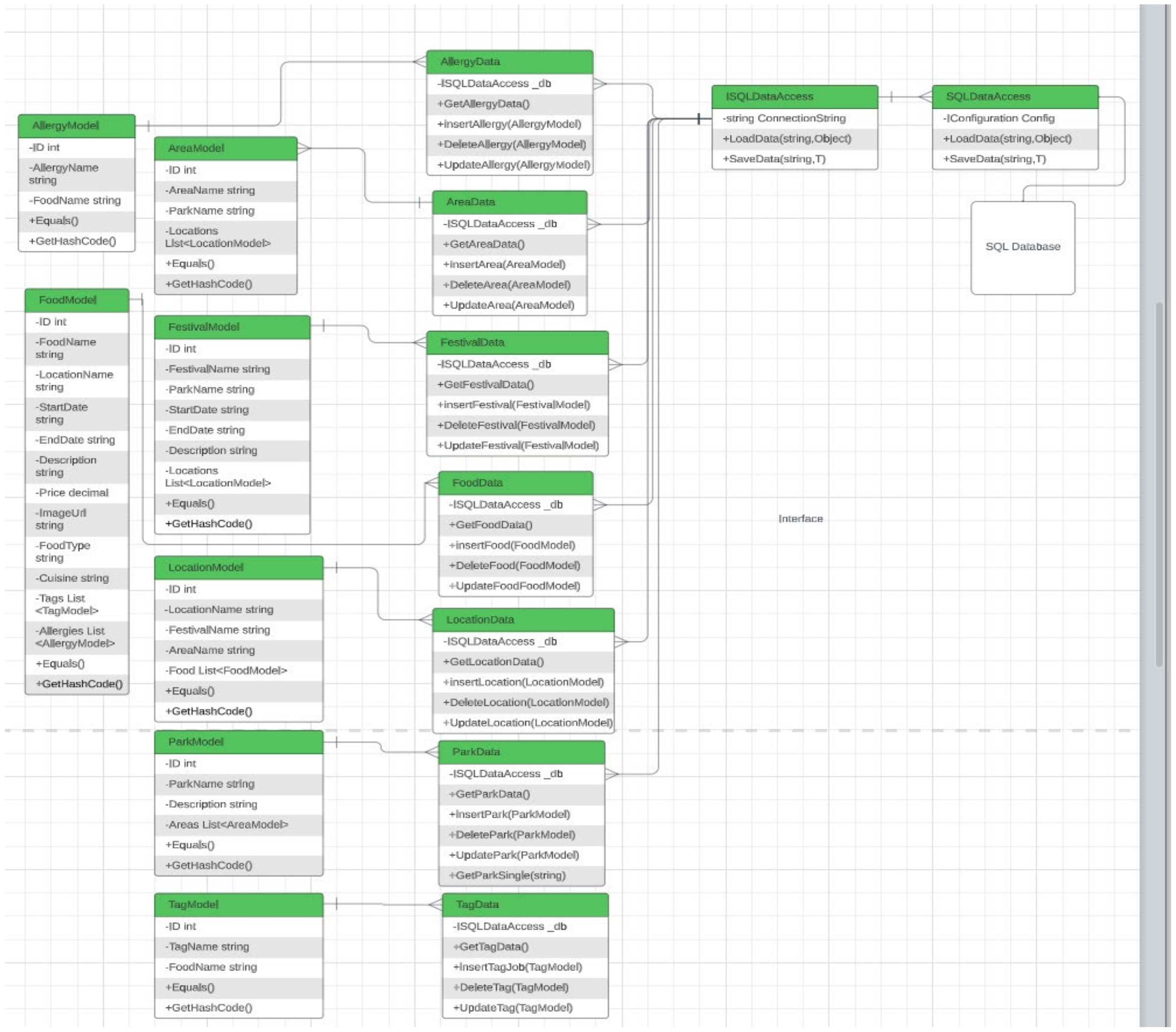


Figure 4

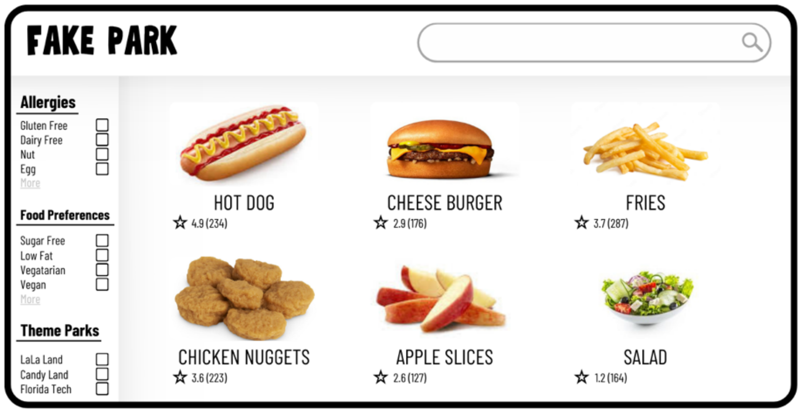
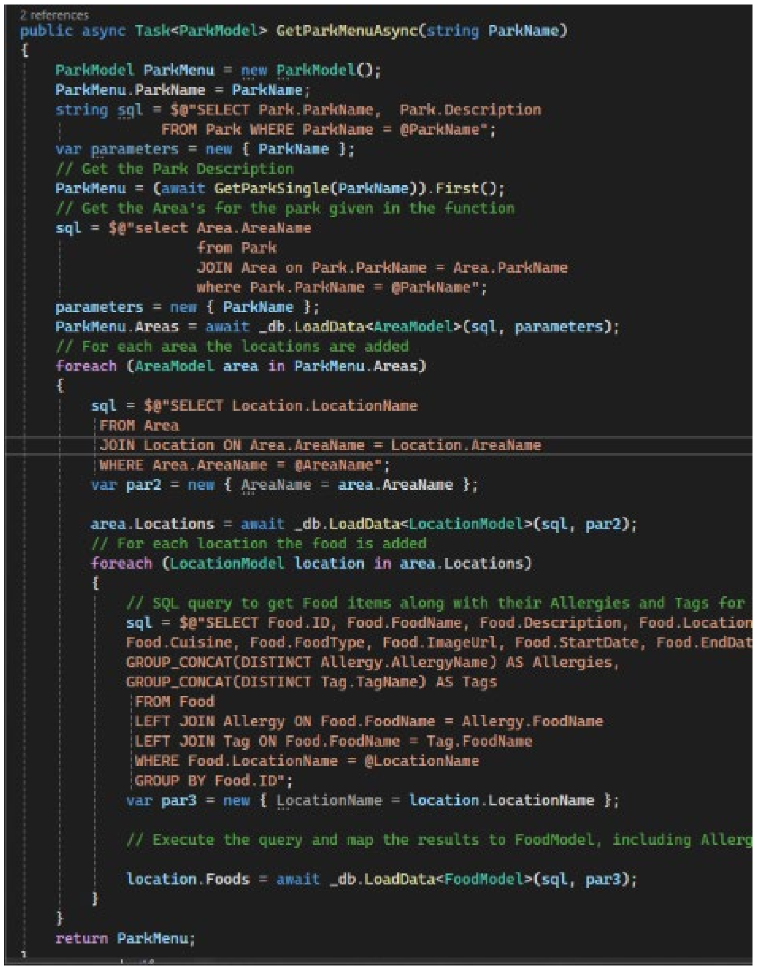


Figure 5



```
2 references
public async Task<ParkModel> GetParkMenuAsync(string ParkName)
{
    ParkModel ParkMenu = new ParkModel();
    ParkMenu.ParkName = ParkName;
    string sql = $@"SELECT Park.ParkName,  Park.Description
                FROM Park WHERE ParkName = @ParkName";
    var parameters = new { ParkName };
    // Get the Park Description
    ParkMenu = (await GetParkSingle(ParkName)).First();
    // Get the Area's for the park given in the function
    sql = $@"select Area.AreaName
                    from Park
                    JOIN Area on Park.ParkName = Area.ParkName
                    where Park.ParkName = @ParkName";
    parameters = new { ParkName };
    ParkMenu.Areas = await _db.LoadData<AreaModel>(sql, parameters);
    // For each area the locations are added
    foreach (AreaModel area in ParkMenu.Areas)
    {
        sql = $@"SELECT Location.LocationName
         FROM Area
         JOIN Location ON Area.AreaName = Location.AreaName
         WHERE Area.AreaName = @AreaName";
        var par2 = new { AreaName = area.AreaName };

        area.Locations = await _db.LoadData<LocationModel>(sql, par2);
        // For each location the food is added
        foreach (LocationModel location in area.Locations)
        {
            // SQL query to get Food items along with their Allergies and Tags for
            sql = $@"SELECT Food.ID, Food.FoodName, Food.Description, Food.Location
            Food.Cuisine, Food.FoodType, Food.ImageUrl, Food.StartDate, Food.EndDat
            GROUP_CONCAT(DISTINCT Allergy.AllergyName) AS Allergies,
            GROUP_CONCAT(DISTINCT Tag.TagName) AS Tags
             FROM Food
             LEFT JOIN Allergy ON Food.FoodName = Allergy.FoodName
             LEFT JOIN Tag ON Food.FoodName = Tag.FoodName
             WHERE Food.LocationName = @LocationName
             GROUP BY Food.ID";
            var par3 = new { LocationName = location.LocationName };

            // Execute the query and map the results to FoodModel, including Allerg

            location.Foods = await _db.LoadData<FoodModel>(sql, par3);
        }
    }
    return ParkMenu;
}
```