**A report submitted in partial fulfilment**

**of the regulations governing the award of**

**the Degree of**

**BSc. (Honours) Programme Name**

**at the University of Northumbria at Newcastle**

**Project Report**

**Table Reservation System for a Restaurant**

**General Computing Project\***

***Scott Mains***

***2021 / 2022***

**DECLARATIONS**

I declare the following:

(1) that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and references to **ALL** sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation is ...................................

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on the eLearning Portal (Blackboard), if deemed appropriate, to allow future students the opportunity to see examples of past dissertations.  I understand that if displayed on eLearning Portal it would be made available for no longer than five years and that students would be able to print off copies or download.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other School or from other institutions using the service.

In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

(5) I have read the Northumbria University/Engineering and Environment Policy Statement on Ethics in Research and Consultancy and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED:

##### ACKNOWLEDGEMENTS

I would like to express gratitude to my project supervisor, John Rooksby, for his support and guidance throughout the project.

I would also like to thank Slice for allowing me to create a booking system for their restaurant and providing me with great feedback.

# Abstract

This project aims to create a web application for a pizza restaurant that functions as a table reservation system; including an admin dashboard to deal with customers and bookings. The reason to create this project is that there has been an increase in customers using online booking systems since the COVID pandemic, and it is a great way to collect customer data for Customer Relationship Management. The system was built using HTML, CSS, PHP, and JavaScript (React Framework). Different react packages implemented using NPM were also included to make the web application more dynamic and fluid.

# Introduction

## Aims of Project

The aims of the project are:

* To investigate how a table reservation system is created and how the data can be used in Customer Relationship Management.
* To build a web application for a pizza restaurant that integrates a table reservation system and a CRM for the restaurant owner.

## Project Objectives

* Create a literature and technology review:
  + Look at existing booking/reservation systems and identify their good and bad features. Take ideas that could be implemented in my design.
  + Examine existing plugins used for calendars that can be integrated into the reservation system.
  + Find existing CRM systems used in restaurants and find ways I can use user data created from the reservation system.
  + Find academic articles on CRM systems and table reservation systems.
  + UCD design for restaurant website and booking systems.
  + Look at privacy and data protection issues regarding storing user data.
* Establish and prioritize the requirements of the product:
  + Find out the client’s requirements
  + Create a storyboard and initial vision of the application
  + Find target audience(s)
  + Persona(s) and Scenarios
  + User requirements document
* Create designs for the web application:
  + Initial design storyboard and wireframes.
  + Low & High-fidelity prototypes
  + Use case diagrams
  + Class diagrams
  + Entity-Relationship diagrams
* Create the product based on the design specification.
* Test the product:
  + Functional testing
  + Usability testing (Users and admin)
* Make changes to the application based on the testing process.
* Evaluate the product.
* Evaluate the processes and my performance

## 1.3 Project Overview

This project is a thorough investigation into how a table reservation system for a restaurant is created and how it can be utilised for a Customer Relationship Management system. A functioning reservation system as a web application has been created for a pizza restaurant to support this. The customer will be presented with a functioning dynamic web application created with React which will then redirect them to the reservation system which allows the user to book a table slot at the pizza restaurant. The customer will be able to book the table as either a guest or create a user account, which will allow them to collect loyalty points and receive special promotions. The admin of the website will have access to a dashboard that gives them an overview of all bookings made on the website and will give them the power to cancel and create bookings. There will be lots of other extra functionality that will act as a Customer Relationship Management (CRM) system, such as being able to send out marketing emails to all registered users and applying “comments” to customers which will allow them to personalise the experience for each guest. The application will be as secure as possible and will have authentication integrated for both the admin and the users using a JWT token.

## Project Context and Justification

Since the COVID-19 pandemic, it has become a necessity to reserve tables at restaurants to avoid queues and prevent congestion from customer walk-ins due to the new government health guidelines. It is also useful for restaurant owners to minimise queues from walk-ins so that the customers don’t have to wait long to be seated and in turn be disappointed. A table reservation system can help tackle these problems by allowing the user to reserve a table for a specific time slot. The table will then become available again for another user once the allotted time has occurred. With the data collected from the table bookings, the restaurant owner can also see when his restaurant is most popular on a seasonal, weekly, and daily basis. On top of this, the user will have the ability to create an account and opt-in for special offers from the restaurant. This data can be very important in a customer relationship context, as they can promote their brand on popular trends with special offers etc. The restaurant needs to try and build a customer relationship, as it is a sure-fire way to boost sales.

A client has reached out to me to design a system that allows the user to book a table at their restaurant based on such parameters as date, time, and table size. They have requested that the reservation system be integrated into a website that promotes their restaurant, with a booking page that takes the user directly to the table booking form. Being able to view the table layout visually would also be a bonus so that they can choose exactly where they want to sit inside the restaurant. The client has requested an interface for him as the admin, so they can view all the bookings that are submitted on a daily and monthly basis. The admin page can also display analytics for the data collected from the reservation system; so that they can see the extent of when most bookings occur and on what days. The user can also create an account when reserving a table at the restaurant. With the account, the user can build up loyalty points with every successful booking they have at the restaurant. This is great for both the restaurant owner and the user, as the user gets a discount, and the owner gets invaluable data and a returning customer.

So overall, the project idea was chosen because of the need of the client and the increase in demand for online alternatives due to the COVID pandemic.

## Summary of Tools and Techniques

The following technology was used in the project to build the web application:

* React – This was used to create the client-side of my project. Everything that the user sees was created using React, and of course standard HTML and CSS as well. Various plugins and packages were also integrated from Node Package Manager such as the calendar component.
* PHP – This was used for the back-end of the web application. PHP can interact with the database and update it with new bookings, update existing bookings or delete entries.
* MySQL – This was used to create a database for the system. The MySQL database stores the user data and booking data created from the reservation system.
* TailwindCSS – I decided to use TailwindCSS to create the design for my project rather than basic CSS. Tailwind is great as you can integrate ready-made components that look sleek and modern. This allowed me to focus more time on the actual functionality of the system without having to worry about the design looking subpar.
* SCRUM – This is an AGILE framework that was employed throughout my project. It split up my workload into different parts and I tackled each bit through “Sprints”. It also made me have close contact with the stakeholder throughout the design process and also when we evaluated it at the end for feedback.

Analysis Chapters

# 2. Literature Review

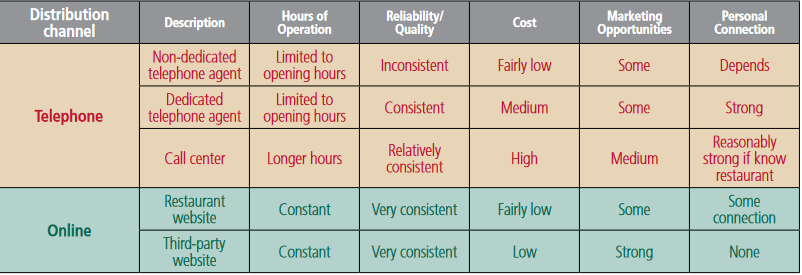
## 2.1 Introduction

This section will discuss literature that is relevant to booking and reservation systems, and how the user data can be used to create a Customer Relationship System to boost customer relations and increase repeat customers. This will help understand the challenges that surround restaurant reservation systems and the best way to tackle integrating one. I will also discuss how useful booking systems are overall, especially due to the increase in demand for them due to the COVID pandemic.

## 2.2 Online Reservation Systems

Reservation Systems have become the norm in modern times when scheduling table slots within restaurants. It has been noted that technologically savvy Generation X and Millennials expect easy and smooth access to restaurant services using their mobile devices [1]. The report by Flynn and Buchan, stated that the best practices for Netwaiter (a notorious restaurant marketing platform) to attract local customers is to offer online reservations and have a mobile-friendly website [1]. Alongside this, it also states that you should have a static web page for the menu within your web application. I will be integrating all these practices within my design.

Now that the



*Figure 1 Comparison of Reservation Methods*

In a Hospitality report from Cornell University, a study was conducted to garner customers’ views on online reservations compared to phone call reservations [2]. There was evidence that both methods had their perks and weaknesses. A good reason why online reservation systems are beneficial is that hours of operation are constant. This means that the customer can reserve a table outside operating hours, unlike telephone reservations. Alongside this, it is also noted that it is much more reliable and keeps costs low as nobody needs to man the landline to answer calls. In terms of negative aspects, it is important to note that with online reservations there is the chance of reducing personal connection with the customer – especially using third party web applications. Because of this, both lines of communication must be open to the user. It also makes it more personalised if the customer can book directly from the restaurant’s website rather than a third party such as OpenTable, which is exactly how I will be implementing it.

## 2.3 The effects of COVID-19

The need for such a system has increased dramatically due to the COVID pandemic. This is so the restaurant can control store capacity to reduce the spread and enable no congested waiting times for customers. According to Quidini, 32% of customers believe that scheduling a time slot would make them feel safer against COVID-19 [1].

COVID has also exponentially increased the amount of online traffic toward e-businesses and online sales channels. Despite a restaurant not being able to completely remove the offline aspect, it can try and shift certain parts of the business online so that it can shift with this growing trend of online channels.

## 2.4 CRM Systems

CRM stands for customer relationship management, and it is essentially a system that helps business owners nurture their relationships with their clientele [3]. The emergence of things such as the Internet of Things has meant that businesses are now trying to identify business strategies to personalise their relationship with their customers.

There are 4 main components of the relationship between a business and its customers: customer acquisition, customer retention, relationship expansion and defection [8]. Within my system, I am primarily going to focus on how I can improve customer acquisition and retention.

## 2.5 UCD Approach

UCD means User-Centred Design, and the UCD approach essentially means that throughout the iterative design process we always have the user at the centre of the design. Throughout the development process, I am going to make sure I consider the user every step of the way - their needs, objectives, and feedback [10]. There are several key design principles when following UCD. One of these is involving users at the very beginning of the design process. I am going to do this using the questionnaires and the interview with the client. Including these at the start will make sure that the user’s requirements are thoroughly met, and that I do not go down the “wrong path” and create something I can’t reverse.

In a survey that asked questions about UCD methods and their effectiveness [11], it is shown that

The process of easily reading and translating documents can be helped by using the UCD method. UCD techniques, methods, tools, and procedures can be implemented that help design interactive systems that are built based on user experience [19]. UCD is essentially translating human participation and experience into designs. [20]

## 2.6 RESTful API

Diagram

Description automatically generated

*Figure 1 RESTful API architecture*

Representational state transfer (REST) or RESTful web services is an architectural style for an application program interface. This type of interface deals specifically with HTTP requests to access and use data. The information can be delivered in several different formats: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. I will be primarily sending and receiving JSON in my web app as it is language agnostic and readable by both humans and computers [15].

The diagram shown in Figure 2 shows the architecture of a RESTful Web application.

### 2.6.1 JSON Web Token

I am going to implement the JSON Web Token (JWT) authentication process for this application, which is a token-based authentication and new industry-standard practice for RESTful web applications such as the one I am creating. Within this section, I am going to discuss the architecture for token-based authentication and the best standard practices for storing JWTs.

## 2.7 Security Risks

There are plenty of security risks that could occur during the development of the system. The web application is going to handle mildly sensitive data such as phone numbers and emails – and of course passwords. I also need to be able to restrict access to the admin dashboard so that unauthorised users cannot access this. I am going to discuss how I plan to secure my web application and the variety of security risks that could occur and how I could combat them.

#### 2.6.2 XSS Attacks

An XSS attack, also known as Cross-Site Scripting, is a type of injection in which malicious scripts are injected into your website [12]. This type of attack occurs when an attacker sends code in the form of a browser side script to an unsuspecting user. The browser doesn’t have any way to know if the script is trustworthy. The flaws that allow these attacks to succeed are common and can be found whenever a web application accepts user input in its output without verifying or encoding it.

#### 2.6.3 SQL Injection

A SQL injection attack involves inserting or "injecting" a SQL query into the application via the client's input data [13]. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), perform database administration operations (such as shutting down the DBMS), recover the content of a given file on the DBMS file system, and, in some cases, issue commands to the operating system. SQL injection attacks are injection attacks in which SQL commands are inserted into data-plane input to influence the execution of predetermined SQL commands.

#### 2.6.4 URL Manipulation Attacks

The URL of a website (Uniform Resource Locator) is an address to a unique resource on the web. If a hacker manipulated certain parts of a URL, they could then deliver pages that they aren’t supposed to have access to [13]. On dynamic websites, parameters are usually passed via the URL. An example of this would be the end of the URL ending in “/api/authors?id=59657”. In this example, the URL is accessing an API and using the authors and its ID as parameters to access specific information from the API. If the creator of the website doesn’t secure these parameters, then they could run the risk of exposing web pages and information that they are not supposed to access.

#### 2.6.5 Storing User Data

# 3. Requirements Capture

## 3.1 Introduction

A requirement capture plan is a useful exercise to undertake early in a project life cycle to establish the scope of the project. The main reason is to understand the system from a user’s perspective and find their common needs and expectations. Within this section, I am going to discuss the research I gathered during this exercise from the various artefacts I created, and how useful they will be going forward in the overall design of the booking system.

## 3.2 Capture Process

The requirement process is the foundation of every successful software project [4]. It’s been reported by the IDC that 25% of IT projects experience outright failure, and 50% require reworking [5]. This is in part due to the lack of requirements captured in the initial planning stage.

The way I am going to capture my requirements for my system is by creating several different artefacts. Through these artefacts, I will develop the needs of the user in both a customer context and a staff/admin context. I am initially going to conduct a competitor review to gather important functional requirements that already exist in similar systems. Once I have gathered important features from the competitor analysis, I will then conduct a client interview with the stakeholder. Through this interview, I will propose features that I have gathered from the competitor analysis and gather information on what they believe are important requirements for the system. Both artefacts should be enough to gauge the functional requirements from a staff/admin perspective.

I will then send out a questionnaire to prospective users of the system. This questionnaire will gather intelligence on what users look for when trying to reserve a table using an online web-based system.

## 3.3 Competitor Review

The competitor review will consist of the currently existing booking and reservation systems on the market. I have looked specifically at the features that these systems incorporate and how they have utilised them to make it a seamless experience for the user [Appendix A]. I also did a separate analysis looking at the different systems from an admin perspective [Appendix B]. I am going to summarise key points discovered in the competitor analysis below and my most important findings.

### 3.3.1 OpenTable

OpenTable is a restaurant reservation service that was founded in California but now expands internationally [2]. OpenTable is a hub for restaurants in the user’s local area and acts as a discovery platform for restaurants as well as a booking system. The user simply chooses the restaurant they want to book, and they are taken to the restaurant’s page which allows them to see available times and dates. They can then reserve a table according to their needs.

From a client perspective, there are lots of different features available to make it a seamless experience for the user to reserve a table at their chosen restaurant. The first thing worth mentioning is how intuitive the whole process is for finding available tables and reserving a slot. The first thing the user needs to choose is their party size, followed by the date on which they want to book the table. This then shows the time availability according to both parameters. If there are no times available for that specific slot, then the system proposes times within 2.5 hours of the initial time slot. I think this is a great process compared to other systems as it allows the user to immediately know whether there is any availability for their circumstances.

Once the user has chosen a slot, they are taken to a page on which they can sign in with an account or proceed with the booking as a guest. Creating an account allows the user to collect loyalty points that can be exchanged for discounts in future bookings. This creates value for the user so that they will be more interested in creating an account. The overall layout is easily readable and simple. It consists of a very basic colour scheme of red, white, and black so all the text stands out. Any important information can then be put in a different colour to make it stand out from the rest of the page. An example of this is how they show the safety precaution methods in a darker grey colour.

There are many different features available from an admin perspective. OpenTable manages customer relationship management very well and offers many

### 3.3.2 TheFork

TheFork is another restaurant booking and discovery platform now joined with the huge review platform Trip Advisor.

### 3.3.3 SimplyBook.Me

This system is most similar to what I am trying to accomplish with my design. The website is designed first, and the reservation system is built directly into the website. The other competitors usually have been added to a hub, like that of uber eats, rather than have a website specifically designed for the brand and the booking system directly integrated.

### 3.3.4 BookingNinja

### 2.3.5 Most important findings

## 3.2 Requirements for Customers

To establish the requirements for potential customers, I created a Requirement Specification [Appendix F] which includes all the functional requirements that a customer would need when using the booking system. I established these requirements through previous artefacts such as the competitor review, questionnaire, user stories and personas.

## 3.3 Requirements for Staff

To establish the requirements for the admin of the restaurant, I again created a Requirement Specification [Appendix F] which includes all the functional requirements that an Admin would need when using the booking system. I established these requirements through the client interview I conducted in [Appendix D].

# 4. Tools and Techniques

## 4.1 Introduction

The following section is going to go through all the different languages and technologies I will be using throughout the project development. I am then going to discuss the project framework I will utilise to make everything go smoothly throughout the lifecycle of the project.

## 4.2 Adobe XD

Adobe XD is design software used for web and mobile applications. It is a great tool to create prototypes and mapping out the design and layout of my application. I will be using Adobe XD to create the low fidelity wireframe of my reservation system

## 4.2 Database

I have opted to use MySQL to store the data for the reservation system. MySQL is a relational database management system and is the most widely used database technology used across many huge companies.

## 4.2 Programming Languages

As my project is full-stack, I will be splitting it up into server and client-side tasks that will require various languages. For the server-side, I will be using the PHP language in conjunction with React for the client-side.

## 4.2.1 PHP

PHP is an open-sourced scripting language that is primarily used in web development. I have decided to opt for PHP as it has great versatility.

As my application is going to be an API - in specific a RESTful API - I wanted to measure up all the appropriate languages that could be used.

## 4.2.2 ReactJS

React is a JavaScript library developed and maintained by Meta (formerly Facebook).

## 4.3 Project Framework

The project framework I will be implementing throughout the project life cycle is AGILE. Despite Agile being used primarily within a team context (mainly SCRUM), many aspects can be applied to the solo developer to create a successful project. I am going to discuss some of the things I have implemented from the SCRUM framework below.

The agile methodology is an iterative process that lends itself to rapid application development [5] –fitting perfectly into my project. Everything I do and create throughout the life cycle will be test-driven and done in small sprints. A small sprint is essentially a time-boxed period to accomplish a chosen user story that was set in the research stage. I am going to create a Trello board and turn my user stories into a product backlog; this is a list of all the items I intend to complete at some stage for the product.

The main method I’m going to borrow from AGILE is keeping close contact with the client. It is important that the client is closely engaged in the development and can change the requirements or accept any suggestions proposed.

# Summary

Within this section, relevant literature was investigated that will be applied throughout the project cycle. Requirements for the web application were then established through a multitude of different artefacts: Competitor reviews, Persona(s) and Scenarios, Questionnaires & Interviews. These artefacts were then condensed into a requirement specification which will be used to create the overall design for the application, which will be discussed in the next section.

Synthesis

## Introduction

This section is split into two main chapters that make up the synthesis. The purpose of the design section is to utilise the requirements I have created and make a strong design for my application which accounts for all the requirements. The second chapter is based on the implementation of the design. The software is going to be designed and implemented using the AGILE methodology which implies that a strong design leads to a more coherent development.

# 5. Design

## 5. Design Specification

The design specification is then created from the user requirements established in the analysis section. I first created use case descriptions for each main requirement for the system, followed by diagrams to help organise my thoughts and pinpoint every feature I want to include in my wireframe. I then created a low-fidelity wireframe to map out everything, with emphasis on looking at the interface and making it as accessible as possible for the user. I decided against using a High-Fidelity wireframe as I have opted to use TailwindCSS which will handle a lot of the design aspects.

## 5.1 Use case Descriptions and Diagrams

Use case descriptions are a logical flow of how a user would perform tasks on the website. In the requirement capture plan, I generated user stories from both the customer and admin perspectives. In this section, I am going to break down these user story activities into a use case and how the user would achieve it from their perspective. All the use case descriptions are present in [Appendix \*], but I will break down important things I learned from these descriptions.

## 5.2 Sequence Diagram

The next part of the design was to map out the flow of data within the system. A sequence diagram is great at showing the interaction between the user and the system and mapping out how the data flows between them. I decided to create several different sequence diagrams to show the flow between users and important components in the system.

## 5.3 Prototypes

The initial wireframe design was basic, and its entire purpose was to map out the features that I proposed in the use case descriptions. The web application is more than just the reservation system, but I am going to focus strictly on the wireframe design of the booking system and the admin dashboard rather than the other static pages present on the website (Menu, contact, home page etc.). The designs shown in [Appendix \*] show the initial low-fidelity wireframe that I created. I then showed the client to see if he was happy with how I mapped out the requirements and then I began designing the high-fidelity prototype.

The second prototype looked specifically at how the booking system would look and the different pages of the dashboard that the admin will be interacting with. I created this prototype based on the feedback from the client. This allowed me to focus on elements that I know that the client deemed necessary.

## 5.4 Entity Relationship

Table

Description automatically generated

Figure 1 Entity Relationship Diagram

The entity-relationship diagram for my system was simple. I decided to split up the people reserving a table into “users” and “guests”. I did this so the customer didn’t feel pressured to sign up and have their details stored for emails and promotions. Once the user goes through the booking process, the booking information will be stored in the “bookings” table and will have an FK attached to it depending on whether a user booked it or a guest.

The restaurant table isn’t specifically connected to the other tables and will be used more as a configuration table.

## 6. Project Structure

The project structure was split up into two section folders. The first folder held all the client-side coding (React) which was labelled “front-end”, and the other folder was all the server-side coding (PHP) labelled “back-end”.

Text

Description automatically generated with low confidenceThe front end was split into 4 main folders. The first folder is the “asset” folder, which contains all the images and fonts used within the project. The next folder is the “component” folder. This folder contains the bulk of the logic and coding for the booking form. I like to keep all the reusable functions in the components folder as it logically makes the most sense and keeps the structure clean. The next folder is the “containers'' folder, which is essentially most of the HTML and client-side components that the user visibly sees on the web pages. The containers can be reused across the website. Finally, we have the pages folder. The pages folder contains each page on the website, and it initialises all the components and containers created in the other folders. Overall, I feel like my client structure is very intuitive and clean.A picture containing calendar

Description automatically generated

I maintained a similar structure for my backend files. I took an object-oriented approach for my server code and used classes throughout. In the config folder, I created an autoloader magic function that uses an “include\_once” function for all class names in my directory. I then created an exception handler and error handler. The exception handler would output errors depending on whether the request was HTML or JSON. I then create the actual config file which initialises the other functions and defines important information such as basepath and database.

Implementation

# Introduction

Now that the design of both the server and client has been produced, I am going to discuss how these designs came to fruition. The focus of the following section is to document the progress that I made throughout the application development and all the issues I encountered. There were several times throughout the project in which I took the development down a different route, and I am going to discuss these decisions and how I came up with specific solutions.

The chapter will be broken down into the creation of the React Application, and the creation of the API. To a user that isn’t authenticated, the website only consists of the booking form and other static pages, but if they are authenticated then they have access to the user dashboard in the navigation bar. If the user is authenticated as an admin, then they will have further access to the admin dashboard. The booking form, user dashboard and admin dashboard all interact with the API created on the server-side, but the user never directly uses any information that is generated from the API – the React App handles it all and displays it for the user.

I will be providing snippets of code to help me explain important functions and logic that I used within the development process. For the full view of the code please see Appendix \*. You can also see the final design of each important page in Appendix \*.

# API

## Database

As mentioned in the Tools and Techniques chapters, I decided to opt for a MySQL database when creating my API. The database suited my requirements and could be run directly from a local server such as XAMPP. XAMPP is perfect for the stack I am using, as it reads scripts written PHP and provides the user with their own MariaDB database. It was installed quickly and seamlessly and has the advantage of being able to transition from a local server to a live server as most web server deployments use the same components as XAMPP [10]. Only four tables were needed for my application, shown in my entity-relationship diagram (Appendix \*).

The database was connected to the API via a Database class which integrates PDO (PHP Data Objects). The database connection function is set as private and can be initialised throughout the gateways using the public functions shown in Figure \* below.

Text

Description automatically generated

I decided to opt for this type of database access layer as it provides a fast and consistent interface for accessing databases in PHP [11]. It also helps separate database related operations from the rest of the code and allows the user to easily perform CRUD (Create, Read, Update, Delete) operations.

### Event scheduler

The event scheduler that is integrated within a MariaDB database solved several issues that I was having with the overall system design. In the event of bookings being added to the database and becoming outdated when the bookingDate has occurred, I wanted these booking entries to be deleted from the database as they are no longer relevant. To do this, I used the event scheduler to set an event that occurs every day after the closing time of the restaurant, which wipes all booking entries that are outdated.

To improve the usage of the event scheduler, I should have changed the active column of the booking entry to “0” rather than “1”. I don’t think deleting the entries is the correct way to go about it as I believe past booking data could be useful to the admin in a CRM context. The reason why I didn’t use the event scheduler in this way is because of the solution below.

### Prevent multiple User Bookings

A key problem that I needed to prevent was the idea of a user spam booking into the restaurant to collect loyalty points. There is a multitude of different ways I could solve this, but I decided to solve this via the database. Every booking that is entered into the database is assigned a userID as a foreign key so that the tables can be joined together, and the user associated with the booking can be displayed. I added a unique key to the userID so that only one booking can be made by the user with that userID. This solved the issue of preventing multiple bookings under one user but created the issue of the booking needing to be deleted from the database until that user can make another booking. The event scheduler solved this issue, but this meant that I could not utilise the “active” column in the bookings table as there wasn’t a reason to use it if the booking was just going to be deleted after the bookingDate occurred.

## Gateways

The next layer of the API is the gateway classes. As previously mentioned, PDO helps separate the code into database related operations and that’s exactly what the gateways deal with. The gateway classes are the only time the server interacts with the database.

Text

Description automatically generated

Figure 2 Abstract Gateway Class

The gateways have an abstract class that they all inherit. The Gateway abstract class provides the getters and setters that interact with the database class. To provide access to our private database variable, I have created a setter that sets it to the Database class and a getter that returns it. These functions are used throughout the other gateway classes to access the database. Getters and setters are used for several different reasons: Scalability, Debugging and Cleanliness. You can easily refractor a getter rather than scour code for different variable assignments. You can also put breakpoints between all the functions to determine exactly where bugs are occurring in the code. They are also much cleaner and whenever you need to make changes, you only need to edit the getter and setter.

Text

Description automatically generated

The above code shows one of the Gateway classes inheriting the abstract class and utilising it within its code. This specific gateway deals only with queries to the database that relate to booking a table slot in the restaurant. The constructor is the first thing that’s called when the class is initialised. It sets the database connection found in the abstract class. The SQL statement is then made using a prepared statement; a prepared statement is when certain variables are left unspecified, then the database parses, compiles and performs query optimization on the template and stores the result with execution [17]. The execution takes place later when the application binds the parameter values to the database request when it is called.

In the $result variable, you can see that the “getDatabase” getter is called which allows the code to interact with the database class and execute the SQL statement. Overall, I think this code provides a very secure solution to the API and reduces the risk of SQL injection and reduces parsing time as the preparation for queries is done only once.

A screenshot of a computer

Description automatically generated with medium confidence

One function of note that pertains to the gateway classes is the “checkBookingDate” found in the reservation gateway. One of the major hurdles in designing the booking logic was finding a way to prevent overbooking at a specific timeslot. The above query helped me solve this issue by searching the database for all time slots (booking starts) on a specific day and grouping up the party sizes on each of these time slots. This information can then be sent to the client and the time slot can be disabled if the overall party size is higher than the max occupancy of the restaurant. More of this issue will be discussed in future sections regarding the front end.

## Requests and Responses

Text

Description automatically generated

The next topic of discussion is the type of request from the client and the type of data format that should be sent back. Above is the Request class, which initially parses the endpoint and stores it in a $path variable to be used. It also decides whether the data is requested via a POST or a GET request by looking at its parameters.

A screenshot of a computer

Description automatically generated with medium confidence

In the index file, the response is then decided by using the getPath function from the request class; if it has “API” in the endpoint URL then the response is going to be formatted as JSON, whereas if it doesn’t then it will be sent back as HTML. My application doesn’t really utilise HTML data so if it is an HTML request it will simply send back an error page.

Text

Description automatically generated

The final thing I find worthy of mentioning about the requests and responses classes is the “JSONResponse” class. All the requests to the API are going to be JSON requests, as that’s the format of information that the client is going to deal with. When the API realises that the request is for JSON, it calls on the class to handle it and set the data appropriately. The headers for the request are first set. Headers are important as they allow HTTP requests to pass between one website domain to another. Because I am only working in localhost, I am integrating a laxer approach using the wild card “Access-Control-Allow-Origin: \*”, which allows most origins to access the API and is a workaround for CORS restraints [18]. If I was going to push this code to a live environment, I would make it more robust by adding credentials to the header and making sure only limited origins can access it.

This class is also where HTTP response codes can be attached to the request if something goes wrong. If everything is fine, then the response will usually attach a “200” code which lets the user know that nothing went wrong. If something occurred, such as failed authenticated, this is where the API can attach a different HTTP status request to let them know there is an issue.

## Controllers

The next thing I would like to discuss is the way I implemented controllers into the API to handle data requests when the endpoints are called in the front-end. The controllers are like gateways in that they both have an abstract class that they inherit [Appendix \* Page \*]. When a controller is first called, it sets the gateway in which it will receive the information, set the type of request that is being made (e.g a “GET” or “POST”) and then will set the type of response it will send back.

Text

Description automatically generated

The main controller example I would like to discuss is the authenticate controller. This controller is called when the user tries to log into the web application. The gateway is set at the start, and then the request functions are called to receive the parameters that were posted from the client. If it is a POST request, and certain requirements are filled from the if statements (such as the “email” and “password” parameters not being empty) then functions are called from the gateways to interact with the database. This specific controller is interesting as it integrates something called a JSON Web Token.

Text

Description automatically generated

The JSON web token was the primary way I integrated authentication into the system. It is vital that certain parts of the website are blocked off to everyone other than the admin so that normal users do not have access to personal information and make requests to the server which could potentially delete bookings or users. The user makes a sign-in request then the API creates a JWT token using a secret key stored on the server-side. This returns the token back to the client application. The client application verifies on its own side whether it’s authentic and prevents the user from having to send their credentials every time they want to access a page.

Within the JWT token is a “payload”, which is data pertaining to the request that the user makes. In my project, I decided to return the user ID of the request, whether they were an admin and when the token expires. This information can then be decoded on the client-side and used across the website to provide admin and user privileges. If the JSON token is not added due to the user inputting incorrect details, the API will send back an “Unauthorized 401” response. If the request is a “GET” request, it will simply tell the user that the method is not allowed. This is unlikely to happen for a customer as they will not be accessing the API endpoints directly and will be handled by the client which will make it so it’s always a “POST” request.

Text

Description automatically generated

The JSON web token was then used in several different controllers to protect endpoints that should only be accessed by admins. An example of this is shown in figure \*. This code shows the process of decoding the web token for the “isAdmin” payload. If the payload contains verification that the user is an admin, then they will be able to request the information from the end point. If not, they will be met with an unauthorised response.

### Reservation Controller

A controller of note that I ran into significant issues with is the reservation controller. The reservation controller is called at the end of the booking form to submit the booking details. I initially only had one controller that handled both the guest reservations and the user reservations but kept having issues with storing the correct guestId and userId with the associated booking information. The controller had an if statement checking if the password variables were populated. If they were, then the API knew that it needed to insert a user into the database alongside the guest information. The issue was calling the addUser function after the addGuest function, as it was using the lastInsertId() of the addGuest function to store the id into the booking table. The reason this caused issues was because the initial design of my database had the user table as an addon of the guest table. The guest information was always created on a booking, and the user was only ever created if the customer decided to opt into it at the end of the booking process. Because of this, sometimes the guestID and userID were different, and the booking function would call lastInsertId() on a userId which was different to the guestId which in turn could potentially send back the wrong personal information.

To solve this issue, I decided to split the user and guest tables up so that the information was stored in both. I then created a separate controller for them which meant that the addReservation function could also grab the correct ID associated with the booking.

A screenshot of a computer

Description automatically generated with medium confidence

### Timeslot Controller

The front-end will make a call to the restaurant table to check when the restaurant is open and closed, and dynamically generate the booking slots depending on the “timeInterval”. For example, if the time interval is set to 30 minutes, then time slots will be generated every 30 minutes from the hours open to the hours closed. The max occupancy will be used to know the maximum amount of people allowed at a given time slot. An example of this would be if the max occupancy was set to 30, then no more than 30 people will be able to book at a specific timeslot on a given day. We can measure how many people have booked at a given time by using the “bookingStart” and “partySize” columns in the bookings table. If there were several bookings on a day with the same “bookingStart”, and the “partySize” of each booking exceeds the max occupancy when added together, then the time slot with the same booking start will be disabled in the booking process so that no more people can have a booking starting at the same time. The code and logic for the timeslot controller can be viewed in Appendix \*.

## Endpoints

Endpoints are the URL paths that the client calls to request data from the API. All my endpoints are written in the index file of the server as shown in [Appendix \* page \*]. The API is structured so that the first point of call is the index file, and the controller that handles the request is called depending on the URL. This is done through the usage of “case” and “switch”. A case is made that initialises the endpoint URL, and then the controller associated with that endpoint URL is created. Whenever that endpoint URL is called, that specific controller will deal with the request. I tried to set up my API so that the logic for each page is split up into its own controller so that unnecessary data is not leaked into other parts of the web application. All paths are rerouted to the index file through the use of a “.htaccess” file. A .htaccess file provides configuration changes on a per-directory basis [19]. In my case, I have made a “Rewrite Rule” which makes all requests to the server go through the index file first.

## Configuration and Error Handling

Text

Description automatically generated

The final thing worth mentioning about the API is the way I configured it and handled errors throughout. Above is the content of my config file, and the things worthy of note are the autoloader, the exception handlers, and the error handlers. It is also the place in which I store important information that can be called throughout the API, such as the SECRET\_KEY for the JWT token and the basepath in which will be added to the beginning of all API URLs.

Text

Description automatically generated

The autoloader function shown in figure \* is great for object-oriented applications as most developers use one source file per class definition. The autoloader removes one of the biggest annoyances of having to write a long list of ‘includes’ at the beginning of each script as it enables classes to be automatically loaded when they are called if they are not currently defined.

Text

Description automatically generated

Errors within programming are very frustrating, and it is important that the developer can pinpoint the issue within their application as quickly as possible. Within my API, I created several different exception handlers which deal with potential bugs and help the developer find the source of the issue as quickly as possible. Figure \* is a function that deals with server errors in JSON format. If the application is in development mode, then all bugs will be logged in the console for the developer to see and rectify. If it’s not in developer mode, then the errors will be logged in a .txt file so that the user cannot see the issues with the server. The format of the handler illustrates what the error is, the file that causes the issue and the exact line where the bug occurs. Exception handlers like this are vital for developers to use so that they are not spending a great bulk of time debugging code and trying to pinpoint exactly where the issues occur.

# React App

In the following section, I will be discussing how the React Application was created and how it utilises the API created in the previous section. The approach I took to creating the application was using functional components rather than making class components. Throughout every component in my application, you will see me creating state variables using Reacts’ built-in “useState” hook shown in figure \*. The first variable in the square brackets is where the data is stored, and the second is the function that is called to set the state of the variable. I will be using this a lot throughout my code.

Text

Description automatically generated

## Register

A major part of the system is how the user signs up and logs into the web application. It is important that the experience is as seamless, and that all information posted to the API is as secure as possible on the front end. The code for my login and signup process is in huge blocks due to tailwindCSS styling the elements within the class tags. Because of this, I will be showing only small snippets of code and explaining key parts.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 5 Beginning of the Sign Up form

Figure \* shows the beginning of the Sign-up form that the user sees when they first open the Sign Up modal. The form has a “handleSubmit” function assigned to it, so whenever the form is submitted this function will be called and all logic associated will be initialised. Important features of this form are the way the variables that will be posted to the API are set in the state using useState (like previously mentioned). The onChange function sets the variable of “name” by calling “setName” and then inputting the value of whatever gets typed into the input text area.

Shape

Description automatically generated

Text

Description automatically generated

Validation for the form played an important role when I designed it. I wanted to make sure that all information is valid before it gets posted to the API, so I included a thorough validation process to prevent this. As you can see in figure \*, when the user inputs a name that meets the validation requirements, they are alerted with a green tick. If they input incorrect details, they are met with a cross, and instructions will appear that tell them how to input correct details.

I accomplish this through the usage of Regular Expression (REGEX). REGEX is a sequence of characters that the user can use when inputting data. If they type something outside the boundaries of the regex pattern, then the user will be alerted.

A picture containing text

Description automatically generated

The useEffect hook plays an important role in validating the form. By using this hook, you tell React that your component needs to do something each time it gets rendered. In this case, every time the user types into the form, the validity of what is typed is checked against its REGEX.

Text

Description automatically generated

Once the user has inputted valid information into the Sign-up form, the handleSubmit function is called which makes a request to the API. Before it does this, it makes one final check to see if the name and password variables are correct – if not then an error is displayed.

The variables that are to be posted are then appended to form data to be sent as an API request. If the API request is successful, then the success variable is triggered to ”true” and the sign-up form content will be replaced with a success message.

Text

Description automatically generated with medium confidence

If the request to the server fails, such as the email they used already exists in the database, then an error message will appear at the top of the form alerting the user.

Graphical user interface, application

Description automatically generated

If everything goes smoothly and the user inputs all the correct details, then a success message will be triggered, and the user will be able to log in to the system once they have confirmed their email address.

## Login

Text

Description automatically generated

When the user signs up successfully, they can use the login form to log in with the details they have just created. The important code in the login component is the handleSubmit function. When the user details are authenticated in the backend, the server then sends back the JSON web token. The client then attaches the web token to local storage which can then be used throughout the rest of the web application.

## JSON Web Token

Text

Description automatically generated

Within the code shown in figure \*, you can see that the JWT token is initially stored in local storage once the user has gone through the login process. With the token, I can decode it and check whether it is still valid by using the expiration date in the payload. If the token is valid, I can then store the other items in the payload in variables. These variables can then be passed across the website using Reacts Context API, which allows you to share information with any component by wrapping it in a context.

## Booking Forms

When the user clicks the book table button on the home page, they will be redirected to a specific booking form depending on whether they are authenticated or not. If they are authenticated, they will be redirected to a booking form that will make a call to the User Reservation endpoint in the API. If they aren’t logged in, then they will make a call to the Guest Reservation API. I decided to split up the forms in this way as I was running into the issue of applying the wrong guestid and userid to the booking table when it was inserted as mentioned previously in the API section. These forms are displayed depending on whether the “authenticated” variable is flagged as true or false from the login process.

Text

Description automatically generated

An important section of code I would like to discuss is the GenerateTimeslots() function within the booking form. This code is responsible for dynamically displaying the timeslots for the restaurant by using the opening time, closing time and time interval in the restaurant table in the database. The first issue to solve was adding the time interval onto the opening hour until it reaches the closing hour. I did this by creating a function that converted the times from the database into minutes. Once the times were converted to minutes, the slots() function took the opening time and closing time and generated an array of objects using the time interval. With this new array of times, I then took the data sent from the API which indicated what timeslots exceeded the max occupancy and pushed these timeslots from the array. The array is then mapped across a series of buttons which will be displayed underneath the calendar.

Once all the booking data is collected, they are appended to form data and posted to the “UserReservation” (or guest) endpoint to be added to the booking table. Every time a booking is made, the loyalty points of the user are increased by “1” and can be viewed in the user dashboard.

### Calendar Plugin

A vital part of the booking form was the calendar component. The calendar component allowed the user to choose the date on which they would like to reserve a table. It was important that I chose a plugin that allowed the functionality I needed, such as providing a date value when a date is pressed.

I decided to opt for the “react-calendar” component which can be downloaded directly from Node Package Manager.

Text

Description automatically generated

This component worked great right out of the box and allowed me to get the value of the date pressed and added to a state variable. When a specific date is pressed by the user, the “onDayPress” function is called.

Text

Description automatically generated

This function converted the date into a format that can be read by MySQL, and then was appended to formData to be sent to the “checkTimeSlots” API endpoint. This endpoint sends back all the bookings that are overbooked on that date so that they can be removed from the timeslot selection.

## Admin Dashboard

The last part I implemented into the system was the admin functionality in the dashboard. If the user is authenticated as an Admin via JSON web token, they will be granted access to the admin dashboard. The admin dashboard consists of a bookings page, a customers’ page, a marketing page, and a settings page.

Text

Description automatically generated

### Booking

The way I displayed all the existing bookings on the bookings page was by mapping it through a child component. In the code shown above, I am mapping all the data received from the Admin Bookings API and passing it down to the child component. Figure \* shows how the data passed down to the child component can be displayed.

# Text Description automatically generated

### 

### Marketing

### Settings

## Features of note

# 7. Testing and Results

## 7.1 Functional Testing

I decided to do my functional testing using the SCRUM methodology. This meant that my software was consistently tested throughout the development cycle, paying specific attention to the complexity, quality, performance, and usability. I did this by testing the code by myself and through Unit Testing using a unit testing framework called PHPUnit.

### 7.1.2 Manual Testing

I conducted functional testing myself across the full project cycle. I would test my code for issues and log them using a test case. You can find the full log of manual testing in [Appendix \*]. Table 1 below shows the format of how I would test my code.

### 7.1.3 Unit Testing

## 7.2 Usability Testing

To test the usability of my system, I recruited 5 different candidates to test the customer side and I then asked the client to test the admin side. I will attach the template I used to conduct the usability testing for both customer and admin. Ethical consent was obtained from all candidates before testing was conducted and is attached in [Appendix \*]

### Customer Testing

Table

Description automatically generated

Figure 6 Customer Testing Plan

The table above shows the full customer testing plan that I conducted with the 5 different candidates. I tried to make sure that the testing plan covered the full extent of the web application

## Admin Testing

\*INSERT ADMIN TESTING PLAN \*

## 7.1.2 Vulnerability Testing

XSS Attack

SQL Injection Attack

URL Attack

## 7.1.3 Cross-Browser Testing

Firefox

Chrome

Edge

Evaluation and Conclusion

# 8. Evaluation

## 8.1 Introduction

Throughout this section, I will evaluate the system I have created through testing and feedback from users/the client. I will also discuss the processes I used within the project development and how they could be improved if I had to redo the whole thing again.

### 8.1.2 Build Quality

The build quality of my system is going to be based on the required criteria that I set out in the initial phase of the project and how well it meets it. I am also going to discuss points in my code which I consider especially high or low quality.

### 8.1.3 Testing

The manual testing process was very insightful and allowed me to identify a few issues which I then rectified. I’ve never conducted testing in previous projects that I’ve worked on, so it was overall a great way to get further experience in making my code more robust. In hindsight, I would have spent more time learning ways to test my code and making it even more solid and secure. This was an issue due to my lack of experience, so will be improved upon next time I set myself on a big task.

### 8.1.4 Requirements

### 8.1.5 Client & User Feedback

### 8.1.6 Literature Evaluation

The literature that I conducted at the start of the project was very useful and I am going to discuss below how the different topics played a role in the final prototype.

#### 8.1.6.1 RESTful API

The literature I gathered regarding RESTful API played a vital role in the way I developed my software architecture. I created my server using controllers and gateways and I dealt with HTTP requests depending on whether they were POST/GET/DELETE. My software architecture wasn’t a direct representation of a RESTful API, as it relies heavily on the client. A RESTful API is usually completely independent, and the client doesn’t need to know anything about the structure of the API. Despite this, I did include lots of relevant points of REST such as the different CRUD operations and basing the authentication process on a JWT token.

If I was to re-do my project, I would put more emphasis on the server rather than trying to deal with most of the data on the client side. My go-to method for making API calls was creating a select query which returned all the data from an endpoint and sent it to the client. I think, in this project context, it worked quite well because I usually wanted to display all the information, but there are some circumstances where I could have attached the “id” of a user or booking to just return that specific parameter. I spent most of my focus on the client and used the server as a “means to an end” to return the data I required, but I should have set up more specific endpoints so that the admin could navigate through information purely with API requests.

#### 8.1.6.2 JSON Web Token

Overall, I think my website utilised the JSON web token in a secure way. It can be argued that storing the token in local storage isn’t the safest way and that it should be stored in an HTTP-only cookie and attached to the user in the database using a refresh token. The refresh token has its own endpoint in the API and is called whenever the user refreshes the page, checking to see if the token is still valid in the database and renewing it if it is. My way of storing it in local storage makes it more prone to XSS attacks, but because I’m not dealing with sensitive data and the project is small, it’s enough protection to ward away attackers.

#### 8.1.6.3 CRM System

### 8.1.7 Tools and Techniques

In this section I am going to discuss all the tools and techniques I proposed at the start of the project and evaluate how I used them and if they were ideal. I will also mention any other tools I used that I may not have thought about using at the start of development.

#### 8.1.7.1 MYSQL Database

MySQL was a good database technology to use within my project. It had functionality that fit well into my project which allowed me to achieve my goals. One of these was how you could apply “events” to the data that is stored within the database. I applied a unique key to the ‘userid’ of each booking that is created, which meant that a user cannot create more than one booking. Using the events functionality of MySQL, I could set an event that is triggered once the booking date is over which deletes the entry in the database. This is a great way to make sure that a user can’t spam booking slots into the database so that they can gain more “loyalty points” for rewards.

If I started the application from scratch, I believe MongoDB would have been a better database to use as it represents the data in JSON objects and is much easier to build applications with its schema-less design.

#### 8.1.7.2 PHP

PHP was used to create the server architecture for the booking system. PHP is a great language to use as I find its syntax very straightforward, and its command functions are simple and easy to implement. Throughout the project cycle, I ensured that I created class components for all the elements of code that I created. This meant that all the code I used was reusable and could be reimplemented across the project without having to rewrite the code.

If I redid the booking system, I would switch the backend architecture from PHP to Node.js. I believe the MERN stack would have allowed me to create the system at a much faster pace as it is more intuitive using Node with React as they are both built upon JavaScript. I also really like how node.js has its own package manager, which essentially means you can install pre-existing packages created by other people straight from the command line. This would have made sections of my backend much easier to implement, and I would have been able to add extra functionality with ease.

#### 8.1.7.3 React

React was a great software to use for the client-side of the project. As previously mentioned, you can install pre-existing packages and components directly into react which makes life much easier and allows you to integrate pre-tested solutions created by other people. An example of this is how I used a “React Calendar” component, which added calendar functionality to my application with ease. It provided me with functions that allowed me to easily grab the date that the user chooses and place it into a variable for submission into the database.

The main issue with react was the decision to initially use Class-based components. Halfway through the development cycle, I realised that using functional components with hooks is much more fluid and modern. It is concluded that developers have entirely replaced class components with functions and hooks [6]. I decided to convert all my previous components which were class-based to functional components instead. This ended up costing me a lot of time and ate into the time I set aside for other functionalities of the application.

#### 8.1.7.4 Visual Studio Code

Visual Studio Code is a code editor that allows the user to be very efficient when writing code. It has a vast extension library that allows the user to add quality of life features to their arsenal when coding. An example of this is the ES7+ Snippets extension. This allowed me to type shortcuts such as “rafce”, which in turn would generate a full template of a functional component.

#### 8.1.7.5 TailwindCSS

I decided to completely replace conventional CSS with a framework called TailwindCSS. Tailwind is a utility-first CSS framework that emphasises responsive design and was used to apply a modern, sleek look to the application.

Tailwind helped speed up my project by a great margin. The concept of applying CSS styling directly to the HTML tags meant that I didn’t have to create multiple cascading style sheets for all the different elements on my page. It also sped up the way I would usually do responsive design. You can apply “breakpoints” to every style you add, so if you check and see that the design isn’t responsive on mobile, then you can add a “breakpoint” which can only apply the style to certain device widths. This works very similarly to media queries but is much faster and you don’t have to think about all the different device widths.

Another great feature of tailwind is that they have their own components page on their website. If I was ever struggling to come up with a good design for an element on my website, I could take influence from their components page and use an existing style that has been created specifically for a modern-looking website. This saved me the hassle of messing around with different design concepts and using something that has already been created by a professional.

#### 8.1.7.6 XAMPP

XAMPP was invaluable to my project and provided me with the ability to host my server-side code on localhost using APACHE. It also provided me with my Database as well.

# 9. Process Evaluation

This section will evaluate the way I conducted the project and the methodologies I used

## Project Life Cycle

Throughout the project lifecycle, I implemented a project framework called AGILE. AGILE is a methodology that prioritises test-driven development, coding in small sprints, and having consistent contact with the client and users.

## Time Management

My time was split up at the start of the project using a GANTT chart. A GANTT chart is a type of bar chart which helps illustrate a project schedule and map out tasks that need to be completed at certain intervals.

## Skills

I developed lots of technical and soft skills during the project development. The primary skill I learned was API development and how the backend and front end communicate with each other through API calls.

## Objectives

I identified a multitude of objectives at the start of the project that I wished to achieve by the final design of my web application. These objectives are shown below:

* Create a literature and technology review:
  + Look at existing booking/reservation systems and identify their good and bad features. Take ideas that could be implemented in my design.
  + Examine existing plugins used for calendars that can be integrated into the reservation system.
  + Find existing CRM systems used in restaurants and find ways I can use user data created from the reservation system.
  + Find academic articles on CRM systems and table reservation systems.
  + UCD design for restaurant website and booking systems.
  + Look at privacy and data protection issues regarding storing user data.
* Establish and prioritize the requirements of the product:
  + Find out the client’s requirements
  + Create a storyboard and initial vision of the application
  + Find target audience(s)
  + Persona(s) and Scenarios
  + User requirements document
* Create designs for the web application:
  + Initial design storyboard and wireframes.
  + Low & High-fidelity prototypes
  + Use case diagrams
  + Class diagrams
  + Entity-Relationship diagrams
* Create the product based on the design specification.
* Test the product:
  + Functional testing
  + Usability testing (Users and admin)
* Make changes to the application based on the testing process.
* Evaluate the product.
* Evaluate the processes and my performance

If I scrutinize each objective and compare it to what is present in my final prototype, I believe I successfully achieved most, if not all, of my objectives.

For the literature review, I identified key areas that I needed to study so that I had a good understanding of what was required in a booking and CRM system. I then took this knowledge and applied it to my designs and then implemented it into my code. An example of this is when I reviewed literature based on RESTful APIs and used my understanding to create one to the best of my ability. I also applied UCD design throughout my project so that the booking process would appeal to the user. Finally, I spent a considerable amount of time making my web application as secure as possible – thanks to what was learnt in the literature review.

I also created every artefact that was required for both the requirements stage and the design stage. These artefacts culminated to make the actual development process much easier. The user requirements document allowed me to pinpoint every feature that was needed in the design stage, and the design documents allowed me to code my final product much more fluidly as there was more structure to guide me through the process.

I also believed I tested my product to the best of my ability. It was hard to conduct thorough testing of my product as it isn’t something I’ve ever done before. I added all the testing procedures and plans that I thought would best suit my application and summarised them as best I could. I believe the testing process was the weakest part of my project and I would spend more time learning how to thoroughly test my code before I embark on another large-scale project.

## Learning Process

## Legal, Social, Professional and Ethical Issues

# 10. Conclusion

In this final section, I will take a brief look at the project and decide whether I have met the overall aims I set out at the beginning. I will then discuss any future work that could be added if I continued and anything that I would change if I created the project from scratch.

## 9.1 Aims

When creating my terms of reference for the project, two aims were established which outlined what I wanted to achieve. The two aims were:

* To investigate how a table reservation system is created and how the data can be used in Customer Relationship Management.
* To build a web application for a pizza restaurant that integrates a table reservation system and a CRM for the restaurant owner.

The first of these aims was primarily research into how my web application can be created and how the data it produces can be utilised to increase relations between restaurants and consumers. I think I did this successfully within the research stage by investigating existing reservation systems and seeing how they use their data. I also had a deep dive into the limited academic articles based on CRM systems and reservation systems.

The second aim was to develop a booking system, implementing all the research I gathered from the first aim to create an application for my client. This was achieved through a thorough requirement capture plan, an extensive design stage and then implementing it all with tools and techniques that I believe would work best. I think the resulting product met all the criteria that I set out in the initial stage and met all the requirements of the client. The overall feedback from the client was positive and they noted that they would be happy to utilise the system if further development was made.

## 9.2 Further Work

I believe I met the criteria for my project to a good standard, but there are several things that I would have changed and improved upon in the final product, and I would have changed a few things within the development cycle to make it a much quicker and smoother process.

I would have liked to spend a lot more time developing the CRM side of the system. I spent most of the project lifecycle developing the functionality of the booking system and creating a dashboard for the admin to change existing bookings. CRM systems have developed a lot in the past years and there is a lot more I could have done with the user information to create a more robust customer relationship to entice users to return to the restaurant.

## 9.4 Summary

I believe the overall project was a success and that all aims and requirements were fulfilled for both the client and what I set out for myself. The web application that I produced could potentially be applied to the real world if further development was made to make it more robust.

REFERENCES

[1] Flynn and Buchan, ‘A REFEREED PUBLICATION OF THE AMERICAN SOCIETY OF BUSINESS AND BEHAVIORAL SCIENCES’.

[2] Kimes, ‘How Restaurant Customers View Online Reservations’.

<https://dspace.mit.edu/bitstream/handle/1721.1/86490/46888364-MIT.pdf?sequence=2>

[3] ‘What Is CRM? The Beginner’s Guide - Keap’. Accessed 1 February 2022. <https://keap.com/product/what-is-crm>.

[2] OpenTable. ‘About Us’. Accessed 7 February 2022. <http://www.opentable.com/about/>.

[4] Hussain, Azham, and Emmanuel O. C. Mkpojiogu. ‘Requirements: Towards an Understanding on Why Software Projects Fail’. *AIP Conference Proceedings* 1761, no. 1 (12 August 2016): 020046. <https://doi.org/10.1063/1.4960886>.

[5] shelleydoll. ‘Agile Programming Works for the Solo Developer’. TechRepublic, 5 August 2002. <https://www.techrepublic.com/article/agile-programming-works-for-the-solo-developer/>.

[6] Luojus, ‘Usability and Adaptation of React Hooks’.

[7] Haekal and Eliyani, ‘Token-Based Authentication Using JSON Web Token on SIKASIR RESTful Web Service’.

[8] Hyun and Perdue, ‘Understanding the Dimensions of Customer Relationships in the Hotel and Restaurant Industries’.

[9] Daud and Aziz, ‘RESTAURANT RESERVATION SYSTEM USING ELECTRONIC CUSTOMER RELATIONSHIP MANAGEMENT’.

[10] Reveall.co. 2022. *A Guide to User-Centered Design | Reveall*. [online] Available at: <<https://www.reveall.co/guides/user-centered-design>>

[11] Vredenburg, K., Mao, J.-Y., Smith, P.W. and Carey, T. (2002). A survey of user-centered design practice. *Proceedings of the SIGCHI conference on Human factors in computing systems Changing our world, changing ourselves - CHI ’02*.

‌[12] ‘Cross Site Scripting (XSS) Software Attack | OWASP Foundation’.

[13] Keracheva, ‘URL Manipulation Attacks’.

[14] Juviler, ‘What Is a Modal and When Should I Use One?’

[15] www.redhat.com. (n.d.). *What is a REST API?* [online] Available at: <https://www.redhat.com/en/topics/api/what-is-a-rest-api#rest>.

[16] ‘XAMPP’. https://en.wikipedia.org/w/index.php?title=XAMPP&oldid=1085323551

‌[17] ‘PHP MySQL Prepared Statements’. https://www.w3schools.com/php/php\_mysql\_prepared\_statements.asp

[18] ‘CORS and the Access-Control-Allow-Origin Response Header | Web Security Academy’.

[19] ‘Apache HTTP Server Tutorial: .Htaccess Files - Apache HTTP Server Version 2.4’.

[20] Lorio Purnomo, ‘Customer Relationship Management (CRM) Analysis and Design to Provide Customer Service in The Culinary Field (Case Study Restaurant XYZ)’.

[21] Kumar, ‘Customer Relationship Management’.

[22] ‘Why Department Store Retailers Need to Offer Appointments | Qudini’.