

**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 19073 words.

(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:

A handwritten signature in black ink, consisting of a stylized 'S' followed by a horizontal line.

ACKNOWLEDGEMENTS

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

Table of Contents

Abstract.....	8
1. Introduction	8
1.1 Aims of Project	8
1.2 Project Objectives	8
1.3 Project Overview.....	9
1.4 Project Context and Justification	9
1.5 Summary of Tools and Techniques	9
2. Literature Review	10
2.1 Introduction	10
2.2 Online Reservation Systems.....	10
2.3 The effects of COVID-19.....	11
2.4 CRM Systems.....	11
2.5 UCD Approach.....	12
2.6 RESTful API	13
2.6.1 JSON Web Token	14
2.7 Security Risks.....	15
3. Requirements Capture.....	17
3.1 Introduction	17
3.2 Capture Process	17
3.3 Competitor Review	17
3.3.1 OpenTable.....	17
3.3.2 BookingNinja	18
2.3.3 Most important findings	18
3.4 Requirements for Customers.....	18
3.5 Requirements for Admin.....	19
3.6 User Stories	19
4. Tools and Techniques.....	19
4.1 Introduction	19
4.2 Adobe XD	19
4.3 Database	19
4.4 Programming Languages.....	20
4.4.1 PHP	20
4.4.2 ReactJS	20
4.5 Project Framework.....	20

Summary	20
Introduction	21
5. Design.....	21
5.1 Design Specification	21
5.2 Use case Descriptions and Diagrams	21
5.3 Sequence Diagram	21
5.4 Prototypes.....	22
5.5 Entity Relationship	22
5.6. Project Structure	23
6. Introduction	23
7. API	24
7.1 Database	24
7.2 Event scheduler.....	25
7.3 Prevent multiple User Bookings.....	25
7.4 Gateways.....	25
7.5 Requests and Responses.....	28
7.6 Controllers.....	29
7.7 Reservation Controller	31
7.8 Timeslot Controller	32
7.9 Endpoints	32
8. PHPMailer	32
8.1 Configuration and Error Handling.....	33
9. React App	34
9.1 Register	35
9.2 Login.....	38
9.3 JSON Web Token.....	38
9.4 Booking Forms	39
9.5 Calendar Plugin	40
9.6 Admin Dashboard	41
9.6.1 Booking	41
9.6.2. Marketing.....	41
9.7 Modals.....	41
10. Testing and Results	42
10.1 Functional Testing.....	42
10.1.2 Manual Testing.....	42
10.1.3 PHPUnit	43

10.2 Usability Testing	43
10.2.1 Customer Testing	44
10.2.2 Admin Testing	45
10.3 Vulnerability Testing	46
10.3.1 XSS Attack	46
10.3.2 XSS Scanner	46
10.3.3 ZAP	47
10.3.4 Self-Test	47
10.3.5 SQL Injection Attack	47
10.3.5.1 User Reservation Endpoint	47
10.3.5.2 Self-Test	48
10.3.6 URL Attack	49
11 Evaluation	49
11.1 Introduction	49
11.2 Build Quality	49
11.3 Testing	49
11.4 Client & User Feedback	50
11.5 Literature Evaluation	50
11.6 Tools and Techniques	51
12. Process Evaluation	53
12.1 Project Life Cycle	53
12.2 Time Management	53
12.3 Objectives	53
12.4 Learning Process	55
12.4.1 Skills	55
12.4.2 Lessons Learnt	55
12.5 Social & Ethical Issues	55
12.5.1 Social	55
12.5.2 Ethical Issues	56
13. Conclusion	56
13.1 Aims	56
13.2 Further Work	56
13.3 Summary	57

Abstract

The goal of this project is to create a web application for a pizza restaurant that serves as a table reservation system, as well as an admin dashboard to manage customers and bookings. This project was created because there has been an increase in customers using online booking systems since the COVID pandemic, and it is an excellent way to collect customer data for Customer Relationship Management. I discovered that the admin could use the customer data obtained from user signups and bookings to strengthen relationships. A loyalty system that can be exchanged for discounts is one example of this. I also added a feature that allows the restaurant owner to leave comments for customers, so that when they make a new booking, their booking slot will be flagged with a comment to prompt the restaurant owner to provide special requirements for them.

I also made certain that all bookings are labelled as either a guest booking or a user booking. 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.

1. Introduction

1.1 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.

1.2 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 an in-depth investigation into how a restaurant table reservation system is created and how it can be used for a Customer Relationship Management system. To support this, a working reservation system in the form of a web application has been developed for a pizza restaurant. The customer will be presented with a fully functional dynamic web application built with React, which will then redirect them to the reservation system, where they can book a table at the pizza restaurant. The customer will have the option of booking the table as a guest or creating a user account, which will allow them to collect loyalty points and receive special promotions. The website administrator will have access to a dashboard that will provide them with an overview of all bookings made on the website as well as the ability to cancel and create bookings. There will be a slew of other features that will serve as a Customer Relationship Management (CRM) system, such as the ability to send marketing emails to all registered users and apply "comments" to customers, allowing them to personalise the experience for each guest.

1.4 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. 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 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.

1.5 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 go over relevant literature to booking and reservation systems, as well as how user data can be used to build a Customer Relationship System to improve customer relations and increase repeat customers. This will aid in understanding the difficulties associated with restaurant reservation systems and the best approach to integrating one. The majority of the review will focus on how REST APIs are created, as well as other important design and security features to include in web applications.

2.2 Online Reservation Systems

When it comes to scheduling table slots within restaurants, reservation systems have become the norm in modern times. It has been observed that the technologically savvy Generation X and Millennials expect easy and seamless access to restaurant services via their mobile devices [1]. According to Flynn and Buchan's report, the best practices for Netwaiter (a well-known restaurant marketing platform) to attract local customers are to offer online reservations and have a mobile-friendly website [1]. Along with this, it states that you should have a static web page for your web application's menu. All of these practices will be incorporated into my design.

Distribution channel	Description	Hours of Operation	Reliability/ Quality	Cost	Marketing Opportunities	Personal Connection
Telephone	Non-dedicated telephone agent	Limited to opening hours	Inconsistent	Fairly low	Some	Depends
	Dedicated telephone agent	Limited to opening hours	Consistent	Medium	Some	Strong
	Call center	Longer hours	Relatively consistent	High	Medium	Reasonably strong if know restaurant
Online	Restaurant website	Constant	Very consistent	Fairly low	Some	Some connection
	Third-party website	Constant	Very consistent	Low	Strong	None

Figure 1 Comparison of Reservation Methods

A study was conducted in a Cornell University Hospitality report to gather customer opinions on online reservations versus phone call reservations [2]. There was evidence that both methods had advantages and disadvantages. One reason why online reservation systems are useful is that their hours of operation are consistent. This means that, unlike telephone reservations, the customer can reserve a table outside of operating hours. In addition, it is noted that it is much more reliable and saves money because no one needs to man the landline to answer calls. In terms of disadvantages, it is important to note that online reservations have the potential to reduce personal contact with customers – especially when third-party web applications are used. As a result, the user must have access to both channels of communication. It is also more personalised if the customer can book directly from the restaurant's website rather than through a third party such as OpenTable, which is exactly how I intend to implement 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.

Customer acquisition is defined as the portion of the customer relationship that begins with her/his first encounter with the business and continues through the first transaction until the first repeat purchase [36].

Customer acquisition research has generally concentrated on the effect of marketing variables (e.g., price, promotion, and discounts) on the attraction of new customers. Due to the expanding popularity of the Internet, hospitality businesses have gained additional ways to communicate with their clients outside of the service experience (e.g., reading newsletters, writing reviews, joining a Facebook community, or blogging) [37]. Customers can interact easily with other consumers outside of actual service consumption thanks to the increasing use of smartphones, and the emergence of online social media allowing firms to encourage their customers to become effective brand advocates [38]. Because of this, it is much easier for consumers to expose their brand to the public eye if they utilise these media channels effectively.

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.

According to a survey that asked questions about UCD methods and their effectiveness [11], iterative design, usability evaluation, task analysis, informal expert review, and finally, field studies had the greatest impact in practice. It was also noted that "believers" placed a high value on user requirements. It was ranked quite low due to the high cost of most user requirement processes in a large-scale context. The user requirement methodology I will incorporate will be on a small scale, but it should be adequate for the needs of the projects. The final thing I found interesting about this study was that heuristic evaluation was the most common UCD practice because it was simple to implement and cost less.

Heuristic evaluation [42] is a usability engineering method for identifying usability issues in a user interface design so that they can be addressed as part of an iterative design process. It is usually done with a group of people and they examine the interface, trying to match its compliance with recognized "heuristics". I'd like to target a few heuristic principles in my product. The first component is user control and freedom. When a user performs a task that may have long-term consequences, such as deleting a booking, I want them to be able to "exit." Another issue I'd like to address is error prevention [43]. I want the user to be able to easily recover from mistakes, such as mis clicking on a certain element in the booking form. Finally, the last heuristic I'll address is the concept of recovering from errors. I will not send them "error codes," but will instead translate these error codes into something that the user can understand.

Following the development of my web application, I will need to test and evaluate it. In a study comparing user testing and heuristic evaluation on commercial websites [44], it was discovered that user testing discovered 39% of usability problems, while heuristic evaluation discovered 40 per cent that was unique to heuristics. It was concluded that both types of evaluation complement each other as they solve different usability problems that may arise in a web application. I will definitely be integrating user testing within my project cycle, but the heuristic evaluation is much harder to achieve as it's more intense and will need people that can recognise usability principles.

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

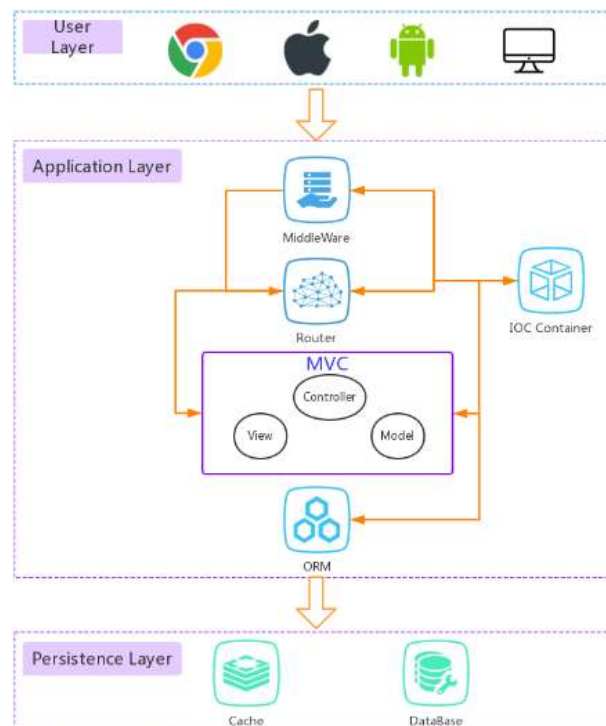


Figure 2 RESTful API architecture

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]. Endpoints make up RESTful APIs, also known as web APIs. Every endpoint is a physical representation of a business process' capabilities. These APIs can be accessed using HTTP verbs such as GET, POST, PUT, and DELETE (hypertext transfer protocol). RESTful APIs are accessed via URIs (Uniform Resource Identifiers).

Roy Fielding suggested representational state transfer (REST) in his PhD dissertation in 2000 [23]. This approach was designed to improve performance, scalability, simplicity, modifiability, communication visibility, portability, and dependability. Fielding developed six controls to achieve these properties: client and server architecture artefacts, statelessness, the ability to cache data, layered system, standard interface, and Code-on-demand. As a result, only the following HTTP verbs are used for RESTful API: DELETE, GET, POST, and PUT.

The GET method retrieves whatever information (in the form of an object) the Request-URI identifies. If the Request-URI (Uniform Resource Identifier) links to a data-producing process, the created data, not the source text of the process, should be sent as the entity in the response unless that text is the process's output. The POST method is used to provide a block of data to a data-handling process. The DELETE method requests that the server delete the resource identified by the Request URI. These 3 methods will be utilized throughout my project to request data from the API and send it back to the client.

On today's web, the design of URIs is formatted like so:

URI = scheme "://" authority "/" path ["?" query] ["#" fragment]

Several different rules should be applied to a URI to match the generic syntax identified in RFC3986 [25]. The first rule is that the slash separator must be used to indicate a hierarchical relationship between the resources. It is important that a trailing forward slash is not left at the end of an endpoint. Despite most web components identifying them the same, it's noted that every single character within the URI counts towards the resource's unique identity [24]. If the user is wanting to increase the readability of the URI, then a hyphen is preferred over an underscore. An underscore is not appropriate as they are often underlined due to being "clickable", which obscures the underscore. The final rule is that lowercase letters should be preferred as sometimes URIs can be case-sensitive.

A major perk of RESTful design is that it is stateless. To improve scale, REST Web service clients must send complete, independent requests. These requests include all data required to be fulfilled so that the components in the intermediary servers can forward, route, and load-balance requests without any state being held locally in between requests. A REST Web service application (or client) includes all the parameters, context, and data required by the server-side component to construct a response in the HTTP headers and body of a request [26].

Attached to each HTTP request and response is an HTTP header. HTTP headers allow the client and server to send extra data with an HTTP request or response. Headers are grouped according to their context: Request headers, Response headers, Representation headers and Payload headers [27]. The request headers contain information about the resource to be fetched, the response contains information about the response, such as its location, the representation headers contain information about the resource's body, and the payload contains information about payload data, such as the length of the content and the encoding used. Within my application, I will be using response headers to add response status codes to send back to the server. Status codes tell the user whether a specific HTTP request has been successful. The client can use this information to display error messages if something goes wrong, such as incorrect credentials, or if the request was successful.

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. JSON Web Tokens are stateless, so it is suitable to be implemented on a RESTful web application. Within this section, I am going to discuss the architecture for token-based authentication and the best standard practices for storing JWTs.

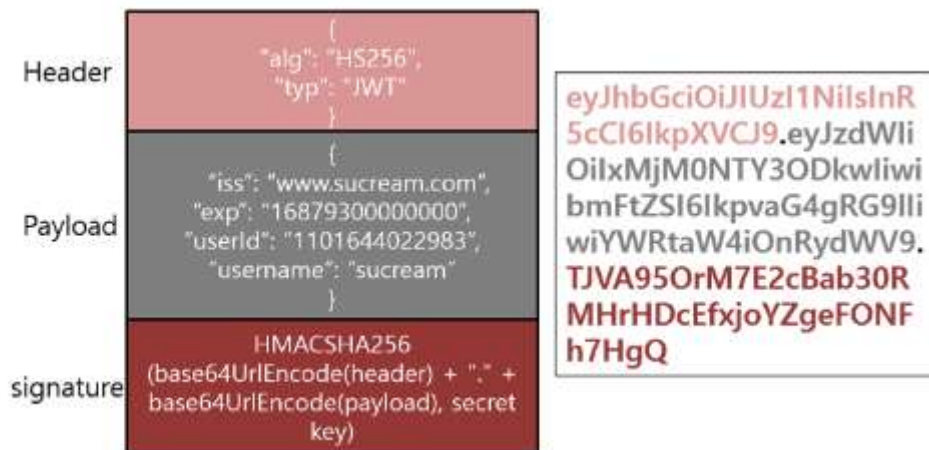


Figure 3 JSON Web Token Structure

A JWT is made up of three layers. The first layer is the header, which includes one for the token type and for the type of hashing algorithm. The hashing algorithm for a JWT is usually HMAC SHA256 or RSA [28]. The second part is the payload. This is the claim that includes information about an object (such as a user) as well as other additional information, such as when the token was issued and when it expires [29]. The final layer is the signature. The signature is applied using the hashing algorithm previously noted in the header using the secret key provided by the user on the server-side. The signature refers to content that is protected by a digital signature, with JSON being the data structure.

When a user successfully logs in, the server will issue a token, similar to how a password works. Additional server requests are made using the token. JWT's primary goal is to encrypt data exchanged between systems via client requests that must be accompanied by tokens. In a study that tests the speed and performance of the JSON web token [30], it is determined that when applied to a RESTful API application, the overall performance is good.

Storing the JWT on the client-side is a highly contentious issue with no clear solution. The token was stored in Local Storage in a study to compare the signed algorithms of JSON Web tokens [44]. The JWT can also be stored in a cookie, but cookie attributes can be accessed via CSRF (cross-site request forgery), which means hackers can use the JWT token to send bogus requests [45]. Overall, storing the token in either Local Storage or Cookie has drawbacks. I believe that storing it in local storage will keep the JWT secure enough for such a small-scale project, but other authentication methods may be required if I were to scale the project.

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.

2.7.1 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.

There are several different XSS attacks that could occur. The first type is usually called a Reflected XSS type [31]. Non-persistent attacks are delivered to victims in a variety of methods, including by e-mail or a link on another website. When a user is tricked into clicking on a malicious link that leads to the submission of a specially created form or simply browsing a malicious site, the attack is reflected in the user's browser, and the browser treats the answer as if it came from the user.

The next attack is a stored XSS attack [31]. This one is much more malicious as it can inject data directly into the sever to be permanently stored. The malicious data stored in the database can be served to the user despite it coming from a trusted source. The final XSS is a DOM-based attack which arises due to not handling the Document Object Model of an HTML file correctly. DOM attributes such as `document.referrer`, `document.url`, and `document.location` can be used to access and modify HTML entities on a page. To carry out such an attack, the attacker can change or access DOM properties. This is all done on the client-side side and the main culprit is JavaScript code. The main way I will suffer from an XSS attack in my React application will be through a DOM-based attack. To prevent this, I will need to make sure all data that flows through my application is validated and not mutate DOM directly [32].

2.7.2 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.

2.7.3 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.

2.7.4 Storing User Data

According to the 2021 Verizon Data Breach Investigations Report (DBIR), nearly a quarter (25 per cent) of the breaches involved hacking and were caused by basic web application attacks. It has more than doubled year over year, accounting for 22% of breaches due to social attacks and malware attacks, 17% owing to misconfigurations, 8% due to unauthorised access, and 4% due to physical attacks. Among the data that was comprised, over 44% were credentials and another 39% were personal [40]. Within my application, I will be storing credentials and personal information, so it is important that I know the best way to store user data.

A brute force attack is a prevalent sort of attack in data breaches. Hackers can write automated programmes that perform log-in attempts, entering innumerable character combinations until they

finally crack someone's password. This could be remedied by suspending login attempts after a certain number of attempts.

Client-side storage enables users to store various sorts of data on the client with the permission of the user and then retrieve them when needed. This enables users to save data for long-term storage, save sites or documents for offline access, maintain user-specific site settings, and more. Despite how useful this is, local storage is much more susceptible to attacks as any user with access to that browser can view the web storage data [41]. To avoid any personal data being leaked to a hacker through local storage, other methods must be implemented to avoid saving anything in local storage other than what is necessary.

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 began in California but has since expanded globally [2]. OpenTable serves as a discovery platform for restaurants as well as a booking system in the user's local area. The user simply selects the restaurant they wish to book and is directed to the restaurant's page, where they can view available times and dates. They can then reserve a table based on their requirements. This web application provided most of my inspiration for the booking process.

From the client's perspective, there are numerous features available to make reserving a table at their preferred restaurant a simple process. The first thing worth mentioning is how simple it is to find available tables and reserve a slot. The user must first select the size of their party, followed by the date on which they wish to book the table. This then displays the time availability based on both parameters. If no times are available for that specific slot, the system will suggest times within 2.5 hours of the initial time slot. In comparison to other systems, I believe this is a great process because it allows the user to immediately know whether there is any availability for their circumstances.

After selecting a time slot, the user is directed to a page where they can sign in with an account or proceed with the booking as a guest. Creating an account allows the user to accumulate loyalty points, which can be redeemed for discounts on future bookings. This adds value to the user's experience, making them more likely to create an account. The overall layout is simple and easy to read. It uses a simple colour scheme of red, white, and black to make all the text stand out. Any critical information can then be highlighted in a different colour to stand out from the rest of the page. One example is how they demonstrate the safety precaution methods in a darker grey colour.

3.3.2 BookingNinja

This system is most like 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.

Booking Ninja helped me establish more of my admin requirements, as it was one of the only applications that gave me a full view of the admin functionality and let me test it out. Booking ninja gave me the idea of adding time intervals between booking slots and being able to set the max occupancy of the restaurant.

2.3.3 Most important findings

The most important findings I gathered from the user's perspective were, first and foremost, the design's intuitiveness. To encourage the user to continue booking, the booking form should follow a logical flow. One example is how OpenTable displays the availability of table slots at the beginning of the booking process rather than at the end. A timer that can be set on bookings to alert the user how much time they have left to book is another great feature that could be included. This instils a sense of urgency in the user, encouraging them to book as soon as possible. The final feature that I believe would be beneficial to incorporate is the concept of having a wealth of restaurant information at their fingertips. (such as menus etc.)

From an admin standpoint, I learned to include a maximum occupancy for the restaurant to avoid overbookings. I also learned about a good CRM feature that involves attaching "tags" to customers so that the admin can be alerted as to whether they are a top spender or not and provide special treatment for them.

3.4 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.

I decided not to include every positive feature found in comparators because it would be excessive for the type of booking system I am developing (a small private one). But I was inspired by their designs and how they would approach making a reservation. I also wanted to include a loyalty

system for the added benefit of CRM. Every time the user makes a reservation at the restaurant, they will be able to accumulate loyalty points, which will increase customer retention from a CRM standpoint. When I proposed this idea in the questionnaire, I received a very positive response from the respondents, who expressed a desire to see it included in the final design.

Finally, I created personas and scenarios which created potential users and a scenario [Appendix D,E,F] that they might be involved in when using the application. I felt like this was a good way to try and establish extra requirements that I might not have thought about initially.

3.5 Requirements for Admin

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] and the Competitor and Comparator review.

The client interview was especially useful in garnering important requirements for the application. Because after all, it is their application that they will be using so they will know what they want.

3.6 User Stories

I opted to create a Trello board to map out user stories after I established the user criteria in the preceding chapters. A Trello board will be extremely useful in the development process since it will be used to choose user requirements to work on through "sprints." Sprints are a part of the AGILE SCRUM methodology, which will be discussed further in the following section. I made the decision to snap screenshots of my current Trello board progress, which can be seen in [Appendix E]. Each of the user stories displayed will be placed on a to-do list and then assigned to the next sprint to be completed within a specified time frame.

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 map out the design and layout of my application. I will be using Adobe XD to create the low fidelity wireframe of my reservation system.

4.3 Database

I have opted to use MySQL to store the data for the reservation system MySQL is a well-known open source database management system that is widely used in web applications due to its speed, flexibility, and dependability. SQL, or Structured Query Language, is used by MySQL to access and process data stored in databases.

There are numerous databases available that would have been suitable for this project. I chose MySQL because I am familiar with it from previous projects and it is integrated with the Northumbria Web Hosting service. I also researched its capabilities and am confident that it will be sufficient to meet my objectives.

4.4 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.

4.4.1 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. REST is the application of HTTP protocol methods to client-server architectures, and PHP is already capable of handling HTTP protocol requests. This means that PHP is perfectly capable of handling the type of web application that I aim to create.

Other server-side options, such as Node.js, were considered, but I decided against them due to a lack of experience. I also considered using frameworks such as Laravel but decided against it because I want to learn the fundamentals of PHP first.

4.4.2 ReactJS

React.js is an open-source JavaScript package that is used to create single-page apps' user interfaces. We can also make reusable UI components with React. Developers may use React to build web applications that can alter data without reloading the page. React's major goal is to be quick, scalable, and easy to use.

The big reason I want to use react is due to reusable components. This means that every part of the interface I've already created can be utilised anywhere in my project by simply calling it from another component. Such a function is written once and can be used wherever. Furthermore, I can group these elements into sections or pages and put them with the same functionality wherever in the project.

4.5 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. I finally selected appropriate tools and techniques which will allow me to complete the design and implementation of the web application.

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

5.1 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. Once the low fidelity was made, I sent it to the client for feedback and improved it.

5.2 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 S].

The use case descriptions aided me in going through the steps of important functional requirements for my online application. Going through the processes of each need logically will help me during the implementation stage. The most important thing I learned from the use case descriptions is exactly what features I want to include in the admin dashboard. I was able to come up with ideas for how to combine various items for the CRM side of the project.

After I finished the descriptions, I created some use case diagrams to go with them. I thought the diagrams were quite redundant, but they did help me logically map out the flow of functionality, which will no doubt help me further down the line when I'm creating the components.

5.3 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 two different sequence diagrams to show the flow between users and important components in the system. I felt like any other sequence diagrams would be

redundant as they are quite generic, I was most interested in how the data would flow in the booking component.

[Appendix N,O] contains the two important data flows that I wanted to map out with a sequence diagram. The first one I made was for the login process. Albeit simple, it shows the process of the user logging in, and the client making a POST request with the login details. If the login details are in the database, then the server generates a JWT token and sends it back to the client. Once the client has the JWT token, the user then has user privileges across the web application.

5.4 Prototypes

The first wireframe design was simple, with the sole objective of mapping out the elements I provided in the use case descriptions. The web application includes more than just the reservation system; however, I will concentrate solely on the wireframe design of the booking system and the admin dashboard rather than the other static pages on the website (Menu, contact, home page etc.). [Appendix V] displays the initial low-fidelity wireframe that I built.

I then showed it to the client to see if he was happy with how I had laid out the criteria, and then I began work on the high-fidelity prototype. The second prototype concentrated on how the booking system would appear as well as the various pages of the dashboard that the admin would interact with from a mobile perspective. Based on feedback from the client, I created the prototype shown in [Appendix V]. The client stated that he and the majority of the customers would be using the web application on a smaller screen. This freed me up to concentrate on elements I knew the client thought were important from a mobile standpoint.

5.5 Entity Relationship

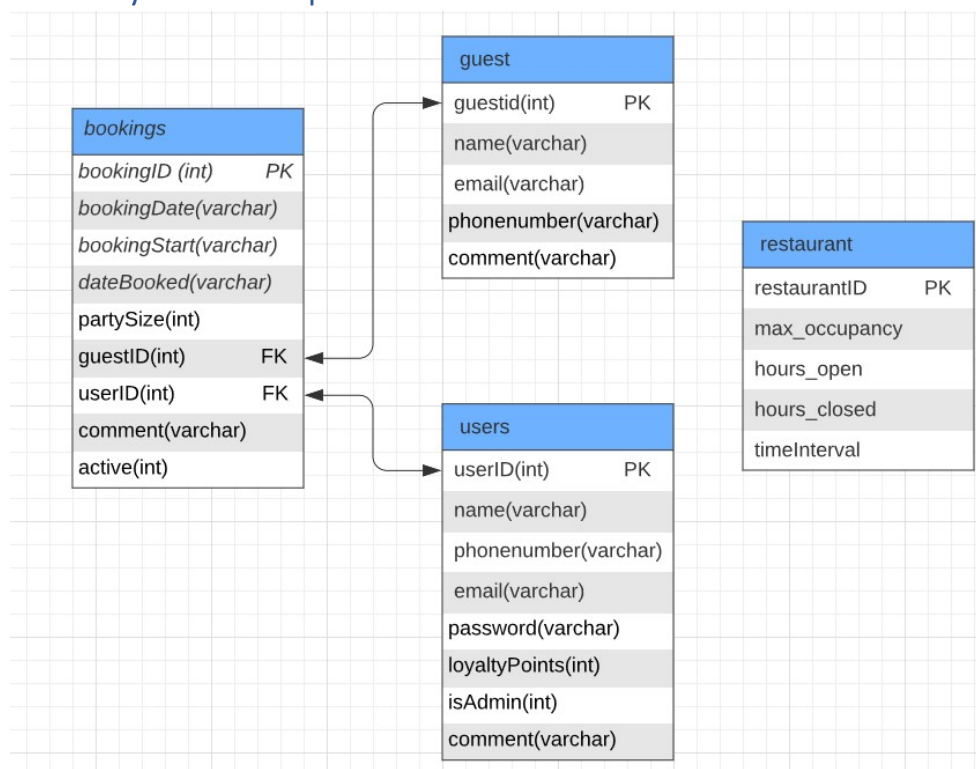


Figure 4 Final Entity Relationship Diagram

The initial entity-relationship diagram that I created shown in [Appendix L] was not the one I decided to use in the final version. The reason for this was that it caused several different issues when trying

to make a booking. The first rendition of the ER diagram didn't have any of the users' personal details saved in the users' table, and instead, all personal information was stored in the guest table. A userID was then attached to the guest table whenever the customer opted to become a user in the booking process. This caused multiple issues down the line which are discussed in the API section of Implementation.

The final entity-relationship diagram shown in figure 4 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. The client calls this table for reference and to store certain variables which can be checked against the incoming data from other tables.

5.6. 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".

The 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.

I used a similar arrangement for my backend files. For my server code, I used classes and used an object-oriented approach. In the config folder, I wrote an autoloader magic method that uses an "include once" function for all class names in my directory. I then wrote an exception handler and an error handler. Depending on whether the request was HTML or JSON, the exception handler would generate errors. I next build the actual configuration file, which initialises the other functions and defines key information such as the basepath and database. The project was then subdivided into controllers, databases, gateways, requests, and responses in the source folder. For extra security and cleanliness, I wanted to keep database requests isolated from controllers via gateways.

Implementation

6. 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 the OneDrive location.

7. API

7.1 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 M).

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 5 below.

```
class Database
{
    private $dbConnection;

    public function __construct() {
        $this->setDbConnection();
    }

    private function setDbConnection() {
        $this->dbConnection = new PDO("mysql:host=localhost;dbname=sliceboro",
            "root", "");
        $this->dbConnection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    }

    public function executeSQL($sql, $params=[]) {
        $stmt = $this->dbConnection->prepare($sql);
        $stmt->execute($params);

        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }

    public function querySQL($sql, $params=[]) {
        $stmt = $this->dbConnection->prepare($sql);
        $stmt->execute($params);
        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    public function lastInsert() {
        $stmt = $this->dbConnection->lastInsertId();
        return $stmt;
    }
}
```

Figure 5 Database class

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.

7.2 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.

7.3 Prevent multiple User Bookings

The idea of a user spam booking into the restaurant to collect loyalty points was a major issue that I needed to avoid. There are numerous approaches I could take to solve this, but I chose to use the database. Every booking entered into the database is assigned a userID as a foreign key, allowing the tables to be joined and the user associated with the booking to be displayed. I added a unique key to the userID so that the user with that userID can only make one booking. This solved the problem of multiple bookings under one user, but it created the problem of the booking having to be deleted from the database before that user could make another booking. The event scheduler solved this problem, but it meant that I couldn't use the "active" column in the bookings table because there was no reason to use it if the booking was going to be deleted after the bookingDate.

7.4 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.

```

abstract class Gateway {

    private $database;
    private $result;

    protected function setDatabase() {
        $this->database = new Database\Database;
    }

    protected function getDatabase() {
        return $this->database;
    }

    protected function setResult($result) {
        $this->result = $result;
    }

    public function getResult() {
        return $this->result;
    }
}

```

Figure 6 Abstract Gateway Class

The gateways share an abstract class from which they all derive. The getters and setters that interact with the database class are provided by the Gateway abstract class. I created a setter that sets it to the Database class and a getter that returns it to provide access to our private database variable. These functions are used to access the database by the other gateway classes. Getters and setters are used for a variety of reasons, including scalability, debugging, and cleanliness. Rather than scouring code for different variable assignments, you can easily refactor a getter. You can also place breakpoints between all of the functions to pinpoint where the bugs are in the code. They are also much cleaner, and if you need to change anything, you only need to change the getter and setter.

```

class ReservationGateway extends Gateway {

    public function __construct() {
        $this->setDatabase();
    }

    public function addGuestReservation($partysize, $bookingdate, $bookingstart)
    {
        $sql = "INSERT into bookings (userid, guestid, partysize, bookingdate, bookingstart, dateBooked) VALUES
        (NULL, LAST_INSERT_ID(), :partysize, :bookingdate, :bookingstart, CURRENT_TIMESTAMP)";
        $params = [":partysize" => $partysize, ":bookingdate" => $bookingdate, ":bookingstart" => $bookingstart];
        $result = $this->getDatabase()->executeSQL($sql, $params);
        $this->setResult($result);
        return $result;
    }
}

```

Figure 7 Reservation Gateway

The code shown in Figure 7 is 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.

```
public function checkBookingDate($bookingdate, $max_occupancy) {  
    $sql = "SELECT bookingDate, bookingStart, SUM(partysize) FROM bookings WHERE bookingDate =:bookingDate  
           GROUP BY bookingStart  
           HAVING SUM(partysize) > $max_occupancy";  
    $params = [":bookingDate" => $bookingdate];  
    $result = $this->getDatabase()->executeSQL($sql, $params);  
    $this->setResult($result);  
}
```

Figure 8 checkBookingDate() function

The "checkBookingDate" function in the reservation gateway is worth noting because it relates to the gateway classes. Finding a way to prevent overbooking at a specific timeslot was one of the most difficult challenges in designing the booking logic. The query above assisted me in resolving this problem by searching the database for all time slots (booking starts) on a specific day and grouping 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 total party size exceeds the restaurant's maximum occupancy. More on this topic will be covered in future sections on the front end.

7.5 Requests and Responses

```
class Request {

    private $basepath = BASEPATH;
    private $path;
    private $requestMethod;

    /**
     *
     * The construct parses the URL requested by the user using $_SERVER["REQUEST_URI"]
     * and extracts its path. The path is then trimmed and sent to determine its
     * request method
     *
     */

    public function __construct() {
        $this->path = parse_url($_SERVER["REQUEST_URI"])[ 'path' ];
        $this->path = strtolower(str_replace($this->basepath, "", $this->path));
        $this->path = trim($this->path, "/");
        $this->requestMethod = $_SERVER["REQUEST_METHOD"];
    }

    public function getPath() {
        return $this->path;
    }

    public function getRequestMethod() {
        return $this->requestMethod;
    }

    public function getParameter($param) {
        if ($this->getRequestMethod() === "GET"){
            $param = filter_input(INPUT_GET, $param, FILTER_SANITIZE_SPECIAL_CHARS);
        }
        if ($this->getRequestMethod() === "POST") {
            $param = filter_input(INPUT_POST, $param, FILTER_SANITIZE_SPECIAL_CHARS);
        }
        return $param;
    }
}
```

Figure 9 Request class

The following topic of discussion is the type of client request and the data format that should be returned. The Request class is shown in Figure 9, which parses the endpoint and stores it in a \$path variable to be used later. It also determines whether the data is requested via POST or GET by inspecting its parameters.

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.

```

class JSONResponse extends Response
{
    private $message;

    protected function headers() {
        header("Access-Control-Allow-Origin: *");
        header("Content-Type: application/json; charset=UTF-8");
    }

    public function setMessage($message) {
        $this->message = $message;
    }

    /** setter method for status code property */
    public function setStatusCode($statusCode) {
        $this->statusCode = $statusCode;
    }

    public function getData() {
        if (is_null($this->data)) {
            $this->data = [];
        }

        /** set different codes within this conditional */
        if (is_null($this->message)) {
            if (count($this->data) > 0) {
                $this->message = "ok";
                $this->setStatusCode(200);
            } else {
                $this->message = "no content";
                $this->setStatusCode(204);
            }
        }

        /** return the given code */
        http_response_code($this->statusCode);

        $response['message'] = $this->message;
        $response['count'] = count($this->data);
        $response['results'] = $this->data;

        return json_encode($response);
    }
}

```

Figure 10 JSON Response class

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.

7.6 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 [SEE CODE ON ONE DRIVE]. 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.

```
class Authenticate extends Controller {

    protected function setGateway() {
        $this->gateway = new Gateway\UserGateway();
    }

    protected function processRequest() {
        $data = [];
        $email = $this->getRequest()->getParameter("email");
        $password = $this->getRequest()->getParameter("password");
        $userid = $this->getGateway()->findPassword($email);
        if ($this->getRequest()->getRequestMethod() === "POST") {
            if (!is_null($email) && !is_null($password)) {
                $this->getGateway()->findPassword($email);
                if (count($this->getGateway()->getResult()) == 1) {
                    $hashpassword = $this->getGateway()->getResult()[0]['password'];

                    if (password_verify($password, $hashpassword)) {
                        $key = SECRET_KEY;

                        $payload = array(
                            "userid" => $userid[0]['userid'],
                            'isAdmin' => $userid[0]['isAdmin'],
                            "exp" => time() + 10000,
                            "iat" => time()
                        );
                        $jwt = JWT::encode($payload, $key, 'HS256');
                        $data = ['success' => 1,
                            'message' => 'You have successfully logged in.',
                            'token' => $jwt,
                            'email' => $email,
                            'userid' => $userid[0]['userid'],
                        ];
                    }
                }
            }
            if (!array_key_exists('token', $data)) {
                $this->getResponse()->setMessage("Unauthorized");
                $this->getResponse()->setStatusCode(401);
            }
        } else {
            $this->getResponse()->setMessage("Method not allowed");
            $this->getResponse()->setStatusCode(405);
        }

        return $data;
    }
}
```

Figure 11 Authenticate logic

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.

```
$payload = array(
    "userid" => $userid[0]['userid'],
    'isAdmin' => $userid[0]['isAdmin'],
    "exp" => time() + 10000,
    "iat" => time()
);
```

Figure 12 JWT payload

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.

```
$key = SECRET_KEY;
$decoded = JWT::decode($token, new Key($key, 'HS256'));
$isAdmin = $decoded->isAdmin;

if ($isAdmin === "1") {
    if (!is_null($openingTime)){
        $this->getGateway()->changeOpeningTime($openingTime);
    }
    elseif (!is_null($closingTime)){
        $this->getGateway()->changeClosingTime($closingTime);
    }
    elseif (!is_null($timeInterval)){
        $this->getGateway()->changeTimeInterval($timeInterval);
    }
    elseif (!is_null($max_occupancy)){
        $this->getGateway()->changeMaxOccupancy($max_occupancy);
    }
}
```

Figure 13 Checking admin through JWT

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 13. 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.

7.7 Reservation Controller

The reservation controller is a notable controller with which I had significant issues. To submit the booking details, the reservation controller is contacted at the end of the booking form. Initially, I only had one controller that handled both guest and user reservations, but I was having trouble storing the correct guestId and userId with the associated booking information. At the time of initial creation, the guest information was added to the database on every booking, and the user information was added depending on whether the customer opted into it. An if statement in the controller checked to see if the password variables were filled in. If they were, the API knew it needed to add a user to the database alongside the guest information.

The problem was that the `addUser` function was called after the `addGuest` function, which was using the `lastInsertId()` of the `addGuest` function to store the id into the booking table. This caused problems because my database's initial design included the user table as an addon to the guest table. The guest information was always created on a booking, and the user information was only created if the customer opted in at the end of the booking process. As a result, the `guestID` and `userID` were sometimes different, and the booking function would call `lastInsertId()` on a `userID` that was different from the `guestID`, potentially sending back incorrect personal information.

To address this issue, I decided to separate the user and guest tables so that the data was stored in both. I then created a separate controller for them, which allowed the `addReservation` function to retrieve the correct ID associated with the booking. In the long run, this actually improved the design of my system because it allowed me to completely separate the two entities without worrying about guest information leaking into the user areas.

7.8 Timeslot Controller

The Timeslot controller's primary function was to make a call to the restaurant table to determine when the restaurant was open and closed, and to dynamically generate booking slots based on the "timeInterval." For example, if the time interval is set to 30 minutes, time slots will be generated every 30 minutes from the open hours to the closed hours. The maximum occupancy will be used to determine the maximum number of individuals permitted at a specific time slot. For example, if the maximum occupancy is set to 30, no more than 30 persons will be able to book at a specified timeslot on a given day. Using the "bookingStart" and "partySize" columns in the bookings database, we can determine how many people have booked at a given time. If there are multiple bookings with the same "bookingStart" on the same day, and the "partySize" of each booking exceeds the maximum occupancy when added together, the time slot with the same booking start will be disabled in the booking process, so that no more people can have a booking start at the same time. One Drive contains the code and logic for the timeslot controller.

7.9 Endpoints

Endpoints are the URL routes used by the client to request data from the API. All of my endpoints are saved in the server's index file, as seen in [One Drive index.php]. The API is designed in such a way that the index file is the first point of call, and the controller that handles the request is called depending on the URL. This is accomplished by employing the terms "case" and "switch." After creating the controller associated with that endpoint URL, a case is produced that initialises the endpoint URL. When the endpoint URL is called, the request is handled by that specific controller. I attempted to configure my API so that the functionality for each page is separated into its own controller, preventing needless data from being spilled into other sections of the web application. Using a ".htaccess" file, all paths are redirected to the index file. A htaccess file allows for per-directory configuration modifications [19]. In my case, I created a "Rewrite Rule" that forces all server requests to go through the index file first.

8. PHPMailer

Using PHPMailer, emails could be sent directly to the user. PHPMailer is a code library that allows you to send emails safely and easily from a web server using PHP code. When a user forgets their password, I integrated PHPMailer into the usergateway to send them a URL to a forgot password form. My google mail account was connected via the SMTP (simple mail transfer protocol) and everything worked fine in sending and receiving mail. Further progress could be made on this to send confirmation emails, but due to time constraints, this feature was never successfully integrated.

8.1 Configuration and Error Handling

```
define('BASEPATH', '/kv6003/backend/');

define('DEVELOPMENT_MODE', true);
define('SECRET_KEY', '&MY[5Lx,xb#b""E/8eB&>x2}T^JDav');

ini_set('display_errors', DEVELOPMENT_MODE);
ini_set('display_startup_errors', DEVELOPMENT_MODE);

include 'config/autoloader.php';
spl_autoload_register("autoloader");

include 'config/htmlExceptionHandler.php';
include 'config/jsonExceptionHandler.php';
set_exception_handler("JSONExceptionHandler");

include 'config/errorHandler.php';
set_error_handler("errorHandler");
```

Figure 14 Config file

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.

```
function autoloader($className) {
    $filename = strtolower($className) . ".php";
    $filename = str_replace('\\', DIRECTORY_SEPARATOR, $filename);
    if (is_readable($filename)) {
        include_once $filename;
    } else {
        throw new exception("File not found: " . $className . " (" . $filename . ")");
    }
}
```

Figure 15 Autoloader class

The autoloader function shown in figure 15 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.

```

function JSONExceptionHandler($e) {
    header("Access-Control-Allow-Origin: *");
    header("Content-Type: application/json; charset=UTF-8");

    $output['error'] = "internal server error! (Status 500)";

    if (DEVELOPMENT_MODE) {
        $output['Message'] = $e->getMessage();
        $output['File'] = $e->getFile();
        $output['Line'] = $e->getLine();
    }
    else {
        error_log(
            "message: " . $output['Message'] = $e->getMessage() .
            "file" . $output['File'] = $e->getFile() .
            "line" . $output['Line'] = $e->getLine()
        );
    }
}

echo json_encode($output);

```

Figure 16 JSON exception handler

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 16 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.

9. 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 React's built-in "useState" hook shown in figure 17. 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.

```

const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const [errMsg, setErrMsg] = useState('');
const [userId, setUserId] = useState('');

```

Figure 17 useState functions

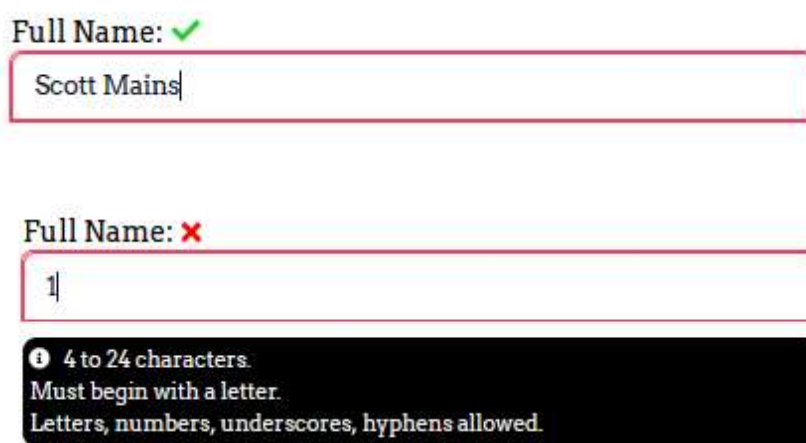
9.1 Register

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.

```
<form className="mt-8 space-y-6" onSubmit={handleSubmit}>
  <div className="rounded-md shadow-sm -space-y-px">
    <label htmlFor="username"> Full Name:
      <FontAwesomeIcon icon={faCheck} className={validName ? "valid" : "hide"} />
      <FontAwesomeIcon icon={faTimes} className={validName || !name ? "hide" : "invalid"} />
    </label>
    <input className="appearance-none rounded-none relative block w-full px-3 py-2 border border-gray-300 placeholder-gray-500 text-gray-900 rounded-t-md focus:outline-none focus:ring-rose-500 focus:border-rose-500 focus:z-10 sm:text-sm" type="text" id="name" ref={userRef} autoComplete="off"
      onChange={(e) => setName(e.target.value)} value={name} required aria-invalid={validName ? "false" : "true"} aria-describedby="uidnote" onFocus={() =>
        setNameFocus(true)} onBlur={() => setNameFocus(false)} /> <p id="uidnote" className={nameFocus && name && !validName ? "instructions" : "offscreen"}>
      <FontAwesomeIcon icon={faInfoCircle} /> 4 to 24 characters. <br /> Must begin with a letter. <br /> Letters, numbers, underscores, hyphens allowed.
    </p>
  </div>
</form>
```

Figure 18 Beginning of the Sign Up form

Figure 18 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.



Full Name: ✓

Scott Mains

Full Name: ✗

1

4 to 24 characters.
Must begin with a letter.
Letters, numbers, underscores, hyphens allowed.

Figure 19 Sign up form validation

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 19, 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.

```
useEffect(() => {
  setValidName(USER_REGEX.test(name));
}, [name])
```

Figure 20 UseEffect function implement REGEX

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.

```
const handleSubmit = async (e) => {
  e.preventDefault();

  const v1 = USER_REGEX.test(name);
  const v2 = PWD_REGEX.test(password);
  if (!v1 || !v2) {
    setErrMsg("Invalid Entry");
    return;
  }
  try {
    const fd = new FormData();
    fd.append('name', name);
    fd.append('phonenumber', phonenumber);
    fd.append('email', email);
    fd.append('password', password);
    const response = await axios.post('http://localhost/kv6003/backend/api/register', fd);
    setSuccess(true);
    setSignUpContent(false);
    //clear state and controlled inputs
    //need value attrib on inputs for this
    setName('');
    setPwd('');
    setMatchPwd('');
  } catch (err) {
    if (!err?.response) {
      setErrMsg('No Server Response');
    } else if (err.response?.status === 418) {
      setErrMsg('Student ID already in use');
    } else if (err.response?.status === 419) {
      setErrMsg('Email Address already in use');
    }
    else {
      setErrMsg('Registration Failed')
    }
    errRef.current.focus();
  }
}
```

Figure 21 handleSubmit function for Sign up

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.

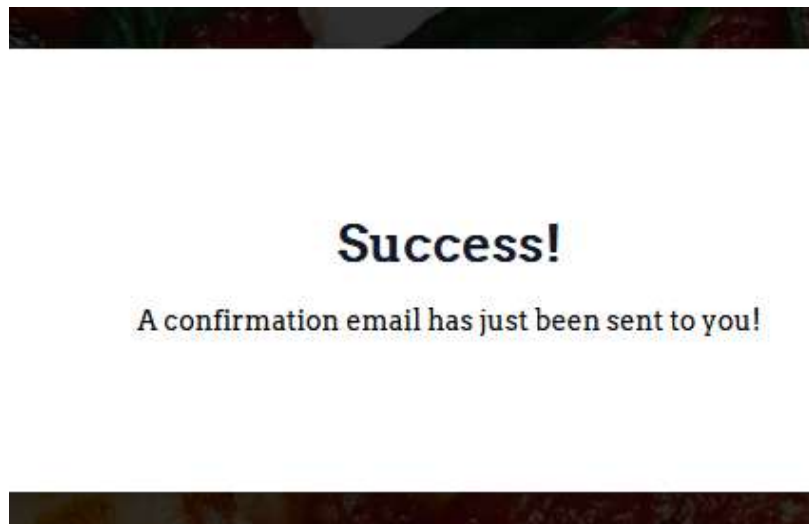
The variables that are to be posted are then appended to form data to be sent as an API request. If the API request is successful, then the success variable is triggered to "true" and the sign-up form content will be replaced with a success message.



The screenshot shows a web form with a pink error banner at the top that reads "Email Address already in use". Below the banner is a large, bold, black "SIGN UP" button. Underneath the button, the text "Full Name: ✓" is displayed. Below this, there is a text input field containing the name "Scott Mains".

Figure 22 Email already in use error

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.



The screenshot shows a web form with a dark green success banner at the top. Below the banner, the word "Success!" is displayed in a large, bold, black font. Underneath "Success!", the text "A confirmation email has just been sent to you!" is displayed in a smaller, bold, black font.

Figure 23 Sign up success

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.

9.2 Login

```
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const fd = new FormData();
    fd.append('email', email);
    fd.append('password', password);
    const response = await axios.post('http://localhost/kv6003/backend/api/authenticate', fd)
    .then((res)=> {
      localStorage.setItem("sliceLogin", res.data.results.token);
      console.log(res.data.results)
      window.location.reload(false);
    });
  }
};
```

Figure 24 handleSubmit for login

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.

9.3 JSON Web Token

```
const App = () => {

  const [userId, setUserId] = useState("");
  const [isAdmin, setIsAdmin] = useState("0");
  const [jwtToken, setJwtToken] = useState("0");
  const [authenticated, setAuthenticated] = useState(false);

  useEffect(() => {

    const token = localStorage.getItem("sliceLogin");
    try {
      const tokenExp = jwt_decode(token);

      if (tokenExp.exp < Date.now() / 1000) {
        setAuthenticated(false)
        localStorage.removeItem('sliceLogin');
      } else {
        setJwtToken(token)
        setAuthenticated(true);
        setIsAdmin(tokenExp.isAdmin);
        setUserId(tokenExp.userid);
      }
    } catch {
      console.log("token doesn't exist");
    }
  }, [authenticated])
}
```

Figure 25 decoded JWT token in App.js

Within the code shown in figure 25, 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.

9.4 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.

```
const GenerateTimeslots = (e) => {
  const toMinutes = str => str.split(":").reduce((h, m) => h * 60 + +m);
  function timeToMins(time) {
    var b = time.split(':');
    return b[0]*60 + +b[1];
  }
  let newTimeInterval = timeToMins(timeInterval);

  const toString = min => (Math.floor(min / 60) + ":" + (min % 60))
    .replace(/\b\d\b/, "0${&}");

  function slots(startStr, endStr=closingTime) {
    let start = toMinutes(startStr);
    let end = toMinutes(endStr);
    return Array.from({length: Math.floor((end - start) / newTimeInterval) + 1}, (_, i) =>
      toString(start + i * newTimeInterval)
    );
  }

  let newTimeSlots = slots(openingTime)
  if (allTimeSlots) {
    const index = newTimeSlots.indexOf(allTimeSlots);
    if (index > -1) {
      newTimeSlots.splice(index, 1);
    }
    console.log(allTimeSlots)
  }
  return (
    newTimeSlots.map(item => (
      <button
        key={item}
        onClick={(e) => {setSelected(item); setTimeSlot(item); }}
        className={item === selected ? "bg-blue-700 text-white font-bold py-2 px-4 rounded" : ""}
        value={item}
      >
        {item}
      </button>
    ))
  )
}
```

Figure 26 generateTimeSlots() function

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.

9.5 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.

```
<Calendar
  className="mx-auto"
  onChange={onChange}
  value={value}
  onClickDay={onDayPress}
  minDate={new Date()}/>
```

Figure 27 Calendar plugin

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.

```
const onDayPress = (value) => {
  let datevalueshow = moment(value).format("DD/MM/YYYY");
  let datevalue = moment(value).format("YYYY/MM/DD");
  setDate(datevalue);
  setDateShow(datevalueshow);
  setShowTime(true);
  let formData = new FormData();
  formData.append('bookingDate', datevalue);
  formData.append('maxoccupancy', maxOccupancy);
  axios.post('http://localhost/kv6003/backend/api/checktimeslots', formData)
    .then(resp => {
      if (resp.data.results) {
        setAllTimeSlots(resp.data.results[0].bookingStart)
      } else {
        setAllTimeSlots(null);
      }
    });
};
```

Figure 28 onDayPress() function

This function transformed the date into a MySQL-readable format, which was then appended to formData and delivered to the "checkTimeSlots" API endpoint. This endpoint returns all overbooked appointments for that date so that they can be eliminated from the timeslot option.

9.6 Admin Dashboard

The admin functionality in the dashboard was the final feature I added to the system. If the user is authenticated as an administrator using a JSON web token, they will have access to the admin dashboard. The admin dashboard is divided into four sections: bookings, customers, marketing, and settings.

9.6.1 Booking

```
let filteredResults = allBookings;
if (allBookings) {
  if ((filteredResults.length > 0) && (search !== undefined)) {
    filteredResults = filteredResults.filter(filterSearch)
  }
}

let content;
if (filteredResults) {
  content = filteredResults.map(booking => <AdminBookingExcerpt key={booking.bookingid} booking={booking} />)
}
```

Figure 29 Admin booking data

I displayed all existing bookings on the bookings page by mapping them via a child component. I'm mapping all of the data received from the Admin Bookings API and passing it down to the child component in the code shown above.

```
<div>
  <p className=""> {booking.partysize} </p>
</div>
```

Figure 30 Passing data to child component

The data that was just received can then be called across the code within the child component. It displays specific elements of the object that was supplied to it.

9.6.2. Marketing

```
const handleEmails = () => {
  setChecked(!checked)
  if (checked) {
    setArray((oldArray) => oldArray.concat(email))
  }
  if(!checked) {
    setArray("")
  }
}
```

Figure 31 Function for adding to email array

The marketing page's goal was to display a list of customers' emails, and the admin would be able to tick a checkbox, resulting in the email address being added to a list within an input field. The administrator might then send promotional offers directly from the page. This was a CRM feature that I intended to add from the start which will help marketing automation.

9.7 Modals

The last point I'd like to touch on is the use of modals. Modals are prompted dialogue boxes that inform the user to new content. I utilised them to ask the user to provide information that is

necessary for the current process to continue, such as cancelling a booking or changing crucial information such as personal details.

```
let modalContent;  
  
if (showModalLoyaltyUpdate) {  
  modalContent = (  
    <UpdateLoyaltyModal setShowModalLoyaltyUpdate={setShowModalLoyaltyUpdate} userId={customer.userId}/>  
  )  
}  
}
```

Figure 32 Modal Content

Modals were triggered by saving modal content in a variable. When a button that invoked the modal was pressed, the if statement was checked to see if it was true and updated the modal content with the modal child component.

10. Testing and Results

10.1 Functional Testing

Functional testing is a way of verifying that a project or product does what it's supposed to do and what people expect. Functional testing is traditionally carried out by a team of testers, but as I am a solo developer, I will be conducting all the tests I deem appropriate by myself. Functional testers capture requirements and give developers and users confidence that the project/product under test meets those requirements, as well as confirm that the User Story/Project/Product is ready for production release.

10.1.2 Manual Testing

Manual Functional testing was conducted throughout the project. I wanted to test all the important functions within my system to check if they were working in their intended way. A full log of functional testing can be found in [Appendix P]. I did it from a client-side perspective, as all relevant server-side code would be called from the client so I would know if there were errors on both sides.

Task	Description	Expected Outcome	Actual Outcome	Resolution
handleSubmitLogin()	Login details are appended as form data and posted to the authenticate controller.	If login details are in the database, a success message appears. If not, then an appropriate error message will appear.	Expected outcome occurred	N/A

Figure 33 Functional testing format

The format I used is shown in Figure 33. I tested the function immediately after it was created and continued to test it throughout the project to ensure that it worked as expected. I recorded the actual outcome, and if it differed from what was expected, I made sure to fix it and added the fix to the resolution column.

10.1.3 PHPUnit

```
PS C:\xampp\htdocs\kv6003\backend> ./vendor/bin/phpunit ./tests/usergatewaytest.php
PHPUnit 9.5.20 #StandWithUkraine

-                                                                    1 / 1 (100%)

Time: 00:00.008, Memory: 4.00 MB

OK (1 test, 3 assertions)
PS C:\xampp\htdocs\kv6003\backend>
```

Figure 34 PHP Unit results

PHPUnit is a framework-independent unit testing library for PHP. Unit testing is a technique for evaluating tiny chunks of code against expected outcomes. Individual components are rarely checked separately in traditional testing. This might lead to problems being undiscovered for a long period, making it difficult to pinpoint the source of a specific bug.

I was completely unfamiliar with PHPUnit once I finished the final prototype of my web application, so testing was very limited. The testing I managed to achieve was done on the gateways of the project to test API responses. I tested whether the received JSON response matched what I expected it to be, and each result came back positive and that it didn't need to be rectified.

10.2 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 customers and admin. Ethical consent was obtained from all candidates before testing was conducted can be provided.

10.2.1 Customer Testing

Task	Description	Expected Outcome	Actual Outcome
User makes a booking using the guest booking form	User goes through the booking form without being authenticated.	Booking will be created and sent to the database. A success page showing their booking details will display.	
User Sign Up with incorrect credentials	User enters sign up credentials without following the validation procedures	The form will alert the user to which part is incorrect and they will not be able to sign up	
User Sign Up with correct credentials	User enters sign up credentials while following the validation procedures	User successfully signs up and will be redirected to the home page showing that they are authenticated.	
User Sign In with incorrect credentials	User enters incorrect login details that do not exist in the database.	If credentials are incorrect then an unauthorised error message will appear.	
User Sign In with incorrect credentials	User enters existing login details present in the database	User successfully signs in and will be redirected to the home page showing that they are authenticated.	
Forgotten Password	User clicks forgot password on the login form which redirects them to a page to enter their email. They will then be sent an email which directs them to a form to reset their password	User follows the forgot password procedure and successfully changes their password.	
User makes a booking using booking form	User goes through the booking form while being authenticated.	Booking will be created and sent to the database. A success page showing their booking details will display.	
User makes a booking while they already have one existing in the system	User goes through the booking form while being authenticated after already making a booking previously	An error message will appear saying that a booking already exists in the system.	

Figure 35 Customer Testing Plan

The table above shows the full customer testing plan that I conducted with 5 different candidates. I tried to make sure that the testing plan covered the full extent of the web application and that any bugs a user might encounter won't slip under the woodwork.

So far, user feedback has been overwhelmingly positive. Nobody encountered any problems that prevented them from completing the task. The system was designed to be simple, so it wasn't prone to breaking down on the user, which could explain why there wasn't much feedback from those who tested it. The results from the five different users who tested it are shown in Table 4 below. Each one completed each task with ease and told me that they thought the system was well designed and that they would use it in a live environment

	SignUp - Correct	SignUp- Incorrect	Guest Booking	Sign In- Correct	Sign In - Incorrect	Forgotten Password	User booking form	User existing Booking	User delete booking	User edit personal details	Signout
TEST 1	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y
TEST 2	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y
TEST 3	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y
TEST 4	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y
TEST 5	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y

Figure 36 Customer testing plan results

Figure 36 shows the results of the customer usability tests. Every activity they carried out worked successfully, except the forgotten password functionality. I was already familiar with the fact that this didn't work. Overall Customer Usability was a success.

10.2.2 Admin Testing

Task	Description	Expected Outcome	Actual Outcome
Admin sign in	The admin signs in to the system	Admin is logged in and navigated to the home page. They will have access to the Admin dashboard via navigation bar	
Admin delete booking	Admin clicks delete booking on a booking slot in the admin page	The booking is deleted from the database and the page refreshes showing that the booking has gone.	
Admin Update Customer Comment Correct	Admin updates the comment attached to the customer on the customer page	Admin clicks update, a modal shows up and the user types in anything they want. The page will refresh reflecting the update	
Admin Update Customer Loyalty Points correct	Admin updates the Loyalty attached to the customer on the customer page	Admin clicks update, a modal shows up and the user types in integer value. The page will refresh reflecting the update	
Admin Update Customer Loyalty Points incorrect	Admin updates the Loyalty attached to the customer on the customer page but uses a value other than integer	Admin clicks update, a modal shows up and the user types in string value. The update will not occur and a box will remind them to use integers	
Admin add email to email registry	Admin selects a checkbox located next to a customer's name.	The customer's email address gets added to the text field area at the top of the page	
Admin send email to customers	Admin types a message in the text area underneath the email registry. Clicks submit	The message typed in the text area gets sent to the customer's email.	
Admin Booking intervals	Admin chooses a booking interval in the settings page and clicks update.	The page refreshes and reflects the updated booking interval	
Admin set max occupancy correct	Admin updates the max occupancy on the settings page and clicks update.	Admin types an integer value and the page refreshes updating the change	
Admin set max occupancy incorrect	Admin updates the max occupancy on the settings page and clicks update.	Admin types a string value and the modal notifies the admin to use an integer	
Admin set Opening closing hours correct	Admin updates the opening and closing hours on the settings page and clicks update	Admin types a time format and the page refreshes reflecting the update	
Admin set Opening closing hours incorrect	Admin updates the opening and closing hours on the settings page and clicks update	Admin types a string value and the modal notifies the admin to use a time.	

Figure 37 Admin testing plan template

Figure 37 shows the format I used to test the web application with the restaurant owner. Because it was only one person testing, I wanted to make sure the test was repeated several times to minimise the number of bugs that could occur. The overall feedback was positive and most of the functionality worked without a hitch.

	SignUp - Correct	SignUp- Incorrect	Update Customer Comment	Update Customer Loyalty Points	add Email registry	Send marketing email	Update booking interval	Update max occupancy	Update closing hours	Update opening hours	Signout
TEST 1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TEST 2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TEST 3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TEST 4	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TEST 5	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 38 Admin testing plan results

Figure 38 shows the results of the admin usability tests. Every functionality worked great, and the restaurant owner only had positive things to say about the system.

10.3 Vulnerability Testing

10.3.1 XSS Attack

To test to see if my application is susceptible to an XSS attack, I am going to use test my code via an XSS scanner. I will then test it again against an attack proxy called ZAP. Finally, I won't depend on the software and conduct a test myself attempting to inject a script.

10.3.2 XSS Scanner



Figure 39 XSS scanner results

The XSS scanner showed minimal information and stated that only basic information would be accessible to the attack. Overall positive results were displayed telling me that it will not have a great impact on me as a host.

10.3.3 ZAP



Figure 40 JSON data returned that should be obscured to public

The ZAP proxy attack tool did in fact display some vulnerabilities within my website. Because the CORS was set to “Access-Control-Allow-Origin: *”, it did expose an endpoint that returned an array of customers that should only be available to the admin. It also returned restaurant opening information. None of the information was strictly confidential or could be used in a harmful way, as it only showed public information and names and emails, but it was still data that I would like to prevent being accessible.

The solution to this issue is to ensure that sensitive data is not available in an unauthenticated manner. I can do this by configuring the CORS HTTP header to a more restrictive set of domains or removing CORS headers entirely, to allow web browsers to enforce the Same Origin Policy in a more restrictive manner. I could also restrict it further by making this information only accessible if the JSON web token is posted.

10.3.4 Self-Test



Unauthorized

Email:

riptlet.html < <<SCRIPT>alert("XSS");//<</SCRIPT> %253cscript

Password:

.....

Figure 41 XSS self test

10.3.5 SQL Injection Attack

I am going to test for SQL injection vulnerabilities using a proven open-source penetration tool called SQL map. This tool automates the process of detecting and exploiting SQL injection flaws. I am going to use this tool against several important endpoints that exist within my API.

10.3.5.1 User Reservation Endpoint

```

[14:43:49] [WARNING] you've provided target URL without any GET parameters (e.g. 'http://www.site.com/article.php?id=1') and without pro
ough option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q] Y
[14:43:49] [INFO] testing connection to the target URL
[14:43:49] [WARNING] the web server responded with an HTTP error code (405) which could interfere with the results of the tests
[14:43:49] [INFO] checking if the target is protected by some kind of WAF/IPS
[14:43:49] [CRITICAL] heuristics detected that the target is protected by some kind of WAF/IPS
are you sure that you want to continue with further target testing? [Y/n] Y
[14:43:49] [WARNING] please consider usage of tamper scripts (option '--tamper')
[14:43:49] [INFO] testing if the target URL content is stable
[14:43:49] [INFO] target URL content is stable
[14:43:49] [INFO] testing if URI parameter '#1*' is dynamic
[14:43:49] [INFO] URI parameter '#1*' appears to be dynamic
[14:43:50] [WARNING] heuristic (basic) test shows that URI parameter '#1*' might not be injectable
[14:43:50] [INFO] testing for SQL injection on URI parameter '#1*'
[14:43:50] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:43:50] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[14:43:50] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[14:43:51] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[14:43:51] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[14:43:51] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[14:43:52] [INFO] testing 'Generic inline queries'
[14:43:52] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[14:43:52] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[14:43:52] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[14:43:53] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[14:43:53] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[14:43:53] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[14:43:53] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to redu
] Y
[14:43:54] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[14:43:54] [WARNING] URI parameter '#1*' does not seem to be injectable
[14:43:54] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you
f you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--
switch '--random-agent'
[14:43:54] [WARNING] HTTP error codes detected during run:
405 (Method Not Allowed) - 2 times

```

Figure 42 SQL map results

The results from the first endpoint indicated that an SQL injection could not be made. This is primarily because an HTTP error code gets flagged twice during the injection process. This test result was mimicked across all the endpoints that I fed into the software. Overall, I am happy that my endpoints cannot be penetrated via SQL injection.

10.3.5.2 Self-Test



Unauthorized

Email:

" OR " ' = " ' "

Password:

••••••••••

Figure 43 SQL injection self test

I decided to test for SQL Injections vulnerabilities myself to be extra careful. To do this I performed a very basic SQL injection on the login form. If the attack was successful, it would simply return all the results in the table. This was prevented using PDO Statements. These are statements which are used instead of standard SQL queries that help prevent these attacks from occurring.

10.3.6 URL Attack

The URL attack was covered in the previous sections' ZAP results. Some API URLs were exposed and needed to be made more secure. To accomplish this, I could make all endpoints that display data require the user to POST a JSON web token in order to view the data. I decided to use this only when data changes are involved. Because the data is only dummy data, I determined that the endpoints exposed did not contain personal information. In the future, I would secure each endpoint with a JSON web token.

Evaluation and Conclusion

11 Evaluation

11.1 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.

11.2 Build Quality

I think the quality of my system was quite robust and all the requirements that were met, were met well.

The error handling had a significant amount of thought put into it, and I think it would work quite well in a live context. Whenever an error occurred on the server or client side, a message would be presented to the user telling them exactly what went wrong and how they can solve the issue.

I also believe the interface was well-designed and mobile-friendly. Every single page worked flawlessly on mobile, and in some cases, even looked better. I also used a variety of react packages as well as styling techniques to make the static pages more appealing to the user. The majority of this was due to tailwindCSS, but there were some parts that I integrated myself that I am very pleased with.

11.3 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.

The user testing was quite successful. It didn't provide me with great feedback as everything that was tested was successful and didn't flag any errors, except the few functions that I was familiar with that didn't work. The usability testing was a great way to get feedback on the design and how they felt the system worked as a whole. The feedback on this was great and positive overall and the users that it was very functional.

The vulnerability testing was quite lacklustre, as I depended on software and scripts that other people created for most of it. The manual vulnerability testing I did was very poor, as I couldn't find any information online that showed me how to test a REST API for those specific vulnerabilities. If done again I would learn more about how to vulnerability test and test the software across a multitude of different systems.

11.4 Client & User Feedback

Overall feedback from both client and user was positive. A lot of the people that responded have told me that they would happily use the system in a live environment. There was no further feedback given other than the fact that they would not make any changes and that the things that didn't work should obviously work in a live context.

11.5 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.

11.5.1 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.

11.5.2 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.

I also discovered that decoding a JWT token on the client is insecure. When a JWT is decoded, the secret key is typically exposed, which is extremely unsafe. My method didn't require a secret key for decoding, which made it much safer. I handled it in such a way that it merely takes the payload and does not necessitate the use of the secret key. It's worth noting that my method isn't the ideal way to get the user id and admin credentials, since it would be much better to just construct a server endpoint to get the information.

11.5.3 CRM System

I tried my hardest to think of several CRM capabilities that could bridge the gap between customer and restaurant once I finished building my booking form and had a way of gathering user data. One of the key ways I did this was by utilising the loyalty system. Every time a customer made a reservation at the restaurant, their loyalty point total increased, and it was reflected in their user profile. On the user profile, there was also a reward section where the user could see what awards

they may aspire for every time they booked into the restaurant. This was a terrific CRM option to add, and it's also a popular one in other booking systems. According to a study [39], this type of customer connection is critical for brand loyalty and results in repeat customers.

Another CRM feature I added to the software was the option for the administrator to add remarks to the customer in the admin dashboard. When an administrator adds a comment, the comment will be displayed on the booking slot anytime that specific user makes a booking. This is a wonderful approach for the administrator to add specific requirements to the user so that they may be catered for individually. This will improve customer relations because giving a unique and special experience for the customer is a significant strategy to retain customers.

11.6 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.

11.6.1 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. It is also much more portable and free to use. I struggled to find a free hosting platform for MySQL which allowed me to host my server-side on Heroku, so I had to just use NUMYSPACE to host everything when I wanted to send out my prototype for some feedback.

11.6.2 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.

11.6.3 React

For the client-side of the project, React was an excellent choice. As previously said, you can easily install pre-existing packages and components into react, making life a lot easier and allowing you to integrate pre-tested solutions produced by others. One example is how I used a "React Calendar" component to easily add calendar features to my application. It provided me with utilities that

allowed me to simply grab the date that the user selects and set it into a variable for database submission.

The primary issue with react was the decision to use Class-based components at first. I realised about halfway through the development cycle that employing functional components with hooks is much more fluid and modern. Developers have completely replaced class components with functions and hooks, according to the findings [6]. I chose to change all my prior class-based components to functional components instead. This ended up costing me a lot of time and eating into the time I had set aside for other programme functions.

Overall, I believe my code in react is clean and well-organized. There are a few things I would tweak to make it more reusable and scalable. For example, the html in the booking forms is disorganised. Instead of breaking up the various portions of the booking form into separate components, I chose to do it all on one page. The main reason for this was a lack of experience when it came to sending down props to child components. I found that keeping all logic in one block of code rather than dividing it apart was much easier at the start of the project. The more I utilised react, the more reusable components I created. When I first started constructing my admin dashboard, I used child components to pass down data from the endpoints, which is an example of how I improved with react. If I had to redo the project, I would absolutely make my code more reusable and clearer.

11.6.4 Visual Studio Code

Visual Studio Code is a code editor that helps the user to be very efficient when developing code. It offers a large extension library that allows the user to add quality of life features to their arsenal when coding. The ES7+ Snippets extension is one example of this. This enabled me to write shorthand like "rafce," which would build a full template of a functional component. This was quite useful for speeding up the development process because I didn't have to worry about typing out the same pieces of code repeatedly. I created way over 40 components in my front end, thus this would have been quite time-consuming.

11.6.5 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.

11.6.6 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.

12. Process Evaluation

12.1 Project Life Cycle

Throughout the project's lifecycle, I used an AGILE project framework. AGILE is a development process that emphasises test-driven development, coding in small sprints, and maintaining regular communication with the customer and users. This framework, I believe, was a wonderful fit for the manner I built this application. Throughout the research stage, I kept in touch with the client to ensure that all their needs were met. After that, I used the low-fidelity prototype to ensure they were satisfied with the initial design. Any comments received were then addressed in the mobile high-fidelity prototype.

The design was then implemented in "small sprints," as recommended by the AGILE SCRUM framework. This meant that I should pick a user requirement from the Trello board I made and make as much work as feasible within the time window I selected. This was an excellent method for attempting to complete work quickly and efficiently. I didn't always have good sprints, and some portions took much longer than others, but it was a terrific tool for me to focus on and complete certain tasks.

12.2 Time Management

My time was split up at the start of the project using a GANTT chart [Appendix W]. 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.

I believe I managed my time well in some "sprints", but in others, I spent way too long. Because of this, I didn't manage to incorporate all the functionality that I planned at the start. I also didn't spend enough time on the research and planning stage. I did a great bulk of it at the start but never finished it all and the work that was left lingered for a long time. This meant that my literature review wasn't as good as I wanted it to be. This also applied to my designs to a lesser extent. I finished my designs to an average standard in the hopes of returning to them before I heavily delved into the coding side. This didn't happen exactly as I wanted to and some were neglected and weren't developed.

I also spent far too much time than I would have liked on some aspects of the coding. As an example, I spent a significant amount of time attempting to disassemble my code into reusable functions and converting all existing classes into functional components. If I were to redo the project, I would keep the existing class components and build on top of them.

12.3 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.

In terms of criticism, I did leave out several aspects that I wanted to include in the specification criteria. Due to timing constraints, the forgot password retrieval process was not entirely functioning. This feature was prioritised as a medium, thus it should have been implemented. I got it to the point where it sent the user an email with the new password link, but it didn't totally function in the sense that it updated the user's password. If I had more time to devote to the application, I would spend more time on this problem.

12.4 Learning Process

12.4.1 Skills

Throughout the project, I learned a wide range of skills that have helped me advance as a developer. The biggest thing I improved on was my problem-solving abilities. At various moments throughout the assignment, I was perplexed as to how I could complete a task. There was a time when I felt I'd never be able to do it since I lacked the necessary experience. After extensive investigation and perseverance, I was able to resolve the problem, and it felt amazing to see the program work as it should. I believe that accomplishments like this will help me tackle challenges at a much faster speed in the future.

Another ability I honed was project management, which is still a work in progress. The Gantt chart generated at the outset and the Trello board utilised throughout the project aided in the development of the application. I obviously did not make the best use of it. I became more frustrated when I spent too much time on one task. It would have been preferable for me to take a break from some aspects of the project and return to them with a fresh perspective. To do this, I should have spent more time writing the report, for example, or working on a UML diagram so that I could think through the problem more clearly.

Throughout the project, I definitely improved my coding skills. I would have considered myself very inept with react at first, but by the end, I was solving problems much faster and didn't need to rely on sites like StackOverflow as much. Despite this, there are some parts of my code that I wish I had more time to clean up. I have the skills to go back into the code I started at the start of the project and make it more reusable, but due to time constraints, I was unable to make the code as robust as I would have liked.

12.4.2 Lessons Learnt

If there is any lesson that I learnt through this experience it is to not get too hung up on something for too long. There were times where I was spending hours scouring through solutions on stack overflow to try and solve my problem, but it usually just ended up in frustration. What actually ended up working for me was actually taking a break from the problem and working on something else, and the solution would work its way into my brain when I was least expecting it (such as late at night in bed).

Another important lesson learned is to devote more time to design rather than jumping right into coding. I don't believe my designs accurately represent the product I've created. I believe that if I sat down and created more detailed designs, my application would be even better. I could have even thought of things to add to the application's functionality that I didn't necessarily decide on in the requirements.

12.5 Social & Ethical Issues

12.5.1 Social

The software's dependability could become a social concern. It is possible that the user will book a slot in the restaurant and an error will occur, resulting in the booking not being placed. As a result, the user may mistakenly believe they are booked into the restaurant when they are not. The consumer expects the programme to perform its duty efficiently and without error, but because it was not built by a team of professionals, errors may occur, causing societal difficulties.

12.5.2 Ethical Issues

Ethical issues were the primary concern when conducting my project. I minimized all personal data retained in the requirement capture process by making sure everyone that participated was anonymous, and that they signed a form acknowledging that I will be collecting data from them and storing it for a limited amount of time. For the client, I made sure that they knew exactly what I was creating for them and what is expected of them. I also made sure they were okay with their restaurant being identifiable, which they were okay with. Despite the restaurant name being known, I made sure that any staff names were not mentioned.

Because my web application was a reservation system, it was inevitable that I needed to store some personal information such as emails, names, and phone numbers. Because of this, I asked any participants in the feedback loop to make sure they used dummy data and not anything that could identify them. This was to make sure that I was in accordance with the GDPR. Although I was confident in the security of my application, I couldn't be too sure that it was unexploitable and that data could be leaked.

13. 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.

13.1 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.

13.2 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.

13.3 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](https://www.redhat.com/en/topics/api/what-is-a-rest-api#rest). (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'.
- [23] Nielsen, Henrik, Jeffrey Mogul, Larry M. Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. 'Hypertext Transfer Protocol – HTTP/1.1'. Request for Comments. Internet Engineering Task Force, June 1999. <https://doi.org/10.17487/RFC2616>.
- [24] Masse, Mark. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc., 2011.
- [25] Berners-Lee, Fielding, and Masinter, 'Uniform Resource Identifier (URI)'.
- [26] Rodriguez, 'RESTful Web Services: The Basics'.
- [27] 'HTTP Headers - HTTP | MDN'.
- [28] Hong et al., 'A Study on a JWT-Based User Authentication and API Assessment Scheme Using IMEI in a Smart Home Environment'.
- [29] Haekal and Eliyani, 'Token-Based Authentication Using JSON Web Token on SIKASIR RESTful Web Service'.
- [30] Rahmatullo, Aldya, and Arifin, 'Stateless Authentication with JSON Web Tokens Using RSA-512 Algorithm'.
- [31] Ankush, 'XSS Attack Prevention Using DOM Based Filtering API'.
- [32] 'React XSS Guide'. <https://www.stackhawk.com/blog/react-xss-guide-examples-and-prevention/>
- [33] *Payloadbox/Sql-Injection-Payload-List*. <https://github.com/payloadbox/sql-injection-payload-list>
- [34] XSS payload Strings <https://github.com/danTaler/detectionString/projects?type=beta>
- [35] 'SQL Injection'. https://www.w3schools.com/sql/sql_injection.asp
- [36] Thomas, 'A Methodology for Linking Customer Acquisition to Customer Retention'.
- [37] Foss, Stone, and Ekinici, 'What Makes for CRM System Success — Or Failure?'
- [38] Malthouse et al., 'Managing Customer Relationships in the Social Media Era'.

- [39] So et al., 'The Role of Customer Engagement in Building Consumer Loyalty to Tourism Brands'.
- [40] '2021 SMB Data Breach Statistics'.
<https://www.verizon.com/business/resources/reports/dbir/2021/smb-data-breaches-deep-dive/>
- [41] West and Pulimood, 'Analysis of Privacy and Security in HTML5 Web Storage'.
- [42] Nielsen and Molich, 'Heuristic Evaluation of User Interfaces'.
- [43] Tan, Liu, and Bishu, 'Web Evaluation'.
- [44] Rahmatulloh, Gunawan, and Nursuwars, 'Performance Comparison of Signed Algorithms on JSON Web Token'.