COMP1004

Computing Practice

2020/2021

Project Title

*Doki-Doki Delight Management System*

|  |  |
| --- | --- |
| GitHub Repository: | <https://github.com/ORG4N/doki-doki-delight-management-system> |

Contents

[Introduction 2](#_Toc72380529)

[Software Development Lifecycle 3](#_Toc72380530)

[Project Description 4](#_Toc72380531)

[Requirements 6](#_Toc72380532)

[Architecture 16](#_Toc72380533)

[Sprint Planning 25](#_Toc72380534)

[Implementation 27](#_Toc72380535)

[Reflection 31](#_Toc72380536)

[Bibliography 32](#_Toc72380537)

# Introduction

This report aims to outline the entire development process of the Single Page Web-Application (SPA) that has been created under the pretext of the business: Doki-Doki Delight, a fictional cafe.

An SPA is an online web application that, instead of reloading entire webpages, dynamically updates a single page’s content in response to user actions. The overall goal of creating an SPA, rather than a traditional website, is to provide a richer user experience.

This aspect of an SPA is significant to Doki-Doki Delight as they require their web-app to be user friendly and provide a welcoming, easy to use service which enables customers to create reservations and do further research into the business.

To coherently document the SPA project, the report will be segmented into the following seven components:

|  |  |
| --- | --- |
| * Software Development Lifecycle * Project description * Requirements * Architecture | * Sprint planning * Implementation * Reflection |

Within each section of this report, the reader will be shown justifications behind the manner in which the SPA project has been carried out, and by the end of the document, the reader will have a thorough understanding of the need for this SPA and how it has been created to be ‘successful’ in accordance with Doki-Doki Delight’s vision.

# Software Development Lifecycle

Long before the existence of software, people were already applying structured approaches to building infrastructure. These models were often linear, wherein certain phases would be completed within a certain order, one by one. And when software became an ongoingly mainstream revelation, the act of planning, designing, and implementing software was no exception to the rigid, linear models. This can be seen in the Waterfall model, which has been in action since the late 20th century (W. Royce, 2021).

Since then, the digital world has progressed with the creation of new models, such as the various Agile methodologies. Nowadays, developers have a variety of different models upon which they can structure their projects around – and this is exactly what the Software Development Lifecycle is – a concept wherein if developers apply a specific structure to their way of development then they can maximise the chance of success.

However, there are a multitude of different types of projects and therefore it is up to the developers, or their sponsors, to consider which methodologies are going to help them achieve the type of success they want to achieve. Success may be: low costs; changeability; scalability; or even, fast production (McConnell, 2014). These are just a few types of successes that a developer may hope to achieve – but not all existing methodologies will work for all of these factors. For instance, the Waterfall model might be perfect for a low cost, low risk project – but it is not suitable for a project where needs and functionality will always be changing to meet the demand of stakeholders (W. Royce, 2021).

But what makes one model different from another model? First, the reader must understand that each model is essentially a framework that is built upon phases/stages. Models generally include the following five stages:

* Requirement Analysis
* Design
* Implementation
* Testing
* Evolution

However, some models may extend beyond five stages – furthermore, each model executes each stage in a different sequence from another model.

The *Doki-Doki Delight Management System* is structured around the Agile framework. Agile is represented by 12 principles (Principles behind the Agile Manifesto, 2021) and together they express the priorities that the model tries to maintain. From these principles one may interpret Agile as the go-to model for customer satisfaction – which is maintained via “continuous delivery of valuable software” (Martin and Martin, 2006), as well as the continuous interaction between developers, stakeholders, and sponsors.

One such factor of Agile that influenced the direction of the project from start to finish was the concept of retrospection. In Agile, the current morale of the team an important factor that a team leader has to consider when coming to decisions (Principles behind the Agile Manifesto, 2021). Individual responsibilities and burdens should not exist within Agile – instead, the team should often reflect on their troubles and “adjust its behaviour” (Martin and Martin, 2006) to overcome issues. By reflecting on the present struggles and developments throughout the project, through writing weekly blog posts, issues we are made known and then able to be worked upon.

Use of the Agile methodology was applied by examining the project’s requirements and then extracting a project backlog from these requirements. From the project backlog, tasks were further decomposed into weekly sprints, which are sets of tasks that should be completed within a set time period (Scrum Guide, 2021). For the first half of the project, outside guidance was sought from bi-weekly meetings with Liz Stewart, who can be interpreted as a project sponsor. Further criticism was received from other staff within the Marketplace Demo where the developer of the app showcased a formal prototype of the app.

Through analysing and applying this feedback to the project, the final deliverable has been constructively built upon to make a useable piece of software that is in line with what real clients would expect. This continuous development of deliverables is in line with what is expected from a project built upon an Agile framework.

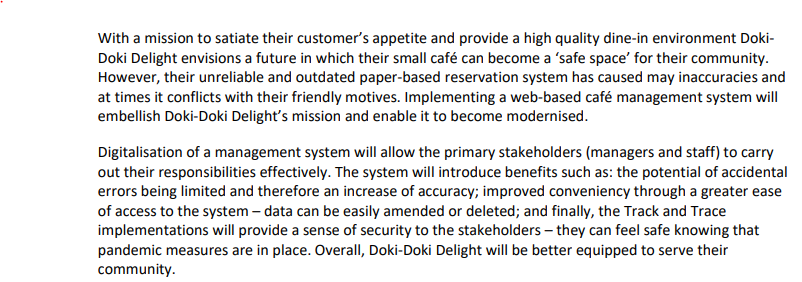
# Project Description

To start this project, a lot of beforehand planning had to be carried out. As such, the following documents were created:

* Project Proposal Document (PID)
* Project Initiation Document

The latter document establishes the vision of Doki-Doki Delight, whilst the former translates this vision into a set of meaningful goals that this project should meet so that the vision can be realised.

Doki-Doki Delight’s vision is as follows (taken from the PID):



Furthermore, additional activities, such as research into requirements and potential issues, formed the basis of Requirements (see page 6). The final stage of planning included designing the structure of the software by creating various UML diagrams. Halfway through the project, these artifacts were reviewed by outside sources and feedback was relayed to the developer of the project, enabling them to reflect on the direction of the project. The project was further reviewed within a Marketplace Demo – allowing the project’s scope to grow as further additional requirements and potential issues were highlighted.

The difference between a ‘good’ café and a ‘bad’ café can ultimately be drawn to the degree of customer satisfaction that the café provides. As a business it is their duty to fulfil their customer’s needs and provide a quality service. Customer satisfaction is a concept that a business can control and maintain by ensuring their services are intuitive and easy to use/access.

Through developing a web-application for Doki-Doki Delight they will be able to grow their business. The app will enable customers to book reservations, of which staff can further manage. Also, as a result of the ongoing pandemic, the application requires Track and Trace functionality to alert customers when they have come into contact with another individual reporting COVID symptoms. The application will also have to be tailored towards the current pandemic regulations (which are constantly changing) to ensure that laws are not breached.

The importance of a web related service for cafés is summarised within a 2013 article (Etemad-Sajadi, 2014) wherein it is stated, “A website … offers [customers] the chance to experience something of its atmosphere, level of service and genre of cuisine”. Used as a tool to gain a customer’s attention and to convince them to visit a business’s premises the webapp can be used as a type of interactive advertisement. Within the same article the author further draws a connection between repeat purchases and ‘highly satisfying’ websites, concluding why a webapp is a modern necessity for businesses.

Furthermore, to ensure that the project can be successfully deployed and maintained I have considered the legalities of the project. Some of these legalities also try to combat social inequalities and maintain ethical practices. The following legislation are significant in relation to developing the web-app:

* The official government guidance for food businesses
* Companies Act 2006
* General Data Protection Regulation
* Equality Act 2010.

The government has restricted (at the time of writing) only to persons travelling in social bubbles of 6 and less and from this group customer contact information must be recorded; likewise, staff schedules should also be recorded (Guidance for food businesses on coronavirus (COVID-19), 2020). As personal information is being collected, the GDPR comes into effect as relates to ethically handling data and being transparent about how data is processed. Likewise, the Companies Act makes it mandatory to supply specific business information on the webapp. Finally, the Equality Act dictates that the webapp should not discriminate against individuals. The application should therefore maintain inclusivity for a variety of different target audiences.

In 2011 the World Health Organisation estimated that near 15% of the human population has some sort of disability (World health organisation, 2011). Within this subset of the population, there are a wide range of disabilities that exit, but the ones that are most significant to consider within this project are mobility and visual disabilities. Without any sort of assistance, these stakeholders are prevented from accessing the service (Schmutz, 2016) as they have no way of interacting with the application.

There are four primary principles in which webpages can be improved for disabled users (Caldwell, 2008):

* Perceivable
* Operable
* Understandable
* Robust

Caldwell’s guidelines suggest that different characteristics of the page can be changed to display content so that it is easier to view, but ways in which users interact with functionality can be different – such as by “making functionality available from a keyboard”.

Overall, the project has been designed help market Doki-Doki Delight and make it easier for customers to create reservations, and whilst developing a solution the other considerations and issues that have been mentioned have not been ignored. Extra attention has been bought to them via the Proposal and Initiation documents and the legalities have been researched. Overall, the core values of usability and inclusivity (Scott, n.d.) have been significant when developing the project and this is reflected within the Requirements (page 6).

# Requirements

Requirements Analysis is a critical stage of every SDLC methodology wherein the developers liaise with their client to form a mutual agreement on what the project is to be. Within this discussion they are certain to discuss the essentials of the project, such as the functionality that should be implemented within the prototype. This phase is significant in ensuring that the project is carried out ‘successfully’. However, as Doki-Doki Delight is its own fabrication, the scale of the café management webapp was singlehandedly evaluated by the developer. This section will present an analysis of the project’s requirements.

The first step in extracting the requirements from the project was by understanding that there are two main types of requirements: functional and non-functional. Functional requirements are categorised by their ability to identify what the prototype should do; whereas non-functional requirements detail the constraints of a prototype and should therefore be measurable.

|  |  |
| --- | --- |
| * Business * Administrative | * User * System |

The above functional requirement categories determine that specific stakeholders should be able to carry out specific tasks (Cox, 2017) – but ‘how’ they might carry out the tasks was intentionally abstracted as this unnecessarily restricts and complicates the analysis phase.

However, the following non-functional requirement classifications (Eriksson, 2012) opened a deeper exploration of the project in terms of statistics, but also made outlining the consideration of external forces – such as laws and regulations.

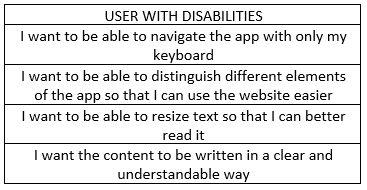
|  |  |  |  |
| --- | --- | --- | --- |
| * Usability * Security | * Readability * Social | * Availability * Ethical | * Performance |

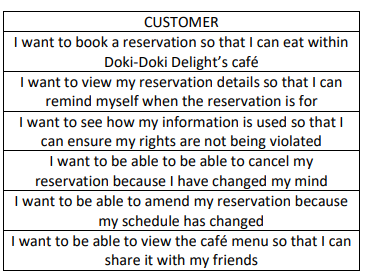
At this stage, there were numerous unrefined requirements that were not in any sort of format. To further extend the analysis of my requirements it was mandatory that they were given structure, and therefore resulting in the usage of a prioritisation technique called MoSCoW.

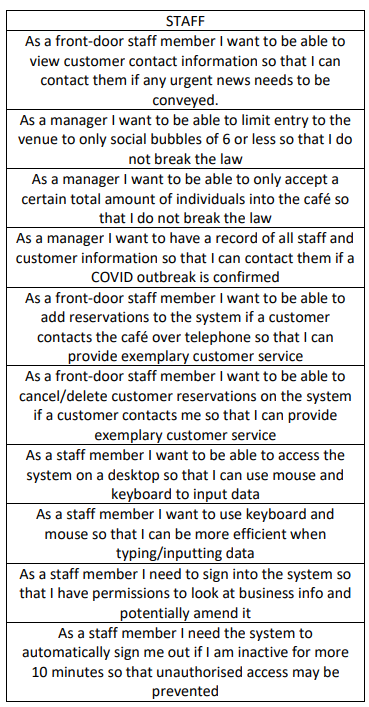
MoSCoW is a method of processing requirements by sorting them into the four tiers: must-have, should-have, could-have, wont-have. The must-have requirements are of the highest priority and these requirements must be satisfied for the project to be successful; the requirements on the opposite side of the spectrum are not essential. Through this process the requirements were refined and ordered – making it much easier to form the user stories.

User stories are concise explanations of the main functionality required from the main users’ perspective (North, n.d.). Determining that the webapp will have two types of users – staff and customers, allowed the creation of stakeholder ‘profiles’. The ‘profiles’ expressed all of the stakeholder’s needs in terms of usage of the webapp – for example, customers want to make reservations with the intent of eating-in at the premises.

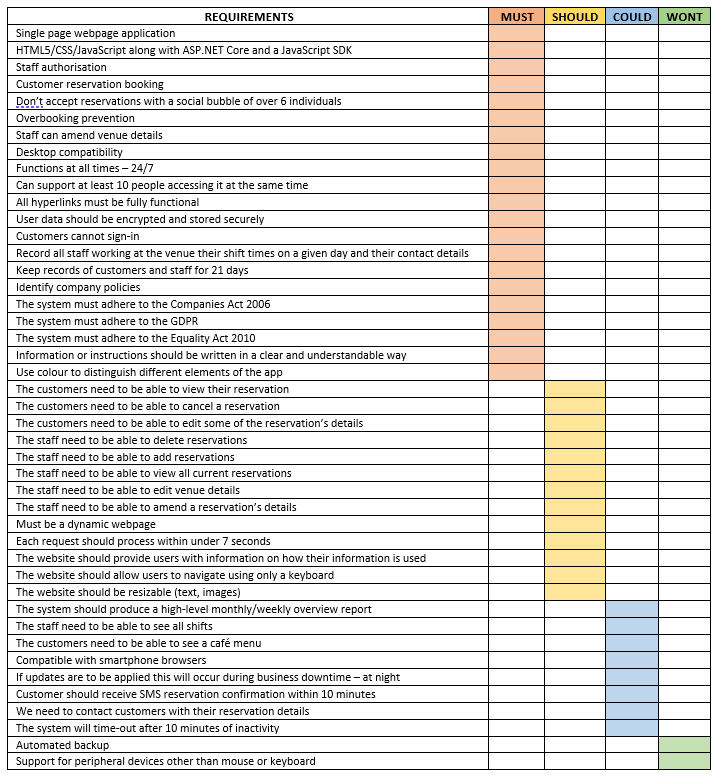
A final, more ambiguous profile has also been created for users with disabilities and it contains some general concerns/needs that a range of different impaired users might require consideration of within the application.



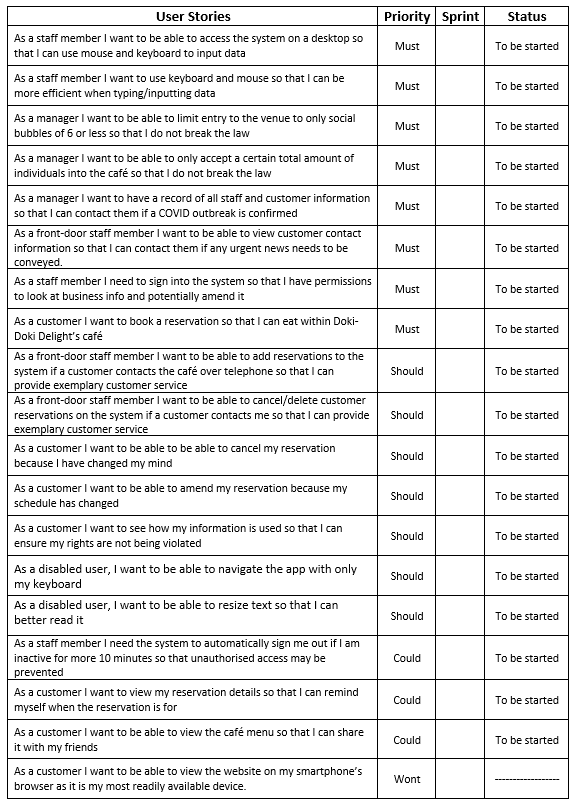




These user stories build upon the refined requirements, and have been directly used within the product backlog, whilst maintaining the priority of each requirement:



(Requirements ordered by priority)

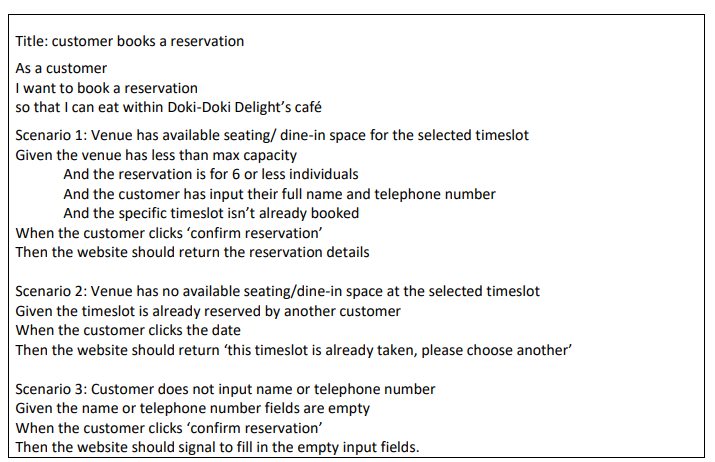


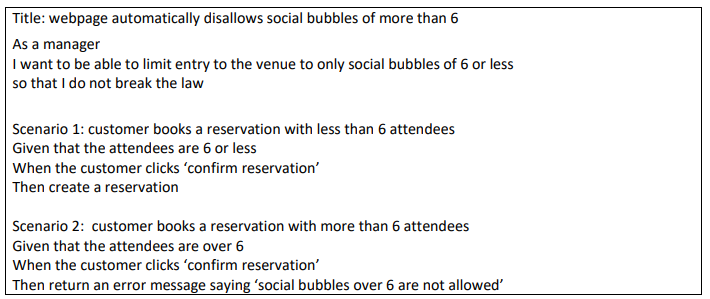
Further processing the user stories, a development tool called Behaviour Driven Development (Hee, 2019) was applied – of which depicts scenarios using the following framework:

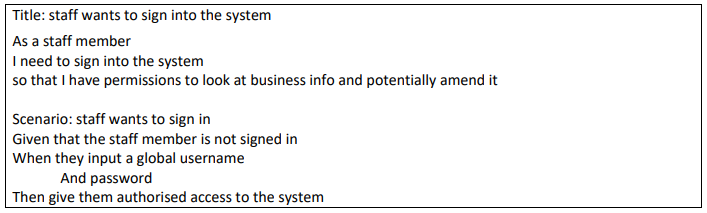
Given…

When…

Then…



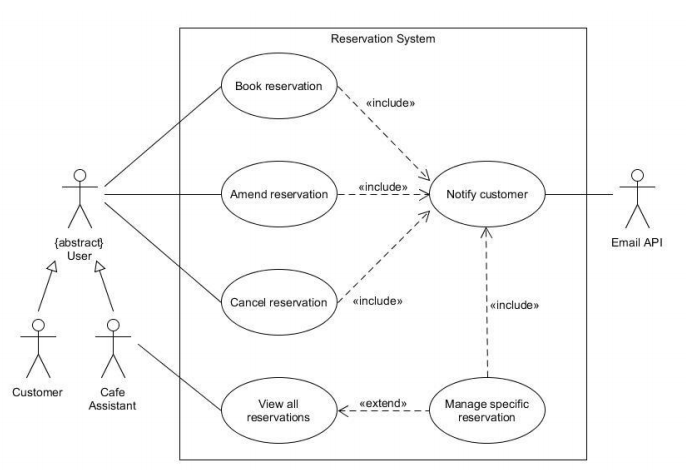


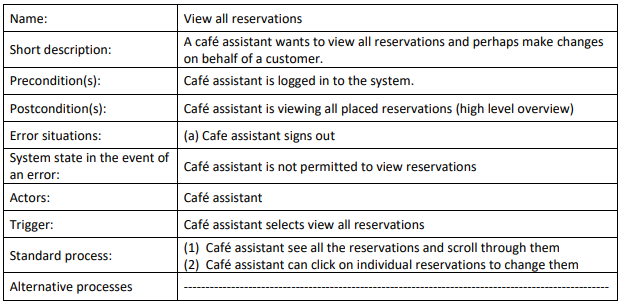


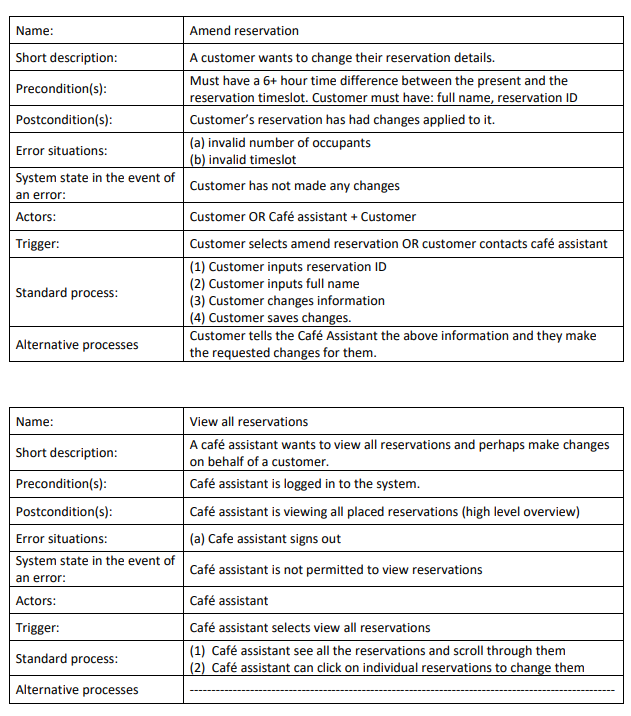
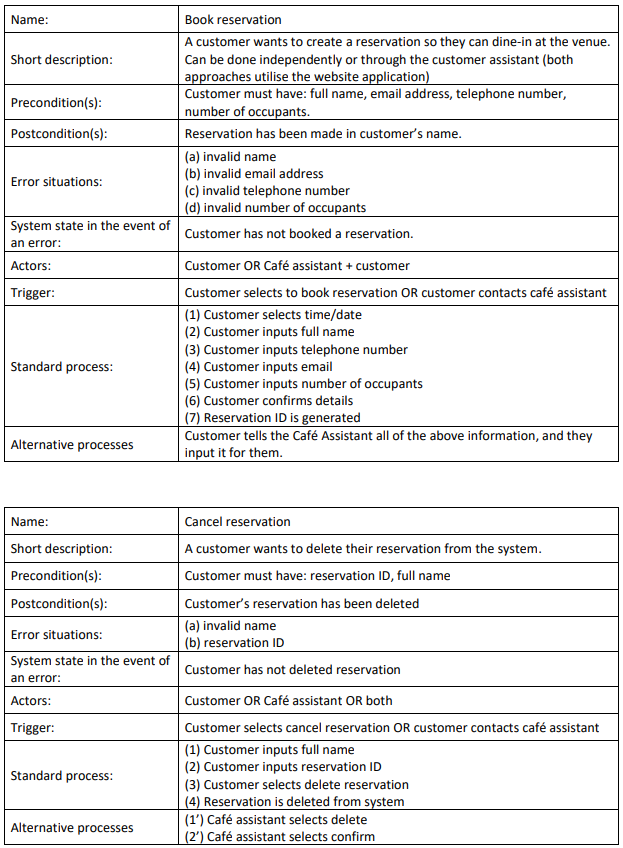
(Some of many BDD illustrations)

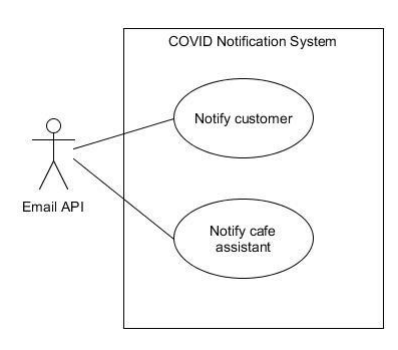
Another tool that has been used within this project to further develop the User Stories was the infamous Use Case diagram. Using UML @ Classroom (Seidl et al., 2014) as a guide, the highest priority user stories were converted into a visual format. A Use Case was created for each of the four critical sections of my project:

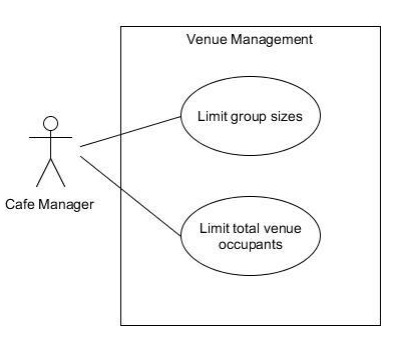
|  |  |
| --- | --- |
| * Reservation booking * COVID track and trace | * Venue management * Notification system |

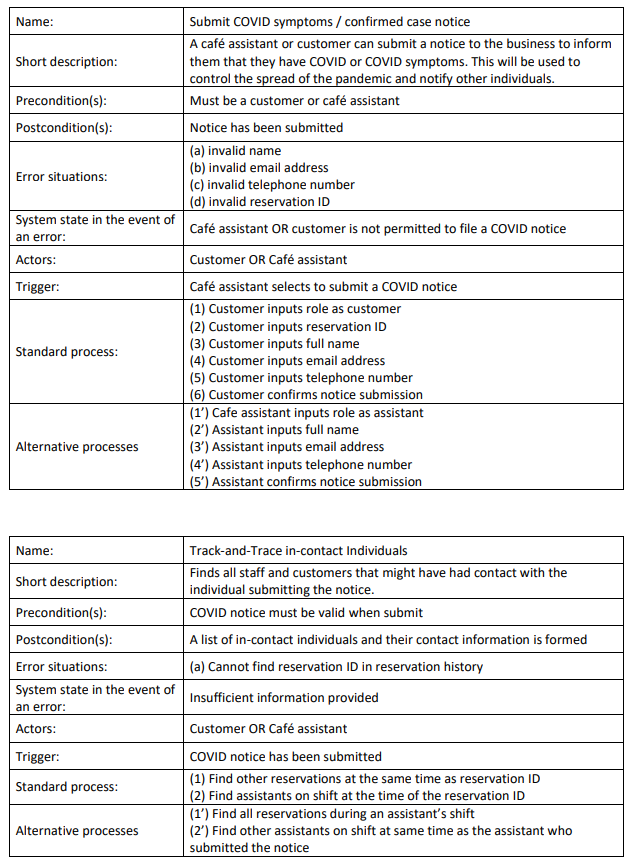
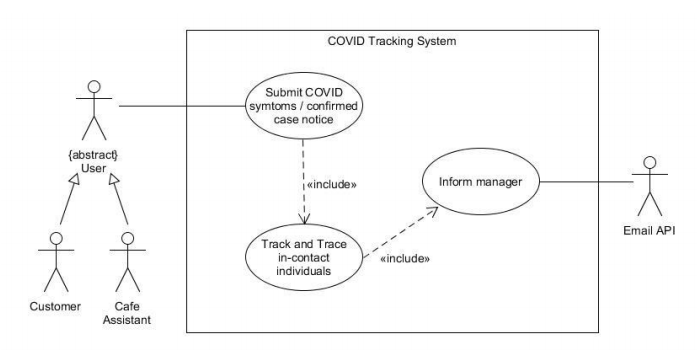












By using user stories as templates for each Use Case diagram there is a portrayal of what each specific role within the system can accomplish by using the app. As stated within UML @ Classroom (Seidl et al., 2014) a Use Case diagram visually illustrates the answer to these questions:

* What is being described?
* Who interacts with the system?
* What can the actors do?

Accompanying each diagram, the reader will find Use Case Descriptions, which intend to offer greater depth and understanding of the relevant diagram.

# Architecture

As previously mentioned, when proceeding throughout the Design stage of my project the developer closely followed the book UML @ Classroom (Seidl et al., 2014) and this enabled them to understand and create the following diagrams:

* Class and Object
* State machine
* Sequence

The developer had decided to create class and object diagrams because it was intended from the start of the project to implement the reservation booking system using the Object-Oriented Paradigm and to achieve this they would use C# through ASP.net. By using classes and objects, the hopeful outcome would be a scalable system – and as the developer has greater experience with C# than JavaScript, they could implement a much more comprehensible system.

These were the initial diagrams that were created before coding the final project, and although they are similar to how the final system has been implemented, they are outdated. These diagrams served as a foundation, but as coding was started, it was realised that these objects could be simplified much more.

|  |
| --- |
|  |

Within the initial diagrams it was notable that in some instances, classes would have ID attributes so that they can be uniquely identified, and this would eventually be used to locate specific objects – such as a customer via their reservation ID.

Alongside each class’s attributes there were also methods, some of which are used to fetch or amend and store data. For example, the manager would have permission to update the venue’s total seating as per the setTotalSeating() method.

The updated class and object diagrams look like this:

Text

Description automatically generated with medium confidenceTable

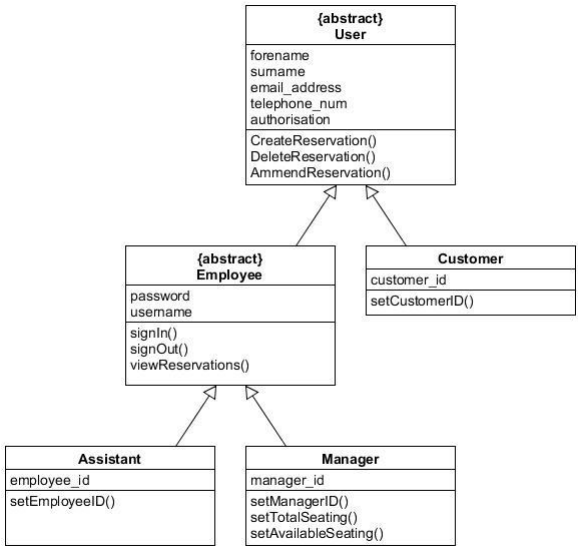
Description automatically generatedText

Description automatically generated

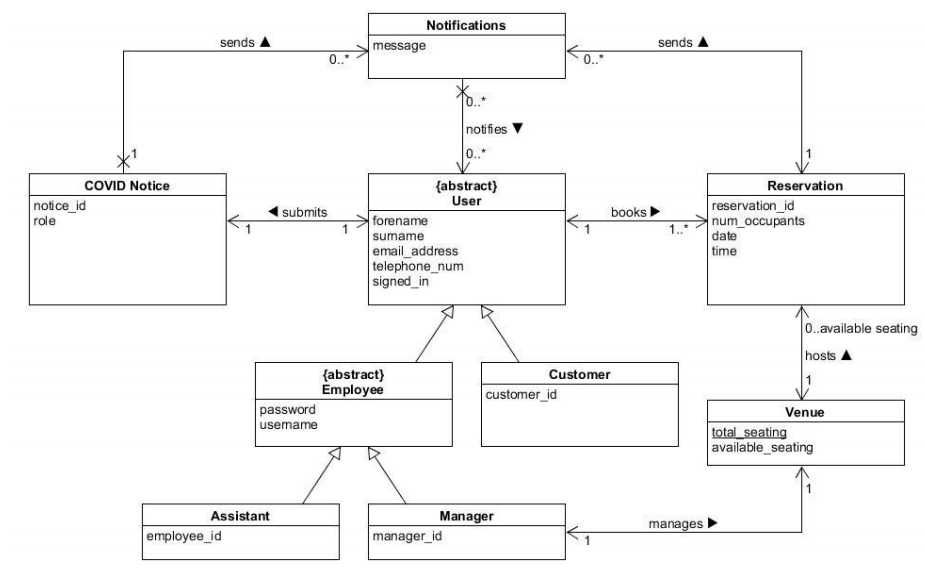
Alongisde the outdated class diagram there is also this outdated inheritance diagram that shows the relationship between the super class “user” and its two inheriting subclasses – Employee and Customer. From observing the diagram the reader can observe that the manager would have an additional set of activities that they can carry out via the webapp – namely, venue management.

However, within the developed solution the code was not structured this way and there was no inheritance between the model classes that were used. This diagram has been kept within this document to show that the intended architecture is different from the currently implemented architecture.

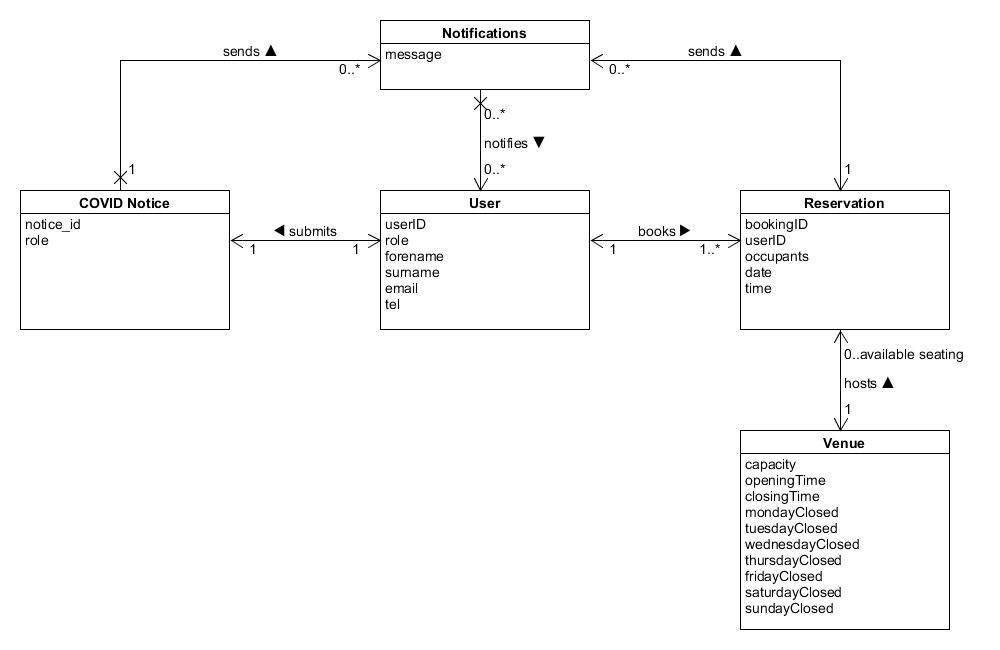
Likewise, because the other diagrams that were created as part of the original architecture are based upon these classes, they too are outdated and this document will provide updated versions.



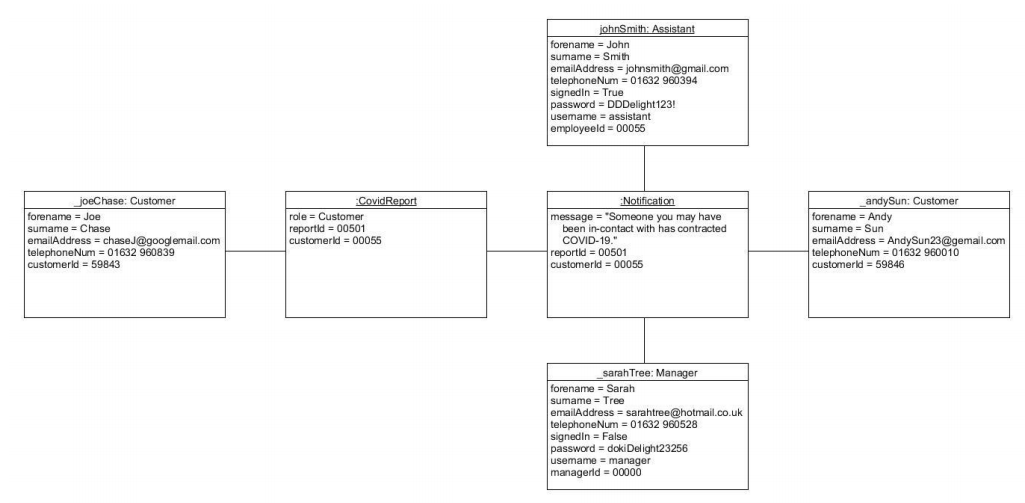
**Outdated:** This diagram conveys a detailed structure of the relationships between each class and its interactions with other classes:

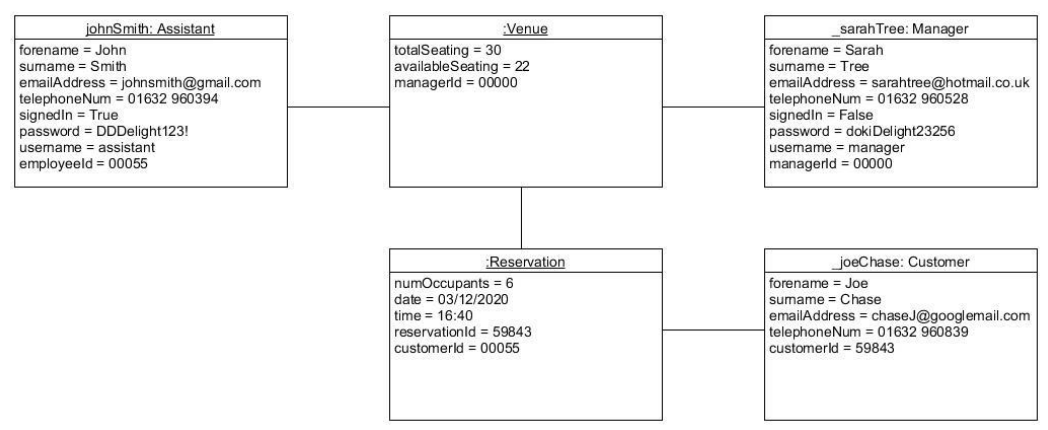


**Updated**: the classes’ attributes have been updated and the subclasses have been removed.



Furthermore, these additional artifacts have been made to show the associations between each class within a given context:

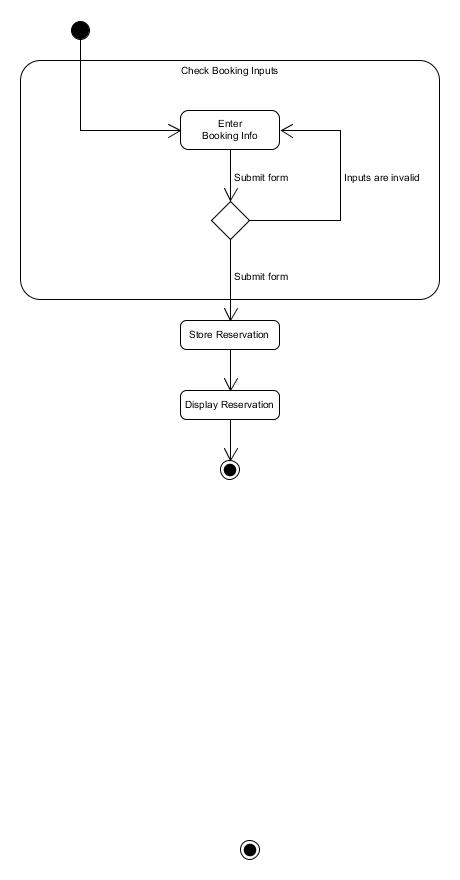


**Scenario:** A customer submits a COVID report and therefore notifications are sent to a collection of individuals that may have been in-contact with them. Details of implementing this tracing algorithm have been abstracted for simplicity.  


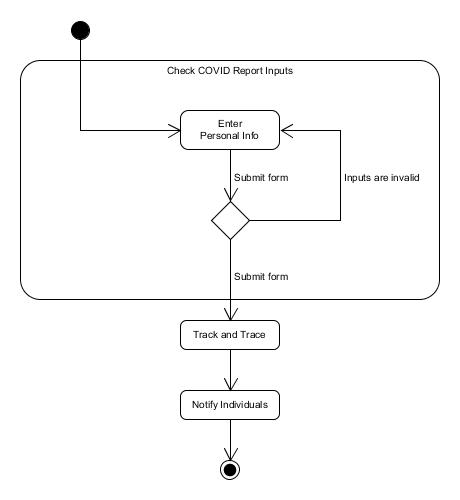
**Scenario:** A reservation can be booked by a Customer, and the Reservation object requires a Venue to be created – the venue will limit the number of reservations being created. A venue will be run by the business’s Manager and Assistant. All users must input their personal information so that they can be contacted

The state machine diagrams convey the responses and behaviours of the system when a trigger is activated.

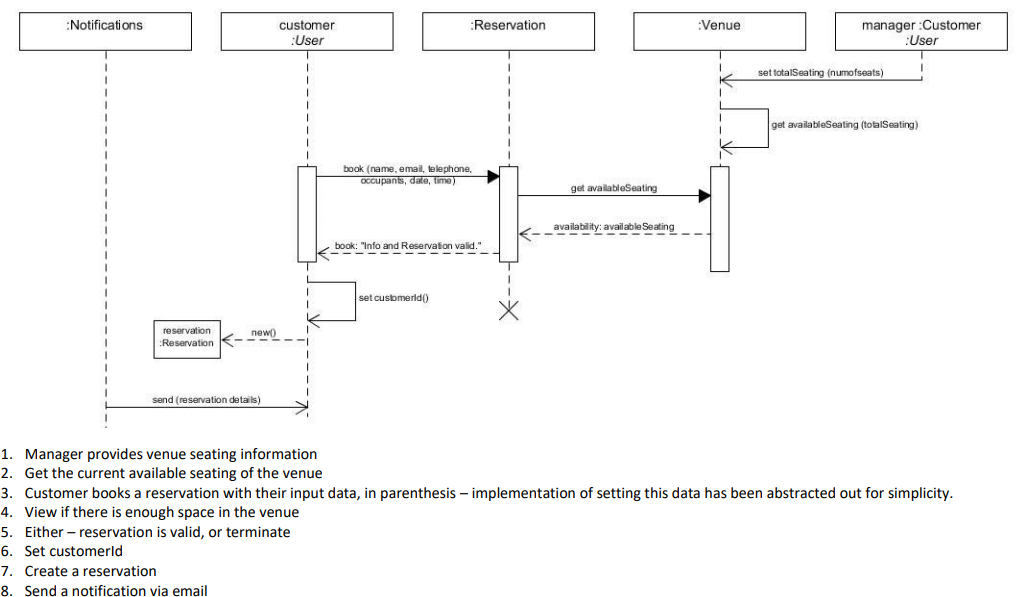
Reservation System:



COVID report system:



Finally, I have a single Sequence diagram that illustrates the event of a customer making a reservation:



Overall, the system has an ASP.NET backend API and reservations will be submitted to this API and multiple different components of the system will be able to access the backend to manipulate the data – for example, the COVID notification system will use the backend to find customer records. The application will also repeatedly request data from the API so that it can be written to the user.

Another example of how the API ties the entire application together is via the venue details and the creating a reservation process – the API will be used to fetch venue details and then evaluate whether the booking should be created or not based on capacity or the time/date the user has selected.

# Sprint Planning

As previously seen within ***Requirements*** (page 6 of this report) a product backlog has been created, and given the properties: sprint, priority, and status. Based on these fields, the backlog has been translated over to Microsoft Planner so that the developer can easily manage what is needed to do each sprint. As the project uses the agile framework, they have sole responsibility of his schedule

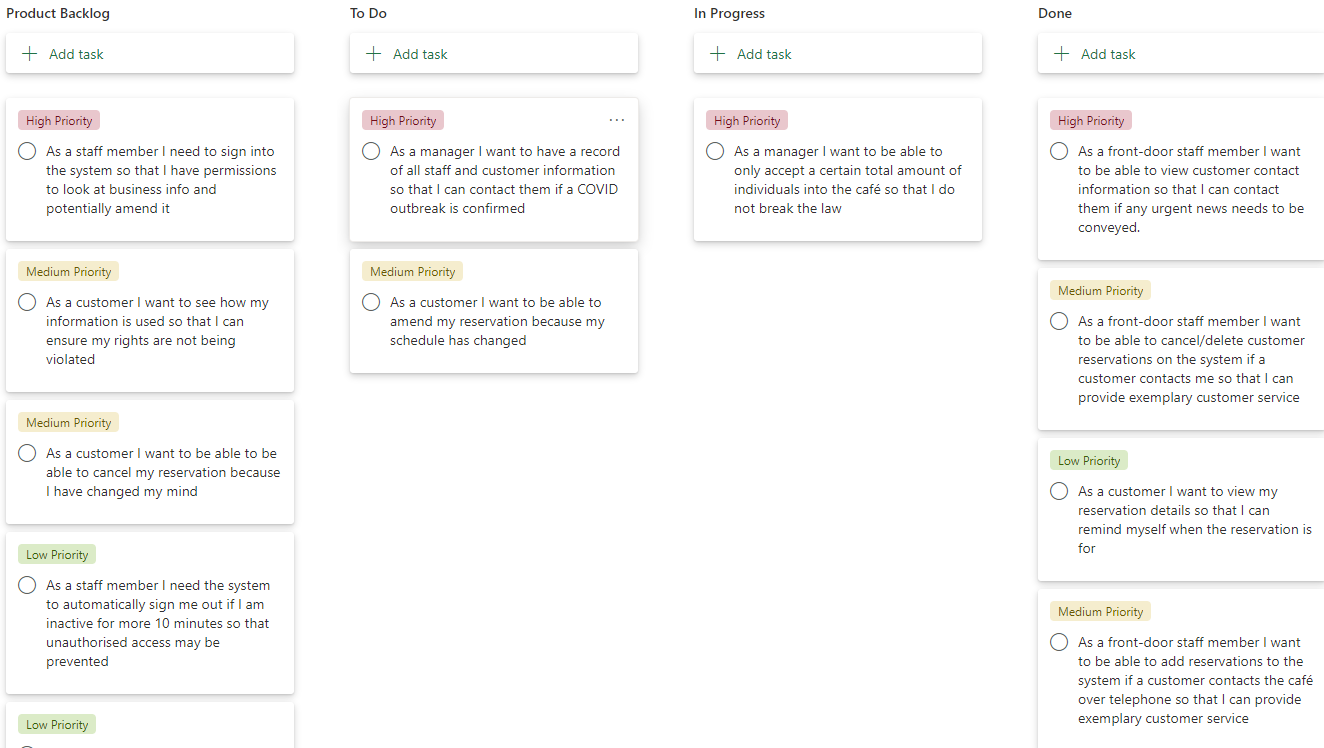
Microsoft Planner link:

<https://tasks.office.com/live.plymouth.ac.uk/Home/PlanViews/KB8DrzSpokCUWjx6hWH765YACcBq?Type=PlanLink&Channel=Link&CreatedTime=637468573538540000>

Throughout this project, sprints have been planned to be worked on across a 2-week basis wherein a review of the deliverables produced will be conducted – and if it were required, the next sprint would have been designated to the same backlog feature.

The planner was laid out using the Kanban structure, wherein the features are stored on the left of the board and as they are worked on, they are adjusted rightwards, until they reach the end and are marked as completed. Going from left to right, the planner is structured like this:

* Product backlog
* To do
* In progress
* Done



The product backlog is formed of user stories and they are ‘tagged’ with priorities that relate to the previous MoSCoW processing. For additional readability, each tier within MoSCoW has been renamed to: high, medium, and low. The wont-have features were intentionally left out to keep the planner very simple.

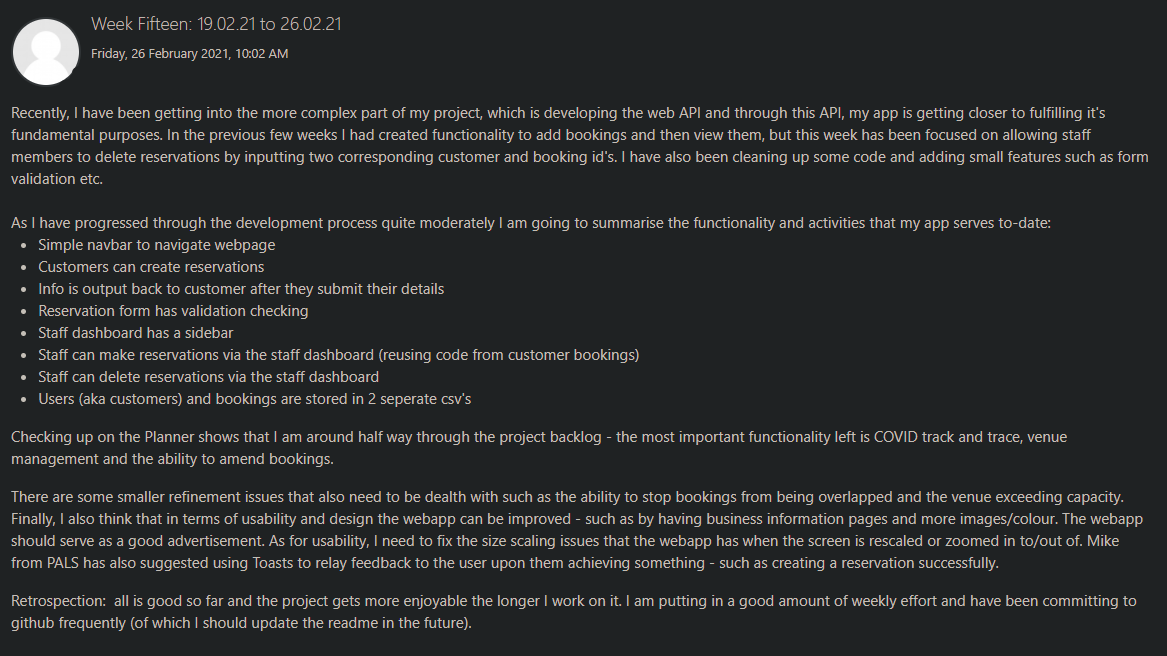
Throughout the project development phase, the process of carrying out a sprint followed this process:

At the start of a sprint, a manageable/achievable amount of user stories are moved to the “To-do” column. Then, the current tasks being worked on are moved to the “In progress” column and when the feature has been implemented it will be moved onto “Done” and reviewed before being marked as complete. If possible, the developer will refer back to the BDD artifacts to test/review each user story and if it fulfils the criteria then they know the feature has been implemented successfully. A new user story will then be fetched from To-do and this will repeat until the sprint is over.

Alongside maintaining the Kanban board (also referred to as the planner), the developer maintained a weekly blog post wherein retrospective activities were carried out to determine how productive they have recently been – but to also highlight achievements. is an important part of the sprints as Agile is about maintaining focus and recognizing weaknesses within a team so that they can be overcome and not inhibit the production of deliverables.

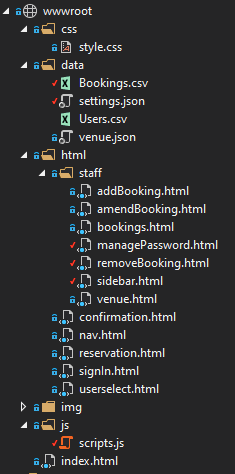
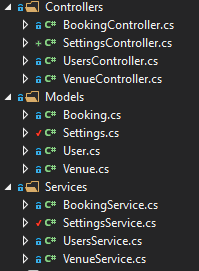
Overall, sprints are a basic way of managing the deliverables being produced and identifying if production is too slow – and the blog reviews work alongside this as they will help find the root cause of issues. From identifying the root cause, the problem can be solved. Throughout the entire project there were many issues that were encountered and without this process of analysing what can be done to overcome the issue, the project would have operated at a much slower pace.

An example of what a weekly blog review is as follows:



# Implementation

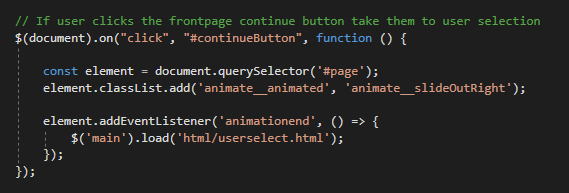
This project can be abstracted into two sections: the frontend and the backend. The frontend of the web application consists of html pages, CSS stylesheets and JavaScript scripts. Meanwhile, the backend has been implemented via ASP.NET, which is used to allow the application to create, store, read, delete, and amend data within text files.

The API is implemented in three factors – Controllers, Models and Services. Models serve as a template for the data being handled, whilst the Controllers act as a gateway between the application and the Services that the API hosts. These services range from read/write methods to methods that generate ID’s for users and reservations.

Structuring the web-app in this manner allowed for an easy way to store data permanently and was relatively simple to learn. Storing data within .txt and .json files was seen as a better alternative to databases because it was simpler, and the application is not expected to be large scale. Another benefit of structuring the backend like this was the ease of finding errors or problems with the written code, because of the encapsulation and separation of different functionalities. Overall, the code was made manageable and easy to expand.

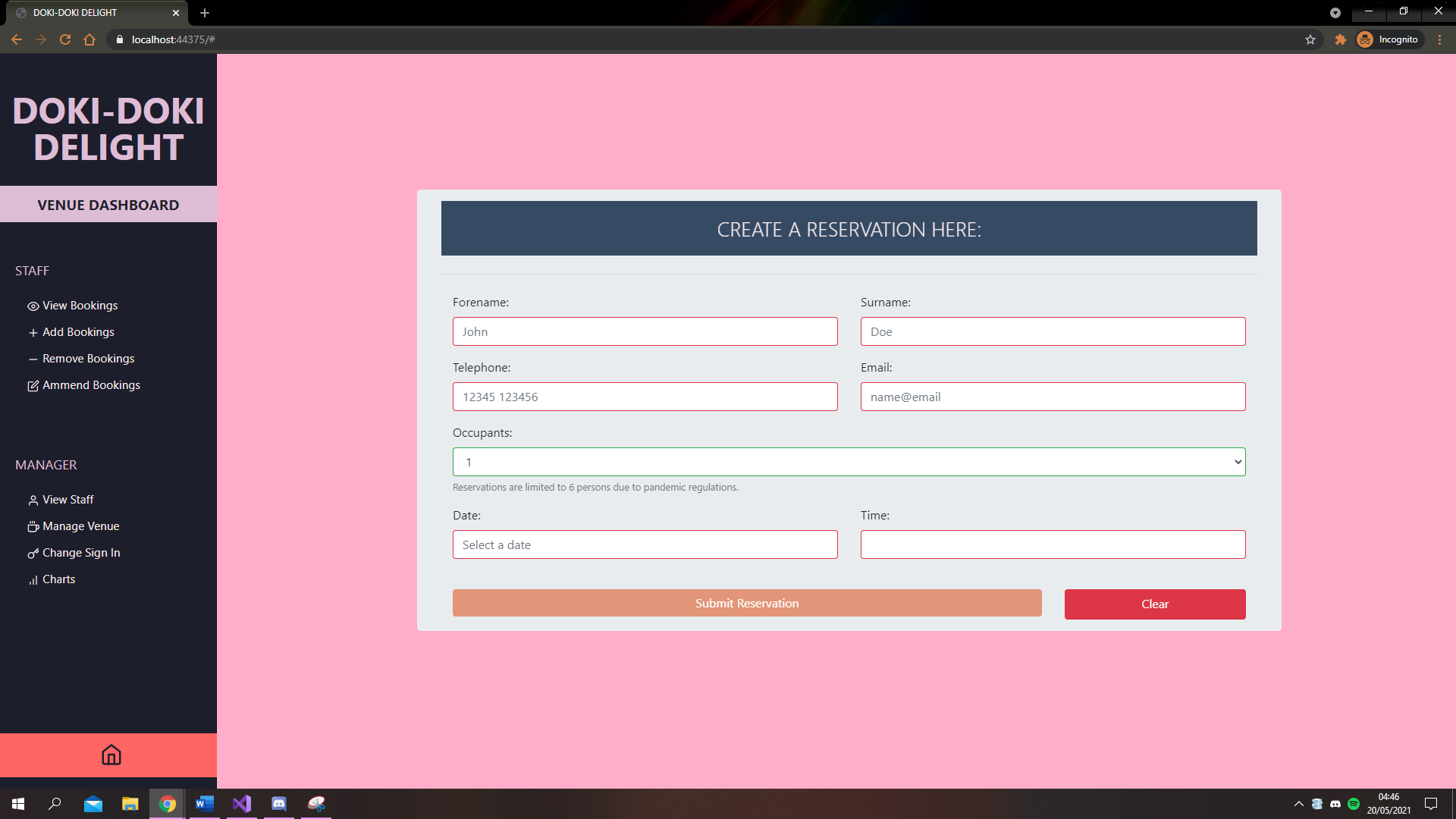
To adhere to the Single Page Application requirement, the application uses JavaScript and jQuery to dynamically load new content onto the index.html page. The html of each different aspect of the web app has been separated into their own html pages so that they are each easy to style and so that issues are easy to find. The code for loading the contents of a different html page into index.html can be seen below:



**Core Features of the Program:**

As a café reservation and management system, the core functionalities of this application were to be able to create reservations and be able to manage them. Another core focus of the application was the COVID-19 Track and Trace functionality – however, this has yet to be implemented due to a greater focus being spent on the development of the hospitality side of the site.

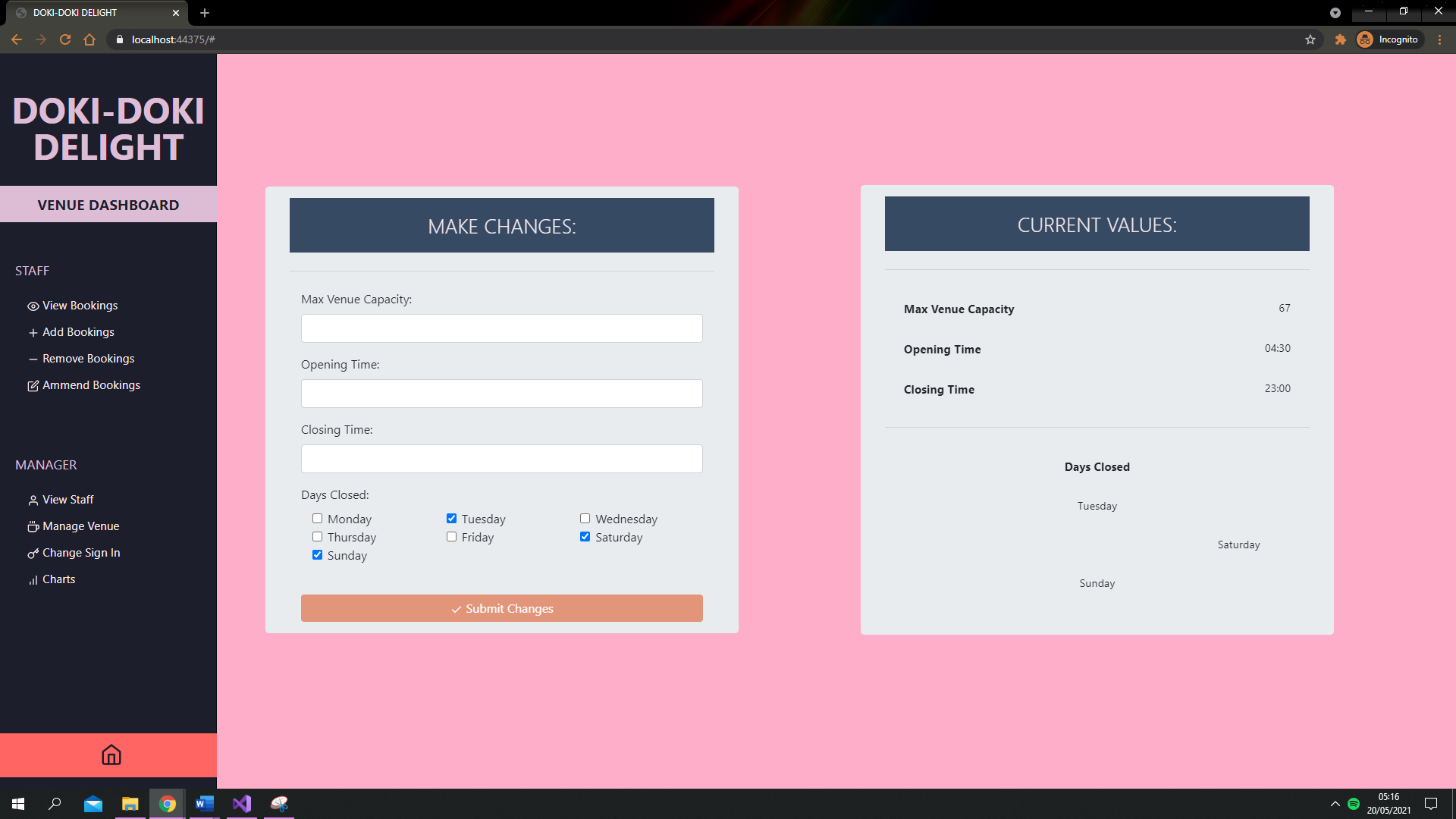
Reservations are made by submitting data via forms, which are validated and POSTed to the API which then records them in a txt file. These reservations can be further amended or deleted by the PUT and DELETE html request methods. In these cases, the application requests the user’s user ID and booking ID.



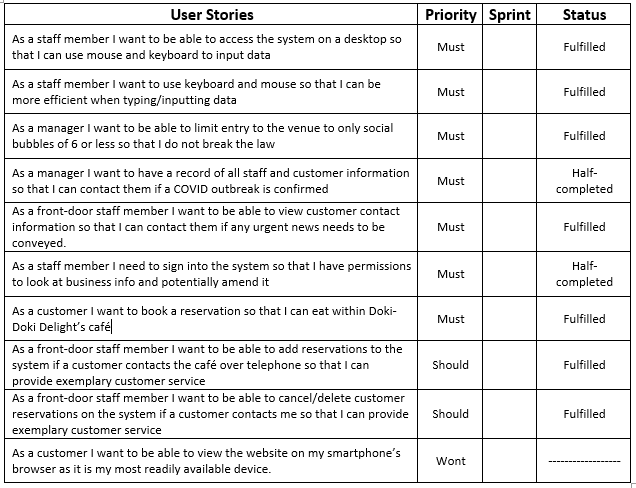
The reservation features were implemented to provide easy access for customers to the café – by allowing reservations to be created through the website, Doki-Doki Delight can potentially draw in new customers and offer them an easy way to eat out. Taking usability into consideration, the app uses colour to distinguish valid input fields from invalid ones (as seen in the red and green highlighting of the fields in the image above).

Colour has also been used within buttons to distinguish buttons with an undesirable affect from buttons that achieve something – such as submitting the form.

Other than reservations, the staff dashboard of the webapp also has a section where the venue details can be managed, and days open/closed can be toggled off as well as opening and closing times changed. This part of the website determines the possible selections shown to the user within the date and time pickers and is therefore important in running a business as customers should not be able to book reservations on closed days. Therefore, this feature adds redundancy.

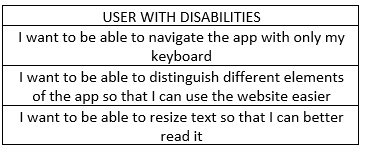


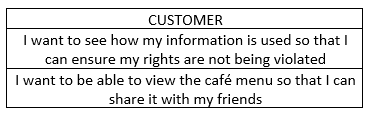
This table shows all user story requirements that have been completed or have been started:

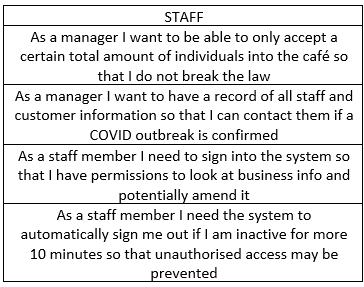


**Features of the program that did not get implemented:**

Unfortunately, not all of the User Stories got implemented within the final deliverable due to the final deadline closing in quicker than the developer planned. The user stories that did not get implemented are as follows:







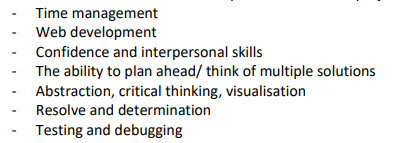
By observing the above user stories, it can be seen that a lot of quality-of-life features were not implemented – as well as staff authorisation as it was not a crucial part of the app – it did not offer any advancement towards Doki-Doki Delight’s vision. Another feature that was not implemented was the maximum total capacity of the café, which would be used to stop reservations being made on specific days if the café were fully booked out.

The main reason for not developing these features was time and bad organisation efforts of the project. In the future, the developer of this project has learnt to use strict planning tools such as Gantt charts to keep themselves on track and to break tasks down into time slots.

# Reflection

Since the submission of the Interim report and the presentation at the Marketplace Demo, the main software deliverable has been developed, and as highlighted within the Interim report, the project was of a manageable scale. Advancements were made in areas of which the developer behind this project had little or no experience within – such as tying in a front-end application with a backend API or using JavaScript to make a website dynamic and functional.

Within the planning stages, the developer outlined within the Project Proposal document that the following skills were necessary to develop to be able to carry out the project:



Within this list there are both soft and hard skills; and by the end of the project the developer had certainly started to train most of these skills. Management skills were displayed via the planning of sprints; development skills were enhanced by experimenting with code and researching existing solutions and adapting them; interpersonal skills were shown within the Marketplace Demo and through interaction with peers.

Through thinking of the system as a whole, as well as individual processes/components, a display of critical thinking was shown and the ability to visualise what the application should do and then implementing this vision is also something that the developer tried to train their brain into doing. As a result, code was well structured.

Of the taught modules in this first year of study, the most relevant to this project was COMP1000 wherein C# skills were developed – and this applied to the project via the ASP.NET backend which required file read/write functionality that had been developed through the COMP1000 coursework. The ASP.NET backend also used classes as templates to create objects from, which is a concept that this module taught. Without the prior C# knowledge, the API would most likely have suffered from unreadable code.

Through this project the significance of planning the architecture and analysing the requirements has been made abundantly obvious – without these factors, the final deliverable would not have been usable or include the intended functionality. The UML diagrams provided a clear starting point, and although the architecture diverted from these plans, the diagrams help the developer think critically and decompose the project into manageable sizes.

Finally, the retrospective blog posts helped the developer maintain confidence and an orderly schedule as they served as an outlet for stress, but also a way to list accomplishments and express which requirements have been completed, are being worked on, and are due to be started.

One factor that should definitely be considered by the reader, the developer of this project, and other developers for their future projects is that stakeholders and clients should be contacted on a frequent basis. This project had some exposure during the Marketplace Demo and to the developers’ close acquaintances, but as new features were constantly being added or changed, more and more feedback should have also been sought.

Therefore, this report will conclude by expressing that if anything could be changed, there are two activities that would be carried out: (1) creation of a focus group wherein users answer a questionnaire after spending an allocated time with the software, and (2) creation of a risk assessment document wherein all recognised issues and risks would be recorded for future reference within other projects, but to also outline the main issues with this current project.

Overall, the result of this project was satisfactory, and all components have built towards this in some way.

# Bibliography

Agilemanifesto.org. 2021. Principles behind the Agile Manifesto. [online] Available at: <https://agilemanifesto.org/principles.html> [Accessed 16 May 2021].

Caldwell, B., Cooper, M., Reid, L. G., Vanderheiden, G. (2008). Web content accessibility guidelines 2.0, World Wide Web Consortium (W3C) recommendation.

Cox, A., 2017. Business Requirements Vs Functional Requirements? Who Cares?. [online] Netmind. Available at: <https://netmind.net/business-vs-functional-requirements-who-cares/> [Accessed 7 November 2020].

Eriksson, U., 2012. Functional Vs Non-Functional Requirements - Understand The Difference. [online] ReQtest. Available at: <https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/> [Accessed 7 November 2020].

Etemad-Sajadi, R., 2014. The influence of a virtual agent on web-users' desire to visit the company. International Journal of Quality & Reliability Management, 31(4), pp.419-434.

GOV.UK. 2020. Guidance For Food Businesses On Coronavirus (COVID-19). [online] Available at: <https://www.gov.uk/government/publications/covid-19-guidance-for-food-businesses/guidance-for-food-businesses-on-coronavirus-covid-19> [Accessed 7 November 2020].

Hee, D., 2019. Applying BDD Acceptance Criteria In User Stories. [online] ThoughtWorks. Available at: <https://www.thoughtworks.com/insights/blog/applying-bdd-acceptance-criteria-user-stories> [Accessed 8 November 2020].

Martin, R. and Martin, M., 2006. Agile Principles, Patterns, And Practices In C. Prentice Hall.

McConnell, S., 2014. Rapid development. Redmond (Washington): Microsoft Press.

North, D., n.d. What’S In A Story?. [online] Dan North & Associates. Available at: <https://dannorth.net/whats-in-a-story/> [Accessed 8 November 2020].

Schmutz, S., Sonderegger, A. and Sauer, J. (2016) ‘Implementing Recommendations From Web Accessibility Guidelines: Would They Also Provide Benefits to Nondisabled Users’, Human Factors, 58(4), pp. 611–629.

Scott, A., n.d. Ethical Web Development. [online] Ethicalweb.org. Available at: <https://ethicalweb.org/> [Accessed 7 November 2020].

Scrumguides.org. 2021. Scrum Guide. [online] Available at: <https://scrumguides.org/scrum-guide.html> [Accessed 16 May 2021].

Seidl, M., Scholz, M., Huemer, C. and Kappel, G., 2015. UML @ Classroom. Springer.

World health organisation. (2011). World report on disability. Geneva, Switzerland, World Health Organization.

W. Royce, W., 2021. Managing the development of large software systems: concepts and techniques. In: Proceedings of the 9th International Conference on Software Engineering. Washington, DC, USA: IEEE Computer Society Press, pp.328–338.