Project Design & Prototyping

Software Developemt for Business

KINSEY, SCOTT ALEXANDER - s2153712

2022

# Table of Contents

[Table of Contents 1](#_Toc108526344)

[Introduction 2](#_Toc108526345)

[Frontend Design and Prototype 2](#_Toc108526346)

[Considerations 2](#_Toc108526347)

[UI Design and Mock-ups 3](#_Toc108526348)

[Frontend Prototype 6](#_Toc108526349)

[Backend Design and Prototyping 10](#_Toc108526350)

[Considerations 10](#_Toc108526351)

[Backend Design 10](#_Toc108526352)

[Backend Prototype 10](#_Toc108526353)

[Database Design and Configuration 10](#_Toc108526354)

[Figure 1 – Colours and Layout 3](#_Toc108526355)

[Figure 2 – Customer search 3](#_Toc108526356)

[Figure 3 – Create customer form 4](#_Toc108526357)

[Figure 4 – Customer overview 4](#_Toc108526358)

[Figure 5 – Appointment details 5](#_Toc108526359)

[Figure 6 – Create appointment form 5](#_Toc108526360)

[Figure 7 – Prototype on Customer overview page 6](#_Toc108526361)

[Figure 8 – Project structure 6](file:////Users/alex/Library/Mobile%20Documents/com~apple~CloudDocs/University/GCU/Year%201/Intergrated%20Project%201/Project%20Design%20and%20Prototyping.docx#_Toc108526362)

[Figure 9 - State management and API fetches 7](file:////Users/alex/Library/Mobile%20Documents/com~apple~CloudDocs/University/GCU/Year%201/Intergrated%20Project%201/Project%20Design%20and%20Prototyping.docx#_Toc108526363)

[Figure 10 - Examples of links to the API 7](file:////Users/alex/Library/Mobile%20Documents/com~apple~CloudDocs/University/GCU/Year%201/Intergrated%20Project%201/Project%20Design%20and%20Prototyping.docx#_Toc108526364)

# Introduction

The goal of this project is to create a prototype of Concierge. Although the full functionality has been outlined in the requirements analysis, the prototype will be limited in functionality. For this prototype to successfully demonstrate the essence of Concierge it must have the following capabilities –

* Searching the database for a customer
* Creating a new customer where no customer is found by searching
* Form to create a new customer
* Customer overview page with information pulled from the database
* Form to create a new appointment
* Appointment details page with information pulled from the database
* Edit customer form to update customer details
* Edit appointment form to update appointment details

The objectives are to develop skills in information gathering for the creation of business requirements, acquire hands-on experience with development tools used at Banking Group, and further skills in testing and evidencing application reliability.

# Frontend Design and Prototype

## Considerations

When designing the interface for Concierge it has been very important to keep the design like Banking Group’s existing suite of web-application. As the source code for this project is publicly available to view, Banking Group has been unable to authorise access to the internal component library. While this has not impacted the usability of the prototype, it has meant all UI components have had to be created, designed, and implemented by hand adding to the production time of the prototype. Colours, page layout, and typography are all in keeping with Banking Group’s existing branding guidelines (fonts which are unavailable for public access have been substituted).

The frontend has been designed in Adobe XD. This application was chosen because of the powerful tools to create high-fidelity mock-ups and demonstrate the flow between pages. The design can be created at a 1:1 scale so that component sizes can be copied directly into the code during implementation.

The prototype has been developed in JavaScript - React using Visual Studio Code(VS Code). This is because React is the dominant library used in Banking Group. One notable package has also been used in addition to React - styled-components. This is to mimic the internal component library used at Banking Group. Styled-components will also allow parts of the interface to be developed once and reused across the codebase, helping to speed up the development process.

## UI Design and Mock-ups

A picture containing chart

Description automatically generated

Figure 1 – Colours and Layout

Figure 1 shows the colour pallet and column layout of Concierge - firstly different levels of black to be used by text, whites for page background and component background, call to action, and two information modal colours. The chosen colours are in keeping with Banking Group’s branding. The column layout will help split content across the page while allowing the design to be responsive to different window sizes.

Graphical user interface, application, website

Description automatically generated

Figure 2 – Customer search

All the different possible outcomes of the customer search page are visible in figure 2. Firstly, when the page is reached and no search has been performed, a search is performed and no customer is found, and when customer records match the search. An information modal at the top of the results will keep the user informed of the results or if any errors occur. To refine a search further the first name and postcode can also be inputted. Links on each of the result cards will take the user to the related customer overview page.

**Graphical user interface, application

Description automatically generated**

Figure 3 – Create customer form

The create customer form is represented in Figure 3. The form is split into two sections. Why the customer is calling, and the customer’s details required. Although the design of the form initially asks the user why the customer is calling, this has later been removed and reserved only for the create appointment form. As the prototype is to demonstrate the functionality of Concierge, it is likely in the future the customer details portion of the form would be broken down into smaller pages, possibly requiring more information.

Graphical user interface, application

Description automatically generated

Figure 4 – Customer overview

There are two versions of the Customer overview page shown in Figure 4. One where no appointments are present. Here a modal will instruct the user that the customer has no appointments. Alternatively, the user will be presented with a list of appointments split by past and upcoming appointments. From this page, the user can edit the customer’s details, view an existing appointment’s details, or create a new appointment.

Graphical user interface, application

Description automatically generated

Figure 5 – Appointment details

Figure 5 shows the Appointment details page. The user can only perform 3 actions from this page. Edit the appointment, edit the customer, and navigate back to Customer overview.

Graphical user interface, application

Description automatically generated

Figure 6 – Create appointment form

The create appointment form, shown in figure 6, is made of two pages – Nature of the call and Appointment details. The Nature of the call page allows the user to select the kind of service the customer requires.

## Frontend Prototype

Graphical user interface, application, website

Description automatically generated

Figure 7 – Prototype on Customer overview page

A picture containing text

Description automatically generatedThe frontend of this application has been created in JavaScript React using VS Code. Some features of the design have been omitted due to the complexity of implementation and time allocated to create the application. The codebase has been split into multiple components which are responsible for each page, which can be seen in figure 8. These are comprised of a folder with the component’s name and an ‘index.js’ file containing the code. Further components can be found in these. For example, ‘Search’ requires 2 further components - the search form and search results. A ‘common’ folder has also been used for components that appear in multiple locations of the application, such as the information/warning modal (depicted in figures 2 and 4 of the UI design), and helper files for commonly used functions, such as a function to correct the case of text. Also depicted are folders which contain all the styles (styled components), images, and services. Services act as the bridge between the front and backend. The ‘App.js’ file is the entry point of the application, which handles the navigation and rendering of all components. While this structure has adequate performance and components are easy to identify, it would be envisioned that some reorganisation of folders and files may take place as the application grows in functionality.

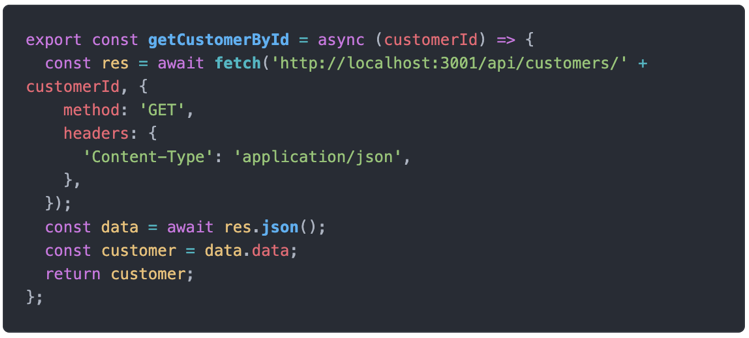
*Figure 8 – Project structure*

Text

Description automatically generatedThe codebase for this protype has been continuously iterated and developed upon, from initial setup to final prototype. During the development process, a major issue was identified in the way customer/appointment data is fetched within components. The issue was causing elements of the UI to render multiple times and occasionally render in unexpected ways. It also prevented the implementation of ‘Edit customer’ and ‘Edit appointment’ flows. A decision was taken to refactor the codebase to resolve these issues. Instead of data being fetched within each page, all data will be fetched at the root level of the application and passed down to all components/pages. Figure 9 depicts the implementation of the refactored solution.

Figure 9 - State management and API fetches

Figure 10 represents examples of how the frontend links to the backend:



The first function is a “POST” request to the API’s endpoint with the details of the customer search as the HTML body. This returns an array of found customers to the UI.

The second function is a “GET” request to the API’s endpoint with “customerId” appended to the end of the URL. This returns a customer object to the UI.

Many more functions exist in this file which all interact with the backend to retrieve, send, update, and delete data. This allows for the full functionality of the database to be used and access from the frontend.

Figure 10 - Examples of links to the API

Text, timeline

Description automatically generated

The application is broken down into the main pages which have each been coded as components. These components are accessed by using different URLs. URLs are generated by buttons within the app.

Text

Description automatically generatedAs mentioned above, Banking Group were unable to provide access to the internal component library, so custom styled components were made using styled-components. This allows for easy implementation of reusable UI components. Components also accept parameters to allow for customisation without creating duplication in the code, such as “full” or “right” to change the look or layout of the component (shown in figure 11). As the components are defined once all instances of them can be updated simultaneously if changes in styling is required.

Also depicted in figure 11 are global styles. Global styles are predefined values that can be reused in styled components or any other CSS. These are particularly useful for colours and sizes of UI elements.

Figure 11 - Styled buttons

# Backend Design and Prototyping

## Considerations

The backend of Concierge is designed to run locally. It has been created using Node 16, Express, and SQLite3. Each endpoint of the backend is created using Express. The SQLite3 package allows the code to communicate and interact with the SQLite database. These packages have been chosen as they are widely used in Backing Group. During the development of this prototype nodemon has also been used to run the backend and instantly reflect changes in the codebase. The database does basic checking of data, but more validation of data would be needed if this application is further developed. For the purposes of this prototype only valid data is used.

If this project was to be further developed, it would be envisioned that this backend would be implemented onto Google Cloud Platform (GCP). This process would require the frontend to be refactored to target new endpoints of the API on GCP.

## Backend Prototype

SQL queries

API setup

# Database Design

SQLite3 has been chosen for the implementation of the database. Concierge requires a relational database in which powerful SQL queries can be used to retrieve data from multiple tables at once. The database is made of three tables – Customers, Appointments, and Consultants.

Graphical user interface, diagram, application, table, Excel

Description automatically generated

Figure X shows the full schema for the database. The relationship between customers and appointments is many to many, and the relationship between consultants and appointments is many to many. It was also decided that lower camel case would be used for the column names of the database, which is not typical naming convention. Typically, the database columns would use snake case and the frontend camel case. This removes the need to implement a data mapper between the front and backend as all related data uses the same name. While the current schema is not an exhaustive list of all the data that Banking Group may require, it does allow for an accurate reflection of how the fully fledge application will perform and interact with the database.

Talk about scripts to create database

Talk about scripts to create dummy data – erase databes if it gets messy.