ZIMTEC Web Developers

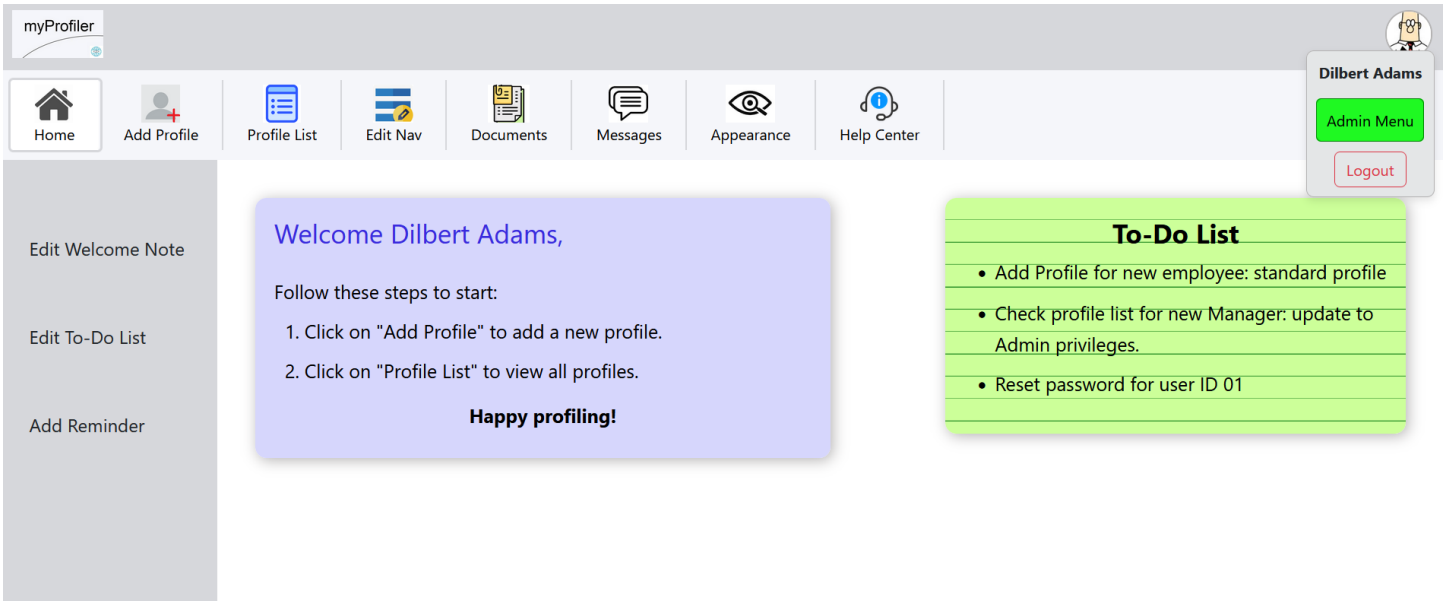
myProfiler

May 2026

myProfiler

Customizable Profile

Web Application



Software Requirements Specification version 1.0 (SRS v1)

Table of Contents

1. Introduction

ZIMTEC's myProfiler, is a web application that allows you to customize a profile page according to your specific needs, share locations, maps, documents and messages in a single workspace. ZIMTEC's myProfiler is easily accessible, simply use your web browser to login from anywhere and view the necessary details of a profile, documents and messages in a single workspace to get your project done efficiently.

A customizable profiling web application will have many benefits within organizations, in particular, inside specialized sectors such as technical, medical and engineering. Flexible naming conventions and descriptions within very detailed object frameworks that cannot reasonably be assumed or known between different departments can now be shared more easily with greater context in a single workspace.

Please note that myProfiler is a desktop web application and in version 1 there is no support for mobile use, that will be considered for future releases. You can expect these same standards with everyday tools and web applications like Microsoft Word for example.

2. System Design

2.1 Overview

The overview section is only intended to give the reader a brief pan over the web application and will omit functionality specifications until the Functionality section 4, Functionality & Use Cases.

Upon log in, a login screen (See Fig. 1) prompts the user to enter in their email and password. The frontend React component will check the privileges in the database and render the correct interface based on those privileges.

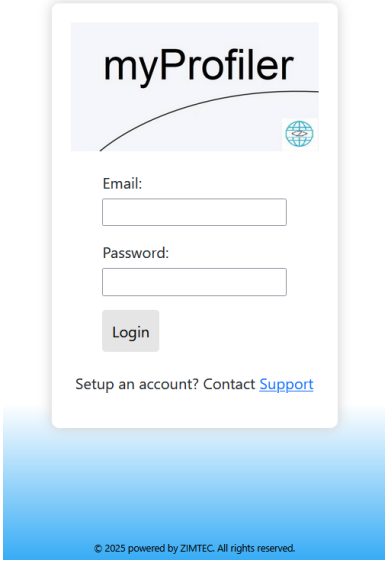


Fig. 1 – Login screen

There are two sets of privileges, Standard and Admin:

- **Standard:** The Admin Button will not be displayed, only basic edits to their own profile may be achieved (See Fig. 2). Other options for Standard users, if the admin user chooses to configure them, will be Documents, Messages and Appearance.
- **Admin:** The Admin Button will be displayed (See Fig. 3) within the User Info area when the logged-in profile picture is clicked. The Admin Button allows access to the Admin Tab (See Fig. 4) which contains all the Admin Menu options. The Admin user customizes the profile for their organization thus for all profile users.

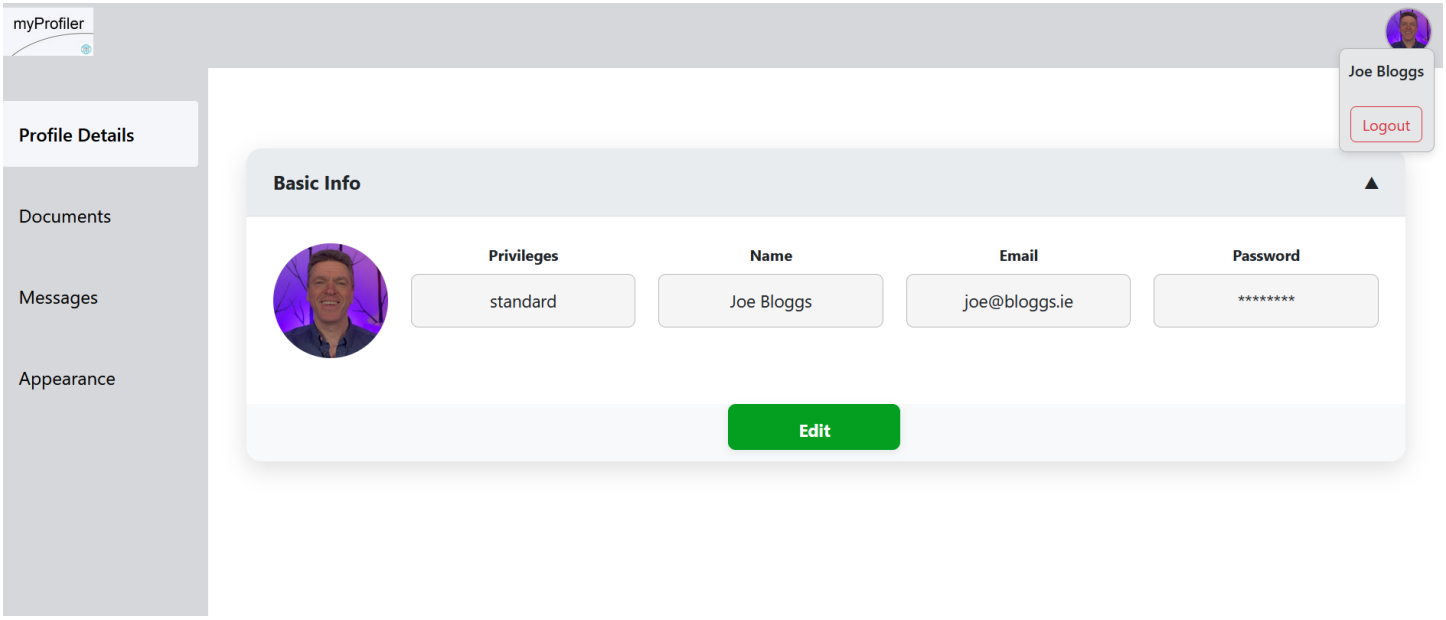


Fig. 2 - Standard Profile

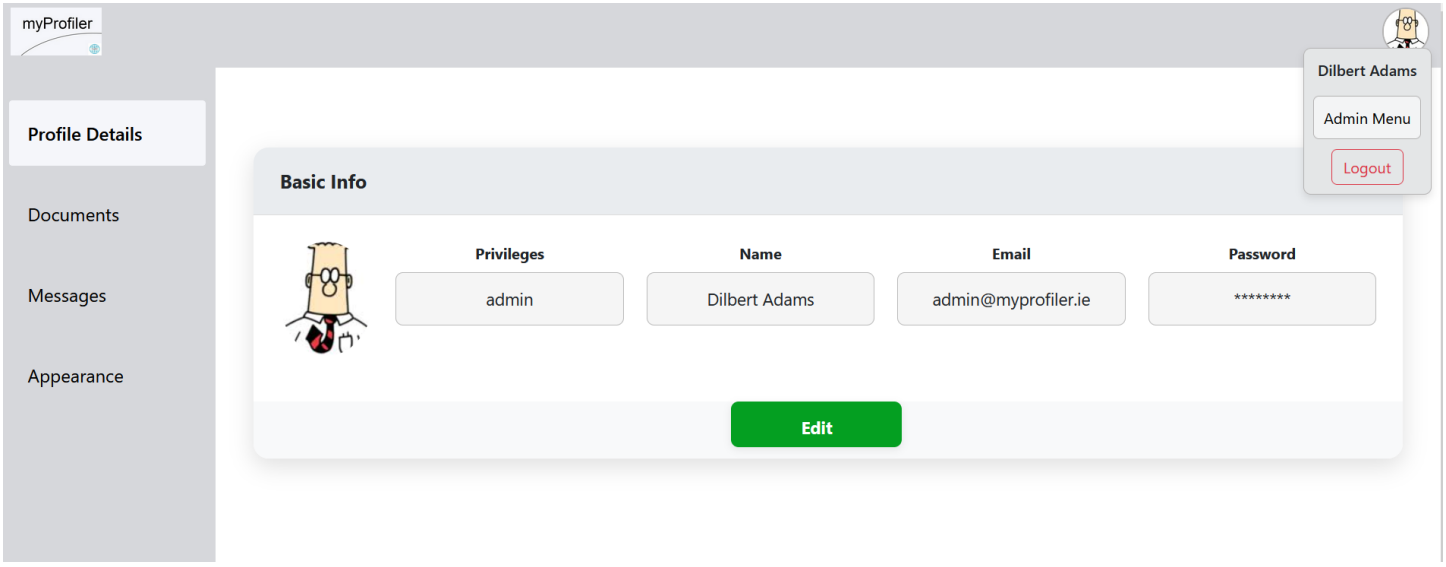


Fig. 3 – Admin Profile

Once the Admin Tab is opened it will display the Admin Menu options for the Admin user. The Left Pane of the Standard view slides off the screen. The Admin Button toggles to green to notify users the Admin Tab and Menu options are open and available.

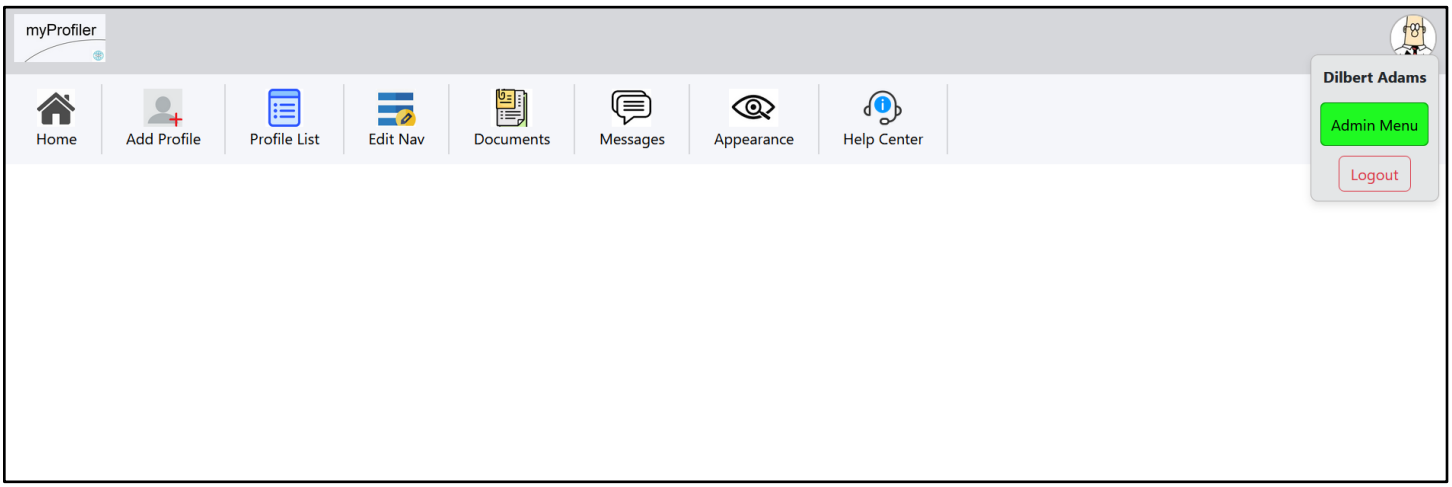


Fig. 4 – Admin Tab & Menu options

2.2 General Layout Design

The general layout comprises of three main areas (See Fig. 5):

1. Nav bar: The Nav Bar contains the Admin Button for Admin users and will open to display the Admin Tab.
2. Left Pane/Sub Menu:
    a. The Left Pane contains standard users' options if the Admin user chooses to display them. The options for Standard users are;
        i. Profile Details: profile details are mandatory and are displayed by default.
        ii. Documents: allows users to share documents.
        iii. Messages: allows users to message one another.
        iv. Appearance: allows users to change the appearance of their profile.
    b. The Left Pane for Admin users with the Admin tab open will display sub-menu options to each of the Admin Tab menu options. The next section, 2.2 Admin Tab Options, will go into further detail about the sub-menu options for each Admin tab menu option.
3. Main Content Area: this is the whitespace which will contain all the components of the web application and is scrollable should any content fall outside the viewport. The rest of the web application is fixed in the viewport

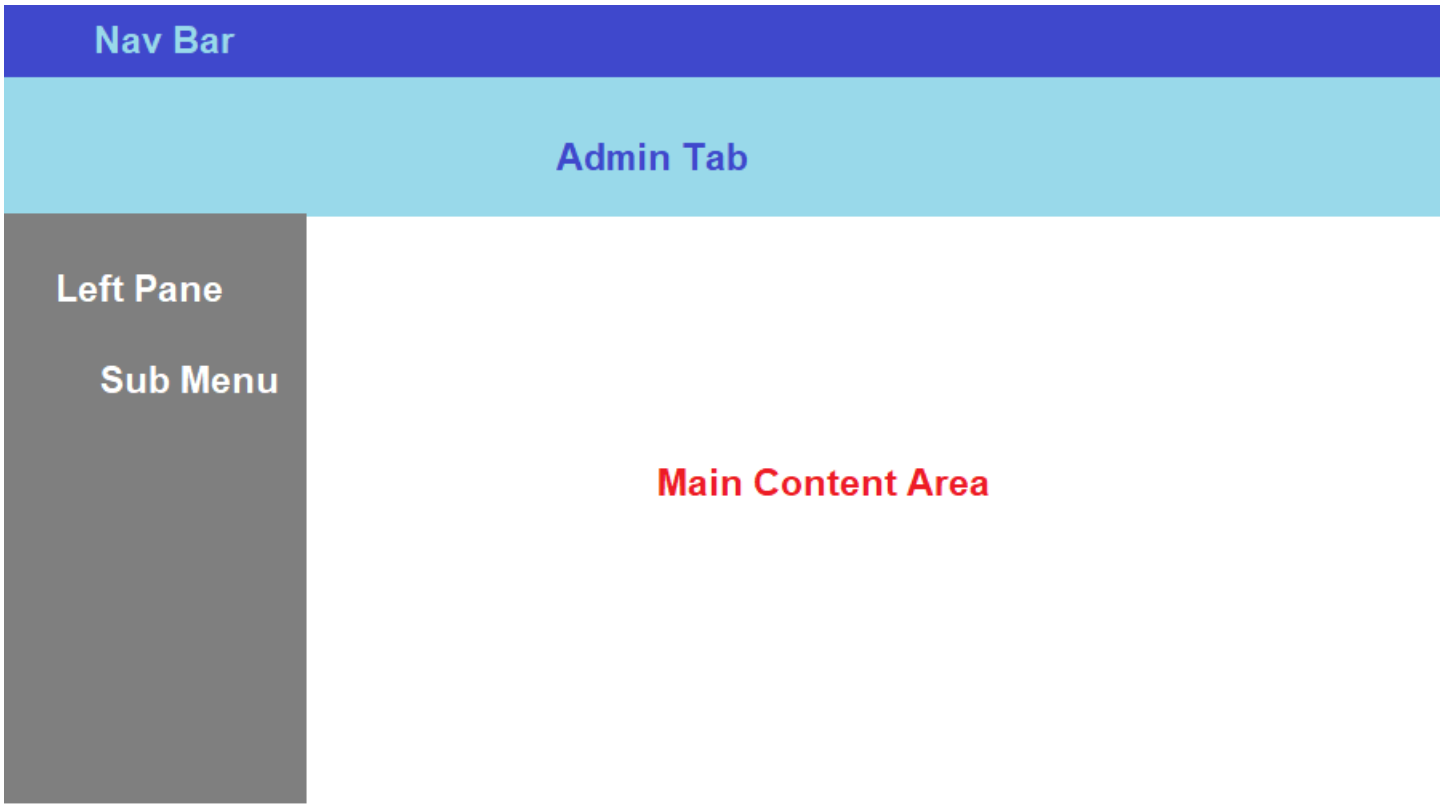


Fig. 5 - Layout Design

2.3 Admin Tab Options

This section will go through the Admin Menu options for an Admin user once logged in. Each time an Admin Menu option is selected, the Left Pane or Nav, will change and display the corresponding Sub-Menu options for the Admin Menu option selected.

2.3.1 Home

The Home page displays a welcome note to the Admin user (See Fig. 6). The Home option has a Sub-Menu with the following options:

- Edit Welcome Note: uses Reacts Quill as a rich text editor to enhance the user's experience and has a default function that will reset the Welcome Note.
- Edit To-Do List: also uses Reacts Quill and has a default function that will reset the To-Do List.
- Add Reminder: uses native text area. The user can select up to three reminders then they are prompted to start a To-Do List if they try to add a fourth.

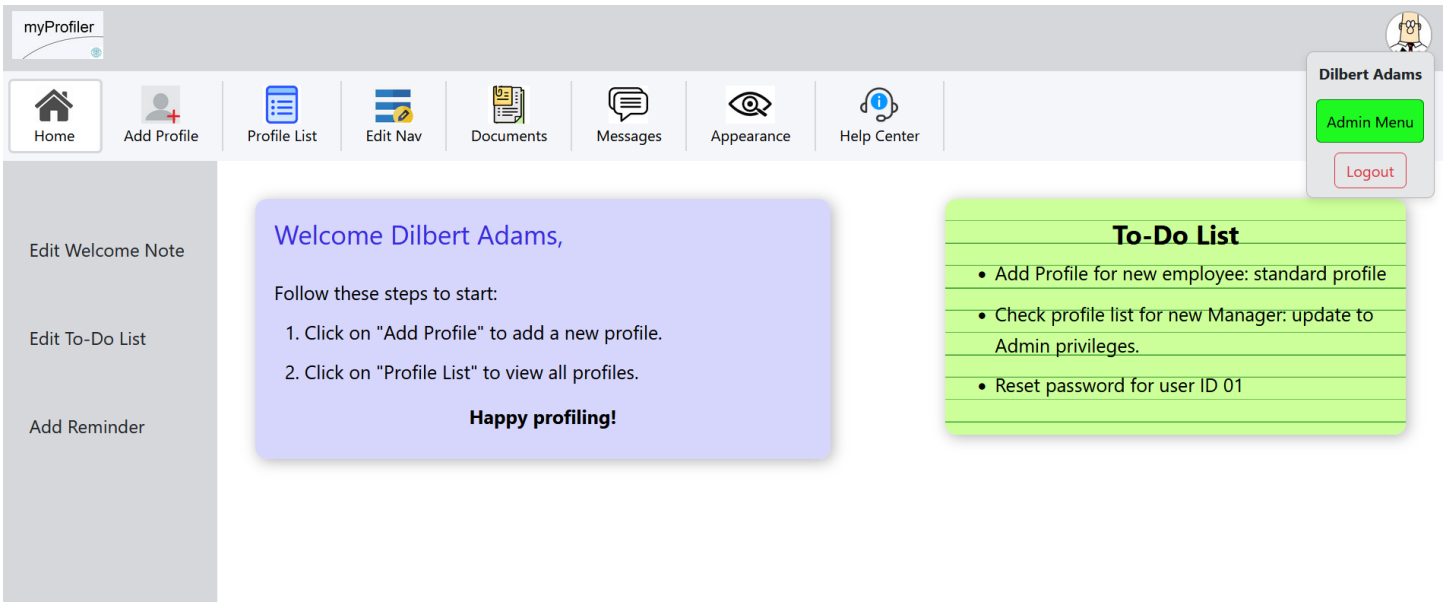


Fig. 6 – Admin Home Page

2.3.2 Add Profile

The Add Profile menu option (See Fig. 7) allows an Admin user to add a new profile. The **Basic Info** accordion holds mandatory information for any given profile. From here, an Admin user can begin to add customization details for their specific organization.

The Add Profile option has the following Sub-Menu options:

- Add Header: opens a new accordion with an editable name.
- Add Label and Box: will add Label and boxes for any given section or accordion,
- Add Description: adds a text field to give more context to a section or accordion or label and box.
- Add Maps

Once a new profile is saved the Admin user is brought to the Profile List menu tab and its corresponding Sub Menu items for the Profile List main Admin Menu option.

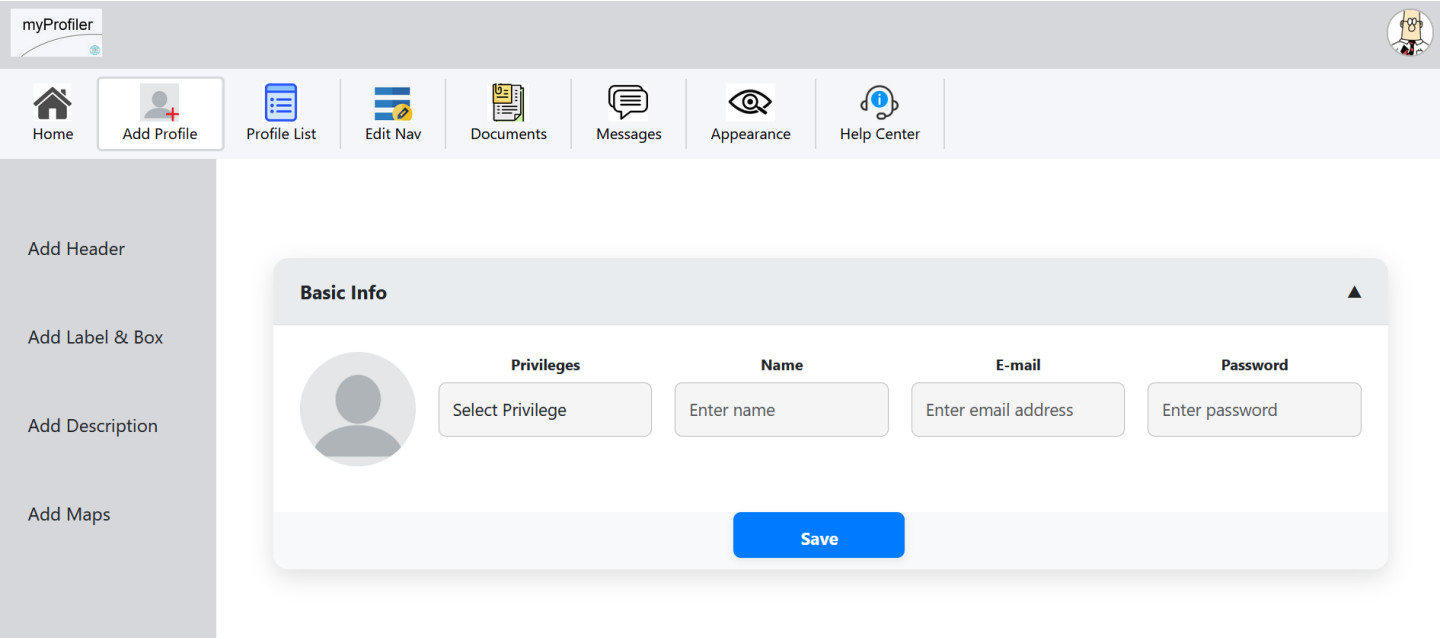


Fig. 7 - Add Profile

2.3.3 Profile List

When the Profile List option is clicked from within the Admin Tab, all profiles in the system will be listed. The Profile List has the following sub-menu options:

- Order by ID #
- Order by A-Z
- Order by Z-A

2.3.4 Edit Nav

The Edit Nav option allows the Admin to Show or Hide the Nav options for both user types, Standard and Admin.

The Profile Details is a mandatory Show. The Edit Nav sub-menu options are:

- Documents
- Messages
- Appearance

2.3.5 Documents

The Documents option allows the organization of documents for all users. The Documents sub-menu options are:

- View as Tiles
- View as List
- View Details

2.3.6 Messages

The Messages option allows the organization of messages for all users. The Messages sub-menu options are:

- Option One
- Option Two
- Option Three

2.3.7 Appearance

The Appearance option allows the organization of the appearance of the web application which can be set individually once global permissions are applied. The Appearance sub-menu options are:

- Theme: **Light** Dark
- Background Picture
- Add Welcome Page

2.3.8 Help Center

The Help Center option provides some useful information for users and developers. An Admin user can also report a bug back to myProfiler developers. The Help Center sub-menu options are:

- Help Pages
- Report a bug
- Developers Guide
- About myProfiler

2.4 System Alerts

The web application helps to guide the user throughout various processes with two assistants prompting the user.

1. **Fifi**: the friendly helpful assistant.
2. **Phil**: the more direct and assertive personality.

3. System Implementation

3.1 Overview

The myProfiler web application implements the Model-View-Controller (MVC) software design pattern. The Model being the Profile data, the View is what the user sees in the browser when accessing the Profile data that the Controller has provided. The Controller being the API's available to execute instructions on the data in the database.

On the browser side (See Fig. 8), the React framework encapsulates the web application while a mySQL database stores the Profile data and the Java Spring Boot framework performs the functions on the Profile data.

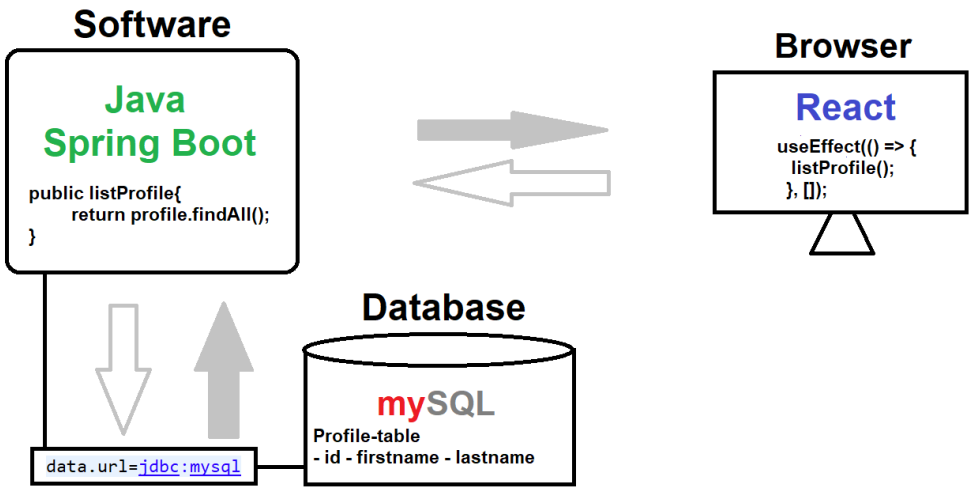


Fig. 8 - Tech mock-up

3.2 Technology Stack Versions

- Java: JDK v22, hosted on Windows 11 OS
- Java: v21 hosted on Railway.com = pom.xml has version 21
    - Spring Boot: v3.3.3, dependencies:
        - Spring Web
        - Spring Data JPA
        - mySQL Driver
- Apache:
    - Tomcat: v10.1.28, hosted on Windows 11 OS
    - Maven: v4.0.0
- MySQL Database: v8.0.39-cll-lve, hosted on Linux OS
- MySQL Database: v9.4 hosted on Railway.com
- React:18.1 hosted on Linux OS
    - Node: v10.8.2
    - NPM: v22.6.0
    - TypeScript: es6

3.3 Demo Details

The myProfiler Web Application is still being developed and is in its early stages, however you are invited along for the journey and demo stable iterations.

Please Book a Demo on our website and we can setup a secure session just for you. Include in your message the **date** between **Monday to Thursday** and what **time** you'd like your session to be.

3,4 System Migration

Migration is a simple 3-step process. Outside the scope of this SRS are port forwarding, router and firewall configurations. The Java app runs on port 8080 and the database 3306 as standard. MyProfiler can be setup on a variety of platforms, servers, Cloud services, etc., for example, Windows, Linux or Railway, Github, etc. Some mySQL control panels require remote access to IP addresses to be added.

Firstly, deploy the Java Application into your JRE environment. The **Environment Variables** and **CORS** will need to be set:

- **DB_USERNAME:** example - root
- **DB_PASSWORD:** example – weqWE"£$£$qwe

- **DB_URL**: typical example - jdbc:mysql://your-domain.com:3306 as the Java App uses the Springboot JDBC driver.
- **Controller.java class**: Edit the CORS so only your domains for development and production sites are the only ones to access the app. Example - `@CrossOrigin(origins = {"http://localhost:3000", "http://zimtec.ie"})`

Secondly, the mySQL database follows best practices in terms of **Database Seeding**, the initial Admin Privileges will need to be inserted manually. Specifically, the following column-values are needed to login through the frontend initially:

- **ID:** set to 1 and **auto-increment** will need to be set on the ID Column itself.
- **Email**: preferably a company domain email of designated Admin person.
- **Password**: for the Admin user.

Lastly, the **Frontend Environment Variables** can also be set for both development and production scenarios with typical APP URL's within Reacts .ENV files:

- Development: REACT_APP_API_URL=http://localhost:8080
- Production: REACT_APP_API_URL=https://your-domain.com

Beware the URL to the backend IP address or DNS will be exposed within Reacts JS bundles. This is standard practice in React Apps as URLs are not secrets. CORS is setup so only the allocated domain within the controller.java class will allow access to the API's.

4. Functionality & Use Cases

The intention of this section is to give further context to overall System Design, Implementation and code commentary. We will tie in Use cases and how they function within the frontend React components and backend Java, mySQL code bases. Firstly, let's divide those views into frontend, backend and give an overview of the structure, classes and components involved in making the web application work.

4.1 Frontend

At the root directory, index.tsx, Login.tsx and App.tsx are written in TypeScript-es6 as are all subsequent components and layout files. Each TypeScript file has a corresponding CSS file to allow for flexible and modularized styling.

The folder structure (See Fig. 9) separates the Components (Admin and Standard privileges) and Layout (See Fi. 9). The core Create, Read, Update, Delete (CRUD) operations were reused from Arjuncodes full-stack JavaScript application. For an explanation on the utils folder, please read the Use cases section, specifically, 4.3.4 Admin Tab -> SubMenu: Edit Welcome Note.

The main <APP /> is wrapped in a Router to facilitate the dynamic action area or white-space container, that is updated depending on the action being executed from the Admin Tab, the Sub Menu/Left Pane or an action within the container itself.

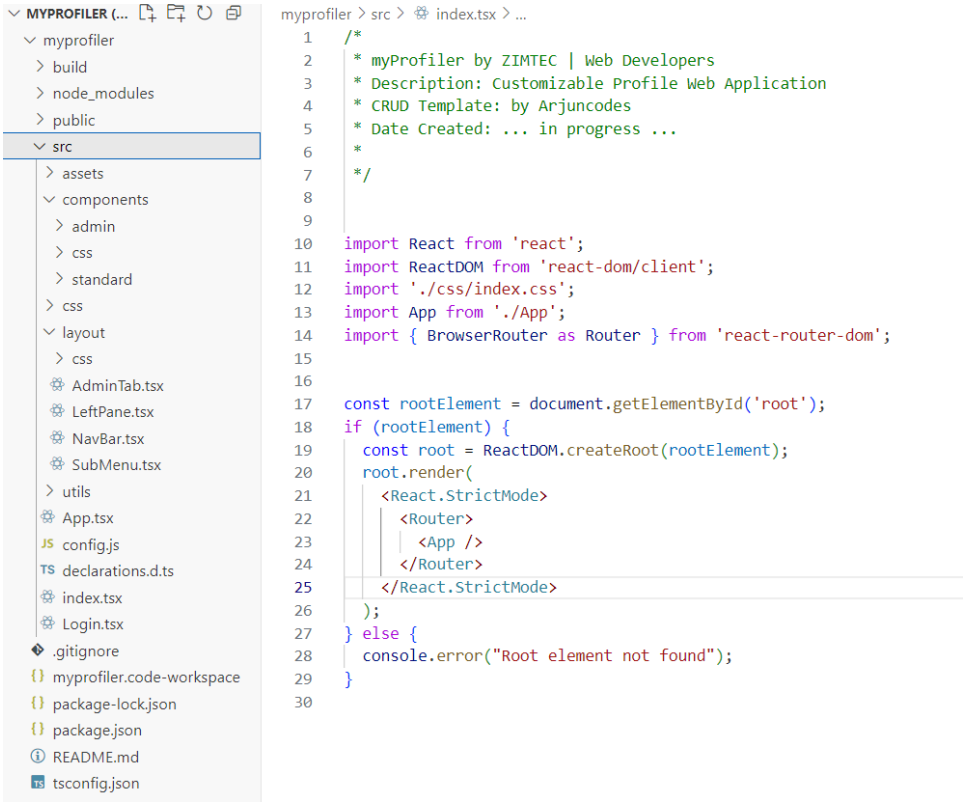


Fig. 9 – React Layout and Components

The config.js file, needs to be updated depending on whether the web application is being developed or built for deployment to production server. The URL for the production server would also need to be provided.

```
// config.js
const useRemoteURL = false; // Set to true for remote URL, false for localhost

const getBaseURL = () => {
  return useRemoteURL
    ? 'http://86.42.108.145:8080'  // Production URL
    : 'http://localhost:8080';   // Localhost URL
};

export default getBaseURL;
```

All subsequent TypeScript files can simply import the config.js function getBaseURL and call a Java web service. For example;

```
import getBaseURL from './config';

  const baseURL = getBaseURL();
  const handleLogin = async (email: string, password: string) => {
    try {
      const result = await axios.get<Profile[]>(`${baseURL}/profiles`);
      const profiles = result.data;
---
    }
  }
```

4.2 Backend

The backend is made up of a Java API connected to a mySQL database.

4.2.1 API - Java

The Java application was initialized with Spring Boot as a maven project using the following dependencies: Spring Web, Spring Data JPA and mySQL Driver. The initial packages, to date, are myProfiler which runs the application and its subsequent packages; Controller, Model, Exception and Repository (See Fig. 10).

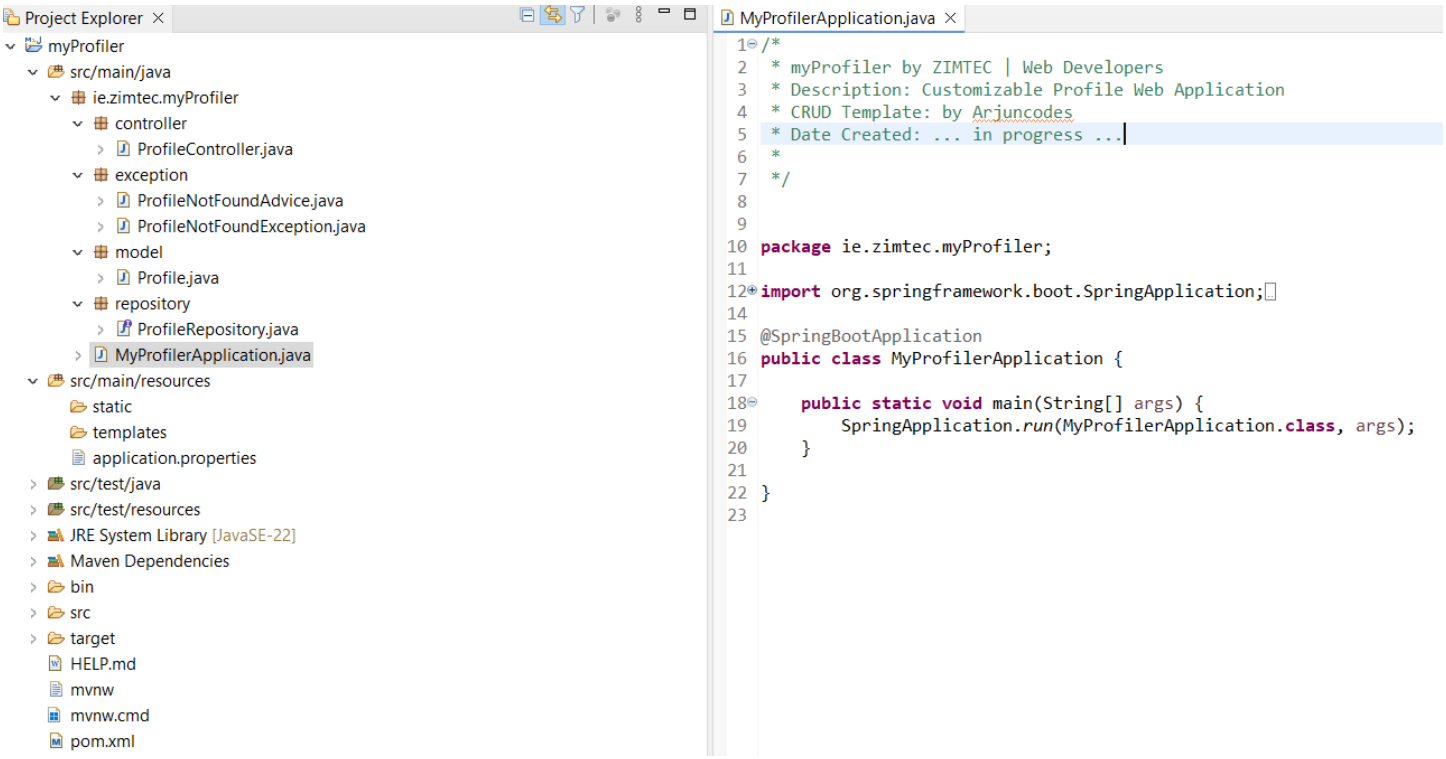


Fig. 10 – Backend Java Packages and Classes

The application is a RESTful API and is mapped from the getters and setters to any calls from the frontend. For example:

Backend API

```
@GetMapping("/profiles")
List<Profile> getAllProfiles() {
return profileRepository.findAll();
}
```

Frontend call

```
const loadProfiles = async () => {
    const result = await axios.get<Profile[]>(`${baseURL}/profiles`);
    setProfiles(result.data);
  };
```

4.2.2 Database - mySQL

The application.properties file from the Java project connects to the mySQL database and must be updated with the relevant access details for a new database.

Java **application.properties** details:

```
spring.application.name=myProfiler
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://zimtec.ie:3306/z599688_myprofiler
spring.datasource.username=z599688_admin
spring.datasource.password=*****
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
...
...
# timeout settings
```

Correspondingly, on the database administration side, an IP address from the Java project must be provided to allow access to the database. Hosting Ireland provided phpMyAdmin for the database interface and cPanel as the administrative tool for the Linux server and database setup.

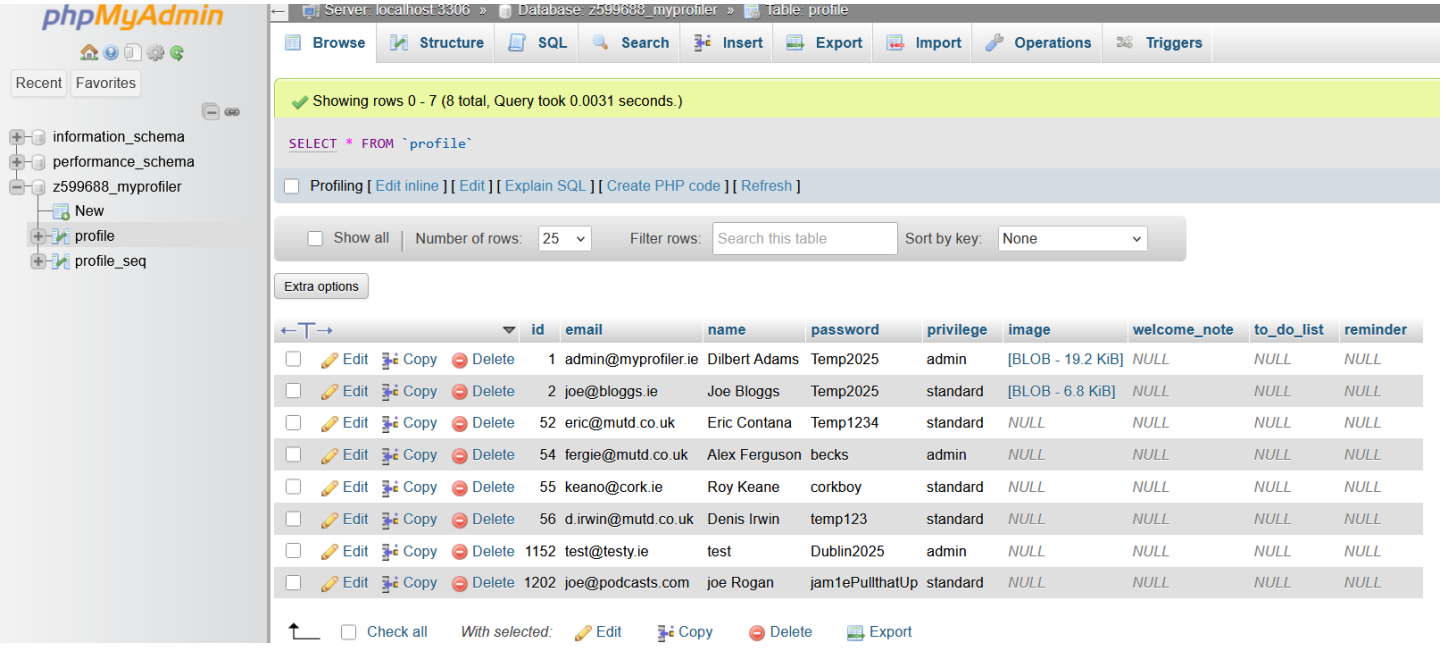


Fig. 11 – mySQL tables

The first time the Java application is run in a new environment, the tables will be created but will have empty rows. The first row must be inserted manually from the phpMyAdmin interface to allow a user to log in, with Admin privileges, to setup the rest of the myProfiler application. For example, see row 1 with ID 1 (See Fig. 11) in table 'profile'. The table 'profile_seq' assists the autoincrement and ensures each profile has a unique primary key.

4.3 Use Cases

4.3.1 Logging in

While logging in there are certain procedures and process that are executed according to the application login logic. These processes cause an overlap between the App.tsx and NavBar.tsx due to the dependency on privileges to load the NavBar with the Admin Tab button and Profile info. The Profile info it your typical top corner icon to clearly display who is logged and their respective profile name and image, together with a log out option.

To accomplish this necessary overlap, interface and props were used to pass settings between components. In general, the log-in logic can be defined as follows;

```
const handleLogin = async (email: string, password: string) => {
      ... // the email and password are correct
      ... // Close the Admin tab after successful login
      ... // Update isAdmin state based on privilege
      ... // sets State for profile logged-in, used for ToggleAdminTab
      ... // load the Profile image
      ... // Navigate to the profile details page of the logged-in user
      ... // update local storage with profile ID, used for deleteProfile in ViewProfile.tsx
}
```

4.3.2 loadProfile()

Profile details load the basic information for a profile; ID, name, email and password. These are all string data types on the frontend and ID is a long data type within the backend Profile Model.

Images are loaded within this function separately and are of data type, byte array. On the database backend its corresponding data type is a longBlob field, to facilitate small images only that are sent through the request header instead of larger images stored on a remote directory linked by a URL in the database.

Other considerations were using a decoder to convert to base64 strings which may facilitate larger images but when inserting into database it can lead to bloating and difficulty with request headers as the string may be too long. The profile image is only intended to be a small image.

Using a decoder can also cause latency issues which is not a relevant consideration when using small profile photos so the limit is set to less than 100kb within the frontend TypeScript inside the handleImageUpload function which is only available for an individual's profile, Edit Profile.tsx page.

We experienced the following problems on different platforms:

- **Edge**: crashed with any file over 100kb in the header: ERROR Maximum call stack size exceeded RangeError: Maximum call stack size exceeded
- **Firefox**: performed fine with files up to 500kb
- **Safari iOS:** crashed when caching an older image and was changed from a desktop browser within the same profile. Clearing the cache resolved this issue.

A person with Admin privileges does not have the ability to edit photos, it is down to the individual profile to upload images otherwise a defaultPic is uploaded until such a time as the profile user uploads their own image.

The image is mapped to the backend RESTful API which is constrained by CORS parameters thus communicating only with the specified development and production servers. The RestController and CrossOrigin are declared in the ProfileController.java class. This constraint is browser driven as the data is larger than simple text-based strings.

4.3.3 deleteProfile()

When an Admin user deletes a profile, the profile is removed from the database and the local storage in the browser for the profiles ID is removed. The Admin user is then navigated back to the Profile List from where they came.

In the case, when an Admin user deletes their own profile, the delete logic checks the local storage to see if the ID’s are matching, if they are, the profile is deleted from the database, the local storage is removed from the browser and the application is navigated to a specified URL, replacing browser history of the previous page to the prevent the logged-in user going back into the application.

4.3.4 Admin Tab -> Sub-Menu: Edit Welcome Note

The Sub-Menu or Left-Pane, will display the corresponding functions for any of the Admin Tab or Menu items that is selected by the user. In this Use Case we will explore the Home Tab, AdminHome.tsx, and

its corresponding Sub-Menu function ‘Edit Welcome Note’, explaining how the Admin Menu connects to the Sub-Menu.

All functions within the Sub-Menu are mapped to the Admin Tab by the FunctionRegistry.tsx within the utils folder (See Fig 9). This was the solution to maintaining clean code and avoiding deeply nested props and/or wrapping the main APP with each function. With the FunctionRegistry.tsx, Sub-Menu functions can be cleanly mapped from a single Left-Pane and update the corresponding whitespace depending on the function being invoked within a particular Admin Tabs’ sub-menu option.

There is no need for a parent component to declare a function then subsequently implement those functions in each child component, in this Use Case SubMenu and AdminHome. Both SubMenu.tsx and AdminHome.tsx import the FunctionRegistry, getFunction and registerFunction respectively. The function ‘Edit Welcome Note’ is called from SubMenu.tsx and implemented within AdminHome.tsx therefore updating the whitespace for Home tab option and its Welcome Note.

In its current implementation Edit Welcome Note, allows an Admin user to edit or reset to the default Welcome Note.

5. Future Considerations and Versions

This section contains most likely additions, improvements and alternative adjustments recognized during development of the current version 1.

5.1 Logging in

- Deeper security authorization
- Tokenization for user sessions: JSON tokens?
- User centric setup? Possibly allow users to setup but then becomes complicated with security within an organization. Maybe better to allow Admin person have the responsibility who is in and who is out within any company.

5.2 Profile Images

- Larger image uploads, scaling etc.

5.3 Mobile Version

Currently, myProfiler is only supporting desktop browsers and not supporting mobile use. You can expect these same standards with everyday tools and web applications like Microsoft Word for example.