Client Handover Guide

COMP30022 IT Project

By team CTRL ALT ELITE

UI Functionality Guide **3**
Browse 3
Search Bar 3
Advanced Search Modal 3
Profile Card 4
Profile (Edit Mode) 4
Profile Picture 4
About Text 4
Social Media Bar 4
Achievements 5
Skills 5
Card 5
Architecture **6**
Architectural Model 6
Architecture Description 7
Description of key functionalities **8**
Login 8
Sign Up (Bonus Feature) 8
Browse 8
Search/Advanced Search 8
Editing 9
Uploading 9
Settings 9
Layouts 9
Themes 9
Description of essential classes (code) **10**
Profile Classes 10
Programming Standards 10
Known Issues 10
MyProfile.js 10
Programming Standards 11
Known Issues 11
PlaceholderProfile.js 11
Programming Standards 11
Known Issues 11
Description of database structures and routes **12**
MongoDB 12

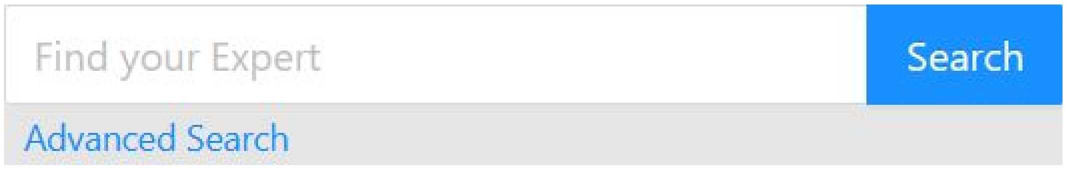
users 12
users_to_profiles 13
profiles 13
Contact 15
School 15
Work 16
Platform 16
Social 17
Reference 17
File 17
Routes 18
Known issues 18
MongoDB routes 19
My Profile routes 20
User routes 21
S3 routes 22
Deployment Guidelines 24
Current deployment 24
Fresh deploying 25
Local development 26
Product Changelog 26
License 29

UI Functionality Guide

This section covers how to use frontend components correctly, while showcasing some cool handy functionalities that one may miss at first glance.

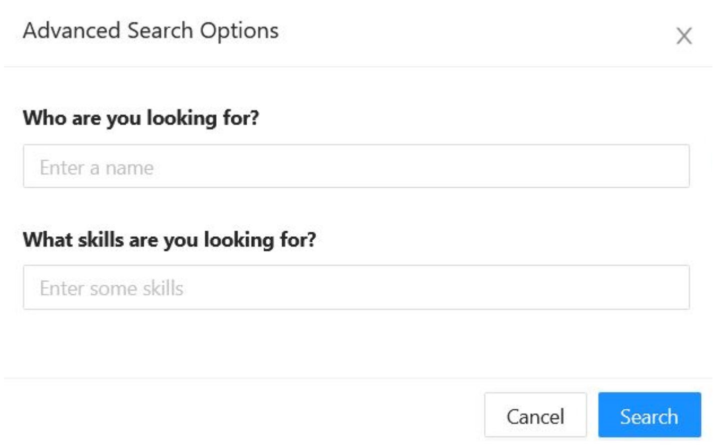
Browse

Search Bar



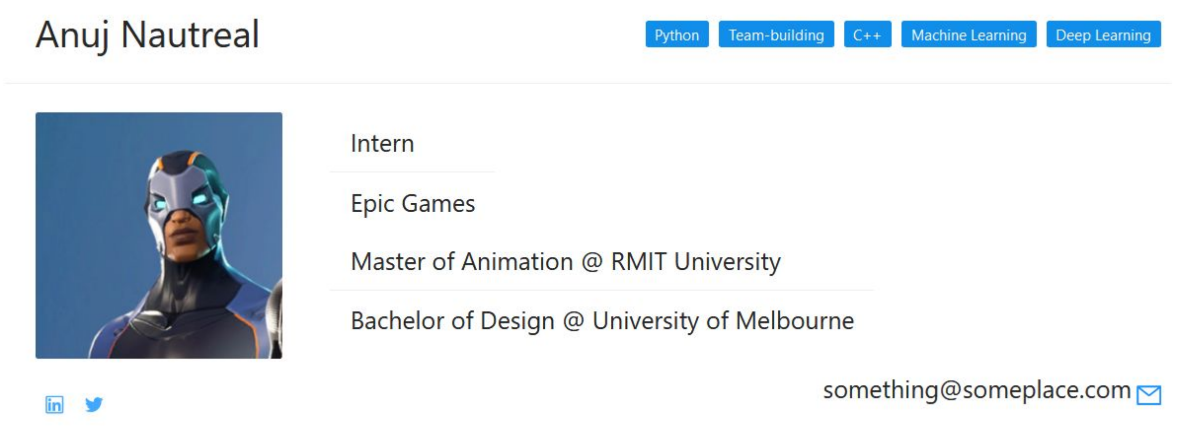
Search Bar -> Textbox -> OnClick -> Becomes focused and allows the user to type.
Search Bar -> Textbox -> OnBlur -> Becomes out of focus while retaining any text input.
Search Bar -> Search Button -> OnClick -> Executes the search query in text box.
Search Bar -> Advanced Search Button -> OnClick -> Pops open the advanced search modal.

Advanced Search Modal



Advanced Search Modal -> OnBlur -> Closes modal.
Advanced Search Modal -> X Button -> OnClick -> Closes modal.
Advanced Search Modal -> Cancel Button -> OnClick -> Closes modal.
Advanced Search Modal -> Textbox -> OnClick -> Becomes focused and allows the user to type.
Advanced Search Modal -> Textbox -> OnBlur -> Becomes out of focus while retaining any text input.
Advanced Search Modal -> Search Button -> OnClick -> Executes search query.

Profile Card


Anuj Nautreal

Intern

Epic Games

Master of Animation @ RMIT University

Bachelor of Design @ University of Melbourne

something@someplace.com

Profile Card -> OnClick -> Direct to profile.
Profile Card -> Email Icon -> OnClick -> Copies email to user's clipboard.
Profile Card -> Skill Tag -> OnClick -> Execute search query for clicked skill.
Profile Card -> Social Media Icon -> OnClick -> Directs to external social media.

Profile (Edit Mode)

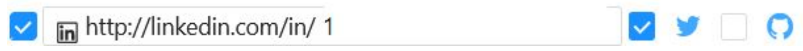
Profile Picture

Profile Picture -> OnHover -> Show change button.
Profile Picture -> Change Button -> Allow user to replace profile picture.

About Text

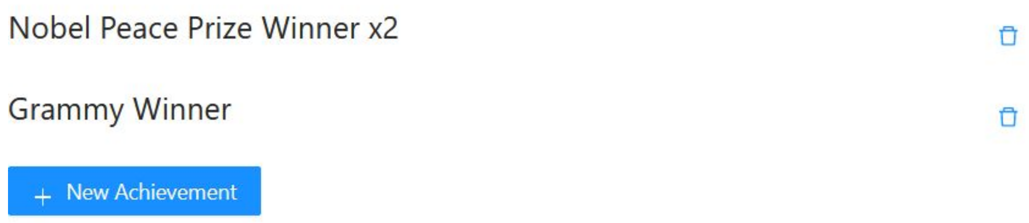
About Text -> Edit Icon -> onClick -> Convert text to textbox.
About Text -> Textbox -> onBlur -> Keep changes.
About Text -> Textbox -> onPressEnter -> Keep changes.

Social Media Bar



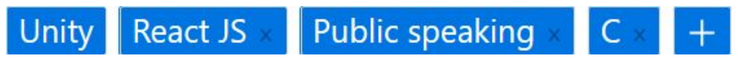
Social Media Checkbox -> Determines whether given social media is displayed on profile.
Social Media Icon -> onDoubleClick -> Open textbox to edit url of social media.
Textbox -> onPressEnter -> Keep changes.

Achievements



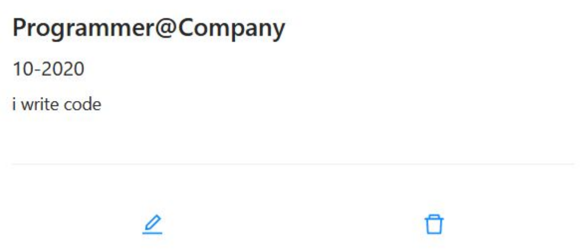
Achievement Item -> OnDoubleClick -> Convert text to textbox.
Achievement Item -> Textbox -> onBlur -> Keep changes.
Achievement Item -> Textbox -> onPressEnter -> Keep changes.
Achievement Item -> Delete button -> onClick -> Remove achievement.
Add Button -> Add a new textbox for the user to add a new achievement.

Skills



Skill Item -> OnDoubleClick -> Convert text to textbox.
Skill Item -> Textbox -> onBlur -> Keep changes.
Skill Item -> Textbox -> onPressEnter -> Keep changes.
Skill Item -> X button -> onClick -> Remove skill.
Add Button -> Add a new textbox for the user to add a new skill.

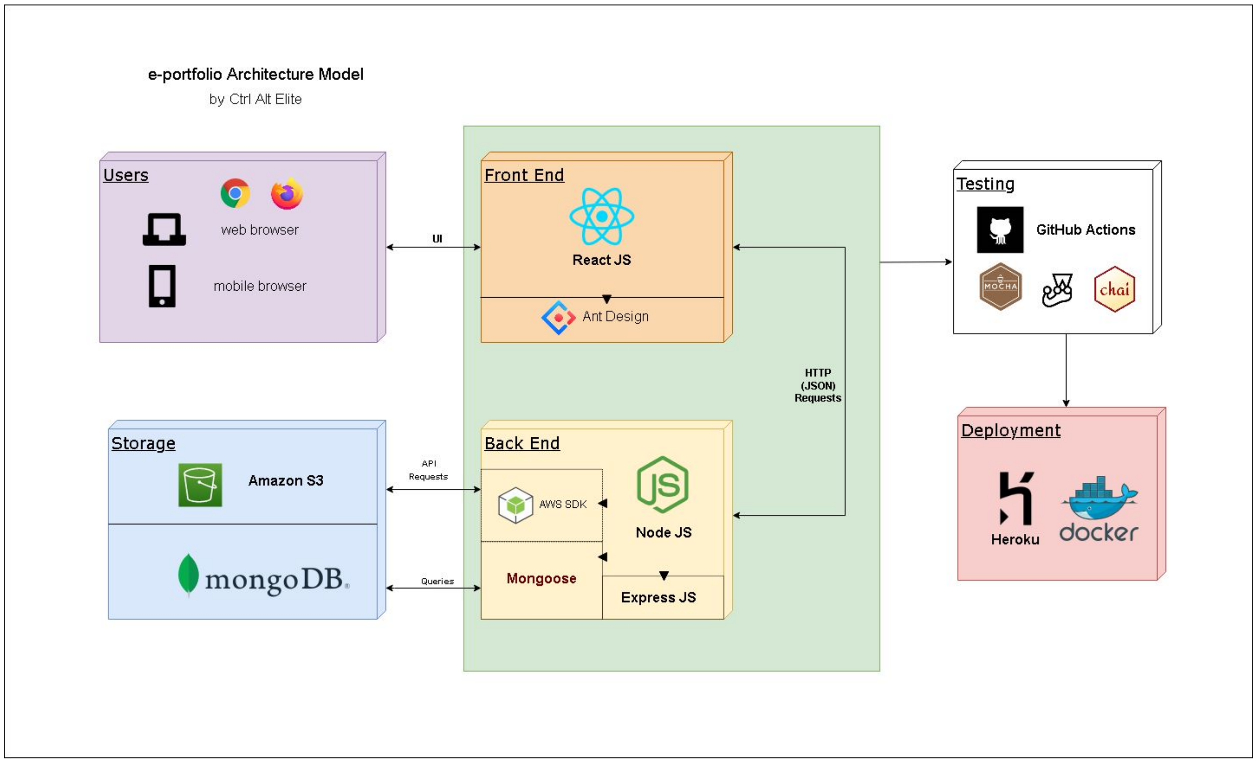
Card



Edit Icon -> Convert card to editable form.
Delete Icon -> Remove card.
Form -> Save Icon -> Keep changes.

Architecture

Architectural Model



Architecture Description

Before we began our first sprint, we agreed on a technology stack that consisted of React JS for front-end, Node JS for back-end, MongoDB for the database layer and Heroku for deployment. We chose React and Node as they both are highly scalable and widely supported. With Node JS, the Express framework is used for routing. MongoDB was chosen because of the heterogeneity of data going to be hosted on our e-portfolio service. It was also chosen because of its high availability and scalability. Heroku was chosen for deployment because it is easy to use and fully managed, allowing us to to focus on the development itself.

After discussing requirements with our clients, we decided to include an Amazon S3 bucket in our architecture to store files such as images and documents. This was again chosen because of its high scalability and availability.

This variation of the MERN stack was chosen because of the highly-documented and convenient APIs between each layer.

Towards the end of sprint 2, we began our automated testing using GitHub actions. Jest was chosen front-end testing because of its ability to run tests in parallel and only test the areas of our code changed. Mocha and Chai were chosen because they enable asynchronous testing for the back-end and have lots of examples available.

Description of key functionalities

Login

The login page presents the user a form in which they can input their login credentials. The form prompts the user if any fields have been left empty. Once the user clicks the login button the form is submitted and the form data is then gathered into an object which is compared against the database entries to verify whether the email and password are correct. If correct, the user is logged in and receives editing privileges for their profile. Otherwise, nothing happens.

Sign Up (Bonus Feature)

Sign up was not originally part of our agreed goals. It was brought up later in the semester by clients and so we decided to allow a few extra days to implement this feature. The sign up page presents the user a form in which they can input their basic details. Once the user clicks the sign up button the form is submitted and the form data is then gathered into an object to create a profile for the user. The user is then redirected to their new profile. For successful registration, it is important that the user enters their first and last name separated by space. If the email entered already exists, the user will be notified and the account will not be created.

Browse

The browse page maps through the profiles of all accounts that have registered and displays key details of their profile in a form of a card. On clicking on this card, Users will be directed to the selected profile page. As part of the key details that are displayed, their email (Click to copy!) and social links (Click to be redirected!) are shown as well as their key skills. These key skills are also used in the Advanced Search functionality- users are able to search for portfolios based on whether or not they have particular skills (Described in further details below).

Search/Advanced Search

The browse page has a feature to search through all the profiles on the website. The search bar on the browse page searches a user by name. There is also an Advanced Search feature that allows searching of persons who possess a certain skill (e.g., Python, Java, etc.). The Advanced Search modal can be accessed via the small text below the main search bar on the browse page. Any number of skills can be searched together to give narrower results. Searching via skills can also be combined with searching by name in the Advanced Search panel. Furthermore, specific skills can also be searched for on the browse page by clicking on them.

These search functionalities are enabled by MongoDB Atlas’ Search Indexes, which are based on the standard Lucene string analyser.

Editing

When a user is logged in to their accounts, on their profile page an edit button will become visible. On clicking the button, the profile enters an editable state, where beside each section of the editable data edit/delete buttons show up performing their respective tasks on clicking. When the profile enters edit mode, the previously labelled “Edit “ button is then labelled “Save”. When the user is satisfied with their new edited profile, clicking the button will save the changes made. Note: the user must click on save if they wish to keep the edited profile.

Uploading

Uploading of documents is done via a Dragger upload component from the external library we are using- Ant Design. When a user enters an editable state, they are given the opportunity to upload relevant documents pertaining to the projects the user has taken a part in or Certificates the user has earned. When attempting to upload a document, a modal pops up allowing the user to enter the Project/Certificate name and description along with a box where the user can drag and drop their document. These Projects/Certificates are displayed in the form of Cards, which are handled by the ProjectManager component. This component receives a Boolean value which indicates whether a card falls under the Projects category or Certificates. The actual upload is done via a post request.

The entire process is handled through Busboy pipelining, which takes in the file as a Blob, and posts it to the relevant route in the backend. The backend then uploads this to the S3 bucket using the AWS SDK and updates the user’s files on MongoDB.

Settings

When a user is logged in, their account name is shown in the navbar. This is a dropdown with 2 buttons- Logout and Settings. On selecting settings, the users are redirected to their profile page and a Settings modal pops up. Here the user can make changes relative to their profile and account. There are options to change layouts, themes, email, password, account name and URL.

Layouts

There are 4 different preset layouts that a user is able to choose from. The options are available in the Settings. Each account stores a value representing the layout that has been chosen for its profile. When a profile page is loaded it refers to this value and loads the corresponding layout component. Upon changing the layout and clicking “save”, This value will change and hence the corresponding layout will change as well.

Themes

There are 5 different preset colour themes available for the user to further customise their profiles. This option can also be found in Settings and works similar to the Layouts feature.

Each theme is mapped to a different combination of 2 colours- primary and secondary. Primary and secondary colours correspond to the colouring of different sections in the layout. While selecting themes, a small preview is displayed showing the 2 colours.

Description of essential classes (code)

Frontend classes are used via React.js. Generally the classes listed here are the most scalable and interchangeable. It is important to carefully look through all the classes listed below when creating a new profile layout template. The main classes that make up key aspects of the frontend system and user-based profile display are:

Profile Classes

The Profile classes (Profile.js, Profile2.js, Profile3.js , Profile5.js) are all profile layout components. For the design of these layouts, we have utilised various components from the Ant Design library to give each of the layouts a unique feel. They all share the same functionality and showcasing of data via the generalised components in the “profileDisplays” folder. They receive an account’s data as props as well as authentication information indicating whether or not the current user is the account owner.

Programming Standards

When creating new profile layout templates, ensure that:

- Render method is returning HTML containers instead of rendering the profile layout directly.
- Check for user authorisation (if the viewer the owner) in componentDidMount() and set the state appropriately.
- A mapStateToProps() method is declared and used to store Redux states into props to be used in the class.
- Correct props are passed into custom components/managers, if any.
- Correctly using the functions provided in ProfileData.js.
- Should have all components under UI Functionality Guide - Profile (Edit Mode), exclusions should be justified appropriately.

Known Issues

- Not correctly fetching Redux states and storing them would result in being unable to get profile data.

MyProfile.js

This class is responsible for taking the user to their own profile page. Here the user has the ability to customise and edit their portfolio. They are also able to change layouts, themes, and account details in the Settings modal present in the class. The class gets the data of the user's profile via an axios get-request. It then checks for whether the user is in fact the owner of the account, and redirects them accordingly to either their profile page or the login page(if the user is not authorized). The redirecting to their profile page is done via the PlaceholderProfile class which is described below.

Programming Standards

- render() method should have all/most HTML containers handled via placeholderProfile.js.
- On mount, the profile for a logged in user should be fetched correctly via axios. If the user is not logged in, should not render personal settings.
- Users should be redirected to the login page if they are not logged in/ authenticated.

Known Issues

- Not including a catch block when fetching user profiles by axios would result in undefined behaviour.
- Not correctly fetching Redux states and storing them would result in being unable to get profile data.

PlaceholderProfile.js

When a profile page is to be rendered, this class gets all the data from the profile via an axios get-request based on the url being visited. It then checks for which layout this profile uses and passes on the information to the corresponding Profile class. Furthermore, this class also handles the redirecting to "MyProfile" if the profile which is trying to be visited is self-profile.

Programming Standards

When adding new layout templates, ensure:

- New profile is imported and listed under *components.*
- Render method is correctly appended with the new, unique layout code, consistent with the declared string in components.

Known Issues

- Calling a state or prop object without checking for its existence explicitly (including its accessed data field) would result in undefined behaviour.
- Adding new profile templates/ changing profile template name, without updating *components* or changing arguments for React.createElement in render(), will result in the profile template not being displayed/undefined.

Description of database structures and routes

Full description of documentation can be found on
https://app.swaggerhub.com/apis/ctrl-alt-elite/E-Portfolio/1#/

MongoDB

The database used for storing user information is hosted on MongoDB for free. The main database is named *cae_users*. Mongoose is used to link backends to the database. In the database, four collections are used to store the information required by the system in their respective structures.

Note:

It is sometimes important to include a $exists keyword in the mongoose query to the database when querying an element that does not have the 'required' field set to true.

Ensure connection to MongoDB has been established before performing database operations. If connection failed, check database credentials, internet connection and IP address whitelist on the database itself.

These four collections are:

users

Stores a User model onto the database. A User model is used to store the login credentials and key user information. Structure consists of:

Element	Datatype	Description	Required?	Default (if any)
_id	ObjectID	Unique document ID	Yes	MongoDB auto
name	String	Name of account user	Yes	-
email	String	Email used for login	Yes	-
password	String	Salted hash of user's password	Yes	-
date	DateTime	Timestamp of account creation	No	Date.now()

users_to_profiles

Maps the ID of a user account to their respective profile ID, consistent with the _id in profiles. This collection is used to provide abstraction from storing login credentials with profile-related data. Structure consists of:

Element	Datatype	Description	Required?	Default (if any)
_id	ObjectID	Unique document ID	Yes	MongoDB auto
uid	String	ID in users	Yes	-
pid	String	ID in profiles	Yes	-

profiles

All profile-related information of each user is stored in this collection. Structure consists of:

Element	Datatype	Description	Required?	Default (if any)
_id	ObjectID	Unique document ID	Yes	MongoDB auto
firstName	String	First name of profile user	Yes	-
lastName	String	Last name of profile user	Yes	-
contact	Contact	Contact details of user	No	-
keySkills	Array of String	Key skills defined by users	No	-
education	Array of School	Education background of user	No	-
workHistory	Array of Work	Professional background of user	No	-
image	String	Path to profile picture stored on	No	“default.png”

		S3		
linkToProfile	String	User-defined path to profile’s web page	Yes	Profile ID
social	Social	Information of user’s 3rd party social media accounts	No	-
reference	Array of Reference	Referrals of user	No	-
about	String	Description given by user	No	“I am cool.”
primaryColour	String	Colour code of primary background chosen by user	No	“#ffffff”
secondaryColour	String	Colour code of secondary background chosen by user	No	“#e5e5e5”
filesAndDocs	Array of File	Uploaded user documents stored in S3	No	-
certificates	Array of File	Uploaded user certificates stored in S3	No	-
achievements	Array of String	User-given description of achievements	No	-
subtitle	String	User-defined short quote	No	“Flex it!”
specialty	String	User’s specialty displayed	No	“Software Engineering and Computer Systems”
timezone	String	Working timezone of user	No	“GMT +8”
layout	String	Layout design	No	“3”

		code chosen by user		
isNewUser	Boolean	Allows user to not display profile publicly	No	false

Additional models used:

Contact

Element	Datatype	Description	Required?	Default (if any)
_id	ObjectID	Unique document ID	Yes	MongoDB auto
email	String	Email address of user	Yes	-
phone	String	Phone number of user	Yes	-
address	String	Home address of user	Yes	-

School

Element	Datatype	Description	Required?	Default (if any)
_id	ObjectID	Unique document ID	Yes	MongoDB auto
institution	String	Place of education	Yes	University of Melbourne
name	String	Name of course undertaken	Yes	Bachelor of Science
level	String	Educational level of course	Yes	Degree
major	String	Major of course, if applicable	No	-
description	String	User description of education	No	-

from	DateTime	Timestamp of commencement	Yes	Date.now()
to	DateTime	Timestamp of completion	No	-

Work

Element	Datatype	Description	Required?	Default (if any)
_id	ObjectID	Unique document ID	Yes	MongoDB auto
role	String	Job title of user	Yes	-
workplace	String	Company name of employment	Yes	-
description	String	User description of employment	No	-
from	DateTime	Timestamp of commencement	Yes	Date.now()
to	DateTime	Timestamp of completion	No	-

Platform

Used in Social model. Structure consists of:

Element	Datatype	Description	Required?	Default (if any)
link	String	Hyperlink to social media	No	-
isEnabled	Boolean	True if user wants to display this information publicly on the profile	Yes	true

Social

Element	Datatype	Description	Required?	Default (if any)
github	Platform	GitHub link	No	-
facebook	Platform	Facebook link	No	-
linkedin	Platform	LinkedIn link	No	-
twitter	Platform	Twitter link	No	-
others	Array of Platform	Other social media links to display	No	-

Reference

Element	Datatype	Description	Required?	Default (if any)
name	String	Name of referral	Yes	-
role	String	Job title of referral	No	Referral
company	String	Workplace of referral	Yes	-
email	String	Contact email of referral	Yes	-
phone	String	Contact phone number of referral	No	-

File

Element	Datatype	Description	Required?	Default (if any)
_id	ObjectID	Unique document ID	Yes	MongoDB auto
name	String	User-defined filename to be displayed	Yes	-

filename	String	Filename in file data	Yes	-
description	String	User description of file	No	-
url	String	Path to stored file on S3	No	-
date	DateTime	Timestamp of upload	No	Date.now()

Routes

All backend routing is handled via Express and calls to backend routes must be made via axios. The following describes each action and the respective routes taken:

Known issues

Error in proxy routing should mean routing in server/app.js has not been correctly defined.

Normal success codes involve 30x which redirects, which is common and perfectly fine.

A status code of 5xx usually means MongoDB error. Check connection to database or any modified entry is correctly set up in MongoDB.

New functions/operations not performing as it should. Ensure core operations are synchronous or binded as a Promise.

Function fetchProfileByUID has to be used with care. The 2nd argument (isRaw) must be carefully considered as misuse of the function could corrupt the data entry on MongoDB.

Performing operations to a corrupted data entry in any collections will crash the system.

Ensure calls to S3 are synchronous to prevent delayed update of information.

Function calls to postUpload and postDelete in server/routes/api/mongo.js should be called following the corresponding upload and delete route for S3. Calling postDelete after failing to delete the actual file entry in S3 will create an unobtainable file on S3 instead. Calling postUpload without following a successful S3 upload route would have undefined behaviour.

MongoDB routes

Routes to profile or MongoDB are stored in server/routes/api/mongo.js.

Route name	/api/mongo/profiles
Operation type	GET
Description	Returns an array of all profiles that has isNewUser=false in the collection profiles.
Usage	Used to fetch all profile entries to be displayed on the Browse page.
Parameters	-
Success Code	200
Other Code	500

Route name	/api/mongo/p/{ID}
Operation type	GET
Description	Take profile ID or linkToProfile as {ID}. Returns the profile information corresponding to the profile ID in the path.
Usage	Used to fetch a profile based on the profile ID given.
Parameters	In path: ID - profile ID or unique link of profile to fetch
Success Code	200
Other Code	204 (No profile was found), 500

Route name	/api/mongo/search{query}
Operation type	GET
Description	Searches key skills of all profiles with or without filters. Returns an array of profiles that matches the query.
Usage	Search/Advanced search functionality on Browse page.
Parameters	In path: query - Encoded URI search query
Success Code	200

Other Code	500

Route name	/api/mongo/p-update/{ID}
Operation type	POST
Description	Used to save all profile changes made by users onto the database. Returns the updated profile entry in MongoDB profiles.
Usage	Takes a JSON object of updated date fields in the request body and saves it to the profile instance in MongoDB.
Note	Middlewares: - Users must be authenticated with their token provided. - If a change to linkToProfile is made, it will be validated by the checkLink function to ensure no duplicates of link.
Parameters	In path: ID - Profile ID of user only In header: x-auth-token - Valid token of user In body: Updated data fields for profiles in JSON format
Success Code	200
Other Code	500

My Profile routes

Generally used for operations for users' own profile.

Route name	/api/my-profile/
Operation type	GET
Description	Fetches the user's own profile entry post-login. Returns the user's profile entry in MongoDB profiles.
Usage	Fetches the profile linked to the User ID given in user.id by calling fetchProfileByUID.
Note	Middlewares: - Users must be authenticated with their token provided. Setting the 2nd argument of fetchProfileByUID to true in this instance could corrupt the profile entry if used haphazardly.
Parameters	In header: x-auth-token - Valid token of user
Success Code	200

Other Code	500

User routes

Used for performing account related operations and user credentials update. Found in server/routes/api/userAuth.js.

Route name	/api/auth/login
Operation type	POST
Description	Validates if a login credential given is valid then returns a signed token back to the user if valid.
Usage	Find the entry of matching email in users collection and then compare the given password with the hashed string. If validated, sign and return a valid token back to the user and redirect the user to his/her profile page.
Note	Token expires in 60 minutes after issue.
Parameters	In body: JSON object of email and password strings
Success Code	200
Other Code	400 (Wrong request body), 401 (Invalid credentials), 500

Route name	/api/auth/google-login
Operation type	POST
Description	Login via Google.
Usage	Login via Google's API. Finds the registered email in users with the verified email returned by Google. If validated, sign and return a valid token back to the user and redirect the user to his/her profile page.
Note	Ensure gmail used to login via Google is registered in users collection.
Parameters	-
Success Code	200
Other Code	401 (Email not verified/ Google error)

Route name	/api/auth/change-password
Operation type	POST
Description	Updates new password onto the users collection with the corresponding user email.
Usage	Hashes the new password with salt and stores the new hash onto the users collection.
Note	Ensure authenticated users (with valid tokens) perform the password change.
Parameters	In header: x-auth-token - Valid user token In body: password - New password string
Success Code	204
Other Code	500

Route name	/api/auth/user
Operation type	GET
Description	Returns a user's account information with profile ID appended.
Usage	Finds the user's entry in users collection by User ID. Fetches user's profile ID and appends it to user account information and returns it.
Note	Middleware: - Users must be validated with the tokens provided.
Parameters	In header: x-auth-token - valid user token
Success Code	200
Other Code	500

S3 routes

Routes to perform operations to AWS S3 bucket. Generally stored in server/routes/api/s3.js. Ensure bucket credentials are correctly provided and linked in GitHub Secrets.

Route name	/api/s3/upload
Operation type	POST
Description	Upload documents, certificates or profile pictures onto S3.
Usage	If “x-cert” is set, “x-pfp-upload” cannot be used, vice versa. Stores the upload entry on profiles via mongo.postUpload Profile picture: Finds existing profile picture path from profiles. Deletes current profile picture from S3 bucket if it is not the default profile picture and replaces the path. Certificate: Uploads file to S3. If successful, append the file entry to the users’ profile field certificates. Documents: Uploads file to S3. If successful, append the file entry to the users’ profile field filesAndDocs.
Note	Middleware: - Users must be validated with the tokens provided. Ensure the flag “x-cert” is set appropriately. “true” for certificate, “false” for documents. Ensure the flag “x-pfp-upload” is set appropriately. “true” for profile pictures. Only one header can be set in each call. Changes must be tested alongside the postUpload function in server/routes/api/mongo.js.
Parameters	In header: x-cert - Flag for signalling certificate or document upload x-pfp-upload - Flag for signalling profile picture upload and replacement x-auth-token - Valid user token In body: name - User-defined file name to be shown description - User-defined file description file - Metadata of file
Success Code	200
Other Code	500

Route name	/api/s3/remove/{file}
Operation type	POST

Description	Removes documents or certificates from S3.
Usage	"x-cert" in the header must be set to "true" if deleting a certificate. By default false and it will attempt to remove the file from filesAndDocs, returning a failed operation if not found. Validation of file owner must be performed prior to deleting, which prevents removing other user's documents without permission. After successful deletion of file data in AWS S3, removes the file/certificate entry from the user's profile instance.
Note	Middleware: - Users must be validated with the tokens provided. Ensure the flag "x-cert" is set appropriately. "true" for certificate, "false" for documents. File-to-be-deleted must be owned by the user requesting the operation via user token. Changes must be tested alongside the postDelete function in server/routes/api/mongo.js.
Parameters	In path: file - path to file stored on S3, usually fetched from a File model's url data field In header: x-cert - Flag for signalling certificate or document upload x-auth-token - Valid user token
Success Code	202
Other Code	403 (File not owned by user), 500

Deployment Guidelines

The deployment process for the product has been streamlined as much as possible to keep the development environment as comfortable as possible.

Current deployment

We deploy on Heroku, through a Docker container that is built within GitHub Actions. Any environment variables that need to be set are defined within the repository's GitHub Secrets, and in Heroku's Config Vars. However, if using/testing on a local environment, there needs to be a file **/server/config/development.json**, with the necessary keys named as mentioned below in the brackets. This file will also be provided with this Handover Document. This is imperative to make sure the product works without the need for environment variables. We have chosen not to upload this file as the repository as the repository has been made public, and it would not be ideal to have these keys openly stored. Ensure that the development.json file is not pushed to the repository at any time.

If a developer wants access to the current repository, then they would need to be manually added as a collaborator. The link to the GitHub repository is:
https://github.com/djatom17/e-portfolio

The Docker container is defined in **/Dockerfile**. This is a set of instructions that is used by Actions to create the container.

Extra code in the workflow ensures that a cache of each workflow is created on GitHub, and can be used later for faster deployment. The external tools used are:

- actions/cache@v2
- docker/setup-buildx-action@v1
- docker/login-action@v1
- docker/build-push-action@v2

The Actions workflow is run in two scenarios:

- When a pull request is made, it only tests the code.
- When a pull request is merged, or a commit is made directly to master, it tests again, and if successful, it deploys the code.

Fresh deploying

Steps to create a new instance/deployment of the product:

1. Create an application on Heroku, and a repository on GitHub with any settings.
2. (optional) Create two MongoDB Atlas databases, and obtain the access URIs - one is the production database and the other is the testing database. Make sure to set the access IP whitelist to 0.0.0.0 on both, as deployment on a free version of Heroku does not provide a static IP. This step is optional as the original database can be used, but is necessary if a fresh start is required.
3. Obtain an API key for the application on Heroku.
4. The following environment variables need to be set through Heroku's Config Vars, and GitHub's Secrets:
    - **JWT_SECRET** (jwtSecret)
    - **MONGODB_URI** (mongoURI)
    - **S3_BUCKET_NAME** (s3Bucket)
    - **S3_SECRET** (iamSecret)
    - **S3_USER** (iamUser)
5. In addition to the above, GitHub specifically needs some extra secrets:
    - **HEROKU_API_KEY**

- **HEROKU_APP_NAME**
- **HEROKU_EMAIL**
- **MONGODB_TEST_URI** (mongoTestURI)

Once this is setup, to create an initial deployment, the workflow provided in /.github/workflows has to be run within GitHub Actions. This will automatically test, build, and deploy the product. Then the normal procedure of deployment will follow.

Local development

There are multiple commands set up to facilitate localhost development. These are defined in **/package.json**.

- **npm run dev** - This should be used almost at all times during development. It creates a temporary build, and runs the server. These instances get updated immediately on saving a file in the source code - and changes are reflected immediately.
- **npm test** - This runs the tests for the frontend portion of the code. Use this to test against snapshots or to create new ones when required.

There are more commands located in **/server/**, to facilitate additional tweaking, defined in **/server/package.json**:

- **npm run localtest** - This runs the tests for the backend/server using Mocha. The tests are run on the testing database.
- **npm run localstart** - This should only be used if testing the server with no requirement for frontend contents, otherwise it is covered by **npm run dev** above.

The other commands that are present, are for deployment purposes only, and should not be changed unless necessary. These simply run the same commands but with different settings, making sure the application knows that it is on a production environment as opposed to a development or testing environment - thus correctly choosing whether to use the production or testing database.

Product Changelog

Note* The below table consists of only major releases/changes of the product and does not represent the entirety of the development process as followed by its definition.

Date	Type	Description	Branch
Aug 26, 2020	feature	Explorable page for viewing	joshDaniel/Frontend

Aug 31, 2020	feature	Server port for heroku	aman/dockerDeploy
Sep 5, 2020	bugfix	Incorrect url path connect fix	master
Sep 8, 2020	feature	Browse page	anjali-placeholder-pages
Sep 8, 2020	buildfix	Decrease delay on page switch	joshDaniel/Frontend
Sep 9, 2020	feature	MongoDB Connection, server deployment	amanNicholas/DBBackend
Sep 10, 2020	feature	Description box	Anjali-UI-skeleton
Sep 12, 2020	feature	Display image from server	amanNicholas/DBBackend
Sep 13, 2020	rework	Enhanced visual for profile page	danielJoshAnjali/profileTouchUp
Sep 17, 2020	feature	Axios for api calls and mongo api routes	master
Sep 21, 2020	feature	Default user profile layout	danielJoshAnjali/profileTouchUp
Sep 22, 2020	feature	Retain login credential	master
Sep 22, 2020	feature	Additional 2 unique profile layout for users to choose	joshAnjali/profile-presets

Sep 23, 2020	feature	Profile attribute Skills	Anjali/profile-skills
Sep 24, 2020	feature	Login, myprofile feature	amanNicholas/loginBackend
Sep 25, 2020	feature	File retrieval, img-upload	aman/githubActions
Sep 30, 2020	rework	New look for all four profile layout	joshDanielAnjali/UI-rework
Oct 3, 2020	feature	Edit mode for all profiles	joshDanielAnjali/UI-rework
Oct 5, 2020	feature	Heroku registry, docker layers for better server connection	aman/deployment
Oct 7, 2020	rework	Edit procedure	joshDanielAnjali/UI-rework
Oct 9, 2020	rework	Enhance profile layout and attribute showcase	joshDanielAnjali/UI-rework
Oct 13, 2020	feature	More user attribute	joshDanielAnjali/UI-rework
Oct 18, 2020	feature	Google login	aman/googleAuth
Oct 21, 2020	feature	File download from user profile	aman/uploadFrontend
Oct 23, 2020	feature	Colour customisation	joshDanielAnjali/UI-rework

Oct 27, 2020	rework	Enhance profile layouts	joshDanielAnjali/UI-rework
Oct 28, 2020	rework	Colour customisation	josh/ColourCustomisation
Oct 29, 2020	feature	Social Media links and icons	master
Oct 30, 2020	bugfix	Browse search issue	joshDanielAnjali/UI-rework
Nov 1, 2020	feature	Project, education folder	joshDanielAnjali/UI-rework
Nov 3, 2020	bugfix	Data retrieval error	joshDanielAnjali/UI-rework

License