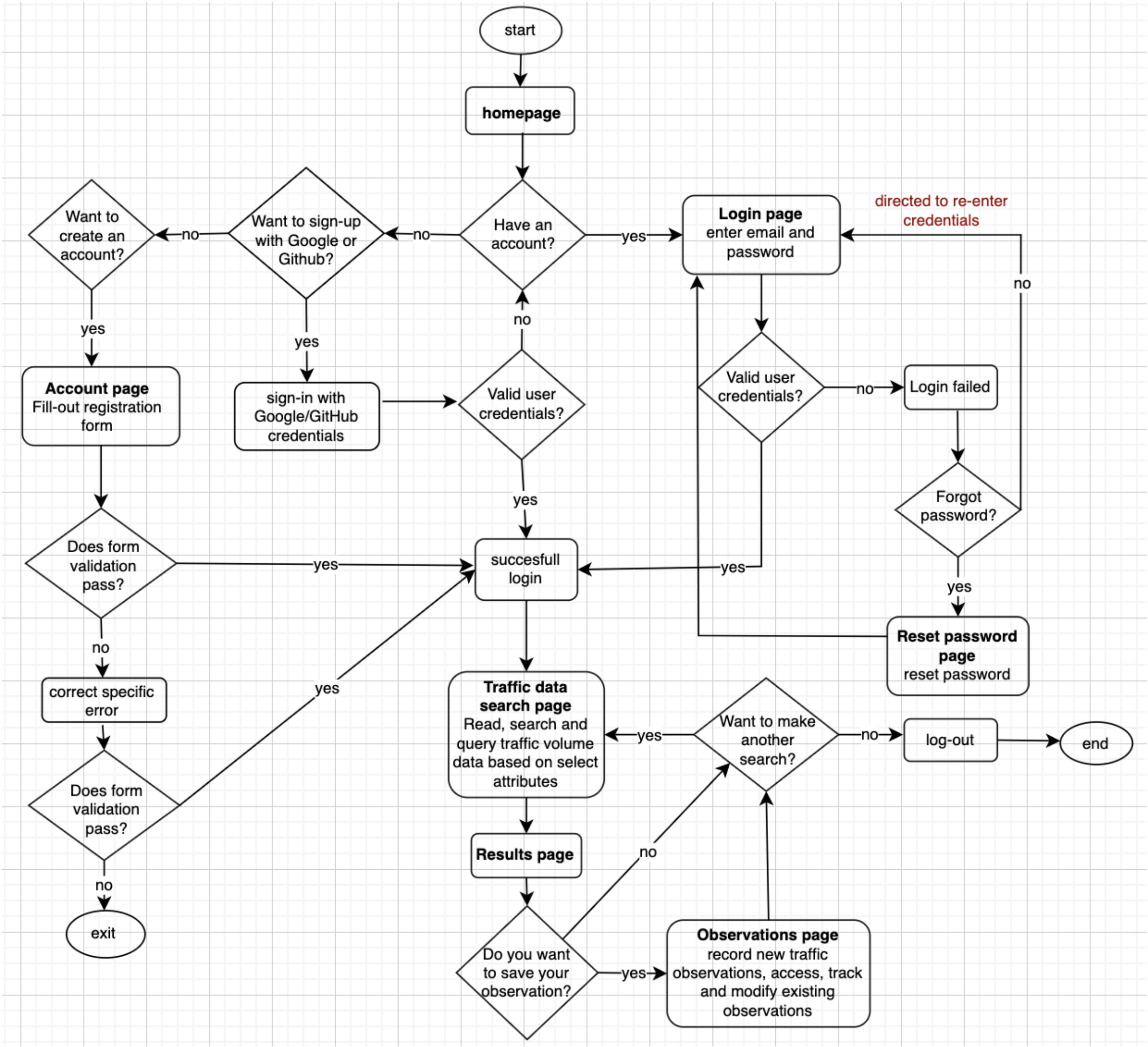
DESIGN

Flow Chart

Below flow chart is created to represent the journey of a user using the web app. Each page of the app is displayed along with steps that need to be taken in order to get to that page. Organizing the tasks in a sequential order, it depicts how the process of the system is performed from the beginning to the end. It also allows identifying tasks by type such as whether its a process, decision or data. By classifying tasks, value is communicated in clarity more efficiently

Flow chart is created to identify essential steps the user has to take within the process, along with alternative scenarios that could happen if the requirements of the steps are not fulfilled. While planning and designing the process, flow charts allowed capturing less obvious features of the process that can be refined to improve its efficiency such as unnecessary steps, flaws or bottlenecks. These features may be challenging to observe from an initial overview of the requirements but are pretty straightforward when the flow chart is created. Hence through its evolving nature, flowchart allowed me to capture the bigger picture of the system. (Rila, no date)

Flow Chart

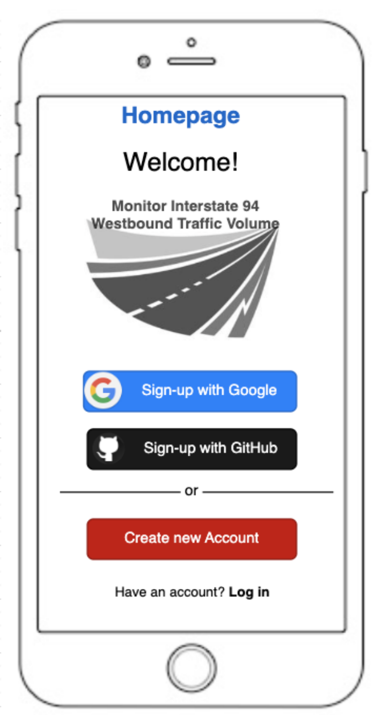

start
homepage
Have an account?
yes
no
Want to sign-up with Google or Github?
Want to create an account?
Login page
enter email and password
directed to re-enter credentials
no
yes
yes
Account page
Fill-out registration form
sign-in with Google/GitHub credentials
Valid user credentials?
Valid user credentials?
Login failed
Forgot password?
yes
Does form validation pass?
yes
succesfull login
yes
Reset password page
reset password
no
correct specific error
yes
Traffic data search page
Read, search and query traffic volume data based on select attributes
Want to make another search?
log-out
end
Does form validation pass?
no
exit
Results page
Do you want to save your observation?
Observations page
record new traffic observations, access, track and modify existing observations

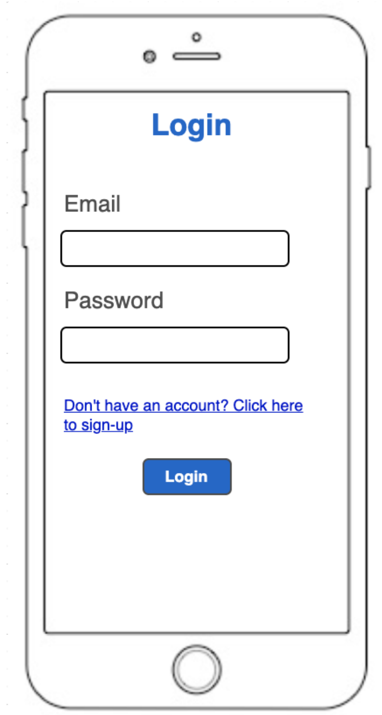
Interface Design

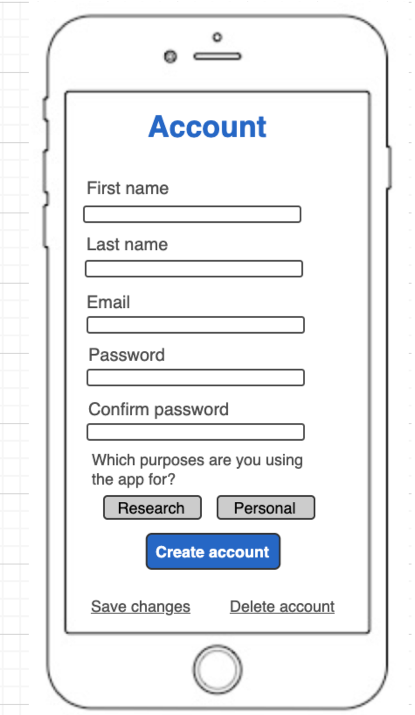
Wireframes

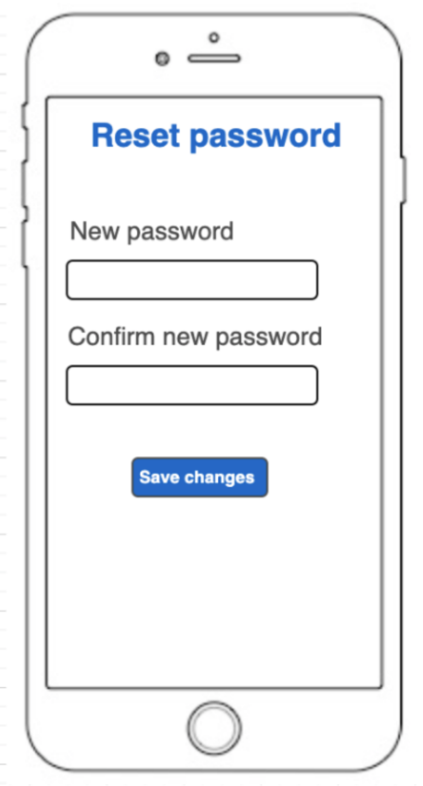
Below wireframes represent the view and design of the user interface that the users will use to interact with the web app. The wireframes are implemented as they design the blueprints for the application interface and provide a reference point for the functionalities that will need to be built.

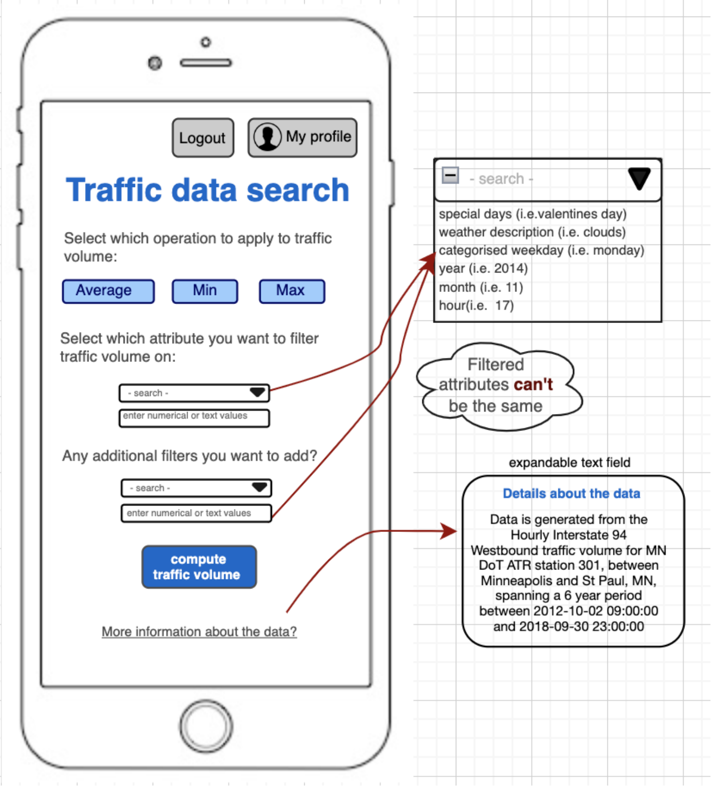
Wireframes of the web pages



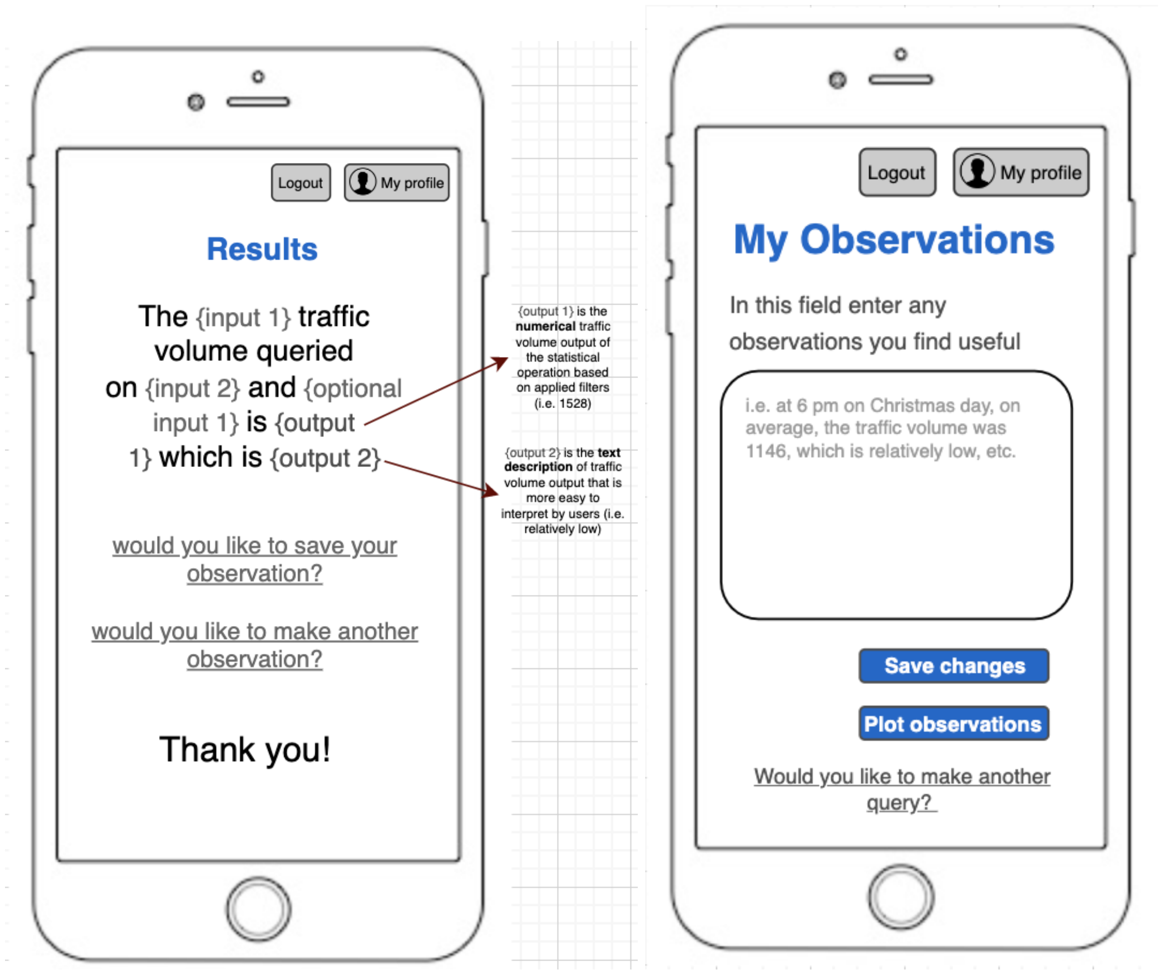







Logout
My profile
Traffic data search
Select which operation to apply to traffic volume:
Average
Min
Max
Select which attribute you want to filter traffic volume on:
- search -
enter numerical or text values
Any additional filters you want to add?
- search -
enter numerical or text values
compute traffic volume
More information about the data?
- search -
special days (i.e.valentines day)
weather description (i.e. clouds)
categorised weekday (i.e. monday)
year (i.e. 2014)
month (i.e. 11)
hour(i.e. 17)
Filtered attributes can't be the same
expandable text field
Details about the data

Data is generated from the Hourly Interstate 94 Westbound traffic volume for MN DoT ATR station 301, between Minneapolis and St Paul, MN, spanning a 6 year period between 2012-10-02 09:00:00 and 2018-09-30 23:00:00



The {input 1} traffic volume queried on {input 2} and {optional input 1} is {output 1} which is {output 2}

would you like to save your observation?

would you like to make another observation?

Thank you!

{output 1} is the **numerical** traffic volume output of the statistical operation based on applied filters (i.e. 1528)

{output 2} is the **text description** of traffic volume output that is more easy to interpret by users (i.e. relatively low)

In this field enter any observations you find useful

i.e. at 6 pm on Christmas day, on average, the traffic volume was 1146, which is relatively low, etc.

Would you like to make another query?

Note: Although not presented in above wireframe views, ‘My profile’ includes a dropdown menu consisting of below features:

- **my account** - where users can access and modify account details
- **my observations** - where users can access and modify recorded observations
- **my plots** - where users can plot their observations and later view and modify their saved plots

Each page of the web app has its own wireframe depicted above. These wireframes show the user interface for the web app for each page and give the initial view of how the app should function. The web pages of the application are explained in detail below:

1. **Homepage** - *Initial screen seen by users prior to login or signup*
   ‘/’ route accepts a GET method, controller function home() which renders the view of the homepage. Users are directed to specify their access type to the system; they have 2 options to sign up. They are opted to choose whether they wish to sign up with an integrated social platform - Google or Github- or create an account. If they already have an account they are directed to the login page to login to the system

   If they choose to sign-up via 3rd party social platforms, they are directed to the ’/account/3rd_party_login’ route which accepts a POST method. Users input their 3rd party credentials, controller function 3rd_party_login() checks the entered account details against the account details retrieved from the 3rd party service, returns error if

details don’t match, if incorrect details are submitted again users are directed to create an account, if details match, they are directed to traffic search page.
Note: For 3rd party login, the traffic application has no authority to validate user credentials. The process of validation and access through 3rd party login is the responsibility of the specific social platform chosen as credentials are saved within the 3rd party platform database and isnt accessible by this application

2. **Login page** - *Presents the view that allows users enter login credentials and log in to the system if credentials match*
   ‘/account/login’(route) accepts POST and GET method, users input login information, controller function login() checks whether the entered credentials match those registered in the database(retrieves credentials). If it's a match, user logs in successfully, if only partial details match, they are directed to the reset password page, if no match is found they are directed to the account page where they can create a new account

3. **Reset password page** - *Presents the view that allows users to reset their password*
   /account/reset_password’ route accepts a POST method, controller function reset_password() renders the view that allows users to reset their password. Users are directed to this page only if they have forgotten or are unable to use their registered password. They create and confirm a new password using their registered email address, if new password is validated it is saved to the database.

4. **Account page** - *Presents the view that allows users initially create an account filling the registration form and can later edit, update or delete account*
   ‘/account/signup’ route accepts a POST method and the controller function account() allows users to create a new account filling up the registration form. If form validation passes users are directed to the login page, otherwise they are prompted to correct specific errors. If error reoccurs they are presented the option to sign-up via 3rd party platforms

   ‘/account/modify’ accepts PUT method, modify_account() - renders the view that allows users to update or modify their account details. As all fields are mandatory they are not able to delete any information, they can only update and change them.

5. **Traffic search page** and **Results page -** *these 2 web-pages are rendered in a single view that initially displays the query form and later the results of the queried calculation*
   Traffic search page renders the view that directs users to fill in the query form where they query traffic volume data with certain specifications. When the form is submitted, if the form fields are valid, parameters are saved to the database and users are redirected to the results page where they can view the results of the requested calculation, otherwise they are prompted to correct specific errors.

   ’/query/view’ route accepts a POST method, controller function query_view(), retrieves the values/parameters from the form, makes a query using the model based on the chosen summary statistic and descriptive parameters and returns the result of the statistical operation based on the queries which is rendered in an updated view - that is the results page. When a query is selected, all records of the selected field are retrieved and the chosen summary statistic is performed on the target attribute (traffic volume).

’/query/results’ route accepts GET method, controller function query_results() renders the view of the results page with the results of the query form. It returns a duple consisting of the integer traffic volume estimate and a relative written description of the interpretation of the traffic volume estimate. By clicking on the relevant link, users are directed to either observations page to save their results or to query page to make another query

6. **Observations page** - *Presents the view that allows users to save, track and later modify their observations from the results page*

   ‘/observations/create’ route accepts GET and POST methods, controller function create_observation() - renders the view of the observations page and allows users to retrieve and save their results of traffic queries with selected parameters. Users can track their observations by retrieving traffic volumes estimates in a text field, and can create a list of observations by adding and saving their results.

   ‘/observations/modify’ route accepts GET, POST, PUT and DELETE methods, controller function modify_observation() renders the view that allows users to retrieve, add, update, delete and save observations. Additionally for practicality reasons users can plot their observations to visualize general tendencies and possibly patterns of certain fluctuations in traffic volumes. They can make further queries by clicking on the relevant link which redirects them to the traffic search page.

Application Design

MVC Design Pattern

MVC design pattern separates an application into three main components: the model, the view, and the controller. As each component is responsible for a specific aspect of the application and each has specific roles and responsibilities, changes to one component does not affect other components keeping the code more modular and easier to maintain as code duplication is limited. This also enforces reusability of components in different parts of the application as MVC separates the business logic of the application from the user interface. As components can be updated and modified independently, MVC pattern also allows for flexibility and scalability in the design and development of an application. This makes it simpler to make changes to the application without having an impact on the entire architecture. This also allows testing of individual components (unit tests) easier as they are isolated from the rest of the application. Hence the process of identifying and fixing bugs is accelerated ensuring that the application is working as intended.

Overall, by improving the structure, organization and maintenance of software applications, MVC pattern contributes to creating software applications easier and faster to develop while maintaining high-quality code. (Anushka, 2021)

While the MVC pattern has many benefits, there are also potential drawbacks that may create complexities within the design of the application. For instance, although in theory, changes to one component does not affect others, in practice, the components in an MVC application are tightly coupled or closely connected. This suggests that if components are not properly separated or isolated, changes to one component can have unintended consequences on the others. Another disadvantage is that there is very limited scope for customization as the MVC pattern dictates how the components are structured and how they should interact. This restricts the ability to customize the architecture to meet the specific needs of the application.

Model View Controller (MVC) Design Pattern

Class(model)	Attributes(view)	Methods(controller functions)

account	first_name:str
	last_name:str
	email_address:str
	password:str
	usage_intention: bool
		verify_password(password:str):bool
		reset_password(password:str):str
		create_account()
		update_account()
		delete_account()

query	special_days: str weather_desc: str categorized_weekday: str year: int month: int hour: int	get_avg(query_1, optional_query_2):int get_min(query_1, optional_query_2):int get_max(query_1, optional_query_2):int (these controller functions take a variable number of arguments chosen from query attributes)

observations	my_observations: str	retrieve_obs()
		add_obs()
		update_obs()
		delete_obs()
		save_obs()
		plot_obs()

URIs

Application Programming Interface’s (APIs) provide a way for developers to access and use an application's data or functionality. Each component is accessed via an API. REST is an architectural style for creating APIs using HTTP endpoints and a RESTful API is one 'pattern' for implementing a microservice. REST uses HTTP endpoints for handling the resources and provides a programmer access to the data. For the scope of this project, resources are records within the interstate traffic data, and they are defined by HTTP verbs; GET, POST, PUT and DELETE. The HTTP verbs are defined in the description of the web pages presented in the wireframes.

RESTful web applications usually define a Uniform Resource Identifier (URI) to locate and identify where a resource(s) can be found on the server hosting the web service. URIs are used in REST APIs to address resources to developers using an API. URIs defined in RESTful web services provide resource representation such as JSON and a set of HTTP Methods. (What are the benefits of routers when the URI can be parsed dynamically?, 2013)

As the primary function of a URI is to identify a resource on the Internet, it allows end-users to access and interact with the resource. URIs can be accessed from any device, allowing easy access to resources regardless of location or device. URIs can also be used to link to other resources, its dynamic parsing functionality allows users to easily navigate between different resources. URIs remain valid and accessible over time (persistent), this suggests that users can access resources even if the location of the resource changes or the resource is moved to a different server.
Hence URIs play a key role for users to access and interact with resources, and through their universal accessibility and persistence they are crucial for navigating the digital world URIs should always make sense and adequately describe the resource. To enhance understandability and usability, URIs should follow a predictable (consistent) and hierarchical structure (data should have a structure and include relationships). (Levin, 2017)

Below diagram represents the formation process of URIs, Route's are the paths that a user can take within an application. It specifies the URL that a user can use to access a particular page or resource within the application. HTTP methods used in each route/URI are provided. Wireframes are visual representations of the layout and structure of the webpages of the application, it is used to plan and design the user interface and it ensures that all necessary elements are included in the final product. To avoid complications the wireframe views of the associated routes are provided. Controller functions handle user input and manipulate the model data to update the view. They help separate the logic of the application from the user interface, making it easier to maintain and update the application. These 3 components work together to create a more efficient, organized and user-friendly experience for users using the software application

Format of Responses - JSON

Response format captures how answers will be collected from the recipients or users. For this application when a REST request is sent, the API users or clients will expect to receive data in JSON (JavaScript Object Notation) format. JSON is the leading format/method used for transmitting structured data in web applications as it is an open standard, text-based format that is easily readable both by humans and machines. Although JSON represents data in JavaScript object syntax, it is not dependent on any language and can be used in several programming languages including Java, Python, C, C++ etc. The JSON response allows APIs to represent structured data in an easily understood and parsed (to allow access to the data within it) format and can be used for all of the available resources within the API.

Its data interchange format allows storing and transmitting objects composed of attribute(key)–value pairs and arrays. A JSON object contains data in the form of a key - value pair where the keys are the strings and the values are the JSON types. While the keys and values are separated by a colon, each key/value pair is separated by a comma. These key/value pairs are created by an object and an array. Each object can have different types of data including string, numerical, array or boolean etc. allowing to the construction of a data hierarchy (Working with JSON, 2022)

URIs

Route	Wireframe	Controller function	HTTP Method
'/'	homepage	home() - renders the view of the home page. Presents users with the options to create an account, signup via 3rd party services, or login to their registered account	GET
'/account/signup'	create account page	account() - renders the view that enables users to create a new account by filling up signup form - if the form fields are valid then credentials are saved to the database and users are redirected to login page, otherwise an error message is displayed prompting users to correct specific error	POST
'/account/login'	login page	login() - renders the view of the login page. Takes the entered user credentials, checks against	GET, POST

		the details in the database, returns error if details incorrect otherwise redirects users to query page	
’/3rd-party-sign-in’	3rd party login page (not a page in the web app, users are directed to the login page of selected social platform)	3rd_party_login() - directs users to the login page of selected social platforms. 3rd party service checks credentials against its database. Returns error if incorrect details are entered, If details match, login is verified by the 3rd party service and user is redirected to traffic search page	POST
‘/account/logout’	logout - in navigation bar	logout() - logs out current user and redirects them to home page	GET
‘/account/<username>’	account page	modify_account(<username>) - renders the view that allows users to view and modify their account details. Takes the current user details, presents account form for user to edit and submit, validates the form, if validation is successful, changes are saved to the database	PUT
‘/account/reset_password’	reset password page	reset_password() - renders the view that allows the user to reset their password if they have forgotten it. If new password is validated and verified, updates and saves the new password to the database	POST
’/query/view’	query page	query_view() - renders the view of the query page with the query form. If the form fields are valid, when the user submits, parameters are saved to database and the page is turned into results page	POST
’/query/results’	results page	query_results() - renders the view of the results page where	GET

		the results are retrieved from the query form. Redirects users to either to observations page to save their results or to query page to make another query	
‘/observations/create’	observations page	create_observation() - renders the view of the observations page and within a text field allows users to retrieve and save observations with selected parameters	GET, POST
‘/observations/modify’	observations page	modify_observation() - renders the view that allows users to access, change, update, delete and plot observations.	GET, POST, PUT, DELETE

References

Anushka, V. (2021) *Benefit of using MVC*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/benefit-of-using-mvc/ (Accessed: December 26, 2022).

Levin, G. (2017) *7 rules for REST API uri design*, *REST API and Beyond*. REST API and Beyond. Available at: https://blog.restcase.com/7-rules-for-rest-api-uri-design/ (Accessed: December 27, 2022).

Rila, L. (no date) *What is the significance of a flowchart? [updated]*, *Culture Online*. Available at: https://www.ucl.ac.uk/culture-online/ask-expert/your-questions-answered/what-significance-flowchart-updated (Accessed: December 25, 2022).

What are the benefits of routers when the URI can be parsed dynamically? (2013) *Stack Overflow*. Available at: https://stackoverflow.com/questions/19230771/what-are-the-benefits-of-routers-when-the-uri-can-be-parsed-dynamically (Accessed: December 27, 2022).

Working with JSON (2022) *Learn web development | MDN*. Available at: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON (Accessed: December 29, 2022).