



DALHOUSIE UNIVERSITY

Subject: CSCI 5308 Advanced Software Development Concepts

Professor: Dr. Tushar Sharma
Final Project Documentation

Prepared By: Group 21 Wanderer

Team Member	Name	Banner Id
1.	Adil Otha	B00900955
2.	Indu Munagapati	B00903180
3.	Janvi Nayanbhai Patel	B00893747
4.	Janvi Patel	B00896196
5.	Meghdoot Ojha	B00854209

Table1: Divisions & Sections

Broad Divisions	Individual Sections
1. Preliminary Material	Title Overview
2. Body	Features of Application Usage Scenario Deployment Working Cycle Backend Dependencies Frontend Dependencies Build and Compile Instructions Best Practices Additional Functionalities Application PROD Access Link Test Coverage of Application
3. Supplementary Material	References

The Wanderer - *Let's travel with us!!*

Overview

Wanderer Application is the travel companion platform where all enthusiastic travellers can find attractive and well-known places of attraction in their nearby locality with real-time map location. This application enriches the travel companions with more detailed information about the location through the reviews given by peer travellers who have previously visited that place. The core idea of this application is to multiply the travellers community where new travellers can get a fresh perspective of places through the memories shared by the other zealous travellers.

Features of our Application

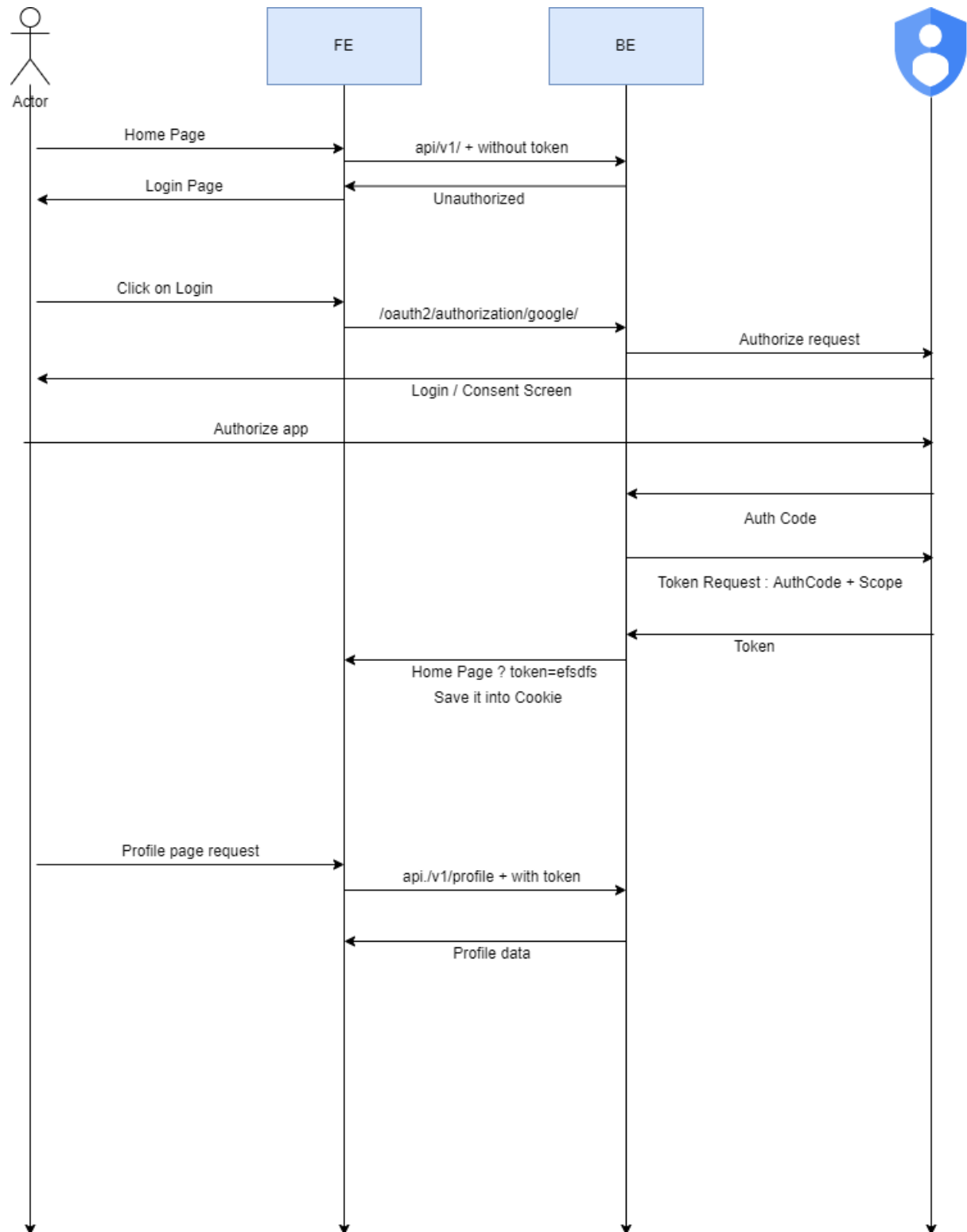
We have supported the following features in our application considering the key point in our mind to provide a seamless user experience.

1. OAuth 2.0 Login Single SignOn Feature

We have supported the social sign-on OAuth login feature through Google where users don't need to share their credentials with us, instead it will be maintained by the third-party Google itself. This feature won't only provide more security to the user but also make their login process into this application more seamless because it is single sign-on and so they don't need to register with our application and the basic requirement of allowing users to access the application is still achieved. In order to make a robust authentication layer, we have implemented token-based authentication, which passes the token in the request's Authorization Header that is stored in the cookies on the frontend side and validates the user on the basis of that token. The extra security layer we have introduced in our application by implementing **Auth Guards** in the frontend where users won't be able to access any other pages of the application if they are not authenticated.

OAuth Flow Diagram

The below diagram depicts the login feature implemented along with the token filtration concept



Step-1: When the user sends the request to the frontend to access the application then the frontend will send the same request to the backend without a token.

Step-2: As the user is unauthorized the backend will invoke the frontend with an unauthorized call and then the frontend will navigate the user to the login page.

Step-3: The user will click on the login button and then the frontend will accept its request and will then send an oauth2 google authorization request to the backend.

Step-4: The backend will send the authorization request to Google and then Google will navigate the user to the consent login screen of Google. Google will handle the responsibility of authorizing the user and will send token and auth code to the backend application.

Step-5: The backend will send the received token to the frontend in the request authorization header and the frontend will then save it to the cookie for further user verification.

Step-6: When the user will request to access a specific page of the application then the token will be sent in the request header and the token authentication filter process will be performed on the backend where it will validate the token first whether it is expired or not and after then it will validate the user.

2. Map Integration Feature

We have used the Google Maps API (AGM Core Library) to support the map integration feature with our application. Using this feature users can navigate through specific locations on the map through mouse and keyboard controls. The locations will be fetched based on the user current location and set radius.

Pin Creation

- We have implemented the real-time updates feature of the pin where users can create the pin on a specific location and that created pin can be visible to the others. We have achieved the real-time update feature requirement by using the Firestore which gives the real-time updates.
- Moreover, users can add/edit the pin details while pin creation by enabling users to add location name, description details of the place, and can insert multiple images, to the specific location that can help other users to get insights into that place.
- Only the Pin owned by a User can be updated with new data such as location name, description, coordinates, and images. The application will not allow the modification of Pins created by other Users.
- Since there can be innumerable Pins created on the Map by several registered Users, we have categorized the Pins into different Colors for a better User Experience. The Pins are categorized as follows:
 - Green Pins: The logged-in User-created Pins will be displayed using the Green Colour.
 - Red Pins: When a user logs in, the Pins created by the other Users will be displayed using the Red Colour.
 - Yellow Pin: Denotes a temporary Pin. On clicking on the Map, a Yellow colored Pin will be created. This Pin will be temporary as the User will select a precise location by dragging the Pin around the Map before deciding on the

location. On saving the Pin, this Pin will turn into a Green Pin which denotes that it is saved in the database and its location is fixed.

- Blue Pin: Denotes that the Pin is in “Edit Mode”. On opening the Pin Modal and selecting the “Edit Coordinates” button, the selected Pin will go into Edit Mode. This pin will turn blue to denote that this Pin can be dragged around and then saved again to update its coordinates.

Pin Images

- When a Pin is created, the user is prompted to upload images of the newly created pin. Images can be accepted in any of the two formats, png or jpeg. Several validations are applied to avoid duplicity and load on the application. A maximum of 5 images can be added to your location Pin. If a pin already exists, already available images will be replaced with new uploads or will upload fresh images if no pre-images are located.

Pin Description

- Users can add a description of pins. The maximum allowed number of characters is 1000.

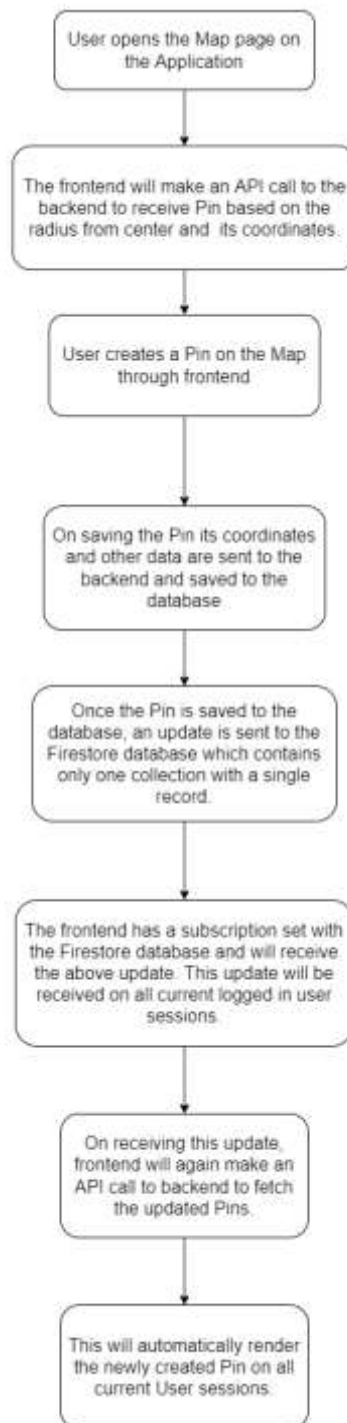
Pin Rating

- Any logged-in user can add the ratings to the pin and the average of the cumulative ratings will be shown.

Pin Comment

- Any logged-in user can comment on the existing pin and all the comments added to the pin will be shown.

Map Integration Flow Diagram for Real Time Pin Creation Updates



3. Bucket List Feature

We have supported the bucket list feature where users can add the specifically created pin to their bucket list and this can help the user to maintain their future dream places. The bucket list feature is private to the specific user. Once the specific pin is added to the bucket list by the user then in order to avoid the user from adding the specific pin again in the bucket list we enabled the delete bucket list option to the user for that specific pin and through which user can delete the specific pin from their bucket list record. If the pin is deleted for the specific location then automatically the bucket list added will show only the remaining pins in the bucket. In addition to this, the user can see the list of all the pins added to the bucket list.

4. Future Trip Feature

We have supported future trips where the enthusiastic traveller can find their travel companion and can accompany the other travellers. Using this feature users can create a future trip where they can enter the trip name, trip description, image, and trip start date for the specific location where the pin is created. Create future trip option will be enabled to the user only if the pin is created on the specific location, else the option will be disabled to the user. This feature is publicly visible to all the users and so another user can view all the future trips created on that specific pin by navigating to the future trips tab. It will be only in read view mode for the other users who are viewing the future trips. There is the **‘Your Future Trips’** tab that enables the functionality where users can see their created future trips diary. Moreover, users can edit and delete their created future trips and on deleting the future trip the pin won't be deleted so they are loosely coupled. Also, the user cannot create future trips for past dates.

5. Blog Feature

The blog feature of the application is where travel enthusiasts can share their experiences and stories. This feature can be also called a **‘travel diary’** which can be helpful to other users because it can act as a treasure box of knowledge for them. Using this feature users can navigate to the **‘Blog Feed’** feature where they can see all the posted blogs, they can also add a new blog, and navigate to the **‘Your Blog’** to view their created blogs. The blog feed is a view-only mode. Any logged-in user can add and view comments on the blogs.

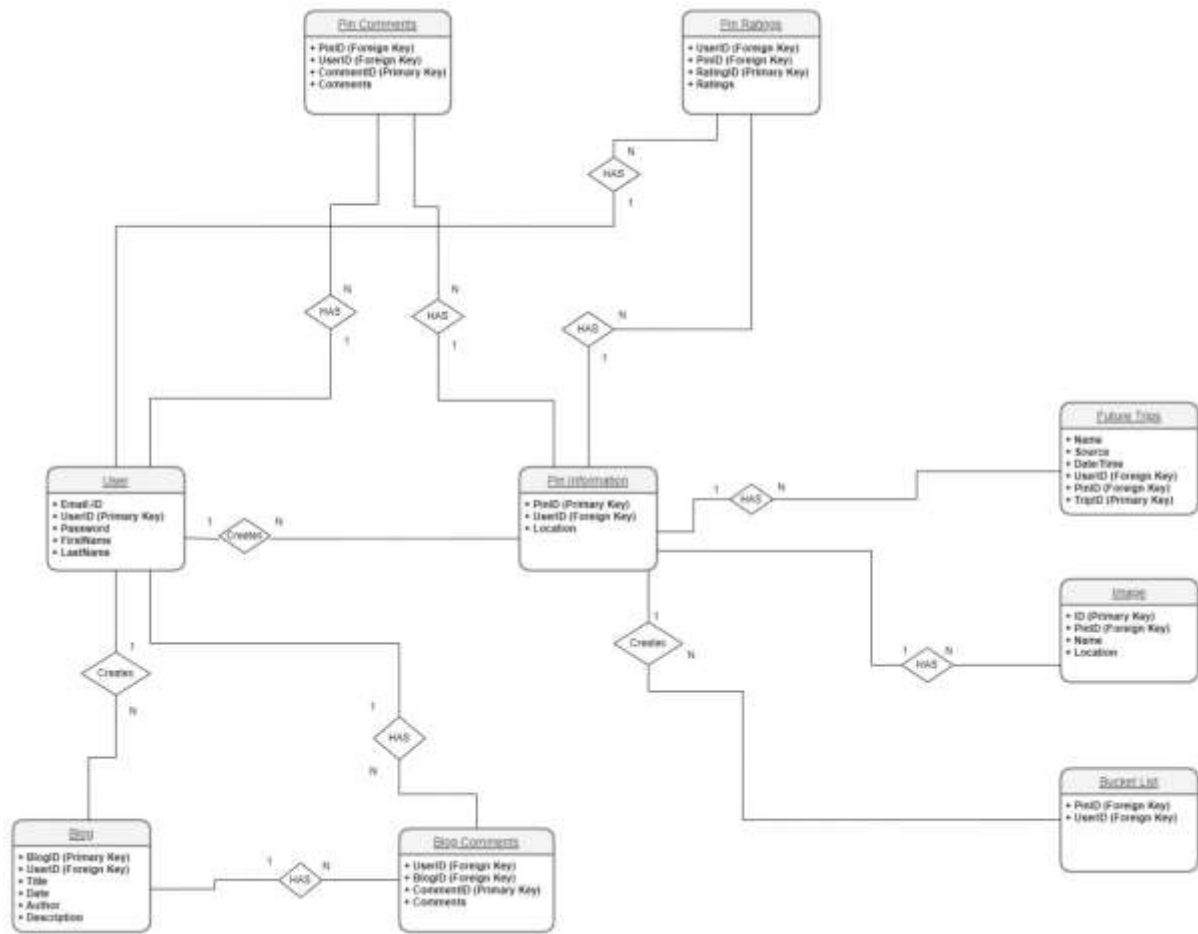
6. User Profile Feature

We have supported the user profile feature where users can view their profile information such as name and email id. Also, they can edit their first name and last name or upload new images on their profiles.

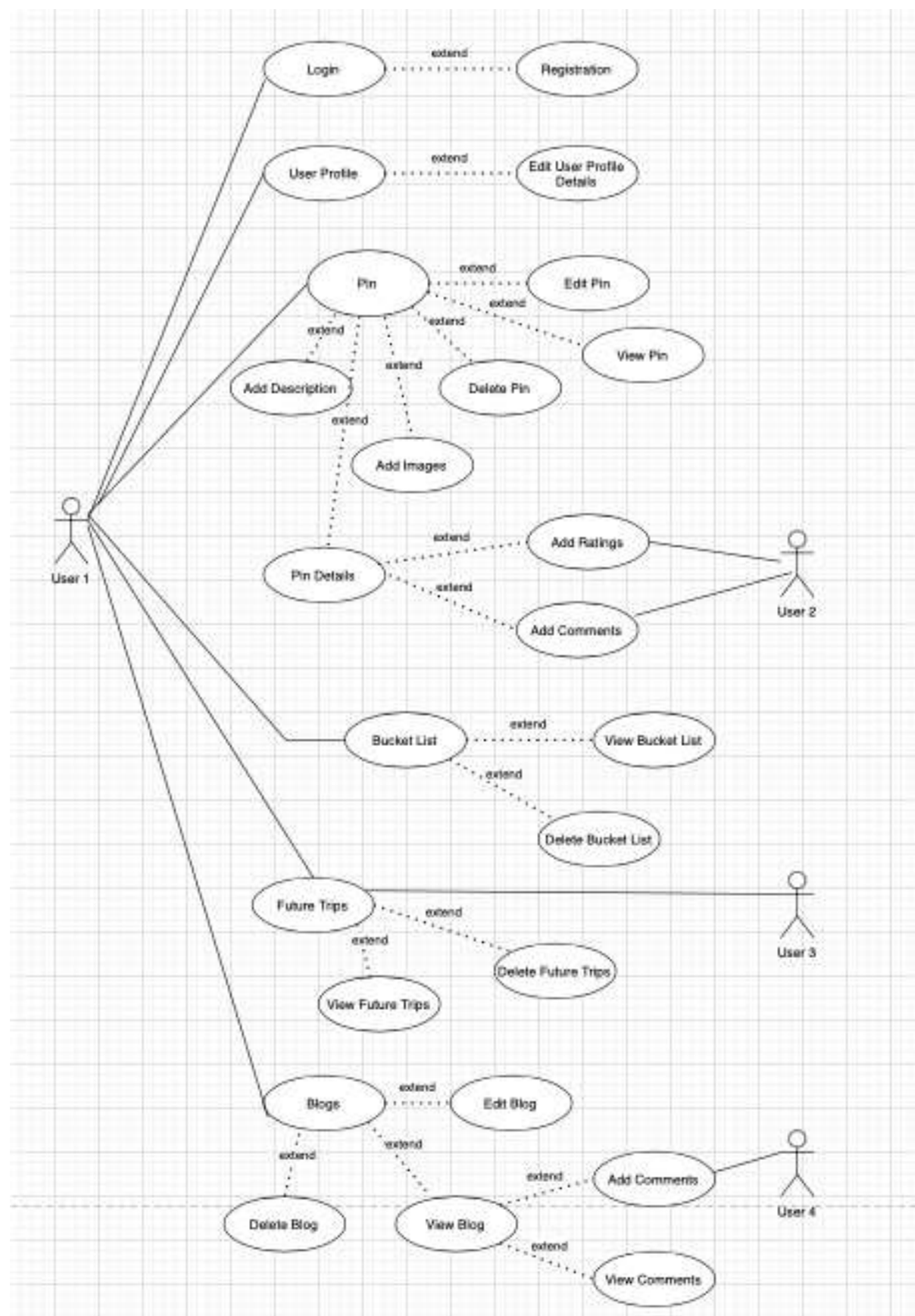
7. Log Out

We have supported the logout feature where users can log out from the Wanderer application and tokens of that specific user will be deleted from the cookies.

ER-Diagram

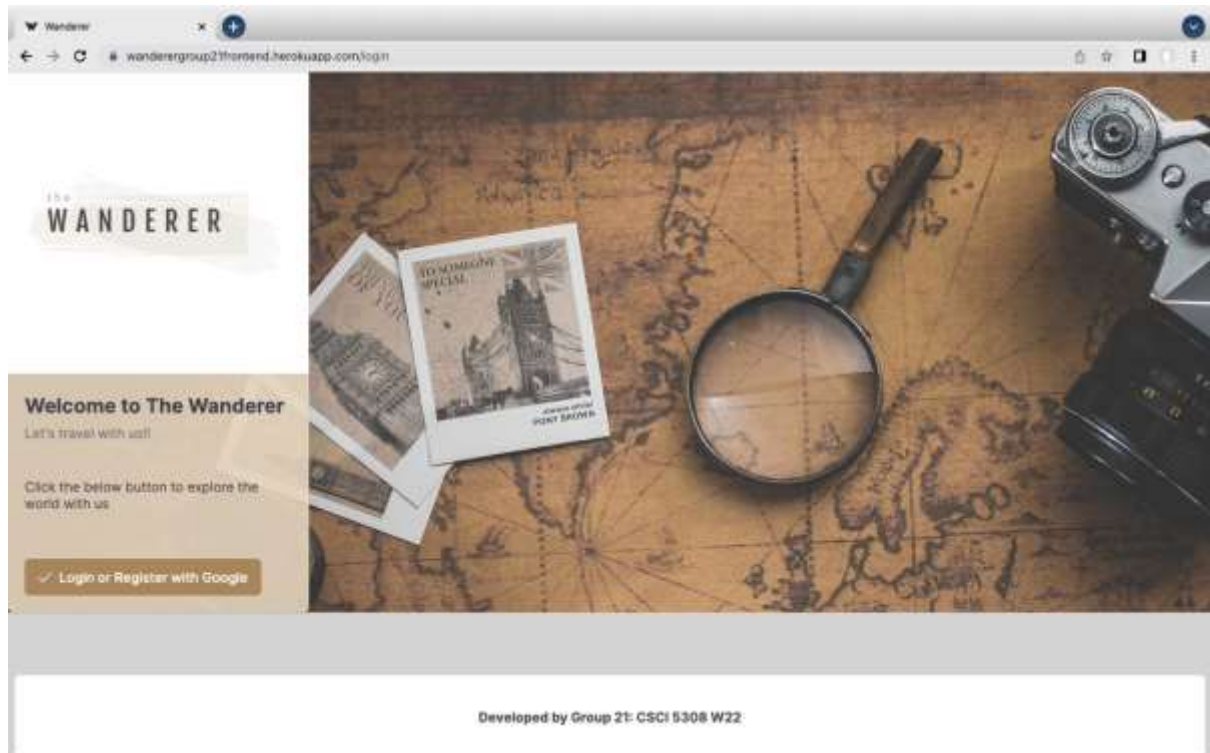


Use-Case Diagram

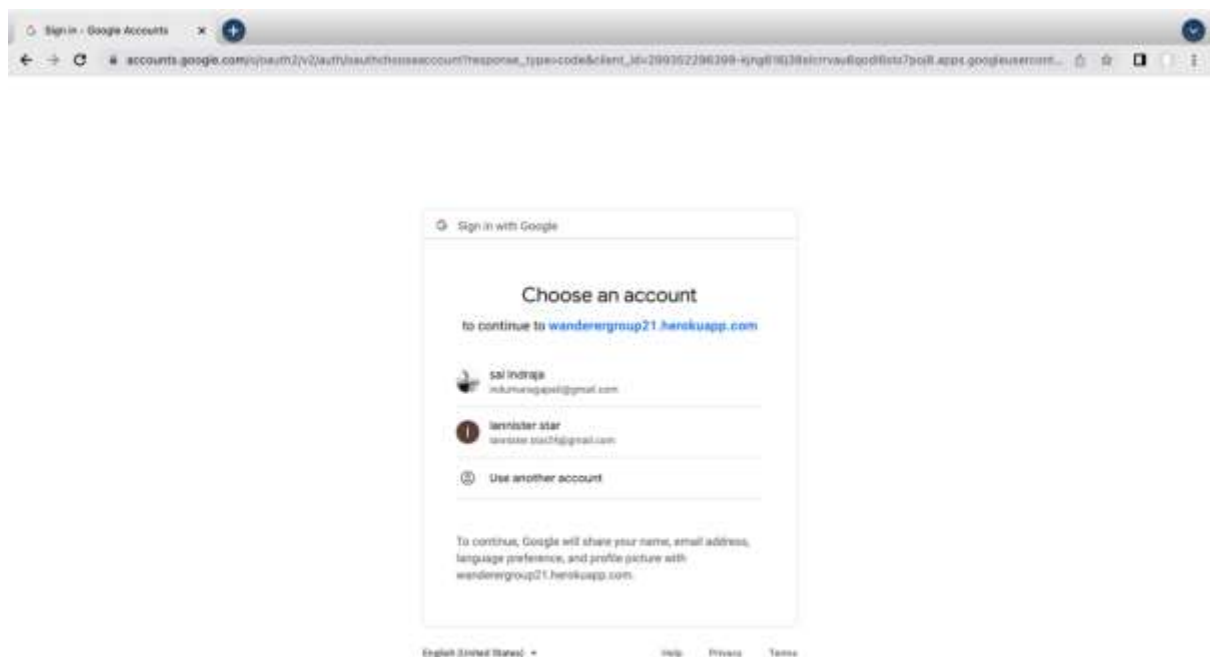


Usage Scenario

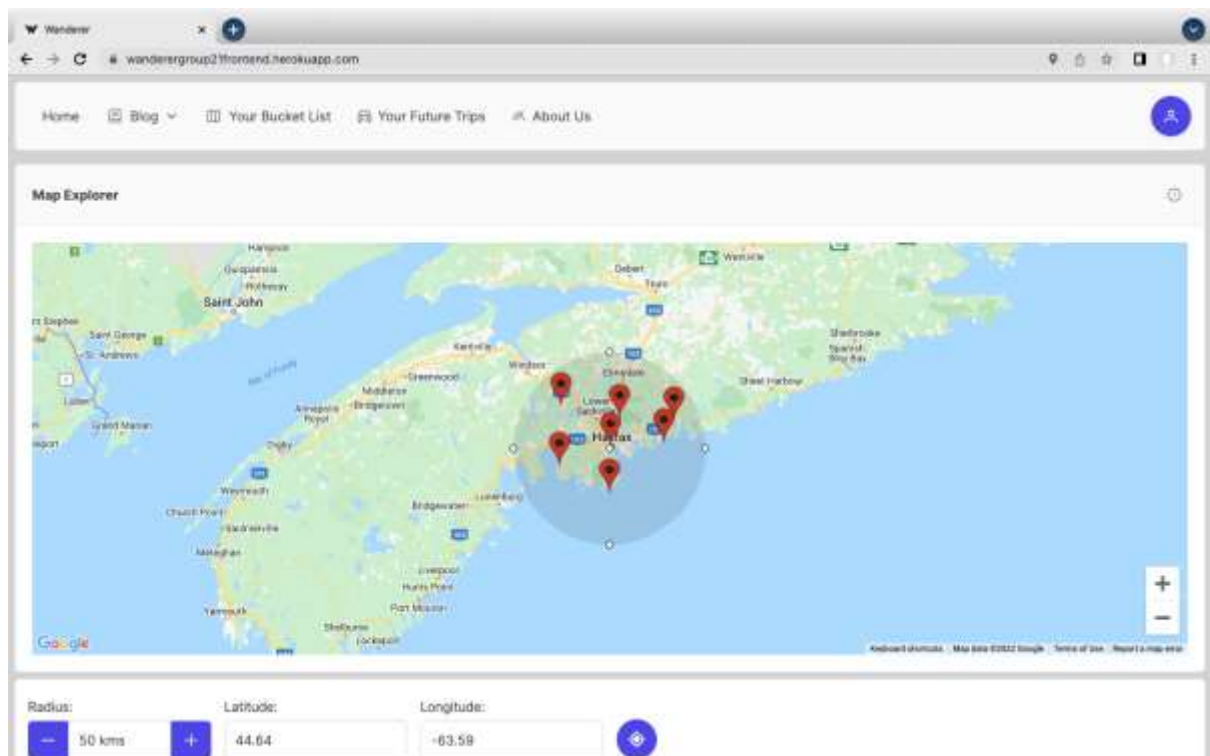
This is Login or Register page for WANDERER



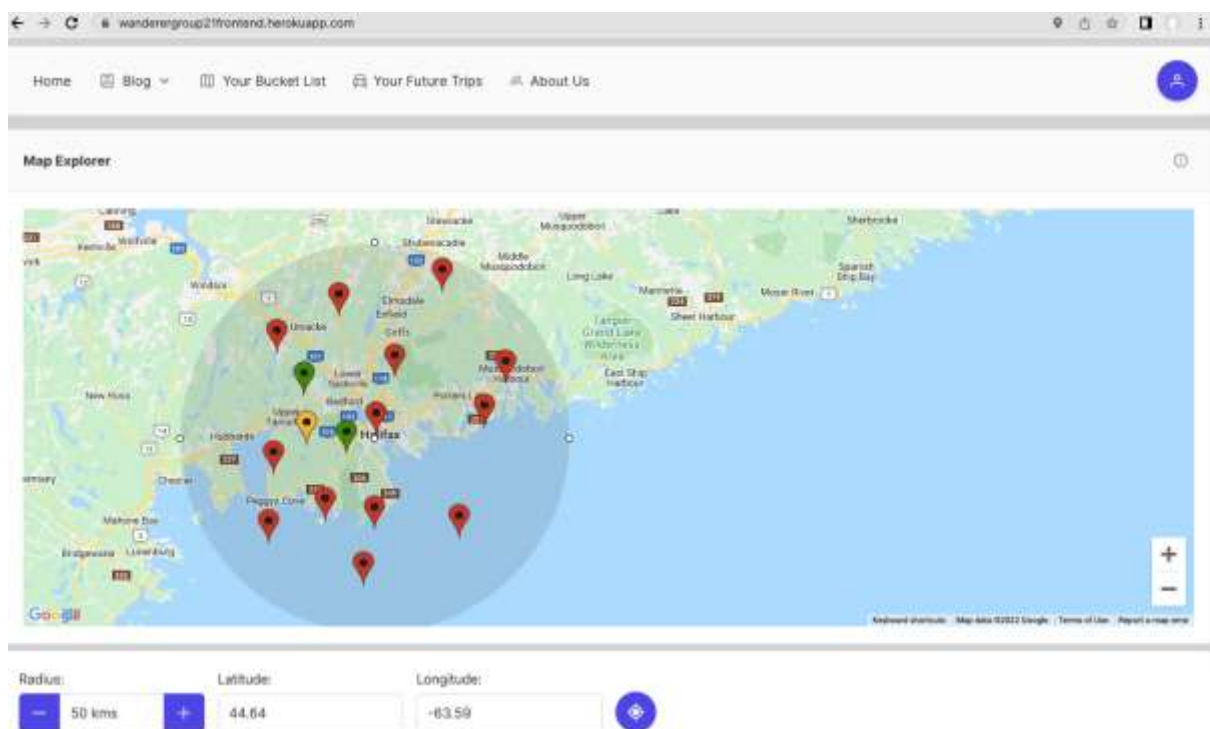
Once the user clicks Login or Registration, he will be redirected to “Sign in with Google” page.



Once the user is logged in, this is the home page with the map view.



Create a pin by clicking the location on the map



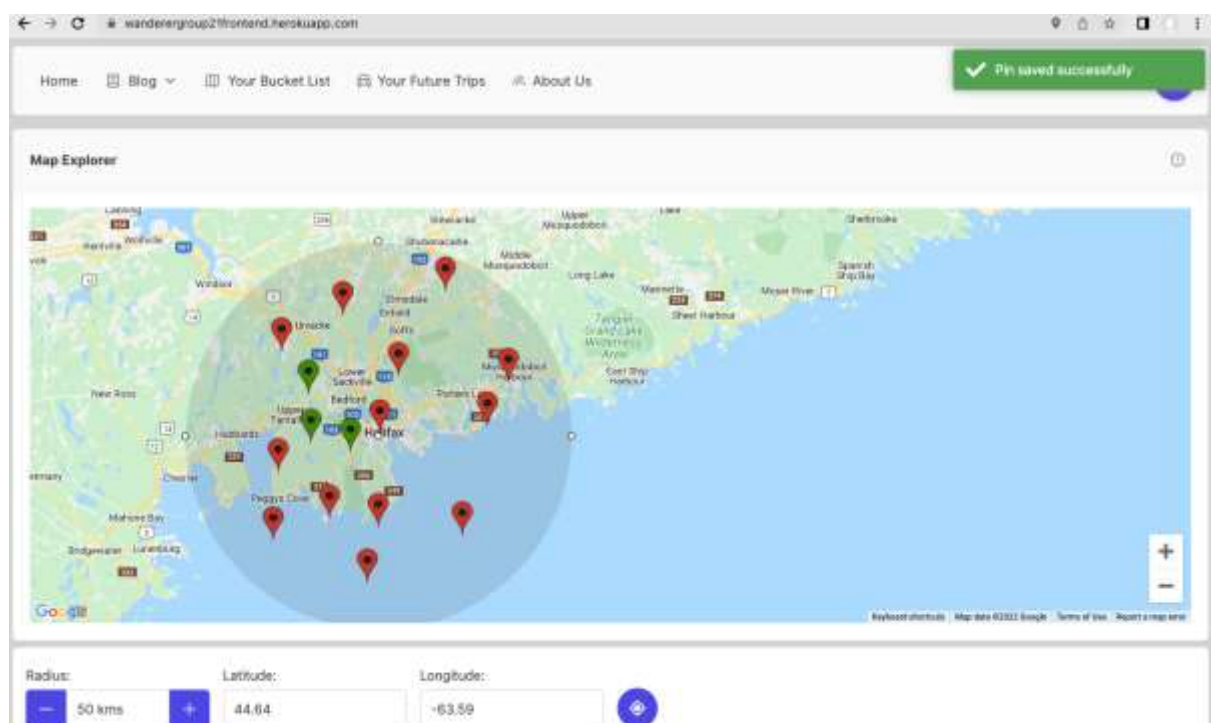
Add the location name, description, and the user can add up to five images

The screenshot shows a modal window titled "Add/Edit Pin Details" with three tabs: "Pin Info", "Create Your Future Trip", and "Future Trips". The "Pin Info" tab is active. It contains the following fields:

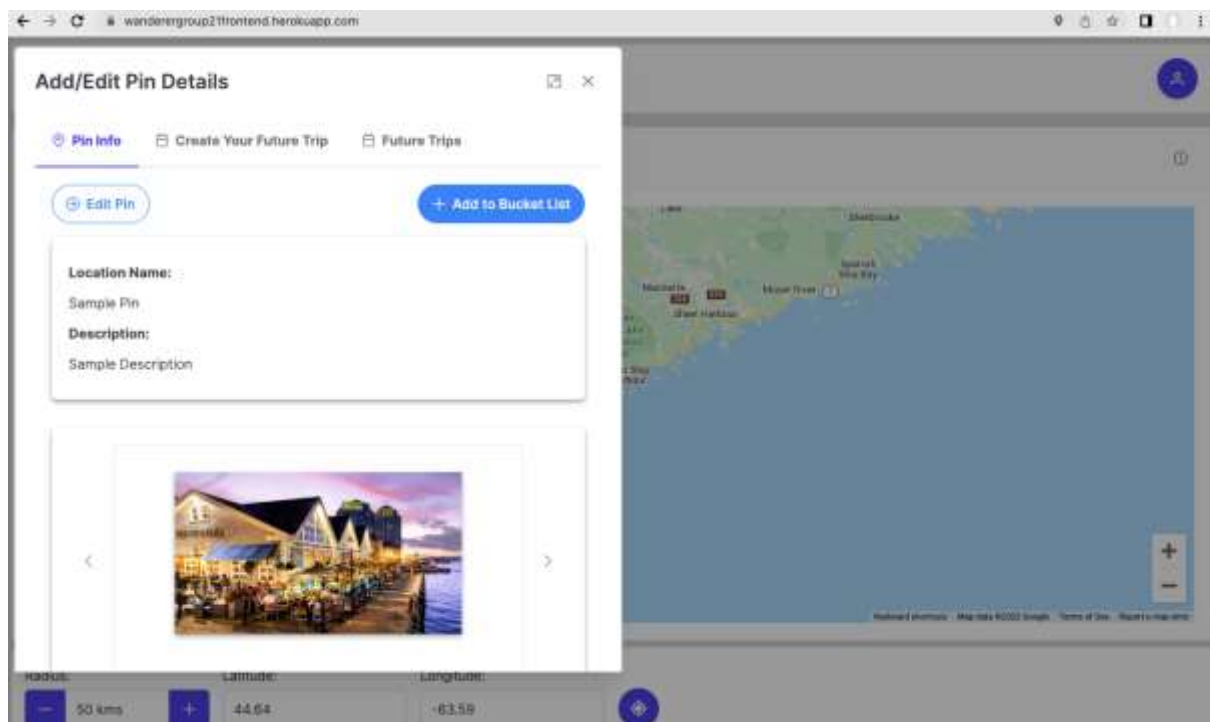
- Location Name:** A text input field containing "Halifax Waterfront".
- Description:** A text area containing "Beautiful waterfront boardwalk, even on a cold, though sunny day. Different things happening along the way, restaurants and cafes, and even a small Christmas fair with a stage and a cool young band playing."
- Latitude:** A text input field containing "44.64554126032372".
- Longitude:** A text input field containing "-63.57045662294489".

At the bottom of the modal, there are four buttons: "+ Choose" (with a plus icon), "X Cancel", "Edit Coordinates" (with a pencil icon), and "Save" (with a checkmark icon). The background shows a map of the Halifax area with several pins.

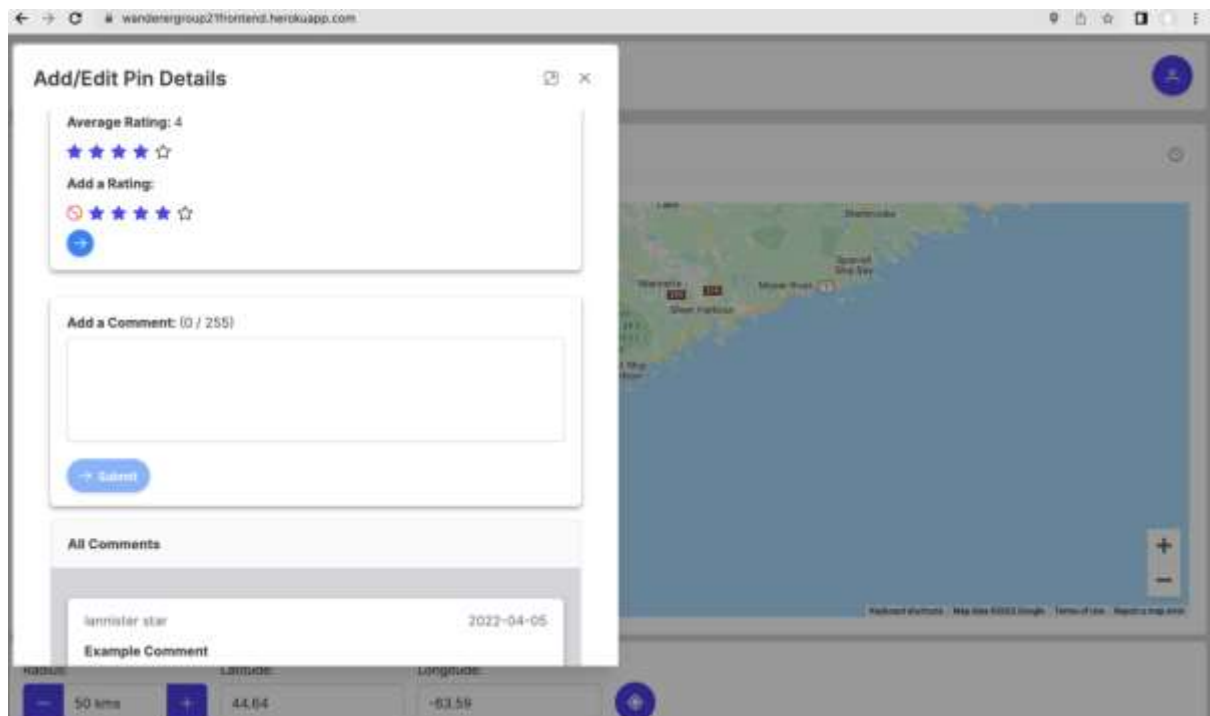
Pin has been successfully saved



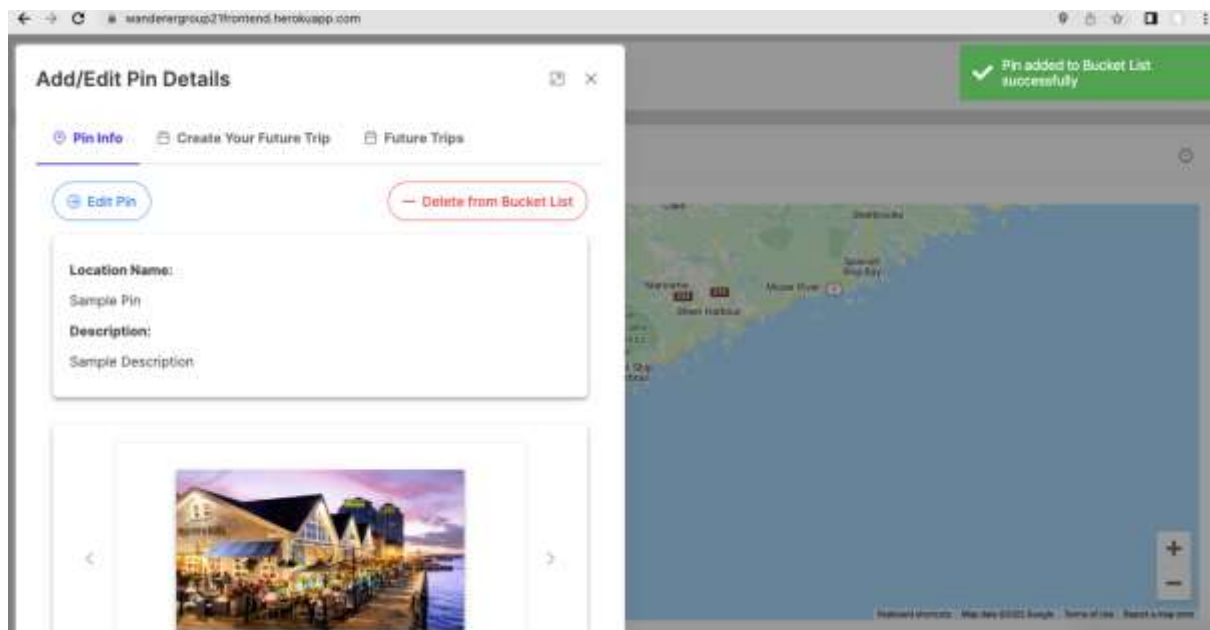
Click the saved pin to edit.



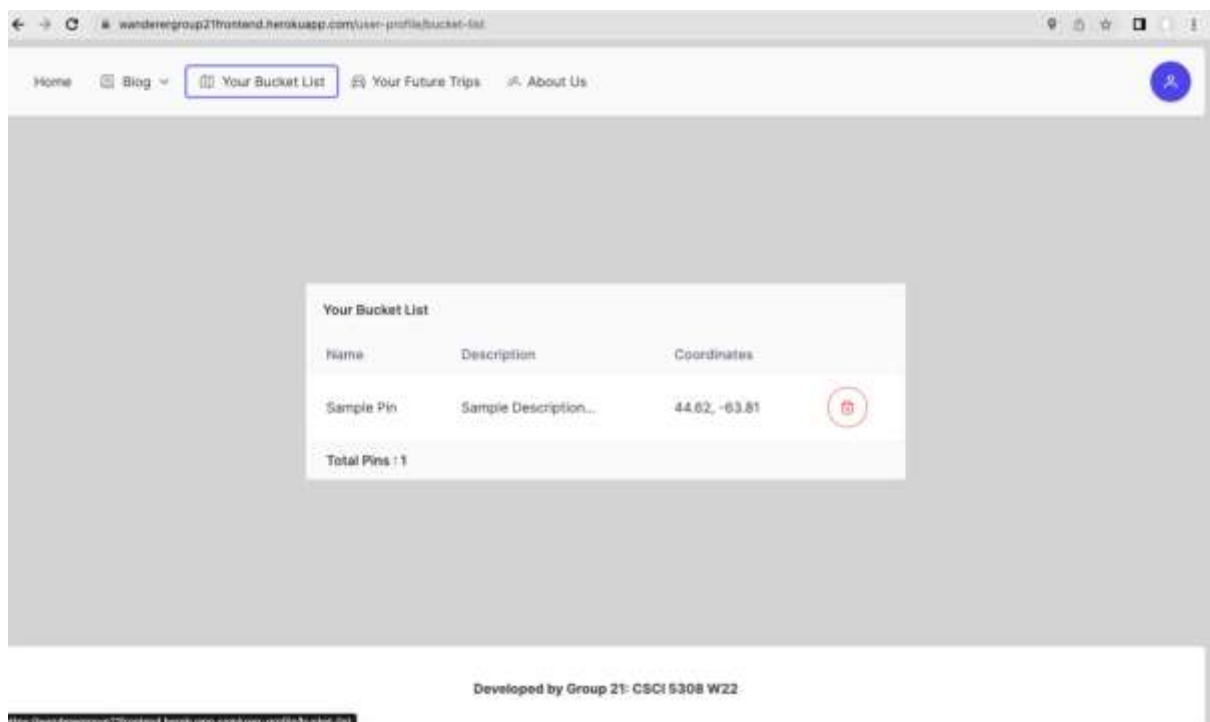
Other logged-in users can add ratings and comments.



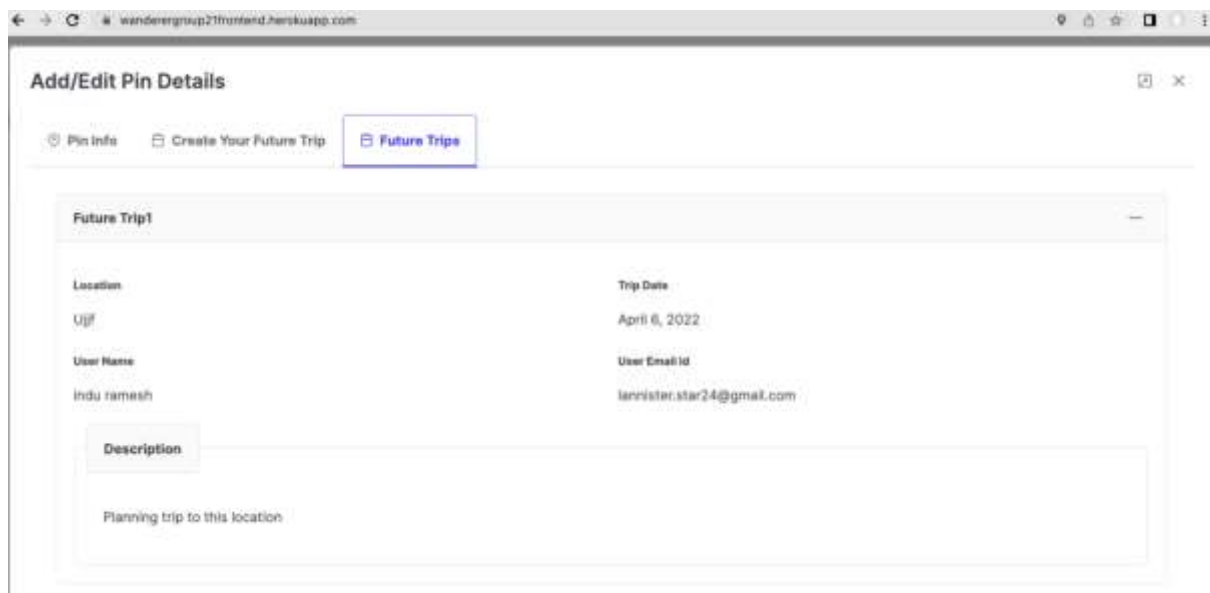
Click the “Add to Bucket List” button to add this pin to the bucket list



Your Bucket List shows all the pins that are added by the user



Select a pin and click on “Create Your Future Trip” to create a new future trip. The existing and newly created future trips for that particular pin can be seen in the “Future trips” section

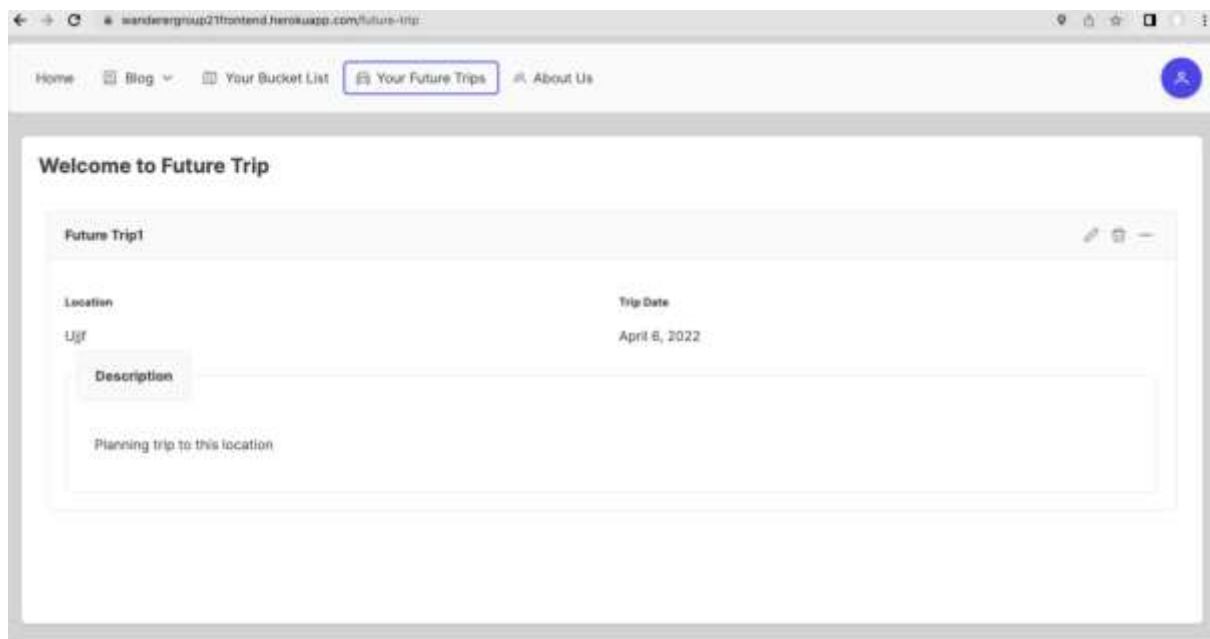


The screenshot shows a web browser window with the URL `wanderergroup21frontend.herokuapp.com`. The page title is "Add/Edit Pin Details". There are three tabs: "Pin Info", "Create Your Future Trip", and "Future Trips", with the last one being active. The main content area is titled "Future Trip1" and contains a form with the following fields:

Location	Trip Date
Ujjf	April 6, 2022
User Name	User Email Id
Indu ramesh	lennister.star24@gmail.com

Below the form is a "Description" section with the text "Planning trip to this location".

“Your Future trips” lists all the trips created by the user

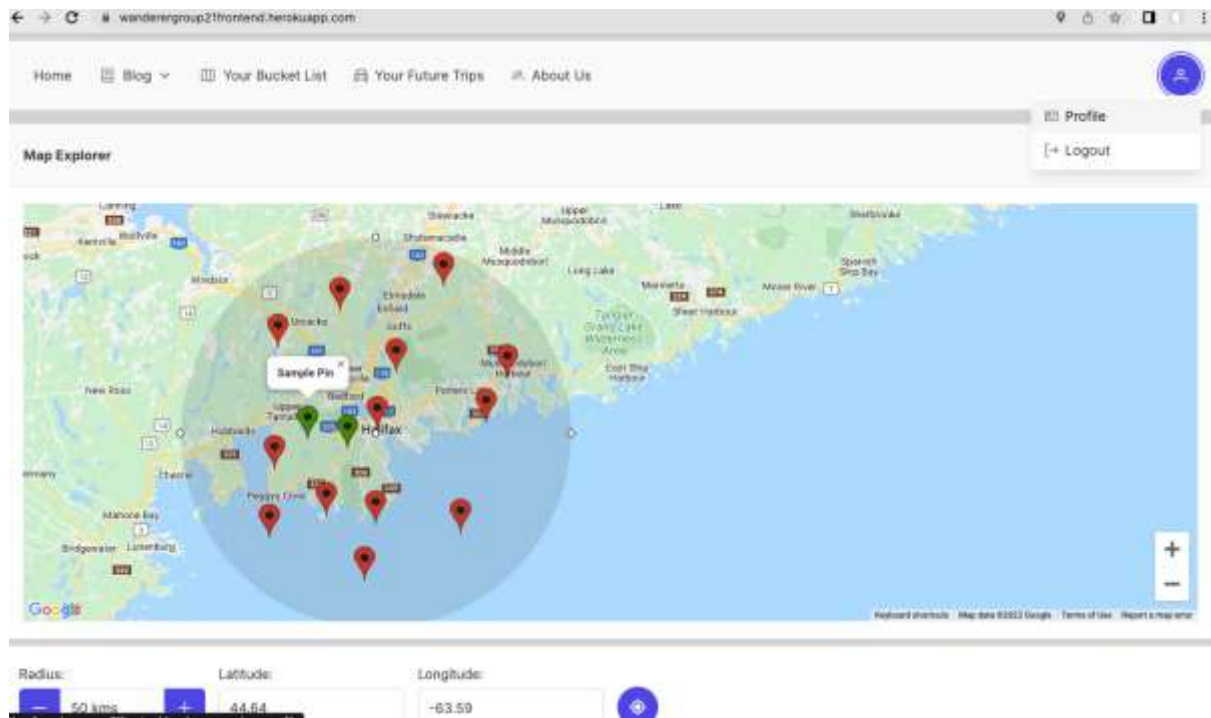


The screenshot shows a web browser window with the URL `wanderergroup21frontend.herokuapp.com/future-trip`. The page title is "Welcome to Future Trip". The navigation bar includes links for "Home", "Blog", "Your Bucket List", "Your Future Trips" (which is active), and "About Us". The main content area is titled "Future Trip1" and contains a form with the following fields:

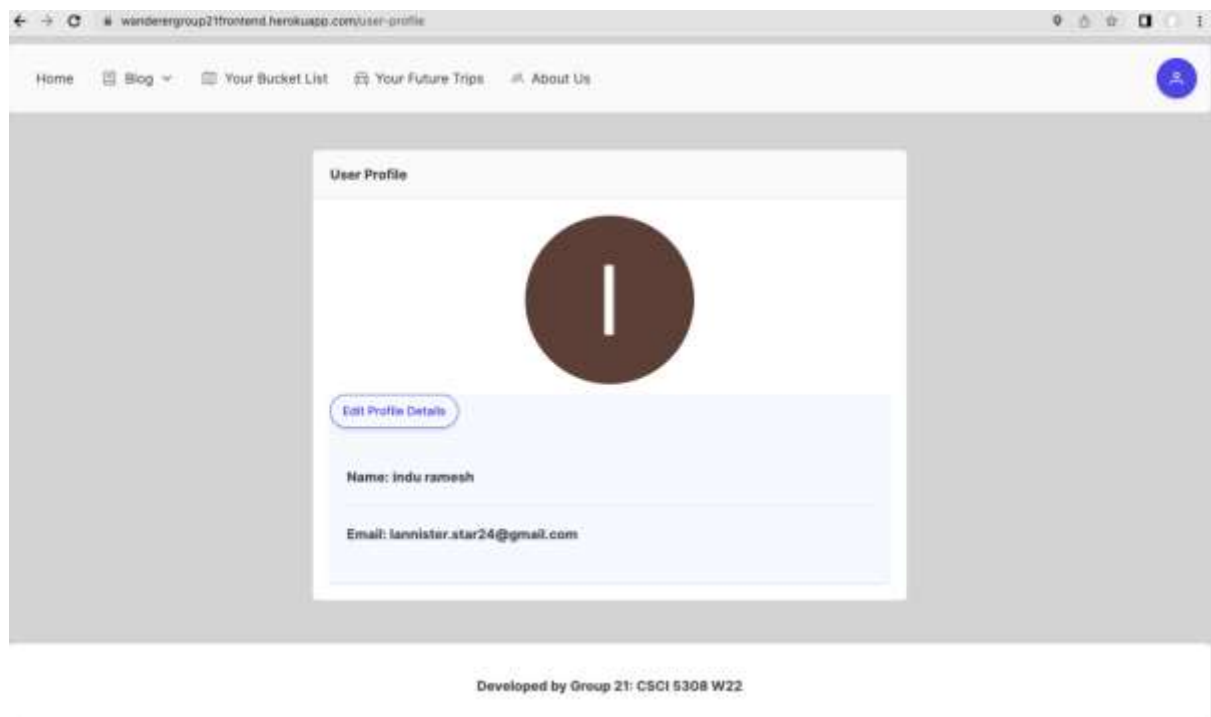
Location	Trip Date
Ujjf	April 6, 2022
Description	
Planning trip to this location	

Developed by Group 21: CSCI 5308 W22

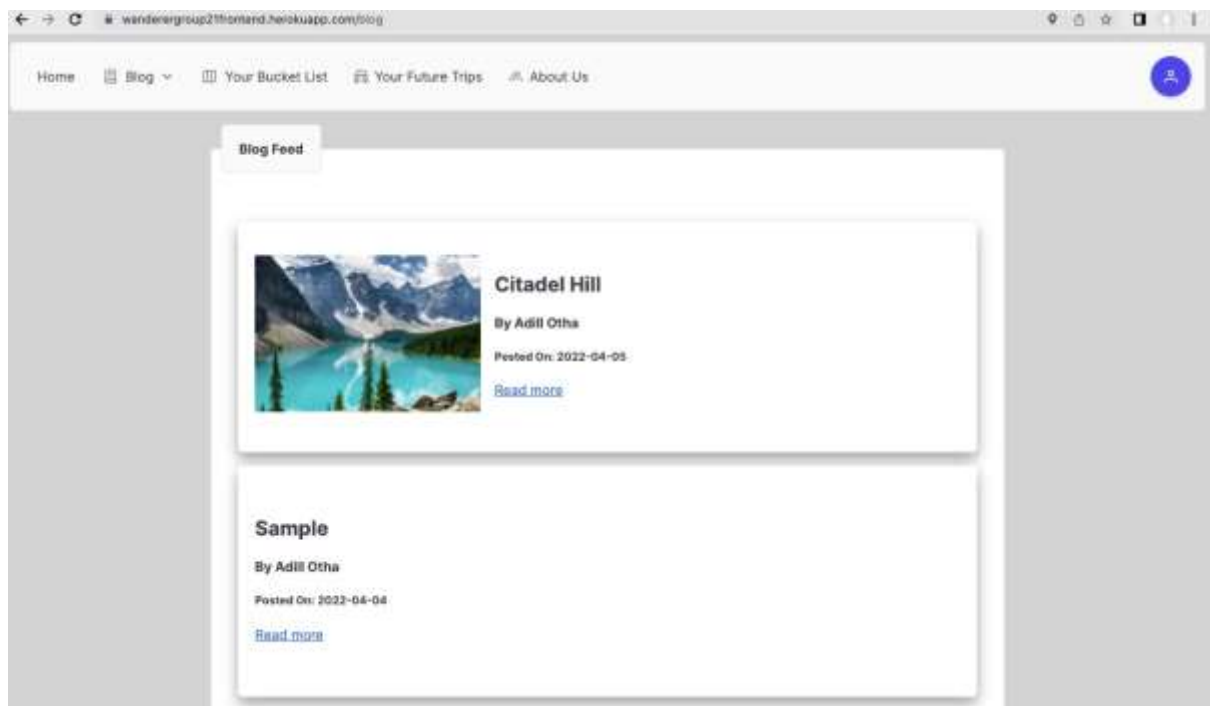
To navigate to User profile details and Logout from the application, users can click the “Profile” icon



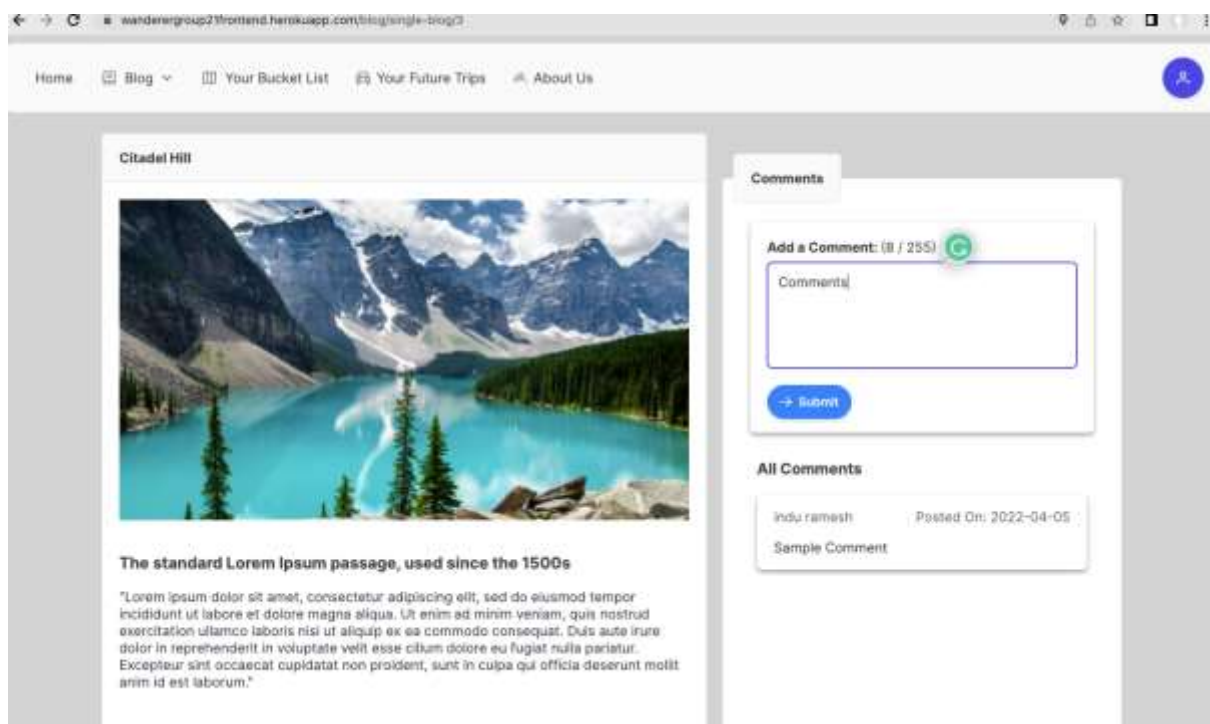
User can view or edit the profile details

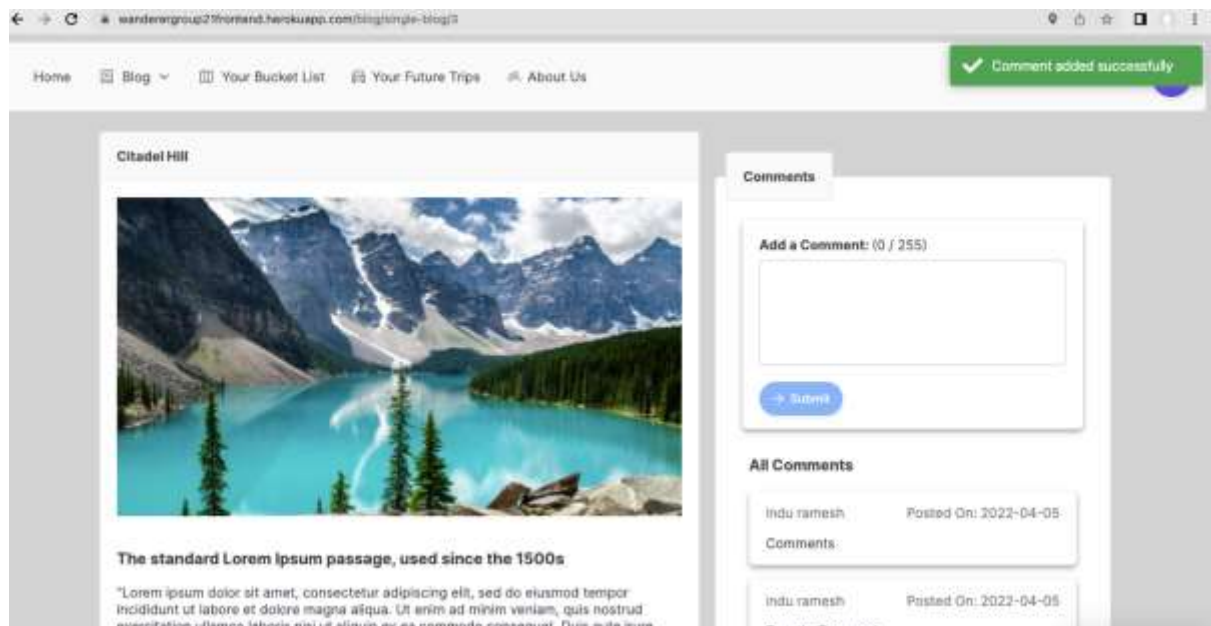


Blog feed shows the blogs added by all the users

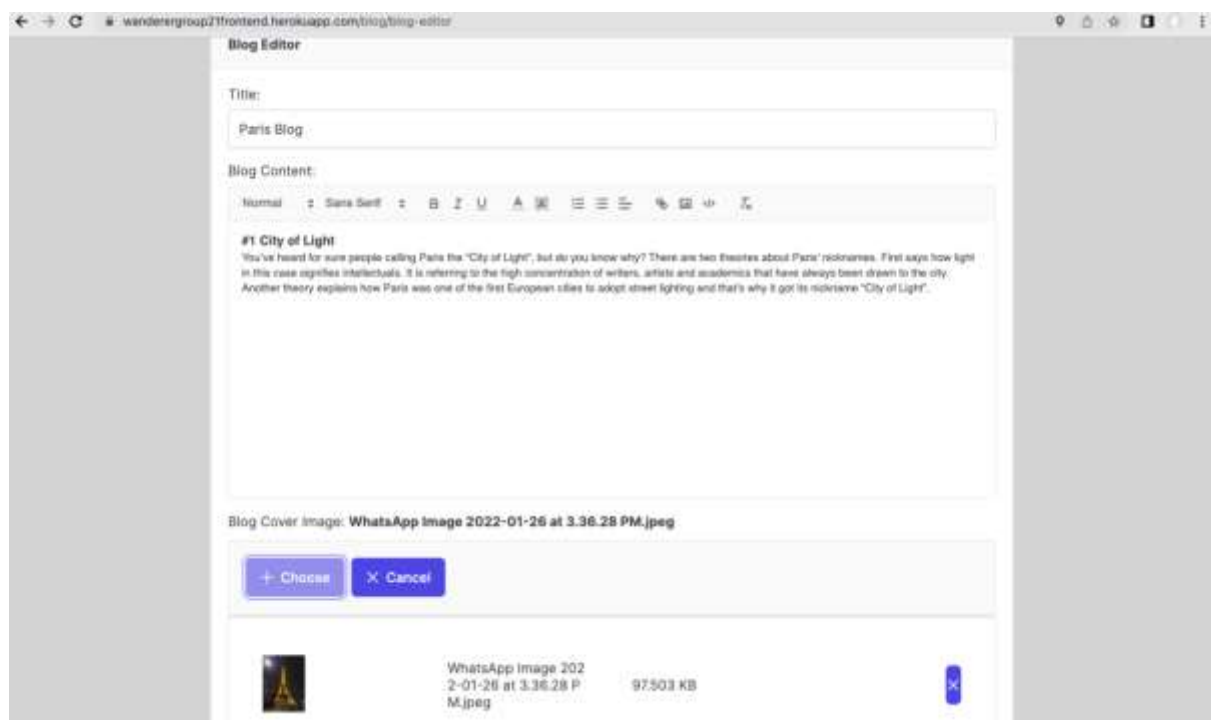


Other logged-in users can add comments to the existing blogs

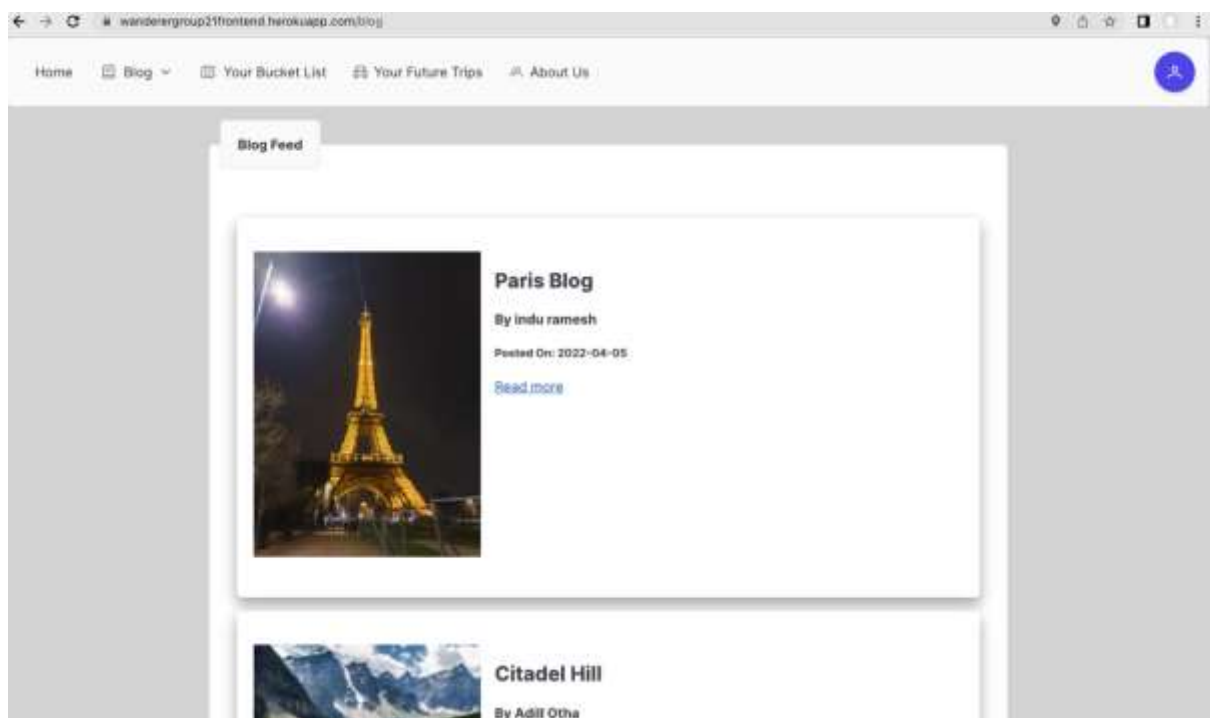
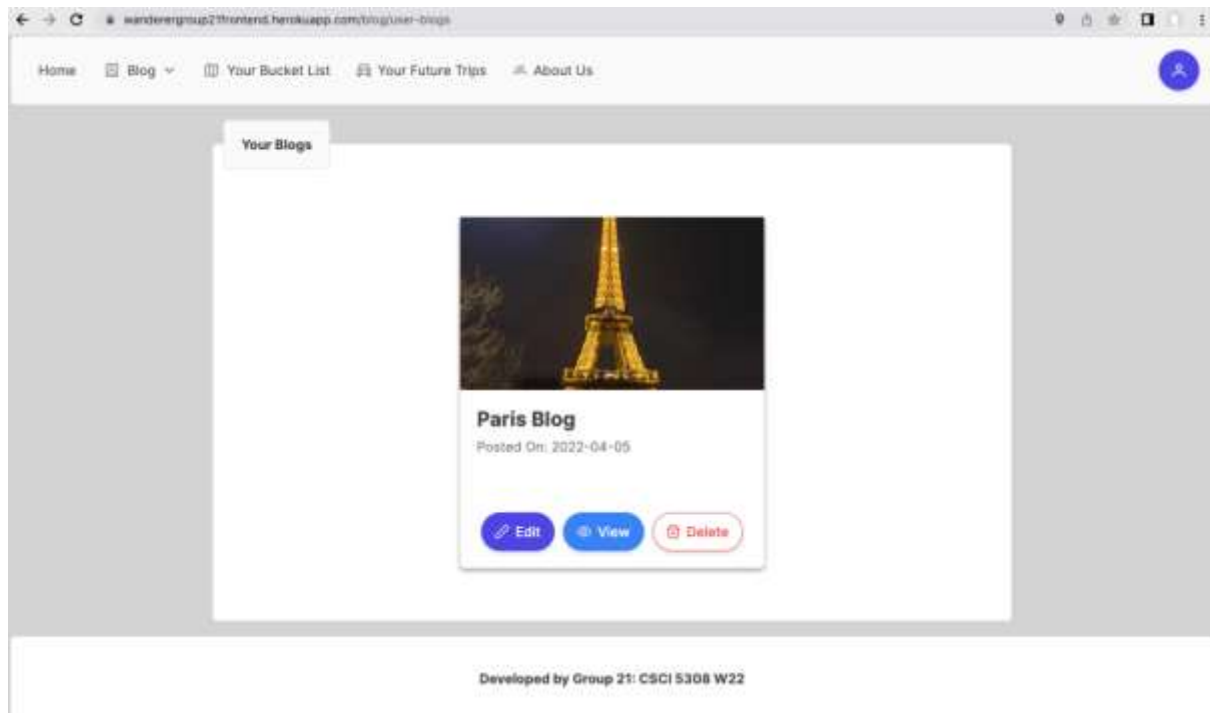




Users can create a Blog and can add an image

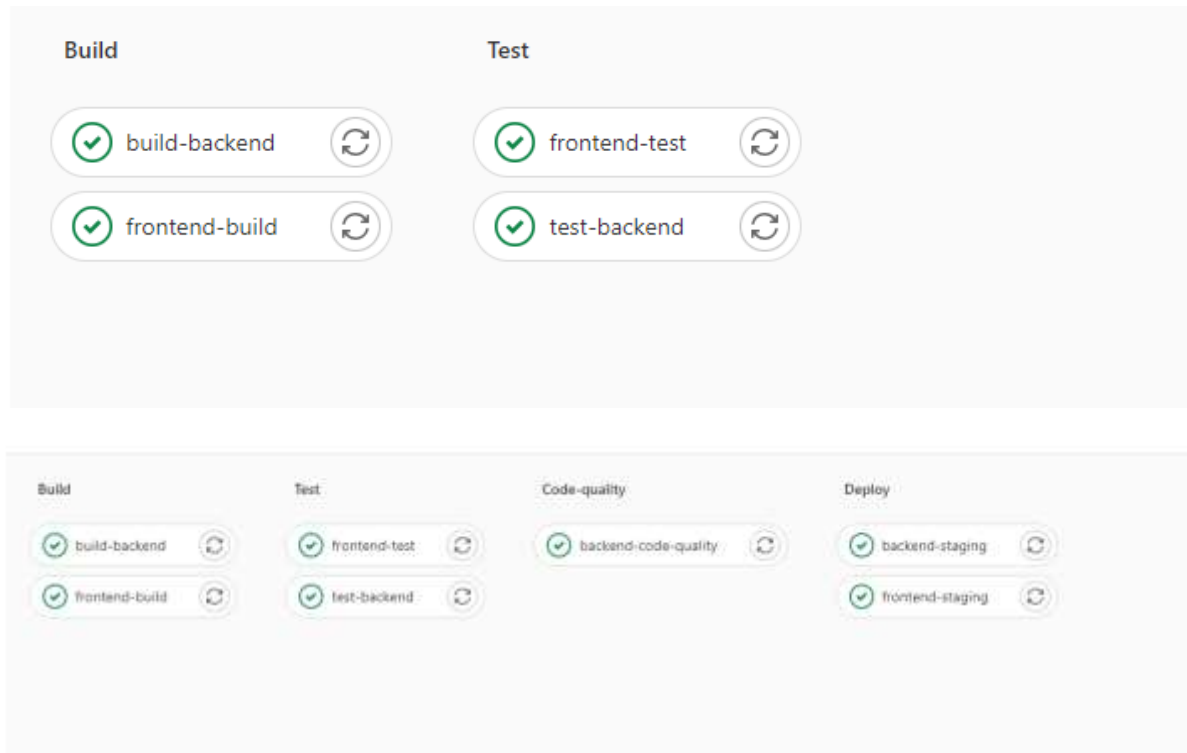


Once the blog is created, the user will be redirected to “Your Blogs” page, where all the blogs of that particular user will be visible. From here users can edit or delete the blog.



Deployment Working Cycle

We decided to go with the monolithic approach-based deployment process, so we created a single repository for our project which consists of both backend and frontend code. We implemented the pipeline for CI and PROD environments by introducing the build and test stage for both backend and frontend respectively, code quality and deploy stages.



We introduced the rules for the pipeline by creating `.yaml` a script in such a way that the build and test stage pipeline will trigger only when a merge request is created and if only the pipeline is succeeded for the build and test stage then only all the stages along with the deployment and code quality stage will be triggered when the merge is triggered. Moreover, we have implemented the script in such a way that if the changes are made only in the backend, then the build and test stage of the backend will only be triggered at the time of merge request and if the changes are made to the frontend, then only build and test stage for a frontend will be triggered.

We also enabled the branches to rule in the pipeline, where in the first stage the code will be merged with the develop branch only, and after testing everything on the develop stage the merge request for develop to master will be created where only the production stages of frontend and backend staging will be triggered. Apart from this, the test stage will run only if the build stage passes, and frontend staging for the deployment run only if the backend staging is passed and this, we have achieved by using the “**needs**” tag in the `.yaml` script.

```

.base-rules:
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
      when: always
    - if: $CI_COMMIT_BRANCH == 'master'
      when: always
    - if: $CI_PIPELINE_SOURCE == "push"
      when: never
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      changes:
        - $RULES_CHANGES_PATH
    - when: manual
      allow_failure: false

```

In the above image, we are defining the base rules on when to trigger the pipeline based on the branches.

```

.backend:
  extends: .base-rules
  variables:
    RULES_CHANGES_PATH: "backend/**/*"

.frontend:
  extends: .base-rules
  variables:
    RULES_CHANGES_PATH: "frontend/**/*"

```

In the above image, we are implementing the script in such a way that pipeline will be triggered for the backend and frontend depending upon the changes made. For which the changes that have been made can be made using **RULES_CHANGES_PATH**.

```

backend-staging:
  image: ruby:latest
  stage: deploy
  before_script:
    - cd backend
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
  script:
    - dpl --provider=heroku --app=wanderengroup21 --api-key=$HEROKU_STAGING_API_KEY_BACKEND
  only:
    - develop

frontend-staging:
  stage: deploy
  image: ruby:latest
  needs: [ "backend-staging" ]
  before_script:
    - cd frontend
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
  script:
    - dpl --provider=heroku --app=wanderengroup21frontend --api-key=$HEROKU_STAGING_API_KEY_BACKEND
  only:
    - develop

```

The above image defines the frontend and backend staging for the develop branch only.

```

- develop

backend-staging-production:
  image: ruby:latest
  stage: deploy
  before_script:
    - cd backend
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
  script:
    - dpl --provider=heroku --app=wanderer-backend-live --api-key=$HEROKU_STAGING_API_KEY_BACKEND
  only:
    - master

frontend-staging-production:
  stage: deploy
  image: ruby:latest
  needs: [ "backend-staging-production" ]
  before_script:
    - cd frontend
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
  script:
    - dpl --provider=heroku --app=wanderer-live --api-key=$HEROKU_STAGING_API_KEY_BACKEND
  only:
    - master

```

The above image defines the frontend and backend staging for the master only.

We have also implemented the profile-based system on the backend side by introducing the application-properties files for the ci and prod environment. Using this profile-based management system will help in managing different configurations for multiple environments. In order to activate the specific profile, we configured the environment variable on the Heroku application.

Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

Hide Config Vars

SPRING_PROFILES_ACTIVE	ci	
KEY	VALUE	Add

Activate profiles for CI environment on Heroku

Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

Hide Config Vars

SPRING_PROFILES_ACTIVE	prod	
KEY	VALUE	Add

Activate profiles for the PROD environment on Heroku

We also implemented the profile-based management on the front by setting up the proxy for the specific environment variable and we configured that environment variables on the Heroku application for the frontend.

Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

Hide Config Vars

API_HOST	https://wanderengroup21.herokuapp.com	
NODE_BUILD_FLAGS	--configuration=ci	
KEY	VALUE	Add

Configured environment variables for the CI environment

If no environment is specified then it will take the production environment as a default environment. So, no need to set up the variable for the production on the Heroku application. In order to deploy our frontend application on Heroku, we introduced the server.js layer on the frontend side which will serve the frontend application on Heroku. Moreover, included proxy middleware which intercepts the frontend API call to our environment-specific backend server.

```
// Proxy api to a backend server.
const apiProxy = middleware.createProxyMiddleware({ target: process.env.API_HOST, changeOrigin: true });
```

We also configured our backend host URL in the frontend application environment variable on Heroku and which will be used for setting up the proxy for API endpoints.



Backend Host URL for PROD Stage



Backend Host URL for CI Stage

We are serving our frontend application on Heroku using expressjs server.

Backend Dependencies

We have included the following backend dependencies in our pom.xml file.

1. In order to support the spring-boot application, we have included the following dependencies of the spring boot

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

2. In order to enable the MySQL connection, we included the following dependency for MySQL

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.25</version>
</dependency>
```

3. In order to enable the swagger for making API testing easy by making it more user convenient we included the following dependency

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
```

4. For supporting the real-time update feature in our application we included the following firestore dependency for it

```
<dependency>
    <groupId>com.google.firebase</groupId>
    <artifactId>firebase-admin</artifactId>
    <version>8.0.1</version>
    <scope>compile</scope>
</dependency>
```

5. Dependency for enabling the mocks for test cases

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>3.9.0</version>
</dependency>
```

6. Dependency for enabling the Junit testing

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.1</version>
    <scope>test</scope>
</dependency>
```

7. Dependency for enabling the jwt token

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-jwt</artifactId>
    <version>1.1.1.RELEASE</version>
</dependency>
```

8. Dependency for creating the in-memory database for the test cases

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>test</scope>
</dependency>
```

Frontend Dependencies

1. Dependency for enabling the express for serving the application on Heroku

```
@angular/cli@15.0.0: "15.0.0"
"express": "^4.17.3",
```

2. Dependency for enabling the firebase for supporting real-time update feature

```
"firebase": "^9.6.7",
```

3. Dependency for enabling the proxy middleware service

```
"http-proxy-middleware": "^2.0.3",
```

4. Dependencies for using angular prime ng functionalities

```
"ngx-spinner": "^13.1.1",
"ngx-toastr": "^14.2.2",
"primeflex": "^3.1.2",
"primeicons": "^5.0.0",
"primeng": "^13.1.1",
"quill": "^1.3.7",
```

5. Dependency for enabling the testing for angular

```
"devDependencies": {
  "@angular-devkit/build-angular": "~13.2.2",
  "@angular/cli": "~13.2.2",
  "@angular/compiler-cli": "~13.2.0",
  "@types/jasmine": "~3.10.0",
  "@types/mapbox-gl": "^2.6.1",
  "@types/node": "^12.11.1",
  "jasmine-core": "~4.0.0",
  "karma": "~6.3.0",
  "karma-chrome-launcher": "~3.1.0",
  "karma-coverage": "~2.1.0",
  "karma-jasmine": "~4.0.0",
  "karma-jasmine-html-reporter": "~1.7.0",
  "typescript": "~4.5.2"
}
```

Build and Compile Instructions

To build and compile the backend spring-boot application along with test cases
mvn clean install

To build and compile the frontend Angular application

```
D:\group21\group21\frontend>ng serve --proxy-config proxy.conf.json_
```

Run the above command to compile the angular project.

ng build command to build the project

ng test command to execute the unit tests

Best Practices

We followed the below best practices for the Backend:

1. Profile Based Deployment
2. API versioning
3. We segregated the deployment process into three branch stages: feature, develop, and master branch.
4. Separated the configuration from the code.
5. Followed Layered-based Architecture.
6. Added the comments for explaining the '**why**' method is implemented.
7. We implemented the Global Exceptional Handler for customizing the exceptions and following the same exceptional handling practices. This will also keep the exceptions uniform both on the frontend and backend, which indirectly make it easy to handle on the frontend side.
8. We integrated Swagger for testing our APIs. We opted for the option of swagger instead of swagger because of the easy user experience.
9. We integrated H2 in-memory database for using the dummy database while running the test cases instead of using the actual database.
10. We implemented the '**logs**' mechanism to maintain the logs of our application which can be useful to refer to whenever an error is encountered.
11. We designed our database structure through ER diagrams before starting actual implementation.

We followed the below best practices for the Frontend:

1. Module-based structure for each feature.
2. Central configuration for API request redirection to backend and Authorization Header Injection including some cors headers by Intercepting the call.
3. Inceptors are added in the frontend app which intercepts the request and adds the Authorization Header and Cors Header and forwards the request. If the Backend responded with 401 then it will redirect to a Login Page.
4. Added the comments for explaining the '**why**' method is implemented.
5. Added the toaster messages to inform the user about performing a specific action.
6. We considered the main point of making the UI experience seamless for the users.
7. We implemented the frontend parallelly with the backend implementation.
8. We created wireframes before starting the frontend implementation to gain insight into how our application will look and which eventually makes implementation easy because we have pictures in front of us.
9. We decided to use the Angular PrimeNG package for making the UI more user attractive and convenient.

Additional functionalities and features

1. Map Integration

- Implemented the map interface in such a way that it can be controlled using the mouse or the keyboard.
- The controls for the map are bound using two-way binding, so any changes made using the mouse will also be reflected on the keyboard input controls and vice-versa.
- The state of the pin will be shown using different colors for a better user experience.

2. Future Trip

- Achieved all CRUD operations for the future trip module even though we were required to only display the future trips to the users.
- Integrated a WYSIWYG editor for the future trip description instead of plain text and also supported the image uploading functionality.

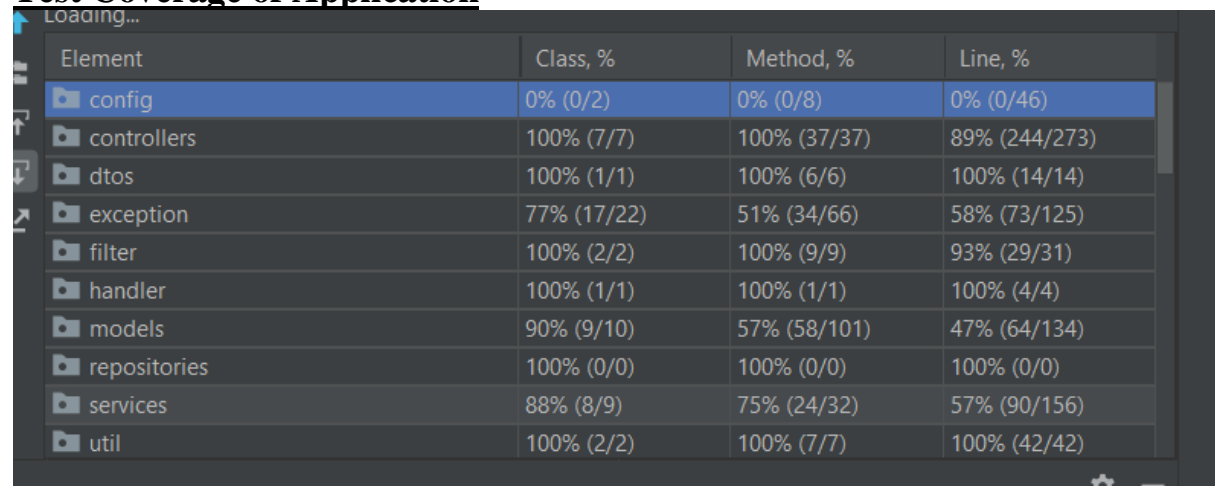
3. Blog

- Integrated a WYSIWYG editor for the blog description instead of plain text.

Application PROD Access Link

<https://wanderer-live.herokuapp.com/login>

Test Coverage of Application



Element	Class, %	Method, %	Line, %
config	0% (0/2)	0% (0/8)	0% (0/46)
controllers	100% (7/7)	100% (37/37)	89% (244/273)
dtos	100% (1/1)	100% (6/6)	100% (14/14)
exception	77% (17/22)	51% (34/66)	58% (73/125)
filter	100% (2/2)	100% (9/9)	93% (29/31)
handler	100% (1/1)	100% (1/1)	100% (4/4)
models	90% (9/10)	57% (58/101)	47% (64/134)
repositories	100% (0/0)	100% (0/0)	100% (0/0)
services	88% (8/9)	75% (24/32)	57% (90/156)
util	100% (2/2)	100% (7/7)	100% (42/42)

References

- [1] “Querying MySQL for latitude and longitude coordinates that are within a given mile radius,” *Stack Overflow*. [Online]. Available: <https://stackoverflow.com/questions/29553895/querying-mysql-for-latitude-and-longitude-coordinates-that-are-within-a-given-mi>. [Accessed: 06-Apr-2022].
- [2] “Swagger for Spring Rest API,” *Baeldung.com*. [Online]. Available: <https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api>. [Accessed: 06-Apr-2022].
- [3] “Spring Profiles,” *Baeldung.com*. [Online]. Available: <https://www.baeldung.com/spring-profiles>. [Accessed: 06-Apr-2022].
- [4] “Http-proxy-middleware.” [Online]. Available: <https://github.com/chimurai/http-proxy-middleware>. [Accessed: 06-Apr-2022].
- [5] “Environment variables in node.js,” *Engineering Education (EngEd) Program / Section*. [Online]. Available: <https://www.section.io/engineering-education/nodejs-environment-variables/>. [Accessed: 06-Apr-2022].
- [6] “Proxy Routing on Angular App deployed on Heroku,” *Stack Overflow*. [Online]. Available: <https://stackoverflow.com/questions/54300582/proxy-routing-on-angular-app-deployed-on-heroku>. [Accessed: 06-Apr-2022].
- [7] “Continuous delivery of a Spring Boot application with GitLab CI and Kubernetes,” *GitLab*. [Online]. Available: <https://about.gitlab.com/blog/2016/12/14/continuous-delivery-of-a-spring-boot-application-with-gitlab-ci-and-kubernetes/>. [Accessed: 06-Apr-2022].
- [8] “Mockito-Annotations,” *Baeldung.com*. [Online]. Available: <https://www.baeldung.com/mockito-annotations>. [Accessed: 06-Apr-2022].
- [9] S. Matt, “Auto deploy Spring Boot app using Gitlab CI/CD - DZone integration,” *dzone.com*, 14-Sep-2021. [Online]. Available: <https://dzone.com/articles/auto-deploy-spring-boot-app-using-gitlab-cicd>. [Accessed: 06-Apr-2022].
- [10] “h2database: H2 is an embeddable RDBMS written in Java.” [Online]. Available: <https://github.com/h2database/h2database>. [Accessed: 06-Apr-2022].

- [11] “PrimeNG,” *Primefaces.org*. [Online]. Available: <https://www.primefaces.org/primeng/showcase/#/card>. [Accessed: 06-Apr-2022].