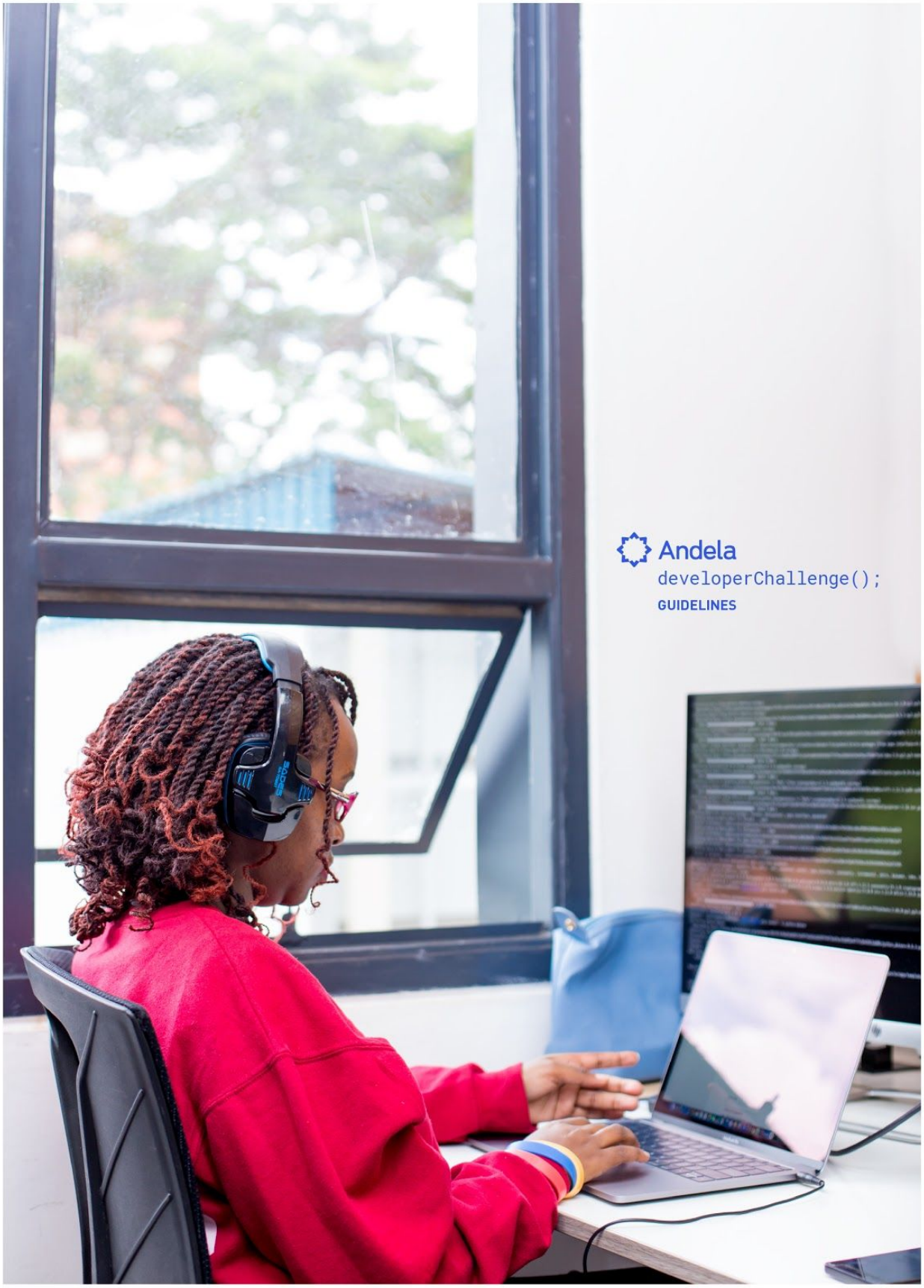
Andela



Andela

DevCTrainingWithAndela
Capstone Project

Build A Product: **Teamwork**

Andela

BUILD A PRODUCT: Teamwork

Project Overview

Teamwork is an **internal social network** for employees of an organization. The goal of this application is to facilitate more interaction between colleagues and promote team bonding.

Project Timelines

- **Total Duration**: 5 weeks
- **Deadline**: 29th November 2019

Required Features

1. Admin can create an employee user account.
2. Admin/Employees can sign in.
3. Employees can post gifs.
4. Employees can write and post articles.
5. Employees can edit their articles.
6. Employees can delete their articles.
7. Employees can delete their gifs post.
8. Employees can comment on other colleagues' article post.
9. Employees can comment on other colleagues' gif post.
10. Employees can view all articles and gifs, showing the most recently posted articles or gifs first.
11. Employees can view a specific article.
12. Employees can view a specific gif post.

Optional Features

- Employees can view all articles that belong to a category (tag).
- Employees can flag a comment, article and/or gif as inappropriate.
- Admin can delete a comment, article and/or gif flagged as inappropriate.

Andela

Preparation Guidelines

These are the steps you ought to take to get ready to start building the project

Steps

1. Create a **GitHub Repository, add a README, and clone it to your computer**.

   ***Tip**: find how to create a Github Repository here.*

2. Create a **GitHub Project Board** on your newly created repo.

   ***Tip**: find how to create a GitHub Project here.*

Andela

Challenge 1: Create API endpoints & Integrate a Database

Challenge Summary

You are expected to create a set of **API endpoints** listed under the **API Endpoints Specification** section and ensure you persist your data using a PostgreSQL database.
You are to write SQL queries that will help you write to and read from your database. The endpoints are to be secured with JSON Web Token (JWT).

Timelines

- **Duration:** 2 weeks

NB:

- *You are to create a separate branch for each feature in this challenge and then merge into your **develop** branch.*
- *Do not use any ORM library for your database activities.*

Tools

- Server-side Framework: **Node**/**Express**
- Linting Library: ***ESLint***
- Style Guide: ***Airbnb***
- Testing Framework: ***Mocha*** *or* ***Jasmine***
- Database: ***PostgreSQL***

Guidelines

1. **Setup ESLint and ensure that your codebase follows the specified style guide requirements.**
2. **Setup the test framework.**
3. **Setup a PostgreSQL database.**
4. **Write unit-tests for all API endpoints.**
5. **Version your API using URL versioning starting with the letter "v". A simple ordinal number would be appropriate and avoid dot notation such as 1.0. An example of this will be https://somewebapp.com/api/v1.**
6. **Implement token-based authentication using JSON Web Token (JWT) and secure all routes requiring authentication, using JSON Web Token (JWT).**

7. **On GitHub Project Management Board, create user stories for setting up and testing API endpoints that do the following using databases:**
    - Admin can create an employee user account.
    - Admin/Employees can sign in.
    - Employees can create and share gifs with other colleagues.
    - Employees can write and/or share articles with colleagues on topics of interest to them.
    - Employees can edit their articles.
    - Employees can delete their articles.
    - Employees can delete their gifs post.
    - Employees can comment on other colleagues' article post.
    - Employees can comment on other colleagues' gif post.
    - Employees can view all articles, showing the most recently posted articles first.
    - Employees can view a specific article.

8. **Use Cloudinary to store your gif files and only save the URL in your application's database. See cloudinary npm package.**
9. **Use API Blueprint, Slate, Apiary, Postman, ReDoc or Swagger to document your API. API documentation should be accessible via your application's URL.**
10. **On GitHub Project Management Board, create stories to capture any other tasks not captured above. The tasks can be feature, bug or chore for this challenge.**
11. **Setup the server-side of the application using the specified web framework.**
12. **Ensure to test all endpoints and see that they work using Postman.**
13. **Integrate TravisCI for Continuous Integration in your repository (with *ReadMe* badge).**
14. **Integrate test coverage reporting (e.g. Coveralls) with a badge in the *ReadMe*.**
15. **Obtain CI badges (e.g. Code Climate) and add to *ReadMe*.**
16. **At the minimum, you should have the API endpoints listed under the API endpoints specification on page 10 working.**
17. **On Github Project Management Board create user story to implement any or all of these optional features:**
    - Employees can view all articles that belong to a category (tag).
    - Employees can flag a comment, article and/or gif as inappropriate.
    - Admin can delete a comment, article and/or gif flagged as inappropriate.

NB: Executing one or more features from the extra credits, in addition to the required features means you have exceeded expectations.

18. **Ensure the app gets hosted on Heroku or any other publicly available cloud application platform.**

API Response Specification

The API endpoints should respond with a JSON object specifying the status of the operation, and either a ***data*** property (on success) or an ***error property*** (on failure). When present, the ***data*** property is always an ***object*** *or an* ***array***.

On Success

```
{
  "status": "success",
  "data": {...} or [{...}, {...}]
}
```

On Error

```
{
  "status": "error",
  "error" : "relevant-error-message"
}
```

Target skills

After completing this challenge, you should have learned and able to demonstrate the following skills.

Skill	Description	Helpful Links
Project management	Using a project management tool (GitHub Project Management Board) to manage your progress while working on tasks.	• To get started with GitHub Projects use GitHub Project Board quick start. • Here is a sample template for creating GitHub Project Management Board user stories.
Version control with GIT	Using GIT to manage and track changes in your project.	• Use the recommended git **branch** and **commit message** naming convention. • Use the recommended Git Workflow, Commit Message standards.

Andela

Cloudinary with Node.js	How to use Cloudinary Node.js SDK	• https://cloudinary.com/documentation/node_integration • https://itnext.io/file-uploading-using-cloudinary-complete-workflow-abfab093fdab
HTTP & Web services	Creating API endpoints that will be consumed using Postman	• Guide to Restful API design • Best Practices for a pragmatic RESTful API
Test-driven development	Writing tests for functions or features.	• Getting started with TDD in Node Node.Js
Data structures	Using data structures in JavaScript.	• JavaScript data types and data structures.
Databases	Using a database to store data in Node.js application.	• Node-postgres • Node.js postgresql tutorial
Continuous Integration	Using tools that automate build and testing when the code is committed to a version control system.	• Setup Continuous Integration with Travis CI in Your Nodejs App
Holistic Thinking and big-picture thinking	An understanding of the project goals and how it affects end-users before starting on the project.	

Self Assessment Guidelines

Use this as general guidelines to assess the quality of your work. Peers, mentors, and facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does Not Meet Expectations	Meets Expectations	Exceed Expectations
Project management	Fails to break down modules into smaller, manageable tasks. Cannot tell the difference between chores, bugs, and features.	Breaks down each module into smaller tasks and classifies them. Constantly updates the tool with progress or lack of it.	Accurately assigns points to the tasks. Informs stakeholders of project progress/blockers in a timely manner.
Version Control with Git	Fails to utilize branching and commits to the master branch directly instead.	Utilizes branching, merges to the develop branch. Use of	Adheres to recommended GIT workflow and uses

		recommended commit messages.	badges.
Programming logic	The code does not work in accordance with the ideas in the problem definition.	The code meets all the requirements listed in the problem definition.	The code handles more cases than specified in the problem definition. The code also incorporates best practices and optimizations.
Test-Driven development	Unable to write tests. The solution did not attempt to use TDD.	Writes tests <= 60% test coverage.	Writes tests with coverage greater than 60%.
HTTP & Web Services	Fails to develop an API that meets the requirements specified.	Successfully develops an API that gives access to all the specified endpoints.	Handles a wide array of HTTP error code and the error messages are specific.
Data Structures	Fails to implement CRUD or Implements CRUD with persistence.	Implements CRUD without persistence.	Uses the most optimal data structure for each operation.
Databases	Unable to create database models for the given project.	Has a database of normalized tables with relationships. Can store, update and retrieve records using SQL.	Creates table relationships.
Token-Based Authentication	Does not use Token-Based authentication.	Makes appropriate use of Token-Based authentication to validate incoming HTTP requests.	Secures all private endpoints with appropriate access rights.
Security	Fails to implement authentication and authorization in given project.	Successfully implements authentication and authorization in the project.	Creates custom and descriptive error messages.
Continuous Integration • Travis CI • Coverall	Fails to integrate all required CI tools.	Successfully integrates all tools with relevant badges added to ReadMe.	

Andela

API Endpoint Specification

Endpoint: POST /auth/create-user
Create user account

Note:

- Admin can create an employee user account.
- Put some of your newly created user data in your token's payload.

Request spec: (body)

```
{
  "firstName": String,
  "lastName": String,
  "email": String,
  "password": String,
  "gender": String,
  "jobRole": String,
  "department": String,
  "address": String,
   ...
}
```

Response spec:

```
{
  "status" : "success",
  "data" : {
   "message": "User account successfully created",
   "token" :  String,
   "userId":  Integer,
    ...
  }
}
```

Endpoint: POST /auth/signin
Login a user

Note:

- Admin/Employees can sign in
- Put some of your user data in your token's payload.

Request spec: (body)

```
{
  "email": String,
  "password": String,
   ...
}
```

Andela

Response spec:

```
{
  “status” : “success”,
  “data” : {
   “token” :  String,
   “userId”:  Integer,
     ...
  }
}
```

Endpoint: POST /gifs

Create a gif.

Request spec: (Header)

```
{
  “token”: String,
    ...
}
```

Request spec: (Body)

```
{
  “image”: image/gif,
  “title”: String,
    ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” : {
   “gifId” :  Integer,
   “message” :  “GIF image successfully posted”,
   “createdOn” :  DateTime,
   “title” :  String,
   “imageUrl” :  String,
     ...
  }
}
```

Andela

Endpoint: POST /articles

Create an article

Request spec: (Header)

```
{
  “token”: String,
   ...
}
```

Request spec: (Body)

```
{
  “title”: String,
  “article”: String,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” : {
   “message” :  “Article successfully posted”,
   “articleId” :  Integer,
   “createdOn” :  DateTime,
   “title” :  String,
    ...
  }
}
```

Endpoint: PATCH /articles/<:articleId>

Edit an article

Request spec: (Header)

```
{
  “token”: String,
   ...
}
```

Request spec: (Body)

```
{
  “title”: String,
  “article”: String,
   ...
}
```

Response spec:

Andela

```
{
  “status” : “success”,
  “data” : {
   “message” :  “Article successfully updated”,
   “title” :  String,
   “article” :  String,
    ...
  }
}
```

Endpoint: DELETE /articles/<:articleId>
Employees can delete their articles

Request spec: (Header)

```
{
  “token”: String,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” :   {
     “message”: “Article successfully deleted”,
      ...
  }
}
```

Endpoint: DELETE /gifs/<:gifId>
Employees can delete their gifs

Request spec: (Header)

```
{
  “token”: String,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” :   {
     “message”: “gif post successfully deleted”,
      ...
  }
}
```

Andela

Endpoint: POST /articles/<articleId>/comment

Employees can comment on other colleagues' article post.

Request spec: (Header)

```
{
  "token": String,
   ...
}
```

Request spec: (body)

```
{
  "comment": String,
   ...
}
```

Response spec:

```
{
  "status" : "success",
  "data" :   {
     "message": "Comment successfully created",
     "createdOn": DateTime,
     "articleTitle": String,
     "article": String,
     "comment": String,
      ...
  }
}
```

Endpoint: POST /gifs/<:gifId>/comment

Employees can comment on other colleagues' gif post.

Request spec: (Header)

```
{
  "token": String,
   ...
}
```

Request spec: (body)

```
{
  "comment": String,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” :   {
     “message”: “comment successfully created”,
     “createdOn”: DateTime,
     “gifTitle”: String,
     “comment”: String,
      ...
  }
}
```

Endpoint: GET /feed

Employees can view all articles or gifs, showing the most recently posted articles or gifs first.

Request spec: (Header)

```
{
  “token”: String,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” :   [
    {
     “id”: Integer,
     “createdOn”: DateTime,
     “title”: String,
     “article/url”: String,  //use url for gif post and article for articles
     “authorId” :  Integer,
      ...
    }, {
     “id”: Integer,
     “createdOn”: DateTime,
     “title”: String,
     “article/url”: String,  //use url for gif post and article for articles
     “authorId” :  Integer,
      ...
    }, {
     “id”: Integer,
     “createdOn”: DateTime,
     “title”: String,
     “article/url”: String,  //use url for gif post and article for articles
     “authorId” :  Integer,
      ...
    },
```

```
  ]
}
```

Endpoint: GET /articles/<:articleId>

Employees can view a specific article.

Request spec: (Header)

```
{
  "token": String,
   ...
}
```

Response spec:

```
{
  "status" : "success",
  "data" :   {
     "id": Integer,
     "createdOn": DateTime,
     "title": String,
     "article": String,
     "comments": [
       {
         "commentId": Integer,
         "comment": String,
         "authorId" :  Integer,
       },
       {
         "commentId": Integer,
         "comment": String,
         "authorId" :  Integer,
       },
     ]
  }
}
```

Andela

Endpoint: GET /gifs/<:gifId>

Employees can view a specific gif post.

Request spec: (Header)

```
{
  “token”: String,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” :   {
     “id”: Integer,
     “createdOn”: DateTime,
     “title”: String,
     “url”: String,
     “comments”: [
       {
         “commentId”: Integer,
         “authorId”: Integer,
         “comment”: String,
       },
       {
         “commentId”: Integer,
         “useauthorIdrId”: Integer,
         “comment”: String,
       }
     ]
  }
}
```

Andela

Challenge 2 - Create Frontend Application

Challenge Summary

This challenge requires that you implement your front-end using **React** and standard language capabilities (e.g., **ES6** for JavaScript).

Timelines

- **Suggested duration:** 3 weeks

NB:

- *Ensure that Challenge 1 is completed and merged to the **develop** branch before you get started with this challenge.*
- *You are expected to use **ReactJS** for your front-end app.*
- *You are to make use of the native browser **Fetch API** for making HTTP requests to the backend built in Challenge 1.*
- **Create a new repo for your frontend application.**
- *You are to create a separate branch for each feature in this challenge and then merge into your **develop** branch.*
- *Do not use an already built website template.*

Tools

- Framework: ***React***
- Linting Library: ***ESLint***
- Style Guide: ***Airbnb***
- State Management: ***Redux***

Guidelines

1. **At a minimum, you should have a working frontend application consuming all your API endpoints.**
2. **On GitHub Project Management Board, create user stories to set up the User Interface (UI) elements:**
    a. A page/pages where an **Admin** can do the following:
        i. Create an employee user account.
    b. A page/pages where an **Admin/Employees** can do the following:
        i. Sign in.
        ii. Create employee user profile.

c. A page/pages where **Employees** can do the following
   i. Employees can post gifs.
   ii. Employees can write and post articles.
   iii. Employees can edit their articles.
   iv. Employees can delete their articles.
   v. Employees can delete their gifs post.
   vi. Employees can comment on other colleagues' article post.
   vii. Employees can comment on other colleagues' gif post.
   viii. Employees can view all articles and gif posts, showing the most recently posted articles/gifs first.
   ix. Employees can view a specific article.
   x. Employees can view a specific gif post.

d. **Deploy your front-end to GitHub Pages while the backend API should be deployed to Heroku.**

e. **On GitHub Project Management Board, create stories to capture any other tasks not captured above. A task can be a feature, bug or chore for this challenge.**

Tip: *It is recommended that you create a* ***gh-pages*** *branch off the branch containing your UI template. When following the GitHub Pages guide, select* ***"Project site" >> "Start from scratch"****. Remember to choose the* ***gh-pages*** *branch as the* ***source*** *when configuring Repository Settings. See a detailed guide* ***here****.*

Target skills

After completing this challenge, you should have learned and be able to demonstrate the following skills.

Skill	Description	Helpful Links
Project management	Using a project management tool (GitHub Project Management Board) to manage your progress while working on tasks.	• To get started with GitHub Projects use GitHub Project Board quick start. • Here is a sample template for creating user stories.
Version control with GIT	Using GIT to manage and track changes in your project.	• Use the recommended git **branch** and **commit message** naming convention.

Andela

		• Use the recommended Git Workflow, Commit Message and standards.
Front-End Development	Using HTML and CSS to create user interfaces.	• See this tutorial • See this tutorial also
UI/UX	Creating good UI interface and user experience	• See rules for good UI design here • See this article for More guide • For color palettes, see this link
Styling react components	Different approaches to styling React components	• Styling and CSS • How to use styles in react • Five ways to style react components

Self / Peer Assessment Guidelines

Use this as general guidelines to assess the quality of your work. Peers, mentors, and facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does Not Meet Expectations	Meets Expectations	Exceed Expectations
Project management	Fails to break down modules into smaller, manageable tasks. Cannot tell the difference between chores, bugs, and features.	Breaks down each module into smaller tasks and classifies them. Constantly updates the tool with progress or lack of it.	Accurately, assigns points to the tasks. Informs stakeholders of project progress/blockers in a timely manner
Version Control with Git	Fails to utilize branching and commits to the master branch directly instead.	Utilizes branching, and merges to the develop branch. Use of recommended commit messages.	Adheres to recommended GIT workflow and uses badges.
Front-End Development	Fails to develop the specified user interface using React.js or uses an already built out template (with an exception for the create-react-app tool),	Successfully develops user interface using React.js while observing standards for building components like unidirectional data flow, separation of concerns, routing, state management with redux, prop-types, and ES6 syntax. Uses semantic HTML5 markup and modular CSS.	Uses testing frameworks e.g. jest, enzyme etc. to write automated tests. Use of CORs for security
UI/UX	The page is unresponsive,	The page is responsive (at	UI is responsive,

	elements are not proportional, the color scheme is not complementary and uses alerts to display user feedback.	least across mobile, tablet and desktops), the color scheme is complimentary and uses properly designed dialog boxes to give the user feedback.	functional and uses basic accessibility features such as alt attribute for images, labels for form fields etc..