# USER GUIDE AND IMPLEMENTATION OVERVIEW FOR SPA WEB APPLICATION

Installation of the system

**Ubuntu Linux**

To run this project in your development machine, follow these steps:

On a Linux system, create a new Python environment

**Python3 -m venv venv**

Then activate the environment

**Source /venv/bin.activate**

Install Python dependencies (main folder):

**$ pip install -r requirements.txt**

On the root folder

Install JavaScript dependencies (from the 'frontend' folder):

**$ npm install**

If everything is alright, you should be able to start the Django development server from the main folder:

**$ python manage.py runserver**

. Open your browser and go to <http://localhost:8000>.

On the login page, you can use this to login in

**Admin username:** james123

**Admin password:** jame123

**Normal Username:** johnmoe

**Password:** United@2023

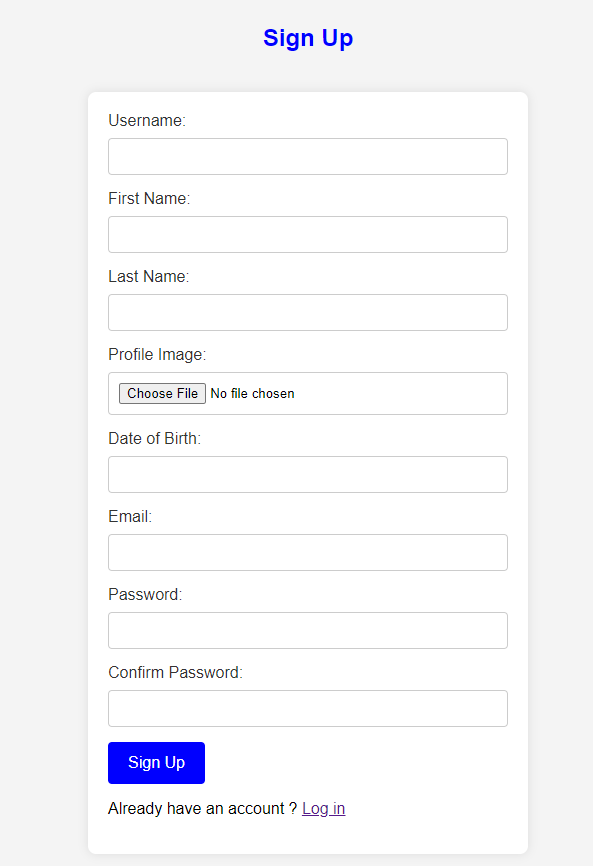
To edit any model, you can use <http://localhost:8000/admin> , with the admin credentials.

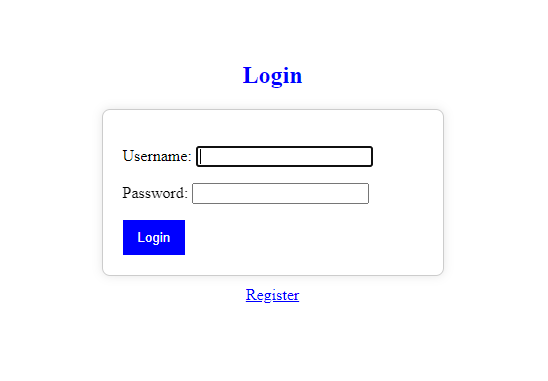
**System Overview**

This document offers a guide with a detailed overview of how to use the SPA- application. In addition, it will state the key functionalities, and aspects of code implementations.

**Account Creation and Authentication:**

To start, users navigate to the home which will redirect them to the login page. Here they can create accounts or log in if they already have an account.





After a successful registration, users can log in to the application using the login page.

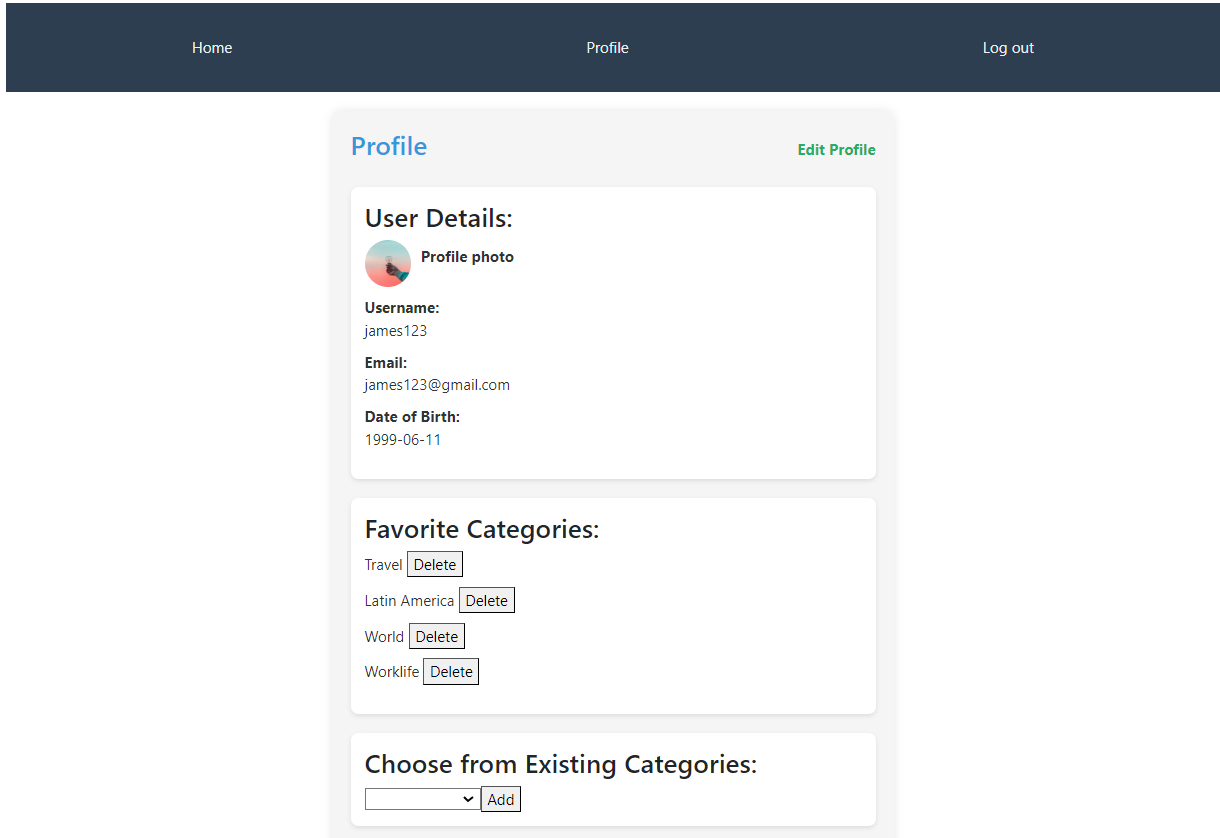
The application's authentication is handled securely through Django's authentication framework, this is server-side rendered.

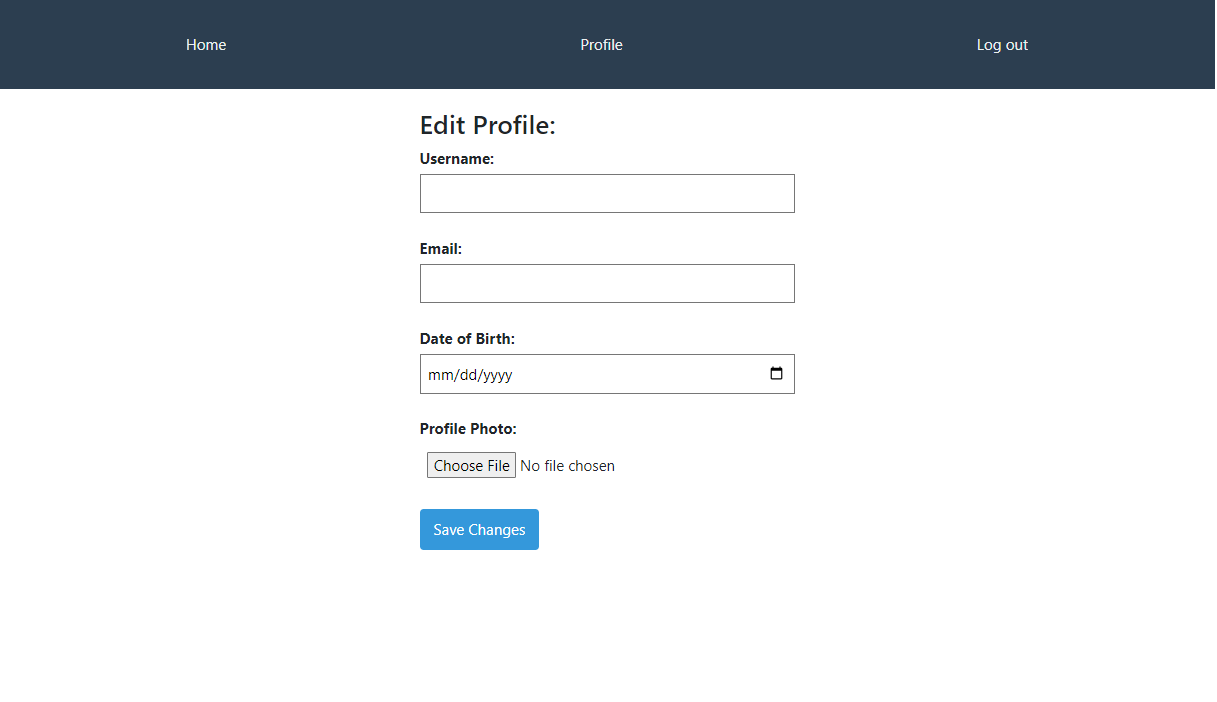
**Profile Management**

After logging in, users are directed to their profile page. On this page, users can view and edit their profile information through the edit profile button at the top.

The details on the profile include a profile picture of the user, email address, date of birth, and a list of favorite news categories according to when they were picked.

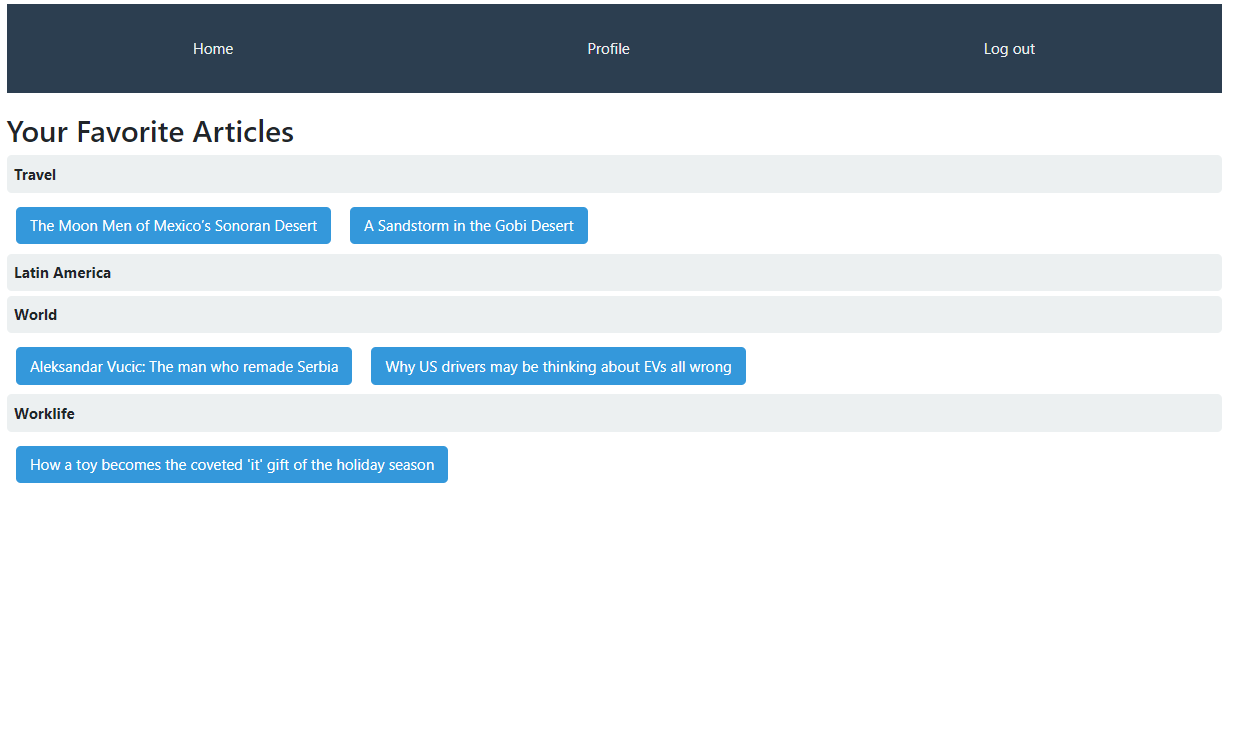
Users can upload or change their profile picture, update their email and date of birth, and manage their preferred news categories.



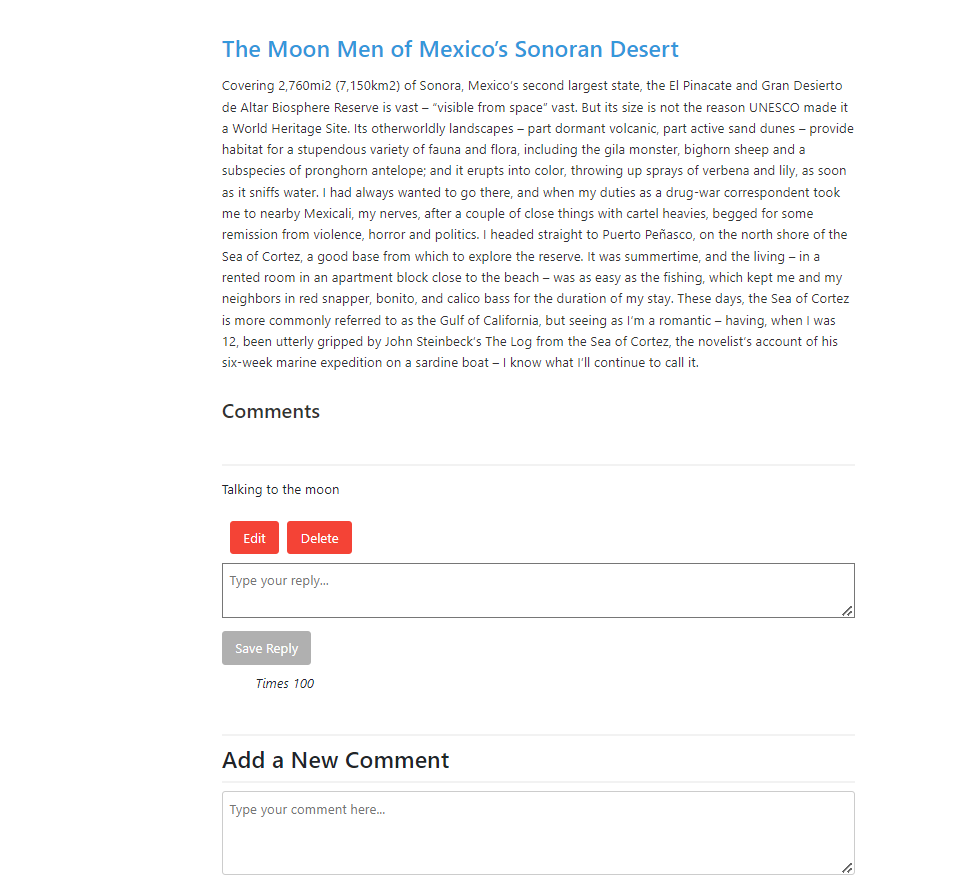


**News Articles**

The application home page is a dedicated page displaying a list of news articles under the user’s favorite category. Users can change articles based on their favorite categories. Clicking on an article provides a detailed view, including the article title, content, and a comment section.



**Specific Article**



**Commenting and Interaction:**

Users can post new comments on articles and reply to existing comments. A user is bale to edit or delete their comment and not of other users. This is implemented using Ajax and fetch API.

**CODE IMPLEMENTATION OVERVIEW**

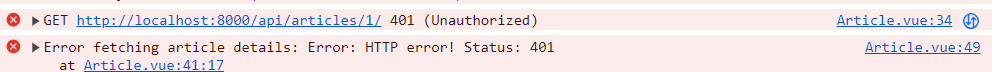
The single-page application utilizes Django and Vue.js. This typically involves Django serving as the backend, handling data storage, authentication, and providing a RESTful API. Vue.js operates on the front end, managing the user interface and making asynchronous requests to Django's API. The front end and back end are decoupled, enabling a clear separation of concerns. Django REST Framework facilitates API development, while Vue Router handles client-side routing in the SPA. The using the Django framework, which has a custom User model inherited from Abstract user. The app's authentication is handed Django's authentication framework

The models on the backend include

User Model, Article, categories, Comments, Replies, and UserFavouriteCategory

There are also API front end interacts with the backend through HTTP requests, allowing seamless communication. User authentication is managed by Django which renders server-side pages of the signup page and login page and then provides tokens used for secure API access. The backend is developed endpoints and views which interact with the front end.

Vue with TypeScript is employed for the front end, creating a reactive and dynamic user interface. Vue router is implemented for frontend routing, ensuring a smooth navigation experience. Pinia, a global store, is utilized for managing states across different components, this includes user details, states, and access tokens to the database. Errors from the client side can be seen from the browser console where you inspect and get them.



The above error need to a user to logout then log in.

Architecture Overview:

The application follows a client-server architecture. The front end is built using Vue.js while the backend is built using Django. Vue handles user interface and sending of API’s while django manages authentication and server-side logic. The database being used is MySQL lite.

Front-End Interaction:

Vue.js handles the client side and user interactions. For example, when a user decides to add a new comment, Vue.js captures the input through a comment form component. It will then send the comment through an API endpoint in Django backend.

API Communication:

Communication between Vue.js and Django occurs through RESTful API endpoints. Vue.js constructs HTTP requests, including necessary authentication details (access token), and sends them to corresponding Django API endpoints. This is commonly achieved using the fetch API. The methods allowed are POST – used to create a new resource, PUT used to edit or update a resource, GET used to fetch a given resource e and DELETE used to remove a given resource from the database,

Back-End Processing:

Django systematically processes incoming requests by routing them to appropriate views, which manage user authentication and data interactions. When authenticating a user Django will serve the login and signup pages using the server side. IT uses its framework's authentication system verifies user credentials, ensuring secure access to user-specific functionalities. After authentication a user will be redirected to vue.js on the client side.

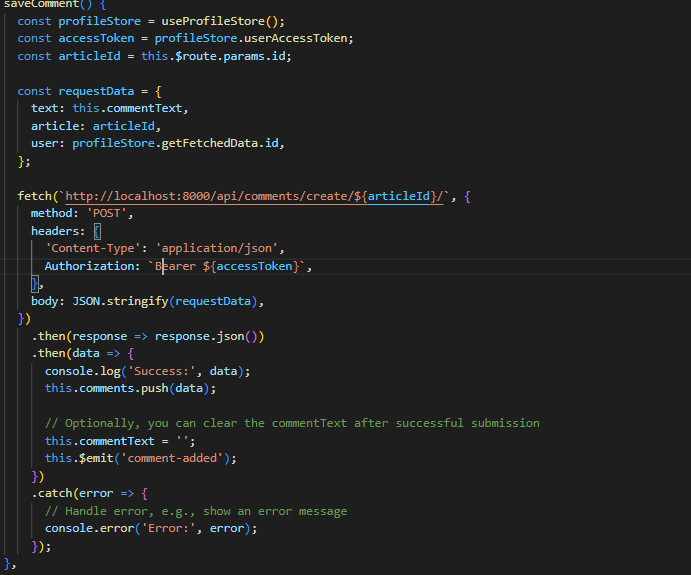
When handling requests involving data creation or updates, Django validates and stores information according to the model's specifications. For data retrieval requests, Django constructs queries, fetches relevant data from the database, and returns it as part of the HTTP response. The format being used is JSON to easily send data between the two apps.

Response Handling:

Following a user-initiated action, such as submitting a fetching comment or updating a comment, the Django server responds with the specific information which will be JSON of a list for fetching all comment s or a response of OK from deleting comments.   
Vue.js having a reactivity model, captures and interprets these responses. Instead of necessitating a full page reload, Vue.js dynamically updates the user interface (UI) to reflect the changes triggered by the server response.

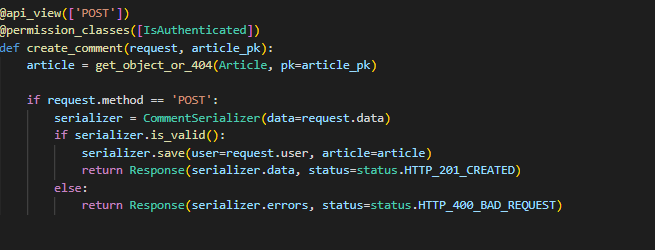
For example, when a user decides to add a new comment, Vue.js captures the comment text from the form component. The front end then constructs a POST request with the comment data and sends it to the Django back-end’s API endpoint.

Front end vue.js code (CommentForm.Vue file)



The code “this.$emit(‘comment-added’);” is used to refresh the comment list.

Django processes the request, validates the user's authentication using the accessToken from the front-end, and stores the new comment in the database. The server responds to vue.js with a success message, triggering Vue.js to update the UI dynamically.

**Django views (api/views.py)**

**Modified Folder and files.**

**API Folder:**

admin.py: Adjusted to register models for easy manipulation.

serializers.py: Created to be utilized by API views.

urls.py: Modified to handle API URLs.

**Project Folder:**

Settings: Modified to include additional settings.

**User Folder or Django app (created to provide user creation, sign up and login)**

Admin.py: created to register user to admin

forms.py: Created to handle forms for Django during sign-up and log-in rendering.

models.py: Adjusted to accommodate given models.

urls.py: Modified to handle signup, login and logout urls.

**Front End:**

**src Folder:**

main.ts: Modified the application during creation to incorporate Pinia and router functionality.

Components Folder: Created to house components imported by pages.

CommentForm.vue: Created to contain code for comment creation.

CommentList: Created to manage a list of comments with editing and deleting methods.

CommentSection: Contains a combination of two sections: comments and reply sections.

Reply Form: Contains code for creating a new reply.

**Pages Folder:**

Article: Contains code for displaying an article and acts as a parent to the comment section.

Home: Entry point of the application.

LoginSuccessful: Stores user details and redirects to the home page.

Logout: Clears all details and redirects to the login page.

Profile: Contains information for the profile page.

Router:

index.ts: Contains a list of Vue routes.

Store:

store.js: Used to globally store user details with Pinia.