High-Level Design Document

Overview

The goal of this proof of concept is to create a simple web application that allows users to input and save basic profile information using the Django framework for the backend, Django Rest Framework (DRF) for the API, React for the frontend, and PostgreSQL as the database. The application will be deployed using Docker.

Architecture

Frontend Architecture (React)

- The frontend will be developed using React with Typescript to create a responsive and dynamic user interface.
- Created interface types for input and return types for components and API payload.
- Utilize React components to manage the different sections of the user interface.
- Implement state management to handle user input and data fetching.
- Implement router for page navigation.
- Implement a popup toaster to display error and success messages.
- Created a circular spinner component to show during making API calls.
- Utilize ESLint to analyze and improve code.

Backend Architecture (Django and DRF)

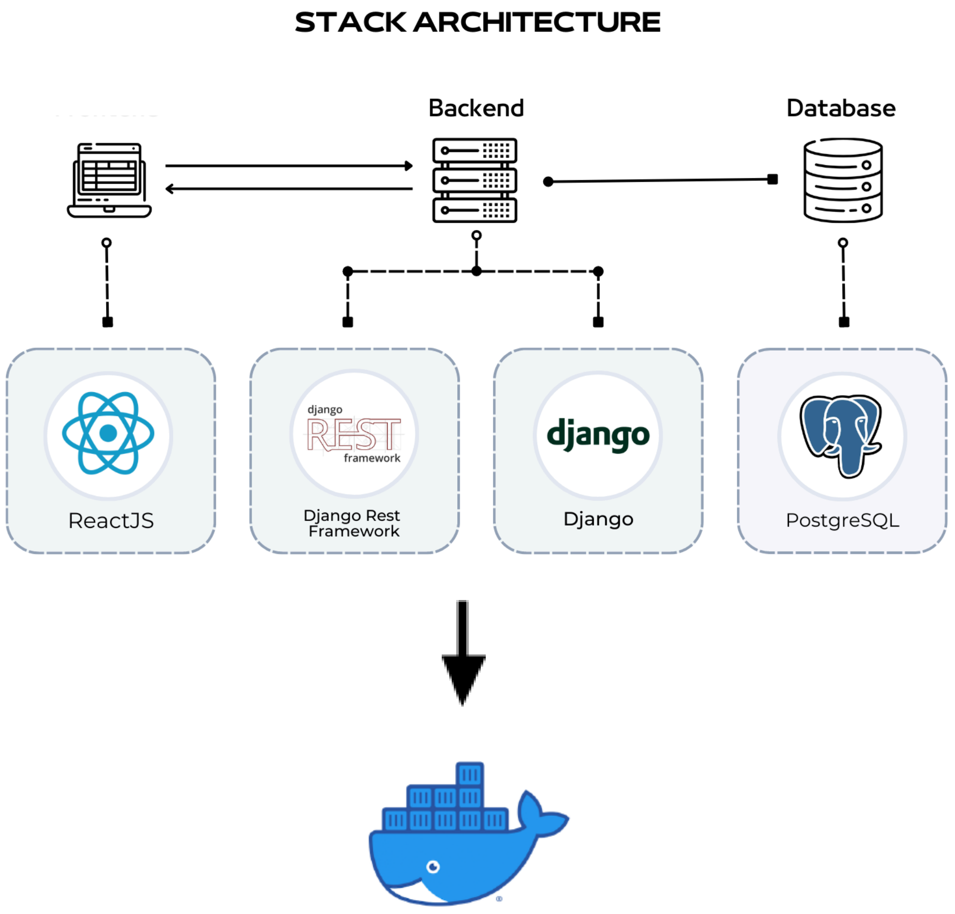
- Django will be used for the backend to manage the application logic and interact with the database.
- Django Rest Framework (DRF) will be employed to create a RESTful API for communication between the frontend and backend.
- Implementation of token based authentication and authorization with custom and built in permission classes.
- Implementation of versioning in the API. Created v1 version containing current features.
- PostgreSQL will be the database of choice to store user profile information.
- Created unit tests using django’s built in TestCase module.
    - Utilized Factory Boy and Faker for data generation.

Docker

- Used Docker for containerization, enabling easy deployment and scalability.
- Created separate Dockerfiles for both frontend and backend.
- Docker Compose will be used to define and manage the application's services.

Database

- Using Postgresql as a database.
- A single table, let's call it User, will store user profile information.
- Fields: id (auto-generated, index=True), first_name, last_name, email, and phone_number.



Backend API Endpoints

Create User:

Endpoint: /api/v1/user/register/
Method: POST
Description: Creates a new user account.
Parameters: first_name, last_name, phone, email, password.

Login User:

Endpoint: /api/v1/user/login/
Method: POST
Description: Retrieves an authentication token of a user.
Parameters: email, password

Get User:

Endpoint: /api/v1/user/
Method: GET
Description: Retrieves the user account details with the specified authentication token in the request headers.
Headers: Authorization: Token <token>

Update User:

Endpoint: /api/v1/user/update/
Method: PATCH
Description: Updates the user profile with the specified authentication token in the request headers.
Parameters: first_name, last_name, phone, email, password.
Headers: Authorization: Token <token>

Frontend Pages

Main Page:

URL: http:0.0.0.0:3000/

A form component to capture user profile information. And creates a user account using backend API. If a user is already registered and logged in then the profile details will be shown in the same form. Users can update it on the same page.

Login Page:

URL: http:0.0.0.0:3000/login
A form component to capture user email and password. It makes an API request to the backend and saves the authentication token to the browser's local storage. After successful authentication it will redirect the user to the update form where all the information is pre-filled.

Data Validation

Email Format Validation:

Ensured that the entered email address follows the correct format. Using format validators on both frontend and backend.

Duplicate Check:

Before saving a new profile, check if the entered email or phone number already exists in the database. For this I created a model cleaning hook in django to check the existence and raise a validation error.

Deployment

- Used Dockerfiles and Docker Compose to manage the deployment of the entire application stack.
- Define services for the frontend, backend, and PostgreSQL.
- Set environment variables for configuration (e.g., database connection details).

Conclusion

This proof of concept outlines the high-level design of a web application that captures and manages user profile information. The integration of Django, DRF, React, PostgreSQL, and Docker provides a scalable and modular architecture. The defined API endpoints, database schema, and validation processes ensure data integrity and user-friendly interactions.