**Introduction:**

The purpose of this project is to build a frontend for a RESTful API that performs CRUD operations on a person object with the attributes of name, age, id, gender, and email. The frontend must display a list of all people in the database, allow users to add a new person, update an existing person, and delete a person from the database.

**The requirements need to run this project:**

1. **The frontend :**

This is a registration form built using HTML, CSS, and JavaScript. The form allows users to select a method from a dropdown menu and enter their personal information based on the selected method. The form has options to get all records, get records by ID, post new records, update records by ID, and delete records by ID.

The form has been styled using CSS and the form is responsive and has been designed to work on smaller screens as well.

The JavaScript code includes functions to handle the user input and send requests to a backend server at <http://localhost:3000/persons>. The functions use the Fetch API to communicate with the server and handle the server's response.

The Dockerfile included in the code is used to build a Docker image for the NGINX web server. The NGINX server is used to serve the HTML, CSS, and JavaScript files to users.The container is set up to listen on port 80 to allow incoming traffic.

1. **The backend:**

This is a backend application built using Flask, a Python web framework. The application provides a RESTful API for managing a list of people. The application allows clients to perform the following operations:

* GET /persons: Retrieve a list of all person objects
* POST /persons: Create a new person object
* GET /persons/{id}: Retrieve a specific person object by its ID
* PUT /persons/{id}: Update a specific person object by its ID
* DELETE /persons/{id}: Delete a specific person object by its ID

The application uses Flask-CORS to enable Cross-Origin Resource Sharing (CORS) for the API. CORS is a security feature that restricts web pages from making requests to a different domain than the one that served the web page. The application also uses Flask\_SQLAlchemy to interact with a database.

The application is containerized using Docker and can be deployed on any platform that supports Docker. The Docker file includes the necessary instructions to install the required dependencies, set up the environment, and run the application.

To run the application, execute the command python app.py or flask run in the terminal. The application will start running on the local host at port 5000. To access the various endpoints, use a REST client such as Postman.

The application is designed to be scalable and can handle multiple requests concurrently. It uses Gunicorn, a Python WSGI HTTP Server for UNIX, to serve the application. The -w option specifies the number of worker processes, and the -b option specifies the socket to bind.

1. **Deckor compose.yml:**

Docker Compose file that defines two services, backend and frontend, and their respective configurations.

The backend service is built from the. /Backend directory and creates an image named backimage. The container name is set to the back container. The service exposes port 80 and maps it to port 3000 on the host machine. The environment variable NAME is set to World, and the. /backend directory is mounted as a volume in the container. The restart policy is set to unless-stopped, which means that the container will automatically restart unless it is stopped manually.

The front-end service is built from the./front-end directory and creates an image named front image. The container name is set to front container. The./frontend directory is mounted as a volume in the container, and the service exposes port 80, which is mapped to port 8000 on the host machine. The restart policy is set to unless-stopped, and the service depends on the backend service, which means that the backend service must be started before the frontend service.