## Project Summary

### Objective:

This project aims to develop a fully functional To-Do List web application using the Vue.js frontend framework, Django backend framework, and PostgreSQL database. The application is designed to enable users to create, manage, and share to-do lists.

### Features:

- User registration and login
- To-do list creation, editing, and deletion
- Individual to-do item addition, completion, and deletion
- To-do list sharing and collaboration
- User profile management
- Responsive design
- RESTful API

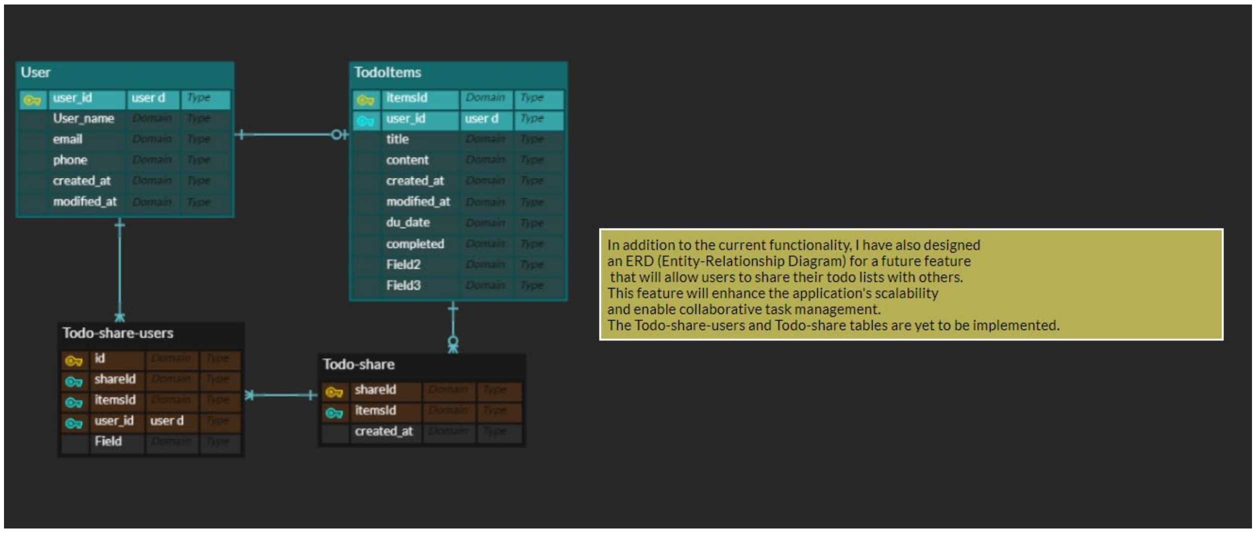
### Technologies Used:

- **Frontend:**
  - Vue.js 3 framework
  - ViteJS build tool
  - Vuex store management
  - Axios library (API communication)
  - Bootstrap CSS framework
- **Backend:**
  - Django 4.1.7 framework
  - Python 3.11 programming language
  - PostgreSQL database
- **Authentication:**
  - JWT (JSON Web Token)
- **Deployment:**
  - Docker
  - Nginx
  - Gunicorn

## Development Process

### 1. Planning and Initial Setup

- Analyze project requirements and define functionalities
- Select technology stack and development tools (ViteJS, Vue.js, Vuex, Axios, Bootstrap, Django, PostgreSQL, Docker, Nginx, Gunicorn)
- Set up Git version control system (GitHub, etc.)

- Configure project structure and directories
- ERD
- 

In addition to the current functionality, I have also designed
an ERD (Entity-Relationship Diagram) for a future feature
that will allow users to share their todo lists with others.
This feature will enhance the application's scalability
and enable collaborative task management.
The Todo-share-users and Todo-share tables are yet to be implemented.

## 2. Frontend Development

- Implement user interface using Vue.js 3 framework
  - Build reusable UI components using Vue components
  - Manage application routing and page navigation with Vue Router
- Utilize ViteJS build tool for efficient development and build environment
  - Leverage fast hot reloading capabilities
  - Generate optimized builds
- Employ Vuex store management system to handle application state
  - Store user information, to-do lists, UI settings, etc.
- Use Axios library to communicate with Django backend API
  - Send API requests, process responses, and handle errors
- Implement responsive design using Bootstrap CSS framework
  - Adapt UI for various devices

## 3. Backend Development

- Implement RESTful API using Django 3.2 framework
  - Develop user management, to-do list management, and API endpoints
- Utilize PostgreSQL database for data storage and management
  - Store user information, to-do lists, and other data
- Define database schema using Django models
  - Use model classes to define data structure and relationships
- Build API endpoints using Django REST framework
  - Implement API functionalities using serializers, views, URL patterns, etc.
- Implement user authentication and authorization with JWT system
  - Handle user login, token issuance, and authenticated request processing

## 4. Frontend-Backend Integration

- Invoke Django backend API from Vue.js application
    - Send API requests using Axios library
- Update frontend UI using API response data
    - Update Vuex store and render UI accordingly
- Handle errors and exceptional situations
    - Process API errors, network errors, and notify users

### 5. Authentication Implementation

- Implement user registration and login functionalities
    - Utilize Django REST framework's authentication backend
    - Store user information, issue tokens, and manage sessions
- Implement JWT token-based authentication

### 6. Additional Feature Implementation (Optional)

- Implement user profile management functionality: Enable users to modify profile, names, emails, etc.
- Implement to-do list sharing and collaboration functionality: Allow users to share and collaborate on to-do lists with others.
- Implement notification functionality: Notify users about new to-do items, completed tasks, upcoming deadlines, etc.

### 7. Deployment

- Configure Nginx web server
    - Modify Nginx configuration file to serve frontend application static files
    - Set up proxy to Gunicorn WSGI server
- Configure Gunicorn WSGI server
    - Modify Gunicorn configuration file to run Django backend API
    - Receive and process requests from Nginx proxy server
- Dockerize application
    - Isolate Vue.js frontend application, Django backend API, Nginx web server, and Gunicorn WSGI server into separate Docker containers
    - Use Docker Compose to automate container connections and execution
- Deploy to production environment
    - Deploy Docker container images to the production server
    - Run containers and start the application using Docker Compose

Challenges Faced:
Outline any difficulties or challenges you encountered during the development process and how you addressed them.

Code Structure and Quality:
Evaluate the structure and quality of your code. Discuss whether it is well-organized, readable, and follows best practices. Provide examples if necessary.

User Experience:
Assess the user experience of your application. Discuss its responsiveness, intuitiveness, and any feedback mechanisms implemented.

Deployment and Instructions:
Detail the deployment process of your application, including the chosen platform and any deployment scripts used. Provide clear instructions on how to run the application locally.

## Running the To-Do List Web Application Locally

### Prerequisites:

- Git installed and configured
- Docker installed and running
- GitHub account (optional)

### Steps:

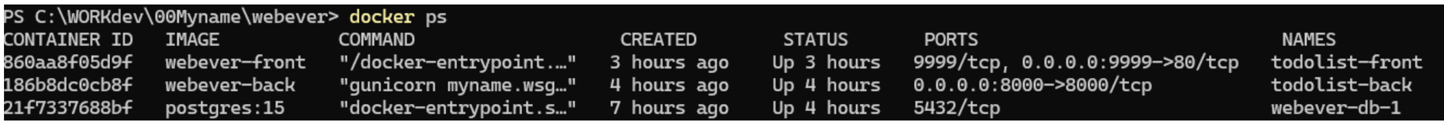
1. **Clone the Git Repository:**
   `git clone https://github.com/kevin-kidong-lim/todolist.git`

2. **Navigate to the Project Directory:**
   `cd todolist`

3. **Pull Docker Images:**
   ```
   docker pull ultrax00/todolist-postgres:latest
   docker pull ultrax00/todolist-front:latest
   docker pull ultrax00/todolist-back:latest
   ```

4. **Start Containers:**
   `docker-compose -f docker-compose.dev.yml up -d`

   `docker ps -a`



```
PS C:\WORKdev\00Myname\webever> docker ps
CONTAINER ID   IMAGE           COMMAND                   CREATED       STATUS       PORTS                            NAMES
860aa8f05d9f   webever-front   "/docker-entrypoint.…"    3 hours ago   Up 3 hours   9999/tcp, 0.0.0.0:9999->80/tcp   todolist-front
186b8dc0cb8f   webever-back    "gunicorn myname.wsg…"    4 hours ago   Up 4 hours   0.0.0.0:8000->8000/tcp           todolist-back
21f7337688bf   postgres:15     "docker-entrypoint.s…"    7 hours ago   Up 4 hours   5432/tcp                         webever-db-1
```

5. **Verify Application:**

   Open a web browser and navigate to `http://localhost:9999/todologin` to access the To-Do List web application.

Additional Comments:
Feel free to add any additional comments, insights, or improvements you would make if given more time.

**While I successfully implemented the core functionality of the To-Do List application within the allotted time, I recognize that additional features and enhancements could have been incorporated. With more time, I would have focused on implementing user-to-user task sharing, time estimation and tracking capabilities, and further code refactoring to improve maintainability. Additionally, I regret not allocating more time to developing comprehensive test cases to ensure the application's quality and reliability.**

Submission Guidelines:
Please submit your report as a PDF document along with any necessary files (e.g., code snippets, screenshots). Ensure your report is well-organized and clearly written.

Deadline:
Please submit your report by Thursday April 11. If you have any questions or need clarification, don't hesitate to reach out via email.