1. **Client-Side Structure:**
    - **node_modules**: This folder contains all the node package dependencies.
    - **public**: it includes an **upload** folder(where the uploaded screenshots are saved) and **index.html**.
    - **src**: The source folder for all the front-end code.
        - **Assets**: contains static files like images, styles, or scripts that are used by the client (react) application.
        - **components**: Holds React components, which are the reusable parts of the UI. The specific components here (**navbar(bar that appears at the top)**
          , **question(a specific question)**, **questions(all questions on the page)**, **replies(replies of accounts on the question)**, **submitQuestion(logic to submit a question and save in database)**) hence a Q&A type of application.
        - **context**: Contains context files (**authContext.js**, **darkModeContext.js**) for managing state and themes across the application.
        - **pages**: Contains React components or scripts for individual pages (**home**, **profile**, **register**, **signIn**), indicating the different views/routes in the application.
        - **App.js**: The main React component that wraps the entire application.
        - **axios.js**: A utility or service file to handle HTTP requests, indicating usage of the Axios library.(a custom axios.js toi define makeRequest function inside to be used by react-query)
        - **index.js**: The entry point for the React application.
2. **Server-Side Structure:**
    - **backend**: The top-level folder for backend code.
        - **controllers**: Contains logic for handling requests (**authent.js**, **like.js**, **question.js**, **user.js**). These files are lresponsible for CRUD operations and logic related to authentication, liking functionality, question handling, and user management.
        - **node_modules**: Contains backend dependencies.
        - **routes**: Defines the API endpoints (**authent.js**, **likes.js**, **questions.js**, **replies.js**, **users.js**) that the front end will communicate with. There's also a **db_connect.js** which manages database connections.
        - **index.js**: entry point for the server application.
        - **package-lock.json** and **package.json**: Node/npm configuration and dependency files.

Design Report:

- The project follows a standard Node.js/React application structure with a clear separation between the client (**client**) and the server (**backend**).
- The **components** folder within the client suggests that the UI is built with React and is likely component-driven, which can help with maintainability and reusability.
- The use of **context** in the client-side implies a state management pattern that avoids prop drilling, which is good for the scalability of state management.
- The backend is structured around RESTful principles, given the **controllers** and **routes** setup, which is a common and effective architecture for web APIs.

- The naming convention is clear and indicative of functionality.

Npm libraries used

```
"bcryptjs": "^2.4.3",  “to hashpasswords”

"cookie-parser": "^1.4.6",” for secure transfer of json data”
"cors": "^2.8.5",”for secure transfer of json data”
"jsonwebtoken": "^9.0.2",",” for secure transfer of json data”
"moment": "^2.29.4",”to get time”
```

```
"@emotion/react": "^11.10.4",
    "@emotion/styled": "^11.10.4",
    "@mui/icons-material": "^5.10.9",
    "@mui/material": "^5.10.10",
    "@testing-library/jest-dom": "^5.16.4",
    "@testing-library/react": "^13.1.1",
    "@testing-library/user-event": "^13.5.0",
“above libraries for UI”

    "axios": "^1.1.3",”to make api calls”
    "moment": "^2.29.4",
    "react": "^18.0.0",
    "react-dom": "^18.0.0",
    "react-query": "^3.39.3",”to manage query handling get data from backend and send”
    "react-router-dom": "^6.4.2",
```