### Objectives

The external design document evaluates how close you have come to a marketable product and honestly examines what is to be done. It is a tool for explaining your design to other software engineers, what works and what does not, and what still needs attention. Examine what you would do differently if you had all the time in the world. Examine what does not work and why. Consider how you would change the design – knowing what you learned from this process. Can you come up with some helpful suggestions for other Software Engineers attempting to solve similar problems?

The objectives also include:

a) Determine that the software meets the goals established by the specification. Therefore, discuss in the design document which features are supported, partially supported, and not supported. Discuss what still needs to be done to meet the original requirements.

b) Guide programmers responsible for maintaining and upgrading software. Therefore, discuss what outstanding bugs there are in the software. Make suggestions on how to improve the software in the future.

c) Assist with quality assurance and the design of test cases. Discuss what has/still needs to be tested.

d) Analyze what you would do differently if you had all the time in the world to change the design and implementation.

d) Analyze the security of the data. **Important**

### Outline

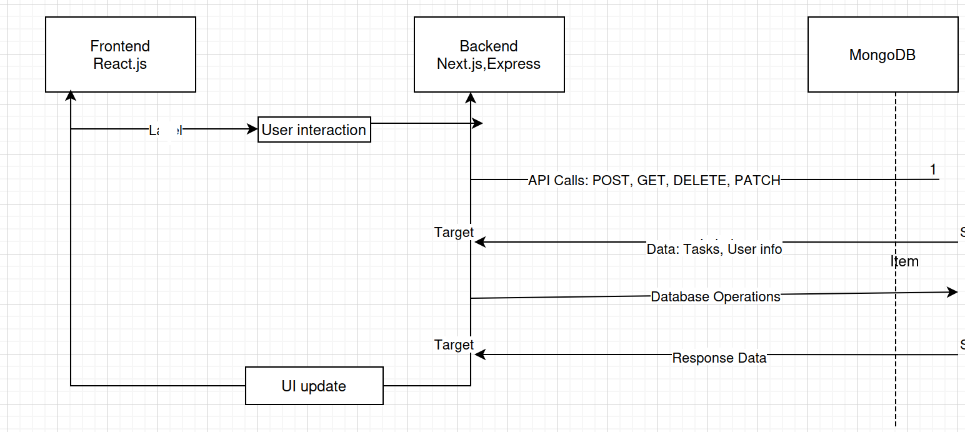
The external design documentation should include the following concepts. Items may be left out if they are not necessary and if they are redundant:

#### • Purpose and Scope of the Software

* **Introduction**
* The project name of the software application is LetsDo Task Manager.
* It is a task-management software application that aims to help users organize and manage various tasks by adding, editing, deleting, and viewing them. The application helps users categorize tasks and prioritize them based on their needs. It contains a crucial feature: user authentication for profile management and aiding task tracking.
* **Software Description (functional and operational overview)**
* **Functional Requirements**
* The application contains various functions, such as:
* a) User authentication—This Function enables users to create an account, sign in to the account, and view their task list.
* b) Task Management—The application allows users to create, edit, delete, and view their task lists.
* c) Categorization and Prioritization – The software application has a unique feature that enables users to prioritize their tasks as High, Medium, or Low.
* d) User Profile – The system enables users to manage their profile thoroughly and see various associated tasks.

#### • Architecture

* **Central component architecture (and subcomponents...)**
* The major components in the LetsDo software system are the Frontend and Backend.
  + **Frontend**: The frontend is designed using React.js, which serves various functions, such as handling the user interface and user interaction and allowing communication to the backend using RESTful APIs.
  + It contains the following components:
  + **Task List** – Displays various tasks to the user. Userd allows actions like delete and edit.
  + **Task Form** – A form that allows users to create and edit tasks in the system.
  + **User Profile** – Displays user data and allows editing the profile.
  + **Authentication** – Allows the users to create an account and sign in with validated details.
  + **Backend**: The backend is designed using Node.js, Express.js, and MongoDB.
  + The backend contains the following components:
  + **User Management**: It facilitates the authorization and validation of user details during signup, login, and user profile management.
  + **Task Management** – The backend facilitates CRUD operations (Create, Read, Update, and Delete).
  + **Database** – The software uses MongoDB to store user data and tasks.
  + **General structure chart showing the data flow**



The Front End (React) is the client interacting with users. It communicates with the Back End (Node.js/Express) through HTTP API calls (POST, GET, PATCH, DELETE).

• The Backend processes the requests, interacts with the Database (MongoDB) to store or fetch data, and sends responses back to the frontend.

• The Database (MongoDB) stores all user data (user profile) and task-related data (task details).

* **General control flow**
* User Request (Frontend): The user interacts with the UI (e.g., adding a task).
* • API Request (Backend): React app sends a request to the backend.
* • Processing (Backend): The backend processes the request, interacts with the database, and returns a response.
* • Response to Frontend: React app updates the UI with the data.
* **Relationships and interdependencies of components**
  + **Frontend and Backend**: The frontend depends on the backend API to fetch and manipulate data, and the backend depends on the MongoDB database for storing and retrieving data.
  + **MongoDB and Backend**: The backend interacts directly with MongoDB to perform CRUD operations on tasks and user data.

#### • Interface Requirements and Environment

* **User Interface**
* • **Login Page**: Users can log in using their email and password.
* • **Task Management Page**: Displays a list of tasks with options to add, edit, and delete tasks.
* • **User Profile Page**: Displays user information and lets users update their details.

#### • Common Internal Specification

* **Files (type of files, number of files, name of files, etc.)**
* **Frontend Files**: React components and JavaScript files for task and user management.
* Example: src/components/TaskList.js, src/components/UserProfile.js
* **Backend Files**: Node.js modules for task and user management, routes, controllers, and models.
* Example: backend/controllers/tasks-controllers.js, backend/routes/users-routes.js
* **Databases (format of data)**
  + **users**: Stores user data (name, email, password).
  + **Tasks**: Stores (title, category, priority, user association).
* **Modules that need to be imported**
* **Frontend**: react, Axios, react-router-dom
* **Backend**: express, mongoose, cors, body-parser, jsonwebtoken.
* **Error formats common to all procedures**
* General Error Format: JSON format with status codes and message.
* Example: { "status": 500, "message": "Internal Server Error" }

#### • Component Internal Design

* **Module/Function The Name**
* function creates a user.
* **VersiUsernd date**
* v1.0, December 2024
* **Description**
* This Function handles the user registration process. After hashing the password, it receives user data, validates the inputs, and saves the user to theUsergoDB database.
* **Inputs/Outputs ... arguments**
* **Inputs**:
* • name (String) – User's full name
* • email (String) – User's email address
* • hashed\_password (String) – Hashed password of the user
* • composed (Number) – User's task completion status (optional)
* • picture (String) – User's profile picture URL (optional)
* • created (Date) – Account creation timestamp (optional)

**Outputs**:

• Success: JSON object with newly created user's detaUser's.g., name, email, completed, etc.)

• Failure: Error message (e.g., User alreadUserists, Failed to create user)

* **Error**
* UserEmail already exists if the email is already in the database.
* • Database connection error if the database operation fails.
* • Invalid input data if the data does not meet validation criteria (e.g., missing fields).
* **Globals that are used**
* • **User (Model)**: Used to interact with the MongoDB users collection.
* • **following (Function)**: This is for handling and passing errors to the next middleware in Express.
* **Globals that are altered**
* • None specifically altered in this Function.
* **Functions that are used**
* • **User.save()**: Saves the new user to the user base.
* • **next()**: Used for passing errors to the global error handler in Express.

#### • Documentation Requirements

* **User Manual**
* • **Sign Up and Login**: Instructions on how to sign up, log in, and manage the profile.
* • **Task Management**: How to add, edit, and delete tasks.
* **Installation Manual**
* **Installation Manual**
* • **Setup Guide**: Steps to clone the repository, install dependencies, set up MongoDB, and run the application.
* ◦ Example:
* ▪ git clone <repo\_url>
* ▪ npm install
* ▪ npm start

#### • Performance Evaluation

**Functionality**

• Currently, all basic functionalities (task management, user authentication) are working.

**Compatibility**

• Works on modern browsers (Chrome et al.).

**Extensibility**

• Can be extended to include features like task prioritization, notifications, or collaboration.

**Portability**

• The portable system can be deployed on various platforms (e.g., Heroku, AWS).

**Reliability**

• The app relies on the stability of MongoDB and Express.

**Maintainability**

• Well-documented code with separate concerns, ensuring developers can easily maintain and update the system.

**Security**

• User passwords are hashed and not stored in plain text.