SOFTWARE DESIGN ASSIGNMENT WEEK 2/05

TASKER task manager Application

SCCE/04598P/2021

**Software Engineering Methodology.**

The software engineering methodology I chose for my project is Kanban Agile methodology.

Kanban focuses on visualizing and optimizing the flow of work. It works on the principles of lean manufacturing, which emphasizes the elimination of waste, and continuous improvement.

1. Visualize the Workflow: To start with, I will create a visual representation of the workflow, which is the process of creating and managing tasks in the task manager software. I will use a physical board or an online tool to represent the stages of the workflow, such as "To Do," "In Progress," "In Review," and "Done." The board should allow me to see the status of all tasks in real-time and manage the workload.

2. Limit Work in Progress (WIP): It is important to limit the number of tasks that are in progress at any given time. This can help prevent overloading myself and ensure that work is completed in a time. I will set a WIP limit for each stage of the workflow.

3. Manage Flow: I will continuously monitor the flow of work through the process and identify bottlenecks and areas for improvement. I will track the time it takes for a task to move through the different stages of the workflow and identify tasks that are stuck or taking longer than expected.

4. Make Process Policies Explicit: I have defined clear rules and policies for each stage of the workflow. The main policy is that all tasks must have a clear description and acceptance criteria before they can be moved to the "In Progress" stage.

5. Implement Feedback Loops: Regularly review the process and note down the feedback so as to identify the areas of improvement

**context models that will serve my software building process:**

**1.project goals**

**2. structure**

**3.User Requirements**

1. Project goals: The goal of the project is to come up with an efficient task management system that users interact with on a daily basis and the software feeds

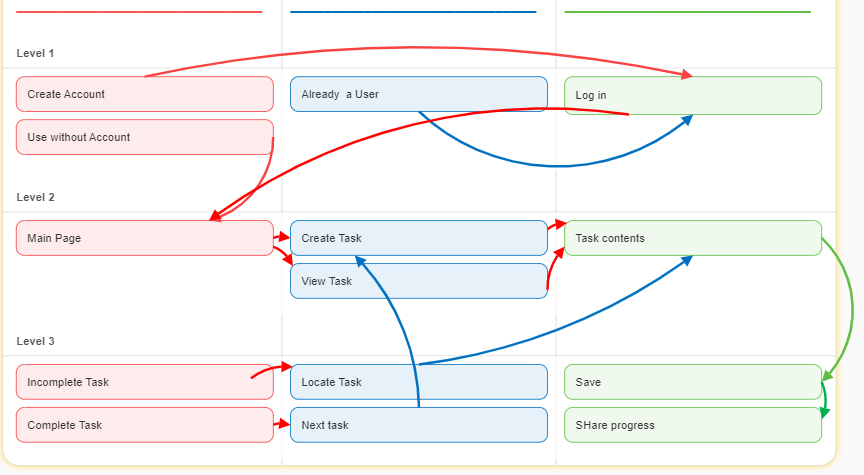
back info to the users in form of notifications upon completion of task or skiving of the task according to the allocated time.

I need to determine the scope of each iteration or sprint and ensure that I am working towards the project's overall objectives.

2.Structure: The type of structural approach am giving the software will influence how long the project building process will take and the tools I’ll need to complete the developing process.

3. User requirements: Agile methodologies prioritize customer satisfaction by involving users in the development process, gathering feedback and adapting to changing requirements. Understanding user needs, preferences, and expectations is needed.

The diagram below outlines all the basic processes of the software that users will interact with to complete service delivery process of the software.



Interactions Model that will serve my software :

1. User Interaction:

- The user interacts with the application through the user interface (UI) components.

- UI components capture user input (such as forms, buttons, etc.) and trigger actions or state changes.

2. State Management:

- The application state is managed using a state management library such as Redux or React Context API.

- UI components subscribe to the application state and update accordingly.

- Actions dispatched by the UI components trigger state updates.

3. Component Interaction:

- UI components interact with each other through props and callbacks.

- Parent components pass data and behavior down to child components through props.

- Child components communicate with parent components using callbacks.

4. API Interaction:

- Components may interact with backend APIs to fetch or submit data.

- API requests are typically made using libraries like Axios or the built-in Fetch API.

- Responses from the API are processed and used to update the application state.

5. Routing and Navigation:

- Navigation between different pages or views is managed using a routing library like React Router.

- Components render based on the current route, and route parameters can be accessed for dynamic content.

6. External Libraries and Services:

- React components may interact with external libraries or services for specific functionalities (e.g., charting libraries, authentication services, etc.).

- Integration with external libraries is typically done through props and callbacks.

7. Testing and Debugging:

- Unit tests, integration tests, and end-to-end tests can be implemented to ensure the correctness of the application.

- Debugging tools like React DevTools or browser developer tools can be used to inspect and debug components.

The structural models that define the organization and relationships between different components and modules are:

1. Component Hierarchy Model:

- Task Manager: The main component that represents the entire application.

- Header: A component for the application header, which typically includes the logo and navigation.

- TaskList: A component that renders the list of tasks.

- TaskItem: A component representing an individual task within the TaskList.

- TaskForm: A component for creating or editing tasks.

- Footer: A component for the application footer.

The component hierarchy model illustrates the parent-child relationships between components, with the Task Manager component at the top and the more granular components nested within.

2. Container-Component Model:

- Containers: These are components responsible for managing data and state, typically connected to a state management library like Redux. Examples of containers in a task manager software could include TaskListContainer and TaskFormContainer.

- Presentational Components: These components focus on rendering UI elements based on the data and state provided by containers. Examples could include TaskList, TaskItem, and TaskForm.

The container-component model helps separate the concerns of data management and UI rendering, making the application more maintainable and testable.

3. Data Model:

- Task: A data model representing a task, typically consisting of properties such as an ID, title, description, due date, status, etc.

- TaskList: A collection of tasks, which can be stored as an array or an object with task IDs as keys.

The data model defines the structure and properties of the task objects and how they are stored and manipulated within the application.

4. Routing Model:

- Routes: Define the different views or pages of the task manager software, such as a task list view, task details view, and task creation/editing view.

- Route Parameters: Used to pass dynamic data through the URL, such as the task ID for the task details view or the task ID for editing an existing task.

The routing model allows the user to navigate between different views and provides the ability to deep-link to specific tasks or views within the application.

It's important to consider behavioral models that describe the functionality and interactions of the system.

1. Use Case Model:

- Identify the different use cases or actions that a user can perform within the task manager software, such as creating a task, editing a task, marking a task as complete, etc.

- Define the actors involved, such as the user and any other external systems or services.

- Specify the flow of events for each use case, including preconditions, main steps, and postconditions.

The use case model helps in understanding the functional requirements of the software and the interactions between the user and the system.

2. User Interaction Model:

- Identify the user interface components involved in each use case.

- Define the user interactions with the components, such as clicking buttons, entering data into forms, selecting options, etc.

- Describe the expected behavior of the components in response to user interactions.

The user interaction model focuses on capturing how the user interacts with the user interface components to perform various tasks.

3. State Transition Model:

- Identify the different states of the task manager software and the events that trigger transitions between these states.

- Define the actions and side effects associated with each state transition, such as updating the task list, persisting data, triggering notifications, etc.

- Specify any constraints or conditions that determine which transitions are allowed.

The state transition model describes the behavior of the software in terms of state changes and the corresponding actions.

4. Data Flow Model:

- Identify the flow of data within the task manager software, from the user interface components to the data storage or backend systems.

- Specify the data transformations and manipulations that occur at different stages, such as data validation, filtering, sorting, etc.

- Define the communication protocols and APIs used for data exchange between the frontend and backend systems, if applicable.

The data flow model helps in understanding how data moves through the system and is processed or transformed along the way.