Coral Task Submission

Kritika Gautam-IIIT Hyderabad

Kanban Dashboard Documentation

Overview

This **Kanban Dashboard** is a task management application that organizes tasks into columns representing different statuses: **To Do**, **In Progress**, **Blocked**, and **Done**. Each task (ticket) follows a predefined transition path enforced by a Finite State Automaton (FSA). The FSA logic ensures that each ticket can only move between specific states, providing a structured workflow.

Features

- **State-Driven Ticket Transitions**: Only specific status transitions are allowed, e.g.:
    - `To Do → In Progress`
    - `In Progress → Blocked or Done`
    - `Blocked → In Progress`

- **Redux for FSA Enforcement**: The Redux state management library is used to handle ticket transitions and enforce valid state transitions.
- **Live Mode Toggle**: Users can toggle live mode to observe real-time updates on ticket movements.
- **Component-Based Architecture**: The project is organized with modular and reusable React components, simplifying maintenance and extension.

Project Structure

The project is organized as follows:

```
php
Copy code
Kanban-Dashboard/
├── public/                     # Public assets
├── src/
│   ├── components/             # React components
│   │   ├── KanbanBoard.js      # Main Kanban board componen
t
│   │   ├── KanbanColumn.js     # Column component for each 
status
│   │   └── TicketCard.js       # Card component for each ti
cket
│   │
│   ├── redux/                  # Redux files
│   │   ├── ticketActions.js    # Action creators for ticket 
transitions
│   │   ├── ticketReducer.js    # Reducer for ticket state m
anagement
│   │   └── store.js            # Redux store configuration
│   │
│   ├── tickets.json            # Sample data for tickets
│   ├── App.js                  # Main app component
│   ├── index.js                # Entry point with Redux Pro
vider
```

```
│   └── styles/                  # CSS files for styling comp
onents
│
├── README.md                    # Project documentation
└── package.json                 # Project dependencies and s
cripts
```

Setup and Installation

Prerequisites

Ensure you have **Node.js** and **npm** installed. You can verify installation by running:

```
bash
Copy code
node -v
npm -v
```

Installation Steps

1. **Clone the repository**:

   ```
   bash
   Copy code
   git clone https://github.com/yourusername/Kanban-Dashboar
   d.git
   cd Kanban-Dashboard
   ```

2. **Install dependencies**:

   ```
   bash
   Copy code
   ```

```
npm install
```

3. **Start the development server**:

   ```
   bash
   Copy code
   npm start
   ```

4. **Access the app**: Open your browser and go to http://localhost:3000.

Components

1. KanbanBoard.js

- **Purpose**: The main dashboard that holds columns for each ticket status.
- **Props**: None.
- **State**:
  - `liveMode` (boolean): Controls live mode toggle.
  - `initialTickets` (object): Initial tickets are fetched from a JSON file and distributed into columns based on their status.
- **Rendering**:
  - Contains a button to toggle live mode.
  - Maps through `initialTickets` to render each column component.

2. KanbanColumn.js

- **Purpose**: Represents a single status column (e.g., To Do, In Progress).
- **Props**:
  - `status` (string): The status represented by this column.
  - `tickets` (array): Tickets associated with the current status.

- **Rendering**:
  - Displays a heading with the column status and ticket count.
  - Maps through `tickets` to render each ticket card within the column.

3. TicketCard.js

- **Purpose**: Displays information for an individual ticket.
- **Props**:
  - `ticket` (object): Contains details of a ticket, including ID, Title, and Description.
- **Rendering**:
  - Shows ticket ID, title, and description.
  - Will later have controls to trigger transitions (e.g., a dropdown or buttons for changing the status based on allowed transitions).

State Management with Redux

1. store.js

- **Purpose**: Configures the Redux store.
- **Setup**:
  - Imports `ticketReducer` to manage the ticket states and transitions.
  - Applies `redux-thunk` for asynchronous action handling if needed.

2. ticketActions.js

- **Purpose**: Defines actions to handle ticket transitions.
- **Actions**:
  - `moveTicket`: Accepts a ticket ID and new status, checks for valid transitions, and dispatches an update if valid.

3. ticketReducer.js

- **Purpose**: Reducer to manage ticket transitions and enforce FSA rules.

- **Implementation**:
    - Maintains a list of tickets, organized by status.
    - Checks each requested transition to ensure it aligns with allowed transitions.
    - Updates the state only if the transition is valid, otherwise returns the existing state.

FSA Logic in Redux

- **Allowed Transitions**:
    - `To Do` → `In Progress`
    - `In Progress` → `Blocked` or `Done`
    - `Blocked` → `In Progress`
- **Invalid Transitions**: If a transition does not match any of these paths, the reducer will block the state update.

Styling

- **KanbanBoard.css**: Styles the main board, arranging columns and aligning content.
- **KanbanColumn.css**: Styles individual columns, defining column headers, spacing, and scroll behavior.
- **TicketCard.css**: Styles each ticket card with padding, border, and font adjustments.

Usage

Viewing Tickets by Status

- Each column displays tickets associated with a specific status, showing the number of tickets in the header.

Live Mode

- Click the "Enable Live Mode" button to toggle live updates.

Ticket Transitions

- Ticket movements are managed through the Redux store.
- Attempting to move a ticket to an invalid state is prevented by the FSA logic in `ticketReducer`.

Example Workflow

1. **Move Ticket**:
    - From **To Do** to **In Progress**: Updates the ticket status if the transition is valid.
    - From **In Progress** to **Done**: Only valid if the ticket has progressed through **In Progress**.