System Design Document for TryAlma

1. Introduction

The **TryAlma** project is a web application developed using **Next.js** to manage lead information efficiently. It provides two main features:

- **Lead Form**: A page where users can submit lead information.
- **Leads Dashboard**: A page where users can view and manage the leads they have submitted.

This document outlines the system design and architecture for TryAlma, focusing on key components, state management, API design, authentication, and scalability considerations.

---

2. System Overview

The application is structured to handle two key use cases:

- **Lead Submission**: Allows users to submit new leads via a form.
- **Lead Management**: Allows users to view and update lead status in a dashboard.

The system is built on **Next.js**, leveraging its features for server-side rendering (SSR) and static site generation (SSG) to ensure optimal performance. The application uses **Redux** for state management, providing a centralized place to store and manipulate lead data.

---

3. Components & Architecture

The application is divided into several modules and components, each fulfilling specific functionality. Here is an overview of the key components:

3.1 Frontend Components

1. **Lead Form Component** (`LeadForm.tsx`):
    - This component is responsible for rendering the lead submission form, including fields like name, email, LinkedIn, visa status, resume, and message.
    - It provides an intuitive interface for users to enter lead details.

2. **Leads Dashboard** (`Leads.tsx`):
    - Displays a list of leads submitted by users.
    - Each lead entry includes information like name, email, visa status, and message. The status of each lead can be updated through the interface.
3. **Layout Component** (`Layout.tsx`):
    - A shared layout component is used to wrap pages with a common navigation bar and footer.

3.2 Backend & API Design

While the application primarily interacts with a mock API during development, the architecture is designed to easily transition to a real backend as required.

1. **API Requests**:
    - **GET /leads**: Fetches the list of all leads.
    - **PUT /lead/:id**: Updates the status of a specific lead.
2. **API Mocking**:
    - In the development phase, API calls are mocked using Redux Thunks. Mocked API responses simulate fetching and updating lead data.
    - The `fetchLeads` action fetches a list of leads, and the `updateLeadStatus` action updates the status of a given lead.

---

4. State Management

State management in TryAlma is handled using **Redux**, which enables centralized management of lead data, loading states, and error handling.

4.1 Redux Store

The Redux store manages the following state:

- **Leads**: Stores the list of all leads fetched from the API.
- **Loading**: A boolean value that indicates whether the application is fetching lead data.
- **Error**: Stores any errors that may occur while fetching or updating leads.

4.2 Actions and Thunks

1. **Fetching Leads**:
    - The `fetchLeads` action retrieves lead data and updates the state when the data is successfully fetched.
    - The action is asynchronous, and a `loading` state is set during the fetching process.
2. **Updating Lead Status**:
    - The `updateLeadStatus` action updates the status of a lead, such as "Pending" or "Reached Out". This is triggered when a user interacts with the button in the dashboard.

4.3 State Flow

- The Redux store is populated with lead data as a result of the `fetchLeads` action. When users interact with the dashboard and update a lead's status, the `updateLeadStatus` action is dispatched, modifying the global state accordingly.

---

5. API Design

The application interacts with a mocked API, with all the data provided via mock responses. Here is a breakdown of the API design:

5.1 Endpoints

1. **GET /leads**: This endpoint retrieves all leads currently stored in the application. It simulates an API call and returns a predefined list of lead objects.

2. **PUT /lead/:id**: This endpoint is used to update the status of a specific lead based on the provided `id`. When a user updates the lead's status, this endpoint is called to simulate the status change.

5.2 Mocked API Logic

- During development, API calls are simulated via Redux Thunks. The `fetchLeads` action dispatches a mocked list of leads, and `updateLeadStatus` simulates updating a lead's status by altering the state.

- If integrated with a real backend, these endpoints would interact with a database to store and retrieve actual lead data.

---

6. Authentication & Security

While the application features a basic mock login for authentication, it is designed to be extended in the future with a full authentication system.

1. **Login Flow**:
    - The app requires users to log in before accessing the Leads Dashboard.
    - Predefined mock credentials (`admin@test.com` and `admin`) are used to simulate authentication.
2. **Security Considerations**:
    - Although the app uses mock authentication, future versions will incorporate security measures such as **JWT tokens** for authentication and **password hashing** for user security.

---

7. Routing & Navigation

The application uses **Next.js**'s file-based routing to manage pages.

1. **Pages**:
    - `/lead-form`: The form for submitting lead information.
    - `/leads`: The dashboard where users can view and update leads.
2. **Navigation**:
    - After submitting a lead, users are redirected to the Leads Dashboard. The routing is handled by **Next.js's `useRouter` hook**.

---

8. Scalability Considerations

The current system design is built with scalability in mind. Some future improvements include:

1. **Pagination for Leads**:

    - As the lead data grows, pagination can be added to the Leads Dashboard to improve performance.
2. **Real Backend Integration**:
    - The mock API will eventually be replaced by real backend endpoints. A full CRUD operation can be implemented, enabling users to add, update, and delete leads.
3. **Authentication Enhancements**:
    - As the user base grows, the authentication system can be enhanced with role-based access control (RBAC), multi-factor authentication (MFA), and session management.

---

9. UI/UX Design

The user interface is designed to be intuitive and easy to use. It features:

- **Lead Submission Form**: A clean form for entering lead information.
- **Leads Dashboard**: A table displaying the submitted leads, where users can manage lead status.
- **Responsive Design**: The UI is responsive, ensuring it is usable on mobile and desktop devices.

The design adheres to best practices in accessibility and usability, ensuring that users can interact with the application effortlessly.

---

10. Conclusion

The TryAlma system is designed to be modular, scalable, and maintainable. It allows users to submit leads, view them on a dashboard, and update their status. This design ensures that the application can grow alongside user needs while maintaining a seamless user experience and efficient data management.