MERN Stack Project Requirements:

Application: Digital Stories

A digital story is a memory that you create. It could be text, image, video. The project should be a usable application that has a front end and a back end, consisting of RESTful API and a database.

# FRONT END (React with Redux)

When a digital stories URL is entered, a user is directed to a landing page. The landing page should have information about the application and Register/Login options

Registration:

To register the user must enter a name, username (unique), and correctly formatted email address with the option to upload a picture. You can also collect other information.

Login:

Once the user logs in, the home page should be shown. We always want to make sure we know who the logged in user is, so information about the logged in user should appear on the page. If someone tries to navigate anywhere by entering the address in the address bar, the user is asked to sign in and then the requested page is shown. The application allows the user to log out and log back in.

Features:

Create a new story for authoring. Author a story with text, image, and video components.

User should be allowed to style the story components with colors and fonts. Style story components with color, font, position, and order. Stories should be persistent for later access to preview, edit, and delete. Stories should be in a list format, one below the other, or in a grid format based on the user's choice

All stories from all users should be visible. A user should be able to upvote/downvote a story but not both. A user should be able to see number of upvotes/downvotes a story has received, and what percentage of total users have reacted to a particular story. A user should also be able to comment on stories.

Trending stories feature should be available at the /trending route. Trending stories include

the most upvoted/commented stories from all users.

In “engagement” a user should be able to see which stories that particular user has reacted to and which of his stories other users have reacted to.

There should be a leaderboard that’s available at the /leaderboard route. Each entry on the leaderboard should contain the following: User’s name, User’s picture, Total number of stories posted, Total upvotes received from all stories combined

For each user, stories should be sortable w.r.t. to creation timestamp, upvotes and downvotes

You should also be able to sort all stories from all users according creation timestamp, upvotes and downvotes

The user should be able to navigate to the leaderboard, to trending stories, and to the form that allows the user to create a new story both from within the app and by typing in the address into the address bar.

Since we want to make sure our application creates a good user experience, the application should show a 404 page if the user is trying to access a story that does not exist. It should also display a navigation bar so that the user can easily navigate anywhere in the application.

For this application, most of the application’s state should be managed by Redux. You’ll find that there are situations where it makes sense to store state outside of the Redux store. Check out what Dan Abramov, the creator of Redux, thinks about [choosing between Redux's store and React's state](https://github.com/reactjs/redux/issues/1287)

Front end should handle all the error messages from backend, it shouldn't crash

Navigation with React router

Asynchronous API calls

Explore the use of Material UI

Clean up functions in useEffect

**Some concepts to include in any practice project of react:**

1. **Component reusing**:

Create a List component in which, render a dynamic list using a ListItem component. For example, in <https://leetcode.com/problemset/all/> list of problems are rendered dynamically creating the list item once and providing different names, acceptance and difficulty values. Also, try to be able to delete a list item from the UI.

1. **Use Context to avoid prop drilling:**

Try using useContext hook and pass a value around to children components using context instead of props.

1. **API calls**:

Sending API calls requires to be done asynchronously, try different methods like JS promises, async await and .then() function to achieve required behaviour.

1. **Loading spinner**:

Try adding any simple loading spinner while you are getting data to render from the API.

1. **Pagination**:

When getting data to render from API, trying to get a whole lot of data at once will slow down your app, try to add pagination to get a limited number of items and once and rest only when called by the user using a next page button. For example in <https://leetcode.com/problemset/all/> , problems list is rendered as 50 per page, the next 50 (51 to 100) if only be fetched once the user calls for it.

1. **Search bar**:

Trying adding a search bar that gets the relevant data from the API when the user clicks search or hits enter. You can also try to add a dynamic search bar that filters the data to render as the search input changes, or you can go a step further and give relevant possible options to choose from as a drop down, just like in <https://www.google.com/> search bar.

1. **Nbar and Routing**:

Create a navbar component that allows changing between components. You can use a router library like <https://www.npmjs.com/package/react-router-dom> to handle routing, and a navbar to navigate between paths to the corresponding component. For example <https://leetcode.com/explore/> navbar navigates between explore, problems, interview etc.

1. **Forms**:

* Try creating any form and collecting the data from all fields into an object in an onSubmitHandler function. I would recommend <https://www.npmjs.com/package/formik> library for it.
* Also try to add client side validation on the form fields and make it responsive. For instance, if we try to add name in a form field for email, it shows an error saying invalid since name doesn’t match an email regex which is supposed to contain @ and not contain spaces etc. Also when an app has specific requirements for passwords to contain, or when the reentered password doesn’t match. Try using <https://www.npmjs.com/package/yup> library from client side validation and responsiveness. (errors showing why and input can’t be accepted). Forms submitting calls should go to the backend until all fields match the validation schemas.
* In addition to this, try to include all possible components in your form from any UI library to choose such as material UI, Chakra UI etc. They can contain checkboxes, radio buttons, dropdowns, data picker, time picker, password etc. Try to render them with dynamic values.
* You can also try creating multi-level forms where when you click next, another form comes up and at the end of the form, all the data is collected into one object to be sent to the backend on a submit call.

1. **Redux**
2. **React hooks:** understanding react hooks concepts and try to utilise them in your projects as required, you may wanna learn about useState, useEffect, UseCallback, useMemo, useNavigate, useSelector etc. (Some of them are not from React library)

# BACKEND:

Database: MongoDB

Discuss the choice to use a normalized vs denormalized database in your application.

Application should use list fields in the schema, Application should use an ORM e.g. Mongoose

Application should be able to perform CRUD Operations on the database

**APIs:**

The application should have authentication and authorization

REST API

**Testing:**

Integration tests

JEST

Cypress testing for front end