**API—INDEX .JS**

Basic platform for journal platform with user authentication, post creation, retrieval functionalities.

1. **Middleware Setup**: It sets up middleware using **app.use()** for JSON parsing, cookie parsing, CORS configuration, and serving static files from the **uploads** directory.
2. **Imports:** The code imports necessary modules like express, cors, mongoose, bcrypt, jsonwebtoken, cookie-parser, multer, and fs.
3. **User Registration**: Handles POST requests to /register for user registration. It hashes the password using bcrypt before saving it to the database.
4. **User Login:** Handles POST requests to /login for user login. It compares the hashed password with the one stored in the database and generates a JWT token upon successful login.
5. **User Profile:** Handles GET requests to /profile to fetch the user's profile using a JWT token for authentication.
6. **User Logout:** Handles POST requests to /logout to remove the JWT token from the cookie.
7. **Post Creation:** Handles POST requests to /post for creating a new post. It also handles file uploads using multer and saves the file path in the database.
8. **Post Update:** Handles PUT requests to /post for updating a post. It allows updating the post details and the cover image.
9. **Post Retrieval:** Handles GET requests to /post to retrieve a list of posts. It also supports pagination and sorting.
10. **Single Post Retrieval:** Handles GET requests to /post/:id to retrieve a single post by its ID.
11. Server Initialization: Starts the Express server and listens on port 5000.

**Bycrpipt:** security risk ke leye

**Jasonwebtoken:** JWTs is used for user authentication. When a user logs in successfully, a JWT containing the user's information (such as username and user ID) is generated and sent to the client. The client includes this JWT in subsequent requests to authenticate the user.

**Cors :** diff port and domain

**API/MODELS--POST.JS**

Go to mogodb for post, under post will come all saved at backend.

In moongoos destructed {Schema, model} to define scehema and create model.

**PostSchema:** It defines the structure of the Post document in MongoDB.

The schema has the following fields:

* title: the title of the post.
* summary: summary or brief description of the post.
* content: A string representing the main content of the post.
* cover: A string representing the path or URL to the cover image of the post.
* author: A reference to the User model, indicating the author of the post. It is stored as a Schema.Types.ObjectId and is linked to the User model using the ref option.

**{ timestamps: true }:** create at and update at. This will automatically be set when a document is created or updated.

**PostModel:** This is Mongoose model created using the model function. It takes two arguments: the name of the model ('Post') and the schema (PostSchema).

**Exporting the Model:** module.exports = PostModel;: This line exports the PostModel so that it can be imported and used in other parts of the application, such as in routes for interacting with Post documents in the database.

**API/MODELS--User.JS**

sets up a Mongoose schema and model for a User with username and password fields, enforcing certain constraints on the data.

User schema have :

**username:** A string representing the username of the user. It is required, must be at least 4 characters long, and must be unique.

**password:** A string representing the password of the user. It is required.

**Client/src—app.js**

App.js file sets up the **routing** and **context provider**. With this I can navigate between different pages based on the URL and provides user context using the UserContextProvider.

* **import {Route, Routes}** from "react-router-dom", Imports the Route and Routes components from react-router-dom for defining routes in your application.
* **import {UserContextProvider}** from "./UserContext",provides user authentication contex.
* **Imp app function…** access with paths eg /post/:id, /edit/:id: Routes to view and edit post pages based on the post ID.
* **export default** for simple import syntax

**App.test.js** is used for testing the App component using the @testing-library/react library.

**Client/src—editor.js**

For providing Rich text editor into a React application, allowing users to input and format text easily. It have ie it receives props for value, theme, onChange, and modules and have toolbar options.

**Client/src—Header.js**

Link component to navigate link, it uses use context to get context of currently logged in user.

The **logout function** is called when the user clicks on the logout link. It sends a POST request to the server to logout the user and sets the user information in the context to null. If the user is logged in, it displays a link to create a new post and a logout link with the username. If the user is not logged in, it displays login and register links.

**Client/src—index.js**

Entry point of a React application. it renders the App component inside a BrowserRouter, which is used to enable routing in the application using the react-router-dom. ReactDOM.createRoot method is used to create a root for the application, and root.render is used to render the App component inside the root. Strict mode for highlighting problems during development. The reportWebVitals function is used to report performance metrics to an analytics server.

**Client/src—layout.js**

Common layout function or application. To have consistent layout throughout=included the Header component at the top and used the Outlet component from react-router-dom to render the content of the current route beneath the header.

**Client/src—post.js**

To display individual single post in my application, making it easy to navigate and read posts. receives various props such as \_id, title, summary, cover, content, createdAt, and author.

* It renders a div with the class name post.
* Inside div=two child elements:

1. img tag, links to the individual post page.
2. A div containing the post title, author's username, post creation date, and a summary of the post content.

* The createdAt date is formatted using the formatISO9075 function from the date-fns library to display it in a readable format.
* The Link components is for creating links to the individual post page, allowing users to click on the post title or image to navigate to the full post content.

**Client/src—reportwebvitals.js**

The reportWebVitals function is used to report performance metrics of your web application, such as Cumulative Layout Shift (CLS), First Input Delay (FID), First Contentful Paint (FCP), Largest Contentful Paint (LCP), and Time to First Byte (TTFB). reportWebVitals function is exported as the default export, making it available for use in other parts of your application.

**Testsetup.js**

simplify your test setup and make your test code more readable by allowing you to use custom matchers like toHaveTextContent directly in your tests.

Apply globally to tests.

**Client/src—usercontext.js**

By this I can manage user information in a centralized way and share it across my React application using the UserContext context and UserContextProvider component.

* **UserContext**: created with createContext({}), where the initial value is an empty object {}. This context will be used to share user information across the application.
* **UserContextProvider**: returns a ‘UserContext.Provider’ component. It uses the useState hook to create a state variable userInfo and a function setUserInfo to update this state. The initial value of userInfo is an empty object {}.
* **UserContext.Provider:** provided by the UserContext context object. It accepts a value prop, which is an object containing userInfo and setUserInfo. This allows any component nested within UserContext.Provider to access and update the userInfo state using UserContext.

**Client/pages—createpost.js**

**useState**: It's used to manage the state of title, summary, content, files, and redirect, it have input value.

**createNewPost:** This function is called when the form is submitted.

The **Editor component** uses the value prop to control its content and the **onChange prop** to update the content state.

**Client/pages—indexpage.js**

It fetches and displays a list of posts on the index page. Each post is rendered using the Post component, which displays the post's title, summary, author, and creation date.

**useState:** It's used to manage the state of posts, which holds an array of post objects fetched from the server.

**Use effect**=list of posts , **Rendering:** renders the Post component for each post in the posts array. It checks if posts.length > 0 to ensure there are posts to render.

**Client/pages—login.js**

pattern is commonly used in React to update the state based on user input. In the onChange event handler, ev.target.value is used to get the new value of the input element. This value is then passed to the setPassword function, which updates the password state in the component.

Yarn add react qill--------instead of text area

Import stylying to not get weird pattern , it will give us editor

In api index.js need to create endpoint do that we can up post request

React time ago gives us like 20 min ago e.t.c

Date fns.org in post.js

Schema type obj id because it will be a reference to our users

.sort created at -1 for latest post at the top

Limit for how much latest post display

We want to grab info about specific post so import use effect

To grab the id of post I can use useParams

Backend =express.js and node.js (api folder)

Frontend= react.js(client)

- **Node.js** is used with Express.js to create a backend server that handles API requests, database operations, and other server-side logic.

-**Express.js** is used to create RESTful APIs for handling user authentication, creating and editing posts, fetching posts, and logging out users.

- **React.js** is used in the frontend to create a dynamic and interactive user interface.

-