**WEEK - 1**

**Step 1: Project Setup**

* Created server.js in **root of backend**.
* Installed dependencies: express, mongoose, cors, dotenv.
* Connected Express server to MongoDB.
* Configured CORS to allow frontend (http://localhost:5173).

**Flow**:

Frontend (http://localhost:5173) → Express (port 8000) → MongoDB

**Step 2: User System (with Portfolio in schema)**

* Created **User model** in models/User.js.
  + Fields: username, email, password, portfolio (array of stock symbols).
* Created **userAuth routes** in routes/userAuth.js.
  + POST /api/auth/signup → Register new user.
  + POST /api/auth/login → Authenticate existing user.
* Hooked routes into server.js under /api/auth.
* Tested with Postman → working fine.

**Flow**:

Client → POST /api/auth/signup → Save user in MongoDB

Client → POST /api/auth/login → Verify user from MongoDB

**🔗 Flow Chart Representation (Till Now)**

┌────────────────────┐

│ Frontend (React) │

│ http://localhost:5173

└─────────┬──────────┘

│

▼

┌────────────────────┐

│ Express Server │

│ (server.js, port 8000)

└──────┬─────────────┘

│

┌─────────────┴────────────────┐

│ │

▼ ▼

┌───────────────────┐ ┌─────────────────────┐

│ /api/auth/signup │ │ /api/auth/login │

│ (register user) │ │ (authenticate user)│

└───────────────────┘ └─────────────────────┘

│

▼

┌────────────────────┐

│ MongoDB (Users) │

│ { username, email, │

│ password, │

│ portfolio: [] } │

└────────────────────┘

**FRONTEND**

**🛠 Workflow (Signup.jsx)**

1. **State Management**
2. const [form, setForm] = useState({ username: "", email: "", password: "" });
3. const [loading, setLoading] = useState(false);
   * form → ek object hai jisme hum **username, email, password** store karte hain.
   * loading → ek flag jo batata hai ki API request chal rahi hai ya nahi.
4. **Navigation Setup**
5. const nav = useNavigate();
   * useNavigate hook React Router se aata hai. Isse signup ke baad user ko **Login page** pe bhej dete hain.
6. **Input Change Handler**
7. const onChange = (e) => setForm({ ...form, [e.target.name]: e.target.value });
   * Har input field ke name attribute ke base par value update hoti hai.
   * Example: agar username input change hua → form.username update hoga.
   * Yeh ek **generic handler** hai jo sabhi input fields ke liye kaam karega.
8. **Form Submission**
9. const submit = async (e) => {
10. e.preventDefault();
11. setLoading(true);
12. try {
13. const res = await API.post("/api/auth/signup", form);
14. alert(res.data.message || "User created");
15. nav("/login");
16. } catch (err) {
17. alert(err.response?.data?.message || err.message);
18. } finally {
19. setLoading(false);
20. }
21. };
    * e.preventDefault() → default form submit behavior (page reload) rokta hai.
    * setLoading(true) → button disable kar dete hain aur message "Creating user..." show karte hain.
    * API call:
    * await API.post("/api/auth/signup", form);

Ye backend ke **signup route** pe form data bhejta hai.

* + Agar success:
    - User ko alert show hota hai (User created)
    - nav("/login") ke through **Login page pe redirect** ho jata hai.
  + Agar fail:
    - Error message alert hota hai (err.response.data.message agar available hai).
  + Finally block:
    - setLoading(false) → request complete hone ke baad loading flag reset hota hai.

1. **Input Fields**
2. <input
3. name="username"
4. value={form.username}
5. onChange={onChange}
6. required
7. />
   * name → decide karta hai form object ka kaunsa property update hoga.
   * value={form.username} → input ko React state ke saath sync rakhta hai.
   * onChange={onChange} → har type karne par form update hota hai.
   * required → field ko empty chhodkar submit nahi kar sakte.
8. **Submit Button**
9. <button type="submit" disabled={loading}>
10. {loading ? "Creating user..." : "Signup"}
11. </button>
    * Agar API call chal rahi hai (loading = true) → button disable + text change.
    * Warna normal "Signup" button show hota hai.

👉 Net result:

* User inputs → React state me store hota hai.
* Submit pe API call → success pe login page redirect.
* Button aur alerts se user ko clear feedback milta hai.

**🛠 Workflow (login.jsx)**

**🔄 Workflow samajhne ke liye**

1. User **form ke input (email + password)** fill karega.
2. Form submit hone par submit() function chalega.
3. Ye API ko request bhejega (/api/auth/login) backend pe.
4. Agar success → user data ko localStorage me save karenge → navigate karega dashboard pe.
5. Agar error → alert me dikha dega.

**🔍 Code Explanation with "Why"**

**1. Imports**

import React, { useState } from "react";

import { useNavigate } from "react-router-dom";

import API from "../api/axios";

* **useState**: form ke andar user jo type karega (email/password), usse track karne ke liye.
* **useNavigate**: react-router ka hook jo hume programmatically dusre page pe bhejne ke kaam aata hai (yaha login ke baad /dashboard).
* **API**: axios instance jo backend se baat karne ke liye banaya gya.

👉 Ye teen import isliye kiye gaye taki **state maintain ho, navigation ho, aur backend se communication ho sake**.

**2. State and navigate**

const [form, setForm] = useState({ email: "", password: "" });

const nav = useNavigate();

* **form** ek object hai jisme email aur password rakhe ja rahe hain.
* Initially dono empty hain.
* **nav** ek function hai jo navigate karega dashboard pe.

👉 Reason: **form input values track karna aur login ke baad redirect karna**.

**3. Input change handler**

**Code:**

const onChange = (e) => setForm({ ...form, [e.target.name]: e.target.value });

**1️ ...form ka use**

* form ek **state object** hai jo abhi ki sari input values hold karta hai:
* { email: "", password: "" }
* Jab user koi ek field type kare (jaise email), hum **existing state ko preserve** karna chahte hain.
* ...form matlab **spread operator**: "form ke sabhi current key-value pairs ko copy karo".
* Agar hum ye nahi lagate, aur sirf { [e.target.name]: e.target.value } likhte → baki fields (jaise password) reset ho jayenge.

**Example:**

form = { email: "abc@gmail.com", password: "12345" }

User type kare email field me: "xyz@gmail.com"

setForm({ [e.target.name]: e.target.value })

// result: { email: "xyz@gmail.com" } → password lost ho gaya

**Isliye hum use karte hain:**

setForm({ ...form, [e.target.name]: e.target.value })

* Result: { email: "xyz@gmail.com", password: "12345" } → baki values safe hain.

**2️ [e.target.name] ka matlab**

* e.target.name → input field ka **name attribute** leta hai (jaise "email" ya "password").
* [e.target.name]: e.target.value ka matlab hai:
  + **dynamic key update**: jis field me user type kar raha hai, uski value update karo.

**Example:**

* Input field:

<input name="email" />

* User type → e.target.name = "email"
* [e.target.name]: e.target.value → { email: "new value" }

**3️ : kyun use hua hai**

* JS me object me key-value define karte hain:

{ key: value }

* [e.target.name]: e.target.value → dynamic key ka syntax
  + Agar key fixed hoti: email: e.target.value → sirf email update hota
  + [e.target.name] → variable ke through key decide hoti (email/password/username)

**4. Submit handler**

const submit = async (e) => {

e.preventDefault();

try {

const res = await API.post("/api/auth/login", form);

const user = res.data.user;

if (user?.password) delete user.password;

localStorage.setItem("user", JSON.stringify(user));

nav("/dashboard");

} catch (err) {

alert(err.response?.data?.message || err.message);

}

};

* **e.preventDefault()**: form submit hone par page reload na ho.
* **API.post("/api/auth/login", form)**: backend ko POST request bhej rahe hain → body me email + password.
* **res.data.user**: backend se user ka data aata hai.
* **Delete password**: backend temporarily password bhej raha hai (learning purpose). Security ke liye localStorage me save karne se pehle delete kar diya.
* **localStorage.setItem**: user data ko browser me save kar diya taaki login hone ka proof rahe.
* **nav("/dashboard")**: agar login success ho gya to dashboard page pe redirect.
* **catch block**: agar error aaya (galat email/password), to alert me dikha do.

👉 Reason: **form ko backend pe bhejna, response handle karna, aur user ko redirect karna**.

**5. UI (JSX Form)**

<form onSubmit={submit}>

<input name="email" ... />

<input name="password" ... />

<button type="submit">Login</button>

</form>

* **onSubmit={submit}**: jab user login button click karega → submit() chalega.
* **name="email" / name="password"**: ye important hai kyunki onChange me humne [e.target.name] use kiya hai.
* **value={form.email} / value={form.password}**: ye state ke saath input ko sync me rakhta hai (controlled component).
* **Button**: submit karne ke liye.

👉 Reason: **inputs ko state se connect karna aur ek proper form ready karna**.

**📝 Simple Summary**

* **useState** → input values track karne ke liye.
* **onChange** → user jo likhe usse update karne ke liye.
* **submit** → backend pe request bhejne, localStorage me user save karne aur redirect karne ke liye.
* **localStorage** → login session maintain karne ke liye.
* **useNavigate** → login ke baad dusre page pe bhejne ke liye.

**🛠 Workflow (Dashboard.jsx)**

**🔄 Workflow**

1. Dashboard component load hote hi **localStorage se user data** fetch hota hai.
2. Agar user login nahi hai → message dikha ke login page pe redirect ka suggestion.
3. Agar user logged in hai → dashboard ke upar welcome message + user ID dikha.
4. PortfolioManager component ko render karte hain, jisme user ka portfolio manage hoga.

**🔍 Code Explanation**

**1️ Fetch user from localStorage**

const raw = localStorage.getItem("user");

* localStorage.getItem("user") → browser ke localStorage se previously save kiya gya user data fetch karta hai.
* **Why**: frontend ko pata hona chahiye ki kaun login hai, tabhi personalized dashboard show hoga.

**2️ Check if user is logged in**

if (!raw) {

return (

<div>

<h2>You are not logged in</h2>

<p>Go to <a href="/login">Login</a></p>

</div>

);

}

* Agar raw null hai → iska matlab user login nahi hai.
* **Return early** → dashboard ke rest components render nahi honge.
* **Why**: Unauthorized access rokna aur user ko login prompt dikhana.

**3️ Parse user object**

const user = JSON.parse(raw);

* localStorage me jo string store hoti hai → JSON.parse se object me convert karte hain.
* **Why**: tab hum user.username, user.\_id etc access kar sakte hain.

**4️ Render welcome message**

<h1>Welcome, {user.username || user.email}</h1>

<p>User ID: {user.\_id}</p>

* **user.username || user.email** → agar username available nahi hai to email show karenge.
* **Why**: Personalized greeting dene ke liye.

**5️ Render PortfolioManager**

<PortfolioManager userId={user.\_id} />

* PortfolioManager component me userId pass kar rahe hain.
* **Why**: PortfolioManager ko backend se portfolio fetch karne ke liye userId chahiye.
* PortfolioManager me CRUD operations (add/remove stocks) user ke liye specific honge.

**✅ Summary**

| **Part** | **Reason** |
| --- | --- |
| localStorage.getItem("user") | Check kaun login hai |
| if (!raw) | Unauthorized user ke liye message/redirect |
| JSON.parse(raw) | String → JS object conversion, username/email access ke liye |
| Welcome message | Personalized UI |
| <PortfolioManager userId={user.\_id} /> | Portfolio management ke liye backend se data fetch aur update |

**🛠 Workflow (PortfolioManager.jsx)**

**🔄 Workflow**

1. Component load hote hi userId ke basis pe portfolio fetch hota hai backend se.
2. Portfolio data state me store hota hai → UI me render hota hai.
3. User naya stock add kare → add function → backend call → state update → input clear.
4. User existing stock remove kare → remove function → backend call → state update.

**🔍 Code Explanation**

**1️ State declarations**

const [portfolio, setPortfolio] = useState([]);

const [symbol, setSymbol] = useState("");

const [loading, setLoading] = useState(false);

* **portfolio** → user ke stocks ka list
* **symbol** → input field ka value
* **loading** → add operation ke time button disable karne ke liye

**Why**: React state se hi UI dynamically update hoti hai.

**2️ Fetch portfolio function**

const fetchPortfolio = async () => {

try {

const res = await API.get(`/api/portfolio/${userId}`);

setPortfolio(res.data.portfolio || []);

} catch (err) {

alert(err.response?.data?.message || err.message);

}

};

* API.get → backend ke endpoint se portfolio fetch karta hai.
* setPortfolio → fetched portfolio ko state me save karta hai.
* **Why**: component mount hote hi user ka latest portfolio dikhe.

**3️ useEffect hook**

useEffect(() => { fetchPortfolio(); }, [userId]);

* **useEffect** → component render ke baad fetchPortfolio call karta hai.
* [userId] → dependency array, agar userId change hua → portfolio refetch hoga.
* **Why**: ensure latest portfolio data fetch ho har user ke liye.

**4️ Add stock function**

const add = async (e) => {

e?.preventDefault();

if (!symbol.trim()) return;

setLoading(true);

try {

const res = await API.post(`/api/portfolio/${userId}/add`, { symbol });

setPortfolio(res.data.portfolio);

setSymbol("");

} catch (err) {

alert(err.response?.data?.message || err.message);

} finally { setLoading(false); }

};

* **e.preventDefault()** → form submit hone par page reload na ho
* **Validation** → empty string check
* **setLoading(true)** → button disable / loading indicator
* **API call** → new symbol backend me add
* **setPortfolio** → state update → UI me reflect
* **setSymbol("")** → input clear

**Reason**: CRUD operation ka frontend flow maintain karne ke liye.

**5️ Remove stock function**

const remove = async (sym) => {

if (!confirm(`Remove ${sym} from portfolio?`)) return;

try {

const res = await API.delete(`/api/portfolio/${userId}/remove/${sym}`);

setPortfolio(res.data.portfolio);

} catch (err) {

alert(err.response?.data?.message || err.message);

}

};

* **Confirm dialog** → user accidentally delete na kare
* **API.delete** → backend se stock remove
* **setPortfolio** → state update → UI refresh

**Reason**: safe deletion and real-time UI update.

**6️ Return / JSX**

<form onSubmit={add}> ... </form>

<ul>

{portfolio.map((s) => (

<li key={s}>

{s} <button onClick={() => remove(s)}>Remove</button>

</li>

))}

</ul>

* Form → naya stock add karne ke liye
* List → portfolio items show
* Remove button → remove function call

**Why**: user interactive UI ke through apna portfolio manage kare.

**✅ Summary**

| **Part** | **Reason** |
| --- | --- |
| useState | Dynamic UI update ke liye |
| fetchPortfolio | Initial aur latest portfolio fetch |
| useEffect | Component mount ke baad API call |
| add function | Naya stock add, input clear aur loading handle |
| remove function | Stock remove + confirm + state update |
| JSX | Form aur list render, button actions bind |

**🛠 Workflow (api/axios.js)**

**🔹 Code Breakdown**

import axios from "axios";

* **axios** → ek popular HTTP client library jo browser aur NodeJS me kaam karta hai.
* **Why**: Ye fetch se better hai because easy syntax, automatic JSON parsing, interceptors, aur error handling provide karta hai.

const BASE = import.meta.env.VITE\_API\_URL || "http://localhost:8000";

* **import.meta.env.VITE\_API\_URL** → Vite framework me environment variable ko access karne ka tarika.
  + .env file me defined VITE\_API\_URL=http://localhost:4000
  + Ye backend ka base URL store karta hai.
* **Fallback** → agar .env variable missing hai, tab http://localhost:8000 default use hoga.
* **Reason**: Project deploy hone par easy switching between local, dev, production backend.

**Example**:  
VITE\_API\_URL = http://localhost:4000  
=> BASE = "http://localhost:4000"

const API = axios.create({

baseURL: BASE,

headers: { "Content-Type": "application/json" },

// withCredentials: true, // keep for later if you enable cookies/sessions

});

**1️ axios.create()**

* Ye ek **custom instance** banata hai jisse hum baar-baar base URL aur headers set kar sake.
* Har API call me ye automatically use hoga.

**2️ baseURL**

* Har request ke liye **base URL** define karta hai.
* API.get("/api/portfolio/123") → full URL banega:
* http://localhost:4000/api/portfolio/123
* **Reason**: Har jagah full URL likhne ki zarurat nahi, DRY principle maintain hota hai.

**3️ headers**

headers: { "Content-Type": "application/json" }

* Har request me ye header automatically include hoga.
* **Why**: Backend ko pata chal sake ki request JSON format me hai.

**4️ withCredentials (commented)**

* Agar backend cookies/session use karega → withCredentials: true enable karna hoga.
* Filhal hum JWT/cookies nahi use kar rahe, isliye comment me rakha hai.

export default API;

* Is custom axios instance ko baaki frontend files me import karenge:

import API from "../api/axios";

* Phir use karenge:

API.get("/api/portfolio/123")

API.post("/api/auth/login", form)

API.delete("/api/portfolio/123/remove/AAPL")

**🔹 Example Full URL Calculation**

1. **Frontend call:**

API.get("/api/portfolio/123")

1. **Base URL**: http://localhost:4000
2. **Combined URL**: http://localhost:4000/api/portfolio/123

Similarly:

API.post("/api/auth/login", { email, password })

* Full URL: http://localhost:4000/api/auth/login
* Body: { email, password }
* Header: Content-Type: application/json

**✅ Key Points / Reasoning**

| **Concept** | **Why Needed** |
| --- | --- |
| axios.create | DRY, reusable instance with baseURL & headers |
| baseURL | Har request ke liye full URL likhne ki zarurat nahi |
| headers | Backend ko pata chale ki request JSON hai |
| withCredentials | Future me cookies/session ke liye optional |
| export default | Ek hi centralized instance har component me use kar sake |

**CONNECTING FRONTEND AND BACKEND (AXIOS.JS)**

**Overview — axios file mein hum kya implement kar rahe hain**

Hum ek centralized HTTP client bana rahe hain jo: backend ki base URL set kare, har request ko JSON headers de, JWT token automatically add kare, global 401 handling kare, aur token ko manage karne ke helper functions provide kare.  
(Ye sab isliye taaki frontend code clean rahe aur auth logic ek jagah ho.)

**1) Base URL banaya gaya**

**Code (one line):**

const BASE = import.meta.env.VITE\_API\_URL || "http://localhost:8000";

**Kyun:** agar environment variable set ho to use karo, warna local dev server use karenge — easy switching between dev/prod.

**2) Axios instance create kiya (config)**

**Code (one line):**

const API = axios.create({ baseURL: BASE, timeout: 10000, headers: { "Content-Type": "application/json" } });

**Kyun:** ek shared client jisme base URL, timeout aur default JSON header set ho — baar-baar same config likhne ki zaroorat nahi.

**3) withCredentials comment (note)**

**Code (one line, commented):**

// withCredentials: true, // keep for later if you enable cookies/sessions

**Kyun:** agar future mein httpOnly cookie based auth use karoge to ye enable karna padega; filhal localStorage JWT ke saath use nahi karte.

**4) Request interceptor — Authorization header automatically add**

**Code (one line core):**

const token = localStorage.getItem("token"); if (token) config.headers.Authorization = `Bearer ${token}`;

**Kyun:** har request ke saath JWT bhejna automatically — har component mein manually header na lagana pade.

**5) Request interceptor — try/catch around localStorage**

**Code (one line):**

try { const token = localStorage.getItem("token"); ... } catch (e) { /\* ignore \*/ }

**Kyun:** SSR ya restricted browser contexts mein localStorage throw kar sakta hai — app crash se bachane ke liye safe guard.

**6) Response interceptor — global 401 handling**

**Code (one line core):**

if (error?.response?.status === 401) { localStorage.removeItem("token"); window.location.replace("/login"); }

**Kyun:** agar token invalid/expired ho to centrally token clear karke user ko login page pe bhej denge — har component mein same logic duplicate na karein.

**7) Response interceptor — rethrow error**

**Code (one line):**

return Promise.reject(error);

**Kyun:** specific components ko bhi error handle karne ka mauka chahiye (e.g., toast messages), isliye error ko propagate karte hain.

**8) setAuthToken helper**

**Code (one line):**

export function setAuthToken(token) { if (!token) return; localStorage.setItem("token", token); }

**Kyun:** login ke baad token store karne ka single place — consistency aur testability badhti hai.

**9) clearAuthToken helper**

**Code (one line):**

export function clearAuthToken() { localStorage.removeItem("token"); }

**Kyun:** logout ya global 401 handling mein token clear karne ka single function — DRY principle.

**10) getAuthToken helper**

**Code (one line):**

export function getAuthToken() { return localStorage.getItem("token"); }

**Kyun:** kahin token chahiye ho (debug/conditional logic) to direct localStorage access na karein — wrapper use karo.

**11) export default API**

**Code (one line):**

export default API;

**Kyun:** baki components sirf import API from "src/api/axios" karke use kar sakte — centralized client se sab requests jayengi.