1. **User Authentication System**

**Objective:** Implement a secure authentication system using JWT (JSON Web Token).

**Features:**

* User registration and login with email & password.
* Hash passwords using **bcrypt.js** before storing in MongoDB.
* Use **JWT** to generate authentication tokens.
* Protect routes (e.g., user profile) using authentication middleware.
* Logout functionality by **clearing the token** from the frontend.

**Implementation Steps:**

1. Set up **Express.js + MongoDB (Mongoose)** backend and react,tailwindcss for frontend.
2. Create a **User model** with username, email, password, and timestamps.
3. Implement **bcrypt.js** to hash passwords before saving to the database.
4. Use **jsonwebtoken (JWT)** to issue and verify tokens.
5. Create a **React frontend** with forms for login and registration.
6. Store JWT in **localStorage**.
7. Protect private routes using **middleware** in Express.js.
8. Display username in home page

**2️.CRUD API for a Blog App**

**Objective:** Build a full-stack blog app where users can create and manage blog posts.

**Features:**

* **Create, Read, Update, Delete** blog posts.
* Store blog data in **MongoDB** using **Mongoose**.
* Use **Express.js** to handle API requests.
* Design a **React,tailwindcss frontend** to display posts dynamically.
* Implement an **edit/delete option** for post owners.

**Implementation Steps:**

1. Create an Express.js **API** with routes: POST /blogs, GET /blogs, PUT /blogs/:id, DELETE /blogs/:id.
2. Use **Mongoose** to define the Blog schema (title, content, author, timestamp).
3. Build a **React frontend** with a form to create/update posts.
4. Fetch blogs from the backend and display them in a list.
5. Implement **edit & delete buttons** with API calls.

**3.E-commerce Cart System**

**Objective:** Create an online shopping cart where users can add/remove products.

**Features:**

* Product listing page with **add to cart** button.
* Cart page with **increase/decrease quantity** feature.
* Store products & cart data in **MongoDB**.
* Checkout system with total price calculation.
* Implement **JWT-based authentication** for logged-in users.

**Implementation Steps:**

1. Build an **Express.js API** to handle product and cart operations.
2. Create a **Product schema** in MongoDB with name, price, image, etc.
3. Implement a **cart schema** linked to users.
4. Build a **React frontend,Tailwindcss** with product cards and cart functionality.
5. Use  **Context API** to manage the cart state.

**4.Weather App with External API**

**Objective:** Fetch live weather data using an API and display it dynamically.

**Features:**

* Search bar to input city names.
* Fetch weather data using the **OpenWeatherMap API**.
* Show temperature, humidity, and weather conditions.
* Store the last searched city in **MongoDB** for returning users.
* Toggle between **light mode & dark mode**.

**Implementation Steps:**

1. Get an **API key** from OpenWeatherMap.
2. Create a **React frontend,Tailwindcss** with an input field and search button.
3. Use **Axios** to fetch weather data.
4. Store **last searched city** in MongoDB.
5. Implement **dark/light mode** using CSS.

**5.File Upload System**

**Objective:** Allow users to upload files and store them in MongoDB or Cloudinary.

**Features:**

* Upload images, PDFs, and other files.
* Store files in **MongoDB (GridFS)** or **Cloudinary**.
* Display uploaded files in a React UI.
* Implement **file validation** (size & format restrictions).

**Implementation Steps:**

1. Set up **Multer** in Express.js for file uploads.
2. Store files in **server**.
3. Build a **React frontend,tailwindcss** with an upload button.
4. Display uploaded files in a **table/list**.