📘 Meetup Lite – Theory-Focused Full Stack MERN Project Breakdown

---

🔤 1. Key Technologies Explained (Before Project Begins)

1. JavaScript (JS)

JavaScript is the **programming language** used both in the **frontend (React)** and **backend (Node.js)**. It is:

- **Dynamically typed** and **event-driven**
- Supports **asynchronous programming** (via async/await, Promises)
- Works well with **JSON**, making data transfer between client and server seamless

2. React.js

React is a **JavaScript library** for building **component-based UIs**. Key concepts:

- **JSX**: Allows writing HTML inside JavaScript
- **Components**: Reusable building blocks like <Navbar />, <EventCard />
- **Hooks**: Functions like useState, useEffect, useContext manage component logic
- **React Router**: Enables navigation without page reloads (Single Page Applications)

3. APIs (Application Programming Interface)

APIs define how software components communicate. In this project:

- The frontend makes **HTTP requests** to Express-based APIs
- The backend responds with **JSON data**
- Follows **REST principles**: URLs represent resources (e.g., /api/events)

4. Node.js + Express.js

- **Node.js** runs JavaScript on the server
- **Express.js** simplifies routing, middleware, and server logic
- Example: app.get('/api/events', ...) handles a GET request

5. MongoDB + Mongoose

- MongoDB is a **NoSQL database** — stores data in flexible JSON-like documents
- Mongoose is an **ODM (Object Data Modeling)** tool
    - Defines schemas (e.g., User schema)
    - Validates and interacts with the database using JavaScript objects

6. JWT (JSON Web Tokens)

JWT provides **stateless authentication**:

- On login, the server issues a token
- The token is stored in the frontend and sent with every API request in headers
- Backend verifies the token to identify users without storing sessions

7. Bcrypt

Used to **hash passwords securely**

- Even if the database is compromised, original passwords can't be retrieved
- Hashing is one-way and uses random salt to prevent guessability

8. Context API

A React feature for **global state management**

- Manages authentication state (user info, token)
- Accessible from any component without prop-drilling

9. Tailwind CSS

A **utility-first CSS framework**

- Instead of writing custom CSS, you apply pre-built classes (e.g., bg-blue-500, p-4)
- Leads to **faster styling**, consistent design, and mobile responsiveness

---

🧩 2. Project Architecture (Folder-by-Folder)

pgsql

CopyEdit

meetup-lite-main/

├── client/ --> React frontend

├── server/ --> Express backend

├── README.md

🎨 3. Frontend (React)

📁 Structure:

- client/src/components/ – Reusable UI elements (Navbar, EventCard)
- client/src/pages/ – Route-specific views (Home, Login, Register, Dashboard)
- client/src/context/ – Contains AuthContext for global state
- client/src/services/ – Functions for making API calls

🔐 AuthContext (React + Context API)

Manages:

- user: Stores logged-in user info
- token: Stores JWT token
- login(), logout(), register() – Interact with backend

Why important?
→ Avoids prop drilling, centralizes user state, and persists sessions via localStorage.

🔀 Routing with React Router

jsx

CopyEdit

<Routes>

<Route path="/" element={<Home />} />

<Route path="/login" element={<Login />} />

<Route path="/dashboard" element={<Dashboard />} />

</Routes>

SPA Behavior:

- Navigation does not reload the page
- Only relevant components re-render

🔧 4. Backend (Node + Express)

📁 server/

- server.js – Application entry point
- routes/ – API endpoints for /users and /events
- controllers/ – Business logic for handling API responses
- models/ – MongoDB schemas (User.js, Event.js)
- middleware/requireAuth.js – Auth token checker

📄 Example API Flow

📜 Route: RSVP to an event

1. React makes a PUT request:

```
js
CopyEdit
fetch("/api/events/64abc/rsvp", {
  method: "PUT",
  headers: {
    Authorization: `Bearer <token>`
  }
});
```

2. Express middleware requireAuth.js:
   - Checks token
   - Extracts user ID and adds to req.userId
3. eventController.js:
   - Finds event by ID
   - Adds req.userId to event's attendees array

🧪 5. MongoDB Schema Design

🔸 User Schema

```
js
CopyEdit
{
  name: String,
  email: String,
  password: String (hashed),
  rsvps: [ObjectId of Event]
}
```

🔸 Event Schema

```
js
CopyEdit
{
  title: String,
  description: String,
```

date: Date,

location: String,

organizer: ObjectId of User,

attendees: [ObjectId of Users]

}

This design supports:

- One-to-many and many-to-many relationships
- Dashboard features like “My Events” and “My RSVPs”

🔁 6. Application Flow Example (Scenario)

🎯 Goal: A user logs in, creates a meetup, RSVPs, and views it in dashboard

Step-by-step:

1. User logs in → token saved in localStorage
2. User creates an event via /api/events → POST request with token
3. User visits homepage → sees all events
4. User RSVPs → PUT /api/events/:id/rsvp with token
5. Dashboard fetches:
    - /api/events/my-events → events created
    - /api/events/my-rsvps → events RSVP’d to

🌐 7. Deployment Theory

Frontend: Vercel or Netlify

- Inside client/, run:

bash

CopyEdit

npm run build

- Upload dist/ folder
- Use VITE_API_URL for backend endpoint

Backend: Render or Railway

- Push to GitHub
- On Render:

- Add environment variables: JWT_SECRET, MONGO_URI
- Backend listens on PORT=5000 or default

📈 8. Future Features & Learning Extensions

Feature	Concept Involved
Google OAuth	Third-party auth integration
Email Notification	NodeMailer or SendGrid
Event Comments	Real-time chat with Socket.io
Admin Role	Role-based access control (RBAC)
Mobile App	Use React Native with same API

✅ Final Thoughts

Meetup Lite is a powerful demonstration of:

- Full-stack knowledge (MERN)
- API consumption and production
- JWT auth & role-based logic
- Component-based frontend
- Database design with Mongoose
- Clean folder architecture

It's perfect for your resume, interviews, or as a starter for real products.