ATMOS- Feel the City. Live the Vibe

**Why VibeMap Is Relevant in 2025**

**1. Post-Pandemic Social Behavior**

* People now prefer **real-time, dynamic data** to decide where to go.
* Safety, crowd levels, and ambiance are major concerns—VibeMap provides a **mood-based discovery layer** for urban spaces.

**2. Rise of Hyperlocal Experiences**

* Platforms like Instagram, Threads, and Google Maps are not mood-contextual.
* VibeMap fills that gap by allowing users to **explore places based on current vibes** like "Chill", "Romantic", or "Too Crowded".

**3. Crowdsourced & Community-Driven**

* Users today love being part of a community—especially Gen Z and millennials.
* VibeMap encourages engagement via **tagging, karma points, and real-time updates**, gamifying exploration.

**4. Urbanization + Overcrowding**

* In metro cities, choosing where to go based on **real-time noise/crowd/weather data** helps users avoid stress.
* It promotes mental well-being by helping users find peaceful or enjoyable spots.

**5. Event Discovery + Mood Sync**

* Integration of AI for **sentiment detection from social posts** allows the app to surface trending areas or hidden gems—**automatically**.

**6. Web3 & Decentralized Identity (future scope)**

* The idea of **anonymous mode with karma reputation** could evolve into Web3 identity badges.

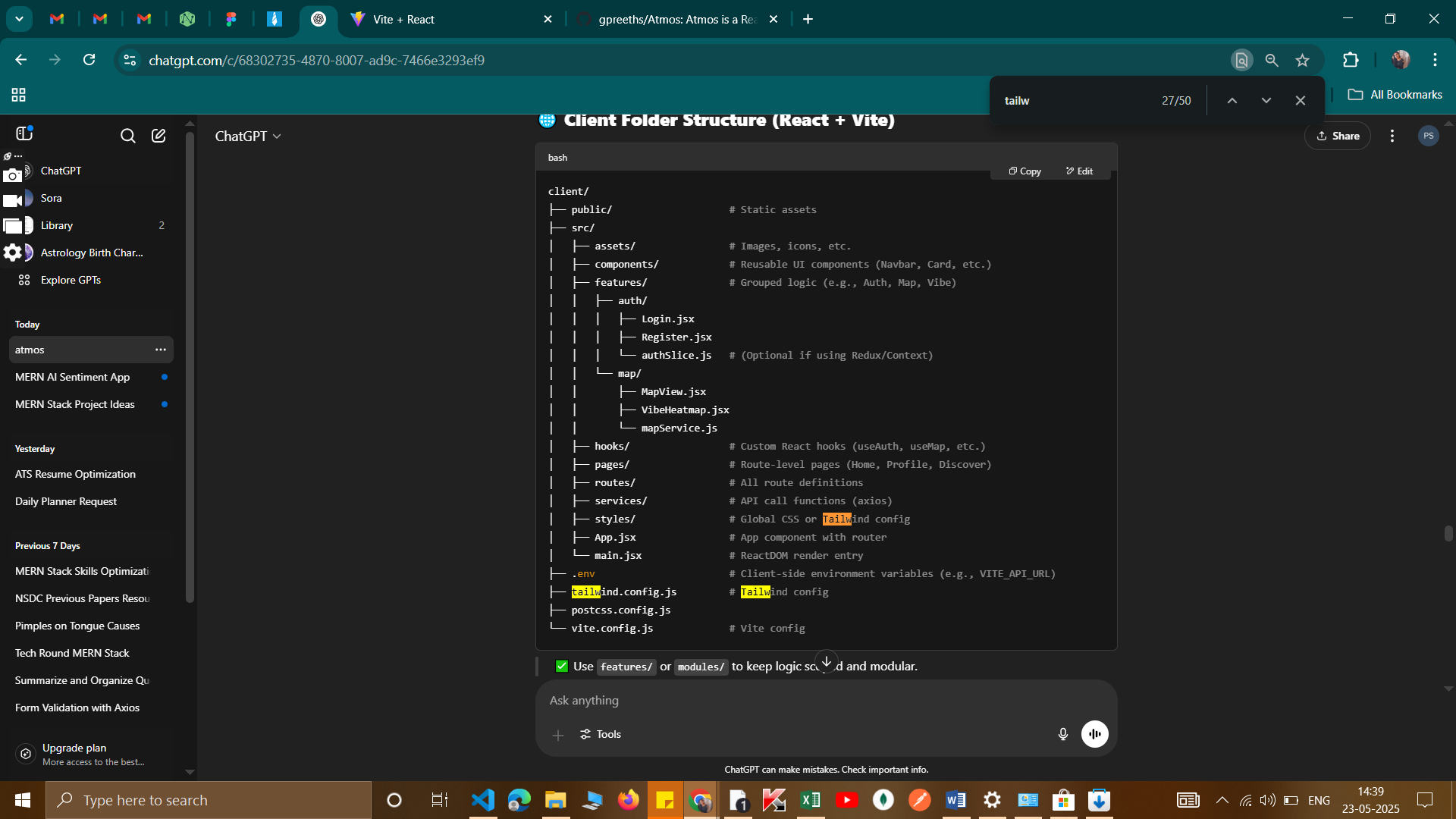
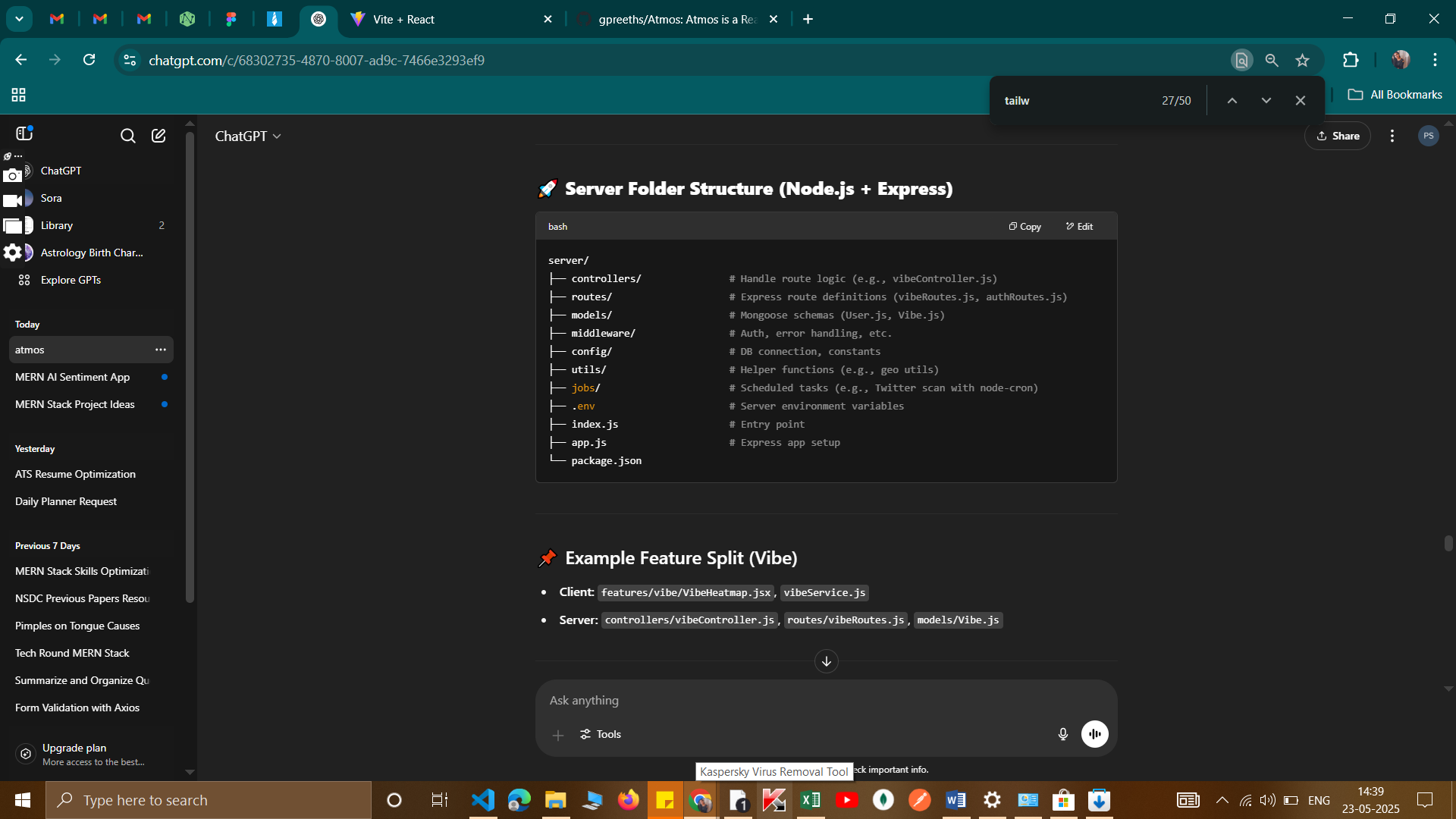
**7. Tech Trends: AI + Real-time Data**

* Combines **AI/ML, live maps, and real-time user interaction**, which are core to modern app ecosystems.

| **Day** | **Focus** |
| --- | --- |
| 1 | Finalize features, ER diagram, folder structure, setup repos |
| 2 | Build MongoDB schemas + User/Auth API (JWT, bcrypt) |
| 3 | Vibe Tag API + Real-time socket setup with socket.io |
| 4 | Heatmap API + Store vibes with geo-coordinates |
| 5 | Frontend setup + Auth pages (Login, Register, Anonymous) |
| 6 | Map view with Google Maps API + Heatmap Layer |
| 7 | Post Vibe UI + Real-time updates with sockets |
| 8 | Mood-based place suggestions (basic filter UI) |
| 9 | Weather/traffic API + Noise level estimator (optional) |
| 10 | Flask microservice – sentiment analysis for geo-tweets (optional) |
| 11 | Integrate AI with backend + real-time mood scoring |
| 12 | End-to-end testing, dummy data seeding |
| 13 | UI polish, responsiveness, gamification elements |
| 14 | Deploy FE, BE, ML microservices + write README + make demo video |

Since you're already familiar with **Render**, you can:

* Use **Render for backend**
* Try **Vercel for frontend** (it’s trending, fast, and developer-friendly)
* Use **MongoDB Atlas**
* Use **Replit** for the ML microservice



Perfect! Let’s define the relationships between your finalized entities:

**1. User**

* Can **create** many events.
* Can **bookmark** many events.
* Can **comment** on many events.
* Can **receive** many notifications.

**2. Admin**

* Can **approve/reject** events.
* Can **send** notifications.

**3. Event**

* **Belongs to** a user (creator).
* **Has many** bookmarks.
* **Has many** comments.
* **Belongs to** a category.
* **Has one** approval status (pending/approved/rejected).
* **Includes** location data (for map display).

**4. Bookmark/Favorite**

* **Belongs to** a user.
* **Belongs to** an event.

**5. Comment**

* **Belongs to** a user.
* **Belongs to** an event.

**6. Notification**

* **Belongs to** a user (receiver).
* **Optionally belongs to** an event (if it’s an event update).

**7. Category**

* **Has many** events.

Would you like a text-based ER diagram or Mongoose schema templates for each entity next?

Models

Event.js-

 **Mongoose doesn't have a Point type** — it uses standard JavaScript types like String, Number, Date, etc.

 In GeoJSON (which MongoDB uses for geolocation), "Point" is a **value**, not a type.

 So we define:

js

CopyEdit

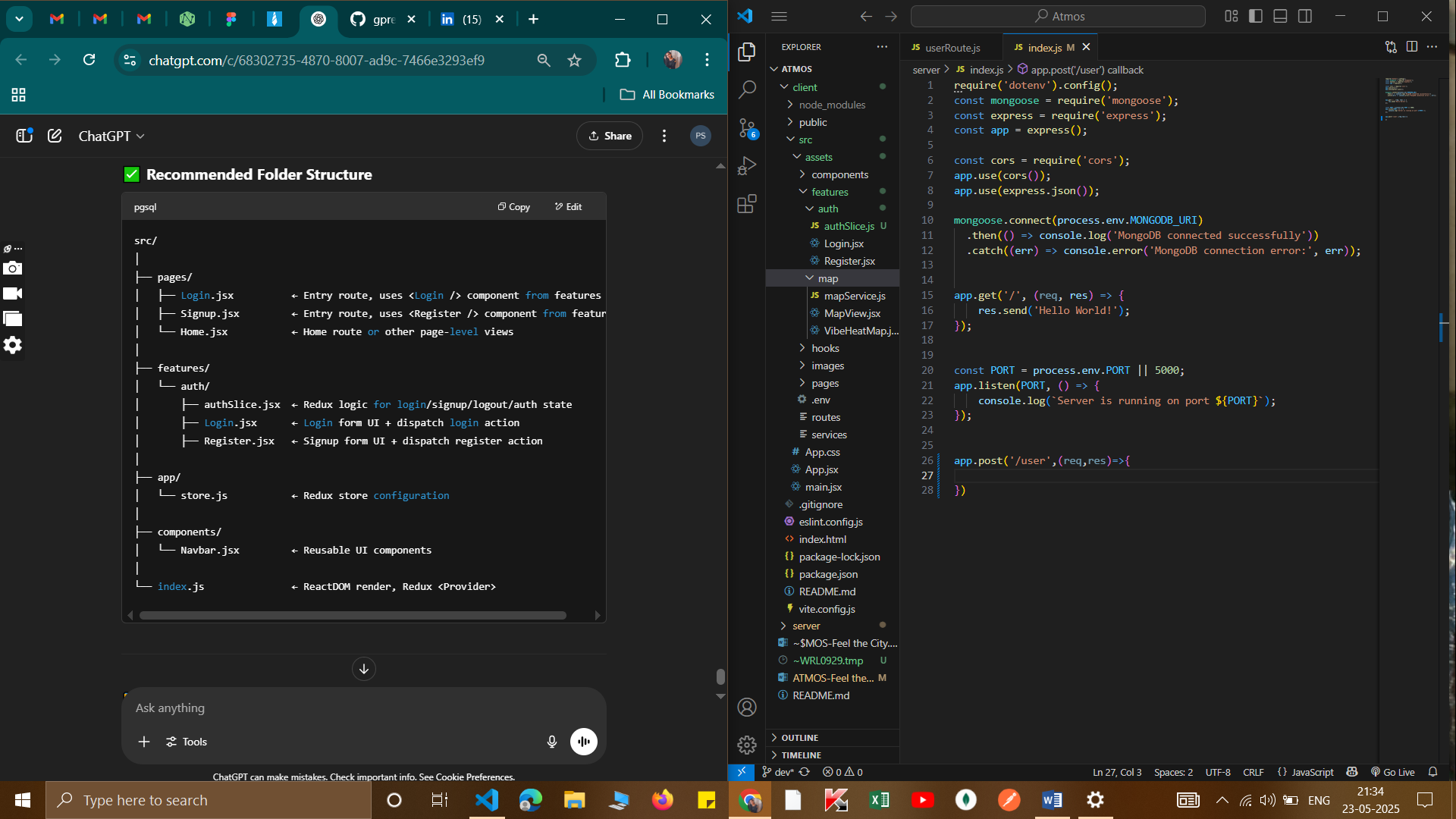
type: {

type: String, // This tells Mongoose "this is a string field"

enum: ['Point'], // Validates that only 'Point' is allowed

default: 'Point' // Automatically assigns 'Point' if not provided

}



Atmos-2.0

Email verification

Signup with google