High Level Design (HLD) for Wayfindr

Joey Driedger
Adam Van Woerden
Keagan Purtell
John Guo

1. Architecture Overview

1.1 Summary

- **Client (Frontend):** Server-rendered EJS views with client-side JavaScript for Map interactions, node UI, and favorite nodes display. Mapbox GL JS renders markers and user favorites on the map.
- **Server (Backend):** Node.js + Express, serving pages and providing REST API endpoints (`/api/nodes`, `/api/favorites`, `/auth`).
- **Data Layer:** Firebase Firestore stores nodes, user profiles, and user favorites. Firebase Auth handles authentication.
- **Authentication:** Firebase Authentication (email/password) with server-side verification via Firebase Admin SDK.
- **Dev Tooling / Infrastructure:** dotenv for configuration, Winston + Morgan + Chalk for logging and colored console output, centralized error handler middleware.
- **Modules:** ESM (`type: module`), modular route design (`auth.js`, `nodes.js`, `favorites.js`) for maintainable code.

1.2 Architectural decisions, rationale, alternatives & trade-offs

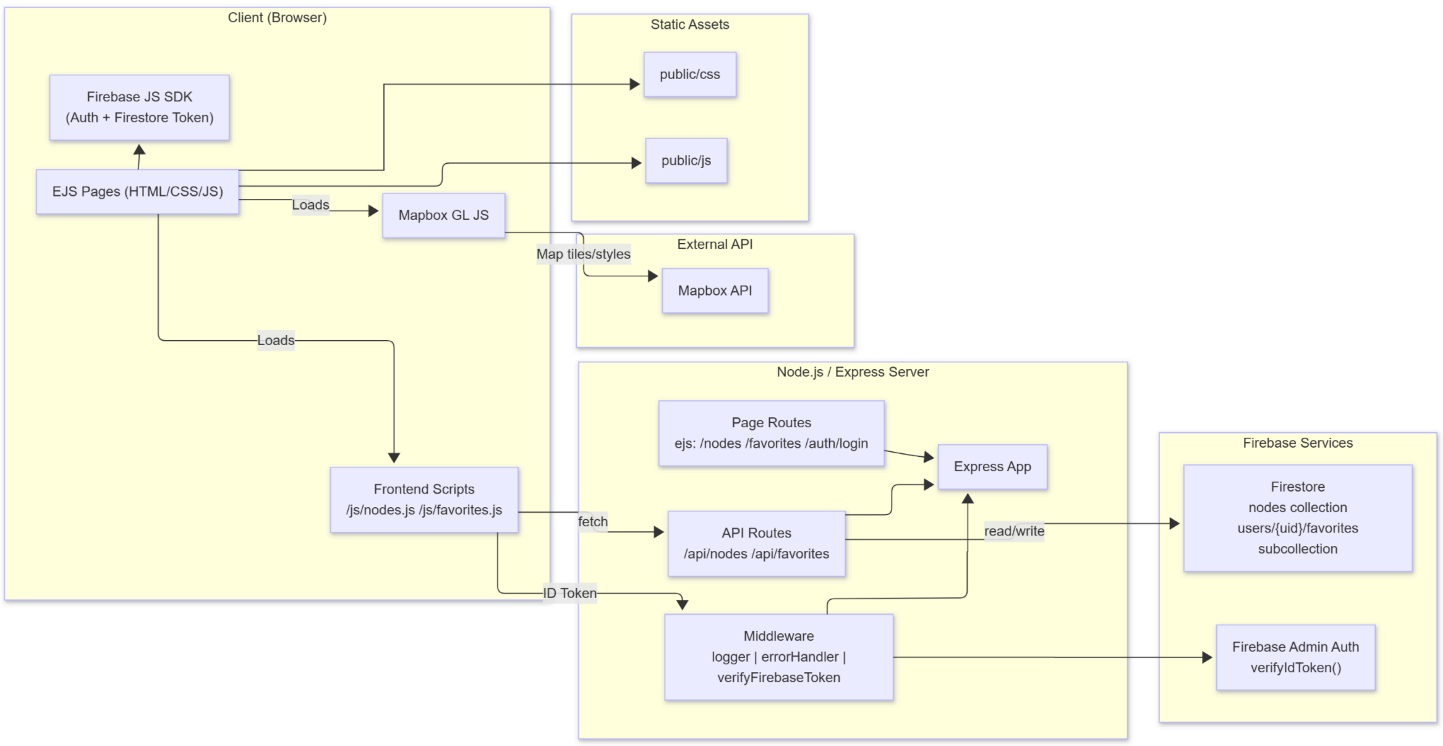
1. **Node.js + Express (server)**
    - **Why**: Quick to iterate, wide ecosystem, straightforward to serve EJS templates and REST APIs.
    - **Alternatives**: Django (Python), Rails (Ruby), Next.js (React SSR).
    - **Trade-offs**: Node.js is non-blocking and performant for I/O; however, using server-side rendered EJS limits heavy client-side single-page-app capabilities (but reduces complexity for MVP).
2. **EJS + Vanilla JS (frontend)**
    - **Why**: Fast to prototype, simple templating for pages, minimal build tooling.
    - **Alternatives**: React, Vue, Angular.
    - **Trade-offs**: Faster dev & simpler deployment for MVP; harder to build very dynamic client state compared to a SPA framework.
3. **Firestore + Firebase Auth**
    - **Why**: Persistent storage, real-time updates, managed auth.
    - **Alternatives**: MongoDB, PostgreSQL.
    - **Trade-offs**: Firestore reduces ops burden but introduces vendor lock-in and network latency.

4. **Mapbox GL JS**
   - **Why**: Provides interactive vector maps, custom styling, and good docs.
   - **Alternatives**: Google Maps, Leaflet.
   - **Trade-offs**: Mapbox requires an access token; Google maps may have different pricing and API differences. Mapbox works well for campus-level vector interaction.
5. **Logging & Error Handling**
   - **Why**: Winston + Morgan + Chalk provides structured logs, request logs, and colored console output.
   - **Alternatives**: Console.log only
   - **Trade-offs**: More maintainable and production-ready, slight setup overhead.

1.3 Constraints & Dependencies

- `Node.js ≥ 18, ESM enabled (type: module).`
- `Firebase SDK 10.x (browser + Admin).`
- `Mapbox API token in .env for local dev.`
- `Firebase service account credentials stored securely (never committed).`
- `Express session cookies require HTTPS in production.`
- `Browser clients need internet access for Mapbox tiles unless caching locally.`
- `REST API design with /api/nodes, /api/favorites, /auth.`

1.4 high-level architecture diagram


Client (Browser)
Firebase JS SDK (Auth + Firestore Token)
EJS Pages (HTML/CSS/JS)
Loads
Mapbox GL JS
Loads
Frontend Scripts /js/nodes.js /js/favorites.js
Static Assets
public/css
public/js
External API
Map tiles/styles
Mapbox API
Node.js / Express Server
Page Routes ejs: /nodes /favorites /auth/login
Express App
fetch
API Routes /api/nodes /api/favorites
read/write
ID Token
Middleware logger | errorHandler | verifyFirebaseToken
Firebase Services
Firestore nodes collection users/{uid}/favorites subcollection
Firebase Admin Auth verifyIdToken()

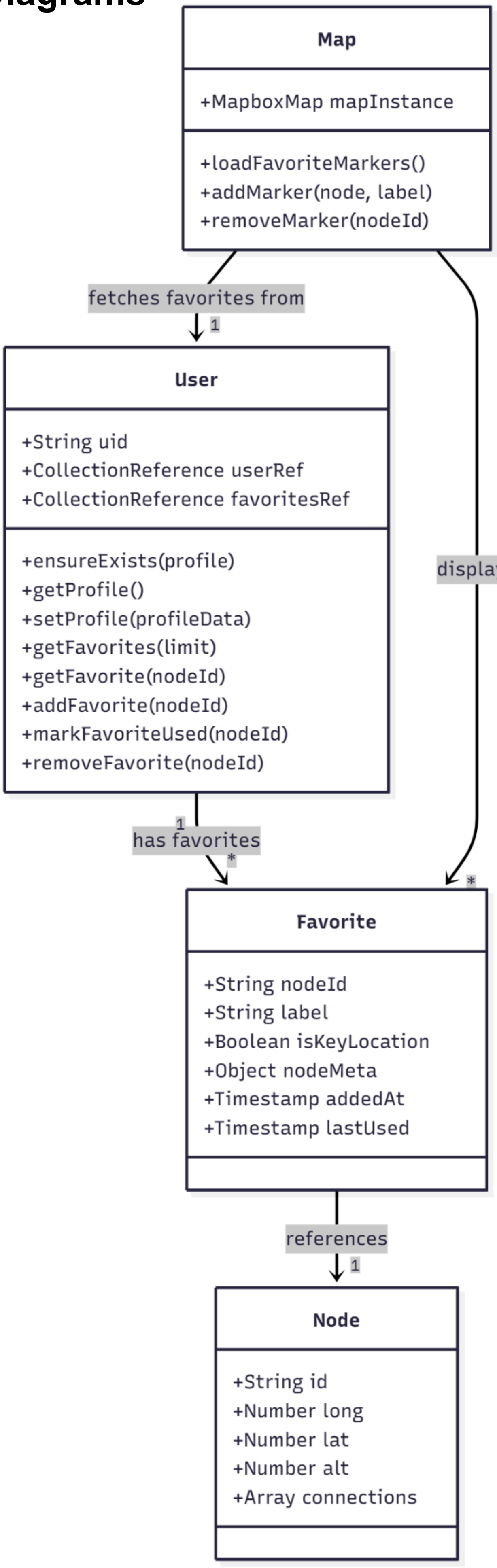
1.5 Architectural Patterns Chosen

- **Client-Server**: Clear separation of concerns; client handles UI and map, server handles business logic, REST APIs, and database interactions.
- **Layered Architecture** Presentation (EJS views) → Application (Express routes/controllers) → Data (Firestore).
- **Pluggable Middleware**: Logging (Morgan/Winston), error handling, and authentication middleware to keep routes lean.
- **REST API Design:** Modular routes for nodes, favorites, and auth, with standardized HTTP methods and JSON responses.

Why appropriate

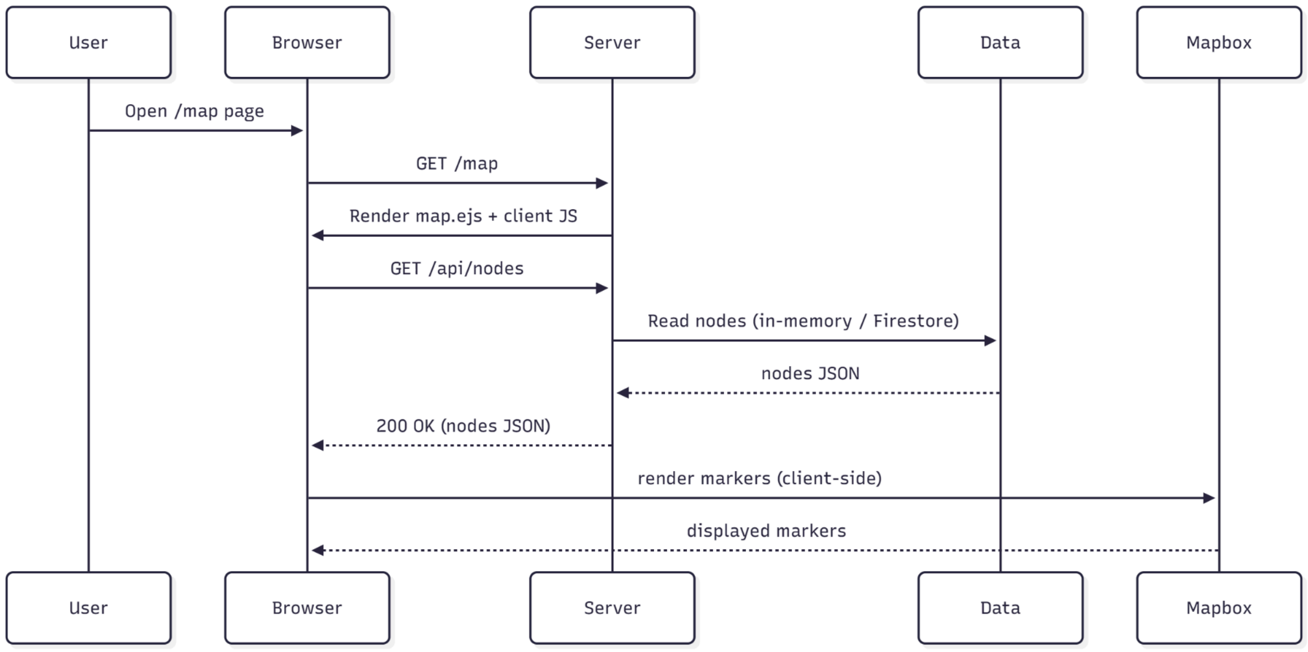
- Supports rapid prototyping for MVP.
- Easy to extend (replace in-memory with Firestore, add auth middleware).
- Aligns with standard web app practices and team collaboration patterns.

2. Class Diagrams

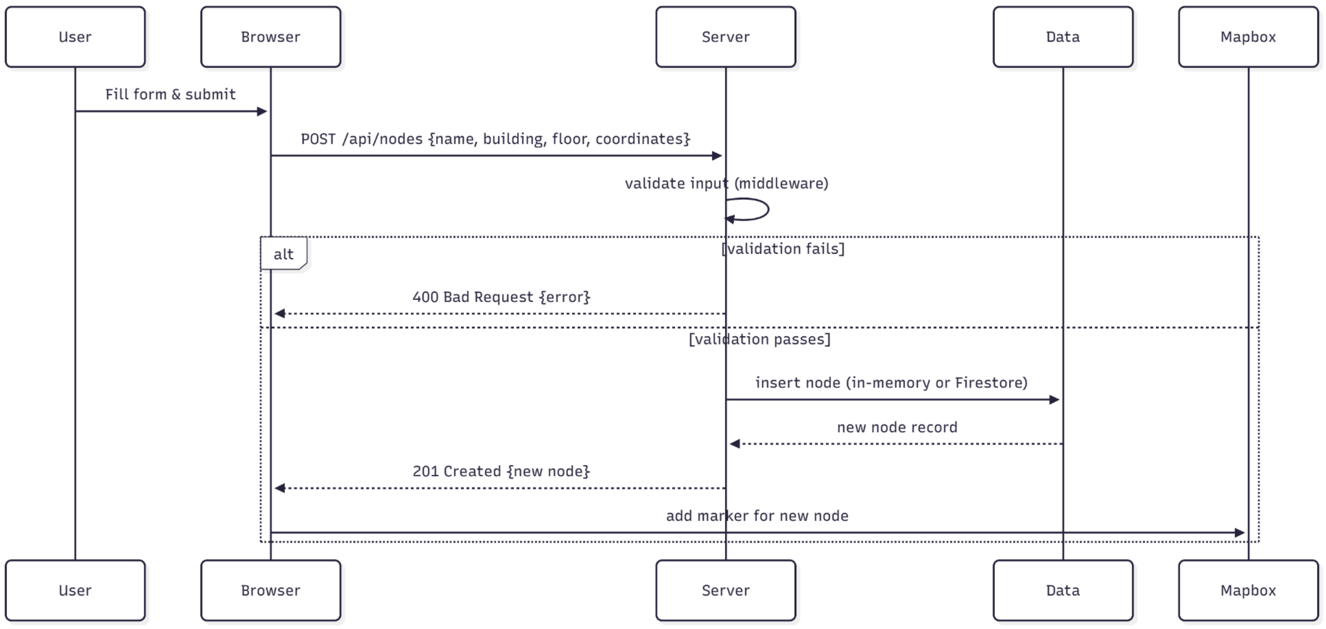


3. Sequence Diagrams

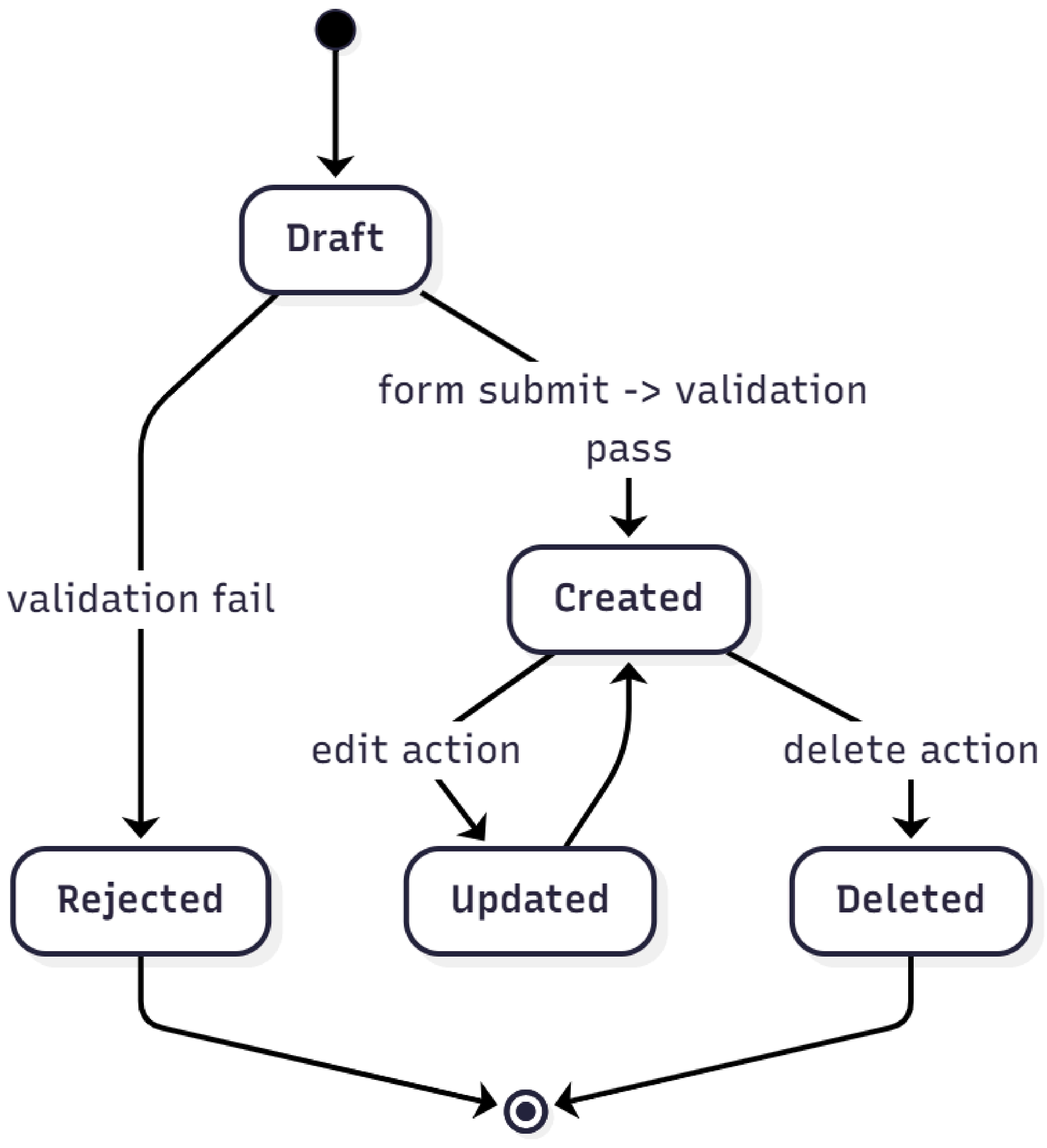
Use Case 1 — View Map & Load Nodes



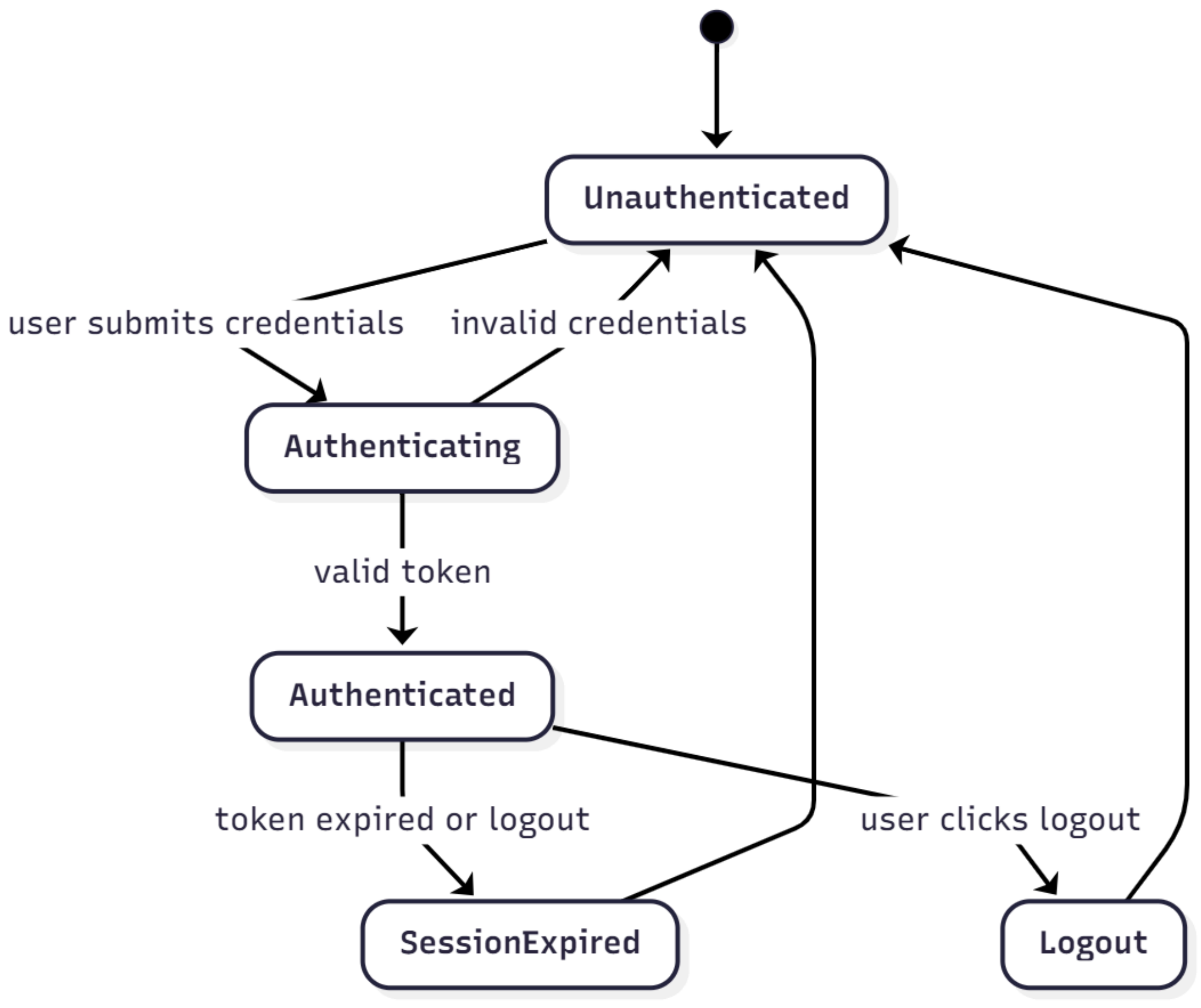
Use Case 2 — Create Node (via form on /nodes)



4. State Diagrams


Node Lifecycle

User Authentication State



5. Key Challenges

5.1 Data Availability Challenges

Challenge:

- Building data for BCIT buildings and rooms was very difficult to find. No known or public information on the coordinates of the BCIT building's rooms themselves.

Reflection:

- Received a GeoJSON file of the BCIT building coordinates (not the most recent data) from BCIT Geomatics.
- Took PDF files of the floorplans from BCIT website and preprocessed them into SVG files which I used the building GeoJSON file with a custom Python script and a custom EJS page to create an approximation of the coordinates for the rooms of the buildings

5.2 Firebase Auth and Express Sessions

Challenge:

- Getting the frontend Firebase ID token sent to the server and verified using Firebase Admin.

Reflection:

- Added proper token extraction from the Authorization header.
- Implemented checkSession middleware to verify ID tokens reliably.
- Updated frontend to consistently send the token with every request, stabilizing authentication flow.

5.3 Firestore Structure Decisions

Challenge:

- Designing the database structure, including using a top-level nodes collection and adding favoriteNodes as a subcollection under each user.

Reflection:

- Standardized structure after testing multiple models.
- Used subcollections to reduce read costs and isolate per-user data.
- Confirmed queries remained simple (/users/{uid}/favoriteNodes) and scalable.

5.4 Node-Based Navigation

Challenge:

- Image Model Navigation does not work at a campus level so we need another solution.

Looking at other map solutions like Google Maps we can use nodes!

Reflection:

- Nodes provide a clean solution for navigation
- Nodes are computationally heavy when iterating through all of them

5.5 Route Structure Cleanup

Challenge:

- Reducing duplicate routes and separating page routes (/nodes) from data routes (/api/nodes).

Reflection:

- Consolidated all REST API logic under /api/... to avoid collisions with EJS-rendered pages.
- Updated frontend fetch calls accordingly.
- Improved maintainability, removed redundant router functions, and clarified server structure.

5.6 New Technologies

Challenge:

- Required to use unfamiliar tools and technologies
- Steep learning curve and complex features
- Troubleshooting issues took extra time and research

Reflection:

- Difficult at first, but we stayed persistent
- Improved our ability to learn new technologies
- Gained confidence through practice and problem-solving

6. Lessons Learned

- How to create new GeoJSON data from existing GeoJSON data
- How to use Mapbox GL API
- Integrating Firebase Auth with a Node.js backend
- Designing scalable Firestore database schemas
- Debugging ES modules & ensuring correct script load order
- The importance of communication
- To properly merge and integrate code earlier
- Not everything can be properly automated

7. Improvements

- Add a schedule page for favorite nodes that could be set up as different classes of the student throughout the week.
- Add "shared favorites" between group, and BCIT events visible by all users
- Full single-page application (SPA) frontend instead of EJS pages
- Update the maps with BCIT's newer buildings (e.g. SW7 - Tall Timber Student Housing)
- Map the entire interior of the BCIT campus
- Improve accuracy of interior navigation