GatherAll - Technical Design Document

Table of Contents

Introduction and System Overview

Introduction

GatherAll is a modern Metaverse Platform designed to enable users to create virtual spaces, invite people and have fun in the Virtual World. The platform offers a seamless user experience with features like real-time updates, user authentication.

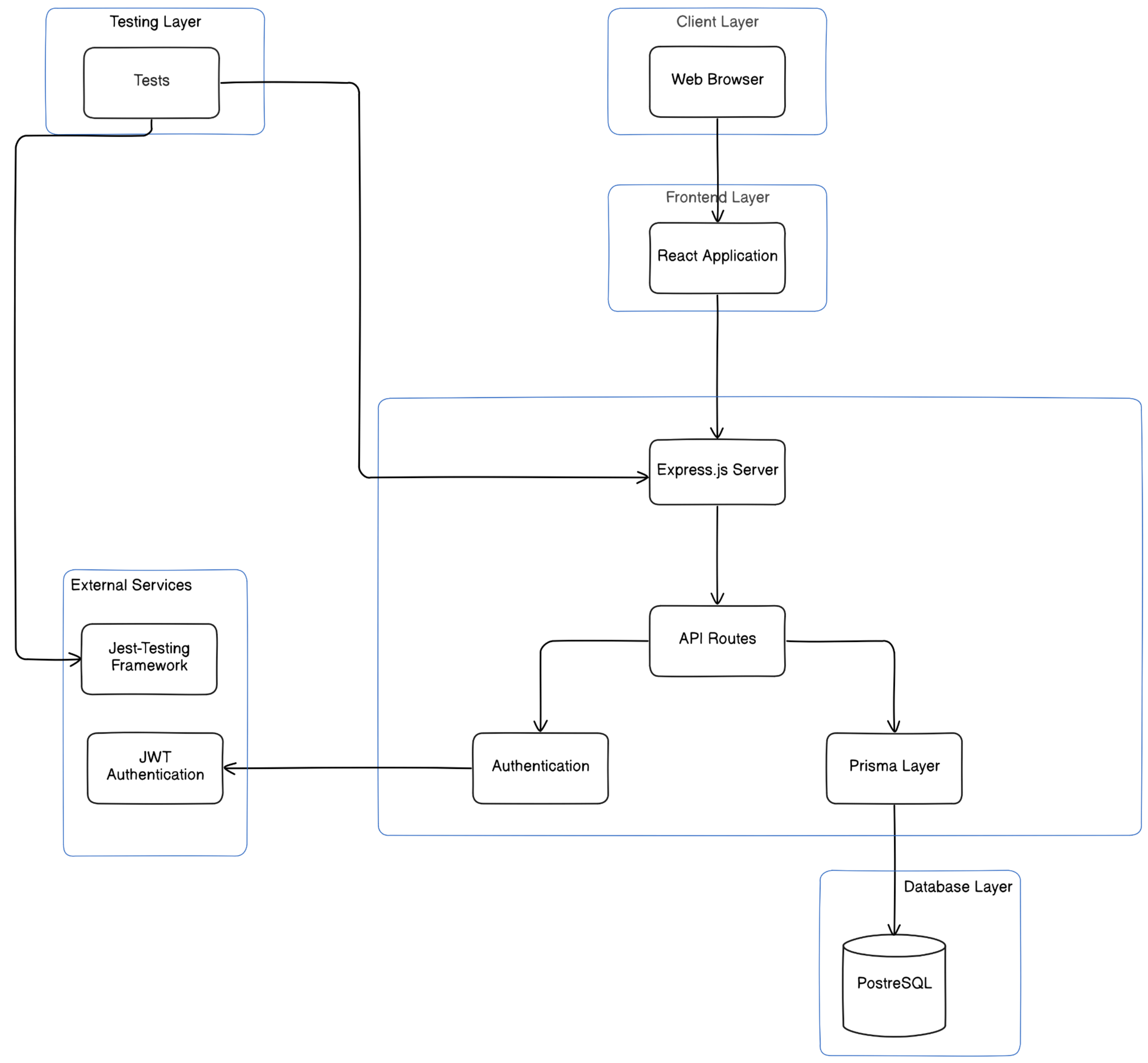
This design document provides a comprehensive overview of the project's architecture, components, technology stack, and design decisions. It aims to serve as a guide for developers, stakeholders, and contributors.

System Overview

Markd is built using the PERN stack (PostrgeSQL, Express.js, React, Node.js), utilizing modern web development practices. The application is structured to provide scalability, maintainability, and a responsive user experience across devices.

Architecture

High-Level Architecture



The system follows a three-tier architecture, comprising:

1. **Frontend**: Developed with React.js, responsible for the client-side user interface and interactions.
2. **Backend API**: Built with Express.js and Node.js, handling server-side logic, API endpoints, authentication, and business logic.
3. **Testing:** Developed using JS Testing library Jest.
4. **Database**: Utilizes PostgreSQL for storing user data and related information.

Backend Design

Technologies Used

- **Node.js**: JavaScript runtime environment.
- **Express.js**: Web application framework for building APIs.
- **PostgreSQL**: SQL database for data storage.
- **Prisma**: ORM (Object Relational Model) library for PostgreSQL.
- **Turbo:** Provides Monorepo template for projects
- **JWT**: JSON Web Tokens for authentication.

Project Structure

- **apps/http/src/index.js**: Entry point of the server application.
- **apps/http/src/middlewares/**: Contains middleware functions, including authentication.
- **packages/db/**:Prisma Package for Database access.
- **apps/http/src/routes/v1**: Defines Version 1 API endpoints for authentication, users, admin, space.
- **apps/http/scrypt.js**: Script for generating Hashing Passwords (Bcrypt was throwing some random errors).
- **apps/ws/:** Contains all the Web Socket Logic
- **packages/db:** All the db logic exported as a module and available for all app use

Backend Design - API, Database Schema and Middlewares

API Design

The backend exposes RESTful API endpoints categorized under:

- **User (`/api/v1`)**
  - `POST /signup`: User Signup.
  - `POST /signin`: User login and JWT token issuance.
  - `POST /user/metadata`: To retrieve User MetaData from the Database.
  - `GET /avatars`: Retrieve the Avatars.
  - `GET /user/metadata/bulK?ids=[x,y,z]`:Retrieve Metadata corresponding to multiple ids.
- **Space (`/api/v1/space`)**
  - `POST /`: Creates a new Space
  - `DELETE /:spaceId`: Deletes the space corresponding to spaceId
  - `GET /all`: Retrieve all existing spaces.
  - `GET /:spaceId` :Get space info corresponding to spaceId.
  - `POST /element`: Creates a new Element.
  - `DELETE /element`: Deletes the Element.
- **Admin (`/api/v1/admin`)**
  - `POST /element`: Creates a new Element
  - `PUT /element/:elementId` : Updates an Element
  - `POST /avatar`: Creates a new Avatar.
  - `GET /map` :Creates a new Map.

WebSocket Design

The backend exposes RESTful API endpoints categorized under:

- **Client Sent Events**
  - `Join A Space`
  - `Move within a Space`
- **Server Sent Events**
  - `Space Joined`
  - `Movement Rejected`
  - `Move`
  - `Leave`
  - `Join Event`

Database Schema

User Model

Fields:

- **username**
- **password**
- **avatarId**
- **role**

Space Model

Fields:

- **name**
- **width**
- **height**
- **thumbnail**

---

spaceElements Model

Fields:

- **elementId**
- **spaceId**
- **x**
- **y**

---

spaceElements Model

Fields:

- **elementId**
- **spaceId**
- **x**
- **y**

Element Model

Fields:

- **width**
- **height**
- **imageUrl**

Map Model

Fields:

- **width**
- **height**
- **name**

---

mapElements Model

Fields:

- **mapId**
- **elementId**
- **x**
- **y**

Avatar Model

Fields:

- **imageUrl**
- **name**

Middlewares

Authentication Middleware

- Validates JWT tokens sent in the `Authorization` header.
- Attaches the authenticated user's information to the request object.
- Protects routes that require authentication.

Frontend Design

Technologies Used

- **React.js**: JavaScript library for building user interfaces.
- **React Router DOM**: Handling client-side routing.
- **Tailwind CSS**: Utility-first CSS framework for styling.
- **Vite**: Build tool for faster development.
- **ESLint**: Linting utility to maintain code quality.

Project Structure

- **main.jsx**: Entry point of the React application.
- **App.jsx**: Main application component.
- **app.css**: Global CSS and Tailwind directives.
- **Game.jsx:** Game logic in totality

Frontend Design - Routing, State Management and Key Components

---

Routing

Implemented using React Router:

- `/`: Landing Page

State Management

- **Data Fetching**:
  - Utilizes `fetch` API.
  - Handles loading and error states.

Key Components

Game Screen

- Main Game screen showing the player token and rooml

Security Considerations

Authentication

- **JWT Tokens**:
  - Securely generated and signed with a secret key.
  - Stored in the client's `localStorage`.
- **Password Security**:
  - Passwords hashed using `scrypt.js` before storing in the database.
  - Plain passwords are never stored or logged.

Authorization

- **Protected Routes**:
  - Backend routes require valid JWT tokens.
  - Frontend routes use higher-order components to restrict access.

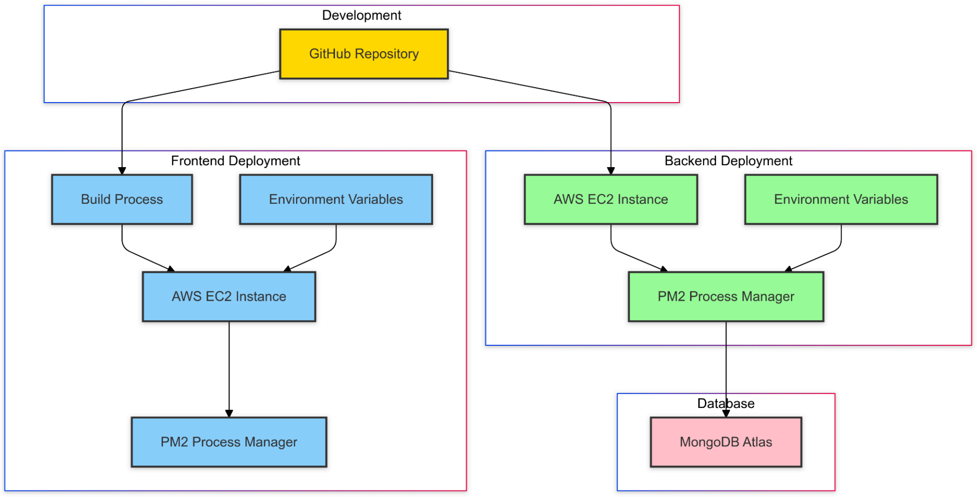
Deployment Plans

Environment Setup

- **Backend Environment Variables**:
  - `JWT_SECRET and JWT_PASSWORD`: Secret key for signing JWTs.
- **Frontend Environment Variables**:
  - `VITE_WS_URL`: Base URL for the WS Server.

Deployment Steps

1. **Backend Deployment**:
   - Host on platforms like Heroku, AWS EC2, or DigitalOcean.
   - Ensure environment variables are securely set.
2. **Frontend Deployment**:
   - Build the React application using `npm run build`.
   - Host static files on services like Vercel or AWS.
3. **Domain and SSL**:
   - Configure a custom domain.
   - Set up SSL certificates for secure HTTPS communication.



Future Enhancements

Technical Improvements

- **Switch to TypeScript**:
  - Introduce TypeScript for type safety and better maintainability.
- **State Management Library**:
  - Implement Redux, Context API, Zustang for more complex state needs.
- **Better UI:**
  - Implement Better UI for the users to enjoy and interact.

Feature Enhancements

- **Voice Chat**:
  - Allow users to voice chat with players in the same room.
- **Video Chat**:
  - Implement a social feature where users can Video Chat each other.
- **Notifications**:
  - Real-time notifications for interactions.
- **Search Functionality**:
  - Implement search to find public spaces by title, users, or tags.
- **Analytics Dashboard**:
  - Provide users with insights on Room info, visits and interactions.

Conclusion

The GatherAll project showcases an emerging concept in virtual environments, focusing on creating an interactive, immersive experience. It integrates JavaScript for functionality, with a clear structure for future development. As the project evolves, further features and improvements could be added, expanding its potential applications in virtual spaces.