Final Project Report

Course: Software Design Assignment
Project Title: Community Event System (Monolith Architecture)
Developers: Davies, Tatenda and Amos
Submission Date: October 25, 2025

---

1. Introduction

The **Community Event System** is a monolithic web application designed to manage event creation, user authentication, role-based access, and real-time RSVP updates. The project was developed using **Elysia.js**, **Prisma ORM**, and a **PostgreSQL database hosted on Neon**. It implements modern software design principles such as modularity, separation of concerns, and scalability while ensuring practical functionality and deployment readiness.

The system allows users to:

- Register and log in securely.
- Create, update, delete, and approve events (based on role).
- RSVP to events and receive live updates through websockets.
- View API documentation via Swagger.
- Access the app live on Render.

---

2. Tools and Technologies

Category	Tool/Technology	Purpose
Backend Framework	**Elysia.js**	API development, routing, and websocket handling
ORM	**Prisma**	Database modeling and queries
Database	**Neon (PostgreSQL)**	Cloud-hosted database
Authentication	**JWT + bcrypt**	Secure user authentication
API Documentation	**@elysiajs/swagger**	Automatic API documentation
Deployment	**Render**	Hosting the final app
Testing	**Insomnia**	Endpoint and realtime testing

---

3. System Architecture

The system follows a **monolith structure** with the following modules:

- **Controllers:** Business logic (auth, events, RSVP)
- **Routes:** Endpoints for HTTP requests
- **Middleware:** Authentication and role-based access control

- **Utils:** Helper functions (JWT, error handling)
- **Prisma:** Database schema and client
- **Index:** Main server configuration and route initialization

This design ensures clean separation of concerns, easy debugging, and maintainability.

---

4. Challenges Faced

a) Database Downtime

Initially, the project used an **AWS-hosted database**, which went down unexpectedly. We migrated to **Neon (PostgreSQL)** and updated the `DATABASE_URL` to restore service. Prisma migrations had to be reconfigured and synchronized using `npx prisma migrate reset`.

b) Node.js Compatibility

During development, we encountered multiple errors due to Node version incompatibility. The Prisma and Elysia versions required **Node.js 20+**, while the local environment was running **Node 18**. We upgraded to Node 20 using `n` and adjusted TypeScript configurations.

c) TypeScript ESM vs CommonJS Errors

Type import/export syntax caused module resolution conflicts. This was resolved by setting `"type": "module"` in `package.json` and adjusting `tsconfig.json` to `"module": "ESNext"` with `"moduleResolution": "node"`.

d) Prisma Drift Errors

After multiple schema updates, Prisma reported “drift detected” issues. We reset the migration history using:

```
npx prisma migrate reset
npx prisma migrate dev --name update-schema
```

This ensured full synchronization with the Neon database.

e) Deployment Configuration

During deployment to **Render**, environment variables were missing at first (e.g., `DATABASE_URL`, `JWT_SECRET`). After adding them in Render’s dashboard, the system deployed successfully and API endpoints worked perfectly in production.

---

5. Testing

All API endpoints were tested with **Insomnia**, verifying:

- Authentication (Signup/Login)

- Event CRUD operations
- Role-based authorization
- RSVP operations
- Realtime updates (via WebSockets)
- Swagger API documentation accessibility at `/swagger`

All tests passed successfully on both local and deployed environments.

---

6. Results and Achievements

-Successful backend deployment on Render
-Full database integration with Neon
-Secure authentication and JWT system
-Role-based control (Admin, Organizer, Attendee)
-Realtime RSVP and event updates
-Fully documented API with Swagger
-Comprehensive local and cloud testing

---

7. Lessons Learned

- Maintaining version compatibility between Node, TypeScript, and Prisma is crucial.
- Modular folder structure simplifies debugging and enhances scalability.
- Cloud databases like Neon improve flexibility but require careful environment management.
- AI-assisted development (via ChatGPT) greatly accelerates debugging and implementation.
- Continuous testing with Insomnia ensures stable deployments.

---

8. Conclusion

The **Community Event System** successfully demonstrates the integration of authentication, event management, and realtime functionality within a clean monolithic architecture. Despite several technical hurdles, the team overcame challenges through persistence, documentation reference, and strategic tool usage. The final system is live, tested, and production-ready.

---

9. References

- Elysia.js Documentation
- Prisma Documentation

- Neon PostgreSQL
- Render Deployment Guide
- JWT & Bcrypt Libraries
- Insomnia API Testing

Final Note:
This project solidified our understanding of software design principles, modern backend development, and cloud deployment pipelines. The challenges faced became powerful learning experiences that strengthened our development workflow.