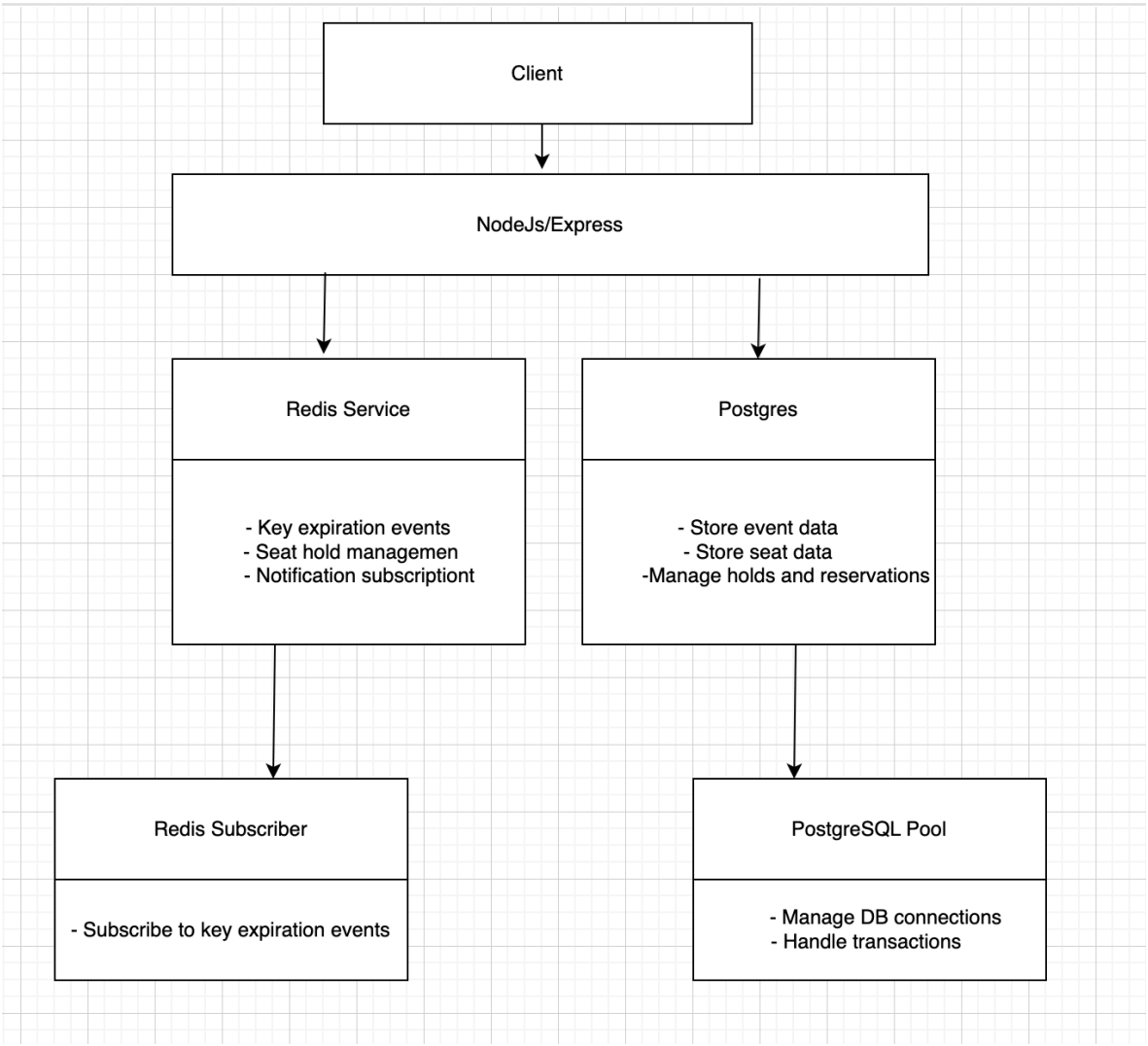
Event Seat Reservation Service

The Event Seat Reservation Service is designed to handle the creation of events, management of seat reservations, and control for seat holds using Redis and PostgreSQL.
This service is containerised using Docker

Architecture

The service follows a micro-services architecture with the following components:
- Redis: For managing seat hold expiration and notifications.
- PostgreSQL (PostGIS): For storing event and seat information.
- Node.js/Express Application: For handling API requests and business logic.



Components

Docker Setup

Defines services for Redis, PostgreSQL (PostGIS), and the main Node.js application using Docker Compose.

Server Initialisation

The server initialisation involves starting the Express application, setting up Redis subscriptions, and handling graceful shutdown processes.
This ensures that the application starts correctly and can shut down gracefully when needed.

App Configuration

The Express application is configured with necessary middleware . The main routes are set up to handle API requests related to event and seat management.

Redis Configuration

Redis is configured to handle key expiration events, which are crucial for managing seat holds. This setup ensures that expired holds are automatically handled, freeing up seats for other users.

Database Interaction

PostgreSQL is used to store all event and seat-related data. The database interaction layer includes:
- Connection management and pooling
- SQL queries for creating events, holding seats, reserving seats, and listing available seats

Business Logic

The business logic layer handles the core functionality of the application. This includes:
- Creating events and seats
- Holding seats with concurrency control
- Reserving seats based on active holds
- Refreshing holds to extend the hold duration
- Listing available seats for an event

API Routes

API routes are defined to handle various operations:
- Creating events
- Holding seats
- Reserving seats
- Refreshing holds
- Listing available seats

Database Schema

The PostgreSQL database schema includes the following tables:
events : Stores event information.
seats: Stores seat information for each event.
holds: Manages temporary holds on seats by users.
reservations: Stores confirmed seat reservations.

Environment Variables

Key environment variables include:
- `PORT`: The port on which the Node.js server runs.
- `POSTGRES_HOST`, `POSTGRES_USER`, `POSTGRES_PASSWORD`, `POSTGRES_DB`, `POSTGRES_PORT`: Database connection details.
- `REDIS_HOST`, `REDIS_PORT`: Redis connection details.
- `DEFAULT_HOLD_DURATION`: Default duration for holding a seat.
- `MAX_SEATS_PER_USER`: Maximum number of seats a user can hold for an event.

Limiting Holds for Events

There is a limit on the number of seats a user can hold for an event. This is implemented using Redis to track the number of holds per user for each event.

Process for Holding a Seat

1. Check Redis for Existing Holds:
   - Check if the seat is already held.
   - Retrieve the count of holds by the user for the event.
2. Validate Hold Limits:
   - If the seat is held, throw an error.
   - If the user has reached the maximum holds limit, throw an error.

Cron Job for clearing hold data

A cron job is scheduled in the PostgreSQL database to periodically clear expired holds. This ensures that the database remains clean and no stale hold data persists,

Cron Job Details:

- The cron job runs every hour.
- It deletes all holds from the holds table where the expires_at timestamp is in the past.

Implementation:

- A PostgreSQL extension, pg_cron, is used to schedule the job.

- The cron job is defined in the init.sql script, which is executed when the PostgreSQL container is initialized.

Testing

Testing includes unit tests using Jest
- Cover all API routes