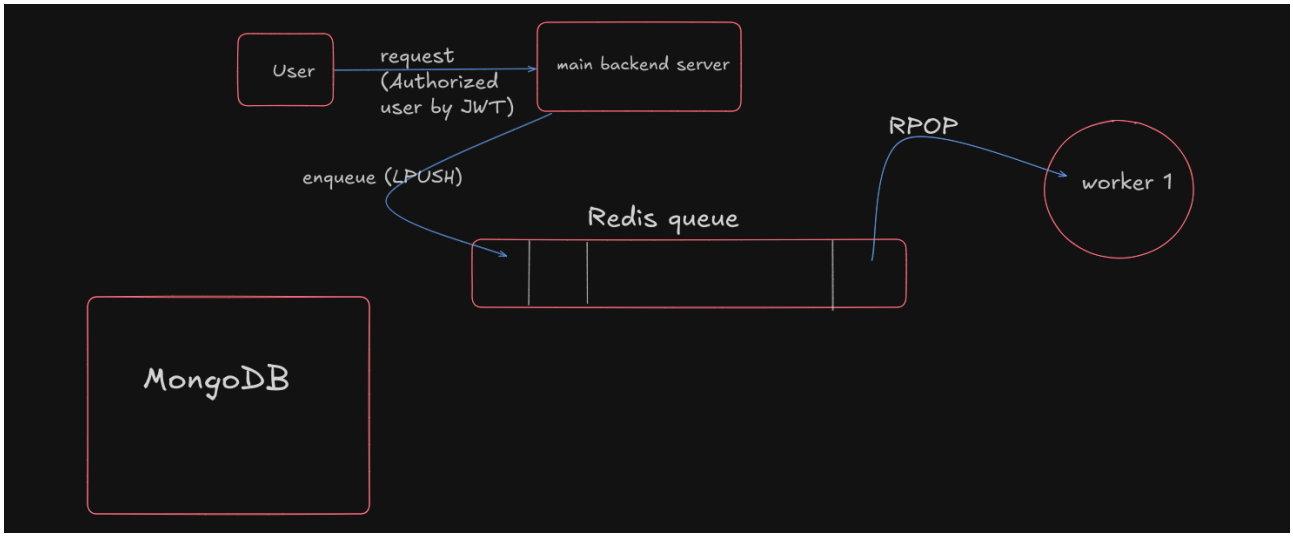
System Design:



Client-Server Model

The system implements a traditional client-server architecture with the following components:

- **User Client**: Initiates requests to the main backend server
- **Main Backend Server**: Handles incoming client requests and authentication
- **Redis Queue**: Acts as the message broker between server and workers
- **Worker Process**: Processes requests from the queue

Queue Management

The system utilizes Redis as the queue management system with the following workflow:

- **Queue Operations**:
  - LPUSH: Server enqueues requests to Redis queue
  - RPOP: Worker dequeues requests for processing
- **Request Flow**:
  - Requests are stored in Redis in FIFO (First-In-First-Out) order
  - Each request maintains its order of arrival
  - Workers process requests sequentially to ensure ordered execution

Worker Architecture

The worker implementation follows these principles:

- **Single Worker Model**: Dedicated worker process that continuously polls the Redis queue
- **Sequential Processing**: Requests are processed one at a time in the order they were received
- **Asynchronous Operation**: Worker operates independently of the main server
- **Reliable Processing**: Uses RPOP to ensure each request is processed exactly once

This architecture provides:

- Scalability through decoupled components
- Reliability through persistent queue storage
- Ordered processing of requests
- Efficient resource utilization
- Easy monitoring and maintenance

The system can be extended by adding more workers or implementing additional queue features as needed.

Backend Tech Stack

- **TypeScript**: Used as the primary programming language, providing type safety and better development experience
- **Node.js**: Runtime environment for executing JavaScript/TypeScript code
- **Express**: Web framework for handling HTTP requests, routing, and middleware implementation
- **MongoDB**: Primary database for storing user information, connected via `db.ts`
- **Redis**: Implements message queue functionality for request processing
    - Uses LPUSH for enqueueing requests from main server
    - Uses RPOP for dequeuing requests by worker process

Authentication & Security

- **bcrypt**: Implements password hashing in `passwordHashing.ts` for secure user credential storage
- **JWT**: Handles user authentication tokens in `jwtAuth.ts` middleware
- **Zod**: Implements request validation schemas in `userSchema.ts` for input validation

Containerization & Services

- **Docker**: Containerizes different components of the application:
  - Main backend server
  - Redis queue
  - Worker process
  - Prometheus
  - Grafana

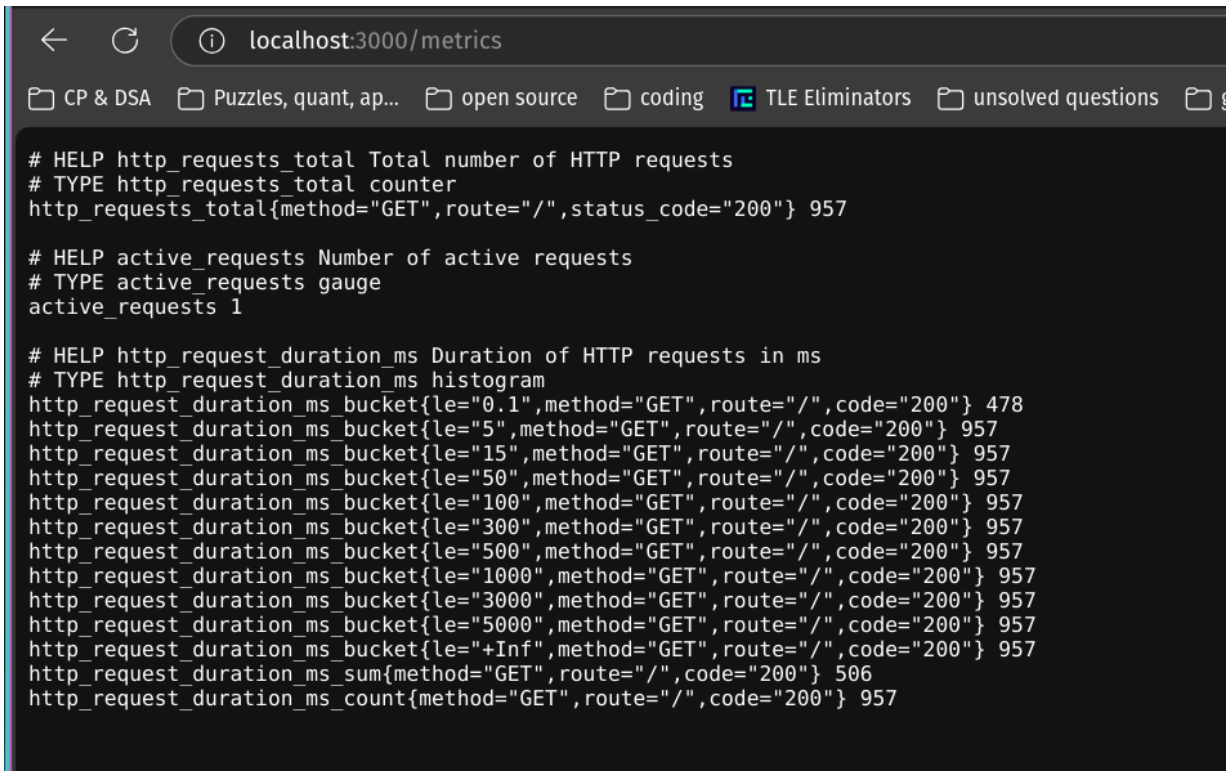
Monitoring & Metrics

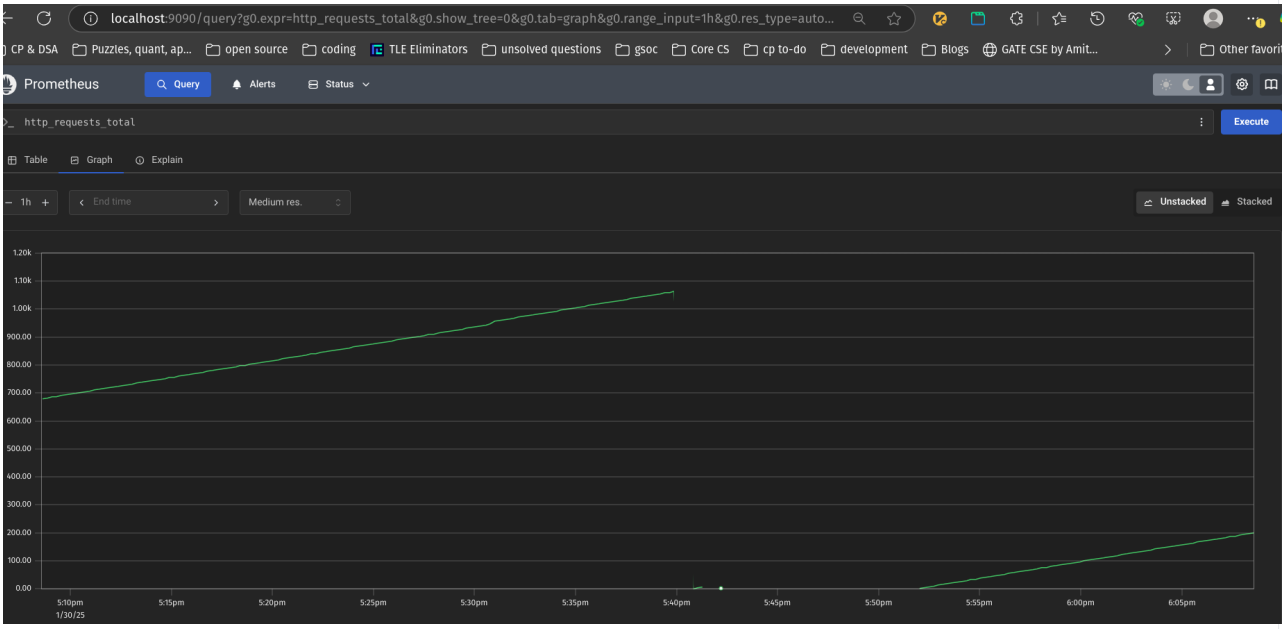
- **Prometheus**: Collects metrics from the application:
  - Active requests tracking
  - Request count monitoring
  - Request time measurements
- **Grafana**: Visualizes metrics collected by Prometheus

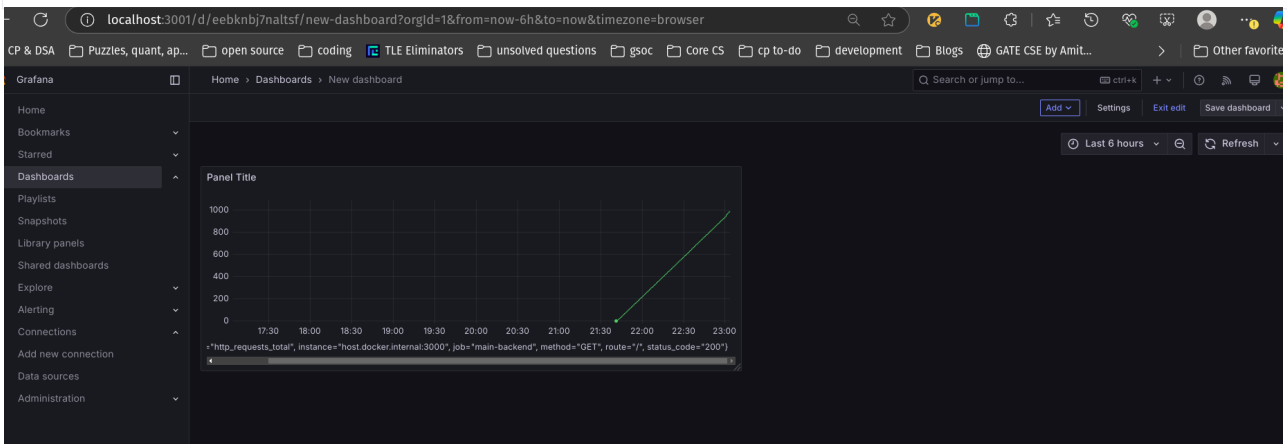
Project Structure

- `/src`: Contains main application code
- `/middlewares`: Authentication and validation middleware
- `/monitoring`: Prometheus metrics configuration
- `/routes`: API endpoint definitions
- `/schemas`: Data validation schemas
- `/tests`: Unit and integration tests

Systems monitoring and logging







Project Setup Guide

1. Redis Setup

Start Redis container:

```
bash
docker run --name my_redis -d -p 6379:6379 redis
```

2. Main Backend Setup

Navigate to main backend directory and install dependencies:

```
bash
cd main-backend
npm install
```

Run unit tests:

```
bash
npm test
```

Start the main backend server:

```
bash
npm run start
```

3. Worker Setup

Navigate to worker directory and install dependencies:

```
bash
cd workers
npm install
```

Start the worker:

```
bash
npm run start
```

4. Prometheus Setup

Start Prometheus container with host access:

```
bash
docker run -p 9090:9090 \
  -v $(pwd)/prometheus.yml:/etc/prometheus/prometheus.yml \
```

```
  --add-host=host.docker.internal:host-gateway \
  prom/prometheus
```

5. Grafana Setup

Start Grafana container:

```
bash
docker run -d -p 3001:3000 --add-host=host.docker.internal:host-gateway --
name=grafana grafana/grafana:latest
```

6. Grafana Configuration

1. Access Grafana at http://localhost:3001
2. Login with credentials:
    - Username: admin
    - Password: admin
3. Add Prometheus data source:
    - Navigate to Configuration → Data Sources
    - Add new data source
    - Select Prometheus
    - Set URL to: http://host.docker.internal:9090
    - Save & Test