Report Assignment 2: Microservices

Yonah Thienpont

May 23, 2024

1 Microservices Decomposition

The application is decomposed into the following microservices:

- **Authentication Service**
- **Event Service**
- **Calendar Service**

Each microservice has their own database called 'auth-db', 'event-db' and 'cal-db', respectively.

1.1 Authentication Service

- **Features**: User registration, login, and user information retrieval.
- **Data**: Stores user credentials and profile information.
- **Connections**: Returns whether a user exists to the calendar services, when we try to share a calendar with a non-existing user.

1.2 Event Service

- **Features**: Create events, view event details, list public events, manage RSVPs, and fetch user-specific events.
- **Data**: Stores event details and RSVP statuses.
- **Connections**: Retrieves user information from the Authentication Service and integrates with the Calendar Service to display events.

1.3 Calendar Service

- **Features**: Share calendar, view shared calendars, and list user-specific events.
- **Data**: Manages shared calendar relationships.
- **Connections**: Interacts with the Authentication Service to verify users and with the Event Service to fetch event data.

2 Explanation of Decomposition

2.1 Grouping of Features

Features were grouped based on their core functionalities:

- **Authentication**: Handles user-related operations like registration and login.
- **Event Management**: Manages event creation, RSVPs, and event visibility.
- **Calendar Management**: Deals with personal and shared calendars.

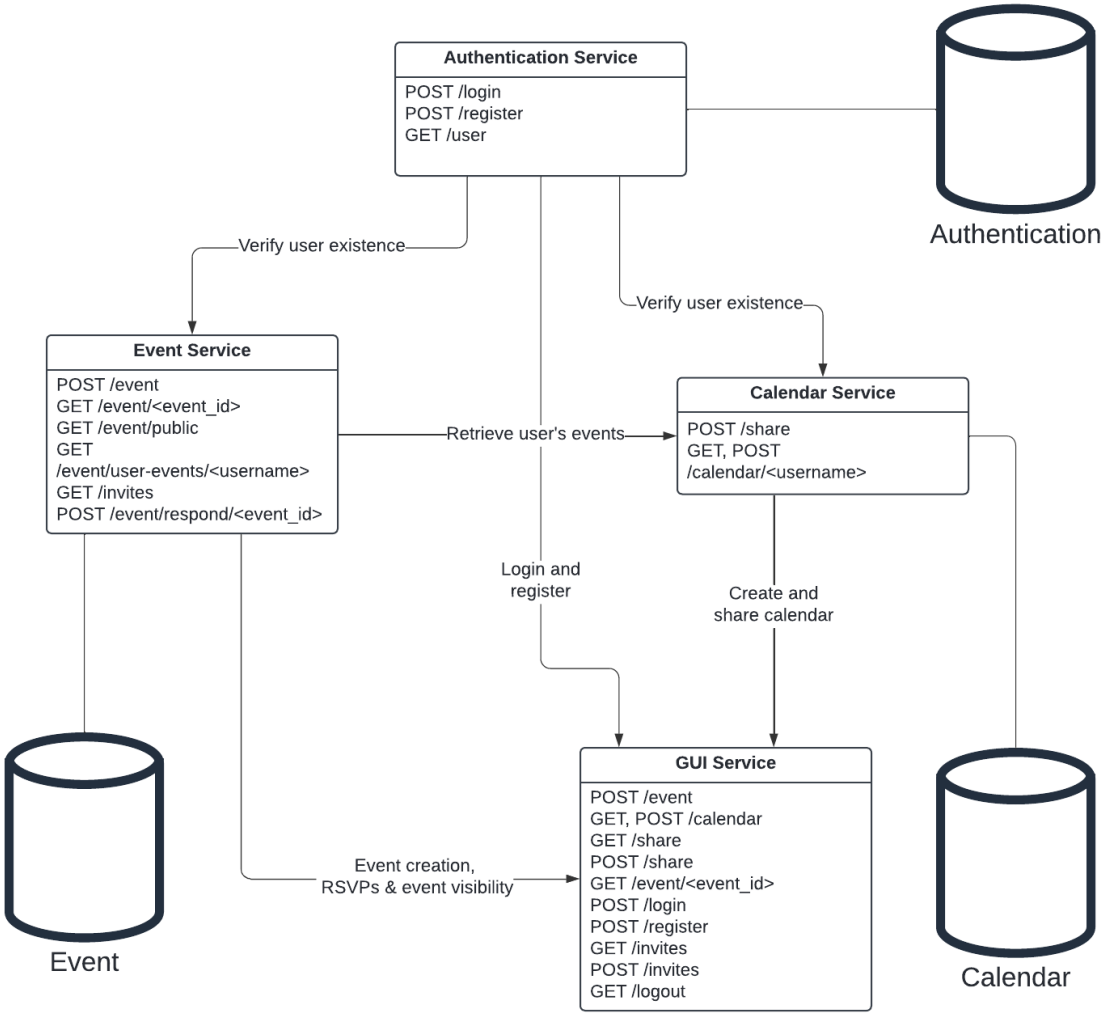

Figure 1: Overview of the Microservice Architecture

2.2 Consequences of Service Failures

- **Authentication Service Failure**: Users cannot log in or register, affecting overall access.
- **Event Service Failure**: Users cannot create or respond to events, affecting event-related activities.
- **Calendar Service Failure**: Users cannot view or share calendars, affecting calendar visibility.

2.3 Scalability

The approach allows each service to scale independently based on demand:

- **Authentication Service**: Can be scaled to handle high login/register traffic.
- **Event Service**: Can be scaled to manage numerous event-related operations.
- **Calendar Service**: Can be scaled to support extensive calendar interactions.

3 Bugs

When responding to an event, you will have to refresh the page manually to see the change take effect. The API calls and their statuses are correct.

4 Implementation

Below are the API endpoints for each required feature:

4.1 Authentication Service

- **Login**

  ```
  POST /login
  ```

- **Register**

  ```
  POST /register
  ```

- **Get User**

  ```
  GET /user/{username}
  ```

4.2 Event Service

- **Create Event**

  ```
  POST /event
  ```

- **View Event**

  ```
  GET /event/{event_id}
  ```

- **Public Events**

  ```
  GET /event/public
  ```

- **User Events**

  ```
  GET /event/user-events/{username}
  ```

- **Pending Invites**

  ```
  POST /invites
  ```

- **Respond to Invite**

  ```
  POST /event/respond/{event_id}
  ```

4.3 Calendar Service

- **Share Calendar**

  ```
  POST /share
  ```

- **View Calendar**

  ```
  GET /calendar/{username}
  POST /calendar/{username}
  ```