SERVICE ORIENTED TRIP PLANNER

Group 10

ABSTRACT

This project develops a comprehensive trip planning and reminder system that seamlessly integrates a user-friendly calendar interface with real-time notifications and authentication services. Employing RESTful APIs and message queuing, it offers efficient trip scheduling, timely reminders, and secure user management.

David Galati, Fil Skulimowski

Service Oriented Architecture

University of Twente

Trip Planner Application

Contents

1. Motivation 2
2. Business processes: 2
3. Architecture: 2
Calendar Service (Trip Planner) 2
Frontend: 2
Backend: 2
Database (Calendar DB): 3
Reminder Service 3
Main Functionality: 3
Database (Reminder DB): 3
Alert/Notification Service 3
Purpose 3
Functionality: 3
Authentication Service 3
Purpose 3
JWT Management 3
User Database Interaction 3
Axios Client Setup on React Frontend 3
Route Service 4
Purpose 4
Functionality: 4
Communication and Data Flow: 4
Deployment: 5
4. Design Decisions: 5
Justification of Choices 7
5. Validation: 7
Unit Testing 7
User Testing 8
Happy flow showcasing all the features of the app 9
6. Conclusion 10
Appendix 11

1. Motivation

In today's fast-paced world, efficient management of time and resources is essential. Many individuals struggle with planning and managing their daily commutes and trips effectively. The absence of a comprehensive solution that integrates trip planning, reminder services, and instant alerts contributes to time mismanagement and increased stress levels. Moreover, the security concerns around user authentication and data privacy further complicate the adoption of a unified platform that can handle these tasks securely and efficiently.

The primary objective of this project was to develop a robust, user-friendly application that simplifies the process of planning and managing trips with enhanced features such as reminders and real-time alerts. This system not only aims to improve individual time management and productivity but also ensures that all interactions are secure, leveraging modern authentication mechanisms.

2. Business processes:

At its core, our system facilitates users in creating accounts, finding routes, planning trips, managing schedules via a calendar service, and receiving timely reminders through an alert service. The calendar feature enables users to organize their trips effectively, while the reminder service ensures they stay informed about upcoming events. Additionally, the platform prioritizes data security to foster user trust and confidence.

3. Architecture:

Calendar Service (Trip Planner)

The Calendar Service acts as a trip planner and schedule manager. It allows users to plan, view, and manage their trips through a calendar interface. Users can input trip details such as start and end locations, start times, and durations, and see these details reflected on their personal calendars.

Frontend:

- **Calendar Interface**: A user-friendly interface where users can interact with a visual calendar to plan and view trips.
- **Trip Form**: A form for entering trip details which, upon submission, sends data to the backend for processing.
- **Google Maps Integration**: Allows users to view the route of a planned trip using Google Maps APIs.

Backend:

- Manages trip data and interacts with the frontend to reflect user actions such as creating or viewing trips.
- Receives trip data from the frontend and stores it in the Calendar Database.
- Serves trip information back to the frontend for display on the calendar and mapping functionality.

Database (Calendar DB):

- Stores trip details including **tripId**, **userId**, **start_position**, **end_position**, **start_time**, and **trip_duration**.

Reminder Service

Purpose: The Reminder Service is responsible for managing reminders for trips. It takes trip reminder requests from the Calendar Service and schedules reminders accordingly.

Main Functionality:

- Processes incoming POST requests to set reminders for trips, which include details such as trip ID and reminder time.
- Utilizes a scheduler to manage and send reminders at the specified times before the trip.

Database (Reminder DB):

- Maintains records of all reminders set by users, with fields such as **reminder_id**, **reminder_time_before_trip**, **trip_id**, and **trip_time**.

Alert/Notification Service

Purpose

The Alert Service is a consumer of reminder notifications from the Reminder Service. It listens for messages on the ActiveMQ queue and triggers alerts to notify users.

Functionality:

- Consumes messages from the ActiveMQ queue, which represent reminders that are due to be sent to users.
- Notifies users through the frontend interface, ensuring that reminders are delivered even if a user's browser is not currently open.

Authentication Service

Purpose

This service provides authentication functionality for the application. It manages user credentials and session tokens, ensuring secure access to the app's features.

JWT Management

- **Token Creation**: Generates secure JWTs using HMAC256 encryption, embedding user-specific details.
- **Token Validation**: Ensures tokens are valid, unexpired, and correctly signed, enhancing security by preventing unauthorized access.

User Database Interaction

- **Credentials Verification**: Authenticates user logins by verifying submitted credentials against the user database.
- **Registration Handling**: Manages user registrations by ensuring data integrity and storing hashed passwords securely.

Axios Client Setup on React Frontend

When a user fills out the **login** or **registration forms** and submits them, Axios sends these details as POST requests to the backend endpoints /login and /register.

- **Secure API Communication**: Configures HTTP requests to include JWTs in headers, ensuring all communications with backend services are authenticated.
- **Token Storage and Management**: Handles token storage in local storage and dynamically appends tokens to HTTP headers for outgoing requests.

Route Service

Purpose

Provides dynamic route planning and navigation features within the application, leveraging Google Maps APIs to deliver real-time location-based services to users.

Functionality:

- **Autocomplete Inputs**: Integrates Google's Place Autocomplete to provide suggestions as users type, streamlining the input of geographical locations.
- **Routes API:** Offers zooming, panning, and detailed route displays, enhancing user engagement and providing comprehensive navigational data.

Communication and Data Flow:

1. **Authentication:**
    - Users register an account or login, Axios sends a POST request to /login or /register, upon submitting user data is stored in Java Spring database.
    - The Authentication Service verifies user credentials and issues JWTs for session management.
2. **Trip Planning:**
    - Authenticated users plan trips using the Calendar Frontend's form.
    - The trip data is sent to the Calendar Backend, which stores it in the Calendar DB.
3. **Reminder Setting:**
    - Users set reminders for trips using the Calendar Frontend.
    - The Calendar Backend sends the reminder details to the Reminder Service.
    - The Reminder Service schedules the reminder and stores the details in the Reminder DB.
4. **Alert Notification**:
    - The Reminder Service places reminder notifications in the ActiveMQ queue based on the scheduled times.
    - The Alert Service listens to the queue, consumes reminders, and sends notifications to users through the Calendar Frontend.
5. **Route Visualization**:
    - Users can request to view the route of a planned trip.
    - The Calendar Frontend uses Google Maps API to display the route.

By utilizing this service architecture, the application provides a seamless and secure user experience for trip planning and reminder management. Each service is designed to perform a set of specific tasks, contributing to the overall functionality of the application.

Deployment:

Each service is containerized using Docker to ensure consistency across different environments and ease the deployment process. Kubernetes is utilized to orchestrate these containers effectively, managing scaling, load balancing, and service health automatically. This deployment strategy enables:

- **Scalability:** Services can be easily scaled up or down based on demand.
- **Resilience:** Kubernetes manages service availability, restarting containers that fail and distributing load.
- **Development and Operations Efficiency:** Streamlines the development process by standardizing environments and automates many operational tasks.

This structured deployment and service architecture allow the application to provide a seamless and secure user experience for trip planning and reminder management. Each service is designed to perform a set of specific tasks, contributing to the overall functionality and reliability of the application.

4. Design Decisions:

This chapter outlines the specific design decisions made in the development of the system, focusing on the chosen architectural elements, communication protocols, and technologies.

Service	**Communication Style**	**Technology Used**	**Reason for Choice**
Authentication	REST	Spring Boot, React	REST is chosen for its statelessness and scalability, allowing easy integration with the frontend.
Calendar Service (Trip Planner)	REST	Spring Boot	REST provides straightforward CRUD operations for trip management, enhancing developer productivity.
Reminder Service	REST, Async Message Queue	Spring Boot, ActiveMQ	REST for API endpoints, and ActiveMQ for decoupled, reliable message delivery between services.
Alert Service	Async Message Queue	Spring Boot, WebSocket	ActiveMQ for receiving reminders, WebSocket for real-time alert delivery to clients.
Route service	REST		

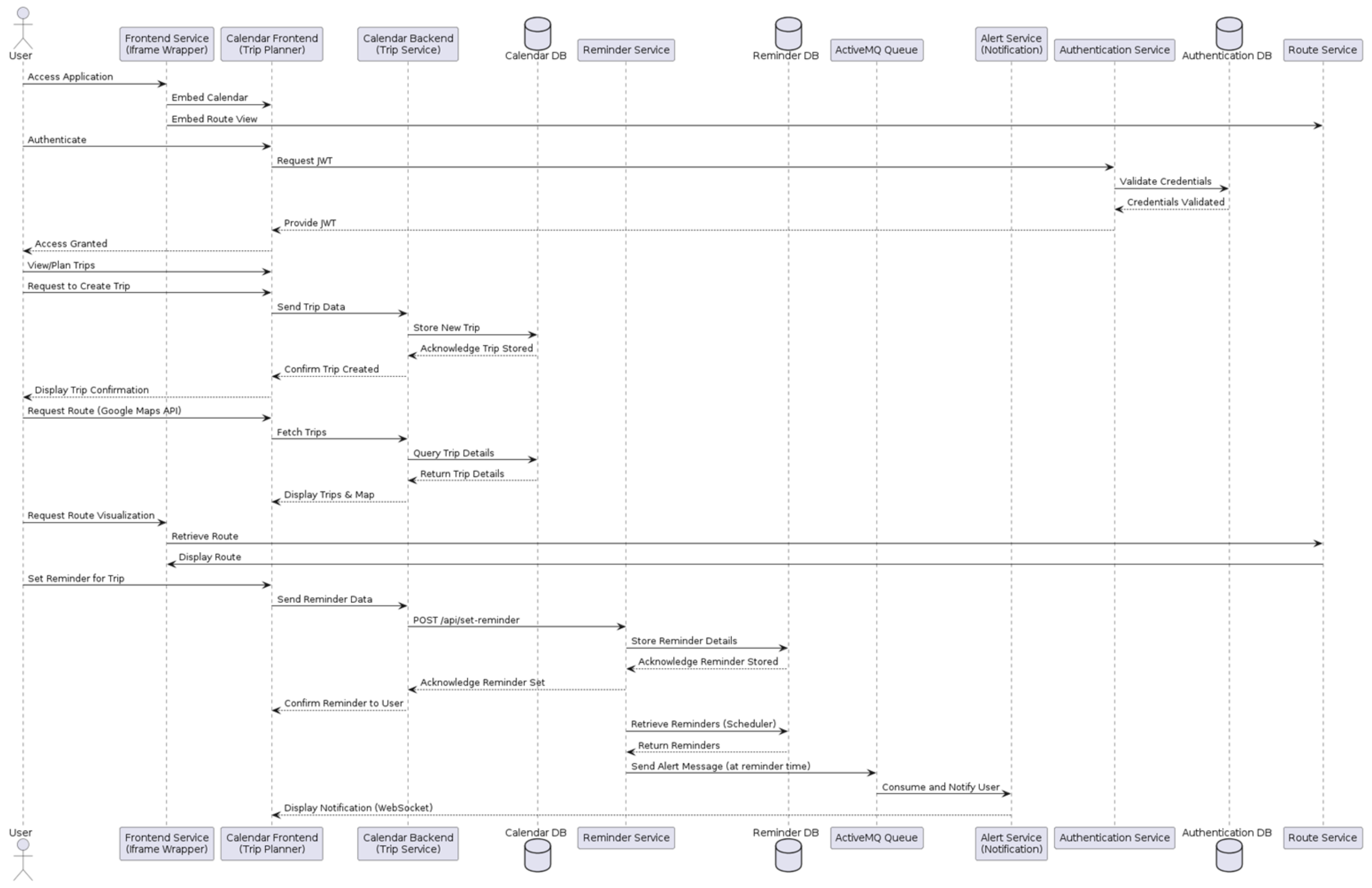


Figure 1 An UML sequence diagram displaying the interaction flow between the services.

Justification of Choices

SOAP vs REST: SOAP was considered too heavy for the services that required simple CRUD operations and real-time interaction. REST was chosen for its simplicity, flexibility, and compatibility with HTTP. It also allows stateless communication, which is beneficial for the scalability of microservices.

REST for APIs: RESTful APIs are widely adopted for their ease of use and their ability to leverage HTTP without additional messaging layers. It is used in services where request-response communication is sufficient and where operations are clearly defined in terms of CRUD.

Asynchronous Message Queue: For services that needed reliable, asynchronous communication, such as the Reminder Service sending notifications to the Alert Service, a message queue was implemented. ActiveMQ was chosen for its performance, reliability, and support for multiple languages and protocols.

WebSocket for Real-time Communication: For immediate delivery of alerts, WebSocket is used to maintain a persistent connection between the client and the server. This enables the Alert Service to push notifications instantly without the client needing to refresh or poll the server.

Our design choices were driven by the need for a scalable, reliable, and efficient system. By leveraging REST for API interactions and asynchronous messaging for decoupled services, we created a responsive and user-friendly experience. The technologies chosen align with best practices and industry standards, ensuring a robust architecture that can evolve as requirements change.

5. Validation:

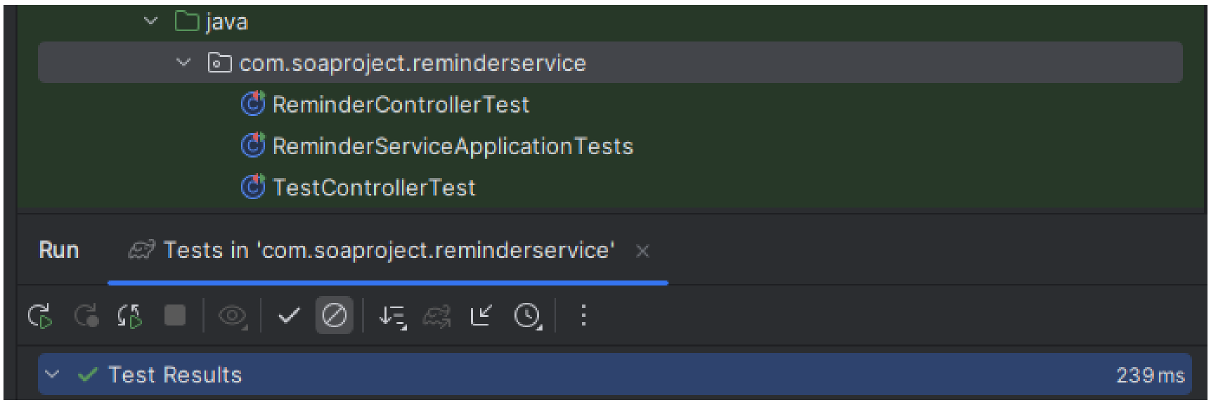
Unit Testing

Unit tests were constructed to validate the core functionalities of each controller within the system. These tests were designed to simulate API calls to the controllers, ensuring that each component behaves as expected when interacting with the front end and other services. The unit tests were performed using JUnit and Mockito for the backend services, while the front end used Jest and React Testing Library.

Key endpoints tested include:

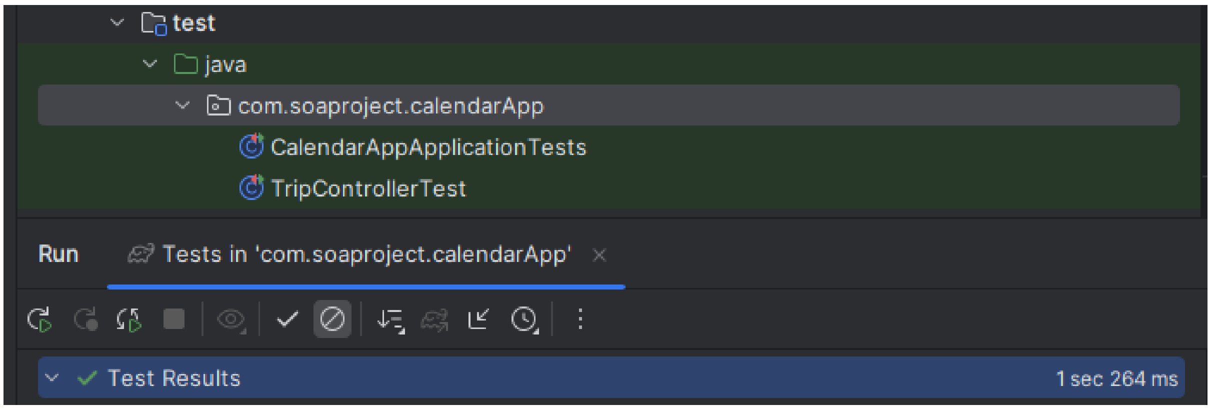
- Calendar Service (Trip Planner Service): Tests for trip planning, retrieval, and management.
- Reminder Service: Tests for setting and managing reminders.
- Alert Service: Tests for receiving and displaying alerts.

Below are two screenshots demonstrating the successful execution of the unit tests:



```
java
  com.soaproject.reminderservice
    ReminderControllerTest
    ReminderServiceApplicationTests
    TestControllerTest
Run   Tests in 'com.soaproject.reminderservice'
✓ Test Results                                   239 ms
```

Figure 2 tests for the Reminder Service



```
test
  java
    com.soaproject.calendarApp
      CalendarAppApplicationTests
      TripControllerTest
Run   Tests in 'com.soaproject.calendarApp'
✓ Test Results                                   1 sec 264 ms
```

Figure 3 tests for the Calendar Service

User Testing

In addition to automated unit tests, user testing was conducted to validate the end-to-end functionality of the system. We have performed a series of tasks that encompassed all aspects of the system, from logging in to planning trips and receiving reminders.

Happy flow showcasing all the features of the app

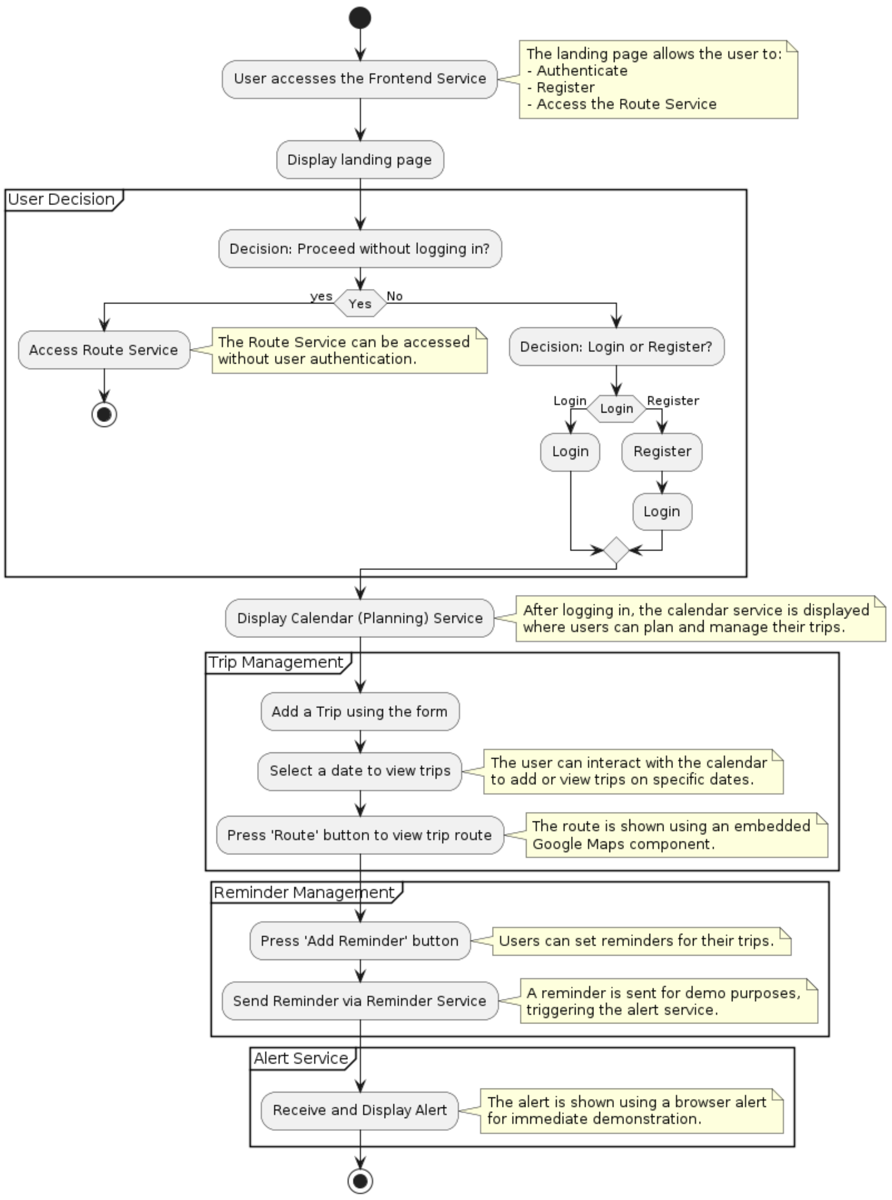

User accesses the Frontend Service
The landing page allows the user to:
- Authenticate
- Register
- Access the Route Service
Display landing page
User Decision
Decision: Proceed without logging in?
yes
Yes
No
Access Route Service
The Route Service can be accessed without user authentication.
Decision: Login or Register?
Login
Login
Register
Login
Register
Login
Display Calendar (Planning) Service
After logging in, the calendar service is displayed where users can plan and manage their trips.
Trip Management
Add a Trip using the form
Select a date to view trips
The user can interact with the calendar to add or view trips on specific dates.
Press 'Route' button to view trip route
The route is shown using an embedded Google Maps component.
Reminder Management
Press 'Add Reminder' button
Users can set reminders for their trips.
Send Reminder via Reminder Service
A reminder is sent for demo purposes, triggering the alert service.
Alert Service
Receive and Display Alert
The alert is shown using a browser alert for immediate demonstration.

Figure 4 An UML activity diagram of the system.

Upon accessing the application through the Frontend service (Figure 5) , users are presented with options to either login, register (Figure 6), or directly navigate to the route service(Figure 8). Once authenticated, the calendar (planning) service becomes available, where users can add a trip using the form on the right-hand side or select a date to view planned trips. For specific trips, users can click the route button to view the journey on a dynamically updated Google Map embedded within the page (Figure 7). Additionally, the calendar interface includes an option to add reminders for trips. For demonstration purposes, a 'send reminder' button is also provided (Figure 9); it triggers an asynchronous call to the Reminder Service, which in turn sends an alert to the Alert Service, displayed through a browser alert for immediate feedback.

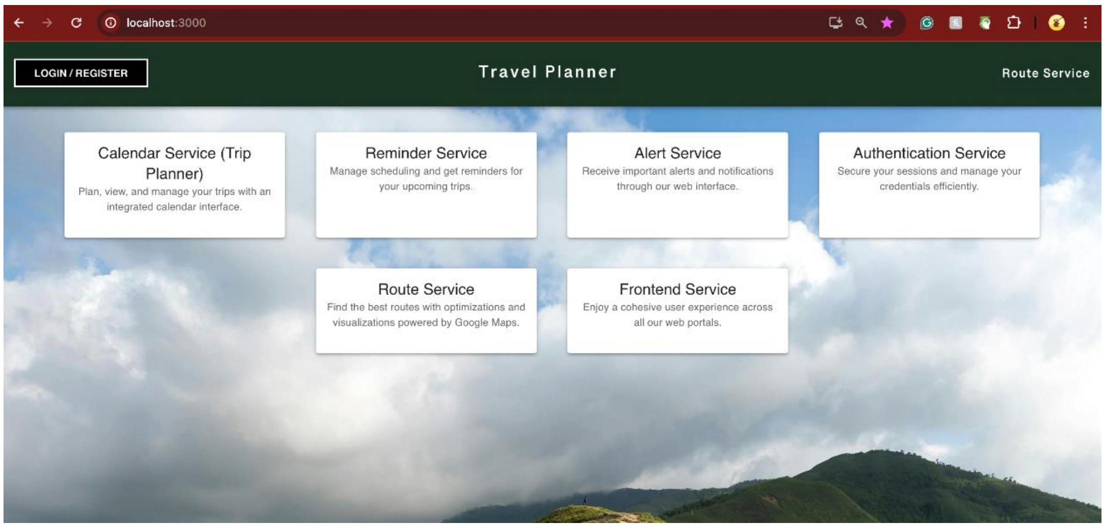
Figure 10 and Figure 11 showcase the docker composition and the Kubernetes Cluster.

6. Conclusion

Our project addresses the challenges individuals face in managing daily commutes and trips efficiently by integrating trip planning, reminders, and real-time alerts into a single platform. Through services like the Calendar Service, Reminder Service, Alert Service, and Route Service, users can organize trips effectively while receiving timely notifications. Architectural decisions focused on scalability and reliability, leveraging technologies like RESTful APIs and WebSocket for real-time communication.

Completing the project has bolstered our confidence in utilizing Docker for service management. Docker has proven instrumental in streamlining both development and deployment processes, ensuring consistency across different environments. Despite the challenges we encountered, our experience has underscored the importance of adaptability and continuous improvement.

Appendix



Calendar Service (Trip Planner)
Plan, view, and manage your trips with an integrated calendar interface.

Reminder Service
Manage scheduling and get reminders for your upcoming trips.

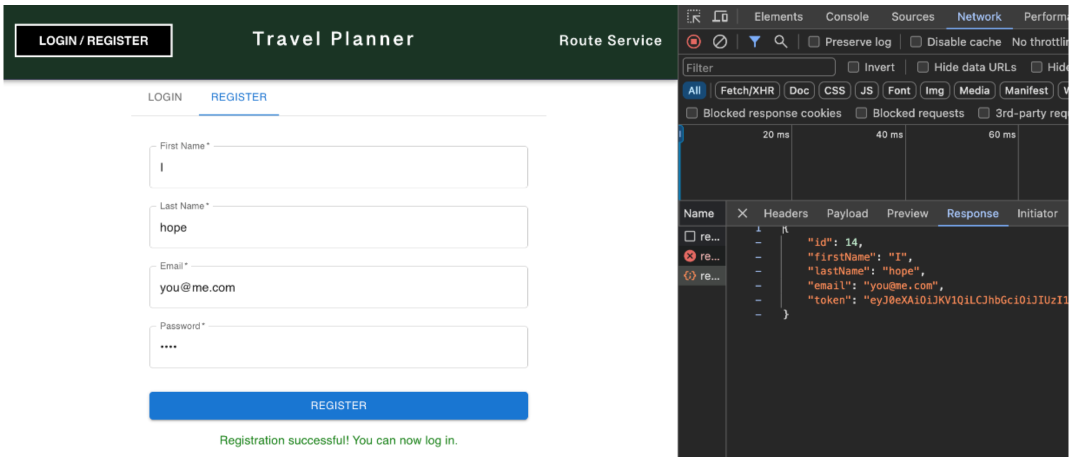
Alert Service
Receive important alerts and notifications through our web interface.

Authentication Service
Secure your sessions and manage your credentials efficiently.

Route Service
Find the best routes with optimizations and visualizations powered by Google Maps.

Frontend Service
Enjoy a cohesive user experience across all our web portals.

Figure 5 Landing Page



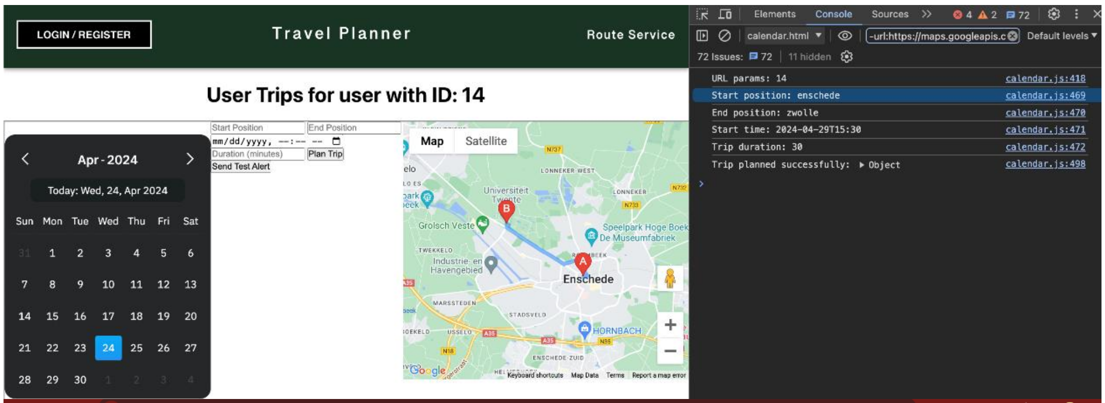
First Name * I
Last Name * hope
Email * you@me.com
Password * ••••

REGISTER

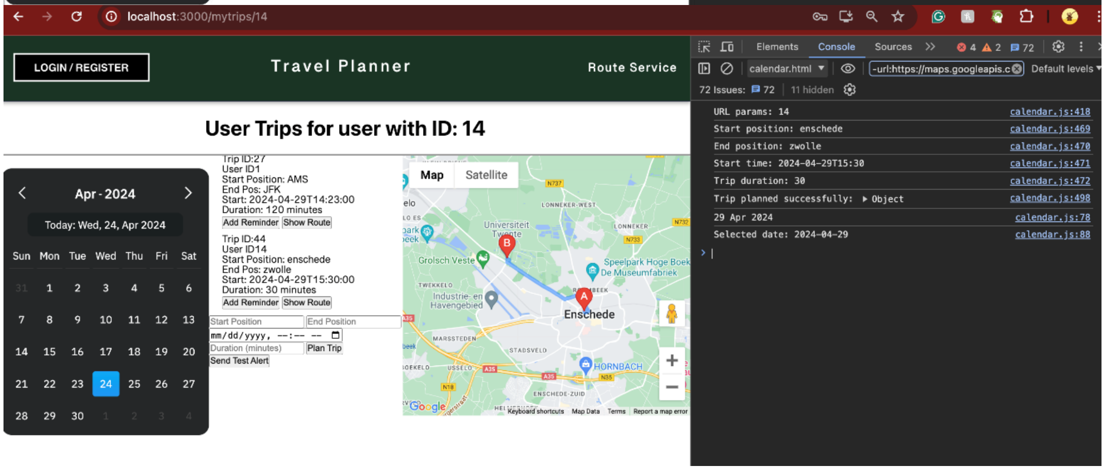
Registration successful! You can now log in.

```
{
    "id": 14,
    "firstName": "I",
    "lastName": "hope",
    "email": "you@me.com",
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1
}
```

Figure 6 Register / Login Screen



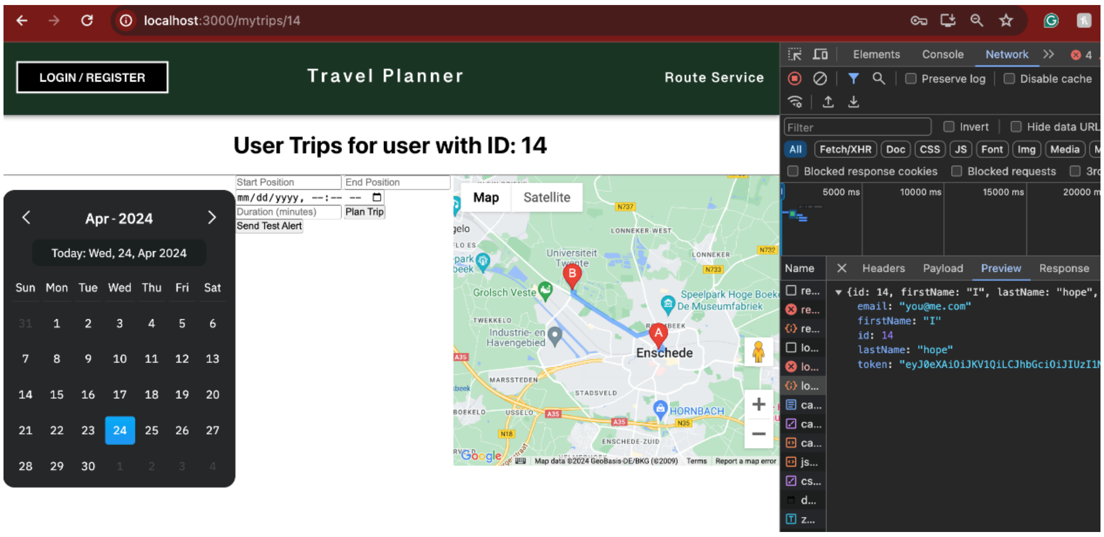
```
URL params: 14                                   calendar.js:418
Start position: enschede                         calendar.js:469
End position: zwolle                             calendar.js:470
Start time: 2024-04-29T15:30                     calendar.js:471
Trip duration: 30                                calendar.js:472
Trip planned successfully: ▶ Object              calendar.js:498
```



Trip ID:27
User ID1
Start Position: AMS
End Pos: JFK
Start: 2024-04-29T14:23:00
Duration: 120 minutes
Add Reminder Show Route

Trip ID:44
User ID14
Start Position: enschede
End Pos: zwolle
Start: 2024-04-29T15:30:00
Duration: 30 minutes
Add Reminder Show Route

```
URL params: 14                                   calendar.js:418
Start position: enschede                         calendar.js:469
End position: zwolle                             calendar.js:470
Start time: 2024-04-29T15:30                     calendar.js:471
Trip duration: 30                                calendar.js:472
Trip planned successfully: ▶ Object              calendar.js:498
29 Apr 2024                                      calendar.js:78
Selected date: 2024-04-29                        calendar.js:88
```



```
▼ {id: 14, firstName: "I", lastName: "hope",
    email: "you@me.com"
    firstName: "I"
    id: 14
    lastName: "hope"
    token: "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1N
```

Figure 7 Calendar Service

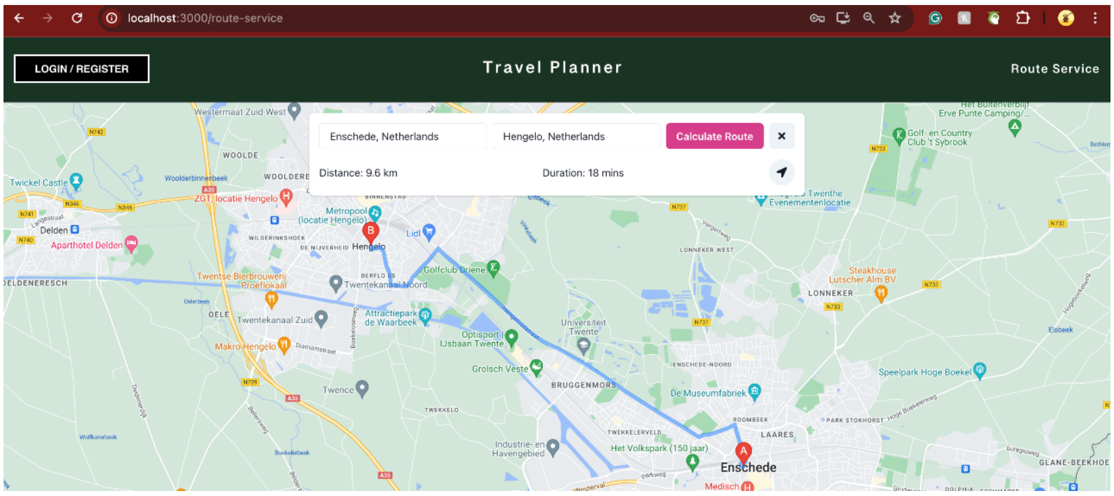
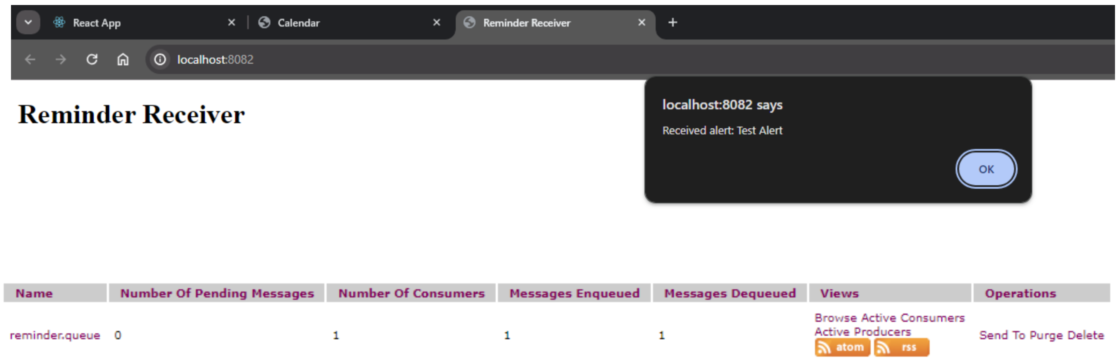


Figure 8 Route Frontend



Reminder Receiver

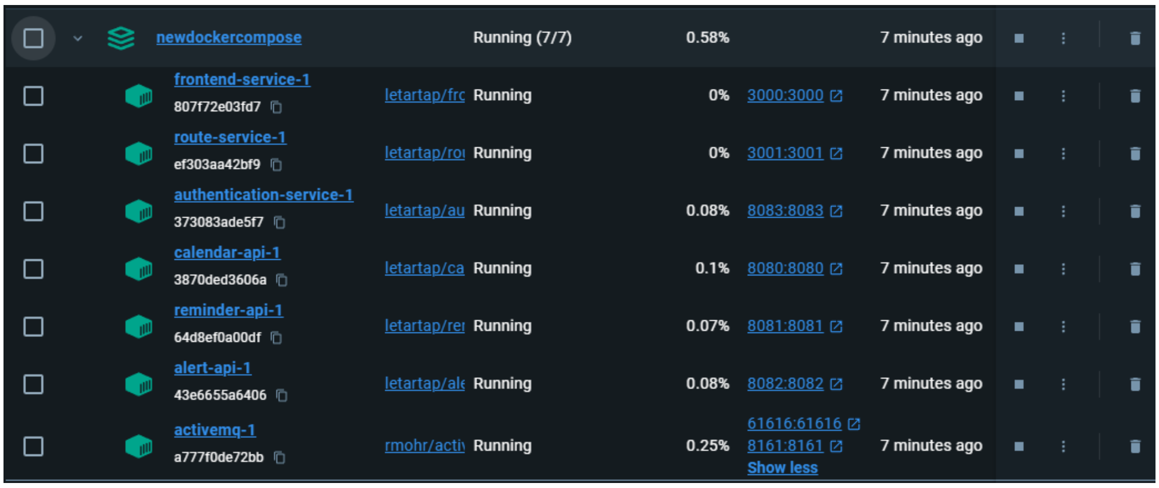
localhost:8082 says

Received alert: Test Alert

OK

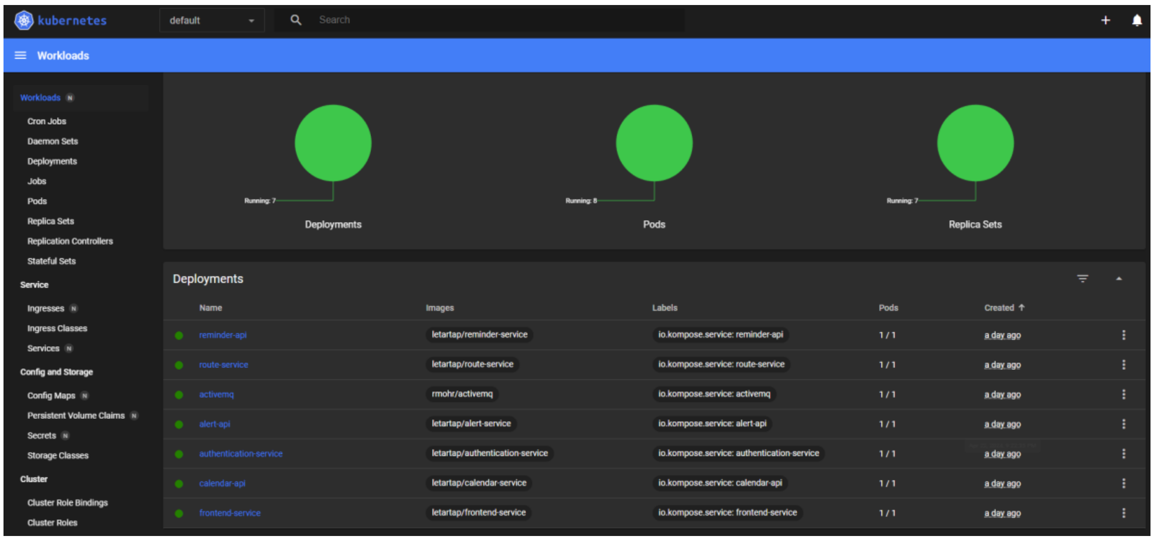
Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
reminder.queue	0	1	1	1	Browse Active Consumers Active Producers atom rss	Send To Purge Delete

Figure 9 Alert Service and ActiveMQ Queues



newdockercompose		Running (7/7)	0.58%		7 minutes ago
frontend-service-1 807f72e03fd7	letartap/frc	Running	0%	3000:3000	7 minutes ago
route-service-1 ef303aa42bf9	letartap/roi	Running	0%	3001:3001	7 minutes ago
authentication-service-1 373083ade5f7	letartap/au	Running	0.08%	8083:8083	7 minutes ago
calendar-api-1 3870ded3606a	letartap/ca	Running	0.1%	8080:8080	7 minutes ago
reminder-api-1 64d8ef0a00df	letartap/rei	Running	0.07%	8081:8081	7 minutes ago
alert-api-1 43e6655a6406	letartap/ale	Running	0.08%	8082:8082	7 minutes ago
activemq-1 a777f0de72bb	rmohr/acti	Running	0.25%	61616:61616 8161:8161 Show less	7 minutes ago

Figure 10 Docker Composition



Deployments

	Name	Images	Labels	Pods	Created ↑
●	reminder-api	letartap/reminder-service	io.kompose.service: reminder-api	1 / 1	a day ago
●	route-service	letartap/route-service	io.kompose.service: route-service	1 / 1	a day ago
●	activemq	rmohr/activemq	io.kompose.service: activemq	1 / 1	a day ago
●	alert-api	letartap/alert-service	io.kompose.service: alert-api	1 / 1	a day ago
●	authentication-service	letartap/authentication-service	io.kompose.service: authentication-service	1 / 1	a day ago
●	calendar-api	letartap/calendar-service	io.kompose.service: calendar-api	1 / 1	a day ago
●	frontend-service	letartap/frontend-service	io.kompose.service: frontend-service	1 / 1	a day ago

Figure 11 Kubernetes Deployments