Distributed Systems CT30A3041

Final Project – Task 1

Joonas Liedes

For this project I've chosen to build a task management system, which is designed to help individuals manage and track their tasks and deadlines. Built using the microservices architecture, the system comprises of three core services that provide the essential functionalities for task management. This approach makes the system lightweight, easily scalable and maintainable, while still providing the key features for task management.

The functional requirements include:

1. User registration and authentication.
2. Task creation, editing, and deletion.
3. Task status tracking.
4. Deadline setting and tracking.
5. Notifications and reminders for task deadlines.

Table of requirements:

Requirement	Microservice	Functionality
1	Authentication	Register a new user account
2	Authentication	Authenticate an existing user (login)
3	Authentication	Authorize user actions (access control)
4	Task	Create a new task
5	Task	Edit an existing task
6	Task	Delete a task
7	Task	Change the status of a task (In Progress, Completed)
8	Task	Assign a deadline to a task
9	Task	Get users task information
10	Notification	Send deadline reminders
11	Notification	Track notification history

Based on these functional requirements, we can identify the following key services:

1. **Authentication Service**
    a. **Scope:** Handles user registration, authentication, and authorization. This service manages user accounts, credentials, and access control, ensuring secure access to the system.
    b. **Interservice Communication:** The Authentication Service does not need to communicate directly with other services for fulfilling its core responsibilities.

However, other services (Task and Notification) will need to verify the user's authentication and authorization status before executing any action. This is achieved by using API tokens (JWT). After successful authentication, the client receives a JWT from the service and stores it. The token can then be sent from the client-side to the Task and Notification services for authorization.

2. **Task Service**
    a. **Scope:** Manages core task-related functionalities, such as task creation, editing, deletion and status tracking. This service handles task details like title, description, status, and deadline. Additionally, it provides and endpoint for tracking the user's tasks.
    b. **Interservice Communication:** The Task Service needs to verify the user's authentication and authorization before processing any request. To do this, it will use API tokens provided by the Authentication Service. The Task Service does not need to communicate directly with the Notification Service, as the client-side application will handle requests related to notifications.
3. **Notification Service**
    a. **Scope:** Sends notifications and reminders related to tasks and deadlines. This service sends notifications via in-app messages, and tracks notification history.
    b. **Interservice Communication:** Similar to the Task service, user authorization is confirmed with JWT's sent from the client side.

By focusing on these three loosely coupled microservices, we provide the necessary functionalities needed for users to manage their tasks while maintaining a lightweight and maintainable architecture. Each service is designed as a stand-alone, autonomous unit that can be developed, deployed, and scaled independently, allowing for increased flexibility and ease of maintenance.

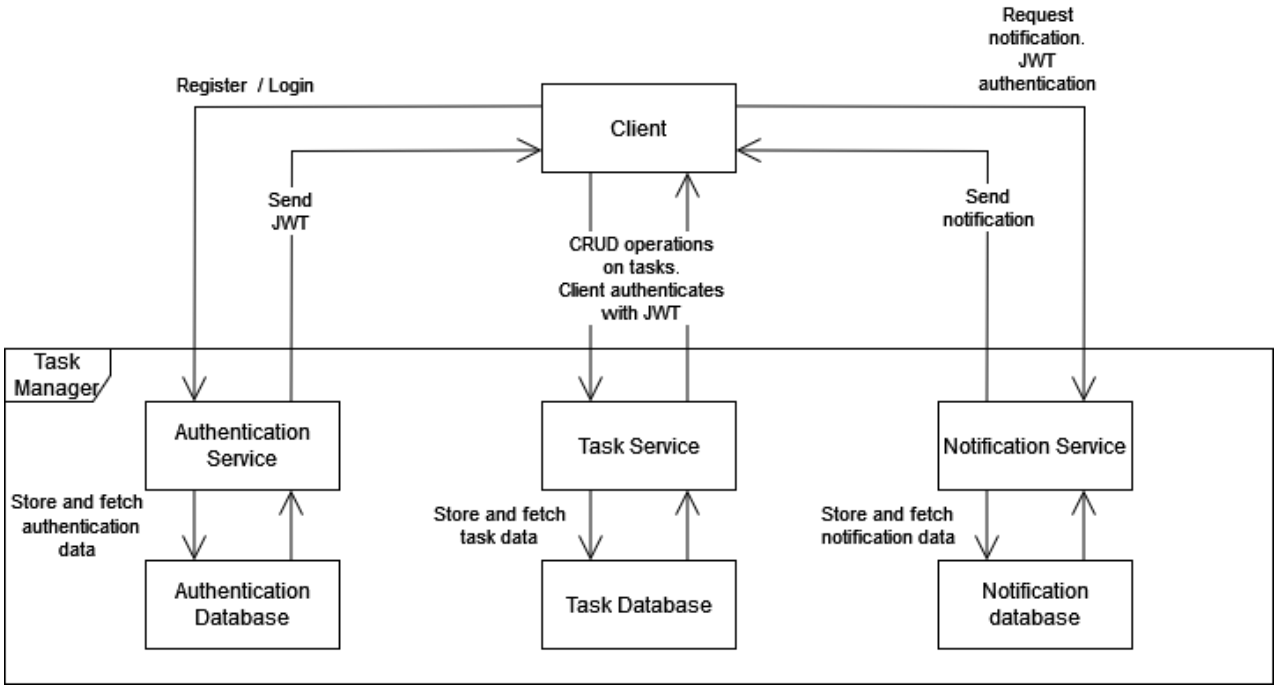

Figure 1. System architecture of the task manager application.

The communication pattern used in this system is RESTful APIs. REST stands for Representational State Transfer and it leverages HTTP methods (GET, POST, PUT, DELETE) for communication between the client-side application and the microservices, as well as between the microservices themselves if necessary. The interactions between the different components of the application as well as the overall architecture is depicted in Figure 1.

Communication Pattern:

1. **Client-side to Microservices:** The client-side application communicates with the Authentication, Task, and Notification microservices using REST APIs. Each microservice exposes well-defined endpoints that the client-side application interacts with to perform the operations previously described.
2. **Microservices to Microservices**: In the given architecture, direct communication between microservices is minimal. The Task Service and Notification Service need to verify the user's authentication and authorization status using JSON web tokens provided by the Authentication Service, which are sent from the client-side.

Limitations around communication for microservices:

1. **Increased complexity:** Microservices architecture introduces additional complexity in terms of communication, as multiple components need to coordinate to provide the desired functionality, which might increase development and maintenance effort especially in very large systems.
2. **Latency:** Since microservices communicate over a network, increased latency can be an issue compared to monolithic applications, where all components reside within the same unit.
3. **Security:** Ensuring secure communication between microservices can be more complicated, as the services communicate via a network and each service should have proper authentication and authorization mechanisms in place.
4. **Data consistency:** Microservices often depend on their own data storage, which can lead to consistency issues if not managed correctly. Ensuring data consistency across microservices can be difficult, especially in large scale distributed environments.
5. **Network reliance:** As microservices depend heavily on networks for communication, network issues may impact the system's functionality and performance.

Despite these limitations, microservices architecture offers several advantages over monolithic systems. These include scalability, flexibility, and better separation of concerns. With careful design and implementation of the communication patterns, these limitations can be mitigated to create robust and efficient systems.