Exercise TD - Plate-Forme Sociale

Name: João Pedro Marçal Storino

Course: ICM 2A

Subject: Informatique - Software Engineering

Date: December 17, 2024

Université Mines Saint-Étienne

Academic Year: 2024

Contents

Introduction

This document presents a complete architectural study and analysis for a social platform application. The goal was to approach the system design using multiple methods to enhance learning and provide a deeper understanding of software architectures. The methods include:

- **Component-Based Architecture:** Defining system components, their responsibilities, interfaces, and dependencies.
- **Architectural Styles:** Exploring two major architectural styles – **Pipe-and-Filter** and **Blackboard** – to demonstrate flexibility and scalability.
- **Microservices Architecture:** Implementing a modular design inspired by the DeathStarBench project, utilizing Apache Thrift for high-performance RPC communication.
- **Modeling Diagrams:** Component, Sequence, and Deployment diagrams were created to visualize the architecture and workflows.

This approach provides a structured methodology for system design while enabling adaptability for new functionalities, such as URL shortening and hashtag support. Each architecture was analyzed to evaluate the impact of these changes, with conclusions drawn regarding their advantages and trade-offs.

1 System Architecture for Social Platform

1.1 Application Components

The components of the social platform and their roles are detailed below:

- **AuthService (Authentication Service):**
  - *Responsibilities:* Manages user authentication, including login and registration.
  - *Interfaces:* Exposes endpoints for login and registration.
  - *Dependencies:* Requires access to the **Database** for user validation.
- **ContentService:**
  - *Responsibilities:* Handles creating, reading, updating, and deleting posts.
  - *Interfaces:* CRUD endpoints for posts.
  - *Dependencies:* Requires **AuthService** for user validation and **MediaService** for media uploads.
- **MediaService:**
  - *Responsibilities:* Manages media files (images and videos), including storage and retrieval.
  - *Dependencies:* Interacts with a file storage system (e.g., AWS S3).
- **HistoryService:**
  - *Responsibilities:* Retrieves a user's post history.
  - *Dependencies:* Queries post data from the **Database**.
- **RecommendationService:**
  - *Responsibilities:* Provides personalized recommendations for users to follow.
  - *Dependencies:* Requires user activity data and **SubscriptionService**.
- **SearchService:**
  - *Responsibilities:* Searches for posts or users based on keywords.
  - *Dependencies:* Accesses indexed data from the **Database**.
- **SubscriptionService:**
  - *Responsibilities:* Handles follow/unfollow actions for users.
  - *Dependencies:* Updates and queries connections in the **Database**.
- **Database:**
  - *Responsibilities:* Centralized storage for users, posts, media metadata, and subscriptions.

1.2 Connectors Between Components

The following connectors facilitate communication between the components:

- **REST API:** Allows synchronous communication for actions like login, post creation, and search queries.
- **RPC (Remote Procedure Call):** Enables efficient inter-service calls, such as media upload or recommendation fetch.
- **Asynchronous Messaging:** Manages notifications and background updates using message queues (e.g., RabbitMQ).
- **Database Connector:** Ensures consistent access to data for all components.
- **File Storage System:** Stores and retrieves large media files for posts.

1.3 Component Diagram

Figure 1 illustrates the high-level architecture, showing how components interact with one another.

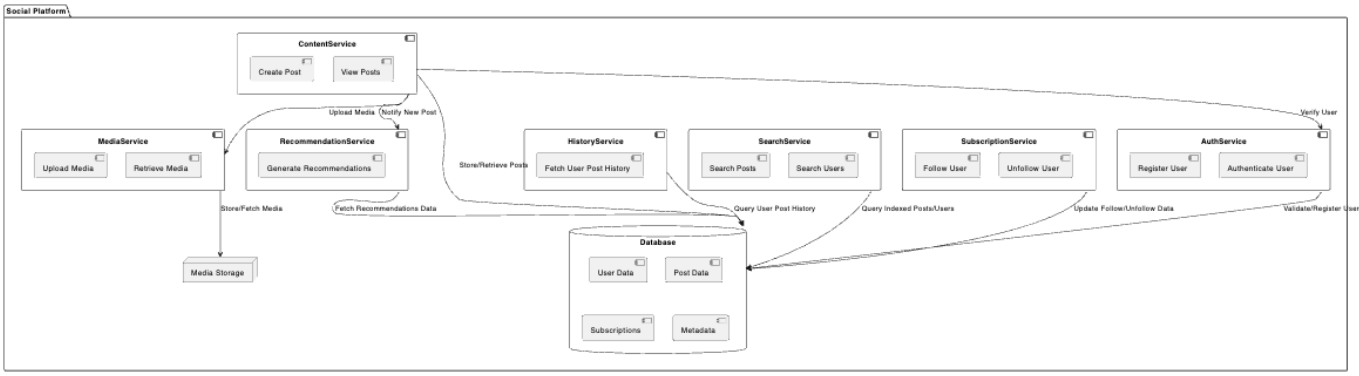

Figure 1: Component Diagram for Social Platform

1.4 Dependencies Between Components

The following dependencies exist:

- **AuthService** requires the **Database** to validate users.
- **ContentService** depends on **AuthService** for user authentication and **MediaService** for handling media.
- **RecommendationService** retrieves user activity and subscription data.
- **HistoryService** queries user post data from the **Database**.

1.5 Sequence Diagram: Full Platform Workflow

The sequence diagram in Figure 2 represents the complete workflow, including registration, login, post creation, media uploads, and user interactions.

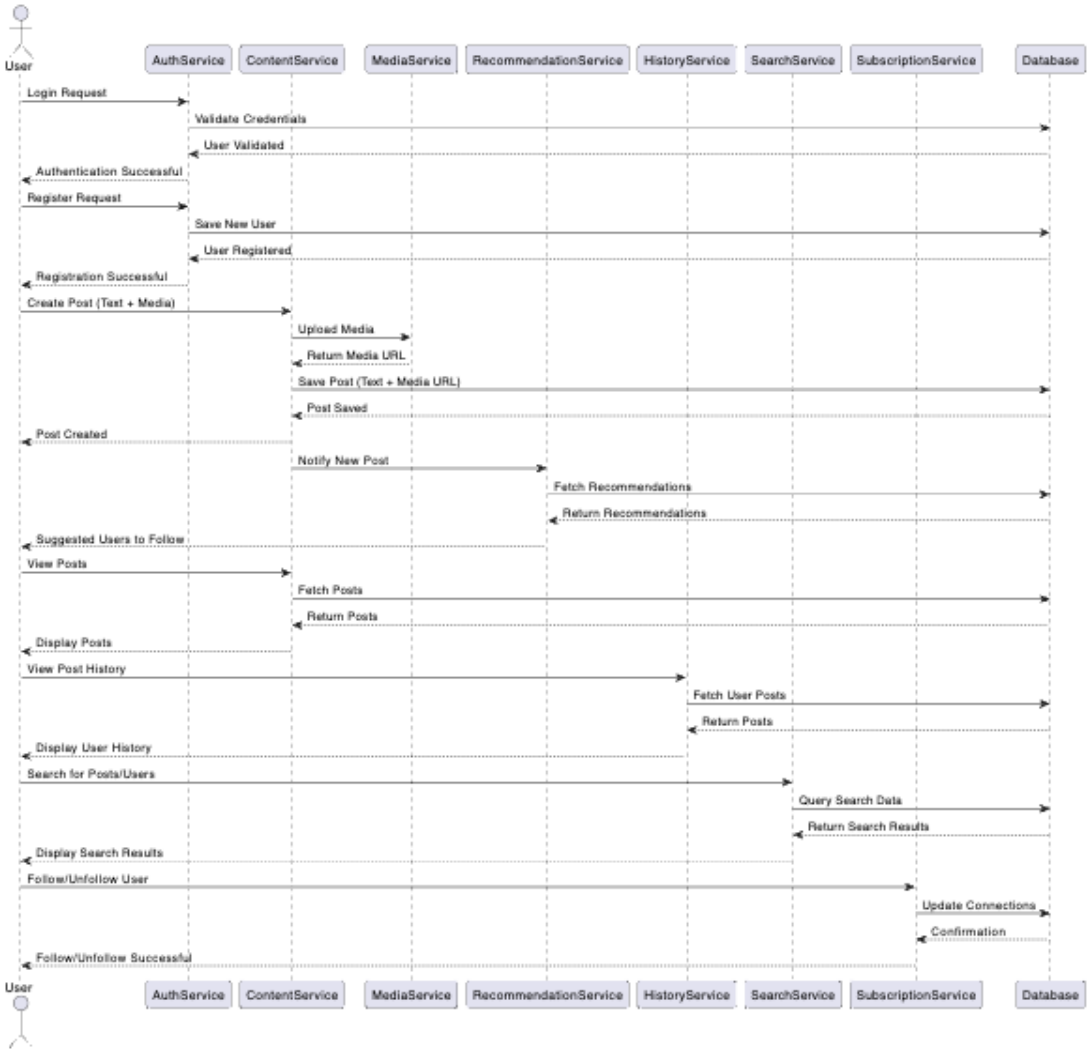

Figure 2: Sequence Diagram for Full Platform Workflow

2 Architectural Styles

2.1 Pipe-and-Filter Architecture

The **pipe-and-filter** architecture processes data sequentially through independent filters. Each filter performs a specific task and passes the output to the next filter.

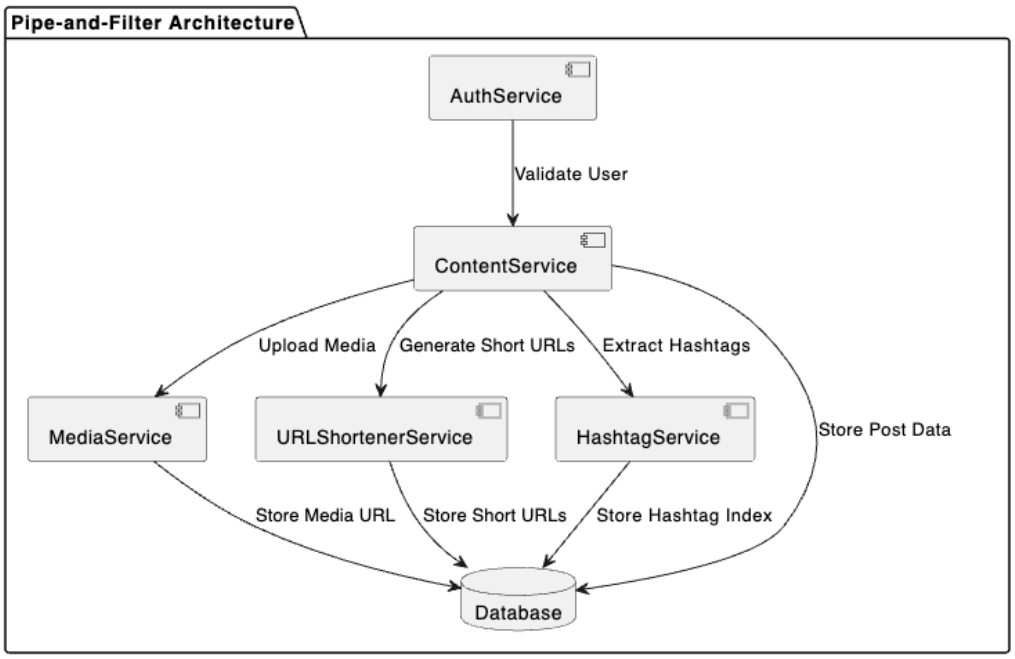

Figure 3: Pipe-and-Filter Architecture

2.2 Blackboard Architecture

The **blackboard** architecture uses a shared data space for interaction among components. Components publish and retrieve information from the blackboard independently.

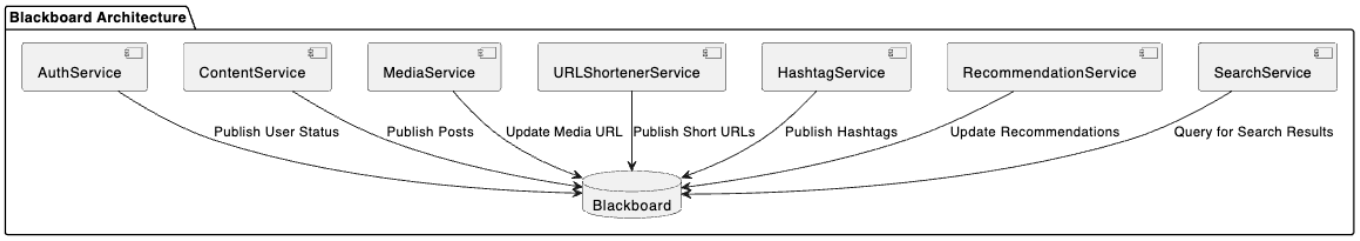

Figure 4: Blackboard Architecture

2.3 Impact of New Functionalities

The new functionalities added are:

URL Shortening Service: Generates and redirects shortened URLs.
Hashtag Support: Allows indexing and searching of hashtags.

Impact Analysis:

- **Pipe-and-Filter:** Requires modifying the pipeline and adding filters. **3 components affected**.
- **Blackboard:** New services integrate smoothly with minimal changes. **2 components affected**.

Conclusion: The blackboard architecture is more adaptable for new features due to its flexibility and minimal impact on existing components.

3 Microservices Architecture

A microservices architecture, inspired by the **DeathStarBench** project, has been proposed to implement the social platform. The DeathStarBench social network application utilizes **Remote Procedure Call (RPC)** as the primary communication method, implemented using the **Apache Thrift** framework. The architecture leverages REST-like principles to ensure efficient and modular service interactions.

3.1 Apache Thrift Overview

Apache Thrift is a framework for scalable cross-language communication. It allows services to define their APIs and data structures using a simple `.thrift` file, which is then compiled into multiple programming languages. The key features include:

- **Cross-Language Support:** Services implemented in different languages can seamlessly interact.
- **High-Performance RPC:** Enables efficient inter-service communication with minimal overhead.
- **Interface Definition Language (IDL):** Facilitates the definition of APIs and their associated data structures.

For this platform, the `social_network.thrift` file defines the microservices required for the system, including endpoints for user authentication, post creation, and search functionality.

3.2 Reconstructed Component Diagram

Figure 5 represents the microservices architecture based on the DeathStarBench model.

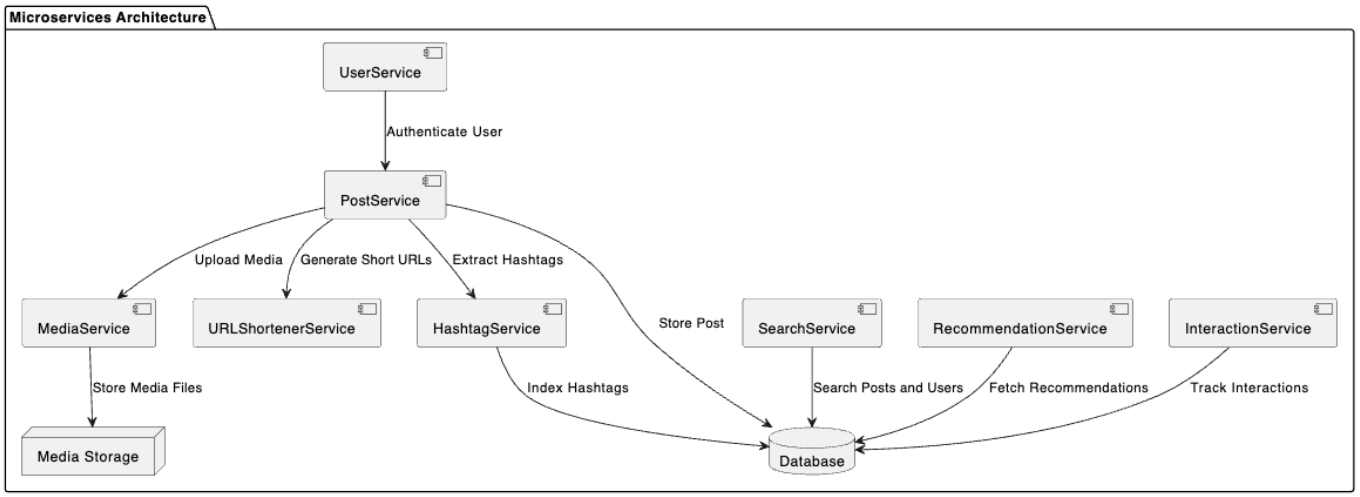

Figure 5: Microservices Component Diagram for Social Platform

The architecture consists of the following microservices:

- **UserService:** Handles user registration, login, and authentication.
- **PostService:** Manages CRUD operations for posts.

- **MediaService:** Handles media uploads, retrieval, and storage.
- **SearchService:** Enables searching for posts, users, and hashtags.
- **RecommendationService:** Generates personalized user-following recommendations.
- **URLShortenerService:** Generates and manages shortened URLs.
- **HashtagService:** Extracts, indexes, and stores hashtags.
- **InteractionService:** Tracks likes, comments, and interactions.

3.3 Sequence Diagram

The sequence diagram (Figure 6) outlines the interactions between the microservices for a post creation workflow with media upload and hashtag indexing.

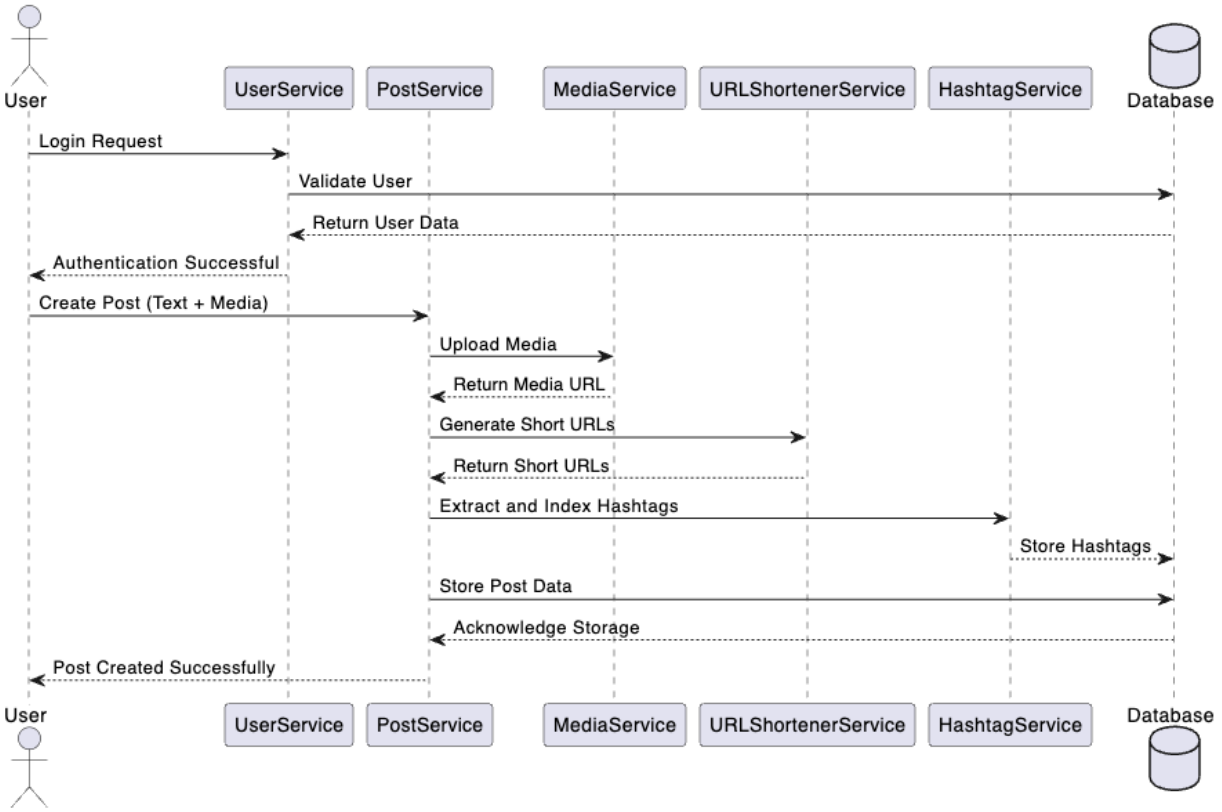

Figure 6: Microservices Sequence Diagram: Post Creation Workflow

Workflow Steps:

1. **UserService:** Authenticates the user.
2. **PostService:** Receives the new post request and validates data.
3. **MediaService:** Uploads media files and provides a media URL.
4. **URLShortenerService:** Generates shortened URLs for external links.
5. **HashtagService:** Extracts hashtags and indexes them for search.
6. **PostService:** Stores the final post in the database.
7. **RecommendationService:** Updates recommendations based on the new post.

3.4 Deployment Diagram

Figure 7 shows the deployment model for the microservices architecture.

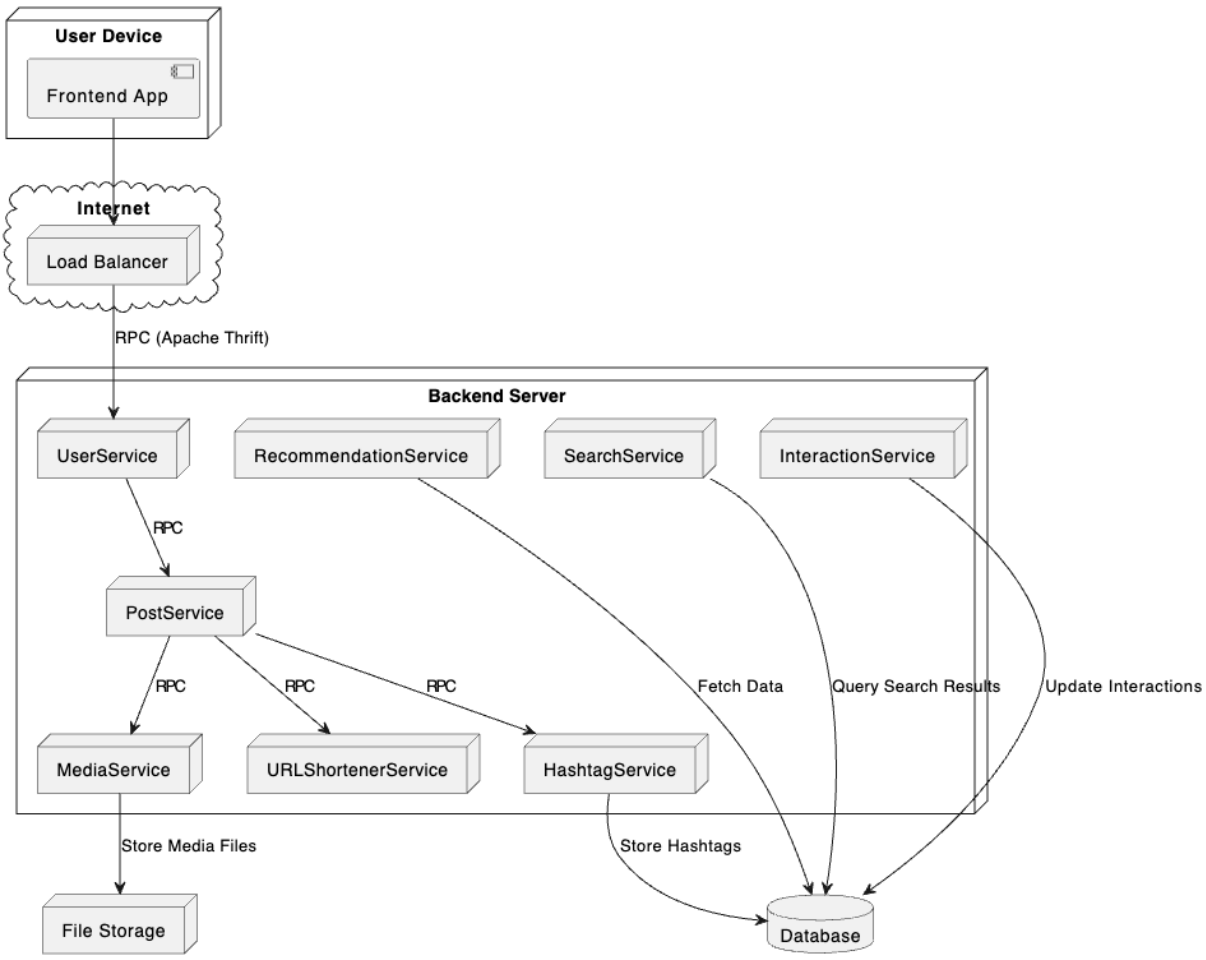

Figure 7: Microservices Deployment Diagram

Deployment Details:

- Each microservice runs in its own container (e.g., Docker).
- Services communicate via RPC over HTTP using Apache Thrift.
- A centralized database stores all persistent data.
- Media files are stored in an external file system (e.g., AWS S3).

4 Conclusion

In this document, we explored the architectural design of a social platform application, integrating various approaches to provide a comprehensive solution. The key insights include:

- **Component-Based Architecture:** A clear definition of components, their responsibilities, and dependencies was provided, ensuring modularity and reusability.
- **Pipe-and-Filter vs Blackboard Styles:** Both architectural styles were analyzed, demonstrating the trade-offs between sequential processing (Pipe-and-Filter) and flexibility through a shared data space (Blackboard).

- **Microservices Architecture:** Inspired by DeathStarBench, the microservices design provided scalability, modularity, and high performance using Apache Thrift RPC.

We further evaluated the impact of new functionalities, such as URL shortening and hashtag support, on these architectures. The **Blackboard architecture** demonstrated higher adaptability due to minimal changes required for integration, whereas the **Pipe-and-Filter** style required modifications to the processing pipeline.

By combining theoretical analysis with practical modeling (component, sequence, and deployment diagrams), this study highlights the importance of selecting the appropriate architectural style based on the system's requirements, scalability needs, and flexibility for future enhancements.

Final Remarks: The microservices approach offers the best solution for real-world scalability and modularity, while the analysis of multiple architectural styles enhances the learning process and prepares us for making informed architectural decisions in complex systems.