German University in Cairo
Department of Computer Science
Assoc. Prof. Mervat Abu El-Kheir

Architecture of Massively Scalable Applications, Spring 2025

Final Project Description

Deadline is 19/05/2025 11:59 PM

1 Introduction

In the final project for the Architecture of Massively Scalable Apps course, each team will propose and implement a comprehensive, scalable system. Project ideas may include a social media application, an e-commerce system, or any high-traffic solution that aligns with the principles discussed throughout the course. The project must use a microservices architecture where every service is independently deployable, scalable, and maintains loose coupling.

This final project motivates you to design, develop, and deploy a robust microservices architecture that can handle high traffic and distributed system complexities. Your project must comprise independently deployable microservices, fully functional APIs, dedicated databases, and comprehensive monitoring and logging systems. Effective team collaboration and a strategic work distribution plan are essential.

2 Deliverables

The final project must implement the following technical components:

a) **Independent Microservices:** Each microservice should be independently deployed and capable of being scaled up or down according to the load.

b) **Communication Methods:** Use RESTful APIs for synchronous communication and message queues for asynchronous messaging between services.

c) **Service Discovery:** Implement service discovery with tools such as Eureka or the native mechanisms provided by Kubernetes.

d) **Database per Service Pattern:** Each microservice must manage its own dedicated database (SQL/NoSQL) to ensure loose coupling and high cohesion.

e) **Load Distribution:** Handle user load effectively through traffic distribution using Kubernetes controllers or NGINX.

f) **Containerization:** Containerize each microservice using Docker and Docker compose.

g) **Deployment and Orchestration:** Deploy and orchestrate the containerized services with Kubernetes.

h) **Monitoring and Logging:** Integrate suitable monitoring and logging tools to ensure rapid fault detection and recovery.

i) **Code Design Patterns:** Apply at least **two different software design patterns** appropriately within your microservices.

3 Team Structure and Work Distribution

3.1 Overall Team Organization

- **Team Size:** Each team should consist of 12 to 15 members. In cases where the team is not fully staffed (i.e. less than 12 members), you may submit the form and additional members can be added later.
- **Scrum Master:** A single scrum master (team leader) will be designated to coordinate the project and act as the only point of communication between the teaching assistants and the team. The scrum master is responsible for submitting the final project form.

3.2 Sub-Team Organization

- The large team is to be divided into sub-teams consisting of 3 to 5 members each.
- Each sub-team should aim to implement at least two microservices or, alternatively, one microservice combined with the load balancing component or message queues or logging & deployment using Kubernetes.
- The microservices should cover critical functionalities of your overall project idea (e.g., user management, transaction processing, content delivery, etc.).

4 Minimum Requirements for Each Sub-Team

Each sub-team must comply with the following minimal criteria to ensure a consistent quality level:

a) **Microservice Implementation:**
   - Each sub-team is required to implement at least **one independent microservice** alongside a component from the architecture (i.e., load balancer, message queues, etc.) or two microservices.
   - In cases where a sub-team handles two services, the combined deliverable must meet the same standards as a standalone microservice.
   - You are required to implement an overall of at least 4 microservices.

b) **Feature Requirements per Microservice:**
   - Every microservice must provide at least **three distinct features or user stories**, which include:
   - **CRUD Operations:** Full Create, Read, Update, and Delete operations on a core entity.
   - **Additional Functionalities:** At least three extra functions beyond CRUD. Examples include:
     - User authentication or authorization (if applicable)
     - Search or filtering capabilities
     - Transaction handling or other domain-specific business logic
     - Data validation, error handling, or advanced logging functionality
   - **Reflection Usage:** You need to use reflection in the project in any part of the project as you find it necessarily (e.g, with message queues).
   - **Minimum of two design patterns implemented per microservice:** Each sub-team must choose and apply at least two relevant code design patterns and explicitly document:
     - **Which patterns were used.**
     - **Why these patterns were chosen.**
     - **How the patterns improved system design or scalability.**

Examples of recommended design patterns:

- **Singleton Pattern:** Used for shared components like database connection managers or logging services.
- **Command Design Pattern:** Ideal for wrapping/encapsulating requests and using it with message queues (RabbitMQ or Kafka), decouples sender and receiver.
- **Factory Pattern:** Useful for creating service objects dynamically.
- **Builder Pattern:** Ideal for constructing complex objects like User, Order, or Product entities.
- **Strategy Pattern:** Allows defining multiple algorithms and switching between them at runtime.
- **Observer Pattern:** Suitable for event-driven communication between services or within services.

c) **Technical Integration:**

- Containerize every microservice using Docker.
- Deploy and orchestrate services with Kubernetes.
- Ensure proper implementation of both RESTful APIs (synchronous communication) and message queues (asynchronous communication) for inter-service communication.
- Set up monitoring and logging to enable fault tolerance and rapid recovery.

5 Submission Guidelines

Create a repository for your Project with separate branches for each team member with the Main branch for integrating your work. Your repository name should be FinalProject - Your Team Number - your Team name (e.g. ***FinalProject-100-Random_01***). Then add *****Scalable-Submissions***** OR *****Scalable2025@gmail.com***** as colaborator to your repository. Your history of commits will be monitored as an indication for the participation for all the team members so make sure you divide the work equally among you.

Appendix A: Sample Microservices Splits

Below are examples of microservices splits that were provided in previous project descriptions. These serve as a guide for how to decompose the overall system into microservices. You can use them to create the same or similar ideas.

Facebook-Replica

- **UserApp:** Handles user registration, login, logout, and profile edits.
- **WallApp:** Generates a user wall from previous entries (recommended posts / pages and followed users / pages) and allows for the creation of new posts.
- **MessagesApp:** Facilitates sending private messages between users.
- **SearchApp:** Enables searching for users and posts.

eBay-Replica

- **UserApp:** Handles user registration, login, logout, and profile edits.
- **MessagesApp:** Manages private messaging between users.
- **SearchApp:** Allows searching for products.
- **MerchantApp:** Enables merchants to add items, sell products, and manage/view stock.
- **BiddingApp:** Supports bidding on specific items.
- **TransactionsApp:** Completes the buying process after a successful auction.

Wallmart-Replica

- **UserApp:** Handles user registration, login, logout, and profile edits.
- **SearchApp:** Enables searching for products.
- **TransactionsApp:** Completes purchases and maintains a history of transactions.
- **AdminApp:** Allows administrators to upload items, create promotions, and update stock.

Amazon-Replica

- **UserApp:** Manages user registration, login, logout, and profile edits.
- **SearchApp:** Provides product search functionality.
- **StoresApp:** Presents a categorized view of the available stores.
- **MerchantApp:** Enables merchants to add items, sell products, and manage/view stock.
- **TransactionsApp:** Finalizes the buying process after a purchase.

Gaming-Platform

- **UserApp:** Sign up, login, delete account, edit profile, block users, report users, follow/unfollow users.
- **PostApp:** Create, edit, delete posts (game invites, achievements, discussions), add, edit, delete comments; like/dislike comments and posts; bookmark favorite content, recommend posts based on engagement and interests.
- **NotificationApp:** Deliver notifications for comments, messages, tags, follows, and reports.
- **ModerationApp:** Manage user reports, ban users, and assign moderators.

Reddit-Replica

- **UserApp:** Sign up, login, delete account, edit profile, block and report users.
- **CommunitesApp:** Create and manage communities, recommend communities and threads based on user's activity.
- **ThreadApp:** Post threads and comments within subreddits, upvote, downvote, edit, and delete comments.
- **ModeratorApp:** Assign moderators, handle reports, and ban users.
- **NotificationApp:** Alerts for replies, mentions, subscriptions, and reports.

Piazza-Replica

- **UserApp:** Sign up, login, delete account, and edit profile.
- **CourseApp:** Create courses, register students, and assign instructors.
- **QuestionApp:** Post public or private questions; mention users, submit, endorse, and delete answers; create polls.
- **NotificationApp:** Alerts for replies, endorsements, polls, and bans.

Todoist-Replica

- **UserApp:** Sign up, login, delete account, edit profile.
- **TaskApp:** Create, read, update, delete tasks and sub-tasks; set due date and priority.
- **BoardApp:** Create and manage boards with tasks and discussions.
- **ReminderApp:** Send reminders and notifications for upcoming deadlines.
- **SearchApp:** Search across tasks, lists, and boards.

Spotify-Replica

- **UserApp:** Sign up/login, edit profile, personalize song, artist, and playlist recommendation, follow artist, like/unlike song.
- **PlaylistApp:** Create, read, update, delete playlists; set visibility to public or private.
- **SubscriptionApp:** Handle free and premium subscription tiers.
- **StreamingApp:** Stream music with ads or ad-free for premium users.
- **SearchApp:** Search tracks, artists, albums, and genres.

LinkedIn-Replica

- **UserApp:** Sign up, login, delete account, edit profile, add skills, add education, work experience and endorsements, block / report a user.
- **PostApp:** Create, edit, delete posts; comment, and like posts.
- **ConnectionApp:** Send, accept, and manage connection requests.
- **SearchApp:** Search people, jobs, companies, and posts.
- **NotificationApp:** Alerts for messages, connections, post activity, and endorsements.

Appendix B: Microservices Architecture and Recommended Tech Stack

In addition to the sample splits, below are recommendations for constructing a scalable microservices architecture:

Microservices Architecture

- **Layered Architecture:** Each microservice is divided into multiple layers (e.g., presentation, business logic, data access) ensuring modularity.
- **API Gateway Pattern:** A single entry point (API gateway) handles client requests and routes them to appropriate microservices, also handles authentication and request aggregation.
- **Event-Driven Architecture:** Microservices communicate through events using message queues (e.g., RabbitMQ or Kafka, in case of data streaming), to enable asynchronous communication.
- **Service Discovery and Load Balancing:** Use service discovery tools (e.g., Eureka or Kubernetes) combined with load balancing (e.g., NGINX) to manage inter-service communication and scale dynamically.

Recommended Tech Stack

- **Backend: Java SpringBoot application**
- **Containerization & Orchestration:**
  - **Containerization:** Docker & Docker Compose.
  - **Orchestration:** Kubernetes (with tools such as Docker Compose for local development).
- **Databases:**
  - **SQL:** PostgreSQL.
  - **NoSQL:** Cassandra, MongoDB, or similar.
- **Caching:** Redis.
- **Messaging/Queues:** RabbitMQ, or Kafka.
- **Service Discovery:** Eureka or Kubernetes-native solutions.
- **Load Balancing:** NGINX, or through Kubernetes.
- **Logging & Monitoring:**
  - **Logging:** Apache Flume, Graylog, or Grafana Loki.
  - **Monitoring:** Grafana and Kibana.
- **CI/CD:** Jenkins or GitHub Actions.

The appendix and recommended tech stack are adapted from Dr. Wael's previous descriptions

Good luck, and we look forward to your innovative, scalable solutions!