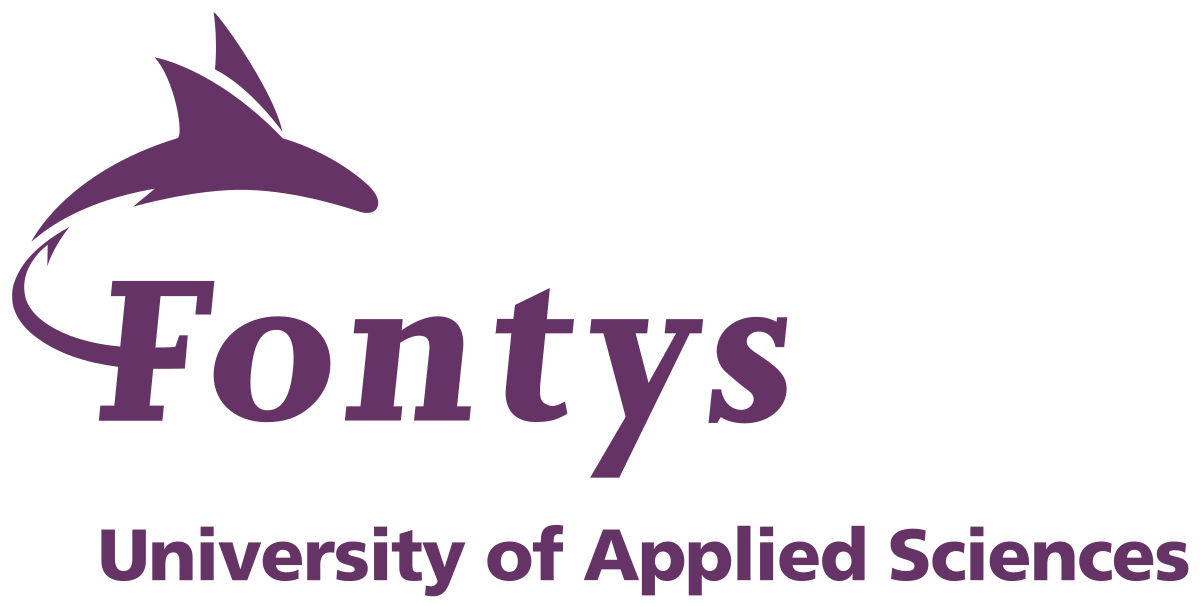
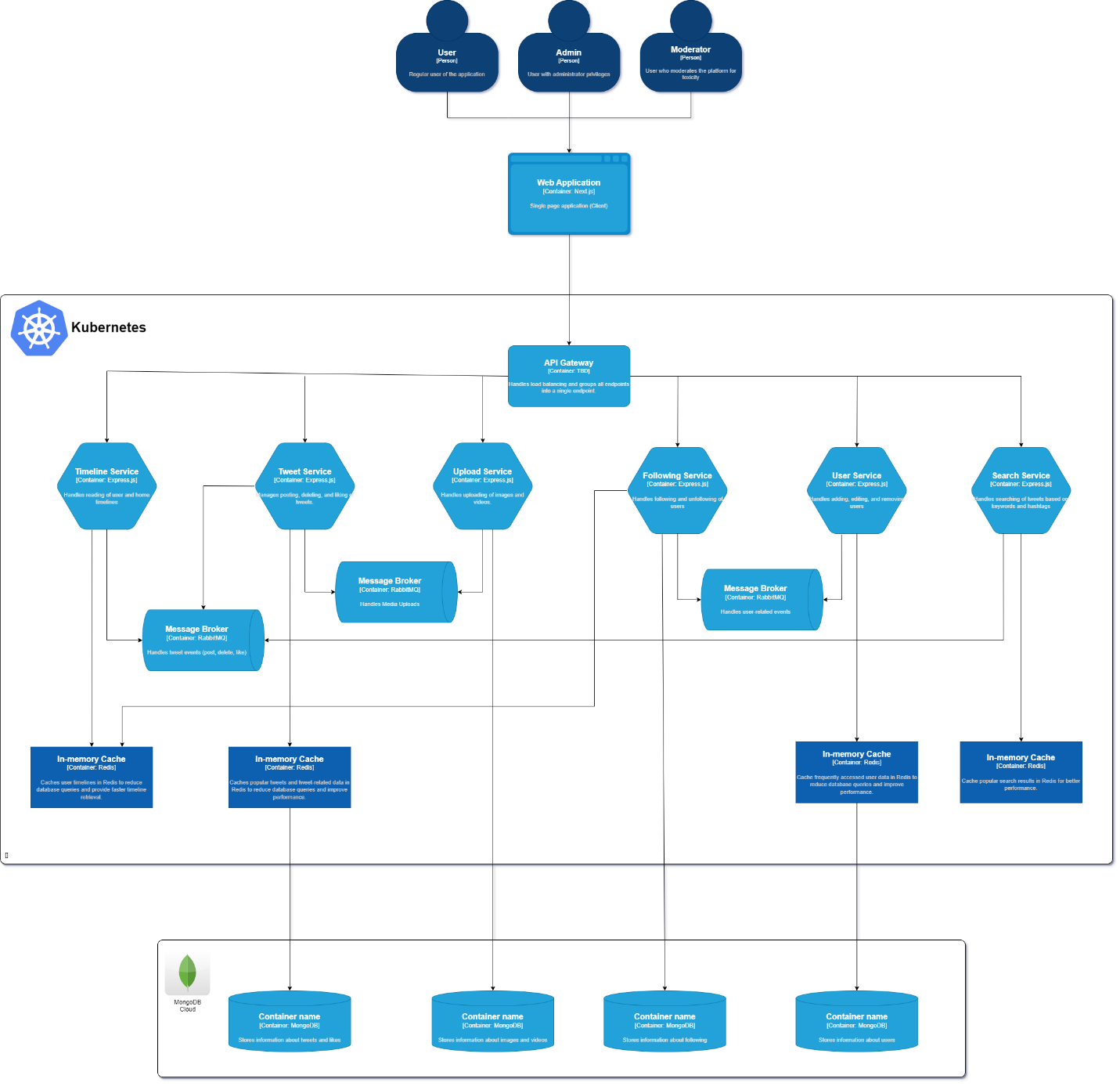
Design Document

Individual Project: Kwetter

|  |
| --- |
| **Date : 04.04.2023** |
| **Version : v1** |
| **Status : Draft** |
| **Author : Stanislav Petkov** |



# C4 Model



# Microservices

Here's a more detailed breakdown of how the microservices should be designed and interact with each other through RabbitMQ and Redis:

## Upload Service

* Handles uploading of images and videos.
* Once a media file is uploaded, the service stores the file in a storage service (e.g., Amazon S3) and publishes a message to a RabbitMQ exchange containing the file metadata (URL, user ID, etc.).

Tweet Service

* Manages posting, deleting, and liking of tweets.
* Listens to the RabbitMQ exchange for media upload events and updates the tweet content accordingly.
* Caches popular tweets and tweet-related data in Redis to reduce database queries and improve performance.

Timeline Service

* Takes care of reading user and home timelines.
* Listens to RabbitMQ exchanges for tweet events (post, delete, like) and updates user timelines accordingly.
* Caches user timelines in Redis to reduce database queries and provide faster timeline retrieval.

Search Service

* Enables searching of tweets based on keywords and hashtags.
* Listens to RabbitMQ exchanges for tweet events (post, delete) and updates its search index accordingly.
* Caches popular search results in Redis for better performance.

## Following Service

* Manages following and unfollowing of users.
* When a user follows or unfollows another user, publish a message to a RabbitMQ exchange containing the relevant user IDs.
* Invalidates cached timelines for affected users in Redis when follow/unfollow events occur.

User Service

* Handles adding, editing, and removing users.
* Listens to RabbitMQ exchanges for user-related events and updates user data accordingly.
* Caches frequently accessed user data in Redis to reduce database queries and improve performance.

# Technology Choices

Next.js (frontend):

1. **Server-side rendering (SSR)**: Next.js supports server-side rendering out-of-the-box, which can help improve the performance and SEO of your application.
2. **Ease of development**: Next.js simplifies the development process by providing features such as file-based routing, automatic code splitting, and built-in CSS support.
3. **API Routes**: Next.js offers an integrated way to create API endpoints, which can simplify front-end and back-end communication.
4. **Large ecosystem**: Being built on top of React, Next.js benefits from the large React ecosystem, including a wide range of community-driven components, libraries, and resources.

Express.js (back-end):

1. **Flexibility**: Express.js is a minimalist web framework that gives you the freedom to structure your application as you see fit, making it an ideal choice for building microservices.
2. **Middleware support**: Express.js allows you to easily add and customize middleware to handle different aspects of the request/response cycle, providing a modular approach to building your microservices.
3. **Easy to learn**: Express.js is relatively easy to learn and understand, especially if you're already familiar with JavaScript and Node.js.

Redis (caching service):

1. **High performance**: Redis is an in-memory data store known for its high performance and low latency, making it an ideal choice for caching frequently accessed data.
2. **Scalability**: Redis supports various data structures and can be easily scaled horizontally and vertically to handle increasing amounts of data and traffic.

MongoDB (database):

1. **Schema flexibility**: MongoDB is a document-based NoSQL database that allows for flexible and dynamic schemas, making it easier to model and store complex data structures such as tweets, media, and user profiles.
2. **Scalability**: MongoDB is designed to scale out by sharding data across multiple servers, which can help ensure high performance and availability as your application grows.
3. **High availability**: MongoDB's built-in replication and automatic failover capabilities can help ensure high availability and data durability.
4. **Developer-friendly**: MongoDB has a straightforward and easy-to-learn syntax, as well as a large community and extensive documentation, making it an accessible choice for developers.

RabbitMQ (message broker):

1. **Ease of use**: RabbitMQ is relatively simple to set up and configure compared to Kafka. Its configuration options and concepts are more straightforward, making it easier for developers to get started with.
2. **Flexibility**: RabbitMQ supports multiple messaging patterns, such as publish/subscribe, request/reply, and point-to-point communication. This flexibility allows you to model different communication scenarios in your application easily.
3. **Lightweight**: RabbitMQ is a lightweight message broker that consumes fewer resources compared to Kafka. This makes it suitable for smaller applications or projects with limited infrastructure requirements.
4. **Management Interface**: RabbitMQ provides a web-based management interface that allows you to monitor and manage your messaging infrastructure, simplifying administration tasks.

# ERD

Graphical user interface, text, application

Description automatically generated