IDG2001 Cloud Technologies Assignment 2

## Group: 6456

## Written by candidates: 10091, 10079, 10083

Contents

[What was done in assignment 1: 3](#_Toc198472331)

[Improvements made from assignment 1: 3](#_Toc198472332)

[Liking tweets and like batcher 3](#_Toc198472333)

[Logging requests and times reading the database: 4](#_Toc198472334)

[Load balancer 4](#_Toc198472335)

[Caching: docker containers and API 4](#_Toc198472336)

[Bonus features 4](#_Toc198472337)

[Unit tests 4](#_Toc198472338)

# What was done in assignment 1:

For the first assignment we implemented a relational database that we used to store user info, tweets and tweet hashtags. We then created a API with routes and controllers allowing users to perform CRUD (Create, read, update and delete) operations on data in the database using Python FastAPI.

We also implemented features that allowed user to search for users, tweets and hashtags, creating a user and log in to the application and post, read and delete tweets.

We created a simple front-end application that communicated with the back-end trough “RESTFUL-API”, that allowed users to see other tweets posted, click users, to see what they had posted, a search-bar that can be used to find users, tweets and hashtags and a component to post your own tweets.

All of this was hosted on different services where the database and back-end API were hosted on render as different services, and the frontend was hosted on “Vercel” (https://zitter-six.vercel.app/).

# Improvements made from assignment 1:

In this assignment we focused on adding more features that would be helpful if application grows and experiences a ton of traffic. This will include a load balancer, splitting traffic between 3 different caches before reaching the API. Some logging middleware that logs the incoming requests and saves them in a file and a like, tweet batcher, bathing up tweets before sending them to the database, limiting update requests to the database.

## Liking tweets and like batcher

Since the original assignment did not include a feature to like tweets, the first thing we did was implement likes to the tweet model…

## Logging requests and times reading the database:

For this project we created a middleware that is run on every incoming request, storing the http method (POST, GET, PATCH, etc), the route request (/users, /tweet/:id, etc) and the time the request was made.

Et bilde som inneholder tekst, skjermbilde, Font

KI-generert innhold kan være feil.   
  
these values get saved in a local file “logs.txt” logging requests in order.

We also implemented a function that counts the number of times a “GET” request reaches the database that will increment the number in the file before the database is accessed.

The function:  
 Et bilde som inneholder tekst, skjermbilde, Font

KI-generert innhold kan være feil.

This function is then included in every “GET” controller and is run after checking cache and right before checking the database like this:  
Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

KI-generert innhold kan være feil.

To see these logs in the API, we implemented a new route, “/logs”, that will use this file to deliver all requests made to the API and a number of times the database was accessed in a “json” format.

Having logs for API requests can be important for both debugging purposes and for tracking the amount of incoming traffic. This implementation allows developers to see what routes gets used the most, so they can see where it is important that the code is efficient, and at what days, time of days have the most amount of requests. Effective logging helps in the maintenance and efficiency of an application.

## Load balancer

For the API to handle an increased amount of requests, it is important to share the load among multiple services, so the system doesn’t slow down. For this project we use “NginX” for load balancing, where requests from “localhost:8080” gets distributed among the different caches in a “round robin” style dividing request equally.

## Caching: docker containers and API

For the caching we decided to use “Redis” both as docker containers and for caching in the API itself.

# Bonus features

## Unit tests

## 