**Distributed systems project**

**Overview**

For this project I produced a forum type of web application, where you are able to create and login to accounts, create posts and comments, and give upvotes to other people’s posts and comments. Users that aren’t logged in will still be able to see all the posts and comments and their upvote counts but will not be able to interact with them. The user has the ability to delete their own posts and comments but not the posts of others, unless a user deletes their own post which will also delete all the comments that were commented on the post. If the user sees the post or comment of another user, they will see an upvote button which they can click to give the post an upvote.

The application’s data is stored in a MongoDB database, and all of the information is stored so that the data is saved if the server goes offline, and the instance can be started up again after the server is turned on. The database houses each of the microservice objects, user accounts, posts, comments, and upvotes.

For authentication, JsonWebTokens (JWTs) are used to check that the user has logged in and that the token is valid.

There are some situations where authentication is required, such as when the user wants to create a new post or post a comment, and JWTs are quite an easy option to check if the user is logged in. I have some experience with JWT-tokens, so I think they are a good option for this task. To store all the user accounts, posts, and comments, I will use a mongoose database to save all the information.

**System requirements**

* All users can see all posts and comments, and their upvote counts.
* Only logged in users can create new posts or comments, or upvote.
* Users can create new accounts, accounts require a username, a valid email, and a password.
* Usernames must be unique, a new account with an existing username can’t be created.
* Upon successfully logging in, the user gets a JsonWebToken into their cookies, which will be used for authentication.
* If a user has a valid JWT-token, they can see an interactable form to create new posts and comments and see buttons to delete or upvote on posts.
* A user can only give one upvote to a post, and if the user has already upvoted a post or comment, they will see a button to delete their upvote.
* The user is required to fill all of the necessary fields to initiate a procedure such as creating a post or creating an account.
* The website should be fully loaded within 1 second of loading onto the webpage.

In this project we can identify 4 different microservices to create the functionality of these system requirements

**The microservices**

One microservice is the **authentication** microservice that takes care of creating new user accounts, and logging into to these accounts with the correct login credential. The user can receive a valid JWT-token by successfully logging in. If the user doesn’t have a JWT-token or the token is expired, the functionality to interact with the website is disabled. With the correct login credentials, the user will receive a JWT-token to their cookies from this authentication microservice and will be able to interact with the application.

The forum **posts** also have their own microservice. The application can call this microservice to create new posts, get the data of the posts, and delete posts. The website can call this microservice when the user wants to create a new post and the microservice will take care of creating the post object into the database. Authentication will be necessary to create new posts, and the user’s JWT-token will be checked to verify that the user is logged in. Fetching the posts however doesn’t require authentication, as everyone should be allowed to see the posts, but not create new ones. Posts have their own upvote counts, and the upvote microservice will keep track of the upvotes. Deleting a post will delete all the comments and the upvote objects from the database.

**Comments** are handled by their own microservice. Comments are linked to a specific post, and each comment stores the information of the ID of the post the comment was left on. The post’s unique postID will be used to fetch all the comments of a specific post and displayed on the website. Creating new comments requires the user to have a valid JWT-token. Users that aren’t logged in can still see all the comments but are not able to create their own comments or interact with existing ones. Since the comments are linked to a specific post, if the original post is deleted, all its comments will also be deleted. When a specific comment is deleted, the upvote object will also be deleted from the database.

To create the upvote functionality, **upvotes** have their own microservice. When a comment or a post is created, an upvote object is created to keep track of the upvote count and the different upvoters of the post or comment. The application can call this microservice to get data of the posts and comments and display their upvote count. If the current user is already on the list of the upvoters, the user will instead have a button to delete their upvote, in which case their upvote will be removed from the count and their name will be removed from the list of the upvoters. They can upvote the post again to add their upvote back to the count. Since the upvotes are linked to posts and comments, if the post or comment is deleted, its upvote object will also be deleted.

**Diagram

Description automatically generatedArchitecture diagram**

In the architecture diagram, we can see that the posts are linked to the upvotes-, and comments-objects. If a post is deleted, all the objects that are related to it are deleted, as they are no longer needed.

A comment’s upvotes are also linked to the comment-object. If a comment is deleted, its upvotes object is also deleted as it is no longer needed.

**Diagram

Description automatically generatedUML-diagram**

From the UML-diagram we can see that all the procedures in the upvote-, comment-, and post-objects are required to be validated by the authenticator microservice. If the user supplies a valid token, they are allowed to do the procedure. However, fetching data from the database doesn’t require a valid token and the user is able to get the data from the database. Posts, comments, and the upvote counts can be fetched by a user that doesn’t have a valid token.

**Communication patterns**

As we are working with a JavaScript application, the application’s front-end and back-end communicate using JSON-objects as it is the most common method in such applications. The front-end fetches data from the back end, and in some situations the request body includes some identifying data, so that the backend can search for the required data. For example, when the upvote data needs to be fetched from the database, the post’s or comment’s unique ID is supplied in the request body, so that the backend is able to find the specified data based on the ID. The backend supplies the found data back to the front-end in JSON-format, and the application uses the data to show it on the user interface. Fetching data from the server happens when the user loads the page, and the posts and comments need to be shown on the website, and when the user interacts with the application by creating new posts and comments, or by upvoting existing posts.

**Limitations**

Since the application is quite simple and the processes we are doing aren’t that complicated, data consistency is quite strong at this point. I made sure that all the necessary fields need to be filled before being able to send a request, so that the data doesn’t lack any necessary information, which might cause errors. Network speeds and latency aren’t a problem at this point as the application is still quite small and there isn’t a lot of data stored on the database. But as the number of comments and posts increase, the amount of requests sent for each load of the page will increase substantially to show all of the posts and comment. Over time the amount of requests sent might overload the server and make it slow for users to load the page. To combat this rising number of requests that need to be processed in the future, I could limit the number of posts and comments that are shown on the website at the same time so that too much data isn’t being requested by a single user. I could for example only show the first 10 or so posts and comments, so that not all the data is visible at the same time. This would cut down on the number of requests that need to be sent from the client to the server as less data is shown at a time.

I only created the bare functionality for a forum application, and the functionality of the app could be expanded upon by adding more features to the components created. For example, adding the functionality for a downvote button could be quite easy to add to the upvote microservice, and the feature to reply to specific comments could be added to the comments microservice. As I wanted to only showcase the general architecture of microservices, I only created simple functionality for the microservices.