

DAT076/DIT127 Project Report

Andreas Maass Magnusson, Marko Horvat, Sara Hedberg
& Zebastian Björkqvist

Introduction

This report will be about the application "Chatter", which can be found following this link:

<https://github.com/ZebastianBjorkqvist/DAT076-DIT127-Web-applications>

Chatter is a simple social media webpage, where users can create posts about various topics and view the posts of other users. Additionally, a user can choose to filter the posts by topics and interact with the posts by liking them. Naturally, the webpage also allows for the creation of a user account, logging in and out of said account, and viewing some information about their account such as the connected email address and statistics about their activity on *Chatter*.

Use Cases

The web-application *Chatter* has a few different use-cases.

- Load posts
- Filter and reload posts
- View profile page
- Create a post
- Like a post

User Manual

Installing the app

To install this app there are a few different steps. The first is of course downloading the code, for example by cloning the github repository linked in the introduction of this report.

The user then has to start the database by installing docker according to the instructions here <https://docs.docker.com/desktop/>. Then start up an instance (container) of PostgreSQL with the command: **docker run -env POSTGRES_HOST_AUTH_METHOD=trust -publish 5432:5432 -name web_apps_db -detach postgres:17** (This way of starting the database is the same as the instructions for lab assignment 5 in the course this report is written for).

Next, the user has to navigate to the server folder in the application and add a **.env** file, since it contains information which is considered secret and should not be uploaded to the github repo. The file has to contain a **SESSION_SECRET** and a **DB_URL**, which values can be found as a comment in the submission of this project.

After that, the user has to navigate to two different files in the terminal or an IDE (opening an integrated terminal for each of the files). The first file is the **start.ts** file found in the **src** folder of the server. Here the user has to first run **npm i** to install all the modules and then **npm run dev** which will start the backend of the application. Finally, the user has to navigate to the **main.tsx** file found in the **src** folder of the client. There the same commands have been run, and runs the application so that it can be accessed on **localhost:5173**.

Since this app is still only running locally, the database will be empty upon starting the program and no posts will be displayed. However, several users can be added from the computer that has completed the install, and all of them can interact with the app and the posts of all local users.

Using the app

To make describing the different ways a user can use the app Chatter, we will be splitting the application up by the different pages and describe the interactions possible on that page. When initially opening the application, the user will be taken to the login-page.

Logging in as a User

The first page you will see as a user is the login-page. If it is the first time opening the app for a user, they will have to create an account before being able to sign in. This can be done simply by pressing the button called "Create user" in the bottom right of the screen's content. A screenshot of the applica-

tion displaying this screen can be found further down in this section in figure 1

A user who already has an account can sign in to the application by filling in their username and password in the input fields with the respective title. After filling in this information, the user can click the "Log in" button. If the username and password are correct, the user will then be taken to the main page of the application. If either field is incorrect, an error message will be displayed and the user can try signing in again.

The image shows the login page for an application named "Chatter". At the top, there is a logo consisting of two overlapping speech bubbles, one light blue and one dark blue, followed by the word "Chatter" in a dark blue, cursive-style font. Below the logo, the text "Welcome to Chatter!" is displayed in a bold, dark blue font. Underneath this, the text "Sign in or create an account" is shown in a smaller, dark blue font. The main part of the page contains two input fields. The first field is labeled "Username" and has a placeholder text "Enter your username". The second field is labeled "Password" and has a placeholder text "Password". Both fields have a light gray border. To the right of the password field, there are two buttons: a dark blue button with the text "Log in" in white, and a light blue button with the text "Create user" in dark blue. The overall background is a light gray gradient.


Figure 1: Screenshot of Chatter's login-page

Creating a User

By using this page a new user can be created. This is done by entering an email address, a username and a password in the fields with the matching headers and then pressing the "Create user" button. If any fields are formatted incorrectly or contain an invalid input an error message will be displayed, and if all fields contain valid information pressing the button will create a user account and redirect to the login page where the new user account can be used to sign into

the app.

If you don't want to create a new account, the link under the "Create user" button can be pressed to navigate back to the login page.



Chatter

Welcome to Chatter!

Sign in or create an account

Create user

Email address

Username

Password

Create user

Already have an account? [Login here](#)

Figure 2: Screenshot of Chatter's page for creating a new user

Contents of the main page

This page contains the feed of different posts created by any user on Chatter. The navigation bar can be found in the top of the page, and will be described further in its own section. Underneath that, you can find a component for filtering the posts by different topic tags. It is used by typing in the topic you want to see, and then either pressing enter or the "Search" button to reload the

posts with the filter applied.

Different posts will be loaded in beneath that with the newest post at the top of the page, where one can see the title of the post on one side and the profile icon as well as the username of the post's creator in the top box. Underneath that in the post one can view what topics the post has been tagged as, and beneath that the content text of the post. In the bottom left of the posts a button for liking the post can be found, and to the right of that button the amount of likes a post has received can be found.

In the bottom right of the screen a floating button for writing a post can be found regardless of how far the user has scrolled. Pressing the button, which has a pen icon on it, will navigate to the page for creating a post.

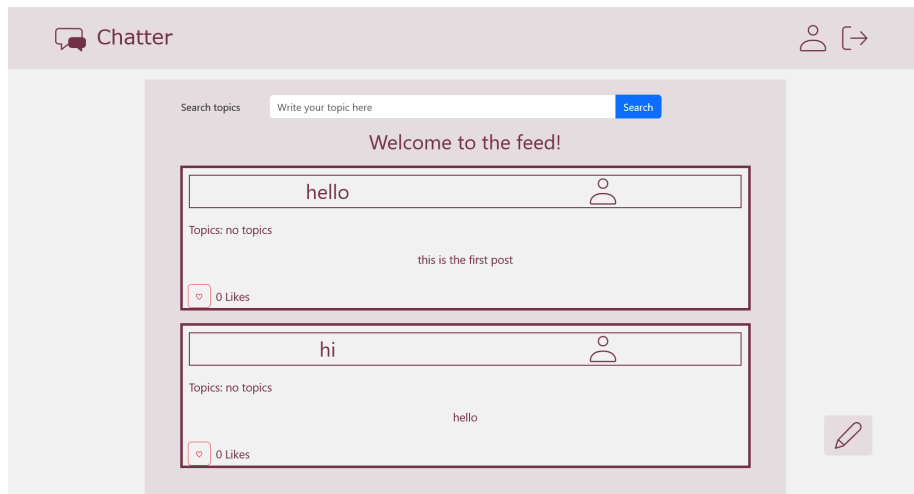


Figure 3: Screenshot of Chatter's main page

Navigation using the navbar

Chatter has a navigation bar which is displayed in all the different pages of the app while a user is logged in. The logo and name of the application can be found in the left part of the navbar, and clicking it will navigate the user to the main page (containing the feed of posts). Two icons can be found in the far right of the navigation bar. The one furthest to the right is for signing out of the app (which also navigates to the login page), and the icon next to it is for navigating to the profile page.

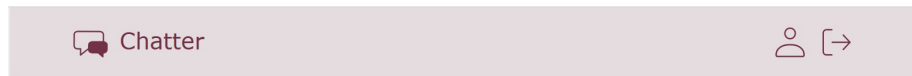


Figure 4: Screenshot of Chatter's navigation bar

Creating a post

This is where a new post can be created, and it is only accessible to a signed in user. To create the post add a title and some text for the main body of the post, and then press the "Submit post" button. Additionally, one can choose to add different topic tags to the post, which can be used to filter the feed to only show feeds about a certain topic. If any fields are filled in incorrectly, such trying to create a post with no topic, an error message will be displayed. The user can submit as many posts as they want before navigating to another screen, by using the "Cancel" button to get back to the main page, or navigating using the navbar in all ways described in the section about it.

A screenshot of the "Create Post" page in the Chatter app. At the top is a navigation bar with a speech bubble icon and the text "Chatter" on the left, and a user profile icon and a right-pointing arrow icon on the right. Below the navigation bar, the title "Create Post" is centered. Underneath the title, there is a text input field labeled "Title". Below that is another text input field labeled "Enter topic". To the right of the "Enter topic" field is a blue button labeled "Add topic". Below these fields is a large text area with the placeholder text "Write your post here...". At the bottom right of the text area are two buttons: a grey "Cancel" button and a blue "Submit Post" button.

Figure 5: Screenshot of Chatter's page for creating a post

Viewing the profile page

In this page of the application a user can find some information about their account and activity on Chatter. This is displayed near the center of the screen and contains the username and email address connected to the account, as well as information about the amount of posts that the user has created and liked on the app.

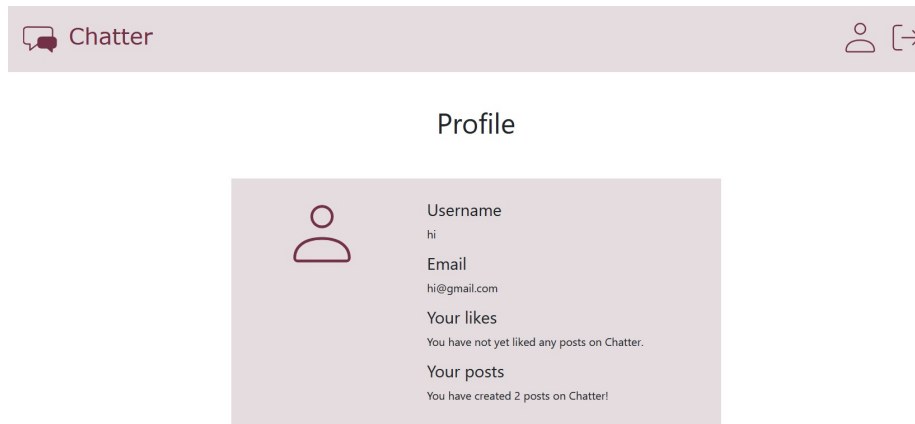


Figure 6: Screenshot of the profile page for a user on Chatter

Design

Frontend

This section will list the different components, including their name, purpose, props, state, and any calls made to the backend. All components are *.tsx* files and have been split into pages, contexts and components in the project and will be split similarly in this part of the report.

The navigation of the application is set up in the *main.tsx* element, the main entry point of the frontend application. It sets up the application's routing, authentication context, and renders the React component tree into the DOM. The file does not itself contain any props, state management or backend calls but initializes components that do.

Pages

This section will contain the different pages of the application, which use different components.

CreatePost page

purpose: Lets the user create a post and save it to the database.

props: None

state: postContent (stores the post's content), postTitle (stores the post's title), topics (stores the post's topics), alert (stores an alert), message

(stores the message used for the redirect-component), resetKey (stores the key used for the chooseTopics component)

calls to backend: createPost (stores new post in the database)

CreateUser page

purpose: This page lets the user create a new user account, which they can later use to login into the page. Acts as a container for the createUser-Form, which contains both different states and calls to the backend with createUser.

props: None

state: None

calls to backend: None

Feed page

purpose: This component contains the 'feed' of the application, ie the main page where different posts are loaded for the user to see. It ensures that the user is authenticated before displaying any posts. Also contains a floating button for navigating to the create-post-page.

props: None

state: posts (stores the fetched posts), title (displays current title, either welcome message or information about the current topic filter), message

calls to backend: getPosts (fetches all posts), getPostByTopic (fetches posts according to topics)

Login page

purpose: This page lets users log in to their account. The page acts as a container for the loginForm and loginHeader component.

props: None

state: None

calls to backend: None

Profile page

purpose: Allows the user to view information about their account on the application.

props: None

state: amtLikes (stores and displays text depending on the amount of posts a user has liked), amtPosts (stores and displays text depending on the amount of posts a user has created), username (stores and displays the username connected to the account), email (stores and displays the email connected to the account)

calls to backend: getCurrentUser (gets information about the user from the backend)

Context

This section contains the context AuthContext, which is used to check authentication and was created after the addition of sessions and users.

AuthContext

purpose: Provides authentication context to the application, manages user authentication state and persists it across page reloads, handles login and logout operations.

props: None

state: isAuthenticated (boolean, that tracks whether the user is authenticated. Is initialized from localStorage)

calls to backend: checkAuth (checks if the user is authenticated when the component mounts), apiLogin (attempts to log in the user), apiLogout (logs the user out)

Other components

This section contains all the components in the component folder, which are all used by different files in the pages folder.

chooseTopics

purpose: Lets the user choose topics to a post. The user could add and delete topics one by one. Deleting or adding topics calls a callback function: onTopicsUpdate, to return the updated list of topics. The component is used by the createPostPage.

props: onTopicsUpdate: (topics: string[]) => void

state: topic, topics (used to store a topic and a list of topics)

calls to backend: None

createUserForm

purpose: Contains the input fields for CreateUser page, handles states and API call for creating a user

props: None

state: email (stores text from the email input field), username (stores text from username input field), password (stores text from password input field), notification (contains the text that will display when clicking on Create user button)

calls to backend: createUser (creates a new user with the email, username and password given)

Feedcard

purpose: A component used by the feedpage. Contains one post which is exported to be displayed in the feedpage

props: uses props sent by the feedpage, as defined in the interface FeedCard-Props title: string; text: string; topics?: string[]; postId: number; user: string;

state: likeCount (stores the amount of likes a post has recieved), liked (tracks if the current user has liked the current post), error (stores error messages if there are any)

calls to backend: fetchLike (gets the like data for a post from the backend), setLikeState (sets the like data of a post in the backend using the postId and the status on if the current user chooses to like the post)

loginForm

purpose: A component used by the LoginPage. Contains a form for inputting a username and password with a button to submit for login. Also contains a button to go to the createUser page.

props: None

state: username (stores the username to log in with), password (stores the password to log in with), notification (contains the text that will display when clicking Log in button, errors (contains errors sent from the server)

calls to backend: login (sends username and password to the server trying to log in)

loginHeader

purpose: A common header with logo for the Login and CreateUser page

props: None

state: None

calls to backend: None

mainHeader

purpose: The header of all pages where a user is signed in to the application. Contains options to navigate/reload to the feedpage and profile page as well as a button for signing out

props: None

state: None

calls to backend: logout (handles logging out a user from the application)

redirectComponent

purpose: A component responsible for redirecting an unlogged in user to the login page when trying to access other pages of the app

props: message : RedirectProps message: string; url ?: string contains the message to display when redirecting and a url to direct to, has standard value "/"

state: None

calls to backend: None

searchComponent

purpose: Allows the user type in a string and pressing enter or "search" the component returns the string through a callback function called: onSearch. Is used by feedPage to let the user type in a topic that the feedPage can filter the posts using.

props: onSearch: (query: string) => void;

state: topic (stores a topic used for searching)

calls to backend: None

topicCard

purpose: Displaying a topic that the user has chosen in the chooseTopic component. The card has a close button, which deletes the card and also calls the callback function: onTopicRemoved. onTopicRemoved is used inside the chooseTopic component to delete the topic of the card of the list of topics that the user chosen.

props: name : string, onTopicRemoved: (topic: string) => void

state: None

calls to backend: None

API

Get All Posts and filter by topic if wanted

Method: GET

Endpoint: /post?topic=topicName

Request Body: None

Response:

- **200 OK:** Returns an array of Post objects filtered by topic, topic can be left empty
- **400 Bad request** If the topic is not a string or undefined
- **Error:** Standard HTTP error codes with a message

Create Post

Method: POST

Endpoint: /post

Request Body: {
 "author": "string",
 "text": "string",
 "title": "string",
 "topics": ["string"]
}

Response:

- **201 Created:** Returns the created Post object
- **400 Bad Request:** If validation fails

- **401 Unauthorized:** If user is not logged in
- **Error:** Standard HTTP error codes with a message

Get likes of a post

Method: GET

Endpoint: /post/{postid}/likes

Request Body: None

Response:

- **200 OK:** If getting the number of likes and if the user has liked the post was successful
- **400 Bad Request:** If post does not exist
- **Error:** Standard HTTP error codes with a message

Like a post

Method: POST

Endpoint: /post/{postid}/like

Request Body: {
 "like": "any",
 }

Response:

- **200 OK:** If post was liked correctly
- **400 Bad Request:** If validation fails or post doesnt exist
- **401 Unauthorized:** If the user is not logged in
- **Error:** Standard HTTP error codes with a message

Delete all posts, only created in testing environment

Method: DELETE

Endpoint: /post/reset

Request Body: None

Response:

- **200 OK:** If posts were deleted
- **Error:** Standard HTTP error codes with a message

Register User

Method: POST

Endpoint: /user

Request Body: {
 "email": "string",
 "password": "string",
 "username": "string"
}

Response:

- **201 Created:** Returns created user
- **400 Bad Request:** If validation fails
- **409 Conflict:** Username already exists
- **Error:** Standard HTTP error codes with a message

Login User

Method: POST

Endpoint: /user/login

Request Body: {
 "username": "string",
 "email": "string",
 "password": "string"
}

Response:

- **200 OK:** Returns authentication session and "Logged in"
- **401 Unauthorized:** If login fails
- **Error:** Standard HTTP error codes with a message

Get Current User Details

Method: GET

Endpoint: /user/current

Request Body: {
 "username": "string",
 "email": "string",
 "password": "string"
}

Response:

- **200 OK:** Returns current users details with username, email, number of posts and number of likes on posts
- **401 Unauthorized:** If not logged in
- **Error:** Standard HTTP error codes with a message

Check if current user is authenticated

Method: POST

Endpoint: /user/check-auth

Request Body: {
 "username": "string",
 "email": "string",
 "password": "string"
}

Response:

- **200 OK:** Returns username of current user
- **401 Unauthorized:** If not logged in
- **Error:** Standard HTTP error codes with a message

Log out user

Method: POST

Endpoint: /user/logout

Request Body: {
 "username": "string",
 "email": "string",
 "password": "string"
}

Response:

- **200 OK:** If user was logged out
- **Error:** Standard HTTP error codes with a message

Delete all users in database, only created in testing environment

Method: DELETE

Endpoint: /user/reset

Request Body: None

Response:

- **200 OK:** If users were deleted
- **Error:** Standard HTTP error codes with a message

Entity-relationship (ER) diagram

PostModels		
PK	id	bigint
	title	varchar(255)
	text	varchar(255)
	author	varchar(255)
	topics	json
	likedBy	json
	createdAt	timestamp with timezone
	updatedAt	timestamp with timezone

UserModels		
PK	id	bigint
	email	varchar(255)
	username	varchar(255)
	password	varchar(255)
	createdAt	timestamp with timezone
	updatedAt	timestamp with timezone

Figure 7: ER Diagram

Responsibilities

During this project different tasks have been split between different team-members. Most of the time, we decided to split things to work individually, but there have also been some splits where two members have chosen to collaborate and have a part of the project where both are equally responsible. Additionally, we have worked together a lot on the lab sessions, and while co-authors often have been added to the git commit it has been quite common to collaborate on problem-solving which has not always been noted in the git repository.

Andreas Maass Magnusson

- Responsible for the create post-page, including fullstack: frontend, API, router and service layer.
- Responsible for user-page, including fullstack: frontend, API, router and service layer. Sara has made the frontend for likes and number of posts.
- Added topics to posts full stack.
- Added filter posts by topic functionality full stack.
- Worked on tests both backend and frontend, mostly related to the functionality above.

Marko Horvat

- Responsible for the Create user frontend
- Implemented user authentication on both the frontend and backend, ensuring that only logged-in users can access API endpoints and view restricted pages.
- Added logout functionality
- Added database models
- Responsible for creating initial user service and its tests
- Responsible for updating backend tests and Profile page frontend tests
- Created an ER Diagram for the report

Sara Hedberg

- Has had main responsibility for the report, including structure and the writing of the sections introduction, User manual, use cases as well as parts of responsibilities and design.
- Main responsibility for the design of the project, including creating and updating the API design document, creating the icons, colorscheme (including checking contrast for accessibility purposes, adding the favicon
- Responsible for the contents of the README file.
- Responsible for most of the feed-page and its tests
- Responsible for the html mock-up of the login-page together with Sebastian.

- Responsible for the initial user router and its tests
- Responsible for the frontend of the profile page's like and post counter
- Created the initial tests for create user frontend
- Responsible for the updating of tests after the addition of sessions
- Responsible for the addition of the author name to posts together with Marko.

Zebastian Björkqvist

- Responsible for doing the initial setup for each lab step
- Responsible for adding like functionality to posts full stack
- Responsible for creating initial login component, header and page
- Responsible for fixing most tests after authentication and database were implemented
- Responsible for maintaining and updating structure of codebase
- Responsible for the initial user router and its tests
- Responsible for the html mock-up of the login-page together with Sara