

# Diary Web Application

An Intuitive Platform for Personal Journaling

DAT076 Web Applications

Project Groups 2

Martin Berntsson  
Melissa Mujanovic  
Narimaun Novak  
Tyra Olofsson  
Fredrik Ström

# 1. Introduction

Link to Git repo

[Branches · JMartin3872/DAT076WebApplications](#)

In this course, we have designed and implemented a simple and intuitive diary application designed to capture and organize your thoughts with ease in a clean and user-friendly interface. Whether you're looking for a quick way of jotting down ideas, writing focused goal-tracking journals, or simply reflecting on your day, this application provides its users with an all-in-one experience for creating, managing, and ultimately providing a reliable way of storing their diaries.

The app starts by greeting the user to a secure login page, where users can register, manage their accounts, and login with their username and password. Once logged in, users are taken to their page where returning users are presented with their list of diaries. New users will find this list empty; an exciting clean canvas waiting to be filled with memories, thoughts and plans. In addition to gaining access to a selected diary, users are also presented with several options to manage their diaries.

Once a diary has been selected, the users are redirected to the page of the corresponding diary. Here, users are able to create, edit, and delete entries in their diary. Again, users of newly created diaries will find this page empty at first, hinting at an exciting fresh start at the first of hopefully many brilliant ideas, exciting recipes, or loving memories ready to be preserved for years to come. To easily navigate, users are invited to pin their favorite notes or sort them by date. Switching between diaries is effortlessly achieved with a straightforward back button, keeping the focus on what's important - content creation.

## 2. Use cases

### Use cases for login and registration

A user should be able to...

- Change their password to a new one.
- Delete their user account if they decide to leave our beloved diary website.

### Use cases for diaries

A logged-in user should be able to...

- Create a new diary by giving it a title.
- Rename a diary they have created.
- Delete a diary they have created.
- Sort their list of diaries by “newest” or “oldest” first.

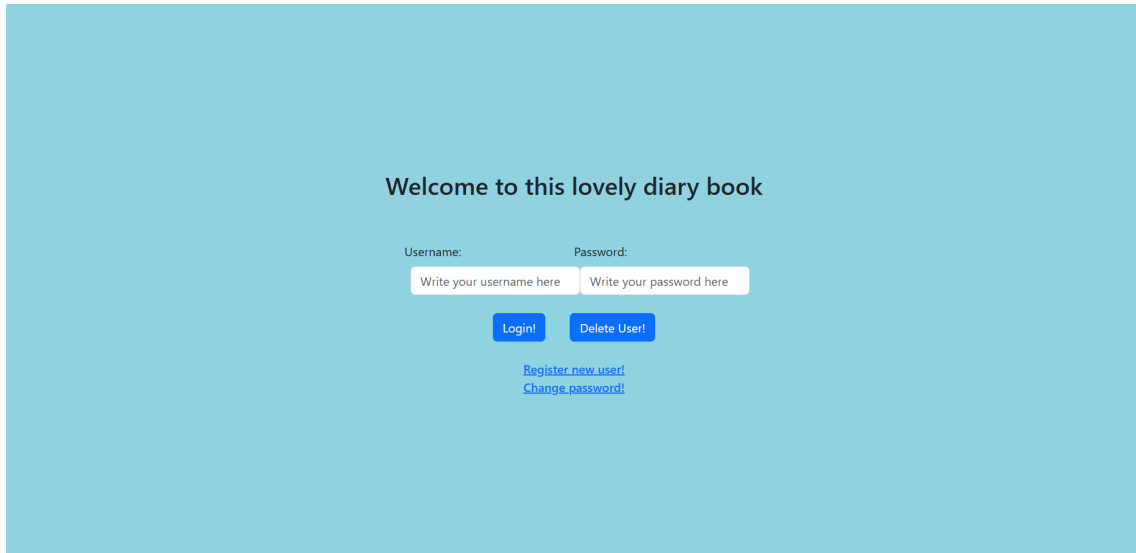
### Use cases for diary entries

In a diary, a logged-in user should be able to...

- Create a new entry in the diary.
- Edit an entry in the diary.
- Delete an entry in the diary.
- Pin entries in the diary to a “Pinned entries” list. Unpin previously pinned entries.
- Sort the list of entries and pinned entries by “newest” or “oldest” first.
- Rename the current diary while in the entry view.

### 3. User manual

The website starts in a logged-out state. To be able to use this website, the user needs to make a new account by entering their newly chosen username and password, this is done by clicking on the register new user link.

The image shows a login page with a light blue background. At the top, it says "Welcome to this lovely diary book". Below this, there are two input fields: "Username:" and "Password:". The "Username:" field has a placeholder text "Write your username here" and the "Password:" field has a placeholder text "Write your password here". Below these fields are two buttons: "Login!" and "Delete User!". At the bottom, there are two links: "Register new user!" and "Change password!".

Welcome to this lovely diary book

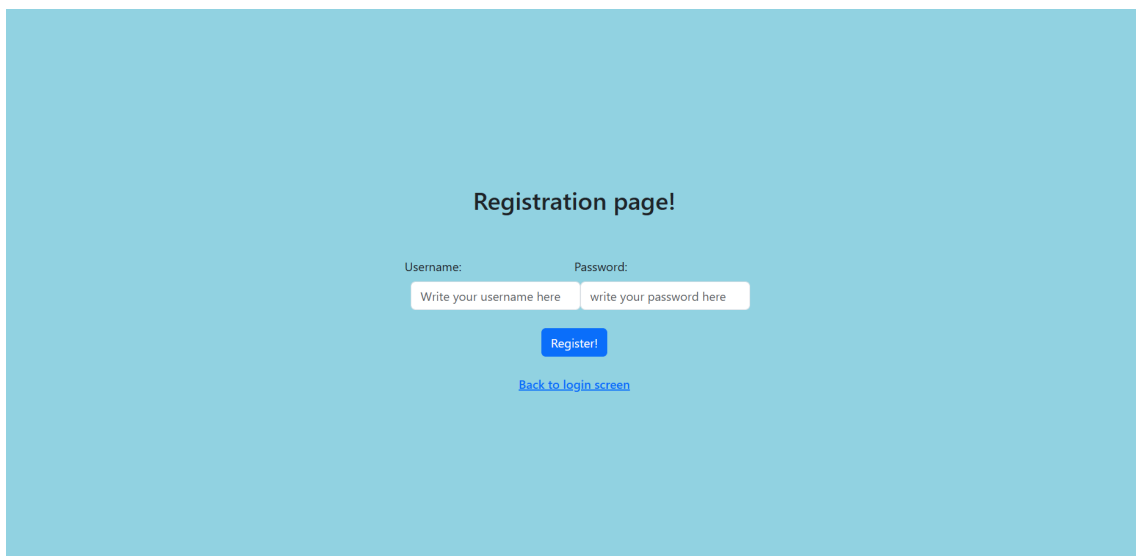
Username: Password:

Write your username here Write your password here

Login! Delete User!

[Register new user!](#)  
[Change password!](#)

*Image 1: The login page.*

The image shows a registration page with a light blue background. At the top, it says "Registration page!". Below this, there are two input fields: "Username:" and "Password:". The "Username:" field has a placeholder text "Write your username here" and the "Password:" field has a placeholder text "write your password here". Below these fields is a button: "Register!". At the bottom, there is a link: "Back to login screen".

Registration page!

Username: Password:

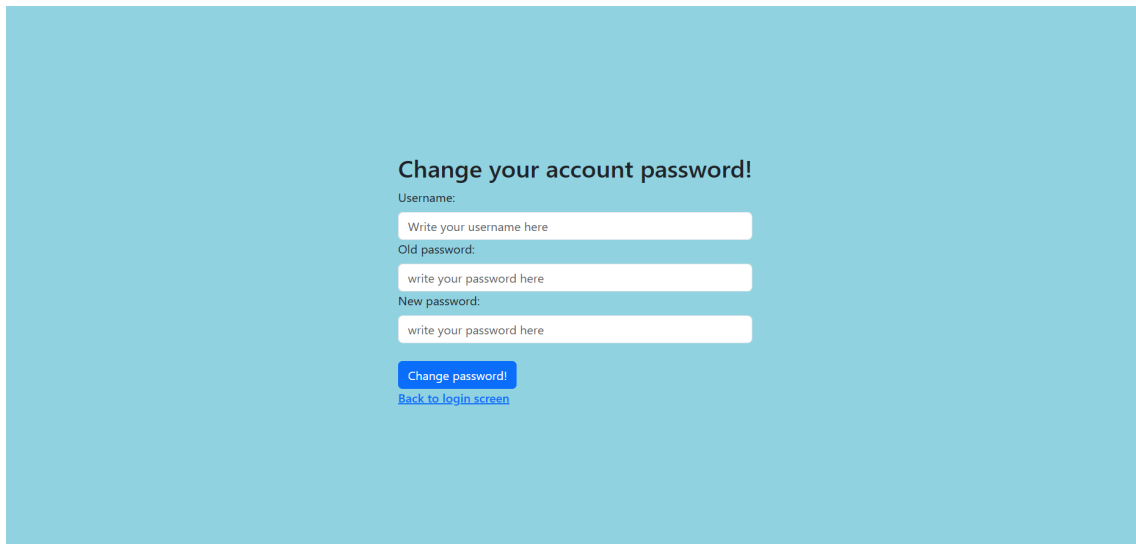
Write your username here write your password here

Register!

[Back to login screen](#)

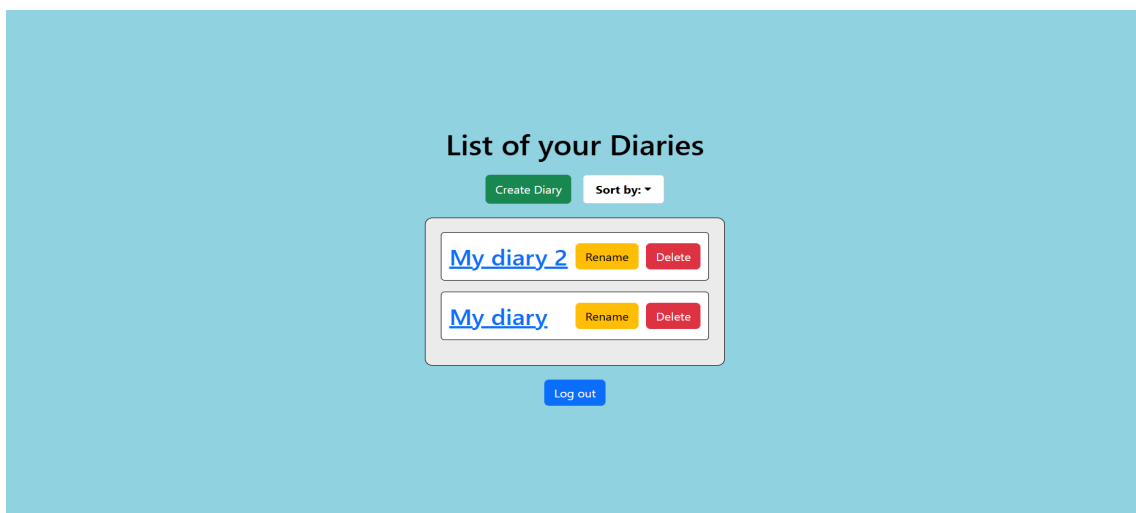
*Image 2: The registration page.*

After successfully registering a new account, the user is taken back to the main login page, where it is possible to use the given username and password and login to access their diaries. It is also possible to press the “Delete User!” button to delete a user completely from the website database, as well as delete all the diaries that are connected to the user. This is not a reversible action and should be done with caution. There is also the option of changing the password of an existing user.



*Image 3: The “change password” page.*

When the user has successfully logged in, a list of the user's diaries is displayed. If the user has just registered, the list will be empty. The user creates a diary by pressing the green button named “Create Diary”, a modal window appears where the user can enter a name. By clicking on the blue button “Create” in the modal window, the diary gets created and appears in the list of diaries. If the user changes their mind about creating a diary, the “cancel” button next to the create button, can be clicked.



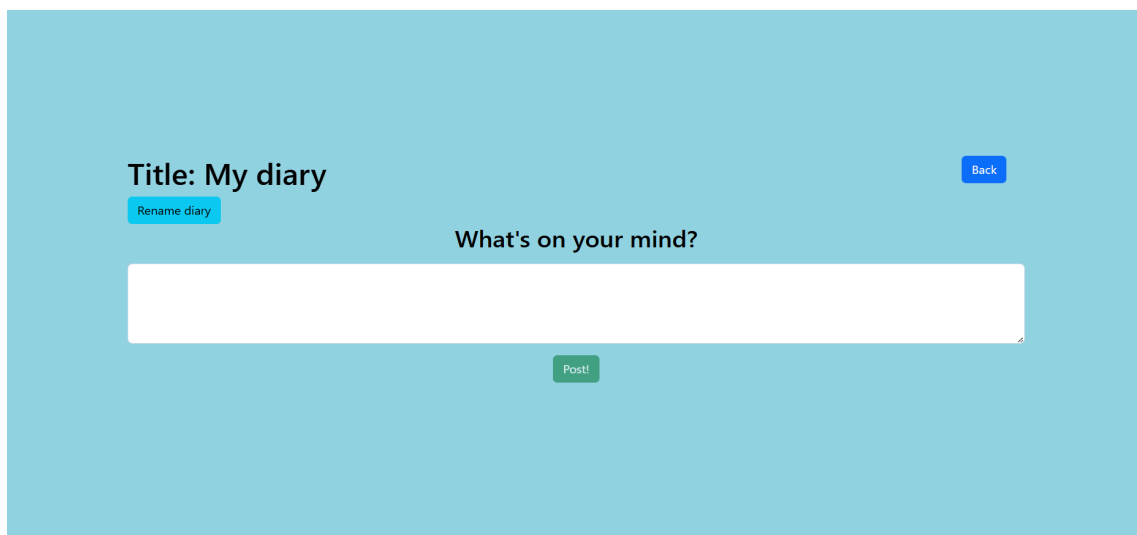
*Image 4: The diary list page for a logged-in user where two diaries have been created.*

Next to each diary of the user, there are two buttons “Rename” and “Delete”. If the user wants to rename a diary, they can click the rename button, and a modal window will appear with a text input field that has the placeholder “Enter diary title”. The user can then choose to press the blue button “rename”, or the “cancel button”. The new title of the diary cannot be the same as the previous one. If the user wishes to delete a diary by pressing the red “delete” button, a confirmation dialog is displayed on the top of the application window: “Are you sure you want to delete this diary?”. If the user clicks “okay”, the diary will be deleted from the users list. This is an action that cannot be undone and the diary along with its entries will be permanently deleted from the database.

Next to the “Create Diary” button is a white button “Sort by” that acts as a dropdown menu. By clicking on this button the user can choose how their list of diaries should be sorted. The diaries can be sorted by “Newest” or “Oldest”. The list is by default sorted by “Newest”, meaning that the most recently created diary is placed at the top of the diary list.

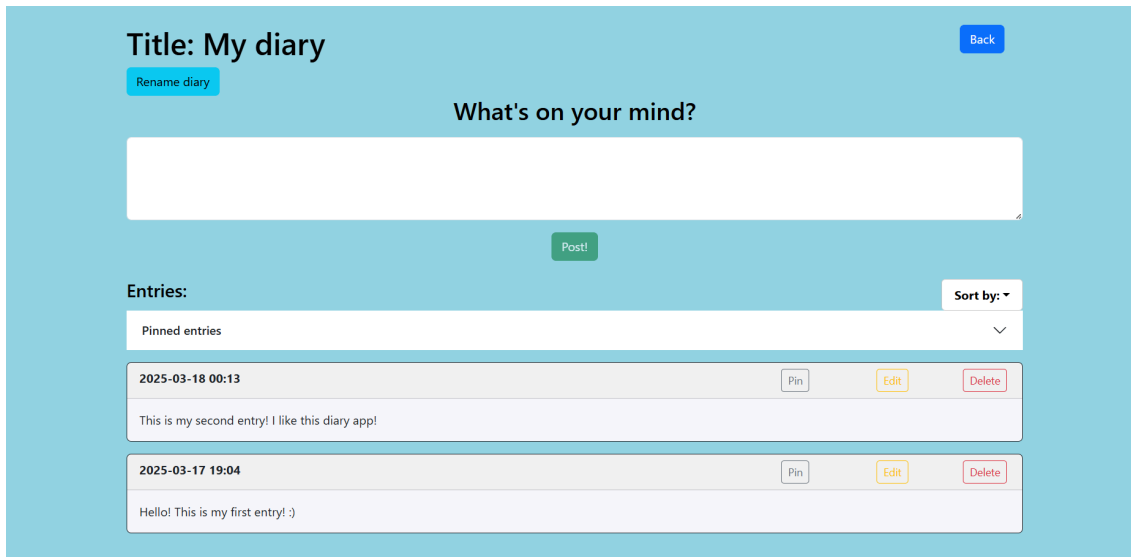
At the bottom of the diary list page, a blue button “Log out” is placed. By pressing the button, the user is placed back into a logged-out state and gets directed back to the login page which acts as the homepage for the application.

In a logged-in state, when having selected a diary by either clicking on a preexisting one in the user’s list of diaries or by creating a new one and clicking on it, the user is taken to the selected diary page. Here, the user is able to view previously created entries or create new entries by clicking in the empty text field and typing on their keyboard. To post a new entry, the user either clicks on the green “Post!” button or simply presses the “Enter” key on the keyboard. The newly created entry now appears under the text field in a new list called “Entries”. Each new entry is logged with a timestamp corresponding to the time of its creation, visible to the user in each entry’s date field.



*Image 5: An opened diary “My diary” without any entries in it.*

By default, the list of entries is sorted by “newest first”, meaning each new entry will appear at the top of the list of entries. The user may choose to sort the list by “oldest first”, simply by clicking the “Sort by” drop-down menu at the top right corner of the list of entries and selecting “Oldest”.



*Image 6: The same opened diary with two added entries.*

The user can edit a previously posted entry, by clicking the yellow “Edit” button of the corresponding entry. A popup window appears with a text field containing the entry’s current text. The user may now edit the entry’s text and save the changes by either clicking the “Save changes” button or by pressing the “Enter” key on the keyboard. A successful edit is followed by the message “Entry edited successfully!”. The edited entry retains its previous timestamp and the edited text is immediately visible to the user in place of the old entry text.

The user may delete any number of entries in a diary, by clicking on the red “Delete” button of the corresponding entry. The message “Are you sure you want to delete this entry?” is displayed and the user is prompted to confirm by clicking “OK”. A successful deletion is followed by the message “Entry deleted!”.

The user can pin entries. To pin an entry, the user clicks the gray “Pin” button of the corresponding entry. The entry will now also appear in a separate list called “Pinned entries”, located above the rest of the entries. The user may unpin a previously pinned entry by clicking the “Unpin” button of the corresponding entry, either in the “Pinned entries” list or in the normal list of entries.

The user may rename the current diary, by clicking on the light blue “Rename diary” button located in the top left corner just under the diary title. A popup window appears with a text box containing the current title of the diary. The user may change the title by clicking and typing in the text field. To save the new diary title, the user either clicks “Save changes” or presses the “Enter” key on the keyboard.

To return to the user’s list of diaries, the user may press the “Back” button, located at the top right corner of the page.

## 4. Design

The design of our application follows the Model-View-Controller software pattern where the view and controller functionality lies on the client side while the model is hosted on the server side. The application's model structure consists of two main parts: Users and diaries. A user consists of a unique username and password, while a diary contains information about the diary, such as id, title, and which user owns said diary as well as a list of entries. Each entry also holds information about itself such as id, the entry text and if it is pinned or not.

### 4.1 Frontend

The application's frontend is dependent on several frameworks and libraries which is listed below:

- **React (React.js):** A front-end JavaScript library used for facilitating the building of dynamic user interfaces.
- **React Bootstrap:** Another front-end framework built for use in combination with React. Provides a powerful grid system and several customizable components.
- **Vite:** A build tool that also enables the use of local servers.
- **Axios:** A JavaScript library used for making HTTP requests from nodes.
- **React Router (react-router-dom):** A JavaScript library used for navigation and rendering multiple views without reloading pages.
- **Jest:** A popular JavaScript testing framework.

#### 4.1.1 Login

The front end of login is written in the React framework and it consists of three components, **LoginPage**, **RegisterPage**, and **ChangePasswordPage**. We start with the LoginPage, this page is the first page that comes up when the user enters the diary application. The purpose of this page is to help the user log in, delete their user account, leave the diary app, change the user password, or register a new user, which the latter two are hyperlinks to two other components that are named above. It is also important to mention that all three components build the screen through HTML, in which the react-bootstrap was used as a helpful framework.

##### *LoginPage*

Contains two states, username and password, in which it saves the typed password and username that the user types. These states are later on used as parameters in the function signIn. signIn is a part of the frontend API, which makes an Axios post call to the backend URL. LoginPage, upon the user succeeding to log in with the correct credentials, calls on "useNavigate" from react-router-dom to navigate to another component located in the diary list, called diaryListComponent, sending the username state and the returned list of diaries from the signIn function call.

##### *RegisterPage*

The other component that is closely related to login, RegisterPage. The user, upon pressing on the registration hyperlink on the login page, will be directed to the registration page of the app. The only



purpose of this component is to get the username and the password of the new user, call the function `registerNewUser`, and get back to the main page (login page). This is done by saving the username and password in two states and passing them in the mentioned function. `registerNewUser` is a part of the frontend API as well, meaning that it makes an Axios post call to the backend URL.

### *ChangePasswordPage*

Lastly, `ChangePasswordPage` has the sole purpose of changing the password of an existing user to a new password. It consists of three states, `username`, `oldPassword`, and `newPassword`. The changing of the password is done by passing all these states to an API function, `changePassword`. This function also calls on an Axios post to the backend URL. After changing the password successfully, the user will be redirected to the main login page of the app.

## 4.1.2 Diary List

The diary list page “`diaryListComponent`” is responsible for displaying the list of diaries to the logged in user. The page contains a button for creating a new diary, when the button is clicked the user is prompted to give the diary a title. The page also displays a list with the diaries the logged in user has created. The diaries can both be renamed and deleted and the list contains buttons for performing both of these actions. If the user wants to rename a diary the new name cannot be the same as the previous one.

- “**diaryListComponent**”:
  - Purpose:
    - This component manages the list of diaries for a logged in user. It allows the user to create, rename, and delete diaries. There is also a dropdown menu that allows the user to sort their diaries from “oldest” or “newest” (based on their diary id). A logout button is also placed under the list of diaries which clears the diary list and directs the user back to the login/home page.
  - Props: None
  - State:
    - **diaryList: Diary []**: Stores the list of the logged-in user’s diaries.
    - **username**: Stores the logged-in username.
    - **showModal**: Its boolean value is used to control the visibility of the create/rename modal.
    - **diaryTitle**: Stores the diary title input for creating/renaming the diary.
    - **selectedDiary**: Stores the diary the user has chosen to rename.
    - **editMode**: Its boolean value is used to determine if the modal is for creating a diary (modal = false) or renaming a diary (modal = true)
    - **sortByNewest**: Its boolean value is used to sort the diaries by newest (true) or oldest (false).

- Calls to backend:
  - **createDiary(username, diaryTitle):** Called in `handleSaveDiary` when `editMode === false`. Creates a new diary for the logged-in user. If successful, the new diary is added to the logged-in user's `diaryList` and the database.
  - **renameDiary(username, diaryId, newTitle, false):** Also called in `handleSaveDiary` but when `editMode === true`. Renames an existing diary for the logged in user. If successful, it updates the `diaryList` with the renamed diary. The database is also updated with the new title of the diary.
  - **deleteDiary(username, diaryId):** Called in `handleDeleteDiary(diaryId)`. If successful it deletes an existing diary from the logged in user's diary list and the database.

#### 4.1.3 Diary

When a diary has been selected in the overview of a user's diaries, the user is redirected to `/diary`. The `/diary` page represents the content of any given diary and contains several different components that have different responsibilities when it comes to displaying content and allowing the user to send requests to the backend for changing said content:

- **“diaryComponent”:**
  - Purpose:
    - The main component for the `/diary` page. It contains a header for displaying the diary's title as well as a button for changing the title, it also contains a back button that lets the user navigate back to `/List-of-diaries`. The component is also the parent of the “`diaryInputComponent`” and the “`entryListComponent`”. “`diaryComponent`” provides each of its two children with a set of callback functions for making calls to the backend through the API at specified events.
  - Props: None.
  - State:
    - **diary:** The diary which contents should be displayed on the page.
    - **username:** The user's username.
    - **showRename:** Boolean value used to determine if the modal for changing the diary's title should be shown.
    - **newTitle:** String value representing the new title for a diary.
  - Calls to backend:
    - **getUserDiaryRequest:** Gets the list of user diaries before navigating to `/List-of-diaries`.
    - **renameDiary:** Request to rename the current diary after entering a new title in the “Rename diary” modal and saving changes.

- **“diaryInputComponent”:**

- Purpose:
  - The component responsible for reading user input, contains a header, textarea and a post button. Makes a call to the backend to add an entry when the user has entered some text in the textfield and pressed the post button.
- Props:
  - **onAdd:** A function for making a call to the backend when the post button is pressed.
- State:
  - **entryText:** The text entered in the textarea.
- Calls to backend:
  - **addEntry:** Called through the provided callback function from “diaryComponent”.

- **“entryListComponent”:**

- Purpose:
  - To display and order a diary’s list of entries. Contains a header, dropdown button for ordering entries, a list of pinned entries in an accordion component and a list of all the diary’s entries.
- Props:
  - **myDiary:** The current diary.
  - **onEntryEdit:** Callback function for editing an entry’s text, to be used by each “entryComponent”.
  - **onEntryDelete:** Callback function for deleting an entry, to be used by each “entryComponent”.
  - **onTogglePin:** Callback function for toggling the pinning of an entry, to be used by each “entryComponent”.
- State:
  - **diary:** The diary which contents should be displayed on the page.
  - **sortByNewest:** Boolean value used to determine if list of entries should be ordered by newest first or oldest first.
- Calls to backend: None

- **“entryComponent”:**

- Purpose:
  - This component represents an individual entry from a diary’s list of entries. It is a card where the header contains the time for when the entry was created as well as three buttons for pinning, editing or deleting the entry. The card’s body contains the entry text.

- Props:
  - **myEntry:** The corresponding entry from the diary's list of entries.
  - **onEdit:** Callback function for editing the entry's text
  - **onDelete:** Callback function for deleting the entry.
  - **onTogglePin:** Callback function for toggling the pinning of the entry.
- State:
  - **showEdit:** Boolean value used to determine if the modal for editing the entry's text should be shown.
  - **editedText:** String value holding the new text for an entry which should be edited.

## 4.2 API and backend

The backend of the application also depends on a few frameworks and libraries for its functionality:

- **Express.js** (and **express-session**): A framework used for building RESTful APIs.
- **CORS:** A JavaScript library that is used to configure how cross-origin requests should be handled.
- **Bcrypt:** A JavaScript library used for hashing passwords.
- **Dotenv:** A JavaScript library used for loading environment variables such as secrets.
- **Sequelize:** A JavaScript library used for object-relational mapping to a Postgres database.

### 4.2.1 Login

The backend of the login consists of three parts, the model, the service, and the router. The model of course is the data that is sent back and forth. The service is the meat and bone of the calculations part. It takes care of registering a user for the first time, trying to log in as a user, changing the password of an already existing account, and lastly, deleting an account altogether with corresponding diaries. The login service is in contact with the SQL database with the help of the Sequelize framework.

The router connects the frontend request to the backend. It has a login base URL (`../login/...`) and it accepts five types of requests.

The login service also uses the Bcrypt library for hashing and encrypting the user password before saving it in the database. This is also used when decrypting the user's password for logging-in purposes.

**"../login/register"** is an endpoint that takes care of registering a new user. Through a POST request, It calls the appropriate login service layer function, `registerUser`, for this purpose. The body of this endpoint consists of a username and password in string format.

This endpoint sends a “201 created” upon succeeding to register a user, or it will send a status code “400” if the password or username is in an incorrect format, and it sends a “500” if the code catches an unexpected error.

**“../login/”** is another endpoint in which it facilitates the login part of the backend. The username and password in string format are in the body of the request payload, and this endpoint, through a POST request, calls the appropriate login service layer function, tryLogin, to log in as a user.

In the making of backend endpoints, sessions were also used. This is done with the logic that, if a user succeeds in logging in to the system, a session is given to this user that has the same name as the username of the account. This endpoint sends a “200” status code upon succeeding, as well as returning the result of calling the function from the service layer, which is a list of diaries that belong to the user. A status code of “400” will be returned in case the username or password format is not correct, and a “500” if any unexpected error is caught.

**“../login/changepassword”** endpoint, facilitates changing the password of the user by making a PATCH request, by calling the changePassword function in the login service layer. Here, the body consists of the username, oldPassword, and the newPassword in string format.

This endpoint sends a “201” status code upon successfully changing the password of the user, a “400” will be returned in case the username or password format is not correct, and a “500” if any unexpected error is caught. A status code of “401” will be sent if the credentials are wrong.

**“../login/logout”** endpoint lets a user log out of the application through a POST request. This endpoint makes sure that it is possible to delete the session that has been assigned to the specific user. If logged out successfully, the status code “200” will be sent as the response. If no active session is found, a status code “400” will be sent, and if logging out fails, a “500” status code will be sent.

**“../login/deleteuser”** endpoint lets a user delete their account entirely through a POST request. This means, besides deleting the credentials, the endpoint will access the database, through the service layer, and delete all the diaries that the account is the owner of. The body consists of a username and password in string format.

A status code of “200” is sent if this is successful, otherwise “401” for wrong credentials, or “500” if any unexpected errors happen during runtime.

#### 4.2.2 Diaries

The backend logic for the diaries is divided into three different layers; “Model”, “Router”, and “Service”.

The “Router” layer (server/router/diary.ts) is responsible for authenticating and authorizing users as they try to create, delete, or rename the diaries. The layer controls the validity of the incoming request by checking the session username. If the incoming request to either create, rename or delete a diary comes from a user who is not logged in, the request is denied with a “401 Unauthorized” error. The request is also denied with the same error if a user tries to rename or delete a diary they are not the owner of. If the user is logged in and tries to rename or delete a diary they are in fact the owner of, the request is accepted and forwarded to the service layer, and a “200 OK” response is sent to the client.

The “Service” layer (server/service/diary.ts) is responsible for handling the logic related to the diaries along their entries. The service layer interacts with the router layer and the database to retrieve, store, and modify the data of diaries. Whenever a client in the front end of the application requests to create, delete, or rename a diary the task first goes through the router layer. If the request is successful it is sent to the service layer where the request is processed and the database is interacted with depending on the request. A response is thereafter sent to the API.

The “Model” layer describes the data structure of a diary which is used by the server and sent to the front end.

The database contains a table named “diaries” which is used for storing information related to the diaries themselves. The diary database is implemented using Sequelize and each diary is stored with their diaryId, title, and owner.

Each request for the diary part of the API is presented below:

- Verb: **createDiary**
  - Endpoint: A POST request to /diary/creatediary
  - Body: {username : string, title : string}
  - Responses:
    - A 201 response containing the newly added diary upon a valid request.
    - A 401 response if the request’s session is missing or invalid.
    - A 400 response on a bad request.
    - A 500 response upon catching any exception.
- Verb: **deleteDiary**
  - Endpoint: A DELETE request to /diary/deletediary
  - Body: {username : string, diaryId: number}
  - Responses:
    - A 200 response containing the remaining diaries after successful deletion.
    - A 401 response if the request’s session is missing or invalid.
    - A 400 response on a bad request.
    - A 500 response upon catching any exception.
- Verb: **renameDiary**
  - Endpoint: A PATCH request to /diary/renamediary
  - Body: {username : string, diaryId: number, newTitle : string, onlyTitle: boolean}
  - Responses:
    - A 200 response containing the updated diary list upon a valid request.
    - A 401 response if the request’s session is missing or invalid.
    - A 400 response if the diary is not found or if the title is already taken.

- A 500 response upon catching any exception.
- Verb: **getListOfDiaries**
  - Endpoint: A GET request to /diary/userdiaries
  - Body: {username : string}
  - Responses:
    - A 200 response containing a list of the user's diaries upon a valid request.
    - A 401 response if the request's session is missing or invalid.
    - A 500 response upon catching any exception.

#### 4.2.3 Entries

As with the other parts of the backend, the logic for handling entry requests is divided into three different layers: “Router”, “Service” and “Model”. These layers are used together with communication with the database to serve each request which arrives from the client side.

The “Router” layer controls the validity of incoming requests. If a request is valid it is forwarded to the service layer and a 2xx response is sent to the client after the router layer receives a successful result from the service layer. In case of a client error a 4xx response is sent instead or if there is a server error the router layer responds with a 5xx response.

The “Service” layer handles all calculations and modifications of data related to creating, editing and deleting entries. It does so by converting the data the server received to a format which can be stored in the database and vice versa. The result of any operation made by the service layer is then returned to the router layer.

The “Model” layer describes the data structure of an entry which is used by the server and sent to the front end.

The database contains a table named “entries” which is used for persistent storage of information related to entries, namely: Their ID, which diary they belong to, their text, if the entry is pinned, and the time the entry was created.

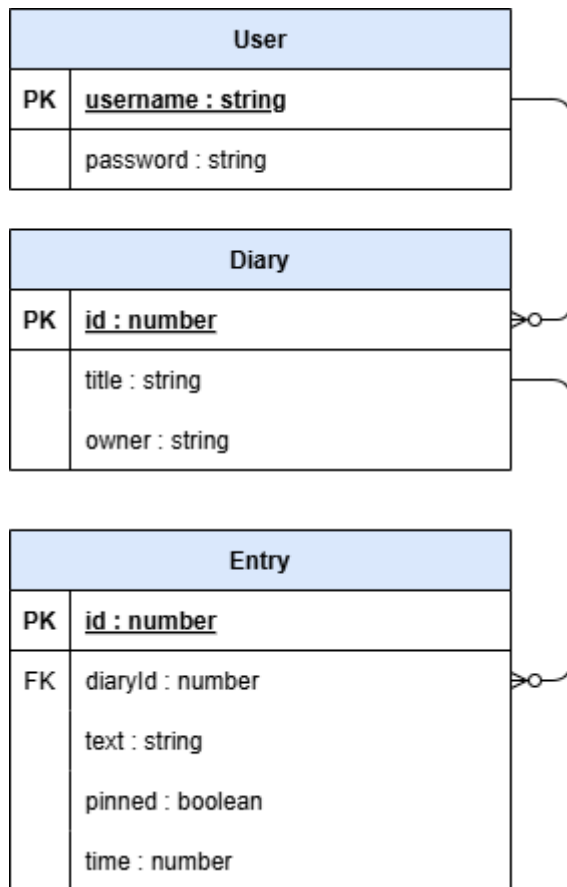
Each request for the entry part of the API is presented below:

- Verb: **addEntry**
  - Endpoint: A POST request to /diary/addentry
  - Body: {username : string, diaryId : number, text : string}
  - Responses:
    - A 201 response containing the newly added entry upon a valid request.
    - A 401 response if the request's session is missing or invalid.
    - A 400 response on a bad request.
    - a 500 response upon catching any exception.

- Verb: **editEntry**
  - Endpoint: A PATCH request to /diary/editentry
  - Body: {username : string, diaryId : number, entryId : number editedText : string, pinned : boolean}
  - Responses:
    - A 200 response containing the updated list of entries.
    - A 401 response if the request's session is missing or invalid.
    - A 400 response on a bad request.
    - a 500 response upon catching any exception.
- Verb: **deleteEntry**
  - Endpoint: A DELETE request to /diary/deleteentry
  - Body: {username : string, diaryId : number, entryId : number}
  - Responses:
    - A 200 response containing the list of remaining entries.
    - A 401 response if the request's session is missing or invalid.
    - A 400 response on a bad request.
    - a 500 response upon catching any exception.



#### 4.4 Entity-relationship diagram for database



## 5. Responsibilities

### Martin Berntsson

- Has together with Fredrik Ström designed, implemented, and tested all logic related to the diaries' entries. Includes the following parts:
  - **Frontend:** The view and controller of an opened diary (localhost:5173/diary/)
  - **Backend:** Routing, service, and model for entry requests.
  - **Database:** Handling reading and writing to the database model for entries.
- Setup and initialization of the project's database.
- Wrote the design section of entries in the report.

### Fredrik Ström

- Has together with Martin Berntsson designed, implemented and tested all logic related to the diaries' entries. Includes the following parts:
  - **Frontend:** The view and the controller part of an opened diary (localhost:5173/diary/)
  - **Backend:** Routing, service, and model for entry requests.
  - **Database:** Handling reading and writing to the database model for entries.
- Wrote the introduction section, use case section concerning entries as well as user manual section concerning entries in the report. Wrote the README.MD file in the repo.

### Tyra Olofsson

- Has together with Melissa Mujanovic, designed, implemented, and tested all logic related to the diaries themselves. This includes the following parts:
  - **Frontend:** The view and controller of the list of diaries
  - **Backend:** Routing, service, and model for diary requests.
  - **Database:** Handling reading and writing to the database model for diaries.
- Wrote the sections "Use cases" (2), "Diary List" (4.1.2), and "API and backend" (4.2.2) concerning the diaries in the report. Added images from the application to the user manual (3).

## Melissa Mujanovic

- Has together with Tyra Olofsson, designed, implemented, and tested all logic related to the diaries themselves. This includes the following parts:
  - **Frontend:** The view and controller of the list of diaries.
  - **Backend:** Routing, service, and model for diary requests.
  - **Database:** Handling reading and writing to the database model for diaries.
- Wrote the section “User manual” (3) in the report.

## Narimaun Novak

- Has designed, implemented, and tested all logic related to the login side of the application. This includes:
  - **Frontend:** The view and controller of the login, register, and change password page.
  - **Backend:** Routing, service, and model for login, logout, register, change password, and delete user account requests.
  - **Database:** Handling reading and writing to the database model for users.
- Wrote the design section of the login side in the report.