DAT070 - Final report

Carl Toreborg, Casper Hansen, Erik Persson

March 2025

1 Introduction

The app that our group has developed is a Study app. A companion for organizing you studies. Here you can see what you need to do with the todo-page, find your notes from lectures in the notes-page and even connect those to the todos to track where you got the task from. The notes also includes a short preview that is automatically collected from the document the you are uploading. The app also has an integrated pomodoro timer to help you focus on the tasks you need to study for.

The source code can be found on github: Labbgrupp42Dat076

Contents

2 User manual

Follow these simple steps to get the app running on your local machine.

2.1 Installation Instructions

1. Clone the Repository First, clone the repository to your local machine using Git:

```
git clone https://github.com/Labbgrupp42Dat076/labs.git
```

2. Install Dependencies After cloning, navigate to the project directory and install the dependencies:

```
cd labs
npm install --force
```

2.2 Docker Installation

Ensure you have Docker installed, as it will be used for the database. If not, follow the installation instructions on Docker's official website: `https://www.docker.com/products/docker-desktop`.

2.3 Database Setup

Run the following command to start a PostgreSQL database:

```
docker run --env POSTGRES_USER=<username_in_your_env_file>
--env POSTGRES_PASSWORD=<password_in_your_env_file>--publish
5432:5432 --name web_apps_db --detach postgres:17
```

2.4 Configuration

Before running the app, you will need a `.env` file in the server directory with the following content:

```
SESSION_SECRET=your_session_secret_here
DATABASE_URL=your_postgres_url_here
DATABASE_USER=your_postgres_username_here
DATABASE_PASSWORD=your_postgres_password_here
```

Either you can create a new file or copy the already existing `.env.example` file in the server directory

2.5 Running the Application

Once the dependencies are installed, start the application. To start the entire application from the root directory:

```
npm run dev
```

There are also scripts to start each part of the application individually:

To start only the PostgreSQL database:

```
npm run database
```

To start the server, navigate to the `server` directory and run:

```
npm run dev
```

To start the client, navigate to the `client` directory and run:

```
npm run dev
```

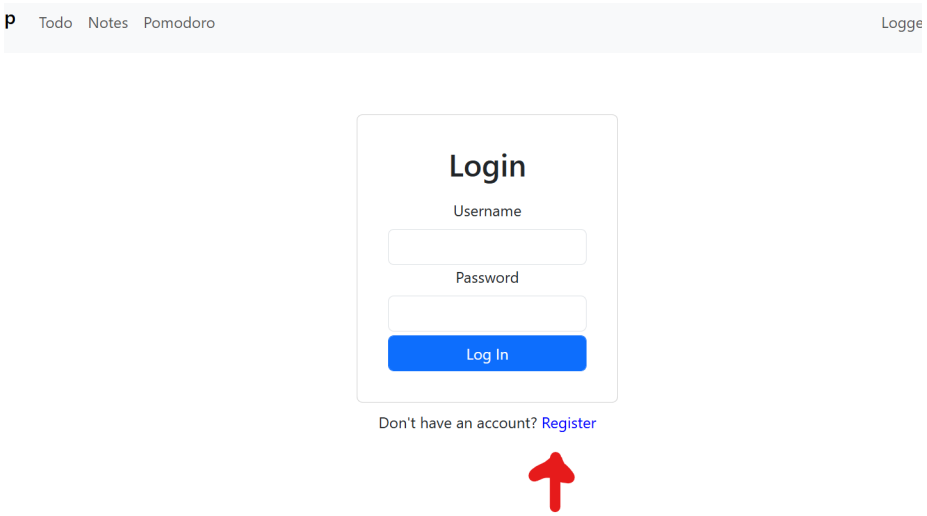
2.6 Using the Application

After completing the installation and configuration, you can start using the app. Open your browser and navigate to the local client URL (usually `http://localhost:5147`).

You can interact with various features provided in the app, such as creating study tasks, setting study goals, or tracking progress.

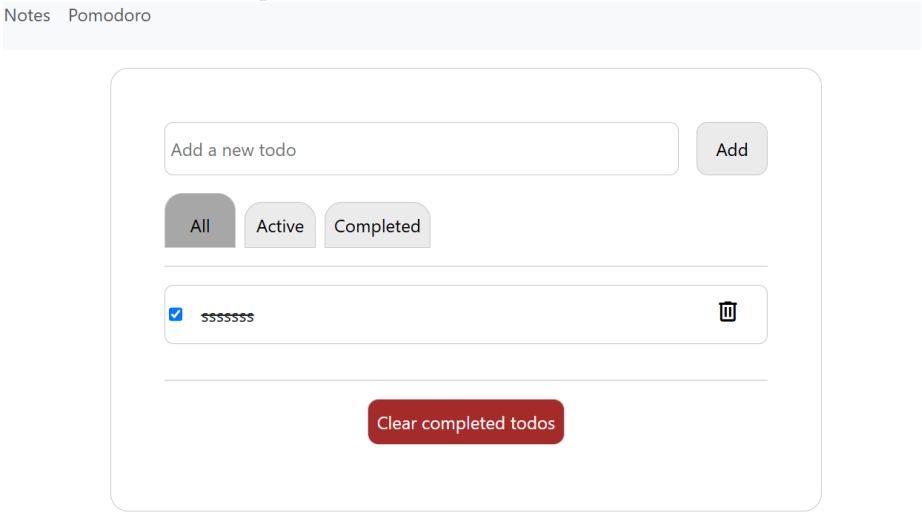
2.6.1 Log in

Before you can use the features in the the application you have to register and then log in. You do this by clicking "register here" and inputting your desired details. A successful register will also log you in

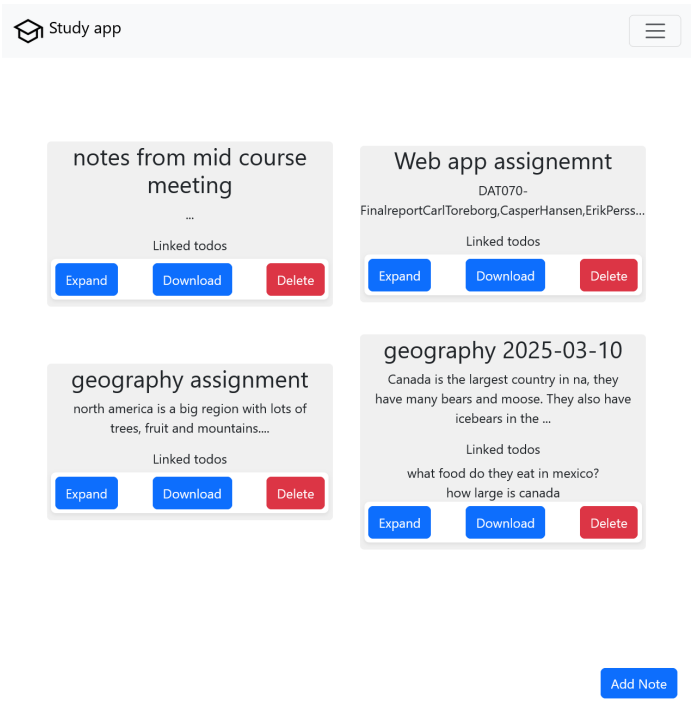


2.6.2 Todo

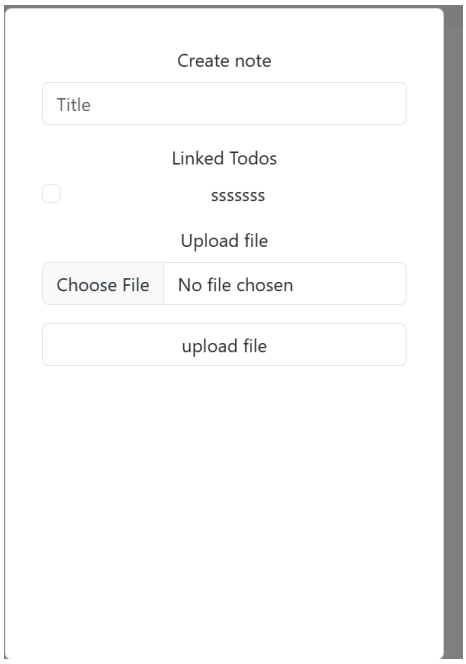
In the todo page you can: add a todo, check it off and delete it, There is also the ability to filter the todos using the tabs at the top. The button "clear completed todos" will delete all completed todos.



2.6.3 Note



The notes main page shows you a list of your notes and a small preview of whats inside them. If it is a document containing text such as a pfd or tex document a text preview will appear. You can delete or download the files the notes are connected to. You can also expand the text area a bit to see more of the preview. These note previews also show the todos that are connected to that note.



Notes can be added using the add note form. Here you can add a title, link todos to the note and upload a file.

2.6.4 Pomodoro

A pomodorotimer is a timer that switches between a set study time and a shorter time for break. In this app the standard time of 25 minutes for study and 5 minute break is set to default. The contents of the app is divided into two blocks, the right (mobile top-) block is the timer block where you can start/stop the timer but also reset the studytimer and force a break.

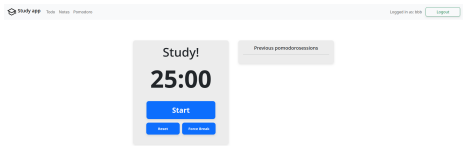


Figure 1: Caption

To start the timer you press the start button as shown in figure 1.

When pressing start you also start a session. That session lasts until you leave the page or app. The previous sessions are displayed in the left (mobile bottom-)

part in a descending list. This list will only display the latest 5 sessions. See figure 2

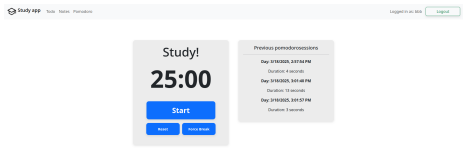


Figure 2: Caption

2.7 Testing the Application

To ensure everything is working properly, you can run tests for both the frontend and backend.
To test both parts together:

```
npm run test
```

To test them separately: Frontend tests:

```
npm run test-client
```

Backend tests:

```
npm run test-server
```

3 Design

3.1 Frontend Components

3.1.1 AddNoteOverlay

Purpose: A component for adding notes, uploading files, and linking them to existing todos.

Props

Prop Name	Type	Description
title	string	The title of the note to be created.
todos	TodoData[]	The list of todos that can be linked to the note.
fileId	number — null	The ID of the uploaded file.

State

State Variable	Type	Description
fileId	number — null	Stores the uploaded file ID.
todos	TodoData[]	Holds the list of fetched todos from the server.

Backend Calls

API Endpoint	HTTP Method	Purpose
/todos	GET	Fetches all todos from the server.
/note	POST	Adds a new note with selected todos and an optional file.
/file	POST	Uploads a file and returns its ID.
/user/notes/:noteId	POST	links the note to the user

3.1.2 LoginCard

Purpose: A component that provides a login form for users to authenticate.

Props

Prop Name	Type	Description
username	string	The username entered by the user.
password	string	The password entered by the user.

State

State Variable	Type	Description
username	string	Stores the entered username.
password	string	Stores the entered password.

Backend Calls

API Endpoint	HTTP Method	Purpose
/login	POST	Authenticates the user with the provided credentials.

3.1.3 NavigationPanel

Purpose: A navigation bar component that provides links to different sections of the application and handles user authentication.

Props

Prop Name	Type	Description
None	N/A	This component does not accept props.

State

State Variable	Type	Description
None	N/A	This component does not use internal state.

Backend Calls

API Endpoint	HTTP Method	Purpose
/logout	POST	Logs out the current user by clearing session data.

3.1.4 Note

Purpose: A component for displaying a note with its content, connected todos, and various action buttons.

Props

Prop Name	Type	Description
id	string	The unique identifier for the note.
title	string	The name/title of the note.
preview	string	The main content of the note.
todoIds	string[]	A list of todo IDs associated with the note.

State

State Variable	Type	Description
todos	TodoData[]	Stores the list of fetched todos from the server.

Backend Calls

API Endpoint	HTTP Method	Purpose
/todos	GET	Fetches all todos from the server to display connected ones.
/note/:id	DELETE	Deletes the note from the server.
/file/download/:fileId	GET	downloads the file linked to the note

3.1.5 RegisterCard

Purpose: A React component that provides a registration form for new users. It includes input fields for username and password, and submits the data to register the user.

Props

Prop Name	Type	Description
None	N/A	This component does not accept props.

State

State Variable	Type	Description
username	string	Stores the username entered by the user.
password	string	Stores the password entered by the user.

Backend Calls

API Endpoint	HTTP Method	Purpose
/register	POST	Sends the username and password to register a new user.

3.1.6 LoginPage

Purpose: A React page component that renders both the login and register forms. It automatically checks if the user is already logged in upon loading.

Props

Prop Name	Type	Description
None	N/A	This component does not accept props.

State

State Variable	Type	Description

Backend Calls

API Endpoint	HTTP Method	Purpose
/login-status	GET	Checks if the user is already logged in.

3.1.7 Notes

Purpose: A React component that displays a list of notes fetched from the backend. It includes functionality for adding new notes via an overlay and linking notes to todos.

Props

Prop Name	Type	Description
None	N/A	This component does not accept props.

State

State Variable	Type	Description
notes: Array		

Backend Calls

API Endpoint	HTTP Method	Purpose
/notes	GET	Retrieves all notes from the backend.

3.1.8 TodoPage

Purpose: A React component that displays the list of todos. Can be filtered on all, done or undone todos. Also has functinallity to add new todos, mark existing todos as done/undone and delete specific todos, or all done todos.

Props

Prop Name	Type	Description
None	N/A	This component does not accept props.

State

State Variable	Type	Description
todos	Todo[]	A list of all the current users todo items
newTodo	string	A string that holds the value of the input field where users type the title of a new todo item. It is used to store the text entered by the user before the todo is added to the list. When a new todo is successfully added, the state is reset to an empty string.
display	string	A string that determines which todos are displayed. It can be 'all' to show all todos, 'active' to show only incomplete todos, or 'completed' to show only completed todos.

Backend Calls

API Endpoint	HTTP Method	Purpose
/todos	GET	Fetches all todo items from the backend.
/todos	POST	Adds a new todo to the database.
/todos/:id	POST	Toggles the completion status of a todo.
/todos/:id	DELETE	Deletes a todo from the database.

3.1.9 Pomodoro

Purpose: A react component with a timer in which switches from 25 minutes to 5 minutes indicating a break. It also shows previous sessions for a user

Props

Prop Name	Type	Description
None	N/A	This component does not accept props.

State

State Variable	Type	Description
minutes	number	the number of minutes to count down
seconds	number	the number of seconds to count down
isActive	bool	wether the timer is active or not
isBreak	bool	wether the time is counting break time or active time
sessionFlag	bool	indicates if a session is started or not
pomodoroSessions	PomodoroObject[]	the local list of pomodoro sessions held
pomodoroObject	PomodoroObject	an object created on timer start that is filled in with an end time and sent to the backend when the timer is finnished

Backend Calls

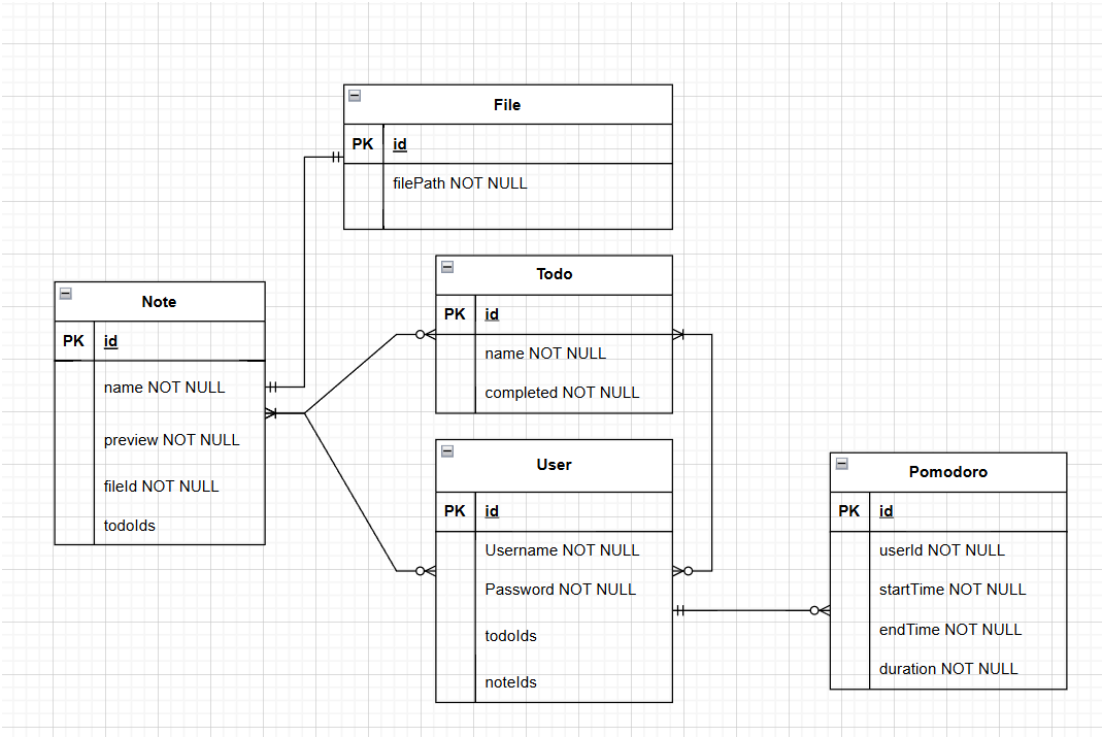
API Endpoint	HTTP Method	Purpose
/pomodoro	POST	uploads a pomodoro object
/pomodoro	GET	gets all the pomodoro objects from a user

3.1.10 ErrorPage

This displays any error code given to it with a fun cat image

4 ER Diagram

The ER diagram illustrates the design of our database, detailing the structure of entities and their relationships. You may also view it on this link: View the ER Diagram



5 API Specification

The following section provides a list of requests the backend API accepts and the corresponding responses it sends for both valid and invalid requests.

5.1 User Endpoints

Endpoint	Description	Response
`GET /user/id`	Retrieve user by ID.	**200 OK**: { "id": 1, "name": "John Doe", "todoIds": [1, 2], "noteIds": [3, 4] } **404 Not Found**: User not found. **500 Internal Server Error**: Server error occurred.
`POST /user/login`	Log in a user.	**200 OK**: { "message": "Login successful", "user": { "id": 1, "name": "John Doe" } } **401 Unauthorized**: Incorrect credentials. **500 Internal Server Error**: Server error occurred.
`POST /user/logout`	Log out the current user.	**200 OK**: { "message": "Logout successful" } **500 Internal Server Error**: Server error occurred.
`POST /user/register`	Register a new user.	**200 OK**: { "message": "Registration successful" } **400 Bad Request**: Missing required fields or user already exists. **500 Internal Server Error**: Server error occurred.
`DELETE /user/notes/noteId`	Delete the link between the user and a note.	**200 OK**: { "message": "Note deleted" } **401 Unauthorized**: User not logged in **400** error in request **500 Internal Server Error**: Server error occurred.
`POST /user/notes`	Add a note to the user.	**200 OK**: { "message": "Note added" } **401 Unauthorized**: User not logged in **400** Note already added **500 Internal Server Error**: Server error occurred.
`DELETE /user/todos/todoId`	Delete the link between the user and a todo.	**200 OK**: { "message": "Todo deleted" } **401 Unauthorized**: User not logged in **400** error in request **500 Internal Server Error**: Server error occurred.
`POST /user/todo`	Add a todo to the user.	**200 OK**: { "message": "Todo added", "id": 1 } **401 Unauthorized**: User not logged in **400** Todo already added **500 Internal Server Error**: Server error occurred.
`PUT /user/name`	Update the username of the user.	**200 OK**: { "message": "Name updated" } **401 Unauthorized**: User not logged in **500 Internal Server Error**: Server error occurred.
`PUT /user/password`	Update the password of the user.	**200 OK**: { "message": "Password updated" } **401 Unauthorized**: User not logged in **500 Internal Server Error**: Server error occurred.

5.2 Todo Endpoints

Endpoint	Description	Response
`GET /todo`	Retrieve all todos for the current user.	**200 OK**: { "id": 1, "title": "Buy groceries", "completed": false } **401 Unauthorized**: User is not authenticated. **500 Internal Server Error**: Server error occurred.
`GET /todo/list`	Retrieve todos by a list of IDs.	**200 OK**: { "id": 1, "title": "Buy groceries", "completed": false } **400 Bad Request**: Invalid request. **500 Internal Server Error**: Server error occurred.
`DELETE /todo/id`	Delete a todo by ID.	**200 OK**: { "message": "Todo deleted" } **401 Unauthorized**: User is not authenticated. **404 Not Found**: todo not found **500 Internal Server Error**: Server error occurred.
`POST /todo`	Add a new todo.	**200 OK**: { "message": "Todo added", "id": 1 } **400 Bad Request**: Missing "title" in body. **500 Internal Server Error**: Server error occurred.
`POST /todo/id/done`	Mark a todo as done.	**200 OK**: { "message": "Todo done" } **400 Bad Request**: Invalid request. **404 Not Found**: todo not found **500 Internal Server Error**: Server error occurred.
`POST /todo/id/undone`	Mark a todo as not done.	**200 OK**: { "message": "Todo undone" } **400 Bad Request**: Invalid request. **404 Not Found**: todo not found **500 Internal Server Error**: Server error occurred.

5.3 Pomodoro Endpoints

Endpoint	Description	Response
`GET /Pomodoro`	Retrieve a list of pomodoro sessions.	**200 OK**: List of pomodoro sessions **401 Unauthorized**: User is not logged in. **500 Internal Server Error**: Server error occurred.
`POST /Pomodoro`	Create a new pomodoro session.	**Request Body**: { "pomodoroObject": { "startTime": "2025-01-01T10:00:00Z", "endTime": "2025-01-01T10:25:00Z", "duration": 25, "userId: 23489203489} } **200 OK**: Pomodoro session created **400 Bad Request**: Missing "pomodoroObject". **401 Unauthorized**: User not logged in. **500 Internal Server Error**: Server error occurred.
`DELETE /Pomodoro/{id}`	Delete a pomodoro session.	**Parameter**: `id` - Pomodoro ID (e.g., 1) **200 OK**: Pomodoro session deleted **401 Unauthorized**: User is not logged in. **500 Internal Server Error**: Server error occurred.

5.4 Note Endpoints

Endpoint	Description	Response
`GET /note`	Get all notes for the authenticated user.	**200 OK**: List of notes **401 Unauthorized**: User is not authenticated. **404 Not Found**: Notes not found.
`POST /note`	Create a new note.	**Request Body**: { "title": "database lecture 4", "fileId": 343434, "content": ["string"] } **200 OK**: Note created with `id` and `message` **404 Not Found**: File linked to note does not exist.
`DELETE /note/{id}`	Delete a note.	**Parameter**: `id` - Note ID **200 OK**: Note deleted with `message` **404 Not Found**: Note not found.

5.5 File Endpoints

Endpoint	Description	Response
`GET /file/id`	Retrieve a file by ID.	**200 OK**: File details retrieved successfully. **404 Not Found**: File not found.
`GET /file/download/id`	Download a file by ID.	**200 OK**: File downloaded successfully. **404 Not Found**: File not found. **400 Bad Request**: The file extension is not supported.
`POST /file`	Upload a file.	**200 OK**: File uploaded successfully. **500 Internal Server Error**: Server error occurred.
`DELETE /file/id`	Delete a file by ID.	**Parameter**: `id` - File ID **200 OK**: File deleted successfully. **404 Not Found**: File not found.

6 Responsibilities

Our team worked closely together to develop and document the application, with each member taking on different areas while ensuring seamless integration. To facilitate collaboration, we divided responsibilities based on the three main features, with each person focusing on one: Casper worked on the Notes section, Erik handled the To-Do page, and Carl developed the Pomodoro timer.

In addition to these core features, Carl set up the React project, laying the foundation for the frontend. Erik wrote most of the frontend and backend documentation and API endpoint documentation using Swagger and also developed the file upload service. Casper implemented the login and registration pages and handled the database setup, designing the schema and ensuring smooth data management as well as having a hand in file downloading and parsing. Tests were written as needed by all team members. Most changes were also peer reviewed using pull requests making sure everyone was familiar with the code written in different places.

By distributing tasks this way, we were able to efficiently develop and integrate the different parts of the application while maintaining clear documentation for future development.