DAT076 Web applications - report

Group 7

Pouya Shirin

Albin Sundström

Emil Svensson

DAT076 Web applications

Software engineering

Chalmers University of Technology

Introduction

This report will present a ticket system application made for the course DAT076. The application offers the ability for users to create and manage their own tickets and a messaging system that allows admins to assist users with their tickets. Both admins and the user have the option to close or open tickets. Furthermore, the application has three different user ranks - user, admin, and super admin - each with varying levels of access and functionality. Users can change their passwords, while admins and super admins can do all of that as well as viewing all tickets and users.

In this report, you will find a comprehensive list of use cases, a user manual complete with installation instructions, and a detailed design section that covers our libraries, frameworks, and APIs. Additionally, we have included a responsibilities section that outlines the specific roles and contributions of each team member to various aspects of the project.

Link to GitHub page

Use cases

Finished use cases

User case: Sign up
As a visitor, I want to sign up so I can make use of the application's features.

User case: Login
As a user, I want to login in order to make use of the application's features.

User case: Sign out
As a user, I want to sign out so I can protect my account from anybody else accessing it.

User case: Change password
As a user, I want change my password so I can keep it secure.

User case: Create ticket
As a user, I want to create a ticket so I can either get help with a problem I have, or to highlight a problem to the developers.

User case: View a ticket
As a user, I want to view a ticket so I can read its description.

User case: Open/close ticket

As a user or admin, I want to open or close a ticket in order to mark the ticket as solved.

User case: View my tickets

As a user, I want to view my tickets I have made, so I can easier navigate and find them.

User case: View all tickets

As an admin, I want to view all tickets that have been made, so I can easier navigate and find them.

User case: View all open/closed tickets

As an admin, I want to filter all tickets by open or closed so I can easier find the tickets I am looking for.

User case: View all users

As an admin, I want to view all users so I know which users that exists.

User case: View sorted users

As an admin, I want to sort all users by their rank so I can easier identify all users by a certain rank.

User case: Post a message

As a user, I want to post a message to a ticket, so I communicate with the other person in order to solve the ticket.

User case: Read ticket messages

As a user, I want to read messages posted to a ticket, so I can get updated on the problem.

Unfinished use cases

These are use cases that we planned to implement and is relevant to the existing use cases.

User case: Assign ticket

As an admin, I want to assign myself to a ticket so other developers know that it's taken.

User case: View assigned tickets

As an admin, I want to view the tickets I have assigned so I can easier navigate to them.

User case: User tools

As an admin, I want to edit a user, so I can either give them admin permissions, ban or other relevant functions.

User manual

These instructions assume that git and Node.js version 16+ is installed on the computer.

1. Use a terminal of your choice and clone the repository from GitHub using the following command:

   ```
   git clone https://github.com/Kexon/DAT076.git
   ```

2. Head into the cloned folder:

   ```
   cd DAT076
   ```

3. Create a .env file with the following content:

   ```
   SECRET_KEY="funny_secret_bere"
   DB_URI="<DB_URI to a MongoDB database>"
   DB_URI_TEST="<DB_URI to a MongoDB test database>"
   ```

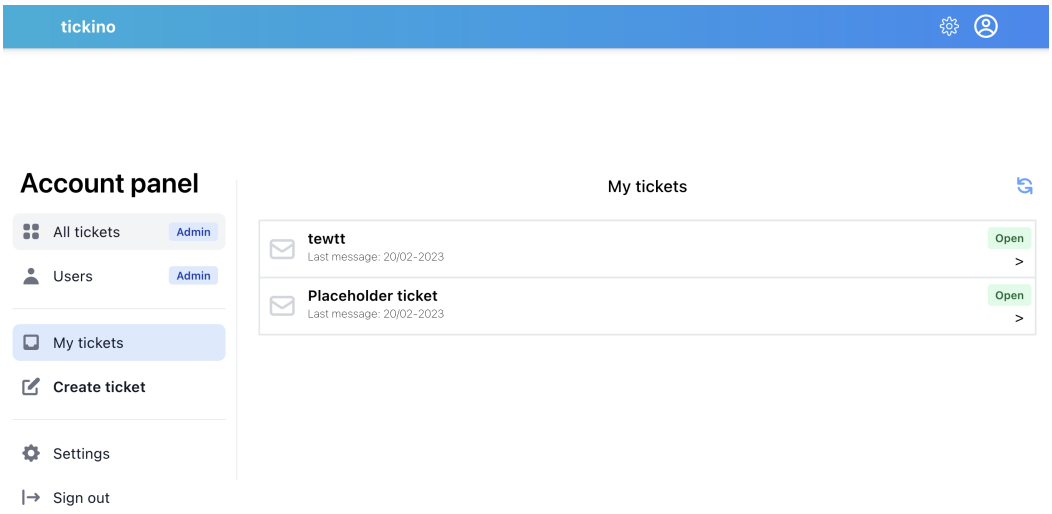
4. Run the following two commands:

   ```
   cd client && npm install && npm run build
   cd .. && cd server && npm install
   ```

5. From the server folder, run the following command to start the application:

   ```
   npm run dev
   ```

6. Go to http://localhost:8080 to use the application. Note that you need to be logged in to be able to access any features. You can create an account from http://localhost:8080/signup.



Figur 1: Screenshot of the front page of the application.

Design

Libraries and frameworks

Node

Express

TypeScript

React

MongoDB

Mongoose

Axios

TailwindCSS

Flowbite

ESLint

ESLint was installed and setup to enforce the Airbnb style guide. This ensures a consistent code style across all team members, which makes the code easier to understand. We also used ESLint to enforce other best practices in TypeScript, and enabled CI to automatically run the checks on pull requests.

React Router

Jest and Supertest

Cors

REST API

/ticket

Description: Endpoint for managing tickets.

Methods:

- GET: Returns a list of all tickets (status code 200).
- POST: Creates a new ticket (status code 201).

Request payload for POST method:

```
{
  "title": "Ticket title",
  "description": "Ticket description"
}
```

Response payload for GET method on success:

```
[
    {
        "_id": "64022132132d81be5f8e9407",
        "title": "Lost items",
        "open": false,
        "owner": {
            "_id": "63fe09e0becb05ba0b48b1fd",
            "username": "test",
            "level": 1,
            "__v": 0,
            "id": "63fe09e0becb05ba0b48b1fd"
        },
```

```
        "__v": 0,
        "id": "64022132132d81be5f8e9407"
    }
]
```


- Supports the query parameter **authorId** that returns the tickets for a specific user.
- Returns ”Invalid request” with status code 401 when supplying incorrect query parameters.
- Returns ”You are not logged in” with status code 401 when no user is authenticated.
- Returns the exception message with status code 500 on unexpected failures.

/ticket/:id

Description: Endpoint for managing a specific ticket.

Methods:

- GET: Returns a specific ticket (status code 200).
- PATCH: Updates a specific ticket (status code 200).

Request payload for PATCH method:


```
{
  "title": "New ticket title",
}
```


Response payload for GET method on success:


```
{
  "_id": "36022132132d81be5f8e9407",
  "title": "New ticket title",
  "open": false,
  "owner": {
    "_id": "63fe09e0becb05ba0b48b1fd",
    "username": "test",
    "level": 1,
    "__v": 0,
```

```
    "id": "63fe09e0becb05ba0b48b1fd"
  },
  "__v": 0,
  "id": "64022132132d81be5f8e9407"
}
```

- The PATCH method accepts updating of **title** or **open**, either independently or both at the same time.
- Returns ”Invalid id” with status code 400 when supplying an invalid ID format.
- Returns ”You are not logged in” with status code 401 when no user is authenticated.
- Returns ”You are not authorized to do this” with status code 403 if a user is not allowed to edit the ticket.
- Returns the exception message with status code 500 on unexpected failures.

/user

Description: Endpoint for managing users.

Methods:

- GET: Returns the current logged in user (status code 200).
- POST: Creates a new user (status code 201).
- PATCH: Updates the password of the current logged in user (status code 200).

Request payload for POST method:

```
{
  "username": "penguin",
  "password": "toyota"
}
```

Request payload for PATCH method:

```
{
  "currentPassword": "hello",
```

```
  "newPassword": "supersecurepw"
}
```

Response payload for GET method on success:

```
{
    "_id": "63fdc1567084eb60806e718d",
    "username": "emil",
    "level": 3,
    "__v": 0,
    "id": "63fdc1567084eb60806e718d"
}
```

- Returns ”You are not logged in” with status code 401 when no user is authenticated.
- Returns ”Username already exists” with status code 409 when attempting to register a user with a username that already exists.
- Returns the exception message with status code 500 on unexpected failures.

/user/:id

Description: Endpoint for managing a specific user.

Methods:

- GET: Returns a specific ticket. (status code 200).
- PATCH: Updates the password of the current logged in user (status code 200).

Request payload for PATCH method:

```
{
  "currentPassword": "hello",
  "newPassword": "supersecurepw"
}
```

Response payload for GET method on success:

```
{
    "_id": "63fdc1567084eb60806e718d",
    "username": "emil",
    "level": 3,
    "__v": 0,
    "id": "63fdc1567084eb60806e718d"
}
```


- Returns ”You are not logged in” with status code 401 when no user is authenticated.
- The PATCH call is only allowed for super admins and will return ”You are not allowed to perform this operation” with status code 401 otherwise.
- Returns the exception message with status code 500 on unexpected failures.

/user/login

Description: Endpoint for logging in a user.

Methods:

- POST: Authenticates a user and returns the logged in user on success (status code 200).

Request payload for POST method:


```
{
  "username": "emil",
  "password": "supersecretpw"
}
```


Response payload for POST method on success:


```
{
    "_id": "63fdc1567084eb60806e718d",
    "username": "emil",
    "level": 3,
    "__v": 0,
    "id": "63fdc1567084eb60806e718d"
}
```

- Returns ”Wrong username or password” with status code 401 on failure.

/user/logout

Description: Endpoint for logging out a user.

Methods:

- POST: Terminates the current session (status code 200).

- Returns ”Successfully logged out"” with status code 200 on success.
- Returns ”Failed to log out"” with status code 500 on failure.

/message/:id

Description: Endpoint for managing a chat message.

Methods:

- GET: Returns the chat message (status code 200).

Response payload for GET method on success:


```
{
  "_id": "6409d48b1633fa6121ce7606",
  "chatId": "6409d48b1633fa6121ce7602",
  "timestamp": "2023-03-09T12:43:55.607Z",
  "sender": {
    "_id": "63fe09e0becb05ba0b48b1fd",
    "username": "test",
    "level": 1,
    "__v": 0,
    "id": "63fe09e0becb05ba0b48b1fd"
  },
  "content": "created a ticket",
  "systemMessage": true,
  "__v": 0,
```

```
  "id": "6409d48b1633fa6121ce7606"
}
```

- Returns ”You are not logged in” with status code 401 when no user is authenticated.
- Returns ”Invalid id” with status code 400 when supplying an invalid ID format.
- Returns the exception message with status code 500 on unexpected failures.

/message/chat

Description: Endpoint for managing chats.

Methods:

- POST: Create a message to a chat (status code 201).

```
{
  "chatId": "6409d48b1633fa6121ce7602",
  "content": "this is a message"
}
```

Response payload for POST method on success:

```
{
  "chatId": "6409d48b1633fa6121ce7602",
  "timestamp": "2023-03-13T09:55:48.641Z",
  "sender": {
    "_id": "63fdc1567084eb60806e718d",
    "username": "emil",
    "level": 3,
    "__v": 0,
    "id": "63fdc1567084eb60806e718d"
  },
  "content": "this is a message",
  "systemMessage": false,
  "_id": "640ef324d59030cb97b01bc9",
  "__v": 0,
  "id": "640ef324d59030cb97b01bc9"
```

```
}
```

- The **chatId** is always the **ticketId** in our implementation.
- Returns ”You are not logged in” with status code 401 when no user is authenticated.
- Returns ”Invalid id” with status code 400 when supplying an invalid ID format.
- Returns ”Invalid chat content” with status code 400 when supplying an empty message.

/message/chat/:chatId

Description: Endpoint for managing a specific chat.

Methods:

- GET: Returns a list of messages for a specific chat (status code 200).

Response payload for GET method on success:

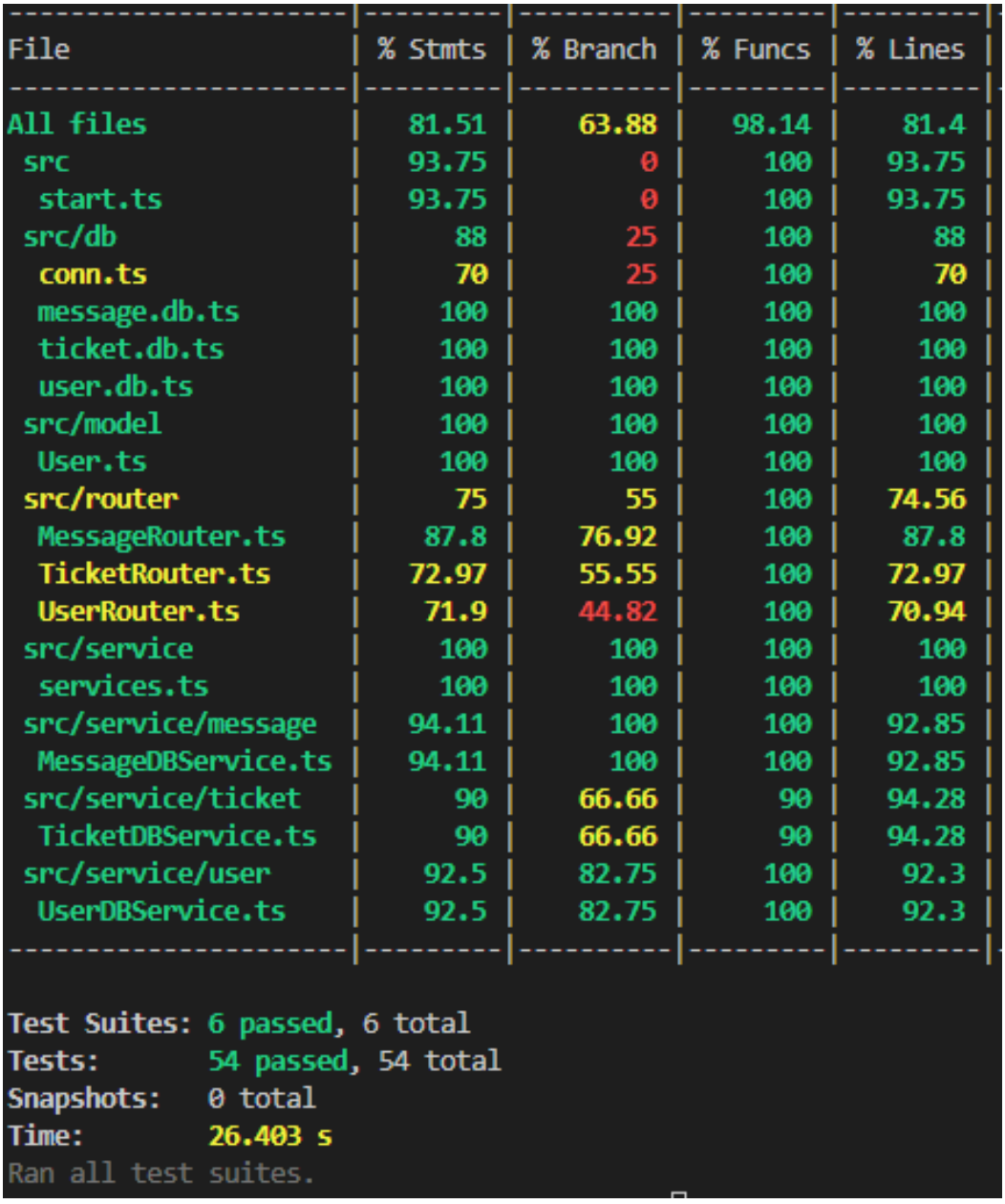

```
[
  {
    "_id": "6409d48b1633fa6121ce760a",
    "chatId": "6409d48b1633fa6121ce7602",
    "timestamp": "2023-03-09T12:43:55.658Z",
    "sender": {
      "_id": "63fe09e0becb05ba0b48b1fd",
      "username": "test",
      "level": 1,
      "__v": 0,
      "id": "63fe09e0becb05ba0b48b1fd"
    },
    "content": "Test description",
    "systemMessage": false,
    "__v": 0,
    "id": "6409d48b1633fa6121ce760a"
  },
  {
    "_id": "640ef324d59030cb97b01bc9",
```

```
    "chatId": "6409d48b1633fa6121ce7602",
    "timestamp": "2023-03-13T09:55:48.641Z",
    "sender": {
      "_id": "63fdc1567084eb60806e718d",
      "username": "emil",
      "level": 3,
      "__v": 0,
      "id": "63fdc1567084eb60806e718d"
    },
    "content": "sup guys",
    "systemMessage": false,
    "__v": 0,
    "id": "640ef324d59030cb97b01bc9"
  }
]
```


- Returns ”You are not logged in” with status code 401 when no user is authenticated.
- Returns ”Invalid id” with status code 400 when supplying an invalid ID format.
- Returns the exception message with status code 500 on unexpected failures.

Test coverage

Backend

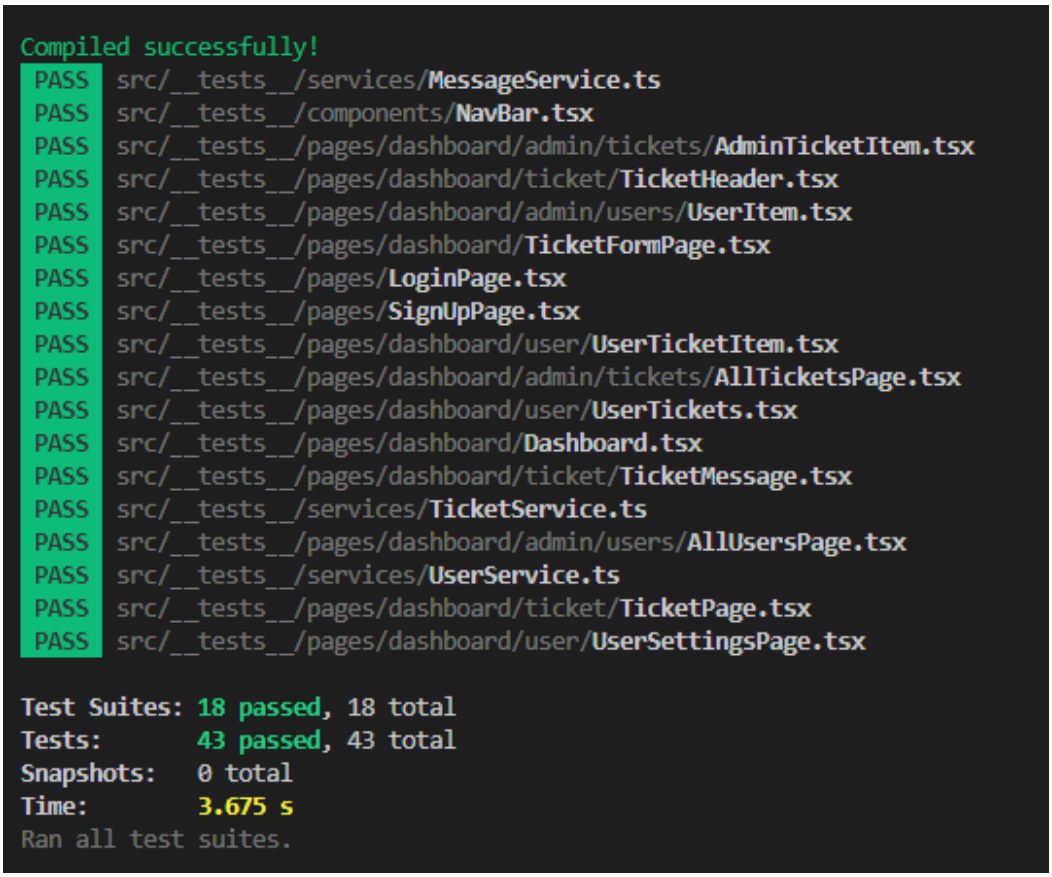


File	% Stmts	% Branch	% Funcs	% Lines
All files	81.51	63.88	98.14	81.4
src	93.75	0	100	93.75
start.ts	93.75	0	100	93.75
src/db	88	25	100	88
conn.ts	70	25	100	70
message.db.ts	100	100	100	100
ticket.db.ts	100	100	100	100
user.db.ts	100	100	100	100
src/model	100	100	100	100
User.ts	100	100	100	100
src/router	75	55	100	74.56
MessageRouter.ts	87.8	76.92	100	87.8
TicketRouter.ts	72.97	55.55	100	72.97
UserRouter.ts	71.9	44.82	100	70.94
src/service	100	100	100	100
services.ts	100	100	100	100
src/service/message	94.11	100	100	92.85
MessageDBService.ts	94.11	100	100	92.85
src/service/ticket	90	66.66	90	94.28
TicketDBService.ts	90	66.66	90	94.28
src/service/user	92.5	82.75	100	92.3
UserDBService.ts	92.5	82.75	100	92.3

```
Test Suites: 6 passed, 6 total
Tests:       54 passed, 54 total
Snapshots:   0 total
Time:        26.403 s
Ran all test suites.
```

Figur 2: Screenshot of the test coverage for the backend tests.

Frontend



Figur 3: Screenshot of the frontend tests.

Responsibilities

All group members participated in creating the foundation of the application - the database, backend and frontend. It made sense program together as the point of this project was not only to create an application, but also for learning purposes. Once the foundation was established, each group member was assigned to cover a major feature. Some features were large enough to be covered by two people.

Albin

- Sign up page
  - Implemented frontend

- Create new ticket
  - Implemented frontend
  - Implemented part of backend
- Navigation
  - Implemented frontend for nav bar and hamburger menu
- Backend testing
  - Wrote all tests for the service layer
  - Wrote all tests for the Ticket router and User router

Emil

- Setup the base project with ESLint
- Setup React Router
- Implemented authentication
  - Created the useAuth hook in frontend
  - Created the RequireAuth component
- Implemented the MongoDB database
  - Set up the database in Atlas
  - Created the mongoose schemas
  - Refactored the code to use the database instead
- Services in backend
- Routers in backend
- Utils & services in frontend
- Messaging system (backend & frontend)
- Login view

Pouya

- Messaging system
  - Implemented in frontend and backend
- Dashboard
  - Implemented in frontend
  - Including the resizing all pages to fit the view
- Settings page
  - Implemented in frontend
  - and backend
- All Tickets page
- All User page
- My tickets page
- Majority of the documentation of the app
- Majority of the frontend testing