VRIJE
UNIVERSITEIT
BRUSSEL

Gevorderde Webapplicaties
Master of Science in de Industriële Wetenschappen:
Elektronica-ICT - netwerken

TASK MANAGER

Nellie Van Eeckhaute

12/01/2025

INGENIEURSWETENSCHAPPEN

Inhoudsopgave

Inleiding

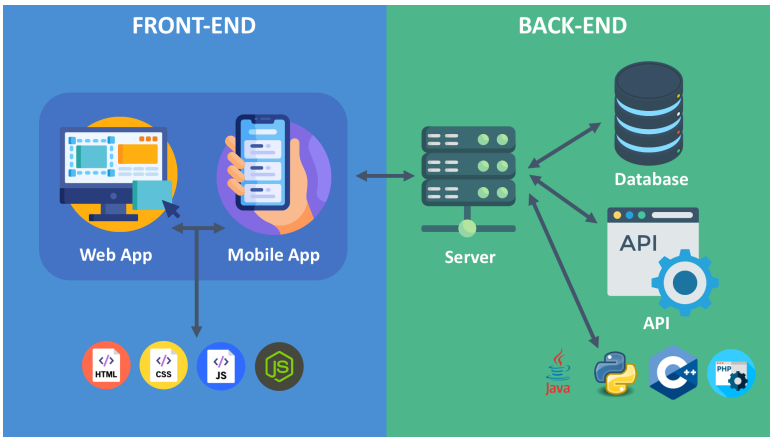
Als opdracht voor het vak *Gevorderde Webapplicaties* heb ik een task manager applicatie gemaakt, die bestaat uit zowel een front- als backend.

De task manager is ontwikkeld om samenwerking binnen teams te verbeteren door een overzichtelijk systeem voor takenbeheer te bieden, waarbij gebruikers zich kunnen registreren, inloggen, projecten en taken kunnen beheren en rollen kunnen toewijzen. De applicatie is ontworpen met een focus op functionaliteit, beveiliging en gebruiksvriendelijkheid.

De backend bevat API-endpoints voor het ondersteunen van acties op de frontend, en is gemaakt met *Node.js*. Deze API-endpoints zijn als het ware het receptenboek van de applicatie en zullen verder in het verslag uitgelegd worden.

De frontend is gemaakt met *React*, en houdt rekening met een responsive design.

Naast de basisvereisten zijn er ook een aantal extra vereisten in dit project mogelijk. Deze worden verder in het verslag aangehaald.



Figuur 1.1: RESTful API model

2.1 Overzicht

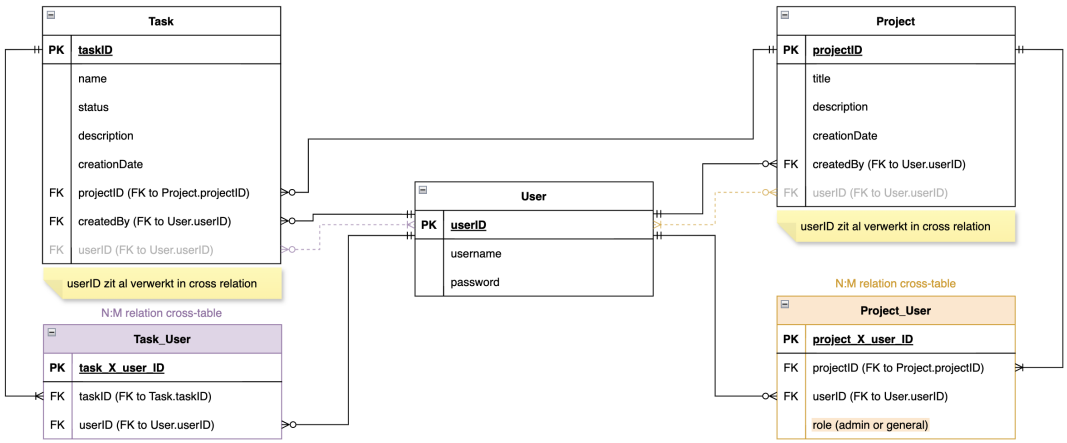
De applicatie maakt gebruik van een client-serverarchitectuur, bestaande uit:

- **Front-end:** Gemaakt in React, zorgt voor een responsieve en interactieve interface.
- **Back-end:** Gebouwd met Node.js, inclusief een RESTful API en een SQLite-database voor gegevensopslag.
- **Authenticatie:** JWT (JSON Web Tokens) voor veilige gebruikersverificatie.

Voor het bijhouden van wijzigingen en het beheren van het project heb ik gebruik gemaakt van GitHub. De link naar mijn repository is deze: `https://github.com/nve03g/task_manager_webdev`.

2.2 ER-diagram

Vooraleer ik aan het project ben begonnen heb ik een ER-diagram opgesteld om een overzicht te kunnen houden over de structuur van de database.



Figuur 2.1: ER-diagram dat de structuur van de database aangeeft

De redenering achter dit diagram is de volgende:

- Alle gebruikers op het systeem kunnen een project aanmaken. Wanneer een project wordt aangemaakt is deze gebruiker automatisch aangewezen als beheerder van dat project ("administrator").
- Elke projectbeheerder kan andere gebruikers aan dat project toevoegen, en per user kiezen of het om een normale gebruiker of ook om een beheerder gaat.
- Projectbeheerders kunnen taken toevoegen aan het project, taken in het project wijzigen (zoals bijvoorbeeld ook de status van de taak) en gebruikers toewijzen aan een taak binnen het project.
- Normale gebruikers binnen een project kunnen de status van toegewezen taken binnen dat project aanpassen.
- Een taakloos project kan bestaan, maar een taak moet altijd aan juist één project toegevoegd worden.
- Een taak kan toegewezen zijn aan meerdere gebruikers, maar altijd minstens één.
- Een project bestaat minstens uit één projectbeheerder, en dat is de gebruiker die het project heeft aangemaakt.

Backend

3.1 Overzicht

De backend is ontwikkeld in *Node.js* en biedt een RESTful API met verschillende endpoints voor gebruikers-, project- en taakbeheer. SQLite is gebruikt als database vanwege de eenvoud, lichtgewicht architectuur, en de mogelijkheid om gemakkelijk lokaal te ontwikkelen.

3.2 Technische implementatie

3.2.1 Databaseontwerp

De database bestaat uit verschillende tabellen:

- `User`: Bevat gebruikersinformatie zoals `userID`, `username`, en gehashte wachtwoorden.
- `Project`: Houdt projecten bij, inclusief informatie over de maker (`createdBy`) en de aanmaakdatum.
- `Task`: Bevat taken gekoppeld aan projecten, met statusinformatie en een beschrijving.
- `Project_User` en `Task_User`: Relatietabellen die respectievelijk gebruikers aan projecten en taken koppelen, samen met rollen.

3.2.2 Gebruikersbeheer

Gebruikers kunnen zich registreren met een unieke gebruikersnaam en wachtwoord. Het wachtwoord wordt gehasht met behulp van **bcrypt** voordat het wordt opgeslagen in de SQLite-database. Dit voorkomt dat wachtwoorden in platte tekst worden opgeslagen, wat een belangrijke beveiligingspraktijk is. Bij het inloggen wordt het ingevoerde wachtwoord geverifieerd door het te vergelijken met de opgeslagen hash.

JWT wordt gebruikt voor sessiebeheer. Bij een succesvolle login genereert de server een token dat wordt teruggestuurd naar de client. Dit token bevat versleutelde gebruikersinformatie (zoals `userID`) en een vervaldatum, en wordt gebruikt voor elke volgende aanvraag om authenticatie te garanderen.

Middleware zoals `checkTokenBlacklist` zorgt ervoor dat tokens kunnen worden gevalideerd of ingetrokken (bijvoorbeeld bij een logout). Dit verhoogt de veiligheid doordat verlopen of ingetrokken tokens direct worden geweigerd.

Autorisatie wordt geïmplementeerd door rollen te controleren (bv. admin of general) en te valideren of de gebruiker toegang heeft tot een specifieke bron, zoals een project of taak.

3.2.3 Projectbeheer

API-endpoints zoals `POST /projects` stellen beheerders in staat nieuwe projecten aan te maken. Bij het aanmaken van een project wordt de maker automatisch toegewezen als admin aan dat project. Andere gebruikers kunnen worden toegevoegd of verwijderd, afhankelijk van hun rol en autorisatie.

3.2.4 Takenbeheer

Bij het toevoegen van een taak (`POST /projects/:projectId/tasks`) worden de volgende stappen uitgevoerd:

1. Validatie van de invoer, zoals de naam en status van de taak.
2. Controle of de gebruiker bevoegd is (admin in het project).
3. Aanmaak van de taak in de database, met automatische registratie van de aanmaakdatum.
4. Toevoegen van gebruikers aan de taak via de `Task_User`-crosstable.

3.2.5 Database-operaties

Om transacties betrouwbaar te maken (bijvoorbeeld bij het aanmaken van een project of het toewijzen van gebruikers), wordt gebruik gemaakt van *database transactions*. Dit garandeert dat alle gerelateerde operaties slagen of worden teruggedraaid bij een fout.

Voorbeeld:

```
await new Promise((resolve, reject) => {
    db.run('BEGIN TRANSACTION;', (err) => {
        if (err) reject(err);
        else resolve();
    });
});
// andere database-operaties hiertussen
await new Promise((resolve, reject) => {
    db.run('COMMIT;', (err) => {
        if (err) reject(err);
        else resolve();
    });
});
```

4.1 Overzicht

De frontend van de applicatie is gebouwd met *React*, wat componentgebaseerde app-ontwikkeling ondersteunt. Dit maakt het mogelijk om de gebruikersinterface modulair, onderhoudbaar en herbruikbaar te maken. CSS-frameworks zoals *Bootstrap* zijn gebruikt om een responsief en visueel aantrekkelijk design te realiseren.

4.2 Technische implementatie

4.2.1 Structuur en componenten

De frontend is gestructureerd rondom een reeks *React*-componenten die elk een specifiek doel hebben:

- **Navbar:** Bevat navigatieopties die afhankelijk zijn van de authenticatiestatus van de gebruiker.
- **Dashboard:** Geeft een overzicht van de projecten en taken waaraan de gebruiker is toegewezen.
- **Login en registratie:** Formulieren voor gebruikersauthenticatie, gekoppeld aan API-endpoints.
- **Projectbeheer:** Pagina's voor het aanmaken, bewerken en bekijken van projecten, inclusief het toewijzen van gebruikers.
- **Takenbeheer:** Pagina's voor het aanmaken, bewerken en bekijken van taken, inclusief het toewijzen van gebruikers.
- **Profielpagina:** Toont gebruikersinformatie en statistieken zoals toegewezen projecten en het aantal toegewezen taken.

4.2.2 *React router*

Voor navigatie binnen de applicatie is gebruik gemaakt van *React Router*. Dit maakt client-side routing mogelijk, waardoor pagina's snel en zonder herladen van de browser kunnen worden weergegeven. Beschermde routes zijn geïmplementeerd met middleware-achtige componenten zoals `PrivateRoute`, die ervoor zorgen dat alleen geauthenticeerde gebruikers toegang hebben tot specifieke pagina's.

4.2.3 Statusbeheer

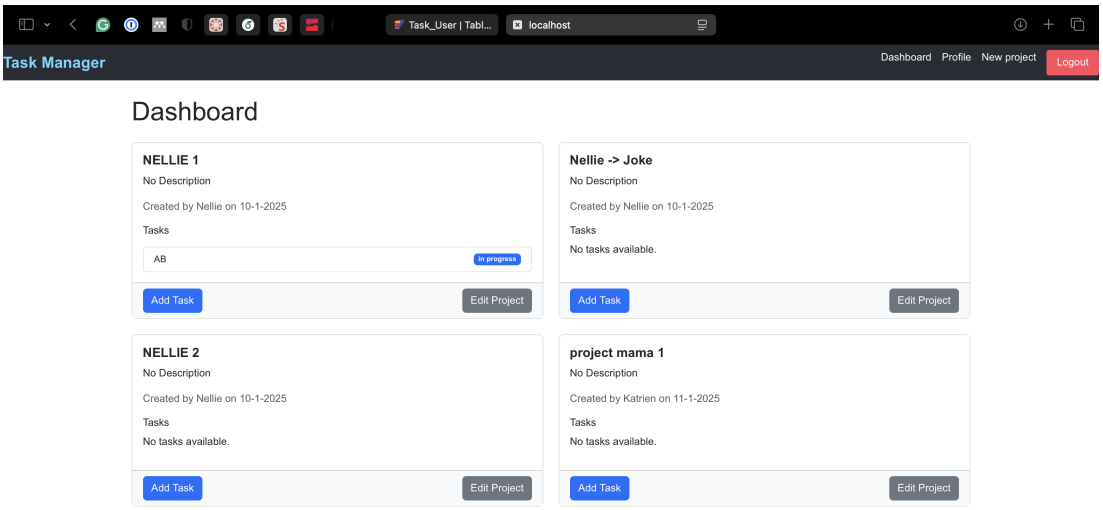
Het beheer van de applicatiestatus wordt verzorgd door een combinatie van *React*'s ingebouwde statusbeheer en de `AuthContext` API. Deze context biedt een centrale opslag voor authenticatiegegevens en stelt componenten in staat om toegang te krijgen tot de huidige gebruiker. Dit voorkomt het fenomeen van *prop-drilling*, waarbij data door meerdere tussenliggende componenten moet worden doorgegeven om de gewenste component te bereiken. In plaats daarvan maakt de context het mogelijk om data direct beschikbaar te stellen aan de componenten die deze nodig hebben, wat de code overzichtelijker en beter te onderhouden maakt.

4.2.4 Responsiveness en styling

De styling is gerealiseerd met behulp van *Bootstrap* en CSS-files. Hiermee wordt gezorgd dat de applicatie goed functioneert op zowel desktops als mobiele apparaten.

4.2.5 Mogelijke verbeteringen in de toekomst

- Implementeren van een zoekfunctie voor projecten en taken.
- Tags aanmaken per project die aan taken kunnen worden toegewezen.
- Filteren en sorteren van takenlijst op bepaalde criteria.
- Meer interactieve elementen, zoals drag-and-drop functionaliteit voor takenbeheer.
- Meer gepersonaliseerd gebruikersprofiel.



Figuur 4.1: Dashboard page of the task management application

5.1 Overzicht

De API-documentatie is een belangrijk onderdeel van de applicatie, omdat het externe developers en gebruikers helpt om de functionaliteit van de RESTful API te begrijpen en te testen. De documentatie is gegenereerd met behulp van *Swagger* en is beschikbaar via `/api-docs`.

5.2 Implementatie

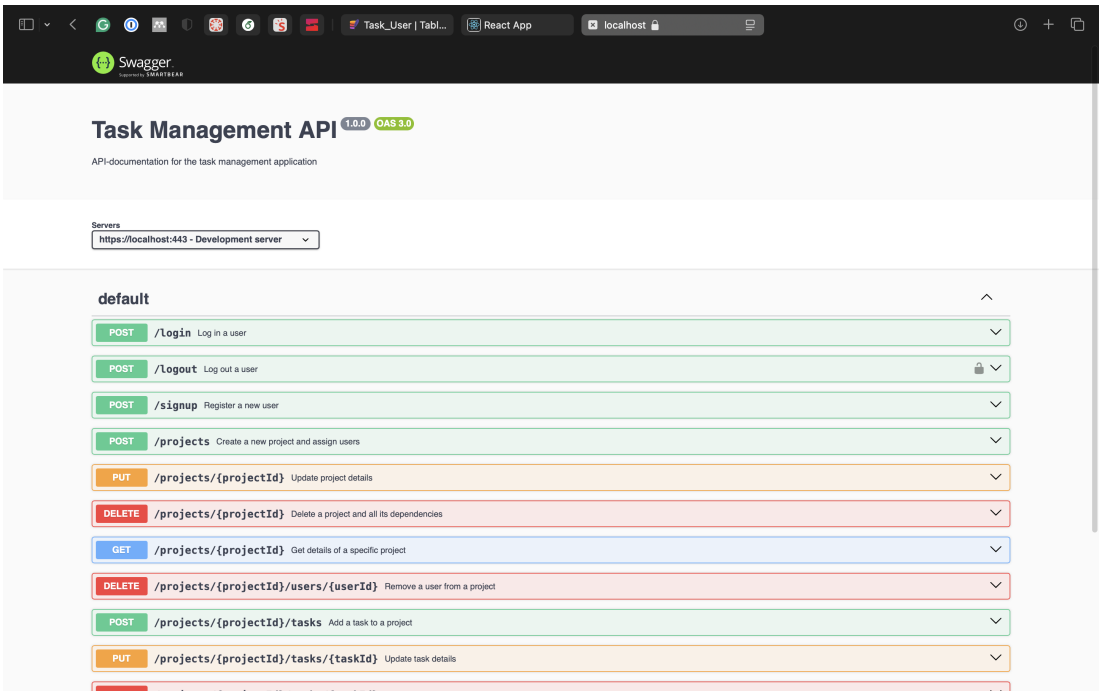
Swagger is geïntegreerd in de backend met behulp van de `swagger-jsdoc` en `swagger-ui-express` pakketten. Deze tools maken het mogelijk om documentatie direct uit de code te genereren door middel van speciale annotaties boven de endpoints.

Voorbeeldannotatie:

```
/**
 * @swagger
 * /projects:
 *   get:
 *     summary: Haalt alle projecten op
 *     responses:
 *       200:
 *         description: Succesvolle reactie - een lijst met projecten
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 type: object
 *                 properties:
 *                   projectID:
 *                     type: integer
 *                   title:
 *                     type: string
 *                   description:
 *                     type: string
 */
```

5.3 Voordelen van *Swagger*

1. **Gebruiksvriendelijkheid:** Ontwikkelaars kunnen direct interactie hebben met de API via de *Swagger UI*.
2. **Duidelijkheid:** Gedetailleerde beschrijvingen van parameters, request bodies en responses.
3. **Consistentie:** Documentatie wordt automatisch bijgewerkt wanneer de code wordt aangepast (dynamische documentatie).



Figuur 5.1: Swagger UI

Deployment

Om de applicatie eenvoudig te deployen/implementeren en te beheren, is gebruik gemaakt van *Docker*. *Docker* stelt ons in staat om zowel de frontend als de backend samen met de benodigde afhankelijkheden te bundelen in containers. Hierdoor kunnen ontwikkelaars de applicatie eenvoudig runnen, ongeacht hun lokale omgeving.

6.1 *Docker* configuratie

De deployment maakt gebruik van een *Dockerfile* voor zowel de frontend als de backend. Daarnaast wordt een `docker-compose.yml` bestand gebruikt om meerdere containers tegelijkertijd te beheren.

Voorbeeld: Dockerfile voor de backend

```
# Gebruik de officiële Node.js image
FROM node:16

# Stel de werkdirectory in
WORKDIR /app

# Kopieer package.json en installeer afhankelijkheden
COPY package.json .
RUN npm install

# Kopieer de rest van de applicatie
COPY . .

# Expose de benodigde poort en start de server
EXPOSE 443
CMD ["node", "server.js"]
```

Voorbeeld: docker-compose.yml

```
version: '3.8'
services:
  backend:
    build: ./backend
    ports:
      - "443:443"
  frontend:
    build: ./frontend
    ports:
```

```
    - "3001:3000"
```

6.2 Voordelen van *Docker*

1. **Consistentie:** Dezelfde omgeving wordt gebruikt voor ontwikkeling, testing en productie.
2. **Eenvoud:** Applicatie kan worden gestart met één commando (`docker-compose up`).
3. **Schaalbaarheid:** Containers kunnen eenvoudig worden geschaald in een productieomgeving.

Bibliografie

[1] *"Containerize your app*
. Okt 2024. URL: https://docs.docker.com/guides/nodejs/containerize/.

[2] *"Node.js*
. Okt 2024. URL: https://docs.docker.com/guides/nodejs/.

[3] *API Documentation Design Tools for Teams | Swagger*. URL: https://swagger.io/.

[4] *API Documentation Made Easy with OpenAPI Swagger*. URL: https://swagger.io/resources/articles/documenting-apis-with-swagger/.

[5] C. Blakely. *Full Stack React Node Project | Build a Notes App from Scratch for your Portfolio*. Sep 2023. URL: https://www.youtube.com/watch?v=2MoSzSlAuNk.

[6] A. Choubey. 'Securing Node.js Applications with JWT, Refresh Tokens, and Redis'. In: (aug 2023). URL: https://medium.com/@choubeyayush4/securing-node-js-applications-with-jwt-refresh-tokens-and-redis-80ffbb54285a.

[7] W. contributors. *List of HTTP status codes*. Dec 2024. URL: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

[8] Fireship. *Learn docker in 7 easy steps - Full beginner's tutorial*. Aug 2020. URL: https://www.youtube.com/watch?v=gAkwW2tuIqE.

[9] *How to Authenticate a Username Password using Express*. URL: https://www.passportjs.org/howtos/password/.

[10] *How to style a checkbox using CSS*. URL: https://stackoverflow.com/questions/4148499/how-to-style-a-checkbox-using-css.

[11] OpenAI. *ChatGPT (January 2025 version)*. Persoonlijke communicatie. (z.d.)

[12] Passport. *GitHub - passport/todos-express-password: Todo app using Express and Passport for sign in with username and password*. URL: https://github.com/passport/todos-express-password.

[13] *React Reference Overview – React*. URL: https://react.dev/reference/react.

[14] *Username Password tutorial*. URL: https://www.passportjs.org/tutorials/password/.

[15] *W3Schools.com*. URL: https://www.w3schools.com/tags/ref_httpmessages.asp.