Web 3

Student opvolging

David Breckx

2022-12-06

Table of Contents

1. Doelstellingen

Met het eindproject wordt er nagegaan of je de volgende doelstellingen bezit:

- REST API ontwikkelen met behulp van een Node.js (Express framework) backend en door gebruik te maken van MySQL databank (Prisma).
- Frontend applicatie ontwikkelen met behulp van React.
- Components, lokale state en props in React
- Gebruik van de React Hooks (useState, useEffect, useContext, …)
- Styling en layout van components
- State management a.h.v. Redux
- Navigatie met verschillende pagina’s in React
- (Authenticatie door gebruik te maken van een Node.js backend met JWT)

2. Inleiding

Het doel van dit project is een opvolgingsysteem voor studenten en lector te realiseren. Met het te implementeren systeem worden een opdracht met eventuele deelopdrachten geselecteerd en na het opstarten van de werkperiode inzake afwerking zoveel mogelijk in realtime opgevolgd.

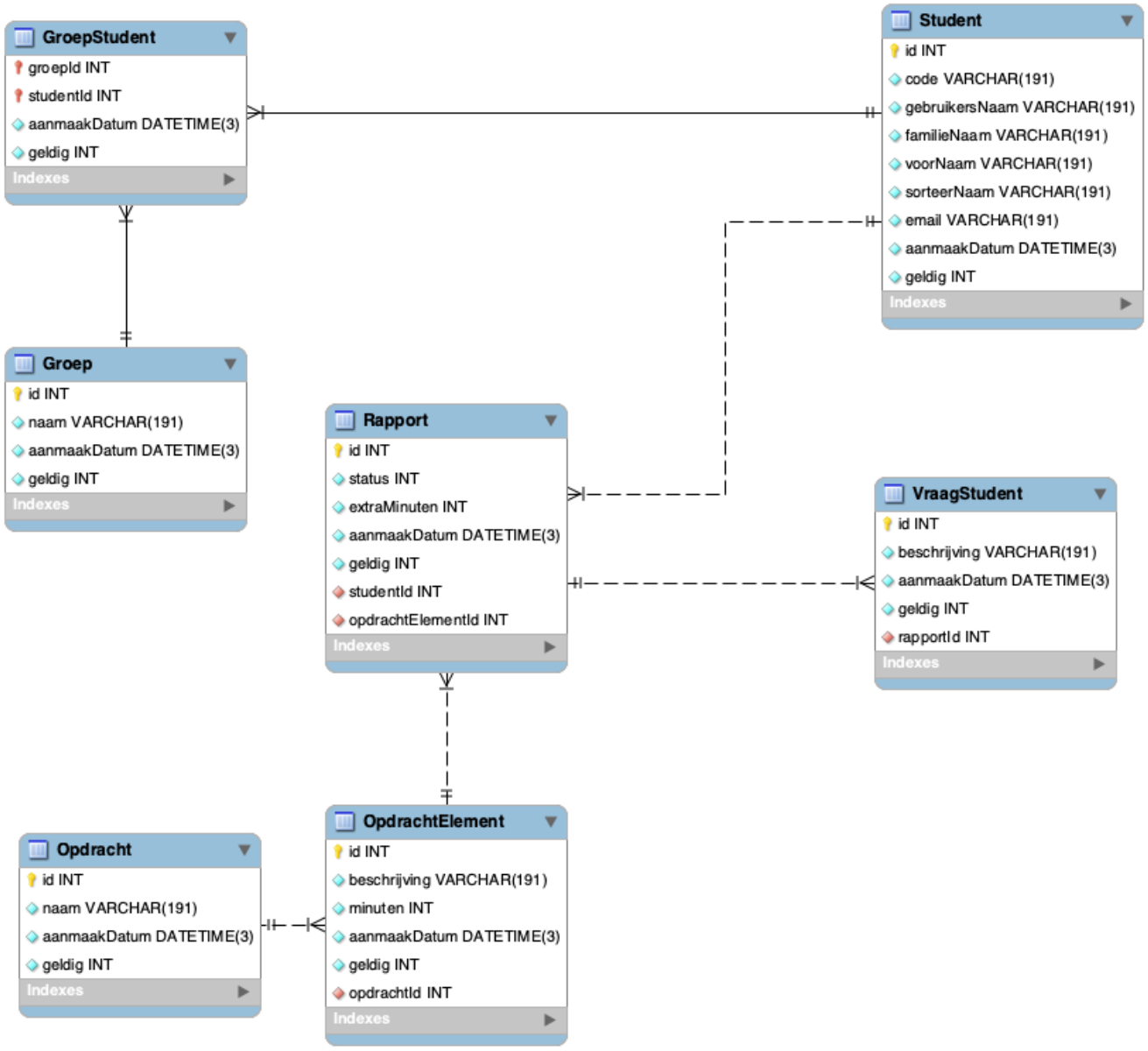
Achtergrond: het is niet altijd makkelijk als lector op te volgen dat studenten (deel)opdrachten al afgerond hebben of niet en in welke tijdspanne dat gebeurde. Het project geeft meer inzicht in het werken met opdrachten die tijdens de les worden gegeven. De onderliggende concepten zijn vergelijkbaar met Kahoot.

Momenteel bestaat een dergelijk systeem nog niet: Web 3 biedt dan ook een uitgelezen gelegenheid om dit uit te werken.

2.1. Backend

Voor de backend moet je gebruik maken van Node.js gecombineerd met het Express framework. De bedoeling is dat je een REST API server maakt, zodanig dat de frontend applicatie de noodzakelijke gegevens kan opvragen ahv. REST operaties.

Je zal uiteraard een databank (MySQL, PostgreSQL, ...) moeten aanmaken en deze koppelen aan je Node.js backend. Je mag hiervoor gebruik maken van een ORM zoals Prisma (besproken in de lessen). Je krijgt hiervoor reeds een relationeel gegevensmodel (terug te vinden in de bijlagen) mee waarvan je kan vertrekken (scaffolding), maar je mag dit model aanvullen met bijkomende informatie.



2.2. Frontend toepassingen (student, lector)

Voor de frontend toepassingen moet je gebruik maken van React: ontwikkel met React de user interfaces die gebruikt zullen worden door de studenten enerzijds en de lector anderzijds. De user interface is verder qua styling vrij te ontwerpen: je hebt in dit opzicht absolute vrijheid (laat je creativiteit werken ;-)).

De frontend applicaties versturen requests naar de backend om data op te vragen/aan te maken, …

3. Structuur

3.1. Structuur backend

Voor de structuur van je backend is het aan te raden om gebruik te maken van de mappenstructuur die we tijdens de lessen gezien hebben:

- config/
- controllers/
  - index.js
  - users_controller.js
- middlewares/
  - index.js
  - auth_middleware.js
- routes/
  - index.js
  - users_route.js
- prisma/
  - schema.prisma
- app.js

Let op de naamgeving van je files in Node.js. Best practice maken we gebruik van snake_case voor onze files bvb. `users_routers.js`.

Gebruik ook de nieuwere ES6 syntax en gebruik dus nergens het `var` keyword.

3.2. Structuur frontend applicaties

Ook voor de structuur van je frontend applicaties is het aan te raden om gebruik te maken van de volgende mappenstructuur:

- public/
- src/
  - assets/
    - images/
  - components/
  - contexts/
  - screens/
  - store/
  - utils/
- App.js
- index.js

Let ook op de naamgeving van React components; deze verloopt steeds via Pascal casing, bvb. `HomeComponent.jsx`.

We gebruiken functional components in React, en dus geen class components, omdat we gebruik maken van hooks.

4. Student view

Dit is de interactieve user interface waarvan een student gebruik maakt om (deel)opdrachten die lopen, te bekijken en om op de vraag tot het afwerken ervan te reageren. De student kan in deze view met andere woorden alle opdrachten en opdrachtelementen bekijken en aangeven of hij/zij voor de hele opdracht of een deel (een set deelopdrachten) meedoet, opgegeven heeft of klaar is. Eenmaal de opdracht gestart is, tellen zichtbaar timers af naast de opdracht en eventuele deelopdrachten die tonen waaraan op het moment gewerkt moet worden en hoeveel tijd er nog overblijft om dit te doen.

Eerst wordt connectie gemaakt met de server, vergelijkbaar met hoe dit gebeurt bij Kahoot. De student moet een URL intikken om naar de app te gaan en een gedeelde pincode invoeren die de sessie identificeert. Eénmaal de connectie naar de server gemaakt is, ziet de student enkel het volgende voor zich:

1. Beschrijving opdracht (Read-Only)
2. Beschrijving opdrachtelement(en) (Read-Only)
3. Aantal minuten/seconden resterend op niveau van de opdracht (Read-Only)
4. Aantal minuten/seconden resterend op niveau van het opdrachtelement (Read-Only)
5. Operaties om extra tijd te vragen +1 / +5 / +10 minuten voor het lopende opdrachtelement.
6. "Handje opsteken" om aan te duiden dat je als student een vraag hebt.
7. Tekstveld waarbij je als student deze vraag kan formuleren (beperkt tot 255 karakters, vergelijkbaar met een Twitter tweet).
8. "Mutually exclusive radio buttons" met slechts een van de volgende statussen tegelijk geselecteerd: **bezig**, **doet niet mee**, **geeft op**, **klaar**.

Een enkele webpagina volstaat.

5. Host view

Dit is de interactieve user interface voor de lector, welke idealiter geprojecteerd wordt op een groot scherm zodat de studenten ook het overzicht zien. Met deze interface kan de lector een opdracht selecteren, deze starten/stoppen, opdrachtelementen verlengen met het gemiddelde aantal minuten dat gevraagd werd door de studenten, een overzicht bekijken van wie al klaar is en wie niet meedoet, zien welke vragen studenten in welke volgorde hebben, …

De host view moet volgende functionaliteiten aanbieden:

1. Een opdracht met eventuele deelopdrachten selecteren.
2. Een opdracht met opdrachtelementen starten en stoppen - de geconnecteerde clients volgen automatisch hierbij mee (er telt na het starten van de opdracht een klokje af naast de lopende opdracht en de lopende deelopdracht).
3. De resterende tijd van de opdracht en/of opdrachtelementen verhogen wanneer met een gemiddelde waarde in minuten duidelijk wordt dat de studenten extra tijd nodig hebben. De

klokjes bij de studenten worden onmiddellijk aangepast en tellen af voor de aangepaste resterende tijd.

4. Het verzoeken om extra minuten door studenten aan- of uitzetten.
5. Een overzicht tonen van wie al klaar is, wie niet meedoet, … (bvb. een kleurovergang van rood naar groen met een oplopend percentage erin).
6. Een overzicht tonen van openstaande opdrachtelementen en eventueel opdrachten en opdrachtelementen deactiveren (overslaan), waarbij de tijden op het overzicht en de student UI automatisch aangepast worden.

Een enkele webpagina volstaat.

6. Administratieve view

De administratieve view maakt het mogelijk om opdrachten en opdrachtelementen te beheren (toevoegen, wijzigen, verwijderen). De groepen studenten moeten niet via de UI beheerd kunnen worden: een knop met een oplaadmogelijkheid vanuit het meegeleverde .CSV voorbeeld is voldoende.

Groepstudent is een tussentabel.

7. Werking systeem

Het studentopvolgingsysteem werkt op de volgende manier. De beschrijving die volgt, laat toe om de entiteiten van de databank beter te koppelen aan de functionaliteit van het systeem.

7.1. Student

De student is een entiteit (een element op databank-niveau) in het studentopvolgingsysteem. De student behoort tot een groep met eventuele andere studenten. Een student heeft de mogelijkheid om de opdrachten te bekijken en te selecteren in het systeem. Elke opdracht heeft ook opdrachtelementen (deeltaken) die gekoppeld worden aan de student. De koppeling gebeurtdoor middel van het rapport. Alles wat de student ingeeft, zoals bij het stellen van een vraag, het vragen om extra tijd, het aangeven of hij/zij meedoet, dat hij/zij klaar is, wordt dus gekoppeld per opdrachtelement via het rapport en dus opgeslagen in de databank.

Voor elke opdracht en de opdrachtelementen moet de resterende tijd getoond worden.

Voor bonuspunten kan je gebruik maken van websockets om het systeem meer een realtime aspect te geven.

Een bijkomende extra kan hier ook zijn om te werken met een login systeem zodanig dat de student niet telkens een sessie moet starten aan de hand van een url en een pincode.

7.2. Lector

De lector is geen aparte entiteit in het studentopvolgingsysteem: elke lector heeft zijn eigen databank, eventueel per vak, met daarin opdrachten en de opvolging ervan. De lector krijgt een volledige aparte applicatie (deze applicatie kan geprojecteerd worden) die hem de volgende functionaliteiten biedt.

- De lector krijgt een volledig overzicht van alle opdrachten.
- Per opdracht moet hij alle opdrachtelementen kunnen zien.
- Per opdrachtelement moet hij alle rapporten kunnen bekijken (automatisch is dit gekoppeld aan een student).
- Per opdrachtelement kan hij dus beoordelen of de studenten extra tijd krijgen, kan hij alle vragen bekijken en kan hij de status van de studenten opvolgen.
- De lector moet ook de openstaande opdrachtelementen kunnen terugvinden en kunnen deactiveren zodanig deze niet meer gemaakt hoeven te worden.
- De lector moet ook nog de mogelijkheid hebben om het vragen achter extra minuten uit te schakelen.

Hier kan je als extra terug websockets integreren om het systeem reactiever te maken.

7.3. Admin

In de administratieve view is het de bedoeling dat je via CRUD operaties entiteiten kan raadplegen/aanmaken/updaten/verwijderen. Dit is ofwel een aparte applicatie ofwel kan deze geïntegreerd worden in de lectorapplicatie als een aparte module. Er wordt gevraagd minstens opdrachten en deelopdrachten volledig beheerbaar te maken via de administratieve user interface.

Voor bonuspunten kan je alle entiteiten via de UI beheerbaar maken, dus ook het beheer van studenten en groepen van studenten, de rapportering, … . Dit is echter strikt genomen niet nodig om te slagen: studenten en groepen moeten wel minstens opgeladen kunnen worden vanuit een .csv bestand (voorbeeld van een dergelijk bestand wordt meegeleverd).

8. Extras

De extras die hier opgesomd staan zijn niet limitatief en dus mag je steeds andere extras toevoegen aan je project. De extras zijn niet verplicht maar leveren bonuspunten op als deze goed geïmplementeerd zijn (je kan dus nooit punten verliezen hiermee).

- **Authenticatie**
  - Je kan authenticatie implementeren in uw Node.js/React applicatie als extra. Maak wel gebruik van veilige en robuuste methodes zoals `bcrypt` - voor het hashen van wachtwoorden en bvb JWT tokens voor de tokens. Je kan de authenticatie state in je React app bijhouden met de Redux store of met een context hook.
- **Websockets**
  - Websockets om de applicatie meer realtime te maken. Door gebruik te maken van `socket.io` (Node.js) en `socket.io-client` (React).
- **CSV-upload**
  - Mogelijkheid om via CSV testen met vragen en duurtijden te uploaden. Toevoegen in de databank via de Node.js REST API.
- **Beheer via UI van studenten en groepen**
  - Opdrachten en deelopdrachten zijn verplicht beheerbaar via UI. Studenten en groepen studenten worden zeker opgeladen. Beheer van studenten en groepen studenten via UI levert bonuspunten op.
- **…**
  - Eventueel andere extras die je wilt toevoegen.

Vermeld in een bestand (tekstbestand/README.MD/…) welke extras je toegevoegd hebt en stuur dit mee in het gezipte project dat je upload op Chamilo.

9. Deadline

Je zal je project op Chamilo indienen onder de opdracht van het project. Na het indienen van de applicatie zal je dit project verdedigen. Voor de verdediging zullen er tijdsloten op Chamilo aangemaakt worden zodanig dat jullie hier 1 kunnen uitkiezen.

Ik verwacht een gezipt project **zonder** de `node_modules` map. **BELANGRIJK:** De map moet jouw naam bevatten dus het formaat moet als volgt zijn `ACHTERNAAM_Voornaam` dus bvb. `BRECKX_David`.

minimaal volgende mappen/applicaties worden verwacht: **backend**, **student**, **lector**, **admin**. In de **backend** map zit het Node.js project in. In de **student** map zit een React applicatie met de view voor de student, in de **lector** map zit een React applicatie met de view voor de lector, in de **admin** map zit een React applicatie met de view voor de admin (dit kan je eventueel ook in de lector applicatie aanmaken met een aparte url bvb. /admin, dan heb je geen aparte **admin** React applicatie/map meer nodig).

De deadline ligt vast dus dit betekent dat je niet zal kunnen indienen na de deadline! Dus organiseer je goed zodanig dat je niet in tijdsnood komt.

9.1. Eerste kans

Deadline voor de **eerste kans** ligt op **8 Januari 2023 - 23h59**

De verdediging voor de **eerste kans** ligt onder voorbehoud op **9 Januari 2023 - Op afspraak**

9.2. Finale kans

Je kan de finale kans enkel meedoen als je tijdens de eerste kans al iets hebt ingediend of als je gewettigde afwezigheid kan aantonen. Als je dus niets hebt ingediend tijdens de eerste kans, kan je in principe het recht op de finale kans kwijt spelen.

Deadline voor de **finale kans** ligt op **22 Januari 2023 - 23h59**

De verdediging voor de **finale kans** ligt onder voorbehoud op **23 Januari 2023 - Op afspraak**

10. Vragen?

Bij vragen kan je steeds een mailtje sturen naar david.breckx@hogent.be

Appendix A: Relationeel model

```
CREATE DATABASE IF NOT EXISTS StudentOpvolging;
USE StudentOpvolging;
CREATE TABLE `groep` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Naam` varchar(255) NOT NULL,
  `Aanmaakdatum` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `Geldig` int NOT NULL DEFAULT '1',
  PRIMARY KEY (`ID`);
);

CREATE TABLE IF NOT EXISTS `groep` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Naam` varchar(255) NOT NULL,
  `Aanmaakdatum` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `Geldig` int NOT NULL DEFAULT '1',
  PRIMARY KEY (`ID`)
);

CREATE TABLE IF NOT EXISTS `groepstudent` (
  `GroepID` int NOT NULL,
  `StudentID` int NOT NULL,
  `Aanmaakdatum` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `Geldig` int NOT NULL DEFAULT '1',
  PRIMARY KEY (`StudentID`,`GroepID`),
  KEY `GroepID` (`GroepID`),
  CONSTRAINT `groepstudent_ibfk_1` FOREIGN KEY (`GroepID`) REFERENCES `groep` (`ID`),
  CONSTRAINT `groepstudent_ibfk_2` FOREIGN KEY (`StudentID`) REFERENCES `student`
(`ID`)
);

CREATE TABLE IF NOT EXISTS `opdracht` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Naam` varchar(512) NOT NULL,
  `Geldig` int NOT NULL DEFAULT '1',
  `Aanmaakdatum` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `Naam_UNIQUE` (`Naam`)
);

CREATE TABLE IF NOT EXISTS `opdrachtelement` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `OpdrachtID` int NOT NULL,
  `Beschrijving` varchar(4096) NOT NULL,
  `Minuten` int NOT NULL DEFAULT '1',
  `Geldig` int NOT NULL DEFAULT '1',
  `Aanmaakdatum` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`ID`,`AanmaakDatum`),
```

```
  KEY `OpdrachtID` (`OpdrachtID`),
  CONSTRAINT `opdrachtelement_ibfk_1` FOREIGN KEY (`OpdrachtID`) REFERENCES `opdracht`
(`ID`)
);

CREATE TABLE IF NOT EXISTS `rapport` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `StudentID` int NOT NULL,
  `OpdrachtElementID` int NOT NULL,
  `Status` int NOT NULL DEFAULT '0',
  `ExtraMinuten` int NOT NULL DEFAULT '0',
  `Aanmaakdatum` datetime DEFAULT CURRENT_TIMESTAMP,
  `Geldig` int NOT NULL DEFAULT '1',
  PRIMARY KEY (`ID`),
  KEY `StudentID` (`StudentID`),
  KEY `OpdrachtElementID` (`OpdrachtElementID`),
  CONSTRAINT `rapport_ibfk_1` FOREIGN KEY (`StudentID`) REFERENCES `student` (`ID`),
  CONSTRAINT `rapport_ibfk_2` FOREIGN KEY (`OpdrachtElementID`) REFERENCES
`opdrachtelement` (`ID`)
);

CREATE TABLE IF NOT EXISTS `student` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Code` varchar(45) NOT NULL,
  `Gebruikersnaam` varchar(45) NOT NULL,
  `Familienaam` varchar(45) NOT NULL,
  `Voornaam` varchar(45) NOT NULL,
  `Sorteernaam` varchar(90) NOT NULL,
  `Email` varchar(90) NOT NULL,
  `Aanmaakdatum` datetime DEFAULT CURRENT_TIMESTAMP,
  `Geldig` int NOT NULL DEFAULT '1',
  PRIMARY KEY (`ID`)
);

CREATE TABLE IF NOT EXISTS `vraagstudent` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `RapportID` int NOT NULL,
  `Beschrijving` varchar(255) NOT NULL,
  `Aanmaakdatum` datetime DEFAULT CURRENT_TIMESTAMP,
  `Geldig` int DEFAULT '1',
  PRIMARY KEY (`ID`),
  KEY `RapportID` (`RapportID`),
  CONSTRAINT `vraagstudent_ibfk_1` FOREIGN KEY (`RapportID`) REFERENCES `rapport`
(`ID`)
);
```

11. Stappenplan

Het ontwikkelen van de applicaties ziet er bijvoorbeeld ongeveer als volgt uit:

1. Maak de databank aan onder MySQL aan de hand van bovenstaande DDL.
2. Schrijf een node-applicatie die het mogelijk maakt het aangeleverde voorbeeldbestand met studenten en groepen op te laden in de databank; pas desgewenst de layout van de databank aan (wijzigingen, aanvullingen, … zijn toegelaten).
3. [OPTIONEEL] Schrijf een node applicatie die het aangeleverde voorbeeldbestand van een opdracht met deelopdrachten oplaadt in de databank (als je dit niet doet, moet je manueel testgegevens voorzien en/of eerst de verplicht te maken forms onder React voor het beheer van (deel)opdrachten implementeren). Dit kan ook een onderdeel zijn van dezelfde node-applicatie die studenten en groepen oplaadt.
4. Ontwerp en implementeer de Web API die toelaat om de gegevens in de databank te beheren en op te vragen en de noodzakelijke functionaliteit te voorzien; vergeet CORS en Helmet niet.
5. Implementeer als een aparte React applicatie de administratieve UI voor de lector (minstens beheer van (deel-)opdrachten). Dit kan desnoods een gescheiden en apart onderdeel worden van de React applicatie die de UI voor de lector vormt (zie volgende stap).
6. Implementeer als een aparte React applicatie de UI voor de lector (selectie opdracht met deelopdrachten, starten / stoppen en opvolgen van de geselecteerde en eventueel lopende opdracht, desactivatie deelopdrachten, verhogen of verlagen van benodigde tijd voor de (deel)-opdrachten, …).
7. Implementeer als een aparte React applicatie de UI voor de student (aangeven status 'bezig', 'opgegeven', 'afgewerkt' per (deel-)opdracht, ingave eventuele vraag, aanvraag bijkomend benodigde tijd, …).
8. Integreer het opladen van studenten en groepen in de administratieve UI, alsook het optionele opladen van opdrachten en deelopdrachten.
9. [OPTIONEEL] Implementeer extra’s voor bonuspunten.

Referenties

- [1] Node.js documentatie. https://nodejs.org/en/
- [2] Express framework documentatie. https://expressjs.com
- [3] Prisma client/CLI documentatie. https://www.prisma.io
- [4] Helmet documentatie. https://helmetjs.github.io
- [5] CORS uitleg. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- [6] CORS package. https://github.com/expressjs/cors#readme
- [7] Web Auth documentatie. https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API
- [7] JWT documentatie. https://jwt.io
- [8] JWT package. https://github.com/auth0/node-jsonwebtoken#readme
- [10] socket.IO documentatie. https://socket.io
- [11] React documentatie. https://reactjs.org/docs/getting-started.html
- [12] React bèta documentatie. https://beta.reactjs.org
- [13] Redux documentatie. https://redux.js.org
- [14] Redux toolkit documentatie. https://redux-toolkit.js.org