Maxime Coens (202186204)

- [ ] Front-end Web Development
  - GitHub repository
  - Online versie
- [x] Web Services: GITHUB URL
  - GitHub repository
  - Online versie

Logingegevens

Gebruiker ADMIN:

- Gebruikersnaam/e-mailadres: admin@sportapp.be
- Wachtwoord: Admin123

Gebruiker TESTER:

- Gebruikersnaam/e-mailadres: e2e-testing@sportapp.be
- Wachtwoord: Tester123

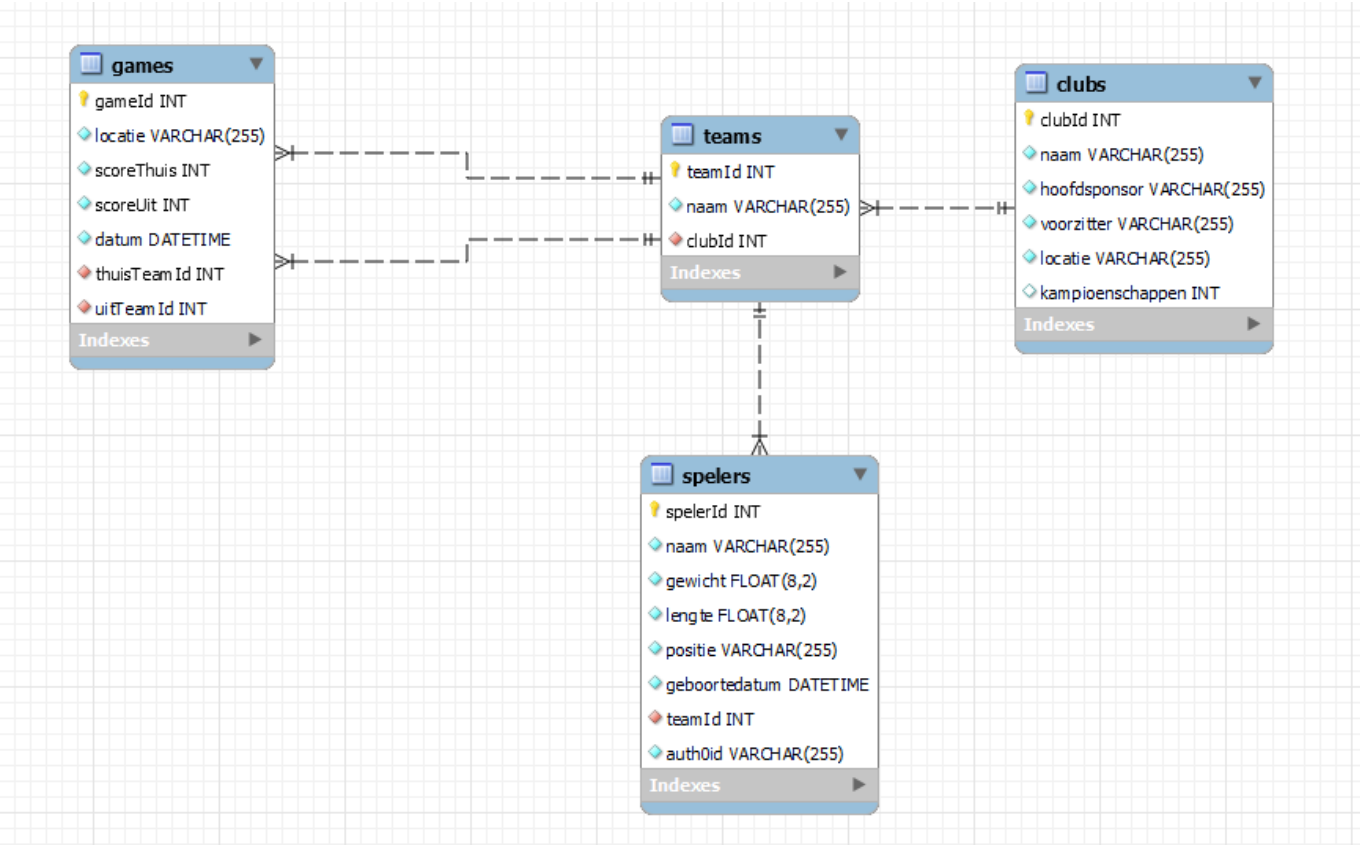
Gebruiker USER:

- Gebruikersnaam/e-mailadres: user@sportapp.be
- Wachtwoord: Sportappuser123

Projectbeschrijving

Over dit semester heen heb ik een API gemaakt in NodeJS. Aangezien ik enkel het vak Web Services opneem, heb ik geen Front-end en zal mijn gemaakte API niet als back-end kunnen werken voor een zelf gemaakte Front-end. Ik heb er voor gekozen een API te maken in de lijn met mijn passie, sport/basketbal. Er zijn vier tabellen gemaakt in het ERD (zoals je ziet in de foto hier onder): clubs, teams, spelers en games. De API bevat dus gegevens over spelers of sporters die aan een sport doen in clubverband. Hierbij heb ik er voor gekozen dit specifiek voor basketbal te maken. Een speler kan zich registreren aan de hand van een team dat hij zichzelf kan in plaatsen, dit team behoort tot een club en heeft de mogelijkheid om games te spelen. Deze games bestaan door 2 teams die tegen elkaar zullen spelen. Een speler die de rol admin heeft kan specifiek teams, clubs en games aanmaken, verwijderen en updaten.

ERD



Screenshots

Geen screenshots, niet nodig doordat ik enkel het vak `Web Services` had en dus geen Front-end development. Hierdoor kan ik geen screenshots van een applicatie tonen.

Behaalde minimumvereisten

Front-end Web Development

Aangezien ik geen Front-end doe worden volgende velden opengelaten.

- **componenten**
  - ☐ heeft meerdere componenten - dom & slim (naast login/register)
  - ☐ definieert constanten (variabelen, functies en componenten) buiten de component
  - ☐ minstens één form met validatie (naast login/register)
  - ☐ login systeem (eigen of extern zoals bv. Auth0)
- **routing**
  - ☐ heeft minstens 2 pagina's (naast login/register)
  - ☐ routes worden afgeschermd met authenticatie en autorisatie
- **state-management**
  - ☐ meerdere API calls (naast login/register)
  - ☐ degelijke foutmeldingen indien API call faalt
  - ☐ gebruikt useState enkel voor lokale state

    - [ ] gebruikt Context, useReducer, Redux... voor globale state

- **hooks**
    - [ ] kent het verschil tussen de hooks (useCallback, useEffect...)
    - [ ] gebruikt de hooks op de juiste manier

- **varia**
    - [ ] een aantal niet-triviale testen (unit en/of e2e en/of ui)
    - [ ] minstens één extra technologie
    - [ ] duidelijke en volledige README.md
    - [ ] volledig en tijdig ingediend dossier

Web Services

- **datalaag**
    - [x] voldoende complex (meer dan één tabel)
    - [x] één module beheert de connectie + connectie wordt gesloten bij sluiten server
    - [x] heeft migraties
    - [x] heeft seeds

- **repositorylaag**
    - [x] definieert één repository per entiteit (niet voor tussentabellen) - indien van toepassing
    - [x] mapt OO-rijke data naar relationele tabellen en vice versa

- **servicelaag met een zekere complexiteit**
    - [x] bevat alle domeinlogica
    - [x] bevat geen SQL-queries of databank-gerelateerde code

- **REST-laag**
    - [x] meerdere routes met invoervalidatie
    - [x] degelijke foutboodschappen
    - [x] volgt de conventies van een RESTful API
    - [x] bevat geen domeinlogica
    - [x] degelijke authorisatie/authenticatie op alle routes

- **varia**
    - [x] een aantal niet-triviale testen (min. 1 controller >=80% coverage)
    - [x] minstens één extra technologie
    - [x] duidelijke en volledige `README.md`
    - [x] maakt gebruik van de laatste ES6-features (object destructuring, spread operator...)
    - [x] volledig en tijdig ingediend dossier

Projectstructuur

Web Services

De API werd opgebouwd via een gelaagde structuur. Waarbij we een `repositorylaag` hebben waar alle SQL statements worden uitgevoerd voor het opvragen en manipuleren van de data die werd meegegeven in de https request. Daarnaast hebben we de `servicelaag` die vooral voor de logica bestaat en dus de juiste data en opdrachten zal doorgeven repositorylaag. En ten slotte is er een `restlaag` waar de https request worden gelezen en daarna doorgegeven aan de servicelaag. Ook validaties worden daar uitgevoerd en de permissies zullen worden gecheckt.

In de `datalaag` vinden we de seedings en migrations voor de databank. Ook de mock-data, die niet meer wordt gebruikt staat daar ook nog. Daarnaast wordt ook de databank daar geïnitialiseerd en maken we hier een knexinstantie aan.

Extra technologie

Web Services

Als extra technologie heb ik gekozen voor `Swagger`, dit is een technologie, waarmee je jouw API-structuur kunt beschrijven en lezen. Hier werd er dus een interactieve API-documentatie gebouwd. Via de `Swagger UI` krijgen we een duidelijker beeld van hoe de API in elkaar zit aan de hand van voorbeelden, probeermogelijkheden van requests, ...

npm packages

- Swagger: https://www.npmjs.com/package/swagger
- Swagger UI: https://www.npmjs.com/package/swagger-ui

Testresultaten

Web Services

Van teams, clubs en games werden voor voor elke table de GET, GET /:id, POST, PUT en DELETE HTTPS methods getest op de juiste werking van deze methodes. Bij spelers werden enkel de GET, GET /:id en DELETE HTTPS methods getest. Aan de hand van mock data die telkens in de database 186204mc_test werd geplaatst, werden enkele opties getest zoals: de juiste status weergeven bij een request, zorgen dat alle data juist werd teruggegeven bij de methodes door het kijken naar de body van het response.

```
PS C:\Users\maxim\OneDrive\Documenten\HoGent 2223\WEB SERVICES\project-sportapp\2223-webservices-maximecoens> yarn test:coverage
yarn run v1.22.19
$ npx env-cmd -f .env.test jest --coverage
 PASS  __tests__/rest/spelers.spec.js
 PASS  __tests__/rest/clubs.spec.js
 PASS  __tests__/rest/teams.spec.js
 PASS  __tests__/rest/games.spec.js
-----------------------------------|---------|----------|---------|---------|-----------------------------------
File                               | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------------------------|---------|----------|---------|---------|-----------------------------------
All files                          |   78.52 |    49.56 |   77.77 |   79.04 |
 __tests__                         |     100 |      100 |     100 |     100 |
  index.js                         |     100 |      100 |     100 |     100 |
 src/service                       |   86.71 |    72.72 |   81.48 |    87.7 |
  club.js                          |   96.55 |       80 |     100 |   96.42 | 21
  game.js                          |   96.55 |       80 |     100 |   96.42 | 21
  health.js                        |   66.66 |      100 |       0 |     100 |
  speler.js                        |   65.71 |    57.14 |   57.14 |    64.7 | 21,28-37,42-51,55-57
  team.js                          |   96.55 |       80 |     100 |   96.42 | 20
-----------------------------------|---------|----------|---------|---------|-----------------------------------

Test Suites: 4 passed, 4 total
Tests:       18 passed, 18 total
Snapshots:   0 total
Time:        4.255 s, estimated 5 s
Ran all test suites.
Done in 6.99s.
PS C:\Users\maxim\OneDrive\Documenten\HoGent 2223\WEB SERVICES\project-sportapp\2223-webservices-maximecoens>
```

Gekende bugs

Web Services

Meertaligheid
De code van de backend staat niet geschreven in 1 taal. Dit is een combinatie van Nederlands en Engels. De code zelf werd in het engels geschreven maar enkele Error messages en andere loggings werden in het Nederlands geschreven.