Jules Goubert (202291428)

- [ ] Front-end Web Development
- [x] Web Services:
  - Github Reository
  - Laatste online versie

Logingegevens

localhost

admin:

- e-mailadres: bob@gmail.com
- Wachtwoord: 12345678

user:

- e-mailadres: alice@gmail.com
- Wachtwoord: 12345678

online

admin:

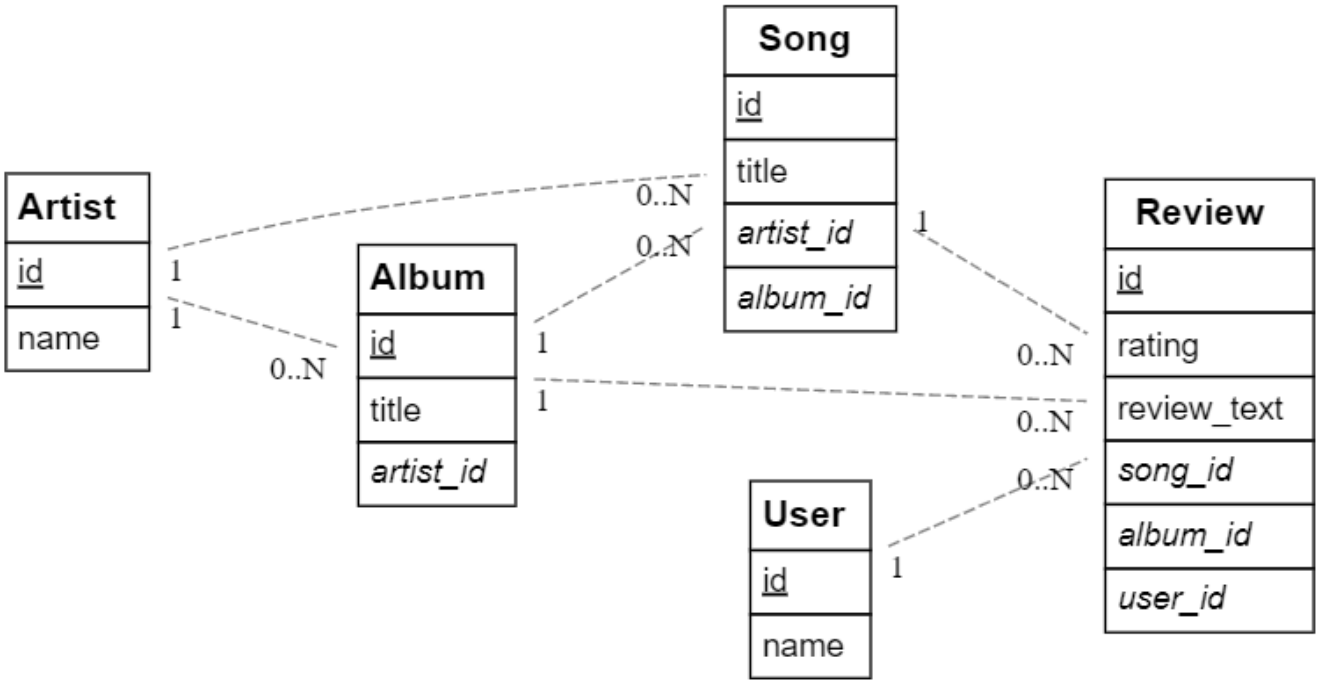
- e-mailadres: admin@gmail.com
- Wachtwoord: 12345678

user:

- e-mailadres: bob@gmail.com
- Wachtwoord: 12345678

Projectbeschrijving

Ik heb ervoor gekozen om een CRUD API te maken waar gebruikers reviews kunnen schrijven voor liedjes en albums. Mijn databank bevat vijf entiteiten, een User, Review, Song, Album en Artist. Voor elk van deze entiteiten heb ik alle CRUD operaties geïmplementeerd. Mijn EERD ziet er als volgt uit:



API calls

Users

- `GET /api/users`: alle users ophalen
- `GET /api/users/:id`: user met een bepaald id ophalen
- `POST /api/users/login`: user inloggen
- `POST /api/users/register`: user registreren
- `PUT /api/users/:id`: user met een bepaald id updaten
- `DEL /api/users/:id`: user met een bepaald id verwijderen

Artists

- `GET /api/artists`: alle artists ophalen
- `GET /api/artists/:id`: artist met een bepaald id ophalen
- `POST /api/artists`: artist aanmaken
- `PUT /api/artists/:id`: artist met een bepaald id updaten
- `DEL /api/artists/:id`: artist met een bepaald id verwijderen

Albums

- `GET /api/albums`: alle albums ophalen
- `GET /api/albums/:id`: album met een bepaald id ophalen
- `POST /api/albums`: album aanmaken
- `PUT /api/albums/:id`: album met een bepaald id updaten
- `DEL /api/albums/:id`: album met een bepaald id verwijderen

Songs

- `GET /api/songs`: alle songs ophalen
- `GET /api/songs/:id`: song met een bepaald id ophalen

- `POST /api/songs`: song aanmaken
- `PUT /api/songs/:id`: song met een bepaald id updaten
- `DEL /api/songs/:id`: song met een bepaald id verwijderen

Reviews

- `GET /api/reviews`: alle reviews ophalen
- `GET /api/reviews/:id`: review met een bepaald id ophalen
- `POST /api/reviews`: review aanmaken
- `PUT /api/reviews/:id`: review met een bepaald id updaten
- `DEL /api/reviews/:id`: review met een bepaald id verwijderen

Health

- `GET /api/health/ping`: geeft pong terug als de API actief is
- `GET /api/health/version`: versie van de API ophalen

Behaalde minimumvereisten

- **datalaag**
  - ☑ voldoende complex (meer dan één tabel, 2 een-op-veel of veel-op-veel relaties)
  - ☑ één module beheert de connectie + connectie wordt gesloten bij sluiten server
  - ☑ heeft migraties - indien van toepassing
  - ☑ heeft seeds
- **repositorylaag**
  - ☑ definieert één repository per entiteit (niet voor tussentabellen) - indien van toepassing
  - ☑ mapt OO-rijke data naar relationele tabellen en vice versa - indien van toepassing
- **servicelaag met een zekere complexiteit**
  - ☑ bevat alle domeinlogica
  - ☑ bevat geen SQL-queries of databank-gerelateerde code
- **REST-laag**
  - ☑ meerdere routes met invoervalidatie
  - ☑ degelijke foutboodschappen
  - ☑ volgt de conventies van een RESTful API
  - ☑ bevat geen domeinlogica
  - ☑ geen API calls voor entiteiten die geen zin hebben zonder hun ouder (bvb tussentabellen)
  - ☑ degelijke authorisatie/authenticatie op alle routes
- **algemeen**
  - ☑ er is een minimum aan logging voorzien
  - ☑ een aantal niet-triviale integratietesten (min. 1 controller >=80% coverage)
  - ☑ minstens één extra technologie

- [x] maakt gebruik van de laatste ES-features (async/await, object destructuring, spread operator...)
- [x] duidelijke en volledige README.md
- [x] volledig en tijdig ingediend dossier en voldoende commits

Projectstructuur

> Hoe heb je jouw applicatie gestructureerd (mappen, design patterns...)?

Ik heb gewerkt met een `src` map waar alle logica van de applicatie staat. In die `src` map staan de `core, data, repsitory, rest en service` mappen.

- In de `service` map staat de business logica.
- In de `data` wordt de data beheert.
- In de `rest` map staan alle API endpoints beschreven.
- In de `repository` map staat de communicatie tussen de API en de databank beschreven
- In de `core` map staan algemene zaken die nodig zijn voor authenticatie, authorisatie, validatie, ...

Extra technologie

De extra technologie die ik gebruikt heb is de Spotify API. Ik heb deze gebruikt om de `songs, artists en albums` tabellen in de databank al in te vullen met de 100 populairste liedjes van Spotify met hun respectievelijke artiesten en albums bij het aanmaken van de databank. Om deze technologie te gebruiken heb ik een Spotify developer account moeten aanmaken. Met dat account kreeg ik een Client ID en een Secret ID waarmee ik de Spotify API kon aanspreken. Ik heb voor deze technologie geen npm package moeten downloaden, ik heb wel gebruik gemaakt van de ingebouwde `Fetch API`.

Testresultaten

Ik heb integratietesten geschreven voor elk endpoint van mijn API. Voor elk endpoint wordt er een test uitgevoerd voor een scenario waarin er niets zou moeten mislopen en één of meerder testen voor scenario's

waarin er wel iets moet mislopen. Dit zijn de resultaten van mijn testen:

```
$ env-cmd -f .env.test jest --runInBand --coverage
 PASS  __tests__/users.spec.js
 PASS  __tests__/reviews.spec.js
 PASS  __tests__/songs.spec.js
 PASS  __tests__/albums.spec.js
 PASS  __tests__/artists.spec.js
 PASS  __tests__/health.spec.js
-------------------|---------|----------|---------|---------|-------------------------------------------
File               | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------|---------|----------|---------|---------|-------------------------------------------
All files          |    91.9 |    81.02 |   96.07 |    91.9 |
 repository        |   96.47 |    82.05 |   87.87 |   96.47 |
  album.js         |   96.38 |     87.5 |   85.71 |   96.38 | 32-34
  artist.js        |   92.85 |      100 |   83.33 |   92.85 | 8-10
  review.js        |   97.82 |       70 |     100 |   97.82 | 118-120
  song.js          |   97.27 |    77.77 |   85.71 |   97.27 | 50-52
  user.js          |   94.33 |    85.71 |   85.71 |   94.33 | 8-10
 rest              |     100 |      100 |     100 |     100 |
  album.js         |     100 |      100 |     100 |     100 |
  artist.js        |     100 |      100 |     100 |     100 |
  health.js        |     100 |      100 |     100 |     100 |
  index.js         |     100 |      100 |     100 |     100 |
  review.js        |     100 |      100 |     100 |     100 |
  song.js          |     100 |      100 |     100 |     100 |
  user.js          |     100 |      100 |     100 |     100 |
 service           |   79.05 |    70.58 |     100 |   79.05 |
  _handleDBError.js|   31.81 |    33.33 |     100 |   31.81 | 7-29,32-38
  album.js         |   83.82 |       75 |     100 |   83.82 | 26-27,33-34,40-44,51-52
  artist.js        |   92.15 |       80 |     100 |   92.15 | 25-26,34-35
  health.js        |     100 |      100 |     100 |     100 |
  review.js        |   77.27 |    65.38 |     100 |   77.27 | 31-36,43-44,54-58,71-72,87-91,95-102,115-116
  song.js          |   72.82 |    72.72 |     100 |   72.82 | 27-28,33-37,43-44,53-59,64-70,77-78
  user.js          |   90.98 |    69.56 |     100 |   90.98 | 22-24,31-34,88-89,97-98
-------------------|---------|----------|---------|---------|-------------------------------------------

Test Suites: 6 passed, 6 total
Tests:       122 passed, 122 total
Snapshots:   0 total
Time:        9.479 s
Ran all test suites.
Done in 10.85s.
```

Gekende bugs

Er zitten geen gekende bugs in mijn API.