Dossier

- Student: Jesse Vaerendonck
- Studentennummer: 202399060
- E-mailadres: mailto:jesse.vaerendonck@student.hogent.be

- Demo: <DEMO_LINK_HIER>
- GitHub-repository: https://github.com/HOGENT-frontendweb/frontendweb-2425-Jessevdonck.git
- Front-end Web Development
    - Online versie: https://frontendweb-kskcolle.onrender.com
- Web Services:
    - Online versie: https://kskcolle-backend.onrender.com

Logingegevens

Lokaal

Standaard User

- Gebruikersnaam/e-mailadres: testuser@mail.com
- Wachtwoord: 12345678
-

Admin User

- Gebruikersnaam/e-mailadres: adminuser@mail.com
- Wachtwoord: 12345678

Online

Standaard User

- Gebruikersnaam/e-mailadres: testuser@mail.com
- Wachtwoord: 12345678
-

Admin User

- Gebruikersnaam/e-mailadres: adminuser@mail.com
- Wachtwoord: 12345678

Projectbeschrijving

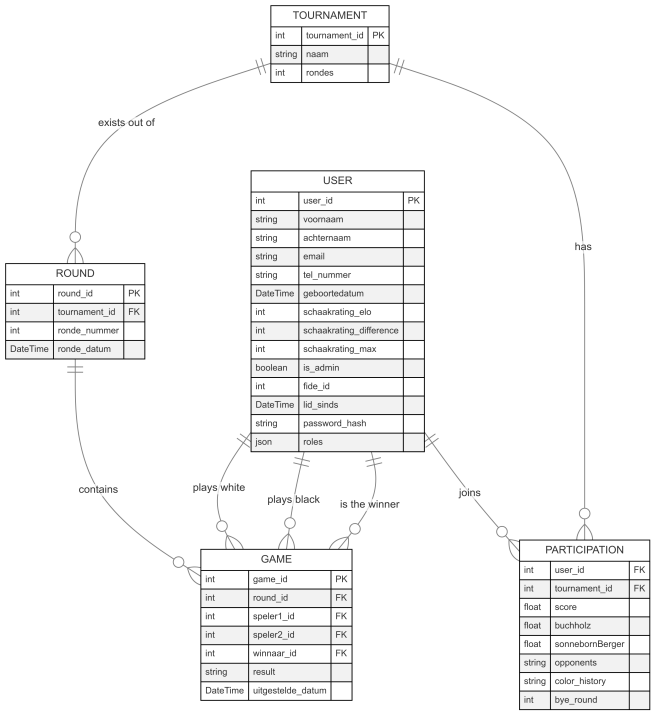
Voor mijn project heb ik gekozen om de site van de schaakclub KSK Colle een "make over" te geven. Deze site dient in de eerste plaats om de nieuwkomers duidelijk te maken wat de schaakclub doet en waarvoor het staat. Ten tweede dient de site ook voor leden- en toernooibeheer. Dit houdt in dat een admin via een dashboard moet kunnen maken dat er toernooien en leden aangemaakt, aangepast en verwijderd moeten kunnen worden. Dit gebeurt allemaal via een apart dashboard waar alleen mensen met de rol "admin" toegang tot hebben. Normale leden hebben overigens een apart profiel waar ze snel hun "rating" en 5 meest

recent gespeelde games kunnen zien. Ingelogde gebruikers hebben dan ook toegang om andere profielen van spelers waar te nemen.

ERD in Mermaid:

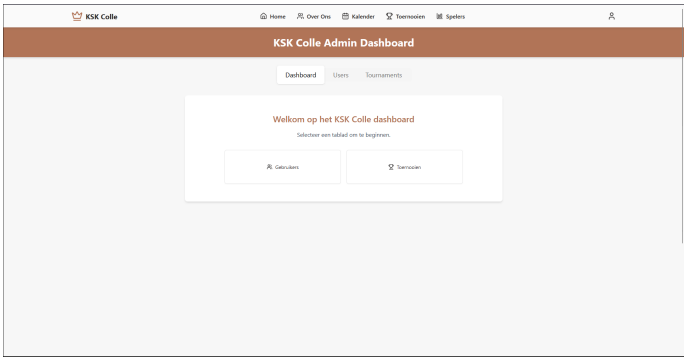
```
erDiagram

    TOURNAMENT {
        int tournament_id PK
        string naam
        int rondes
    }

    USER {
        int user_id PK
        string voornaam
        string achternaam
        string email
        string tel_nummer
        DateTime geboortedatum
        int schaakrating_elo
        int schaakrating_difference
        int schaakrating_max
        boolean is_admin
        int fide_id
        DateTime lid_sinds
        string password_hash
        json roles
    }

    PARTICIPATION {
        int user_id FK
        int tournament_id FK
        float score
        float buchholz
        float sonnebornBerger
        string opponents
        string color_history
        int bye_round
    }

    ROUND {
        int round_id PK
        int tournament_id FK
        int ronde_nummer
        DateTime ronde_datum
    }

    GAME {
        int game_id PK
        int round_id FK
        int speler1_id FK
```

```
        int speler2_id FK
        int winnaar_id FK
        string result
        DateTime uitgestelde_datum
    }

    %% Relaties
    TOURNAMENT ||--o{ ROUND : "exists out of"
    USER ||--o{ PARTICIPATION : "joins"
    TOURNAMENT ||--o{ PARTICIPATION : "has"
    ROUND ||--o{ GAME : "contains"
    USER ||--o{ GAME : "plays white"
    USER ||--o{ GAME : "plays black"
    USER ||--o{ GAME : "is the winner"
```

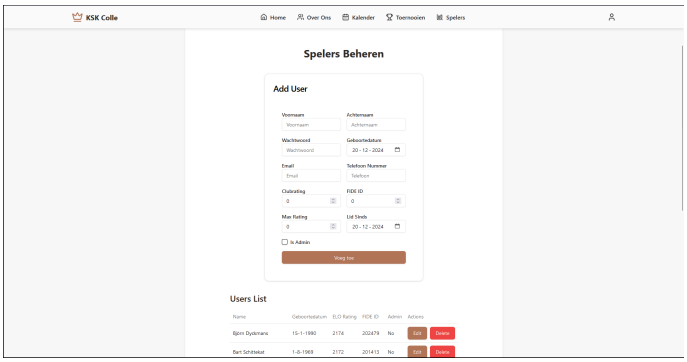


ERD ⬆

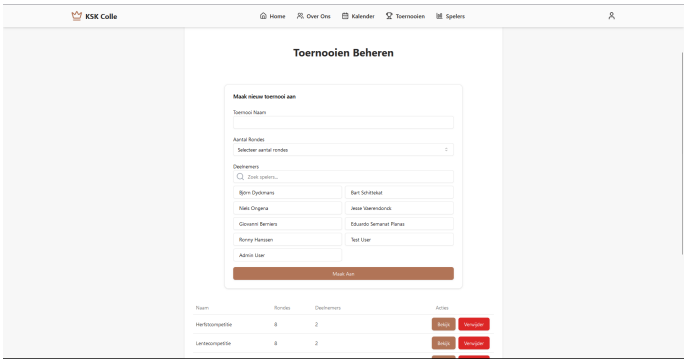
Screenshots



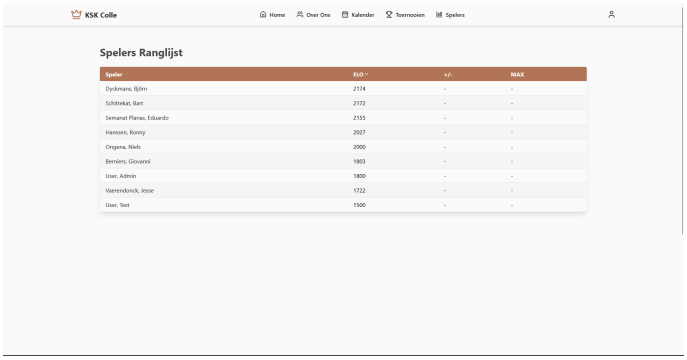
Admin Dashboard Landing Page ⬆



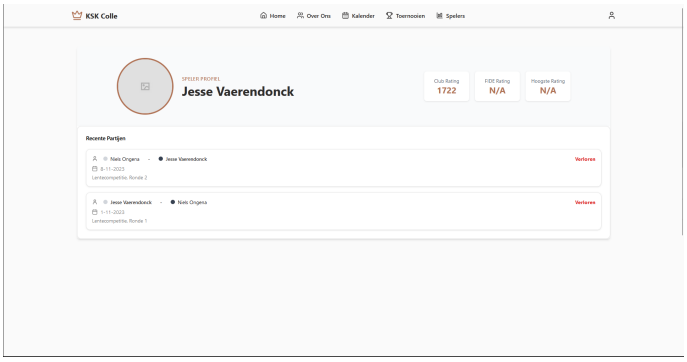
Admin Dashboard User Page ⬆



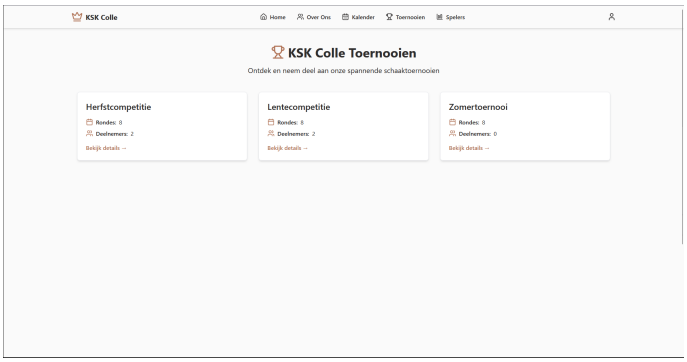
Admin Dashboard Tournament Page ⬆



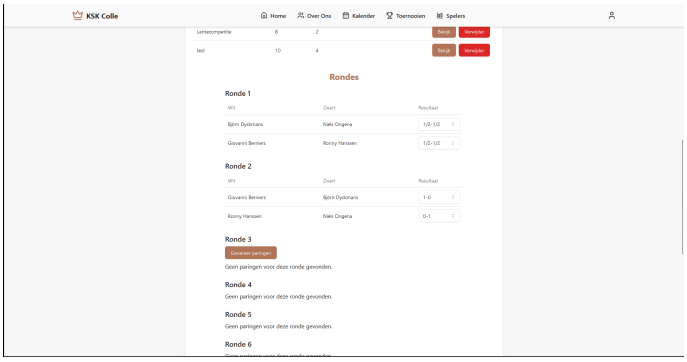
Player Overview ⬆



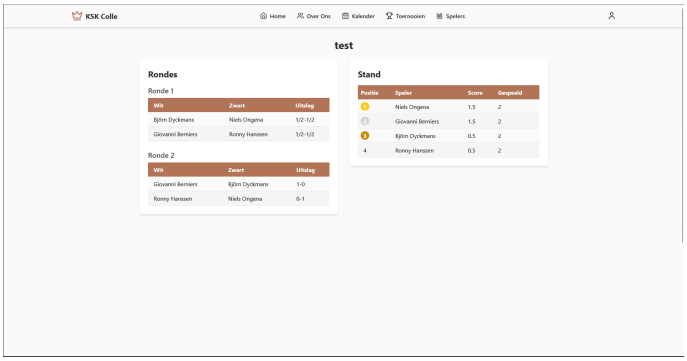
Profile Page ⬆



Tournament Overview ⬆



Generate Pairings And Edit Results ⬆



Tournament Standings ⬆

API calls

Health

- `GET /api/ping`: Ping de server
- `GET /api/version`: Geeft de informatie over de server versie

Session

- `POST /api/sessions`: Logt de gebruiker in

Users

- `GET /api/users`: Alle gebruikers ophalen
- `GET /api/users/publicUsers`: Alle gebruikers ophalen zonder sensitieve informatie

- `GET /api/users/:user_id`: Gebruiker met een bepaald ID ophalen
- `GET /api/users/by-name`: Gebruiker met een bepaalde naam ophalen
- `POST /api/users`: Maak een nieuwe gebruiker aan
- `PUT /api/users/:user_id`: Update een bestaande gebruiker
- `PUT /api/users/:user_id/password`: Update het wachtwoord van een bestaande gebruiker
- `DELETE /api/users/:user_id`: Verwijder een specifieke gebruiker

Toernooien

- `GET /api/tournaments`: Alle toernooien ophalen
- `GET /api/tournaments/:tournament_id`: Toernooi met specifiek ID ophalen
- `POST /api/tournaments`: Maak een nieuw toernooi aan
- `POST /api/tournaments/tournament_id/pairings/roundNumber`: Maak paringen aan voor een bepaalde ronde
- `PUT /api/tournaments/:tournament_id`: Toernooi met specifiek ID aanpassen
- `DELETE /api/tournaments/:tournament_id`: Toernooi met specifiek ID verwijderen

Rondes

- `GET /api/rondes/tournaments`: Alle rondes ophalen
- `GET /api/rondes/:tournament_id/rondes`: Geef alle rondes van een bepaald toernooi
- `GET /api/rondes/:round_id`: Geef ronde met een specifiek ID
- `POST /api/tournaments/:tournament_id/rondes`: Maak een nieuwe ronde aan
- `PUT /api/rondes/:round_id`: Update een ronde met specifiek ID
- `DELETE /api/rondes/:round_id`: Verwijder ronde met specifiek ID

Spellen

- `GET /api/spel`: Alle spellen ophalen
- `GET /api/spel/:game_id`: Haal een spel met specifiek ID op
- `GET /api/spel/speler/user_id`: Haal alle spellen op van een specifieke user.
- `POST /api/spel`: Maak een nieuw spel
- `PUT /api/spel/:game_id`: Update een spel met specifiek ID
- `DELETE /api/spel/:game_id`: Verwijder een spel met specifiek ID

Behaalde minimumvereisten

Front-end Web Development

Componenten

- ☑ heeft meerdere componenten - dom & slim (naast login/register)
- ☑ applicatie is voldoende complex
- ☑ definieert constanten (variabelen, functies en componenten) buiten de component
- ☑ minstens één form met meerdere velden met validatie (naast login/register)
- ☑ login systeem

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 gepast state management voor globale state - indien van toepassing

Hooks

- ☑ gebruikt de hooks op de juiste manier

Algemeen

- ☑ een aantal niet-triviale én werkende e2e testen
- ☑ minstens één extra technologie
- ☑ node_modules, .env, productiecredentials... werden niet gepushed op GitHub
- ☑ maakt gebruik van de laatste ES-features (async/await, object destructuring, spread operator...)
- ☑ de applicatie start zonder problemen op gebruikmakend van de instructies in de README
- ☑ de applicatie draait online
- ☑ duidelijke en volledige README.md
- ☑ er werden voldoende (kleine) commits gemaakt
- ☑ volledig en tijdig ingediend dossier

Web Services

Datalaag

- ☑ voldoende complex en correct (meer dan één tabel (naast de user tabel), tabellen bevatten meerdere kolommen, 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 - indien van toepassing
- ☑ mapt OO-rijke data naar relationele tabellen en vice versa - indien van toepassing
- ☑ er worden kindrelaties opgevraagd (m.b.v. JOINs) - indien van toepassing

Servicelaag met een zekere complexiteit

- ☑ bevat alle domeinlogica
- ☑ er wordt gerelateerde data uit meerdere tabellen opgevraagd
- ☑ bevat geen services voor entiteiten die geen zin hebben zonder hun ouder (bv. tussentabellen)
- ☑ bevat geen SQL-queries of databank-gerelateerde code

REST-laag

- ☑ meerdere routes met invoervalidatie
- ☑ meerdere entiteiten met alle CRUD-operaties
- ☑ degelijke foutboodschappen
- ☑ volgt de conventies van een RESTful API
- ☑ bevat geen domeinlogica
- ☑ geen API calls voor entiteiten die geen zin hebben zonder hun ouder (bv. tussentabellen)
- ☑ degelijke autorisatie/authenticatie op alle routes

Algemeen

- ☑ er is een minimum aan logging en configuratie voorzien
- ☑ een aantal niet-triviale én werkende integratietesten (min. 1 entiteit in REST-laag >= 90% coverage, naast de user testen)
- ☑ node_modules, .env, productiecredentials... werden niet gepushed op GitHub
- ☑ minstens één extra technologie die we niet gezien hebben in de les
- ☑ maakt gebruik van de laatste ES-features (async/await, object destructuring, spread operator...)
- ☑ de applicatie start zonder problemen op gebruikmakend van de instructies in de README
- ☑ de API draait online
- ☑ duidelijke en volledige README.md
- ☑ er werden voldoende (kleine) commits gemaakt
- ☑ volledig en tijdig ingediend dossier

Projectstructuur

Front-end Web Development

Ik heb mij gehouden aan hoe men normaal NextJS hiërarchie onderhoudt. Er is een app folder waarin alle page folders zitten. Men heeft een components folder in de root app folder zitten die componenten die over heel de applicatie gebruikt worden bevat. Buiten dat heeft men ook nog componenten per 'page' folder waar men enkel componenten in heeft zitten die specifiek voor die pagina gebruikt worden.

Wat ik persoonlijk het meest handige vind aan NextJS is hoe men makkelijk aan Server Side Rendering kan doen. Ik heb hiervan zoveel mogelijk gebruik gemaakt als het ging over dynamische data die bv. afhankelijk was van authenticatie. In tegenstelling tot de SSR heb ik ook gebruik gemaakt van Client-Side Rendering wanner dit ging over pagina's die intensief steunde op interactie (bv. het dashboard).

De state management kan men vooral zien bij de authenticatie aangezien ik geen 'dark mode' geïntegreerd heb.

Ik heb altijd geprobeerd om me aan de 'best practices' te houden, maar dit was niet altijd even makkelijk omdat je snel "verdrinkt" in je eigen code en bestanden. Ondanks dat vermoed ik wel dat dit grotendeels in orde zal zijn.

Web Services

Ik heb geprobeerd alles overzichtelijk te houden door een zeer strakke en heldere mappenstructuur aan te houden. Zo heb ik een REST laag met alle endpoints en de bijhorende service map waar alle domeinlogica

zich in verschuilt. Er is ook maar 1 bestand dat de server aanmaakt en die steunt dan op helper bestanden zoals bv. installMiddlewares om in dit geval de middlewares te installeren. Alle sensitieve informatie staat net zoals bij het front-end gedeelte in een .env bestand zodat dit niet toegangelijk is voor bezoekers.

Extra technologie

Front-end Web Development

Ik heb gekozen voor NextJS. Dit is een ander framework dan dat we in de les gezien hebben en dit prefereerde ik boven React Native omdat bepaalde technologieën makkelijker te gebruiken zijn in dit framework zoals de routing en SSR en CSR

Web Services

Ik heb hier gekozen voor apiDoc i.p.v. Swagger. Ik vond apiDoc zeer gebruiksvriendelijk om te gebruiken aangezien men de NPM package moest installeren, een config moest toevoegen en dan de documentatie in de bestanden kon toevoegen. Eens je overal de documentatie had bijgezet kon men een simpel commando (apidoc -i src -o apidoc) uitvoeren om het te laten genereren. Dan was het nog kwestie van het toegangelijk te maken via routing en dit was zo simpel als een lijn code toevoegen in het installMiddlewares.ts bestand. (app.use(serve('apidoc'));).

Gekende bugs

Front-end Web Development

Een gekende bug is dat eens de token vervalt dan past de navigatiebalk zich niet aan en lijkt het alsof je nog steeds bent ingelogd.

Web Services

Er zijn geen gekende bugs

Reflectie

Dit was een project waar je grotendeels wat in werd gegooid, maar dit zorgde ervoor dat je echt moest snappen met wat je bezig was in tegenstelling tot "gewone" lessen krijgen. Ik kan dus gerust zeggen dat ik enorm veel bijgeleerd heb, vooral over het backend gedeelte omdat ik persoonlijk vind dat hier toch altijd wat minder informatie online over te vinden is. De cursus aan zich vond ik ook vrij goed, maar ik vind dat het misschien iets te veel "copy en paste" is bij sommige delen zonder een uitgebreide uitleg waarom je bepaalde dingen op een bepaalde manier moet doen. Dit zorgde voor heel wat tijdsverlies als je de "waarom" erachter wou weten.