**Schaken technisch ontwerp**

**Architectuur**

Het schaakprogramma zal geïmplementeerd worden als server-client programma waarbij de rust API de backend van de server zal zijn en de Tauri GUI het front-end. Hierbij zal het programma ‘OOP’ en MVC moeten supporten.

**Backend (Rust API):**

1. Implementatie van de schaken API in Rust.
2. Gebruik van de dependencies (tauri, serde, serde\_json, num-traits, num, dotenv)
3. Gebruik maken van een .env om te bepalen als we gaan testen of niet, de applicatie zal niet maken opstarten als de sleutel ‘testen’ bestaat (ook niet als het op false staat)
4. De schaak move generatie zal supersnel moeten zijn dus we gaan gebruik maken van bitboards (<https://www.chessprogramming.org/Magic_Bitboards>)
5. De movegeneratie en de backend van Tauri zal beiden getest moeten zijn

**Front-end (Tauri gui):**

1. Ontwikkeling van een crossplatform applicatie met Tauri gebaseerd op Html, CSS en Typescript.
2. Manier om met de front-end en backend te praten via Tauri zodat we bijvoorbeeld mogelijke zetten kunnen weergeven op de front-end, stukken kunnen bewegen op de backend en front-end en kunnen laten zien dat de game is gewonnen.
3. Game moet geüpdatet worden in de backend wanneer iets gebeurt in het front-end

**Spellogica**

De kern van het programma omvat de implementatie van de schaakspellogica binnen de rust API waaronder:

* Representatie van het schaakspel gedaan wordt met bitboards en de move generatie gedaan wordt met magic bitboards.
* Zetten van de magic bitboards gevalideerd worden zodat we niet illegale zetten maken die ons schaakmat zouden kunnen zetten
* Testen als het spel beëindigt is door te testen voor gelijkspel of schaakmat, waarbij gelijkspel zal zijn dat een speler geen zetten kan maken en zijn koning niet onder aanval is en bij schaakmat zal de speler ook geen zetten kunnen maken maar zal de koning wel onder aanval zijn

**Gegevensopslag**

Opslag van het spel zal gedaan kunnen worden in een struct, de informatie in de struct zal zijn:

* Welke kant een zet kan maken (dus wit/zwart)
* Informatie waar de stukken huidig staan in bitboards (er moeten 12 bitboards zijn 1 voor elk stuk)
* Informatie over waar alle stukken staan aan de witte en zwarte kant en aan bijde kanten (<https://www.chessprogramming.org/Occupancy>)
* List van opgeslagen game states, we zullen een game state ELKE zet moeten opslaan zodat ELKE zet die gedaan is ongedaan zou kunnen worden

**Beveiliging**

Doordat momenteel alles lokaal gebeurt zal ik niet inzien waarom de http request beveiligt zullen zijn doordat alles toch al op de pc staat en gehackt zou kunnen worden. Natuurlijk zal dit beter moeten zijn wanneer je dit online zou gaan doen maar hierbij vind ik dat onnodig.

Er zullen toch wel een aantal dingen beveiligt moeten worden aan de server kant zoals:

* Dezelfde speler mag niet meerdere keren een zet maken
* Dezelfde speler mag niet test functies aanroepen die open staan in tauri, dit zal beveiligt moeten worden door een .env variabel genaamd TESTING. Dit mag niet aanstaan wanneer er niet getest wordt dus het programma zal niet mogen starten wanneer deze .env aanstaat