Portfolio S3

Tim Meijvogel

Inhoudsopgave

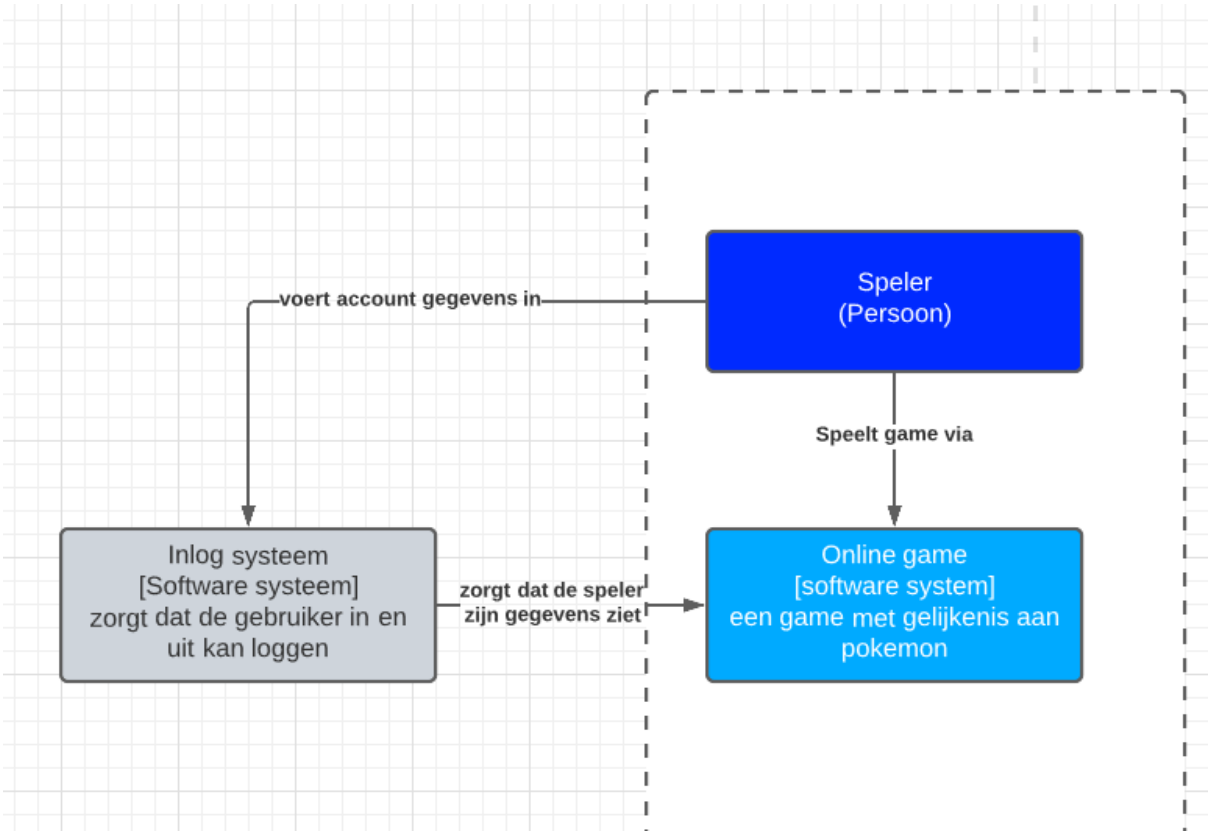
Web application

dit semester werd er van ons verwacht een distributed software systeem te maken. dit is een systeem waar de front en backend zijn verdeeld, ook moesten we gebruik maken van een javascript framework voor de frontend.

Om deze applicatie te maken ben ik begonnen met het opzetten van mijn requirements, in de vorm van user stories. Deze heb ik voor mijzelf in een trello board gezet. Ik heb de keuze gemaakt om trello te gebruiken, aangezien ik alleen werk en dus geen geavanceerde planning functies nodig heb, daarnaast vind ik trello overzichtelijk. Zodra ik al mijn user stories had toegevoegd heb ik een c1 en c2 model gemaakt, om de doorsnede van mijn programma weer te geven. Mijn programma was nog klein genoeg dat hiermee te zien was wat de bedoeling was. Ook heb ik hier technische keuzes bij gezet, zodat het duidelijk was welke framework en talen ik zou gebruiken.

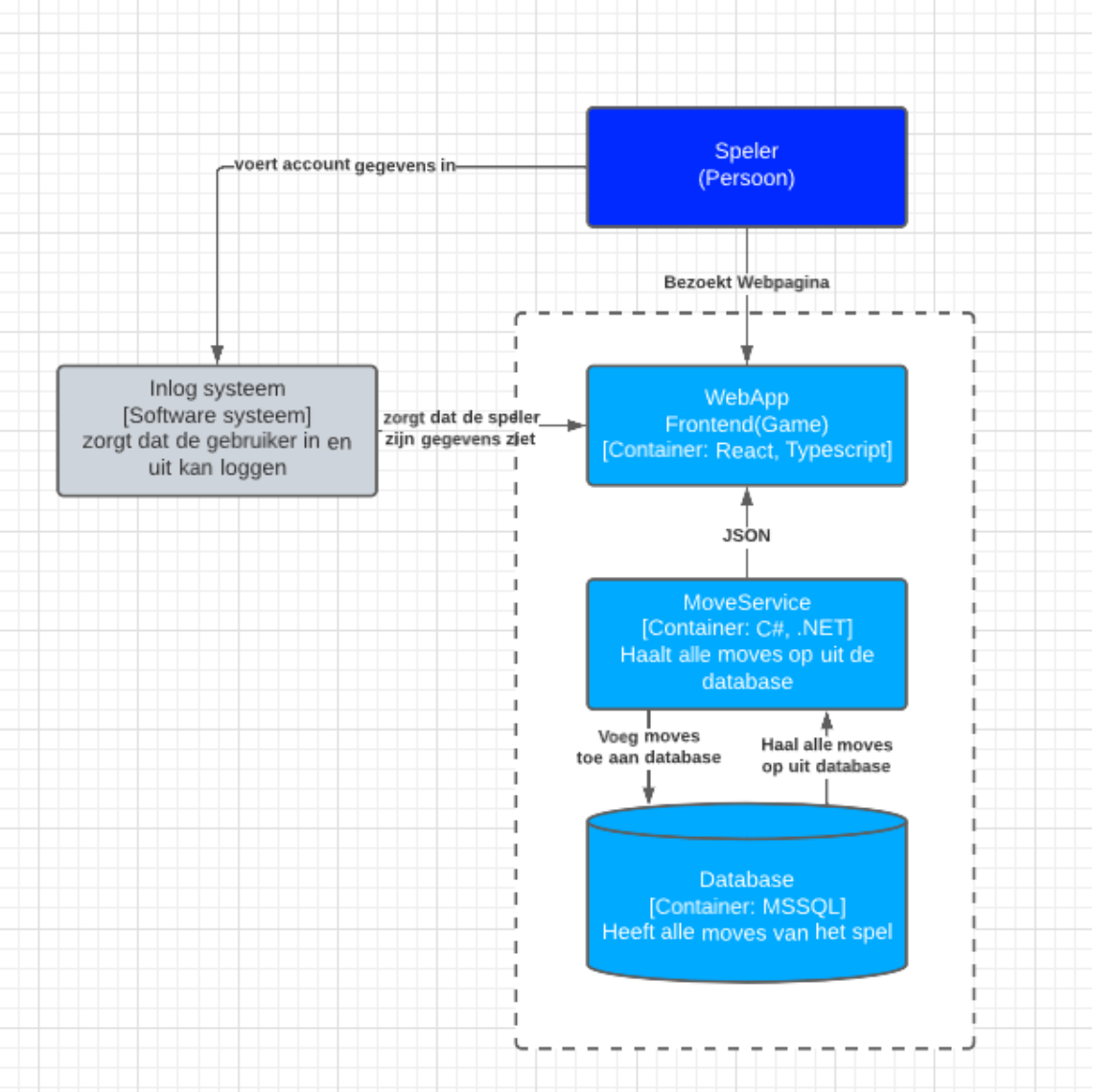
C1

In het C1 model heb ik gevisualiseerd wat ik wil dat mijn applicatie gaat doen op de meest simpele manier. De speler komt op de website en heeft de keus om als gast te spelen of in te loggen en zijn gegevens te laden.



C2

In dit C2 model is te zien hoe mijn huidige programma er uit ziet. Ik heb een eigen service gemaakt, mijn move service. Dit is een CRUD api die alle moves uit de database doorstuurd naar de frontend. Deze api heeft alle crud functionaliteiten en kan dus ook dingen toevoegen, verwijderen en bewerken in de database. Ik heb 1 extern systeem toegevoegd en dit is een inlogsysteem (auth0).



Technische keuzes:

Architectuur:

ik heb de keus gemaakt om in dit project te werken met microservices, ik heb hier nog niet mee gewerkt en denk dat dit een goed moment is om te leren hoe dit werkt. Daarnaast waren in mijn ogen voordelen ten opzichte van een monolithische applicatie, zoals:
Het feit dat niet alle services hoeven te runnen om de applicatie werkend te houden. Een microservice architectuur is ook makkelijker uit te breiden, dit komt omdat een software developer niet het volledige programma hoeft te begrijpen om iets toe te voegen en het risico van het uitbreiden ook kleiner is. Dit komt omdat, wanneer er iets kapot gaat in de toegevoegde service de applicatie nog steeds kan werken. Iedere service kan ook in een andere taal worden geschreven, dit betekend dat je mensen die geen c# kunnen niet uit zal sluiten

Backend:

de backend moest een OO taal zijn, ik heb gekozen om c# te gebruiken. Ik heb al veel kleine applicaties Geschreven in c# en wil mijn kennis uitbreiden. daarnaast had ik nog niet met .NET 6 gewerkt, en wilde ik met de nieuwste versie bijblijven. Een andere reden om in c# te werken was dat ik nog nooit een API had geschreven of ermee had gewerkt. Ik wilde leren hoe dit werkte in c# en achteraf bleek dit vrij simpel met het gebruik van entity framework.

```
[HttpPost]
0 references
public async Task<ActionResult<List<Move>>> AddMove(Move move)
{
    context.Moves.Add(move);
    await context.SaveChangesAsync();
    return Ok(await context.Moves.ToListAsync());
}

[HttpPut]
0 references
public async Task<ActionResult<List<Move>>> UpdateMove(Move req)
{
    var dbMove = context.Moves.Find(req.Id);
    if (dbMove == null)
        return BadRequest("Move not found");

    dbMove.Name = req.Name;
    dbMove.Damage = req.Damage;
    dbMove.Speed = req.Speed;
    dbMove.Accuracy = req.Accuracy;
    dbMove.MoveType = req.MoveType;

    await context.SaveChangesAsync();

    return Ok(await context.Moves.ToListAsync());
}
```

Frontend:

Het front-end framework dat ik heb gekozen is react. Ik heb gekozen voor dit framework omdat, ik al een beetje met react typescript heb gewerkt in semester 2. Ik had dus al een klein beetje kennis van react, maar ik wilde dit uitbreiden, aangezien mijn kennis stopte bij een “hello world” applicatie. Ik heb dit semester besloten ook weer te werken met typescript, dit is om dezelfde reden dat ik voor react heb gekozen. Maar naast de reden dat ik mijn kennis wil uitbreiden over typescript, vond ik het ook prettig dat typescript meteen fouten weergeeft.

```tsx
const App = () => {

  const { isLoading } = useAuth0();

  if (isLoading) {
    return (<Loading />)
  }

  return (
    <div>
      <Navbar />
      <div>
        <Profile />
        <AllMoveButtons />
      </div>
      <div><AddMove/></div>

    </div>

  );

}
```

```tsx
export default function GameScreen() {

    const [Moves, SetMoves] = useState<IMove[]>([]);
    useEffect(() => {
        const api = async () => {
            SetMoves(await GetAllMoves());
        };
        api();
    }, []);

    return (
        <div className="App">
            <div>
                <Container>
                    <Row >
                        {Moves.map((move) => {
                            console.log(Moves);
                            console.log(move.moveType);
                            return <Col className="MoveButtons" xs={3}>
                                <Move name={move.name} damage={move.damage} speed={move.speed} accura
                            </Col>

                        })}
                    </Row>
                </Container>
            </div>
        </div>
    );
}
```

Database:

Wat moest mijn database kunnen doen ?

Het enige dat mijn database moest kunnen doen waren crud operaties. De reden hiervoor was, omdat ik alleen moves op hoefde te slaan en op te vragen Dit was het belangrijkste voor mijn project. Dit is mogelijk voor alle 3 de databases waar ik naar heb gekeken.

Welke database heb ik gebruikt

Als database heb ik gebruik gemaakt van MSSQL, de grootste reden hiervoor is omdat, ik al eerder gebruik heb gemaakt van dit systeem en het prettig vond werken. Ook wist ik dat ik gebruik wilde maken van een sql database. Ik heb gekeken naar MSSQL, MySQL en PostgreSQL, terwijl ik hiernaar keek wist ik al dat ik gebruik wilde maken wan MSSQL.

Mijn front en backend zijn hier te vinden:
Backend: https://github.com/S3IP/MoveService
Frontend: https://github.com/S3IP/elementalcombat

Een demonstratie van mijn App: https://youtu.be/ctzjlr4cHmM

Software quality

De kwaliteit van de software die je schrijft is belangrijk, het moet leesbaar zijn, je moet consistent zijn met coding conventions en het moet werken. om dit aan te tonen heb je verschillende tools, zoals tests en programma's als sonarcloud.

Welke testen zijn er

unit testing

Een unit test is een test die een unit in je programma kan controleren. Door de bronnen die ik heb gelezen ben ik van mening dat een unit test een class kan testen, aangezien dit in mijn ogen 1 unit aan code is. Ik heb online ook meningen voorbij zien komen dat een unit test bedoeld is voor een functie of een service.
Zelf heb ik geen gebruik gemaakt van unit tests. Ik heb deze keuze gemaakt omdat, ik een api heb die alleen CRUD functies uitvoert en ik unit testing geen effectieve manier vind om dit te testen. Het gebruik van swagger vind ik effectiever aangezien dit precies hetzelfde resultaat geeft.

integration testing

Integration tests testen meerdere units code tegelijk. Ik zie dit als meerdere classes die samen worden getest. Maar integration tests kunnen ook worden gebruikt om meerdere services samen te testen.
Aangezien ik maar 1 zelfgemaakte service, met maar 1 class heb die alleen maar crud operaties uitvoert heb ik geen integration tests geschreven voor mijn project.

System testing

End to End testing:
Dit zijn de testing die ik belangrijk vind voor mijn applicatie, aangezien ik een game wilde maken. Een game hangt af van user input en dit is precies wat ik kan testen met end to end tests. Ik heb cypress gebruikt om deze tests te schrijven. Helaas heb ik heb in mijn frontend niet veel functionaliteit, dus de test laat nu alleen zien of iets laad. Hierna wilde ik een test schrijven om te zien of een gebruiker in kon loggen, maar dit was helaas niet mogelijk, aangezien ik niet de elementen van de auth0 inlog pagina kon gebruiken in de tests (zover ik weet).

```
describe("renders the page", () => {
    it("renders correctly", () => {
        cy.visit("/");
        cy.get("div").should('have.class', "App");
    })
})

describe("mock login process", () => {
    it("logs in correctly", () => {
        cy.get(".justify-contend-end > .btn").click();

    })
})
```

In het groepsproject heb ik ook tests geschreven en deze hadden wel user input. In de tests die ik hier had geschreven wordt er gekeken wat er gebeurd na een bepaalde actie van een gebruiker. in dit geval is het een redirect dus ik heb gekeken of de gebruiker naar de juiste pagina werd gebracht.

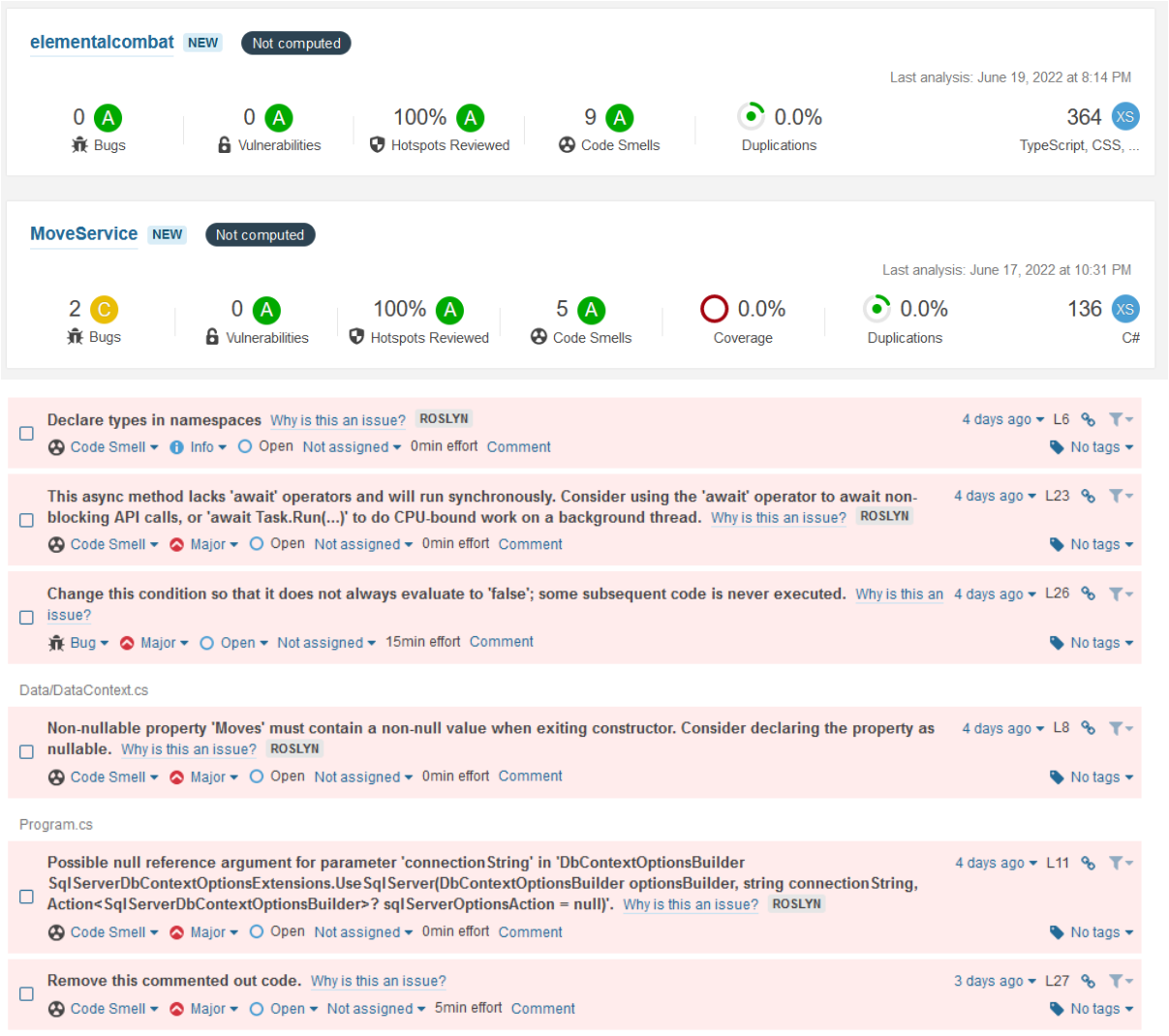
```
describe("redirect works correctly", () => {
    it("redirects to kitchen correctly", () => {
        cy.visit("/");
        cy.get('button').contains('Kitchen').click();
        cy.url().should('include', '/kitchen')
    })

    it("redirects to bar correctly", () => {
        cy.visit("/");
        cy.get('button').contains('Bar').click();
        cy.url().should('include', '/bar')

    })
})
```

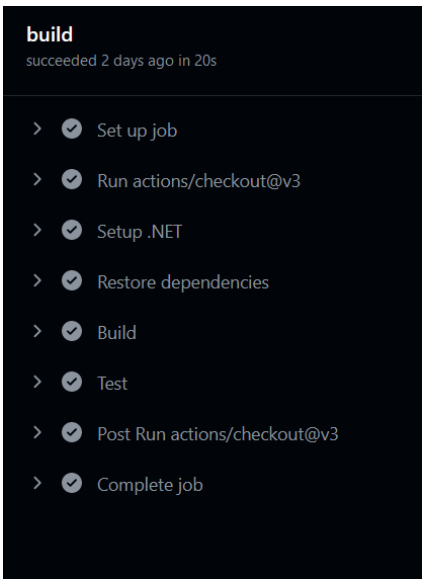
Static code analysis

Ik heb sonar cloud gebruikt voor mijn static code analysis, dit heb ik gedaan, omdat het opzetten van sonar cloud voor je repositories duidelijke stappen had om te volgen.
Het doel voor de static code analysis is om te zorgen dat ik problemen snel en makkelijk op kan sporen en deze kan oplossen voordat ik mijn branches merge.
Zoals te zien in de afbeelding heeft mijn backend 2 bugs, dit komt door een if statement die altijd hetzelfde resultaat zal geven en de naam van een functie in mijn controller. Verder is er ook te zien dat er een aantal code smells zijn.

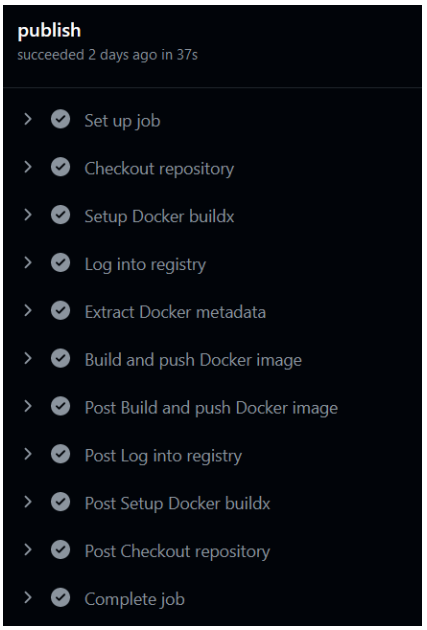


CI/CD

Mijn plan was om ci op te zetten in github actions. dit heb ik gedaan door yml files toe te voegen aan mijn repositories. hierin heb ik gezet dat bij iedere push naar de master branch verschillende commands uitgevoerd moeten worden om te kijken of de nieuwste versie werkt zoals verwacht en niet voor problemen zorgt. dit wordt gedaan voor zowel mijn front als backend.



Voor cd was mijn plan om azure te gebruiken, ik heb hier even naar gekeken en over gelezen, maar door een inschattingsfout die ik maakte over hoeveel tijd ik nog had heb ik besloten om docker te gebruiken. Deze keuze heb ik gemaakt omdat, er veel mensen om mij heen hiermee werkte of al mee gewerkt hadden. Dit betekende dat ik deze mensen om hulp kon vragen en onderzoek met ze kon doen.



De CI/CD pipeline die ik nu heb aangelegd voor mijn individuele project volgt de volgende stappen.
Bij elke push naar de master branch begint github actions mijn YML files uit te voeren, deze beginnen met het downloaden van dependencies en het builden van mijn applicatie, hierna worden de tests in mijn applicatie uitgevoerd (in mijn frontend heb ik deze stap overgeslagen door problemen met cypress, ik had geen tijd meer om deze op te lossen). Wanneer dit allemaal is geslaagd wordt er een docker image gecreëerd en deze wordt geüpload naar docker hub. Ik kan ze hierna pullen in docker desktop en ze met mijn docker compose file runnen.

CI/CD is belangrijk voor software developers, omdat dit veel tijd kan schelen wanneer dit goed is opgezet. Je applicatie wordt gebouwd en testen worden automatisch uitgevoerd. Op deze manier kun je snel zien of de aanpassingen die je hebt gemaakt aan een applicatie voor problemen zorgen op de huidige versie. Ook is het met CI mogelijk om een push af te wijzen als deze voor problemen zou zorgen, dit zorgt dat er nooit een defectieve app in de git repository komt.

Professional

Dit semester moesten we een leerdoel dat “Professional” heette aantonen, dit hield in dat we constant contact hadden met onze stakeholders en feedback bleven vragen, die we ook toepassen, we moesten onze keuzes op een goede manier onderbouwen en research doen.
Aangezien we dit semester op het laatste moment nieuwe docenten kregen voor ons individuele project ga ik me voor dit leerdoel focussen op de research en het onderbouwen van keuzes.
Je keuzes op een goede manier onderbouwen komt vooral voor in dit portfolio, maar onderbouwen was ook een groot deel van de research die we moesten doen. Er waren 2 research documenten die we dit semester moesten maken, een ging over security, de ander was een onderwerp dat we zelf mochten kiezen. Mijn eigen research had als hoofdvraag “welke variatie van het elo systeem is geschikt voor mijn game?” en voor de security research was mijn hoofdvraag “Waarom zou je een authenticatie service gebruiken in plaats van een eigen login systeem?”.
Voor deze research onderzoeken heb ik me vooral aan 4 stappen gehouden. Deze stappen waren het bedenken en beantwoorden van een hoofdvraag, het bedenken en beantwoorden van deelvragen, mijn informatie valideren en het maken van een conclusie.
Ook heb ik gekeken naar het DOT framework. Ik heb een aantal categorieën gebruikt van het domein library. De categorieën die voor mij belangrijk waren, waren vooral “best good and bad practices” (Organise activities aimed at sharing experiences. Depending on your topic, many best, good and bad practices may be available.) en “available product analysis” (Identify existing solutions that may solve the problem (or a part thereof) you are trying to fix with your solution. Decide if it is worth the effort to recreate their work, or whether it is better to simply buy it from them or embed their work in yours.). Dit komt omdat, ik bij mijn research naar elo veel gedeelde meningen tegen kwam onder spelers van veel games. Dit laat zien dat veel game developers dezelfde goede en foute keuzes maken die ik steeds weer tegen kwam. Door deze twee categorieën ben ik uiteindelijk tot mijn conclusie gekomen.

Reflectie

Wat ging goed

De dingen die ik dit semester goed vond gaan waren voornamelijk op technisch gebied. Voor dit semester kon ik niet werken met een javascript framework en had ik nog nooit een api geschreven. Ik ben blij dat dit me is gelukt en dat ik mijn front en backend met elkaar heb kunnen laten communiceren. Ook had ik veel contact met mijn docent, ook al waren de gesprekken niet altijd even nuttig, ik heb hem iedere week gesproken. Ook ben ik van mening dat ik de docenten die we in de laatste weken hebben gekregen voldoende keer heb gesproken en die feedback die ze hebben gegeven verwerkt.

Wat ging minder goed

Tijdens het individuele project heb ik me te veel gefocust op het leren werken met een gedistribueerd systeem, het maken van een api en het leren gebruiken van een javascript framework. Deze onderwerpen waren dit semester natuurlijk belangrijk, maar door mijn focus hierop heb ik te lang gewacht met de andere leerdoelen, waardoor ik deze in een korte tijd moest weten aan te tonen.

Wat zou ik de volgende keer anders doen

Er zijn veel dingen die ik dit semester anders zou doen, het grootste voorbeeld hiervan is het beginnen aan mijn research. Ik onderschatte dit semester hoe belangrijk de research was die we moesten doen. Ook ga ik minder keuzes maken met de reden dat iets me leuk leek om mee te werken. Een voorbeeld hiervan is react, aan het begin van het semester koos ik deze framework, omdat het me leuk leek. Ik had er een redenering bij gezet, waarvan ik nu weet dat die niet zou voldoen aan het leerdoel “Professional”. Hierdoor moest ik later, nadat ik de keuze al had gemaakt onderzoeken wat voor en nadelen waren en hierop mijn keuze onderbouwen.
Mijn plan voor de volgende keer zou zijn om te beginnen met het opzetten van mijn CI/CD, aangezien dit niet veel tijd hoeft te kosten ik dan zeker weet dat ik de communicatie tussen de images goed kan maken. Hierna zou ik de requirements en diagrammen voor mijn app maken, zodat andere mensen hieraan zouden kunnen werken als ik er niet bij zou zijn. Wanneer dit af is ga ik een simpele applicatie opzetten die aan de minimale eisen voldoet van de leerdoelen. Hierna kan ik op tijd aan de research beginnen en zorgen dat ik genoeg tijd overhoudt om bepaalde aspecten aan te passen waar nodig.