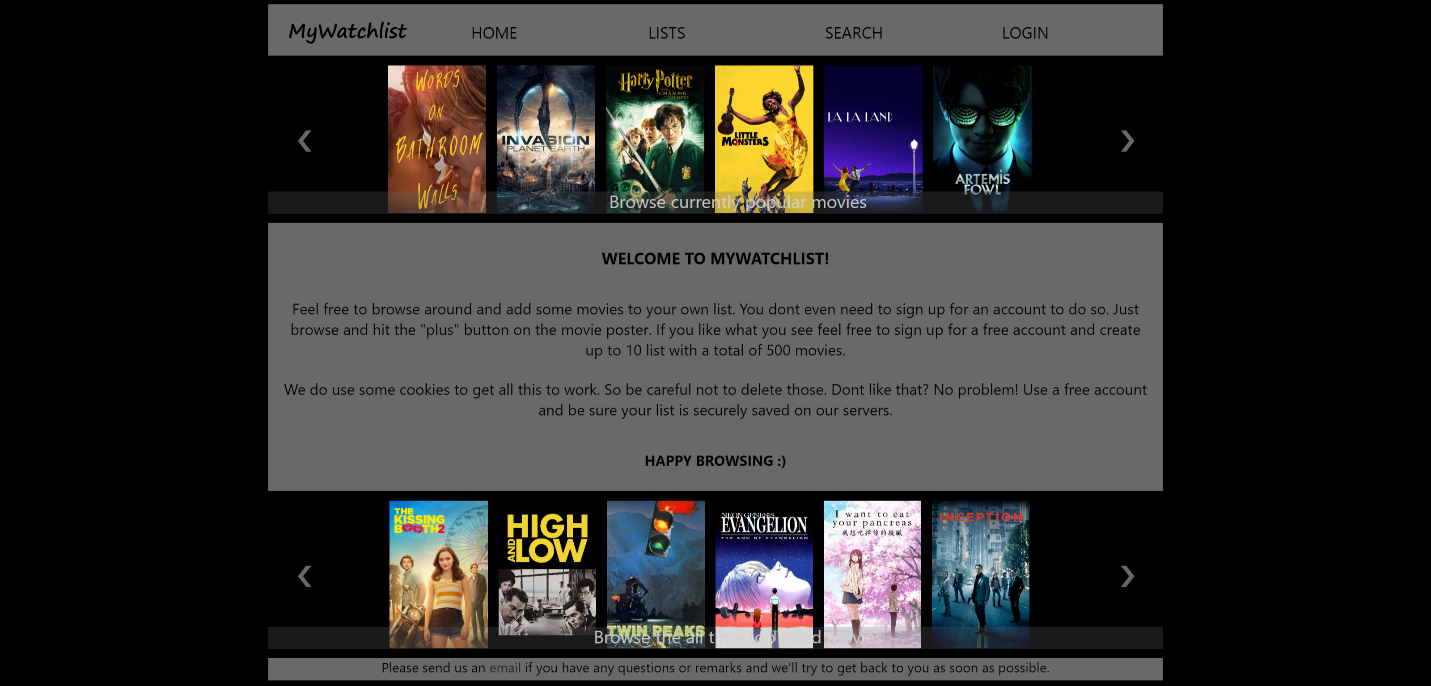
|  |
| --- |
| **Verantwoordingsdocument**  **myWatchlist** |
| **Geschreven door:**  **J.P. Falk**  **Document versie:**  **1.1 / 2020** |
|  |



**Inhoudsopgave**

[1. Introductie 3](#_Toc47286296)

[2. Projectmanagement 4](#_Toc47286297)

[2.1 Projectmanagementmethode 4](#_Toc47286298)

[2.2 Versie controle 4](#_Toc47286299)

[3. Architectuur 5](#_Toc47286300)

[3.1 Frontend 5](#_Toc47286301)

[3.2 Backend 6](#_Toc47286302)

1. **Introductie**

Het doel van dit document is om de motivatie en werkwijze tijdens de ontwikkeling van de applicatie in kaart te brengen. De volgende vragen worden in dit document beantwoord:

1. Welke projectmanagementmethode is toegepast.
2. Hoe is de architectuur van de applicatie opgesteld.
3. Wat zijn de toegepaste technieken in de applicatie.

**2. Projectmanagement**

Vanwege het grote succes van het “agile” werken binnen de industrie is het niet toepassen van deze technieken bijna ondenkbaar. Zo zijn ook in de ontwikkeling van deze applicatie de “agile” principes toegepast. Buiten deze algemene richtlijnen is de focus van de ontwikkeling voornamelijk gelegd op de ontwikkelmethode “Extreme programming”.

**2.1 Projectmanagementmethode**

Er is voor de ontwikkelmethode “Extreme programming” gekozen vanwege de volgende kenmerken van zowel de ontwikkelmethode als het project:

1. Vanwege de relatief korte duur van het project is gekozen om de focus te leggen op de kwaliteit van de geleverde functionaliteit boven de kwantiteit van geleverde functionaliteit in het algemeen.
2. De gelimiteerde omvang van het ontwikkelingsteam sluit goed aan bij de gekozen ontwikkelingsmethode.
3. De ontwikkeling van elke nieuwe functionaliteit wordt gestart nav. de hoogst geldende prioriteit welke om dat moment aanwezig is waardoor de kans van vertragingen in andere functionaliteit, gecreëerd door afhankelijkheid, wordt geminimaliseerd.
4. Dmv. korte sprints waarbij de focus op een enkele functie van de applicatie wordt gelegd die volledig getest wordt voordat men overgaat op de volgende sprint, kan de kwaliteit van de applicatie beter gewaarborgd worden tijdens de ontwikkeling waardoor verrassingen in de vorm van fouten in de code bij de opleving van de applicatie worden geminimaliseerd.

**2.2 Versie controle**

Voor het uitvoeren van versie controle zijn de volgende technieken toegepast:

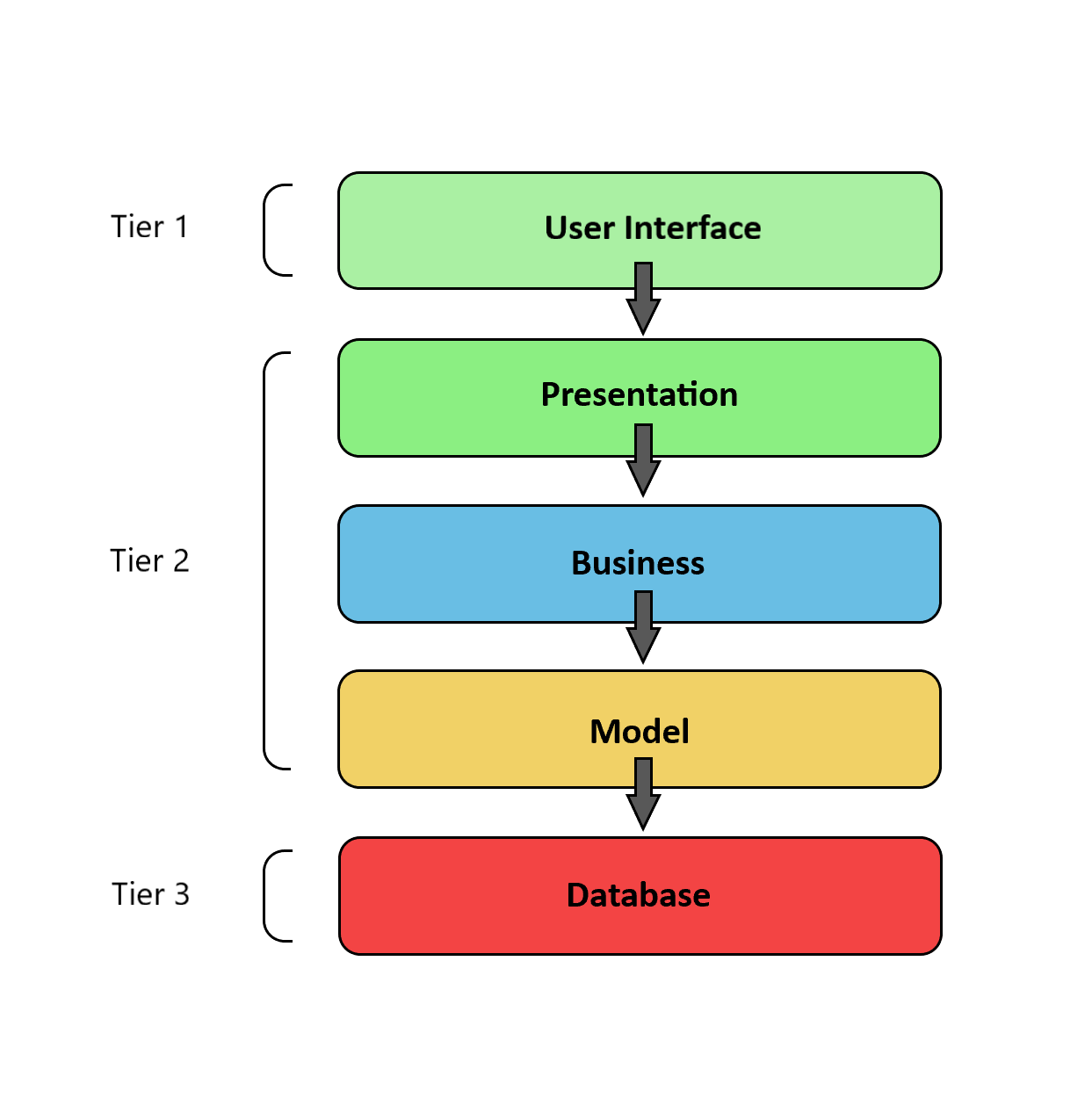
1. Gebruik van “GitHub” via integratie met de “GitHub desktop” applicatie.
2. Gebruik van lokale versie controle in de IDE IntelliJ.
3. Gebruik van een lokale back-up van de ontwikkelomgeving.

Voorafgaande aan de start van de ontwikkeling van een nieuwe functionaliteit werd gedurende het project een nieuwe branch gecreëerd in de GitHub omgeving welke [hier](https://github.com/JochenFalk/gs-myWatchlist-master) is te vinden. Na voltooiing van de relevante functionaliteit werd de branch “ge-merged” met de master.

Voor elke merge met de master werd een lokale backup gemaakt van de ontwikkelomgeving. Tijdens de ontwikkeling van een specifieke functie werd ook gebruik gemaakt van de lokale versie controle in de IDE om recentelijk aangebrachte verandering te kunnen waarnemen.

**3. Architectuur**

De applicatie is opgebouwd tot een multi-tier applicatie mbv. de “Spring Boot Initializr” en maakt gebruik van het “Spring MVC model” en “Maven”, een krachtig project management tool.



Spring MVC is een volledig HTTP-georiënteerd MVC-framework beheerd door het Spring Framework welke gebruik maakt van Servlets. Maven is een build tool welke het ontwikkelen van java-based applicaties vereenvoudigd en tegelijkertijd uniformiteit voorziet aan de ontwikkelomgeving.

De multi-tier benadering zoals hierboven is geïllustreerd faciliteert een logische dataflow binnen de applicatie die dient om de applicatie structuur te geven en om “security boundaries” op te stellen tussen de aanwezige lagen en tiers. Zo communiceert elke laag of tier enkel met de direct aanliggende laag. Dmv. deze benadering wordt directe interactie tussen de gebruiker en het data model evenals de data opslag vermeden. In de tussenliggende lagen kan vervolgens de benodigde validatie en logica uitgevoerd worden om de beveiliging van het systeem te garanderen.

**3.1 Front-end**

De front-end van de applicatie is voornamelijk geschreven in Javascript en Jquery. Met uitzondering van de admin pagina welke ook gebruik maakt van een standalone React\* element (adminPage.jsp r. 45, App.js) Elke pagina van de applicatie heeft een eigen .js file en een eigen .css file. Daarnaast maken de pagina’s gebruik van verschillende .js en .css files om gebruik te maken van gedeelde functionaliteiten.

De gebruiker interacteert met de omgeving overwegend via “on-click events”. Deze events worden aan diverse elementen toegewezen wanneer de desbetreffende pagina geladen wordt (listController.js r. 232) of wanneer bepaalde elementen dynamisch worden toegevoegd (homePageController.js r. 209). In de executie van deze events wordt vervolgens via een “getJSON()” functie (loginController.js r. 135) een http verbinding gemaakt met de desbetreffende controller in de presentatie laag van de applicatie (UserController.java r. 24). In de meeste gevallen zullen bij een dergelijke uitvoering parameters meegegeven worden, maar dat is niet noodzakelijk altijd het geval. Zo wordt in bepaalde situaties bijvoorbeeld enkel een validatie verricht in de backend welke een Boolean retourneert (loginController.js r. 39). Anderzijds kan er in enkele gevallen ook direct gecommuniceerd worden met een controller zoals in het geval van een redirect (loginController.js r. 151, 153). En binnen deze gevallen kan een validatie plaatsvinden in de controller zelf (PageController.java r. 34). In dit voorbeeld wordt de “role” van een gebruiker bevestigd voordat een redirect wordt uitgevoerd.

De front-end communiceert dus enkel met de presentatie laag. De data die doorgegeven wordt aan de front-end kan op verschillende manieren zichtbaar worden gemaakt aan de gebruiker.

Zo wordt bijvoorbeeld op de home page verschillende aanbevelingen van films getoond. Deze “thumbs” worden dynamisch aangemaakt dmv. het klonen van een template element in de html (homePage.jsp r. 45) en deze kloon wordt vervolgens voorzien van de relevante data via het instellen van het “src” attribuut van het betreffende html element (homePageController.js r. 149).

\* Het gebruik van React in de applicatie is zeer beperkt. Reden hiervoor is dat de front-end van de applicatie vrijwel voltooid was bij aanvang van de betreffende module.

**3.2 Backend**

De backend is onderverdeeld in 3 lagen. Namelijk de presentatie laag, de business laag en de data laag.

De presentatie laag bevat enkel de “controller” classes welke de communicatie met de front-end / UI faciliteren. Deze controllers roepen vervolgens methodes aan in de business laag waar de logica wordt uitgevoerd (UserController.java r. 25). In dit voorbeeld wordt in de class “Userbusiness” de methode “login” aangeroepen waarbij 2 variabelen worden meegegeven (gebruikersnaam en paswoord). Deze class maakt onderdeel uit van de business laag. Vanuit deze class wordt vervolgens weer een andere class aangeroepen uit de data laag, ook wel “model” genoemd (UserBusiness.java r. 139). In dit voorbeeld wordt een ArrayList gecreëerd met alle gebruikers en vervolgens wordt er geïtereerd over deze ArrayList om de opgegeven gebruikersnaam te vergelijken met de namen in de ArrayList. De logica vindt dus plaats in de business laag en het verwerken van de data gebeurd met een kopie van de data en niet met de dataset in de database.

In deze methode is ook te zien hoe een specifieke boodschap terug kan worden gecommuniceerd aan de gebruiker afhankelijk van de business logica (UserBisiness.java r. 154). In dit voorbeeld wordt er aan de gebruiker gecommuniceerd dat het registratie proces van de desbetreffende account nog niet is voltooid. Het bericht wordt met de aanduiding “msg” toegevoegd aan een JSON object en de string waarde van dit object wordt terug gekoppeld aan de methode in de controller via de “return” methode. Vervolgens kan in de front-end het bericht uitgelezen worden door de waarde van het attribuut “msg” uit te lezen van de response van de getJSON() functie (zoals beschreven in de vorige sub-paragraaf). Via deze techniek kan de logica volledig plaatsvinden binnen de business laag en wordt de code in de front-end geminimaliseerd omdat hier een generieke actie kan worden uitgevoerd (loginController.js r. 161).

Vanuit de business laag wordt ook gecommuniceerd met eventuele externe services. In deze applicatie wordt met de service van TMDB gecommuniceerd om zoekopdrachten naar films uit te voeren en de relevante data van deze films te bemachtigen. In de class SearchBusiness bevindt zich een methode genaamd “search” (SearchBusiness.java r. 43). In deze methode wordt gebruik gemaakt van een HTTP verbinding met de service van TMDB om een API aan te roepen. In de class “EmailBusiness” bevindt zich een methode genaamd “sendEmail” (EmailBuusiness.java r. 59). In deze methode wordt gebruik gemaakt van de library “javax.mail” om een email te versturen via het “Simple Mail Transfer” protocol waarbij gebruik gemaakt wordt van de Google mail server.

De data laag communiceert met de database en via de classes en methodes in deze laag verlopen alle read / write operaties. In de class “MovieSearchQueries” bevindt zich een methode genaamd “insertSearch” (MovieSearchQueries.java r. 26). In deze methode wordt gebruik gemaakt van meerdere “prepared statement” om SQL statement’s uit te voeren naar de database en data te schrijven naar meerdere tabellen binnen een enkele verbinding. Voordelen van deze techniek zijn dat de dataoverdracht zeer snel verloopt omdat enkel de variabelen in de prepared statement verzonden hoeven te worden en dat deze techniek zeer goed is in het verhinderen van SQL injecties. Om een prepared statement te kunnen uitvoeren zal in sommige gevallen de desbetreffende data omgezet moeten worden naar een ander type. Dit kan voorkomen in gevallen waar de gebruikte data types van de database niet goed aansluiten bij die van de codeer taal.