# Planning

|  |  |  |
| --- | --- | --- |
| Week | Date | Werkzaamheden |
| Week 1 | 11/02 – 17/02 | 1. Connectie Payara en MySQL 2. Opzetten domeinmodel en ERD met Visual Paradigm 3. Project initialiseren met Netbeans 4. Modellen opzetten in project 5. Project koppelen met Github |
| Week 2 | 18/02 – 24/02 | 1. Modellen associaties testen 2. Herhalen en implementeren JPA 3. Database access layer testen |
| Week 3 | 25/02 – 01/03 | 1. Gebruik CDI/EJB – 2. Mockito tests gebruiken 3. Rest API opzetten en aanroepen via Postman 4. Rest laag testen en excepties toevoegen |
| Vakantie | 04/03 - 10/03 | 1. Cucumber integratietests |

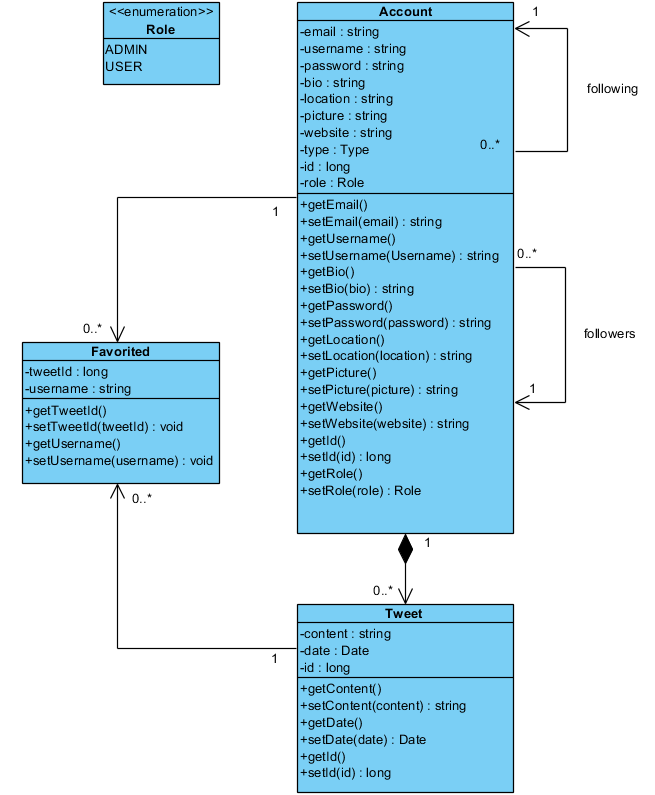
# Planning uit canvas course:

|  |  |
| --- | --- |
| 1. Model | Ontwerp je domeinmodel waarbij de entiteiten op een correcte manier met elkaar geassocieerd zijn.  Creëer unit testen met een representatieve set aan test objecten voor alle functionele scenario’s (10 gebruikers met onderlinge vriendschappen en 10 kweets).  Test de associaties na het aanmaken van de objecten |
| 1. Collection Data Access Layer | Verplaats de objecten uit de unit testen naar een Data Access Layer die via collections de gegevens “in memory” beheert.  Voorzie unit testen voor je Data Access Layer. |
| 1. Service Layer + Collection Data Access Layer | Ontwikkel de gepaste service layers. Maak voorlopig enkel gebruik van de Collection Data Access Layer. Realiseer de methodes voor alle functionele scenario’s. Voer controles uit die ongewenst gebruik met exceptions signaleren.  Maak een unit test voor de service layer en voorzie een integratietest met de Data Access Layer. Test de functionele scenario’s via de service layer. |
| 1. Gebruik CDI/EJB | Bepaal per aangeroepen constructor of je dit door de server kunt laten beheren. Bepaal per klasse de lifecycle van de bean.  Pas je Service Layer en Rest Layer testen aan met gebruik van Mockito zodat de testen zelf weten welke objecten te moeten injecteren. |
| 1. Rest Layer | Bouw een REST api waarbij de methodes van de verschillende service klassen aangesproken kunnen worden, en waarbij het model als datatransfer object gebruikt wordt (en niet bv. String)  Geef enkel JSON terug als resultaat en handel Exceptions bij ongewenst gebruik adequaat af.  Deploy je web applicatie in je container en controleer je REST layer via je Postman of Swagger.  Schrijf een REST test die met een HTTPClient de belangrijkste functionaliteit van je Rest api test. |
| 1. Database Access Layer | Zorg ervoor dat je code kan kiezen welke data access layer implementatie hij nodig heeft  Breng JPA annotaties aan waar nodig en schrijf de named queries voor bv opzoekingen.  Zorg ervoor dat je test objecten ook effectief opgeslagen worden in de database.  Voer dezelfde testen uit als in stap 2/3 maar dan met gebruik van de database access layer. |

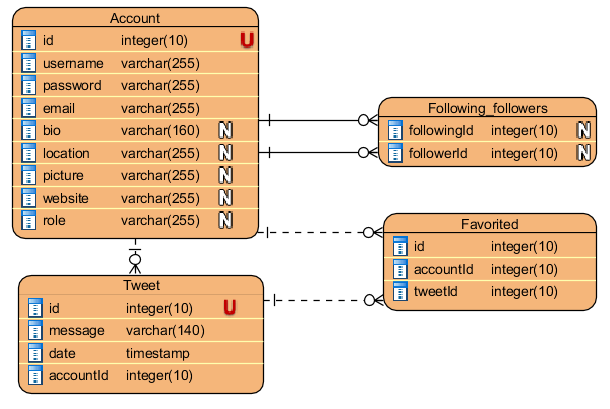
# Domain model

# 

# Class Diagram



# Entity Relationship Diagram



# User stories

**Gebruiker**

Als een gebruiker wil ik een gebruikersnaam kunnen wijzigen wat voor andere gebruikers zichtbaar is.

Als gebruiker wil ik mijn bio, locatie en website kunnen invullen zodat ik kort kan omschrijven wie ik ben.

**Tijdlijn**

Als gebruiker wil ik andere gebruikers kunnen volgen om zo hun berichten op mijn dashboard te ontvangen.

Als gebruiker wil ik een lijst kunnen opvragen van alle gebruikers die ik volg zodat ik snel een overzicht heb van mensen die ik gevolgd heb.

Als gebruiker wil ik andere gebruikers hun activiteit kunnen zien zodat ik snel een overzicht krijg wat iedereen post.

Als gebruiker wil ik berichten kunnen toevoegen aan favorieten zodat ik deze later overzichtelijk terug kan opvragen.

Als gebruiker wil ik kunnen zoeken naar berichten om een overzichtelijk lijst van zoekresultaten te vinden.

Als gebruiker wil ik een tweet kunnen posten zodat andere gebruikers deze kunnen lezen.

**Register/Login**

Als gebruiker wil ik kunnen uitloggen om mijn sessie te beëindigen.

Als gebruiker wil ik kunnen inloggen om een nieuwe sessie te starten.

**Beheerder**

Als beheerder wil ik extra rechten hebben om administratieve functies uit te voeren.

Als beheerder wil ik berichten van gebruikers kunnen verwijderen om onheus taalgebruik tegen te gaan.

Als beheerder wil ik een overzicht van gebruikers kunnen opvragen om rollen en accountgegevens op te vragen.

# Non-Functionele Requirements

**Programma**

Een bio van een gebruiker mag maximaal 160 karakters lang zijn.

Het zoeken van tweets gebeurd met AJAX.

Een tweet is maximaal 140 karakters lang.

*Optioneel:*er kan gekozen worden tussen Engels en Nederlands.

**Code**

Volg SOLID-principes.

Implementeer automatische tests.

Gebruik Javadoc waar van toepassing.

Database met behulp van van JPA, MySQL8 en Payara.

Gebruik EJB/CDI die onderhouden worden via Mockito.

# Automated tests

In dit hoofdstuk worden de testen met vaktaal beargumenteerd en toegelicht.

De automatische tests worden opgezet aan de hand van verschillende frameworks:

* Geen naming convention maar voor mezelf logisch opgezet
* Opgezet aan de hand van AAA (Arrange, Act, Assert) structuur
* Zo veel mogelijk getest wat er fout zou kunnen gaan



Figuur 1: AAA-patroon

Er is op verschillende lagen getest zodat. Hieronder een overzicht:



Figuur : Overzicht testlagen

In een later stadium zou selenium gebruikt kunnen worden om het front-end te testen.

**Model Layer - Domain tests**

De modellen testen van Tweet en Account. Er is gefocust op wat er fout kan gaan en hoe het systeem hiermee opgaat.

**Rest Layer – API tests**

Met Jersey eenvoudige calls doen om te kijken of http-responses goed zijn. Ook via postman verschillende api calls handmatig getest.

**Service Layer - Service tests**

Getest met Mockito. Maakt gebruik van startup bean.

Stappenplan Mockito

1.Update pom.xml (maven)

2.Service klasse opstellen (zie voorbeeldproject canvas)

3.Startup bean maken

4.Service test schrijven (met @injectmocks de juiste service testen)

Error: No database selected = payara fout (localhost 4848) url -> add db name (van mysql workbench) -> url=localhost:3306/kwetter (iets in die aard)

**Data Access Layer – DAO tests**

Een mock-database lokaal opgezet (kwetter\_test) en de dao functies verzonden naar deze database. Maakt gebruik van een tweede persistence file.

**Integratietests**

Met Cucumber en Gherkin enkele eenvoudige scenarios uitgeschreven en automatisch laten testen. API calls verifiëren met behulp van restassured.