In deze opdracht ga je een kwetter backend op basis van Java Enterprise Edition (JavaEE)-technologie in een applicatie container realiseren, zoals Payara of Wildfly. De software architectuur die je hiervoor gebruikt bouwt voort op de gelaagde architectuur uit de WebLog applicatie (service, dao etc.). De User Stories(US) voor de backend zijn gebaseerd op de Kwetter Casus. Met een uitvoerige set testen wordt de kwaliteit van de backend gewaarborgd.

[Analyse en Ontwerp 2](#_Toc502186456)

[Realisatie stappenplan 2](#_Toc502186457)

[Beoordeling 3](#_Toc502186458)

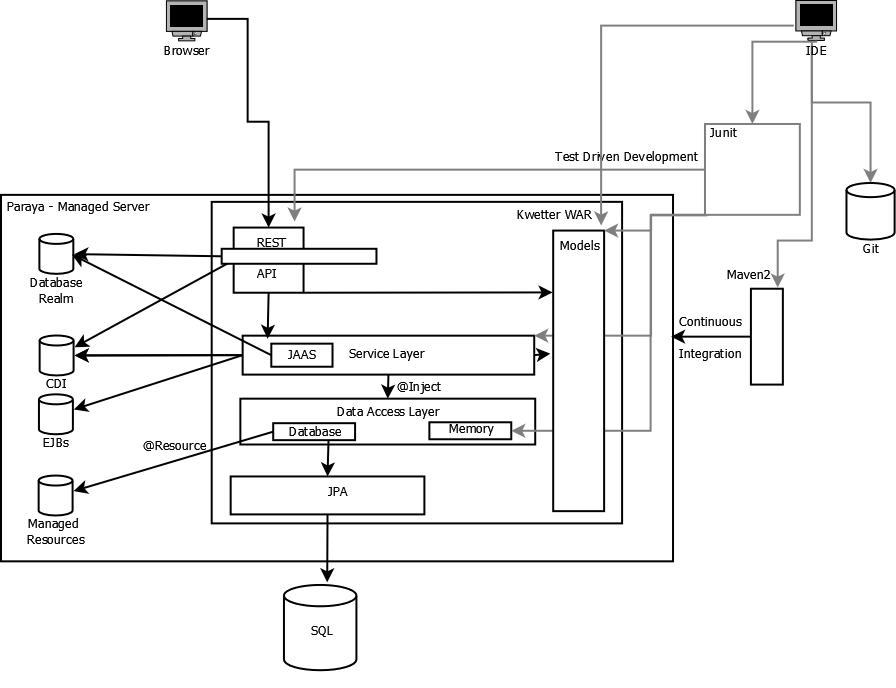
[Literatuurlijst 3](#_Toc502186459)

[Optionele opgaven 4](#_Toc502186460)

[CDI events & Interceptors 4](#_Toc502186461)

[Batch Processing 4](#_Toc502186462)

[Continuous Integration: set-up with build server en docker 5](#_Toc502186463)



Figuur 1 Te realiseren componenten in de backend

# Analyse en Ontwerp

Deliverables voor het ontwerp zijn User Stories, een Domein Model en een specificatie van de service laag. De specificatie van de service laag wordt in JavaDoc beschreven.

Bespreek je ontwerp met de docent voor je in je implementatie duikt.

# Realisatie stappenplan

Deliverable voor de realisatie is een Java EE-applicatie die gedeployed in een JavaEE-container (Payara/Wildfly) runt. Voor het bouwen van de applicatie wordt gebruik gemaakt van JavaEE-technologieën zoals CDI, EJB’s, Events etc. om een “loosely coupled architecture” te realiseren.

Bouw je applicatie geleidelijk op volgens het onderstaande stappenplan. Voorzie telkens automatische testen voor je naar de volgende stap gaat. Voorzie je commits met tags per stap.

|  |  |
| --- | --- |
| 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. |

# Beoordeling

De kwaliteit van de programmacode wordt meegenomen in de beoordeling.

Het voldoende niveau wordt per onderwerp als volgt bepaald:

1. Het opzetten van een lokale ontwikkelomgeving met versiebeheer, geconfigureerde build tool en automatische (scenario) testen
   * Je kunt wijzigingen aan je code documenteren. Je hebt minstens per realisatie een tag voorzien die apart uit te voeren is. Je project dependencies zijn gedocumenteerd en worden automatisch toegevoegd. Je testen kunnen automatisch uitgevoerd worden bij elke build.
2. Het documenteren van een analyse en ontwerp (userstories, klassendiagram) voor een backend applicatie
   * Je kunt je beslissingen beargumenteren en je ontwerp is een goede weerspiegeling van je realisatie.
3. Ontwerp voldoet aan SOLID principes en waar mogelijk zijn design patterns toegepast
   * Je kunt je code verdedigen op vlak van uitbreidbaarheid en onderhoudbaarheid.
4. Opzetten van een lokale enterprise web server die resources en beans kan managen
5. Gebruik maken van de mogelijkheden van de enterprise web server voor object management en resource management
   * Je gebruikt geen SE oplossingen waar EE mogelijk is, en je hebt een bewuste keuze gemaakt tussen CDI en EJB annotaties.
6. Test driven development toepassen op minstens 3 verschillende niveaus: model, service, service + persistence. Container managed testen is optioneel.
   * Je hebt voldoende test coverage voor moeilijkere scenario’s, waarbij je zowel unit als dependency testing gerealiseerd hebt. Je hebt minstens 1 andere integratie test mogelijkheid uitgewerkt voor een extra laag.
7. Web applicatie toegankelijk maken voor HTTP via REST (niveau 1)
   * Je applicatie is zonder GUI maar met gebruik van HTTP REST calls volledig bruikbaar. Je gebruikt de mogelijkheden van JAX-RS voor automatische omzettingen van model naar JSON en afhandelen van exceptions
8. Je gegevens kunnen zowel zonder als in een database bewaard worden. Voor de database maak je gebruik van JPA technieken.
   * Je hebt een werkende applicatie wiens gegevens bij wijzigingen terug te vinden zijn in de database. Je code werkt met JPA entities. Liefst heb je named queries gebruikt.

# Literatuurlijst

Zie de opdrachtomschrijving van Backend-end in Canvas om de hyperlinks terug te vinden naar de bijkomende informatie.

Bestudeer eerst **Test-driven development (TDD),** **Java EE**, **Contexts and Dependency Injection**, **Enterprise JavaBean**, **REST API (simple) & Java Persistence Api**.

# Optionele opgaven

Bekijk de specifieke opdracht pagina’s in Canvas voor meer informatie over deze uitbreidingen.

## CDI events & Interceptors

* **CDI events**

Herontwerp je bestaande Kwetter project zodat de volgende gebeurtenissen door middel van het produceren en consumeren van CDI events worden afgehandeld. Productie en consumptie van een event dient steeds in verschillende componenten plaats te vinden:

1. De server ontvangt een tweet van een gebruiker: hiertoe dient een Tweet-event te worden gepubliceerd. Dit event moet onder andere geconsumeerd worden door de component die de tweet opslaat.
2. Een gebruiker meld zich aan bij je web-applicatie: hiertoe dient een User-event gepubliceerd te worden. Dit event moet geconsumeerd worden door een component die deze gebeurtenis naar het server-log wegschrijft. Het inloggen zelf mag nog steeds gesimuleerd worden.
3. Zoals de vorige deel-opgave, maar dan voor het afmelden van een gebruiker (bijvoorbeeld doordat zijn sessie beëindigd wordt)
4. Een kwetteraar begint een andere kwetteraar te volgen.
5. Een kwetteraar stopt het volgen van een andere kwetteraar.

* **Interceptors**

Vet en cool zijn uit, hard en dik zijn in**.**

Om nieuwe gebruikers van kwetter te helpen om modern en trendy over te komen ga je bij nieuwe tweets de woorden vet of cool automatisch vervangen door dik respectievelijk hard.

Dit doe je door hiervoor een speciale klasse VolgTrendInterceptor te schrijven. Deze klasse neemt de vertaling voor zijn rekening. Zorg er ook voor dat deze interceptor bij de juiste methode aangeroepen (invoked) wordt.

## Batch Processing

Om de kwetteromgeving realistischer te kunnen gebruiken en testen gaan we een grote hoeveelheid tweets importeren. Het verwerken van grote hoeveelheden data faciliteert Java EE met het zogenaamde batch processing.

Ontwikkel nu een programma op basis van de JEE batch processing architectuur om grote hoeveelheden tweets in te lezen. Op Canvas vind je het input bestand JEA kwetter input.zip. Het bestand bestaat uit een collectie tweets in JSON-formaat. Verwerk dit bestand van JSON Objecten om je kwetter applicatie initieel van dat te voorzien.

Een interessantere (schaalbaarheids-) casus is de gegevensverzameling van 50GIG compressed general Tweets die je kunt downloaden vanaf <https://archive.org/details/archiveteam-twitter-stream-2013-08>. Binnen de pagina <https://archive.org/details/twitterstream> zijn nog andere collecties te vinden.

## TDD: integratietesten

Voor automatisch testen van je Java EE applicatie kun je gebruik maken van test-tools zoals JUnit, EasyMock, Cucumber, Arquillian en RestAssurred icm EasyMock. De backend moet met integratie testen worden uitgebreid, zodat op een (embedded) applicatie server getest kan worden.

[https://ordina-jtech.github.io/integration-testing/restassured/ (Links to an external site.)](https://ordina-jtech.github.io/integration-testing/restassured/)

## NoSQL: MongoDB

Je bedenkt en implementeert een Use Case in de Kwetter casus waarmee je de technologie MongoDB kun toepassen.

## Continuous Integration: set-up with build server en docker

Breid je ontwikkelomgeving uit zodat je code automatisch gedeployed wordt via een build server naar een docker virtualisatielaag.