In deze opdracht ga je een kwetter backend op basis van EE-technologie en in een enterprise webcontainer realiseren. De software architectuur die je hiervoor gebruikt bouwt voort op de gelaagde architectuur uit de WebLog applicatie (service, dao etc.). De userstories voor de backend zijn gebaseerd op de [Kwetter Casus](https://fhict.instructure.com/courses/2930/pages/kwetter-casus). Met een uitvoerige set testen wordt de kwaliteit van de backend gewaarborgd.

[Analyse en Ontwerp 1](#_Toc473286800)

[Realisatie 1](#_Toc473286801)

[Beoordeling 2](#_Toc473286802)

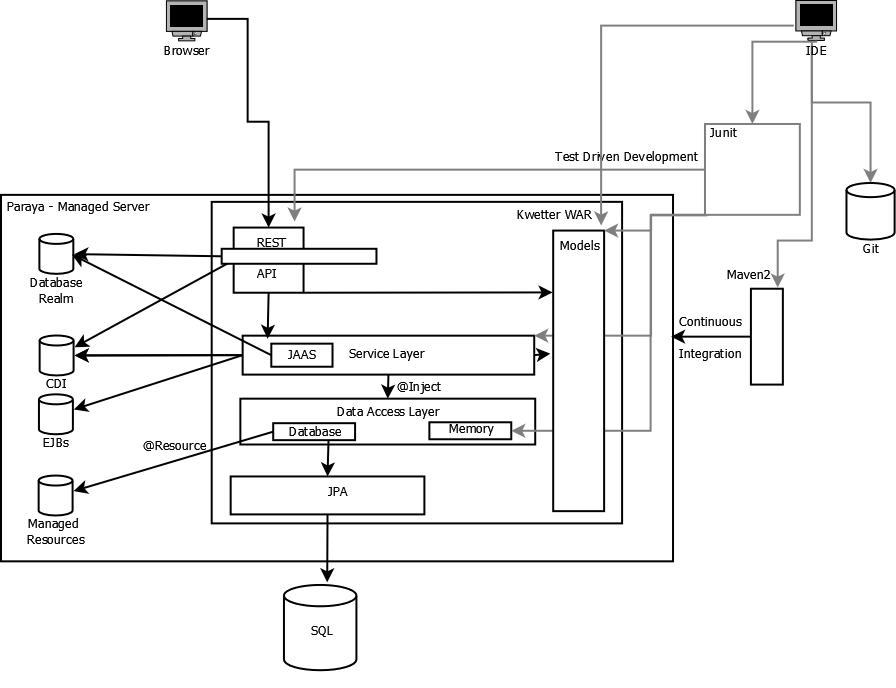
[Literatuurlijst 3](#_Toc473286803)

[Optionele opgaven 3](#_Toc473286804)

[CDI events 3](#_Toc473286805)

[Interceptors 4](#_Toc473286806)

[Batch 4](#_Toc473286807)



Figuur 1 Te realiseren componenten in de backend

# Analyse en Ontwerp

Deliverables voor het ontwerp zijn userstories, een klassendiagram 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 EE-container (Payara) runt. Voor het bouwen van de applicatie wordt gebruik gemaakt van EE-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 | Implementeer 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 | Voorzie de gepaste service layers. Maak voorlopig enkel gebruik van de Collection Data Access Layer.  Voorzie 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. Rest Layer | Voorzie 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 browser of Postman.  Schrijf een REST test die met een HTTPClient de belangrijkste functionaliteit van je Rest api test. |
| 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. Authenticatie en autorisatie | Begin met het configureren van een file realm in je applicatie server  Test of de authenticatie met behulp van BASIC authentication werkt (form mag ook, maar hergebruik waar gepast!)  Definieer de rollen en voeg een testgebruiker toe per rol  Test of de authenticatie voor elke gebruiker de juiste rol geeft  Voeg autorisatie toe aan de web.xml op basis van rol  Test dat sommige URL’s niet meer toegankelijk zijn voor gewone gebruikers  Wijzig je REST layer zodat sommige scenario’s enkel uitgevoerd kunnen worden door een bepaalde rol  Pas je Rest Layer testen aan zodat je authenticatie en autorisatie meeneemt |
| 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 opzoekingen.  Zorg ervoor dat je test objecten ook effectief opgeslagen worden in de database.  Voer dezelfde testen uit als in stap 3 maar dan met gebruik van de database access layer. |
| 1. Database realm | Configureer een managed resource die een database gebruikt als realm.  Pas je project configuratie aan zodat hij gebruik maakt van je nieuwe resource.  Voer je testen opnieuw uit. |

# 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. Web autorisatie laten afhandelen door enterprise web server met gebruik van database realm
   * Je gebruikers worden in je database beheert en kunnen inloggen en slechts gebruik maken van de functionaliteiten waartoe hun rol toegang heeft.
9. Web authenticatie realiseren op service niveau door gebruikt te maken van de JAAS annotaties
   * Je gebruikt het authenticatie mechanisme en de autorisatie mogelijkheden van JAAS.
10. 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

Bestudeer eerst [Test-driven development (TDD)](https://fhict.instructure.com/courses/2930/pages/test-driven-development-tdd), [Java EE](https://fhict.instructure.com/courses/2930/pages/java-ee), [Contexts and Dependency Injection](https://fhict.instructure.com/courses/2930/pages/contexts-and-dependency-injection), [Enterprise JavaBean](https://fhict.instructure.com/courses/2930/pages/enterprise-javabean), [REST API (simple)](https://fhict.instructure.com/courses/2930/pages/rest-api-simple), [Security](https://fhict.instructure.com/courses/2930/pages/security) & [Java Persistence Api.](https://fhict.instructure.com/courses/2930/pages/java-persistence-api)

# Optionele opgaven

## 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.

1. Zoals de vorige deel-opgave, maar dan voor het afmelden van een gebruiker (bijvoorbeeld doordat zijn sessie beëindigd wordt)
2. Een kwetteraar begint een andere kwetteraar te volgen.
3. 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

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 SharePoint vind je het input bestand JEA kwetter input.zip. Het bestand bestaat uit een collectie tweets in JSON formaat. In bovengenoemd tutorial vind je aanwijzingen voor de verwerking van JSON objecten.

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.

## 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.

## Java Mail en automatische taken

Het kunnen versturen van e-mail berichten behoort tot één van de standaardtaken van veel web-servers. De standaard API hiervoor binnen JEE is de Java Mail API.

Gebruikers moeten zich bij Kwetter kunnen registreren, en na activatie, inloggen. Een goede registratie-procedure verifieert email-adressen van nieuwe gebruikers om onzin registraties te kunnen verwijderen. Hiertoe wordt vaak een e-mail naar de nieuwe gebruiker toegestuurd waarin een link staat die de nieuwe gebruikers-account “activeert”. Dit ga je doen door JavaMail in te schakelen. Niet geactiveerde gebruikers-accounts moeten na een instelbare periode weer automatisch worden opgeruimd middels een EJB @Timeout. Het stappenplan is als volgt:

1. Configureer een JavaMail session in GlassFish, deze ga je gebruiken om emails te kunnen *sturen* vanuit de server
2. Maak een registratie-pagina aan voor zover je die nog niet had, inclusief de achterliggende server-side beans. De gebruiker moet hierin een e-mail adres opgeven (tip: vereis dat de userName een email-adres is). Let er op dat wachtwoorden door GlassFish gehashed worden opgeslagen (standaard: SHA-256 ). Hoe je in java een String representatie van een SHA-256 hash van een input-String krijgt vind je makkelijk op bijvoorbeeld [Stack Overflow](http://stackoverflow.com/questions/5531455/how-to-encode-some-string-with-sha256-in-java).
3. In je server-side bean(s) stuur je na registratie een email naar de nieuwe gebruiker met daarin een unieke activatie-link waarop slechts 1 keer geklikt kan worden. De afhandeling op de server “activeert” de gebruiker. Voor activatie mag een gebruiker niet succesvol kunnen inloggen: bedenk tijdig hoe je dit kunt voorkomen en wat er allemaal aan je domein klassen moet worden toegevoegd.
4. Maak een EJB @Timout-methode die automatisch niet-geactiveerde gebruikersaccounts uit het systeem verwijdert na een instelbare periode. De timer moet op elk uur van de dag worden afgevuurd.