Spotitube Opleverdocument

—-

VERSIE	**1**
DATUM	**05 - 01 - 2022**
COURSE	**OOSE - DEA**

STUDENT
Thomas Beumer
585380

DOCENT
Meron Brouwer

Inhoudsopgave

Inleiding

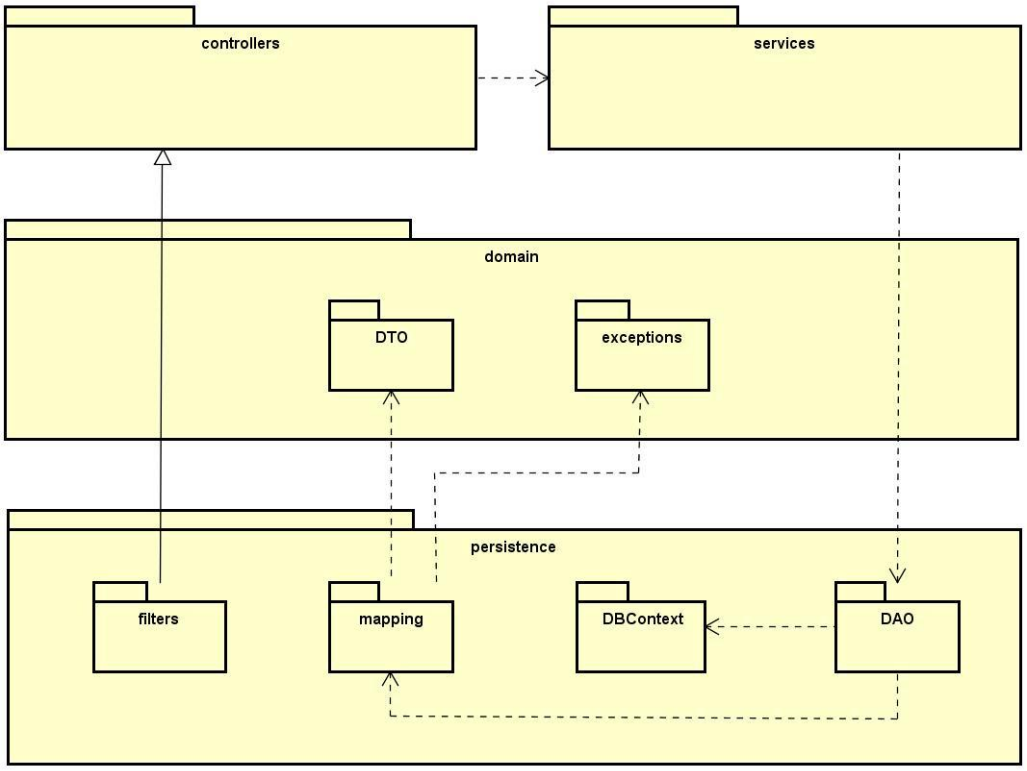
Dit document beschrijft ontwerp keuzes voor de opgeleverde back-end applicatie, “Spotitube”. Voor de opdracht is het back-end ontwikkeld op een JEE-ontwikkelomgeving. De client maakt HTTP requests naar het back-end voor data via restful endpoints die zijn opgezet met JAX-RS.

Eerst wordt er ingegaan op een package diagram, gevolgd door een deployment diagram. De package diagram toont alle systeem afhankelijkheden. De deployment diagram toont de implementatie van het systeem op hardware niveau.

Tot slot licht ik de gemaakte ontwerp keuzes en bijzonderheden tijdens het ontwikkelen toe. Ook kom ik terug op de kwaliteit van het eindproduct in de conclusie.

1. Package diagram

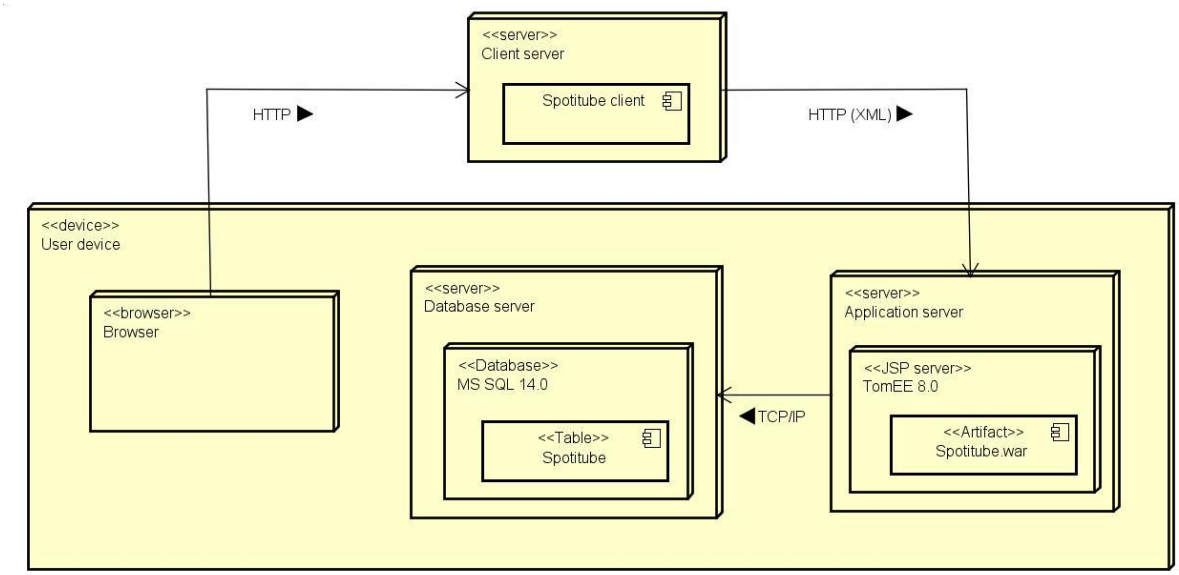
De package diagram maakt zowel de cohesie als koppeling van losse systeemcomponenten (packages) inzichtelijk. Het diagram toont daarmee alle verantwoordelijkheden en afhankelijkheden van het systeem op hoog niveau.



Afbeelding 1 - Package diagram

2. Deployment diagram

De deployment diagram toont op welke omgeving de applicatie draait en via welk protocol gecommuniceerd wordt. Omdat het back-end geheel op localhost draait, zie je in de diagram dat de database en applicatie server onder de “User device” node valt. In het geval dat de applicatie op productie wordt gedeployed, scheidt de node zich. De browser en server(s) staan dan gehost op losse devices.



Afbeelding 2 - Deployment diagram

De gebruiker vraagt via een HTTP request in de browser de client op. De client communiceert restful via XML HTTP request met de applicatie server (TomEE 8.0) die via TCP/IP verbind met de database (MSSQL 14.0).

3. Ontwerp keuzes

Lagen & injection
De applicatie is opgebouwd uit 4 lagen om aan het separation of concerns principe te voldoen. De lagen bestaan uit; “controllers”, “services”, “persistence” en “domain”. Elke laag heeft zijn eigen verantwoordelijkheden. De lagen bevatten zo min mogelijk afhankelijkheden om vrijwel autonoom te functioneren en abstractie te verhogen. Tevens wordt voor die reden CDI depency injection toegepast.

Controllers
“controllers” is een laag uitsluitend bedoeld voor het opzetten van endpoints, het aanroepen van services en het retourneren van response objecten.

Services
De “services” laag staat tussen de controllers en DAO’s in. De laag kan business logica bevatten en is verantwoordelijk voor het aanroepen van een of meerdere DAO’s.

Domain
In “domain” worden data structuren getypeerd in de vorm van DTO’s, interfaces en exceptions. Het is een “primitieve” laag omdat het geen (uitgaande) afhankelijkheden of logica bevat.

Persistence
In de “persistence” laag wordt data geconfigureerd en opgeslagen, er wordt invulling gegeven aan datastructuren. Deze laag staat bekend als data-access-layer en vormt een sessie met een database.

DTO
Een Data Transfer Object (DTO) is verantwoordelijk voor het omzetten van database resultaten naar standaard objecten. Een database result is een object die een sessie met een tabel representeert, ga je die sessie doorsturen dan kun je ongewenste database bewerkingen uitvoeren.

Seperate interfaces
Door seperate interfaces te gebruiken voor de DAO’s, services en mappers worden afhankelijkheden verlaagd waardoor de kwaliteit omhoog gaat. De client (back-end) hoeft namelijk niks te weten over de implementatie waardoor je de implementatie makkelijk kan vervangen/overschrijven met nieuwe code.

Filters
Voor de applicatie heb ik filters toegepast voor het SOLID principe en om data persistens te verhogen. Een filter kan HTTP request en responses ondervangen en business logica toevoegen. Zo wordt per request (m.u.v. het “/login” pad) het token geauthenticeerd. Als het token ongeldig is dan wordt het request geabord. Zoniet, dan wordt de URL herschreven met een user ID. Het response filter gebruik ik om CORS headers toe te voegen aan ieder request.

Mappings
Mappings zijn bedoeld om data om te zetten. Zo gebruik ik in de applicatie data mappers om result sets om te zetten naar DTO's en om exceptions te mappen naar een reponse (als die niet wordt afgevangen).

SQLExcepties
Een probleem waar in tegenaan liep tijdens het ontwikkelen, is de manier waarop een SQL exceptie wordt afgehandeld. JDBC gooit een SQL exceptie bij een leeg resultaat en/of als er een error optreedt door een fout in de query, bijvoorbeeld. Eigenlijk wil je dat niet, bij een fout in de query wil je 500 status code gooien en bij een leeg resultaat meestal een 400 (op z'n minst). Als alternatief bedacht ik om via stored procedures een status code aan de query resultaten toe te voegen. De status code uit het resultaat map je vervolgens naar een exceptie.

Polymorfisme
Om zo veel mogelijk duplicate code weg te werken gebruik ik polymorfie. De interfaces en implementatie van de DAO's' en services extenden abstracte base classes en interfaces die gemeenschappelijke code bevatten. Enkele abstracte classes en interfaces gebruiken generic types om om verschillende data types te ondersteunen

4. ToDo’s

Stored procedures
Querys staan als Sting vermeld in DAO’s. Indien nodig, dan prepare ik de query en voer ik het uit. Daarna wordt het resultaat naar een DTO gemapt. Het is netter om voor de querys, stored procedures te ontwikkelen. Dat is niet alleen beter voor performance, het draagt ook bij aan robuustheid. Momenteel kan een software developer het werk van een database specialist in de weg zitten. Door stored procedures te gebruiken kunnen de verantwoordelijkheden gescheiden blijven.

Builder pattern
Sinds deze course ben ik regelmatig in aanraking gekomen met builder patterns. Het is een ontwerppatroon die readability en robuustheid verhoogd. In de applicatie bouw ik querys aan de hand van de stappen in de alinea hierboven. Dit zijn losse stappen die chronologisch uitgevoerd moeten worden en voor langdradige, dubbele en slecht leesbare code zorgen. Met een builder patroon kunnen we al deze nadelen verhelpen doordat we methode chainen en via een commando (`build()`) de chain wordt geïnitialiseerd.

Hashen
De wachtwoorden in de database zijn niet gehashed. Dat vormt een groot veiligheidsrisico omdat iedereen met toegang tot de database de wachtwoorden kan inzien.

Filter afhankelijkheid
Filters staan nu onder de “persistence” laag. Toch zijn de filters specifiek gericht op de JAX-RS request en responses. Alleen de controllers hebben daar kennis van. De filters horen dus onder de controller laag te staan, om afhankelijkheid te verlagen.

Conclusie

Aan alle criteria die zijn opgesteld voor de opdracht is voldaan. Er is een minimale line-coverage van 80% behaald. De technieken beschreven in de casus zijn toegepast. De documentatie bevat alle vereiste onderdelen. Als kers op de taart, zijn alle functionele tests geslaagd. Daarmee ben ik van mening dat opdracht succesvol is afgerond. Toch is er altijd ruimte voor verbetering, in het hoofdstuk ToDo's benoem ik een aantal punten.