Soulful cursus platform

Naam: Michelle Royer
Datum inlevermoment: 27-2-2022
Onderwijsinstelling: NOVI Hogeschool
Naam Opleiding: Bachelor Full-stack developer
Module: Backend programmeren

Voorwoord

Dit document is gemaakt voor mijn eindopdracht voor de richting backend van Hogeschool Novi Utrecht. In dit document vind je de uitleg en het ontwikkelingsplan voor de Java applicatie van Soulful Yoga Course. Tijdens het bouwen van dit project is er een complete code geschreven om de applicatie werkzaam te krijgen en daarbij een documentatie van hoe alles werkt en hoe alles geïnstalleerd moet worden.

Inhoudsopgave

1. Inleiding

Soulful With Me is een klein eenmansbedrijf(je) dat nu 5 jaar bestaat, opgericht door een dame genaamd Gytha Brooks. Bij Soulful With Me kunnen vrouwen terecht van jong tot oud voor diverse massages en yoga lessen. Nu dat het bedrijf steeds meer begint te groeien, wilt Gytha haar droom bedrijf uitbreiden en daarbij meer mogelijkheden aanbieden. Zij wilt namelijk niet alleen yoga lessen geven, maar ook mensen hierin opleiden. Ook mede zodat ze de focus kan houden op wat ze leuk vind en niet wordt overrompeld in klanten, maar dit dus ook kan verdelen over meerdere yoga docenten. Voor mijn examenopdracht ga ik een applicatie bouwen waar zij deze lessen gemakkelijk kan geven. Hier komt bij dat zij video's en theorie kan uploaden, meerdere mensen een account kunnen aanmaken en er bijgehouden kan worden hoever mensen de stof al hebben behandeld.

2. Opdrachtbeschrijving

De opdracht bestaat uit het volgende: Voor de eindopdracht gaat de student een applicatie bedenken waarvan je alleen de backend gaat programmeren. Je bedenkt wat de gebruiker zou moeten kunnen met jouw applicatie, documenteert dit in een technisch ontwerp en gaat hier vervolgens de backend code voor schrijven.

Het systeem heeft minimaal de volgende eigenschappen:

- Autorisatie en authenticatie
- Communicatie met een relationele database middels CRUD-opdrachten.
- Een rest endpoint die data uit verschillende tabellen combineert en terugkoppelt.
- RESTful webservice.
- Er moeten bestanden geüpload en gedownload kunnen worden.
- Jouw code wordt getest met behulp van unit-testen en @mocken.
- Jouw applicatie vult een database met relevante testdata, zodat de applicatie getest kan worden.

3. Het concept

Voordat ik kon beginnen aan mijn eindproject, moest ik eerst een idee ontwerp inleveren. Hierbij zag mijn idee ontwerp er als volgt uit:

Voor mijn eindopdracht wil ik het backend systeem maken voor de website van een vriendin. Zij is al enige tijd bezig met het geven van yogalessen en massages, nu wilt zij hiervoor ook cursussen gaan geven. Deze cursussen wilt zij geven via een aparte website waarbij:

- Mensen een account kunnen aanmaken (user data - Volledige naam, emailadres, telefoonnummer[eventueel], leeftijd);

- Kunnen inloggen [bescherming voor de gegevens van de klant] ;
- Zij een overzicht heeft van alle deelnemers [geen wachtwoorden/wel de mogelijkheid om een link toe te sturen mocht iemand wachtwoord vergeten zijn en die willen aanpassen];
- Ze per periode filmpjes kan uploaden waarbij zij de oefeningen uitbeeldt (of alles kan plaatsen, maar per periode kiest wat iemand te zien krijgt)[belangrijk dat er genoeg geheugen en byte moet worden ingesteld ook voor de zichtbare kwaliteit van de filmpjes];
- Per periode theoriestof kan uploaden;
- Klant kan zijn/haar gegevens wijzigen of abbo annuleren → dit wordt wel eerst gecommuniceerd naar de platform eigenaar en zij moet dit dan definitief maken;
- Online kunt chatten met haar deelnemers en de deelnemers onderling;
- Je kunt alleen alles inzien als je de cursus volgt;
- Ook de mogelijkheid om live lessen te volgen.

Uiteindelijk wil ik van deze webapplicatie een moeder applicatie maken waarbij meerdere coaches een account kunnen aanmaken en vervolgens zij daarin hun klanten en materiaal kunnen toevoegen.

Hierbij had ik mijn project ook nog in sprints uitgeschreven, om zo een duidelijk beeld te scheppen wat mijn applicatie precies moest kunnen. Hiervoor kun je de bijlage onderaan dit document vinden.

Zie bijlage A.

4. Analyse & ontwerp

In dit deel zullen er verschillende onderdelen en functies van het nieuw te ontwikkelen systeem worden beschreven met behulp van diverse UML modellen. De reden dat er voor UML modellen is gekozen, is omdat dit een universele basis is voor systeemontwikkeling. Daarbij is het ook makkelijk te begrijpen en kun je de lezer een duidelijke uitleg geven over de werking van de applicatie en de processen binnen de applicatie.

4.1. Bedrijfsprocessen Kaart

In dit onderdeel biedt ik een duidelijk inzicht in welke processen allemaal ondersteund worden door de applicatie.

4.2 Use-case diagram beschrijving

In de use-case staat beschreven wat de functie per user is. Hierdoor weet de lezer welke rollen er zijn en hoe deze rollen functioneren binnen de applicatie.

Zie bijlage B.

4.3. Klassendiagram

Hierin staat omschreven hoe de klassen binnen de applicatie functioneren en hoe zij zijn opgebouwd.

Zie bijlage C.

4.4. Sequentie diagram vanuit de admin

Hierin kun je vinden hoe de admin functioneert binnen de applicatie. De admin beschikt over alle functies die een klant ook heeft, maar kan hierbij ook nog gebruikers toevoegen, wijzigen en verwijderen. Verder kan de admin ook content uploaden en toewijzen aan de betreffende klanten.

Zie bijlage D.

4.5. Sequentie diagram vanuit de klant/student

In het sequence diagram voor klanten is het volledige pad te zien die een klant aflegt om de cursus te volgen.

Zie bijlage E.

5. De applicatie

De methode die deze applicatie zal leiden is door middel van een agile methode. De reden dat hiervoor gekozen is, is omdat dit de mogelijkheid biedt de app naarmate het gebruik aan te passen naar de wensen van de gebruiker. Om duidelijker in beeld te brengen wat de applicatie moet kunnen, heb ik een extra bijlage toegevoegd waarin de functionaliteiten in sprints worden vastgesteld. Dit is ook voor mijzelf als goede leidraad geweest bij het opbouwen van de applicatie.

5.1. Technische verantwoording

5.1.1. Unit Testing

Voor de daadwerkelijk lancering van je applicatie is het altijd belangrijk om je code te testen. Bij het testen van mijn code heb ik ervoor gekozen enkel de service en controller klassen te testen, omdat in de overige kasse geen code stond die getest moet worden en enkel alleen informatie bevat.

Voor het testen van de code moest er eerst een dependency worden toegevoegd:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

Vervolgens heb ik het gebruik gemaakt van de @SpringBootTest annotatie. Via deze annotatie kun je je code gaan testen. Het is hierbij belangrijk dat je de @Autowired annotaties in je daadwerkelijke code ook implementeert in je test klassen anders kan de code niet getest worden. Hiervoor gebruiken we de @Mockbean annotatie. Deze annotatie De MockBean vervangt elke bestaande klassen van hetzelfde type, zodat deze gebruikt kan worden in de test klas. Voor het gebruik van objecten in de test klassen heb ik gebruik gemaakt van de @Mock annotatie. Deze is in principe hetzelfde als @Mockbean, maar dan voor objecten in plaatsen van klassen. Hierna moet je bij elk stukje code dat je gaat testen ook de @Test annotatie gebruiken.
zie voorbeeld:

```
@Test
public void test Set Course() {
   //ARRANGE
   Long id = 1L;
   Course course = new Course();

Mockito.when(courseRepository.findById(id)).thenReturn(Optional.of
(course));
```

```
    //ACT
    Optional<Course> result = courseRepository.findById(id);

    //ASSERT
    Assertions.assertEquals(course, result.get());
}
```

In de code zie je de omschrijvingen: Arrange, Act en Assert ook wel AAA genoemd. Bij Arrange wordt de invoerwaarden van de test ingesteld. Bij Act hier wordt de code daadwerkelijk getest en als laatst Assert, hier worden de resultaten van de code geverifieerd.

5.2. De Rollen

De rollen binnen de applicatie zijn: De admin, users/klanten en ik de applicatie developer. Hieronder vind je een korte omschrijving van de rollen.

- Admin: De admin is verantwoordelijk voor het aanleveren/uploaden van de content (video's en theorie), waardoor de overige users dit kunnen inzien. Verder heeft hij/zij ook de autorisatie een user aan te passen of te verwijderen. Kortom, de admin beschikt over alle beschikbare functies bij het gebruik van de applicatie (hier wordt verdere backend ontwikkeling of technische aspecten niet bij berekent).
- User/klanten: Zij kunnen zich aanmelden//inloggen, waardoor ze vervolgens een account krijgen en de lessen die zijn aangeleverd door de admin kunnen inzien en volgen.
- Developer: Houdt de applicatie up-to-date en zorgt ervoor dat alles processen binnen de applicatie goed blijven verlopen.

5.3. Functionaliteiten in het kort

- Klanten moeten de mogelijkheid hebben om aankopen te doen om toegang te krijgen tot de cursus.
- Klanten moeten de mogelijkheid hebben om zich te registreren via de website en in te loggen.
- Klanten moeten de lessen online kunnen volgen en aanvinken welke lessen zij voldaan hebben.
- Klanten moeten via de website een aankoop kunnen doen om de applicatie te kunnen gebruiken.
- De admin moet video en theoretische informatie kunnen uploaden.
- De admin moet inzicht hebben in de gebruikers plus in hoeverre zij de cursus al voldaan hebben.
- De admin heeft volledige autorisatie.
- Alle data moet opgeslagen worden.
- Na 6 maanden hebben de klanten geen inzicht meer in de stof en zouden ze dus een nieuwe aankoop moeten doen.

5.4. Limitaties in de app

Omdat we de applicatie binnen een bepaalde tijd moesten bouwen heb ik niet alles kunnen toepassen, maar ben ik wel van plan dit later in de toekomst te doen. Op dit moment is er nog geen mogelijkheid om online bankieren te koppelen. Allereerst is dit, omdat we dit nog niet hebben geleerd en daarnaast kost dit veel meer tijd om de code hiervoor te schrijven.

Verder zou ik de applicatie nog iets gebruiksvriendelijker willen maken door ook een online communicatie centrum toe te voegen, zodat het gebruik van de applicatie ook als community wordt ervaren. Ook ben ik er nog niet aan toegekomen om de homepagina

5.5. Backlog

Voor de ontwikkeling van deze cursus webapplicatie heb ik alle functionaliteiten omschreven in sprints. Aan de hand hiervan zijn hieronder de meest essentiële functionaliteiten toegelicht:

Zie bijlage A.

6. Unit Testing

Voor de daadwerkelijk lancering van je applicatie is het altijd belangrijk om je code te testen. Bij het testen van mijn code heb ik ervoor gekozen enkel de service en controller klassen te testen, omdat in de overige kasse geen code stond die getest moet worden.

Voor het testen van de code moest er eerst een dependency worden toegevoegd:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

Vervolgens heb ik het gebruik gemaakt van de @SpringBootTest annotatie. Via deze annotatie kun je je code gaan testen. Het is hierbij belangrijk dat je de @Autowired annotaties in je daadwerkelijke code ook implementeert in je test klassen anders kan de code niet getest worden. Hiervoor gebruiken we de @Mockbean annotatie. Deze annotatie De mock vervangt elke bestaande klassen van hetzelfde type, zodat deze gebruikt kan worden in de test klas. Hierna moet je bij elk stukje code dat je gaat testen ook de @Test annotatie gebruiken.
zie voorbeeld:

```
@Test
public void test Set Course() {
```

```
    //ARRANGE
    Long id = 1L;
    Course course = new Course();

Mockito.when(courseRepository.findById(id)).thenReturn(Optional.of
(course));

    //ACT
    Optional<Course> result = courseRepository.findById(id);

    //ASSERT
    Assertions.assertEquals(course, result.get());
}
```

In de code zie je de omschrijvingen: Arrange, Act en Assert ook wel AAA genoemd. Bij Arrange wordt de invoerwaarden van de test ingesteld. Bij Act hier wordt de code daadwerkelijk getest en als laatst Assert, hier worden de resultaten van de code geverifieerd.

7. Installatie Handleiding

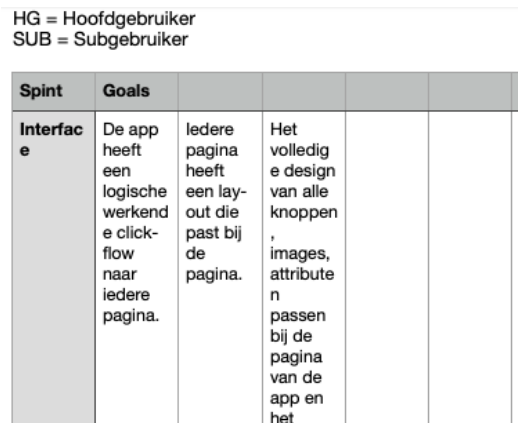
Als laatste gedeelte van dit project moest de installatie handleiding geschreven worden. Een installatie handleiding wordt geschreven voor de gebruiker van de applicatie, zodat hij/zij duidelijk weet allereerst hoe de applicatie werkt en daarnaast wat ervoor nodig is om de applicatie werkend te krijgen. Voor het schrijven van mijn handleiding heb ik gebruik gemaakt van de markdown functie. Ik heb hiervan gebruik gemaakt, omdat dit duidelijk leesbaar is en makkelijk toe te passen in mijn applicatie. Daarnaast wordt markdown ook aangeraden voor Readme Files voor github.

Ik heb ervoor gekozen deze handleiding in het engels te schrijven, zodat de toekomstige het gelezen en toegepast kan worden voor gebruikers van overal.

Bijlagen

Bijlage A

Deze bijlage is overigens nog niet helemaal af, omdat dit verder aangevuld moet worden wanneer ik begin aan de frontend van dit project.



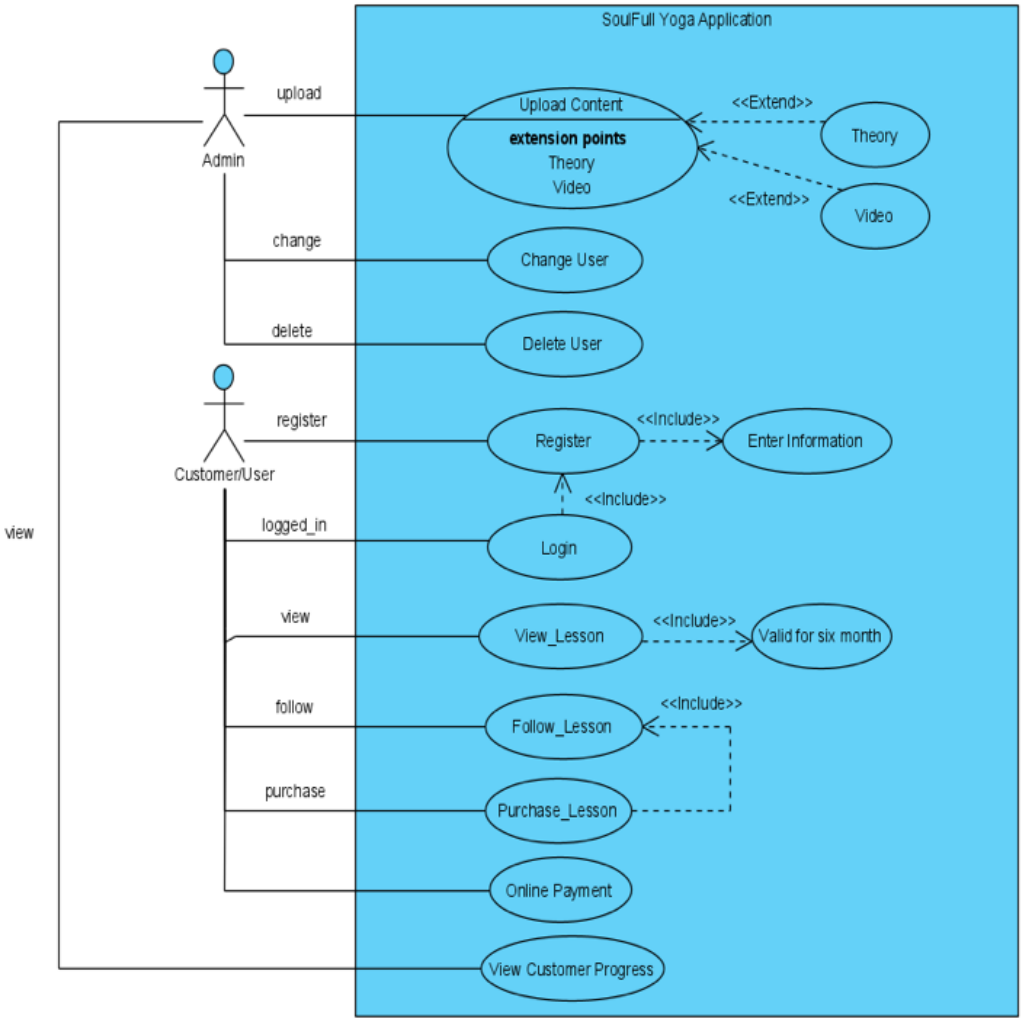
HG = Hoofdgebruiker
SUB = Subgebruiker

Spint	Goals				
Interface	De app heeft een logische werkende click-flow naar iedere pagina.	Iedere pagina heeft een lay-out die past bij de pagina.	Het volledige design van alle knoppen, images, attributen passen bij de pagina van de app en het		

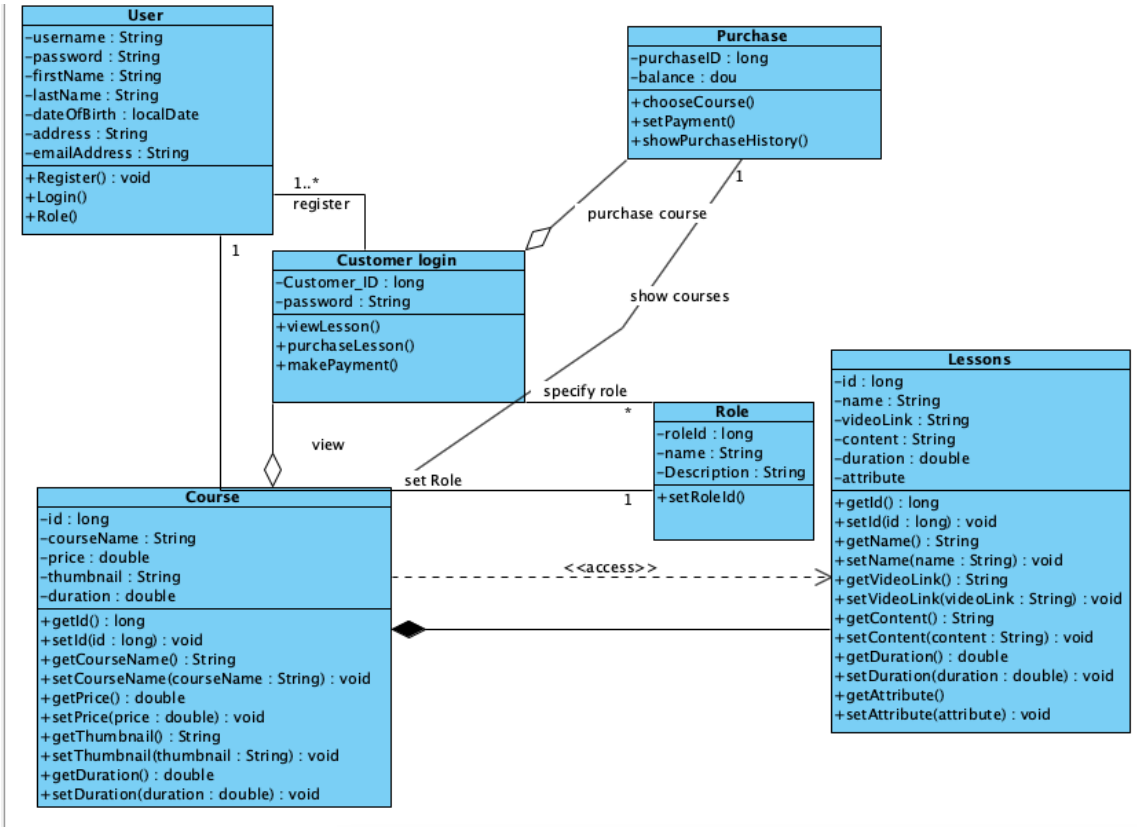
	Spint	Goals					
5	Log in	De app kan inlog informatie aflezen van de externe database.	De app kan veilig versleutelde wachtwoorden ontsleutelen en lezen.	De app kan onderscheid maken tussen de HG en de klanten en weergeeft een homepagina afhankelijk van deze rol.	De app kan automatisch inloggen mits de desbetreffende persoon hiervoor toestemming heeft gegeven en op basis van eerder gebruik.	De app heeft een "wachtwoord vergeten" functie inclusief controle vraag.	

1	Spint	Goals								
9	**Lesstof pagina**	De klant kan zien welke lessen zij kunnen doen en/of waar ze waren gebleven.								
10	**Upload pagina**									
11	**Profielen**	De app kan specifieke gegevens toevoegen aan- en aflezen van de database.	De klant kan zijn/haar vooruitgang hierin bijhouden van de lessen die zij al hebben gevolgd/mee bezig zijn of nog moeten doen.	De HG kan de vooruitgang van iedere klant bekijken.	Klanten kunnen hun gegevens inzien en hebben de mogelijkheid tot bewerken. De HG kan dit van iedere klant.	De klanten & HG hebben de mogelijkheid een profiel foto toe te voegen.	De klant en de HG kunnen een bio schrijven waarbij ze iets vertellen over zichzelf.	De HG kan een lijst inzien met alle huidige deelnemers/klanten en heeft de mogelijkheid hier aanpassingen in te maken.	De HG kan per iedere gebruikslos content aan zijn/haar account toevoegen.	De HG heeft de mogelijkheid een gebruiker te verwijderen.
12	**Testimonial pagina**									
13										
14	**App publicatie**	Iedere pagina (java class) testen en uitproberen voor de publicatie.	De app is vrij van known errors en bugs.	Alle stappen zijn doorlopen en de click-flow is getest voor publicatie.						
15										

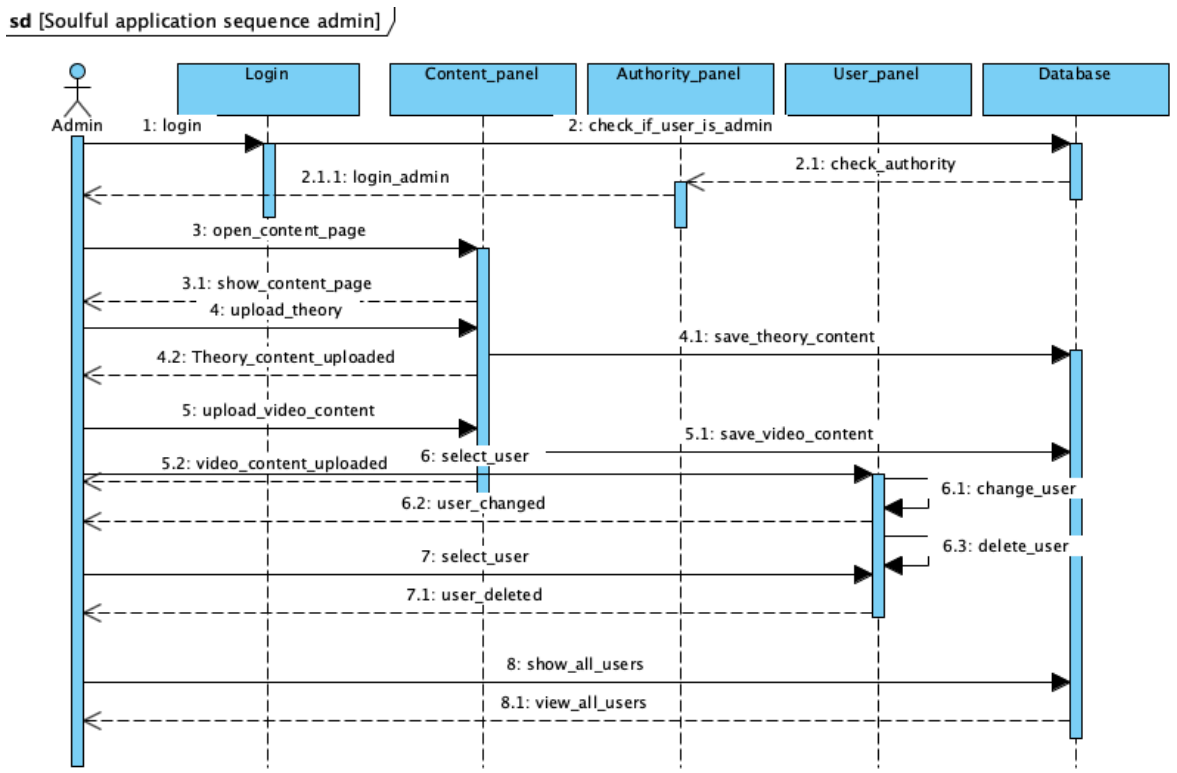
Bijlage B


SoulFull Yoga Application
Admin
Customer/User
upload
change
delete
register
logged_in
view
follow
purchase
view
Upload Content
extension points
Theory
Video
Change User
Delete User
Register
Login
View_Lesson
Follow_Lesson
Purchase_Lesson
Online Payment
View Customer Progress
Theory
Video
Enter Information
Valid for six month
<<Extend>>
<<Extend>>
<<Include>>
<<Include>>
<<Include>>
<<Include>>

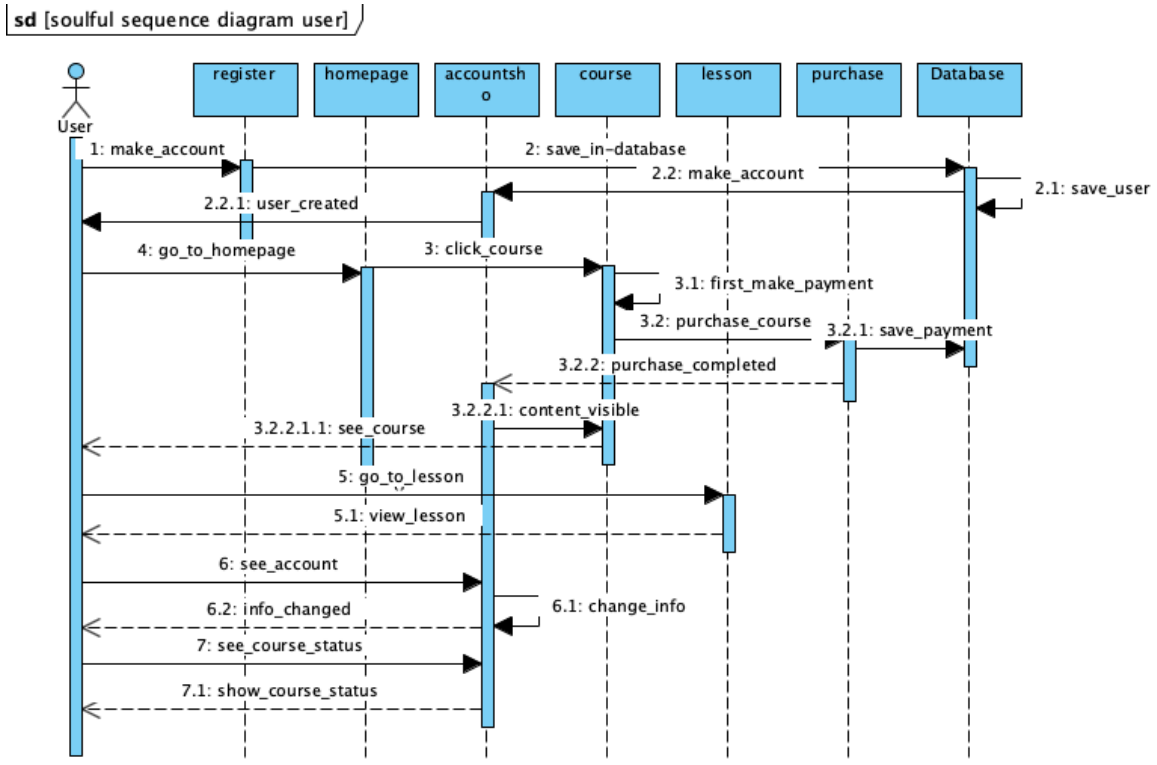
Bijlage C


User
-username : String
-password : String
-firstName : String
-lastName : String
-dateOfBirth : localDate
-address : String
-emailAddress : String
+Register() : void
+Login()
+Role()
Purchase
-purchaseID : long
-balance : dou
+chooseCourse()
+setPayment()
+showPurchaseHistory()
1..*
register
1
purchase course
1
show courses
Customer login
-Customer_ID : long
-password : String
+viewLesson()
+purchaseLesson()
+makePayment()
specify role
*
view
set Role
1
Role
-roleId : long
-name : String
-Description : String
+setRoleId()
Lessons
-id : long
-name : String
-videoLink : String
-content : String
-duration : double
-attribute
+getId() : long
+setId(id : long) : void
+getName() : String
+setName(name : String) : void
+getVideoLink() : String
+setVideoLink(videoLink : String) : void
+getContent() : String
+setContent(content : String) : void
+getDuration() : double
+setDuration(duration : double) : void
+getAttribute()
+setAttribute(attribute) : void
Course
-id : long
-courseName : String
-price : double
-thumbnail : String
-duration : double
+getId() : long
+setId(id : long) : void
+getCourseName() : String
+setCourseName(courseName : String) : void
+getPrice() : double
+setPrice(price : double) : void
+getThumbnail() : String
+setThumbnail(thumbnail : String) : void
+getDuration() : double
+setDuration(duration : double) : void
<<access>>

Bijlage D



Bijlage E



Bronnen

https://creately.com/blog/diagrams/uml-diagram-types-examples/

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

https://spring.io/guides/gs/testing-web/

https://www.baeldung.com/mockito-behavior

https://github.com/PeterAnema/backend_2021_08_hello/tree/master/src/main

https://spring.io/guides/tutorials/rest/