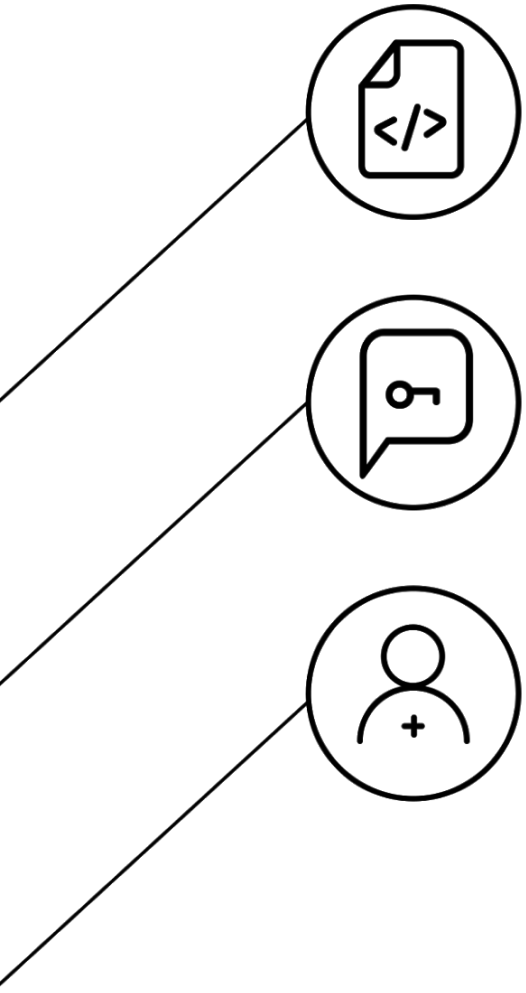

Integrale Eindopdracht

Hbo-bachelor ICT/Software Development

Leerlijn Backend

Opdrachtbeschrijving en deelopdrachten

Randvoorwaarden, structuur en beoordeling

NOVI HOGESCHOOL

Inhoud

Integrale eindopdracht Backend (30 EC) 3

Algemene opdrachtbeschrijving 3

Deelopdrachten 4

Voorbeeldcasus 5

Toelichting voorbeeldcasus en eisen aan de opdracht 5

Deelopdracht 1. Technisch Ontwerp 7

Deelopdracht 2. Verantwoordingsdocument 7

Deelopdracht 3. Broncode Spring Boot 8

Deelopdracht 4. Installatiehandleiding 9

Randvoorwaarden 10

Structuur 10

Beoordelingscriteria 11

Bijlage A: Template herkansingsfeedback Backend 13

Integrale eindopdracht Backend (30 EC)

De leerlijn Backend bevat de cursussen Java programmeren, Backend documentatie, Database Development, Spring Boot en Clean Code & Design patterns. Om deze leerlijn af te ronden dien je de volgende leeruitkomsten te realiseren:

1. **Java Programmeren**
   De student programmeert in Java, waarbij hij OOP-structuren toepast. Hierbij past de student automatisch testen toe en beheert hij externe code met behulp van Maven, waardoor men in een team aan Java-projecten kan werken. (LU1)
2. **Backend Documentatie**
   De student stelt, op basis van de Software Development Life Cycle, technische documentatie op voor de backend van een applicatie, gebruikmakend van UML-diagrammen. (LU2)
3. **Database Development**
   De student ontwerpt een relationele database, waarin data met onderlinge relaties veilig opgeslagen en uitgelezen kan worden, aan de hand van een technisch ontwerp document. Tevens beheert de student de data en rechten van databasegebruikers en voert hij CRUD-opdrachten uit op de database. (LU3)
4. **Spring Boot**
   De student zet een backend applicatie op met behulp van het Spring Boot framework en maakt gebruik van verschillende architecturale lagen binnen Spring. De student test zijn applicatie met unit-testen en het mocken van klassen. Tevens communiceert de applicatie met een database. (LU4)
5. **Clean Code & Design Patterns**
   De student schrijft zijn code volgens de afgesproken conventies van Clean Code en ontwikkelt highly cohesive en loose coupled code, door de toepassing van Design Patterns en SOLID. (LU5)

Algemene opdrachtbeschrijving

Dagelijks gebruiken we online applicaties zoals Microsoft Teams, Google Drive en YouTube. Al deze applicaties zijn voorzien van zowel een frontend als een backend. Bij backend development ligt de focus op dataopslag, server-side logica en beveiliging. Je ontwikkelt het hart van de applicatie en bent daarmee verantwoordelijk voor de verwerking, bewerking en opslag van data in de applicatie.

Opdracht: je gaat een applicatie bedenken en bouwen waarvan je alleen de backend gaat programmeren. Je bedenkt welke behoefte jouw product gaat vervullen en aan welke eisen de applicatie moet voldoen. Hiervoor ga je eerst een plan schrijven en teken je de architectuur uit met behulp van UML. Daarna ga je de backend code schrijven.

Door deze opdracht uit te voeren toon je aan een volwaardige webapplicatie te kunnen bouwen in de backend. In het volgende hoofdstuk vind je meer informatie over de deelopdrachten.

Om de leerlijn Backend met succes af te ronden is een voldoende nodig voor de integrale eindopdracht. Met de deelopdrachten kunnen geen losse cursussen worden afgerond.

Op te leveren producten:

- Technisch ontwerp met daarin o.a. de probleembeschrijving, functionele- en niet-functionele-eisen, klassendiagram en sequentiediagrammen;
- Verantwoordingsdocument;
- De broncode van een Spring Boot webapplicatie;
- Zelfgeschreven installatiehandleiding;

Deze producten worden ingeleverd als één ZIP-bestand.

Deelopdrachten

We gaan jouw vaardigheden als backend ontwikkelaar toetsen door je een beveiligde RESTful webservice te laten bouwen. Door verschillende CRUD-opdrachten in te richten, communiceert de applicatie met de relationele database om data zowel op te slaan als op te halen. Omdat je gebruikersaccounts ondersteunt met verschillende rollen en rechten, zorg jij ook voor authenticatie en autorisatie door jouw endpoints goed te beveiligen.

Voordat je begint met programmeren, maak je een gestructureerd plan van aanpak. Tijdens het ontwikkelen documenteer je jouw keuzes. Al deze stappen zijn opgedeeld in deelopdrachten.

Voordat je kunt starten met de eerste deelopdracht lever je eerst jouw casus ter goedkeuring in bij de docent. Je mag ervoor kiezen om de wensen van de voorbeeldcasus te vertalen naar een idee voor jouw webapplicatie, of om een eigen casus te bedenken. In dit geval zorg je dat je de voorbeeldcasus goed bestudeerd, zodat je weet hoe je alle elementen terug kunt laten komen in jouw eigen casus.

Je beschrijft in maximaal 250 woorden:

- Welke behoefte je wil vervullen met deze applicatie/welke functie de applicatie heeft;
- Welke gebruikersrollen je wil gebruiken;
- Welke data je wil kunnen opslaan;
- Een genummerde opsomming van de 5 belangrijkste functionaliteiten van jouw applicatie;

Na goedkeuring van de docent op de door jou te ontwikkelen applicatie kun je aan de slag met de eerste deelopdracht.

Voorbeeldcasus

Voor mijn eindopdracht ga ik het backend systeem van een autogarage programmeren. In deze garage komen klanten hun auto afleveren voor een reparatie. Een administratief medewerker voegt de klant en de auto toe aan het systeem wanneer de klant en/of de auto voor het eerst bij de garage komen. De medewerker plant vervolgens een moment in om de auto te keuren. Tijdens deze registratie kunnen de autopapieren in pdf-formaat toegevoegd worden.

Een monteur keurt vervolgens de auto en voegt de gevonden tekortkomingen toe aan de auto in het systeem. Nadat de auto gekeurd is, neemt de monteur contact op met de klant. Gaat de klant akkoord met de reparatie, dan maakt de monteur een afspraak om de auto te repareren.

Gaat de klant niet akkoord met de reparatie? Dan zet de monteur dat in het systeem, maakt hij de bon (45 euro) op voor de keuring, kan de klant de auto komen ophalen en wordt de reparatie op 'niet uitvoeren' gezet.

Wanneer de klant akkoord gaat, voegt de monteur toe aan de bon wat afgesproken is en gaat hij de auto repareren. Elk gebruikt onderdeel en handeling worden toegevoegd aan de reparatie. Vervangt de monteur bijvoorbeeld de remschijf? Dan wordt het onderdeel ‘remschijf’ aan de reparatie toegevoegd en wordt de handeling ‘remschijf vervangen’ aan de reparatie toegevoegd.

Al deze onderdelen en handelingen staan al, inclusief prijs, in het systeem. De monteur hoeft deze opgeslagen handelingen en onderdelen alleen maar te selecteren. Omdat een monteur soms iets specifieks moet doen, kan de monteur ook een 'overige' handeling en prijs toevoegen.

Wanneer de klant de auto komt ophalen, zal een kassamedewerker door het systeem een bon laten genereren met alle onderdelen en handelingen die door de monteur zijn toegevoegd. De bon bevat de keuring + bedrag, de handelingen + bedrag en de onderdelen + bedrag. Bij alle bedragen moet het BTW-tarief nog berekend worden voordat de bedragen op de bon getoond worden. Wanneer de klant betaald heeft, wordt de status op betaald gezet.

Daarnaast is er een backoffice medewerker die onderdelen (naam, prijs, voorraad) kan toevoegen aan het systeem, voorraden kan aanpassen en reparatiehandelingen (naam, prijs) kan toevoegen aan het systeem. Alle prijzen in het systeem zijn exclusief BTW.

Toelichting voorbeeldcasus en eisen aan de opdracht

In bovenstaande casus heeft de student verschillende user-rollen genoemd. Hij toont aan dat de applicatie data moet kunnen opslaan in de database, data moet kunnen ophalen uit de database en dat er bestanden geüpload kunnen worden. De opdracht voldoet dus aan de minimale eisen die aan de applicatie gesteld worden.

De verschillende user-rollen die genoemd zijn, monteur, kassamedewerker en backoffice medewerker. Deze rollen hebben allemaal taken die alleen zij kunnen uitvoeren. Met andere woorden, deze rollen geven de gebruiker autorisatie om bepaalde handelingen met het systeem uit te voeren.

Het toevoegen van klanten en auto's, het toevoegen van onderdelen en handelingen aan reparaties en het aanpassen van de reparatiestatus zijn allemaal voorbeelden van CRUD-operaties. Het opmaken van de bon is een voorbeeld van het combineren van data uit verschillende tabellen.

De student heeft dus een casus beschreven die voldoet aan de gestelde functionele eisen.

Deelopdracht 1. Technisch Ontwerp

Wanneer je een webapplicatie gaat bouwen, kun je niet zomaar beginnen met programmeren. Het is belangrijk eerst te inventariseren hoe het product moet werken door functionaliteiten voor verschillende rollen te beschrijven. Functionaliteiten beschrijven wat er mogelijk is met de applicatie, zoals: inloggen, gegevens opslaan, data wijzigen etc. Daarnaast ga je bepalen welke data je nodig hebt. Vervolgens kun je de architectuur uittekenen door middel van een klassendiagram en sequentiediagrammen.

Je gaat een technisch ontwerp maken dat voldoet aan de volgende eisen:

- Bevat een titelblad, inleiding en inhoudsopgave. In het documenten zitten geen verwijzingen naar afbeeldingen en informatie buiten het document zelf.
- Beschrijft het probleem en de manier waarop deze applicatie dat probleem oplost.
- Beschrijft wat de applicatie moet kunnen middels een sommering van 25 functionele en niet-functionele eisen.
- Bevat één klassendiagram van alle entiteiten. Dit klassendiagram is taal- en platformafhankelijk en hoeft geen methodes te bevatten.
- Bevat minimaal twee volledig uitgewerkte sequentiediagrammen waarin alle architecturale lagen (controller, service, repository) voorkomen. Zorg ervoor dat deze diagrammen klasse- en methodenamen bevatten die overeenkomen met de namen die je gebruikt in de broncode (Deelopdracht 2).

Op te leveren: Het technisch ontwerp van de webapplicatie in PDF (.pdf).

Deelopdracht 2. Verantwoordingsdocument

Zodra je begint met programmeren, ga je ook beginnen aan het verantwoordingsdocument. Hierin leg je vast welke technische ontwerpbeslissingen je maakt en *waarom* je deze keuzes gemaakt hebt. Dit werkt het beste als je dit al tijdens het ontwikkelen van jouw product begint te maken. Waarom heb je deze specifieke Java-library gebruikt en niet een andere? Waarom ben je van conventies of architecturale richtlijnen afgeweken? Welke doorontwikkelingen zijn er mogelijk of misschien zelfs wenselijk, en waarom heb je deze zelf niet door kunnen voeren? Heb je bijvoorbeeld iets achterwege gelaten in verband met een tekort aan tijd? Leg dan uit wat je liever had willen doen als je meer tijd had gehad. Reflecteer hierbij ook op je eigen leerproces: wat ging goed en wat kan de volgende keer beter? Let op: technieken die als randvoorwaarden gesteld zijn in de eindopdracht tellen niet mee.

Op te leveren:

- Verantwoordingsdocument in PDF (.pdf) met daarin:
    - Minimaal 10 beargumenteerde technische keuzes.
    - Een beschrijving van limitaties van de applicatie en eventueel argumentatie van mogelijke doorontwikkelingen.
    - Een link naar jouw project op Github.
- Indien je een herkansing inlevert na het krijgen van feedback, lever je ook het ingevulde ‘Template herkansingsfeedback’ (bijlage A) in.

Deelopdracht 3. Broncode Spring Boot

In de eerste opdracht heb je uitgewerkt wat de applicatie moet kunnen en hoe de architectuur eruit moet komen te zien. In deze opdracht ga je jouw technisch ontwerp als basis gebruiken om de webapplicatie te implementeren in Spring Boot. Jouw webapplicatie heeft **minimaal** de volgende eigenschappen:

- Het is een RESTful webservice die data beheert via endpoints;
- De applicatie is beveiligd en bevat minimaal 2 en maximaal 3 user-rollen met verschillende mogelijkheden;
- De applicatie en database zijn onafhankelijk van elkaar waardoor het mogelijk is om eventueel naar een ander database systeem te wisselen (zoals MySQL, PostgreSQL, SQLite);
- Communicatie met de database vindt plaats door middel van repositories. De database kan in de vorm van CRUD operaties of complexere, samengestelde, queries bevraagd worden.
- De database is relationeel en bevat minimaal een 1 one-to-one relatie en 1 one-to-many relatie;
- De applicatie maakt het mogelijk om bestanden (zoals muziek, PDF's of afbeeldingen) te uploaden en te downloaden;
- Het systeem wordt geleverd met een valide set aan data en unit-tests worden voorzien van eigen test data.
- De application context van de applicatie wordt getest met Spring Boot test, WebMvc en JUnit.

Op te leveren:

- Projectmap met daarin de broncode, inclusief de link naar de Github repository.

Deelopdracht 4. Installatiehandleiding

In de voorgaande opdrachten heb je jouw ontwikkelwerk afgerond. Om ervoor te zorgen dat ook andere ontwikkelaars jouw project kunnen gebruiken, is het belangrijk een installatiehandleiding te schrijven waarin beschreven wordt wat zij hiervoor nodig hebben. Je schrijft jouw installatiehandleiding die uitlegt hoe de applicatie geïnstalleerd en gebruikt kan worden. Dit schrijf je voor een ontwikkelaar zonder enige ervaring binnen het backend-landschap.

Jouw installatiehandleiding bevat:

- Een inhoudsopgave en inleiding, met daarin een korte beschrijving van de functionaliteit van de applicatie en de gebruikte technieken;
- Een lijst van benodigdheden om de applicatie te kunnen runnen (zoals applicaties, runtime environments of andere benodigdheden);
- Een stappenplan met installatie instructies;
- Een lijst met (test)gebruikers en user-rollen;
- Een Postman collectie, die gebruikt kan worden om jouw applicatie te testen.
- Een lijst van REST-endpoints, inclusief voorbeelden van de JSON-requests. Deze voorbeelden moeten uitgeschreven zijn zoals in Postman, zodat je ze gemakkelijk kunt selecteren, kopiëren en plakken. Hierin leg je ook uit hoe de endpoints beveiligd zijn.

Op te leveren:

- Installatiehandleiding in PDF (.pdf) of mark down (.md)

Randvoorwaarden

Hieronder vind je een aantal randvoorwaarden waaraan je eindopdracht moet voldoen, naast de gestelde eisen aan de applicatie (zie 'Deelopdracht 3').

- Alleen Spring Boot met een versie van Java voor long term support, zoals Java 11 of Java 17 wordt geaccepteerd als programmeertaal.
- Er wordt gebruikgemaakt van Maven als dependency manager.
- Er wordt gebruikgemaakt van DTO's om data te valideren en vervuiling van de database te voorkomen.
- Het project is geüpload naar een GitHub repository: deze repository staat op *public*. De link is toegevoegd in jouw verantwoordingsdocument.
- Het project wordt ingeleverd *zonder* out-map/target-map/.idea-map en .iml-bestand.
- De applicatie start op zonder crashes.
- Het project wordt aangeleverd in een ZIP-bestand van maximaal 50 MB. (Met de extensie .zip. Projecten die als .rar worden aangeleverd, worden niet geaccepteerd.) Tip: blijkt jouw bestand veel groter dan 50 MB, dan ben je waarschijnlijk vergeten jouw target-map te verwijderen.

Structuur

Het technisch ontwerp is een verzorgd document met een titelblad, inhoudsopgave en inleiding. Het verantwoordingsdocument mag toegevoegd worden als laatste hoofdstuk in het technisch ontwerp.

In de documenten zitten geen verwijzingen naar afbeeldingen, diagrammen of modellen buiten het document zelf. Ook zijn de diagrammen en afbeeldingen goed leesbaar en tekstueel uitgelegd of beschreven. Een goed uitgangspunt hier is dat wanneer het document geprint wordt, het nog steeds volledig beoordeeld kan worden. Er is geen harde richtlijn met betrekking tot het aantal woorden, maar zorg ervoor dat je beknopt en bondig schrijft.

Beoordelingscriteria

De eindopdracht wordt beoordeeld op basis van de volgende beoordelingscriteria. Per criterium kent de beoordelaar een aantal punten toe. Zodoende wordt bepaald in hoeverre je de betreffende leeruitkomst aantoonbaar hebt gerealiseerd.

# Deelopdracht	**Leeruitkomsten en beoordelingscriteria**	**Weging**
1. Functioneel en Technisch Ontwerp	**Bevat aspecten van de leeruitkomsten van de cursussen Backend Documentatie (LU2) en Spring Boot (LU4).**	**20%**
Criterium 1.1	De student vertaalt de casus (of een eigen idee voor een applicatie) naar een opsomming van tenminste 25 relevante functionele en niet-functionele eisen. (LU2)	5%
Criterium 1.2	De student structureert in het technisch ontwerp de functionaliteiten in een overzichtelijke en logisch gestructureerde klassendiagram, waarbij hij kardinaliteiten bij de associaties aangeeft tussen verschillende klassen. Het klassendiagram is taal- en platformonafhankelijk en bevat geen methodes of architecturale lagen. (LU2 en LU4)	10 %
Criterium 1.3	De student legt op logische en correcte wijze alle verschillende architecturale lagen (controller, service en repository) vast in twee duidelijke en complete sequentiediagrammen. Deze sequentiediagrammen bevatten klasse- en methodenamen die overeenkomen met die in het ingeleverde project. (LU2 en LU4)	5%
2. Software schrijven	**Betreft aspecten van de leeruitkomsten van alle cursussen van deze leerlijn.**	**70%**
Criterium 2.1	De student implementeert 5 belangrijke kern- functionaliteiten op correcte en kwalitatieve wijze. (LU1 en LU4)	10%
Criterium 2.2	De student maakt een kwalitatieve Java-applicatie waarbij hij logica implementeert door het gebruik van attributen, methoden, overerving, interfaces en abstracte klassen. (LU1)	10%
Criterium 2.3	De student voert minimaal 2 geslaagde integratie-tests uit door de application context van zijn applicatie te testen met Spring Boot test en WebMvc. Tevens test de student twee of drie klassen uit de service laag met een line coverage van 100% door middel van minimaal 10 unit-tests, gebruikmakend van de drie A's. (LU1 en LU4)	10%
Criterium 2.4	De student past exception handling op effectieve wijze toe bij het beheren van mogelijke errors binnen de applicatie. (LU1)	5%

Criterium 2.5	De student beheert zijn code met correct gebruik van Git om met versiebeheer de voortgang van het project vast te leggen. De student werkt op verschillende branches, maakt kleine commits met compacte en zinvolle commit-messages, maakt pull requests en merged regelmatig. (LU2)	5%
Criterium 2.6	De student maakt logisch gebruik van DTO's waarmee validatie wordt toegepast op data van en naar de database. Tevens stelt hij relevante query's op om testdata uit de database op te halen. (LU3 en LU4)	5%
Criterium 2.7	De student past, door correct gebruik van Spring Security, authenticatie toe op zowel de gebruiker die in de database is opgeslagen als op het gecodeerde wachtwoord. Daarnaast past hij autorisatie toe op de endpoints, door adequaat gebruik te maken van een JWT. (LU3 en LU4)	10%
Criterium 2.8	De student implementeert een kwalitatieve webservice volgens de RESTful richtlijnen, waarbij hij HTTP-methodes gebruikt om de vertaalslag te maken naar acties met de data. (LU3)	10%
Criterium 2.9	De student past de principes van Clean Code en SOLID correct toe. (LU5)	5%
3. Verantwoordings-document en Installatie-handleiding	**Betreft aspecten van de leeruitkomst van de cursus Backend Documentatie. (LU2)**	**10%**
Criterium 3.1	De student schrijft op basis van de Software Development Life Cycle een duidelijke installatiehandleiding waarmee de applicatie door een collega-developer kan worden geïnstalleerd. Deze is voorzien van een stappenplan, een lijst van benodigdheden, testgebruikers, userrollen en rest-endpoints. (LU2)	5%
Criterium 3.2	De student schrijft een volledig, goed gestructureerd en duidelijk verantwoordingsdocument waarin hij een beargumenteerd overzicht geeft van minimaal 10 toegepaste technieken. (LU2)	5%
		Totaal 100%

Bijlage A: Template herkansingsfeedback Backend

Verantwoordingsdocument

Heb je jouw eindopdracht ingeleverd en heb je een onvoldoende gehaald? Dan verwerk je een extra hoofdstuk in jouw verantwoordingsdocument waarin je aantoont wat je precies hebt verbeterd naar aanleiding van de feedback van de docent.
Dit doe je per beoordelingscriterium, waarin je ook de feedback en het originele cijfer benoemt. Gebruik hiervoor onderstaand template. In het geval dat je feedback *niet* verbeterd hebt, geef je dit ook aan.

Criterium 1.1 – De student vertaalt de casus (of een eigen idee voor een applicatie) naar een opsomming van tenminste 25 relevante functionele en niet-functionele eisen.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 1.2 – De student structureert in het technisch ontwerp de functionaliteiten in een overzichtelijke en logisch gestructureerde klassendiagram, waarbij hij kardinaliteiten aangeeft tussen verschillende klassen. Het klassendiagram is taal- en platformonafhankelijk en bevat geen methodes of architecturale lagen	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 1.3 – De student legt op logische en correcte wijze alle verschillende architecturale lagen (controller, service en repository) vast in twee duidelijke en complete sequentiediagrammen. Deze sequentiediagrammen bevatten klasse- en methodenamen die overeenkomen met die in het ingeleverde project.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 2.1 – De student implementeert 5 belangrijke kern- functionaliteiten op correcte en kwalitatieve wijze.	
Feedback docent:	*[cijfer]*

Toelichting verbeteringen student

Criterium 2.2 – De student maakt een kwalitatieve Java-applicatie waarbij hij logica implementeert door het gebruik van attributen, methoden, overerving, interfaces en abstracte klassen.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 2.3 – De student voert minimaal 2 kwalitatieve integratie-tests uit door de application context van zijn applicatie te testen met Spring Boot test en WebMvc. Tevens test de student twee of drie klassen met een line coverage van 100% door middel van minimaal 10 unit-tests, gebruikmakend van de drie A's.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 2.4 – De student past exception handling op effectieve wijze toe bij het beheren van mogelijke errors binnen de applicatie.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 2.5 – De student beheert zijn code met correct gebruik van Git om met versiebeheer de voortgang van het project vast te leggen. De student werkt op verschillende branches, maakt kleine commits met compacte en zinvolle commit-messages, maakt pull requests en merged regelmatig.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 2.6 – De student maakt logisch gebruik van DTO's waarmee validatie wordt toegepast op data van en naar de database. Tevens stelt hij relevante query's op om testdata uit de database op te halen

Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 2.7 – De student past, door correct gebruik van Spring Security, authenticatie toe op zowel de gebruiker die in de database is opgeslagen als op het gecodeerde wachtwoord. Daarnaast past hij autorisatie toe op de endpoints, door adequaat gebruik te maken van een JWT.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 2.8 – De student implementeert een kwalitatieve webservice volgens de RESTful richtlijnen, waarbij hij HTTP-methodes gebruikt om de vertaalslag te maken naar acties met de data.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 2.9 – De student past de principes van Clean Code en SOLID correct toe.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 3.1 – De student schrijft op basis van de Software Development Life Cycle een duidelijke installatiehandleiding waarmee de applicatie door een collega-developer kan worden geïnstalleerd. Deze is voorzien van een stappenplan, een lijst van benodigdheden, testgebruikers, userrollen en rest-endpoints.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	

Criterium 3.1 – De student schrijft een volledig, goed gestructureerd en duidelijk verantwoordingsdocument waarin hij een beargumenteerd overzicht geeft van minimaal 10 toegepaste technieken.	
Feedback docent:	*[cijfer]*
Toelichting verbeteringen student	