Back-end: Spring Boot: D&D Character Generator.

Thomas Gielen.

Scriptie voorgedragen tot het bekomen van de graad van
Graduaat in het Programmeren

Promotor: Luc Vervoort
Co-promotor: David Breckx
Academiejaar: 2024–2025
Eerste examenperiode

Departement IT en Digitale Innovatie .

HO
GENT

Woord vooraf

Voor u ligt mijn graduaatsproef, opgesteld in het kader van het behalen van het diploma Graduaat Programmeren.
Voor dit eindproject koos ik ervoor om een RESTful API te ontwikkelen voor het beheren van Dungeons Dragons-personages. Dit onderwerp combineert mijn interesse in games met mijn passie voor backendontwikkeling, en gaf me de kans om een praktische en boeiende toepassing uit te werken.
Een extra uitdaging was het gebruik van Spring Boot, een technologie waarmee ik tot dit project nog niet had gewerkt. Het was een leerzaam proces waarbij ik veel nieuwe concepten heb ontdekt, en waarbij ik stap voor stap mijn weg heb gevonden in het bouwen van een gestructureerde en functionele API.
Ik wil graag mijn familie en vrienden bedanken voor hun geduld, begrip en steun tijdens dit traject. Hun aanmoediging heeft me geholpen om gemotiveerd te blijven en het project tot een goed einde te brengen.
Met trots presenteer ik het resultaat van mijn inspanningen en hoop ik dat dit project mijn groei als ontwikkelaar weerspiegelt.

Samenvatting

Deze graduaatsproef behandelt de ontwikkeling van een backend API voor het creëren en beheren van Dungeons & Dragons-personages, met behulp van het Java-framework Spring Boot. Het onderwerp is gekozen vanwege de groeiende populariteit van digitale hulpmiddelen in rollenspellen en de behoefte aan flexibele, uitbreidbare oplossingen voor spelers en ontwikkelaars.
Het onderzoek richt zich op de vraag hoe een API ontworpen en geïmplementeerd kan worden die verschillende karakterklassen, rassen en statistieken ondersteunt, terwijl ook gebruikersauthenticatie is geïntegreerd. De doelstelling was het ontwikkelen van een functioneel prototype dat de kernfunctionaliteiten voor het beheren van personages omvat.
De methodologie bestond uit een documentation-first aanpak, waarbij met behulp van Apidog een OpenAPI-specificatie werd opgesteld voorafgaand aan de implementatie. Dit zorgde voor duidelijke richtlijnen tijdens de ontwikkeling en maakte de API goed testbaar en uitbreidbaar.
De resultaten tonen aan dat met Spring Boot een efficiënte en goed gestructureerde API kan worden gerealiseerd die voldoet aan de eisen van de onderzoeksvraag. Het prototype biedt een stabiele basis voor verdere uitbreiding, zoals het toevoegen van een gebruikersinterface of extra spelmechanismen.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van codefragmenten

1

Inleiding

Deze graduaatsproef richt zich op het ontwikkelen van een RESTful API met Spring Boot voor het aanmaken en beheren van Dungeons & Dragons-personages. Dungeons & Dragons (D&D) is een populair rollenspel waarbij spelers unieke personages creëren met eigen rassen, klassen en eigenschappen. De API ondersteunt dit proces digitaal en gestructureerd.
Het project maakt gebruik van Spring Boot, een modern Java-framework dat binnen deze toepassing voor het eerst wordt gebruikt. De API is ontworpen volgens het documentation-first-principe met behulp van Apidog, wat zorgt voor duidelijke en consistente documentatie van de verschillende functionaliteiten.
De toepassing biedt ondersteuning voor het aanmaken, bewerken en verwijderen van personages, inclusief rassen, klassen en gebruikersauthenticatie. Geavanceerde spelmechanismen zoals gevechten of campagnes vallen buiten de scope.
De centrale onderzoeksvraag luidt: Hoe kan met behulp van Spring Boot een goed gestructureerde en bruikbare API ontwikkeld worden voor het beheren van D&D-personages?
Het doel is een functionele en uitbreidbare backendtoepassing te realiseren die voldoet aan moderne ontwikkelingsprincipes.

1.1. Probleemstelling

Het beheren van personages in Dungeons & Dragons gebeurt vaak handmatig of via tools die beperkt, betalend of moeilijk uitbreidbaar zijn. Veel spelers en spelbegeleiders (Dungeon Masters) gebruiken losse documenten, spreadsheets of niet-gecentraliseerde websites, wat leidt tot fouten, onduidelijkheden en verlies van gegevens.
Voor kleine spelgroepen, individuele D&D-spelers en ontwikkelaars van digitale DD-tools is er behoefte aan een eenvoudige, uitbreidbare en gratis backendoplossing

waarmee personages digitaal kunnen worden aangemaakt, bewerkt en bewaard. Vooral gebruikers die een eigen tool, app of interface willen bouwen, hebben nood aan een goed gestructureerde, open API als basis.
Door een gebruiksvriendelijke en goed gedocumenteerde REST API te voorzien, kan deze toepassing een duidelijke meerwaarde bieden aan deze specifieke doelgroep, die vandaag vaak afhankelijk is van gefragmenteerde of commerciële oplossingen.

1.2. Onderzoeksvraag

Hoe kan met behulp van Spring Boot een uitbreidbare en goed gedocumenteerde API ontwikkeld worden die het aanmaken en beheren van Dungeons & Dragons-personages ondersteunt voor kleine spelgroepen en individuele spelers?

1.3. Onderzoeksdoelstelling

Het doel van deze graduaatsproef is het ontwikkelen van een functionele en uitbreidbare RESTful API met behulp van Spring Boot, waarmee Dungeons Dragons-personages kunnen worden aangemaakt, beheerd en verwijderd. De API moet logisch en intuïtief opgebouwd zijn, zodat gebruikers zoals spelers en ontwikkelaars er gemakkelijk mee kunnen werken. Daarnaast is het belangrijk dat de structuur modulair en flexibel is, zodat in de toekomst extra spelregels en functionaliteiten eenvoudig kunnen worden toegevoegd. Om de persoonlijke gegevens van gebruikers te beschermen, wordt er een werkende gebruikersauthenticatie geïmplementeerd. De API zal volledig en duidelijk gedocumenteerd worden via Apidog volgens het documentation-first principe, waardoor externe ontwikkelaars de API eenvoudig kunnen gebruiken en integreren. Dit project wordt gerealiseerd als een proof of concept dat de basisfunctionaliteiten operationeel toont en kan dienen als uitgangspunt voor verdere ontwikkeling of integratie met frontend-applicaties. Met deze graduaatsproef wordt aangetoond dat het mogelijk is om met Spring Boot een degelijke backend voor Dungeons Dragons-personagebeheer te realiseren die aansluit bij de behoeften van kleine spelgroepen en individuele spelers.

1.4. Opzet van deze graduaatsproef

De rest van deze graduaatsproef is als volgt opgebouwd:
In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.
In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.
In Hoofdstuk 6, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toe-

komstig onderzoek binnen dit domein.

2

Stand van zaken

2.1. Stand van Zaken

Dit hoofdstuk geeft een overzicht van de huidige stand van zaken rond digitale hulpmiddelen voor D&D Beyond (**Bradford2025**). De inhoud bouwt voort op de inleiding en spitst zich toe op de technologische ontwikkelingen die het spelen van D&D Beyond digitaal ondersteunen. Zo wordt de lezer volledig geïnformeerd over de state-of-the-art op dit gebied, zodat die het verdere verhaal zonder aanvullende voorkennis kan volgen.

Digitale platforms zoals **D&D Beyond** bieden uitgebreide functionaliteiten voor het creëren en beheren van personages, het bijhouden van campagnes en het integreren van digitale dobbelstenen en kaarten (**Bradford2025**). Andere tools zoals **Roll20** (**Melzer2024**) en **Fantasy Grounds** stellen spelers en Dungeon Masters in staat virtuele tafelopstellingen te gebruiken met geavanceerde functies, waaronder dynamische kaarten, automatische dobbelsteenrollen en geïntegreerde chatfunctionaliteit (**Hall2015**). Hoewel deze platforms een robuuste gebruikerservaring bieden, zijn ze vaak gesloten systemen met beperkte mogelijkheden tot aanpassing en integratie, wat de vraag naar open en uitbreidbare oplossingen versterkt.

Voor de ontwikkeling van webservices wordt vaak gekozen voor **RESTful API's**, die stateless communicatie tussen client en server mogelijk maken en zo schaalbaarheid en eenvoud in het gebruik bevorderen (**restfull**). In de Java-omgeving is **Spring Boot** een gangbaar framework voor het snel opzetten van dergelijke API's, doordat het tal van ingebouwde configuraties en ontwikkeltools aanbiedt (**Pratik2024**). Hierdoor kunnen ontwikkelaars vlot een robuuste backend creëren die gemakkelijk integreert met verschillende frontend-applicaties en systemen.

Een moderne methodologie in API-ontwikkeling is de *API-Design First* benadering, waarbij het ontwerp van de API voorafgaat aan de implementatie. Deze werkwijze bevordert duidelijkheid en verbetert de samenwerking tussen ontwikkelaars

en andere betrokkenen (**apidog**). Het platform **Apidog** ondersteunt deze aanpak door hulpmiddelen te bieden voor visueel ontwerpen, testen en documenteren van API's, wat bijdraagt aan snellere ontwikkeling en hogere kwaliteit.
Tot slot is beveiliging een essentieel aandachtspunt bij API-ontwikkeling, vooral bij het verwerken van gevoelige data. Een veelgebruikte authenticatiemethode is het gebruik van **JSON Web Tokens (JWT)**, die gebruikers op een veilige en efficiënte wijze authenticeren en autoriseren zonder dat servers sessie-informatie hoeven op te slaan (**Gordadze**).

3

Methodologie

3.1. Ontwerp met Apidog

Voor de ontwikkeling van de API werd gekozen voor een *API-Design First* benadering. Hierbij werd direct begonnen met het ontwerpen van de API in **Apidog**, een tool die het mogelijk maakt om API-specificaties visueel en overzichtelijk op te stellen. Met Apidog werden alle benodigde endpoints, request- en responseformaten en validatieregels vastgelegd voordat de daadwerkelijke implementatie startte. Dit zorgde voor duidelijkheid over wat de API moest doen en maakte het mogelijk om vroegtijdig te testen en de documentatie automatisch te genereren. Door deze gestructureerde aanpak werden fouten tijdens de implementatie beperkt en kon het ontwikkelproces efficiënter verlopen.

3.2. Implementatie met Spring Boot

Op basis van het ontwerp uit Apidog werd de backend van de API geïmplementeerd met het **Spring Boot** framework. Spring Boot biedt een snelle en gestroomlijnde manier om RESTful API's te ontwikkelen in Java, met ingebouwde ondersteuning voor onder andere routing, datahandling en beveiliging. Tijdens de implementatie werden functionaliteiten ontwikkeld om Dungeons & Dragons-personages aan te maken, te beheren en gebruikersauthenticatie te ondersteunen. De structuur en specificaties vanuit Apidog dienden als leidraad, zodat de implementatie nauw aansloot bij het vooraf opgestelde ontwerp. Daarnaast werden testen uitgevoerd om te garanderen dat de API correct functioneerde en voldeed aan de gestelde eisen.

4

Ontwerp met Apidog

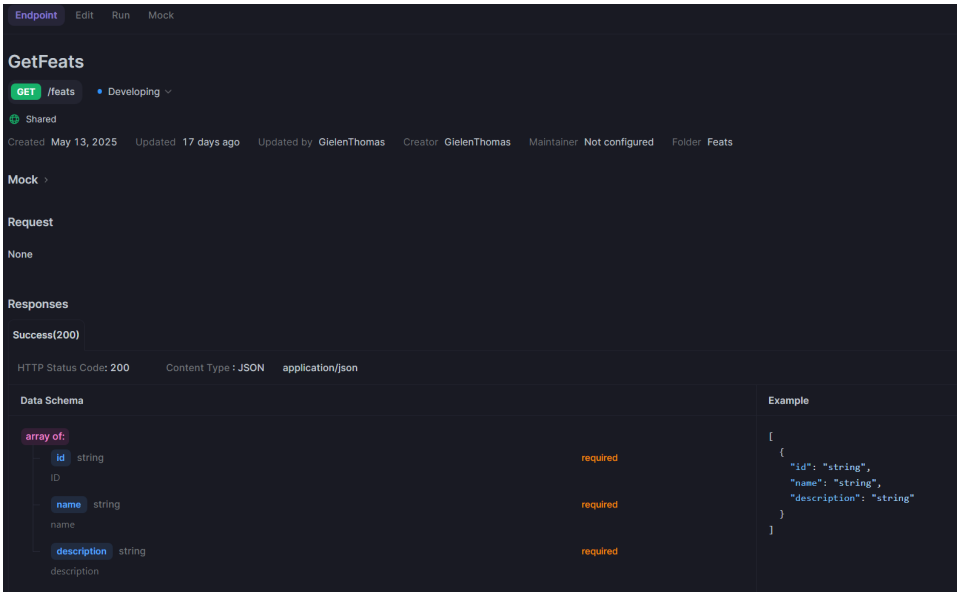
In dit hoofdstuk wordt het gebruik van **Apidog** toegelicht voor het ontwerpen van de API. Apidog maakt het mogelijk om OpenAPI-specificaties visueel op te stellen, te valideren en automatisch te documenteren, wat het ontwikkelproces gestructureerd en transparant maakt.
Met Apidog werden alle API-endpoints, datamodellen en parameters opgesteld volgens de OpenAPI-standaard. Hieronder is een voorbeeld van een datamodel in dogApi. Hieronder staan er voorbeelden van hoe het in de dogAPi er uitziet en wat het genereert.

```
        "/feats": {
            "get": {
                "summary": "GetFeats",
                "deprecated": false,
                "description": "",
                "tags": [],
                "parameters": [],
                "responses": {
                    "200": {
                        "description": "",
                        "content": {
                            "application/json": {
                                "schema": {
                                    "type": "array",
                                    "items": {
                                        "$ref": "#/compo-
                                            nents/schemas/FeatResponse"
                                    }
                                }
                            },
                            "headers": {}
                        }
                    },
                    "security": []
            },
```

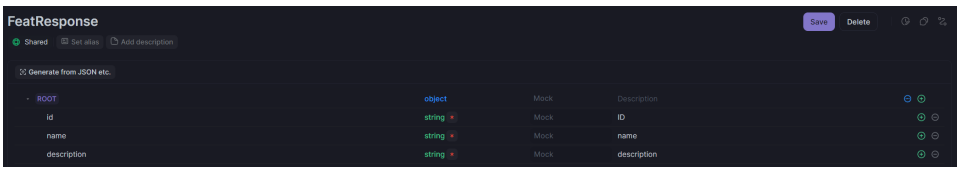
Codefragment 4.1: End-point van de OpenAPI spec

```
        "FeatResponse": {
            "type": "object",
            "properties": {
                "id": {
                    "type": "string",
                    "description": "ID"
                },
                "name": {
                    "type": "string",
                    "description": "name"
                },
                "description": {
                    "type": "string",
                    "description": "description"
                }
            },
            "required": [
            "name",
            "description",
            "id"
            ]
        },
```

Codefragment 4.2: Data model van de OpenApi spec



Figuur 4.1: Voorbeeld van een endpoint in dogApi



Figuur 4.2: Voorbeeld van een data model in dogApi

5

Spring Boot

Om de ontwikkeling te versnellen en consistentie te garanderen tussen de documentatie en de implementatie, werd gebruikgemaakt van de door Apidog gegenereerde OpenAPI-specificatie. Deze specificatie werd vervolgens gebruikt om automatisch Java-klassen te genereren, zoals:

- **Request- en response-DTO's**, gebaseerd op de schemas in de OpenAPI;
- **Controllerinterfaces**, met methodesignatures die overeenkomen met de beschreven endpoints;

De gegenereerde klassen vormen de basis van de Spring Boot-implementatie. Op deze manier werd een documentation-first aanpak gevolgd, waarbij de documentatie leidend is voor de implementatie.

5.0.1. Implementatie van de controllers

De gegenereerde controllerinterfaces definiëren de API-endpoints en methodesignatures, maar bevatten nog geen concrete implementaties. Daarom zijn eigen controllerklassen geschreven die deze interfaces implementeren.
Deze implementatieklassen bevatten de businesslogica en roepen services aan om de gevraagde operaties uit te voeren, zoals het aanmaken of ophalen van Dungeons & Dragons-personages. De `@Secured` controleerd of de gebruiker de juiste rol heeft om deze route aan te roepen.
Deze aanpak zorgt ervoor dat de contracten uit de OpenAPI-specificatie strikt gevolgd worden, terwijl er toch volledige controle is over de uitvoering van de functionaliteit. Hierdoor blijft de API consistent met de documentatie en kan de implementatie makkelijk onderhouden en uitgebreid worden.

```
1         @Override
2         @Secured("ROLE_USER")
3         public ResponseEntity<Void>
              charactersGeneratePost(GenerateCharacterRequest
              generateCharacterRequest) {
4             characterSer-
                  vice.generateCharacter(characterMapper.toCharacterGenerateInfo(gen
5             return new ResponseEntity<>(HttpStatus.CREATED);
6
7         }
```

Codefragment 5.1: Voorbeeld van de generateCharacter route in de CharacterController

```
1         @Mapper(componentModel = "spring")
2         public interface CharacterMapper {
3             CharacterResponse toCharacterResponse(Character
                  character);
4             CharacterGenerateInfo
                  toCharacterGenerateInfo(GenerateCharacterRequest
                  generateCharacterRequest);
5         }
```

Codefragment 5.2: Voorbeeld van een MapStruct interface voor het mappen tussen DTO's en domeinmodellen

5.0.2. DTO-transformatie met MapStruct in de controllerlaag

Om de communicatie tussen de API en de interne domeinlogica te stroomlijnen, worden Data Transfer Objects (DTO's) gebruikt. Deze DTO's zijn de request- en responseklassen die automatisch gegenereerd zijn op basis van de OpenAPI-specificatie. Voor het omzetten van deze DTO's naar de domeinmodellen (en omgekeerd) wordt MapStruct ingezet. MapStruct genereert tijdens compile-tijd efficiënte en typeveilige mappers op basis van interface-definities, wat handmatig schrijfwerk vermindert en fouten voorkomt.
Binnen de controllerlaag wordt MapStruct gebruikt om inkomende request-DTO's te converteren naar domeinobjecten die vervolgens aan de servicelaag worden doorgegeven. Ook worden domeinobjecten terug omgezet naar response-DTO's om aan de client te worden geretourneerd.
Deze aanpak zorgt voor een duidelijke scheiding tussen API-modellen en domeinlogica, terwijl de code toch overzichtelijk en onderhoudbaar blijft.

5.0.3. Gebruik van Jakarta en Lombok in domeinmodellen

Voor het modelleren van de domeinklassen is gebruikgemaakt van Jakarta EE-annotaties en Lombok.
De Jakarta-annotaties, zoals `@Entity`, `@Id` en andere, worden toegepast om de domeinmodellen te verbinden met de onderliggende database en om validatieregels te definiëren. Dit maakt het mogelijk om de persistentie en validatie centraal in het domein te beheren.
Lombok wordt ingezet om boilerplate-code te verminderen. Met annotaties zoals `@Getter`, `@Setter`, `@NoArgsConstructor` en `@AllArgsConstructor` worden automatisch de getters, setters, constructors en andere veelvoorkomende methoden gegenereerd tijdens compile-tijd. Dit zorgt voor een veel compactere en beter leesbare codebasis.
Door het gebruik van deze tools blijft de code overzichtelijk en wordt de ontwikkeling versneld, terwijl de functionaliteit en de structuur van het domein behouden blijven.

5.0.4. Servicelaag

De servicelaag vormt de kern van de applicatielogica en fungeert als tussenlaag tussen de controllerlaag en de domeinmodellen. Deze laag bevat de businessregels en verwerkt de logica die nodig is om de functionaliteiten van de API te realiseren.
In deze laag worden de domeinmodellen gemanipuleerd en worden complexe bewerkingen uitgevoerd, zoals het valideren van gegevens, uitvoeren van berekeningen, en coördineren van verschillende onderdelen van de applicatie.
Door de businesslogica in een aparte servicelaag onder te brengen, blijft de controllerlaag lichtgewicht en gericht op het afhandelen van HTTP-verzoeken en -antwoorden. Dit verhoogt de modulariteit en onderhoudbaarheid van de code.
Deze gelaagde architectuur zorgt ervoor dat de applicatie overzichtelijk blijft en dat wijzigingen in de businesslogica beperkt blijven tot de servicelaag, zonder dat de controller- of domeinlaag aangepast hoeft te worden.

5.0.5. Repositorielaag

De repositorielaag is verantwoordelijk voor de communicatie met de database en het uitvoeren van CRUD-operaties (Create, Read, Update, Delete) op de domeinmodellen. Hiervoor is gebruikgemaakt van Java Persistence API (JPA), een standaard voor object-relational mapping in Java.
Met JPA kunnen domeinobjecten eenvoudig worden opgeslagen, opgehaald en beheerd in een relationele database. Door gebruik te maken van Spring Data JPA zijn de repositoryinterfaces snel en efficiënt opgezet, waarbij standaard methoden zoals `save()`, `findById()` en `delete()` automatisch worden gegenereerd.
Deze laag zorgt ervoor dat de servicelaag zich kan richten op de businesslogica,

```
@Entity
@Table(name = "`Character`")
@Builder(toBuilder = true)
@NoArgsConstructor
@AllArgsConstructor
@Data
public class Character {
    private static final int STARTING_LEVEL = 1;
    private static final int STARTING_XP = 0;

    @Id
    @JdbcTypeCode(SqlTypes.CHAR)
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;
    private String name;
    private String race;
    private String characterClass;
    @Builder.Default
    private int level = STARTING_LEVEL;
    private String background;
    private String alignment;
    @Builder.Default
    private int xp = STARTING_XP;
    private Abilities abilities;
    private Skills skills;


}
```

Codefragment 5.3: Een fragment van de character domein classe

```
1         public List<Character> getCharacters() {
2             User user = authUtil.getCurrentUser();
3             return user.getCharacters();
4         }
5
6         public Character getCharacter(UUID characterId) {
7             User user = authUtil.getCurrentUser();
8
9             return characterRepository.findById(characterId)
10            .filter(character ->
                  user.getCharacters().contains(character))
11            .orElseThrow(() -> new AccessDeniedException("Character
                  not found or does not belong to user"));
12        }
```

Codefragment 5.4: Een fragment van de characterService

```
1         public interface CharacterRepository extends
              JpaRepository<Character, UUID> {
2         }
```

Codefragment 5.5: De CharacterRepository

terwijl de details van data-opslag en databasecommunicatie worden afgehandeld door de repositorielaag. Dit draagt bij aan een duidelijke scheiding van verantwoordelijkheden binnen de applicatie.

5.0.6. Databasebeheer met Liquibase

Voor het beheer van de database en het uitvoeren van schemawijzigingen is gebruikgemaakt van Liquibase. Liquibase is een open-source tool die het mogelijk maakt om databaseversies en migraties op een gestructureerde en herhaalbare manier te beheren.
Met behulp van changelog-bestanden, geschreven in YAML, worden wijzigingen in de database beschreven. Deze bestanden worden door Liquibase gelezen en toegepast, waardoor het opzetten, wijzigen en bijwerken van het databaseschema geautomatiseerd en gecontroleerd verloopt.
Door Liquibase te integreren in het ontwikkelproces, kan het databaseschema consistent worden gehouden tussen verschillende ontwikkelomgevingen en productie, wat fouten en inconsistenties voorkomt.
Deze aanpak maakt het mogelijk om databasewijzigingen veilig en traceerbaar door te voeren, wat de betrouwbaarheid en onderhoudbaarheid van het systeem ten goede komt.

```
databaseChangeLog:
- changeSet:
id: create-feat-table
author: Thomas Gielen
changes:
- createTable:
tableName: feat
columns:
- column:
name: id
type: UUID
constraints:
primaryKey: true
- column:
name: name
type: VARCHAR(255)
constraints:
nullable: false
- column:
name: description
type: VARCHAR(2000)
constraints:
nullable: true

```

Codefragment 5.6: Een fragment van een Liquibase changelog bestand

6

Conclusie

Deze graduaatsproef richtte zich op het ontwikkelen van een RESTful API voor het aanmaken en beheren van Dungeons & Dragons-personages met behulp van het Spring Boot framework en een OpenAPI-gedreven aanpak. Het doel was om een efficiënte, schaalbare en onderhoudbare API te realiseren met moderne technologieën.

Door gebruik te maken van Apidog voor het opstellen van de OpenAPI-specificatie en het automatisch genereren van request- en responseklassen, kon een consistente en gestructureerde API worden gebouwd. Dit verminderde implementatiefouten en zorgde voor een goede afstemming tussen documentatie en code. De implementatie van controllers, services en repositories volgde een duidelijke scheiding van verantwoordelijkheden, wat de onderhoudbaarheid en uitbreidbaarheid bevordert.

MapStruct werd ingezet om DTO's om te zetten naar domeinmodellen, wat het ontwikkelproces vereenvoudigde en de leesbaarheid verbeterde. Voor data-opslag werd JPA gebruikt en Liquibase zorgde voor gecontroleerd databasebeheer via versiebeheer en migraties, wat de betrouwbaarheid verhoogde.

Het resultaat is een werkende API die voldoet aan de functionele eisen, inclusief gebruikersauthenticatie en ondersteuning voor diverse personageklassen en rassen. De modulaire opbouw en het gebruik van gangbare technologieën maken toekomstige uitbreidingen eenvoudiger.

Er zijn echter ook beperkingen vastgesteld. Zo ontbreken uitgebreide performancetests, wat nodig is om de schaalbaarheid te garanderen.

De resultaten kwamen grotendeels overeen met de verwachtingen, maar er ontstonden ook nieuwe vragen, zoals hoe de API beter kan omgaan met hoge belasting en hoe integratie met andere Dungeons & Dragons-tools kan plaatsvinden.

Kortom, deze graduaatsproef leverde een concrete en toepasbare oplossing die bijdraagt aan het vakgebied door een praktische toepassing van moderne Java-

technologieën en een OpenAPI-gedreven ontwikkelproces. Voor toekomstig werk is er ruimte voor een front-end voor deze back-end te ontwikkelen en veredere uitbrijdingen voor de back-en te schrijven zoals bevoorbeeld het moegelijk te maken om de characters te laten levelen.