**Ontwerpdocument BE**

|  |  |
| --- | --- |
| **Productnaam** | Favorite Books Backend |
| **Studentnummer en naam** | 20220545  Eleonora Avaleva |
| **Groep** | Groep 5 |
| **GitHub Repository** | <https://github.com/orgs/School-related-projects/repositories>  <https://github.com/School-related-projects/one-fav-books>  <https://github.com/School-related-projects/two-fav-books-be> |

Inhoudsopgave

[Inleiding 2](#_Toc193568629)

[Projectplan 2](#_Toc193568630)

[Ontwerp backend 3](#_Toc193568631)

[Code review 6](#_Toc193568632)

[Testplan 8](#_Toc193568633)

[Reflectie en Conclusies 9](#_Toc193568634)

[Bronnen 11](#_Toc193568635)

# Inleiding

De applicatie “My Favorite Books” is een full-stack applicatie die een REST-API backend koppelt aan de bestaande frontend uit sprint 1. Deze applicatie stelt gebruikers in staat om hun persoonlijke boekencollectie te behalen en te bekijken. De backend implementatie maakt het mogelijk om boeken toe te voegen, te wijzigen, te verwijderen en lezen via een API-interface. Dit zorgt voor een persistente dataverwerking.

**Vereisten**

* Implementatie van een REST-API voor CRUD-operaties op de boekencollectie
* Ontwikkeling van een Express.js backend die de API-endpoints implementeert
* JSON-server voor het opslaan en beheren van de boekengegevens
* API-documentatie
* Postman collectie voor het testen van de API-endpoints
* Backend integratie met de frontend uit sprint 1
* CORS configureren om communicatie tussen frontend en backend mogelijk te maken

**Optionele functies**

* Swagger-specificatie voor API-documentatie
* Implementatie van Object Relational Mapper(ORM) met Sequelize.js voor databasebeheer
* Deployment van de backen applicatie in Azure Cloud
* Geavanceerde foutafhandeling met specifieke HTTP-statuscodes en foutmeldingen
* Implementatie van authenticatie en autorisatie voor beveiligde toegang tot de API

# Projectplan

Voor de ontwikkeling van het backend gedeelte van de "My Favorite Books" applicatie heb ik een gestructureerde aanpak gehanteerd om ervoor te zorgen dat alle vereisten effectief werden geïmplementeerd. Het project is opgedeeld in verschillende fasen, elk met specifieke doelstellingen en deliverables. Dit heeft me geholpen om de voortgang te bewaken en de kwaliteit van het eindproduct te waarborgen.

De volgende stappen zijn gevolgd bij het uitvoeren van deze opdracht, waarbij elke fase logisch voortbouwt op de vorige. Door deze methodische aanpak kon ik complexe problemen opsplitsen in beheersbare taken en kon ik beslissingen nemen op basis van een duidelijk gedefinieerde structuur. Deze werkwijze stelde me in staat om de opdracht efficiënt en met aandacht voor detail te voltooien.

**Stap 1: Analyse en planning van de API-structuur (Week 1)**

* Bestuderen van HTTP-protocol en request/response cycle
* Onderzoek naar RESTful API en JSON-communicatie
* Analyseren van frontend opdracht 1 voor integratiepunten
* Node.js en overig tools heb ik al op mijn machine staan dus, alleen controle uitvoeren

**Stap 2: Ontwerpen van de API en documentatie (Week 1/2)**

* JSON-server configureren
* Ontwerpen van API-endoints
* Documenteren van API-specificaties

**Stap 4: Implementatie van de API met Express.js (Week 2)**

* Opzetten van Express.js applicatiestructuur
* Implementeren van routes en controllers
* Configureren van CORS voor cross-domain communicatie
* Ontwikkelen en uitvoeren van Postman testscenario's

**Stap 5: (Optioneel) Implementatie van de Express.js backend**

* Ontwerpen van database schema voor boekgegevens
* Implementeren van Sequelize.js ORM (optioneel)
* Migratie van data van JSON-server naar relationele database
* Testen van database interacties
* Implementeren van error handling in frontend

Ik heb besloten om eerst het API-ontwerp te documenteren voordat ik met de implementatie begon. Dit heeft mij een duidelijk begrip van de vereisten gegeven en voorkwam dat ik veranderingen in de architectuur moest maken tijdens latere fasen. Mijn keuze voor JSON-server in de beginfase maakte snelle iteratie mogelijk, terwijl mijn overstap naar een volledig Express.js back-end mij de nodige flexibiliteit bood voor toekomstige uitbreidingen.

Ik koos voor Postman als testtool, wat zorgde voor grondige validatie van alle API-endpoints. Hierdoor kon ik bugs vroeg identificeren en oplossen. Mijn stapsgewijze integratie met de frontend resulteerde in een soepele overgang van hard-gecodeeerde data naar dynamische API-interacties.

Door mijn gestructureerde werkwijze heb ik alle vereisten effectief kunnen implementeren en een robuuste, goed gedocumenteerde API-backend kunnen opleveren die naadloos integreert met de bestaande frontend.

# Ontwerp backend

Deze API is ontworpen voor het beheren van een persoonlijke collectie favoriete boeken. Het stelt gebruikers in staat om hun leeslijst bij te houden, boeken toe te voegen, details aan te passen en boeken uit de collectie te verwijderen.

De API biedt volledig CRUD-functionaliteit voor boekengegevens. De gebruiker kan alle boeken ophalen, specifieke boek vinden op basis van id, nieuwe boek toevoegen aan de collectie, de gegevens van een boek aanpassen, en boek verwijderen.

**Technische implementatie**

JSON-server wordt als persistente dataopslag gebruikt voor de API’s. Dit is een goede oplossing voor prototyping van kleine project als deze. De gegevens worden opgeslagen in een JSON-bestand dat optreedt als een eenvoudige database, waarbij geen rekening wordt gehouden met concepten zoals relationele database model of normaliseren van database tabellen.

Voordelen zijn:

* Snelle setup zonde zonder tijd te veel tijd in het ontwerpen van relationele database of het normaliseren van tabellen.
* Ideaal voor frontend-ontwikkeling en prototyping

Hieronder wordt de API-documentatie in een tabel vorm weergegeven.

|  |  |
| --- | --- |
| **Section** | **Description** |
| Description | This API manages a collection of favorite books |
| Authentication | None required |
| Base URL | <http://localhost:8000/api> |
| Endpoints | - /api/books: Get all books (GET), add a new book (POST)  - /api/books/:id: Get specific book (GET), update a book (PUT), delete a book (DELETE) |
| Resource model | - books: id, title, author, year, genre, summary, cover |
| Request Method | **GET, POST, PUT, DELETE** |
| Parameters | - /api/books (GET): None - /api/books (POST): Book data in the request body  - /api/books/:id (GET): id path parameter  - /api/books/:id (PUT): id path parameter and updated fields in request body  - /api/books/:id (DELETE): id path parameter |
| Response Format | JSON |
| Error Handling | - 200 OK: Successful response  - 201 Created: Book successfully created  - 400 Bad Request: Invalid request data  - 404 Not Found: Book not found  - 500 Internal Server Error: Server issue |
| Example Requests/Responses | - **Request**: GET /api/books  - **Response**:  [{   "id":"167",  "title":"Why The Caged Bird Sings",  "author":"Maya Angelou",  "year":1969,  "genre":"Biography"  }] - Request: GET /api/books/13 - Response:   {  "id":"13",  "title":"To Kill a Mockingbird",  "author":"Harper Lee"  } - Request: POST /api/books - Request Body:  {  "title":"The Great Gatsby",  "author":"F. Scott Fitzgerald"  } - Response:  {  "id":"1678954323","title":"The Great Gatsby"  } - Request: PUT /api/books/:id  - Request Body:  {  "genre":"Psychological Fiction"  }  - Response: Updated book object  - Request: DELETE /api/books/:id  - Response: {"message":"Book deleted successfully"} |

# Code review

Een essentieel onderdeel van het ontwikkelproces was de code review fase, waarin mijn code kritisch werd beoordeeld door medestudenten en waarin ik ook feedback heb gegeven op het werk van anderen. Deze wederzijdse evaluatie heeft niet alleen geholpen om technische problemen op te sporen, maar heeft ook bijgedragen aan het verbeteren van de algehele kwaliteit en functionaliteit van de applicatie een samenvatting van de belangrijkste bevindingen, aanbevelingen en beslissingen.

**Uitgevoerde Code Review door medestudent**

De code review is uitgevoerd in samenwerking met twee medestudent, Ensar, tijdens een peer-review sessie. We hebben elkaars code systematisch doorgenomen met focus op functionaliteit, onderhoudbaarheid en gebruiksvriendelijkheid. De review omvatte zowel de API-architectuur als de JavaScript functionaliteit.

A screenshot of a computer

Description automatically generated

**Gegeven feedback**

|  |
| --- |
| Peerfeedback formulier voor programmeeropdracht |
| Naam student die beoordeeld wordt: Ensar Isiktas |
| Naam beoordelaar: Eleonora Avaleva |
| Datum: 22-03-2025 |
| Onderwerp: Review code backend API-architectuur |

|  |  |
| --- | --- |
| Criteria | Opmerkingen |
| Code Efficiëntie | Helaas heb ik hier onvoldoende kennis om te bepalen of de code efficienter kan. Wel loop je risico dat een onverwachtte type coersion kunt krijgen door het gebruikt te maken van alleen 2 gelijktekens (==). Gebruik in plaats daarvan 3 gelijktekens (===).  De volgende functie (line 485, script)verwacht 1 parameter maar je geeft het 2 mee.  Weet je zeker of het dan nog steeds werkt?  *document*.querySelectorAll('.sort-btn')?.forEach(btn => {  btn.addEventListener('click', () => sortCollection(btn.dataset.field, *currentSort*.ascending)); }); |
| Leesbaarheid en Stijl | Het zal leesbaarder zijn als je de frontend en de backend code in twee verschillende directories zet.  De functies zijn kort en duidelijk, wel kun je ze verdelen over aparte files (met oog op toekomstige uitbreiding van de applicatie) |
| Functionaliteit | De applicatie lijkt te werken met de server APIs. |
| Modulariteit en Herbruikbaarheid | Je zal kunnen overwegen om je frontend code op te splitsen naar, functies die data ophalen, filters en handlers. |
| Documentatie en Commentaren | Er zijn weinig tot geen comments in de code toegevoegd. Wel is de naamgeving van de functies goed, dit maakt dat ik de code makkelijk leesbaar vind. |
| Foutafhandeling en Robuustheid | Je maakt gebruik van statuscodes 400, 404 en 201 aan de server kant. Dat vind ik netjes. Wel kun je nog status 500 (server issue), 200 (request succes) er aan toe voegen. |

|  |
| --- |
| Algemene Opmerkingen: |
| Ik vind het een leuke oplossing! Het is interessant om te zien dat je met zo weinig code toch een mooie en werkende applicatie hebt gemaakt. |

# Testplan

Hier wordt informatie opgenomen over het testen van mijn API's en back-end applicatie. Voor mijn API's heb ik Postman gebruikt om de functionaliteit systematisch te testen. Hieronder volgt een lijst met kritische tests die ik heb uitgevoerd, gerangschikt van meest naar minst belangrijk:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test-nummer** | **Testdoel** | **Beschrijving** | **Resultaat  (geslacht /mislukt)** | **Toelichting** |
| 1 | GET all books | Testen of alle boeken correct worden opgehaald | Geslaagd | API retourneerde correct alle boeken met status 200 |
| 2 | GET book by ID | Controleren of een specifiek boek kan worden opgehaald op basis van ID | Geslaagd | Boek met ID 3 werd correct geretourneerd met alle velden |
| 3 | POST new book | Testen of een nieuw boek correct kan worden toegevoegd | Geslaagd | Nieuw boek werd succesvol toegevoegd met status 201 en uniek ID |
| 4 | PUT update book | Controleren of een bestaand boek kan worden bijgewerkt | Geslaagd | Genre van boek met ID 2 werd succesvol bijgewerkt |
| 5 | DELETE book | Testen of een boek correct kan worden verwijderd | Geslaagd | Boek met ID 7 werd succesvol verwijderd met status 200 |
| 6 | GET non-existent book | Testen van foutafhandeling bij opvragen niet-bestaand ID | Niet geslaagd | API retourneerde status 200 met leeg object in plaats van 404 |
| 7 | POST invalid book data | Testen of validatie correct werkt bij ontbrekende verplichte velden | Niet geslaagd | API accepteerde boek zonder titel en auteur, status 201 in plaats van 400 |
| 8 | PUT with invalid ID | Bijwerken van boek met niet-bestaand ID | Niet geslaagd | Operatie resulteerde in aanmaken nieuw boek in plaats van foutmelding |

# Reflectie en Conclusies

**Afgeronde en niet-afgeronde eisen**

**Succesvol afgeronde eisen:**

* **REST-API implementatie**: Ik heb een volledige REST-API geïmplementeerd die CRUD-operaties ondersteunt voor de boekencollectie. De endpoints zijn correct gedefinieerd en gedocumenteerd.
* **Express.js backend**: De backend is ontwikkeld met Express.js, wat een flexibel framework biedt voor het implementeren van de API-endpoints.
* **JSON-server**: Ik heb JSON-server succesvol geconfigureerd voor het opslaan en beheren van de boekgegevens, wat een goede oplossing is voor prototyping.
* **API-documentatie**: De API is uitgebreid gedocumenteerd, inclusief beschrijvingen van resources, parameters en voorbeelden.
* **Postman collectie**: Ik heb een volledige Postman collectie ontwikkeld voor het testen van alle API-endpoints.
* **CORS-configuratie**: De CORS is geconfigureerd om communicatie tussen frontend en backend mogelijk te maken.

**Niet-afgeronde eisen (optioneel):**

* **Azure publicatie**: Door problemen met mijn Microsoft Student licentie account was ik niet in staat om resources aan te maken en de oplossing te publiceren in Azure.
* **ORM-implementatie**: De implementatie van Object Relational Mapping met Sequelize.js is niet volledig afgerond, omdat ik prioriteit heb gegeven aan de kernfunctionaliteit van de API.
* **Swagger-specificatie**: Hoewel de API is gedocumenteerd, heb ik geen volledige Swagger/OpenAPI-specificatie ontwikkeld.
* **CORS-configuratie:** Gezien ik geen Azure-publicatie heb uitgevoerd, is op dit moment geen juiste referentie gemaakt naar de frontend code.

**Effectiviteit van het stappenplan**

Mijn gekozen stappenplan bleek effectief voor dit project. De methodische aanpak, waarbij ik eerst de API ontwierp en documenteerde voordat ik met de implementatie begon, heeft bijgedragen aan een duidelijk begrip van de vereisten. Dit voorkwam dat ik grote wijzigingen moest aanbrengen tijdens latere fasen van het project.

Het werken met JSON-server in de beginfase was een verstandige keuze. Het stelde me in staat om snel te itereren en de API te testen zonder tijd te besteden aan het opzetten van een complexe databasestructuur. De overstap naar Express.js voor de volledige implementatie gaf me de flexibiliteit die nodig was voor toekomstige uitbreidingen.

Als ik het project opnieuw zou doen, zou ik meer aandacht besteden aan de optionele functies, vooral de Swagger-specificatie, omdat dit de documentatie nog toegankelijker zou maken.

**Uitdagingen en leermomenten**

Tijdens dit project heb ik verschillende uitdagingen ervaren:

**Azure deployment problemen**: De problemen met mijn Microsoft Student licentie waren frustrerend, maar hebben me geleerd om alternatieven te overwegen en niet afhankelijk te zijn van één specifiek platform voor deployment.

**Inputvalidatie**: De niet-geslaagde test voor inputvalidatie liet zien dat ik meer aandacht had moeten besteden aan het valideren van gebruikersinvoer. In toekomstige projecten zal ik een bibliotheek zoals express-validator middleware gebruiken om invoer te valideren voordat het wordt verwerkt.

Een belangrijk leermoment was het inzicht dat goede documentatie en testing cruciaal zijn voor de ontwikkeling van een API. Door tijd te investeren in het ontwerpen en documenteren van de API voordat ik begon met coderen, kon ik efficiënter werken en een beter eindproduct leveren.

Ik heb ook geleerd hoe krachtig Express.js is voor het ontwikkelen van RESTful API's en hoe je middleware kunt gebruiken om veel voorkomende functionaliteit te implementeren, zoals logging, authenticatie en CORS-configuratie.

**Conclusie**

Dit project heeft me een waardevolle ervaring gegeven in het ontwikkelen van een volledige backend API die communiceert met een bestaande frontend. Door een gestructureerde aanpak te volgen en prioriteit te geven aan de kernfunctionaliteit, heb ik een robuuste en goed gedocumenteerde API kunnen ontwikkelen.

Hoewel niet alle optionele functies zijn geïmplementeerd, voldoet de applicatie aan alle kerneisen en biedt het een solide basis voor toekomstige uitbreidingen. De leermomenten en uitdagingen die ik heb ervaren, hebben mijn begrip van backend-ontwikkeling en API-ontwerp aanzienlijk verbeterd.

# Bronnen

Naast de hieronder genoemde referenties, heb ik ook alle door school gegeven literatuur doorgenomen.

Express.js Documentatie. (2023). Opgehaald uit <https://expressjs.com/docs>

JSON Server. (2023). GitHub repository. Opgehaald uit <https://github.com/typicode/json-server>

Mozilla Developer Network. (2023). HTTP Documentatie. Opgehaald uit <https://developer.mozilla.org/en-US/docs/Web/HTTP>

Postman Learning Center. (2023). Writing Tests in Postman. Opgehaald uit <https://learning.postman.com/docs/writing-scripts/test-scripts/>

RESTful API Design Best Practices. (2023). Opgehaald uit <https://restfulapi.net/>

Richardson, L., & Ruby, S. (2007). RESTful Web Services. O'Reilly Media.

Sequelize Documentatie. (2023). Opgehaald uit <https://sequelize.org/>

Express-validator Documentatie. (2024). Opgehaald uit <https://express-validator.github.io/docs/>

Html-modal Documentatie (2025). Opgehaald uit <https://www.w3schools.com/w3css/w3css_modal.asp>

ORM Artikel (2022). Opgehaald uit <https://ryan-m-schleck.medium.com/object-relational-mapping-in-express-ce7a432c5f9>

ChatGPT voor sneller genereren van Swagger schema: <https://chatgpt.com/>