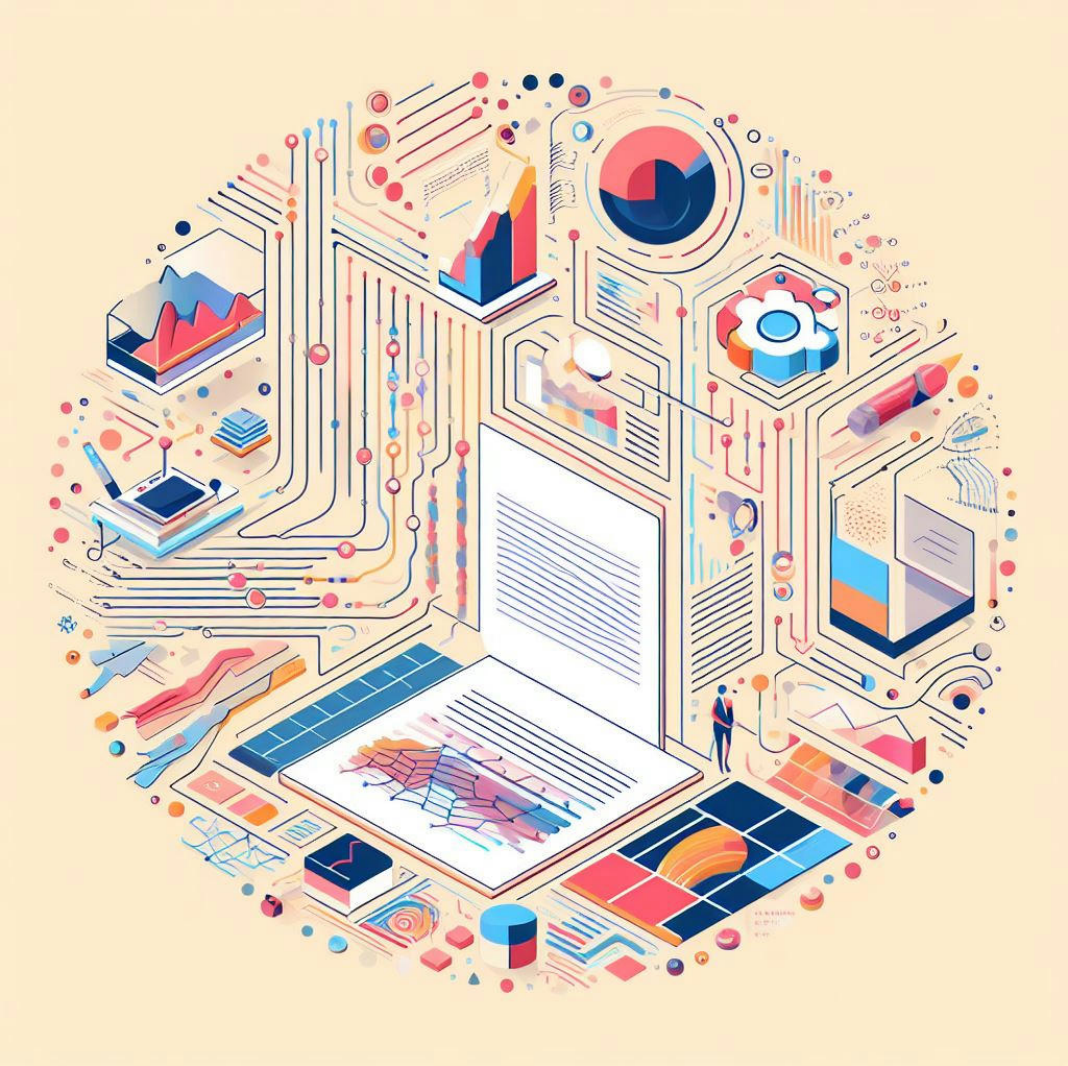
Competentie: Realiseren

Implementatierapport en uitvoering



Student: Levi Leuwol
Studentnummer: 405538
Datum: 2023-06-30
Onderwerp: Competentie: Analyseren
Opleiding: HBO-ICT
Studiejaar: 3

Hanzehogeschool Groningen
University of Applied Sciences

Inhoudsopgave

1 Samenvatting

Tijdens de stage is een datavisualisatietool voor Tenwise ontwikkeld. Het project voorziet in een webinterface voor de analyse van voedselrecepten. De architectuur is gebaseerd op een React frontend en een Symfony backend, geïntegreerd in Docker containers voor eenvoudige schaalbaarheid en implementatie.

In de frontend wordt gebruik gemaakt van React voor herbruikbare en efficiënte componenten, wat een vloeiende gebruikerservaring oplevert. De Symfony-backend biedt een robuuste en flexibele oplossing voor webontwikkeling.

De stappen voor implementatie omvatten het vaststellen van doelen en problemen, verzamelen van requirements, ontwikkeling en advisering van het ontwerp en de daadwerkelijke realisatie. Implementatie is voltooid in Q2, 2023. De student heeft zich gefocust op het ontwikkelen van de backend en optimalisatie.

Voor succes zijn een stabiele API, een efficiënte database en een schaalbare architectuur vereist. Mogelijke risico's zijn een gebrek aan kennis van de gebruikte technologieën en een gebrek aan tijd voor implementatie. De student heeft zich ingezet om deze risico's te beperken door het volgen van cursussen en het opstellen van een planning.

Dit rapport beschrijft de implementatie van de applicatie en de stappen die zijn genomen om de applicatie te ontwikkelen. Het rapport is bedoeld voor de opdrachtgever, de begeleiders, de student en toekomstige ontwikkelaars van de applicatie.

2 Inleiding

2.1 Achtergrondonderzoek

Één van de belangrijkste onderdelen van het project is de dataoverdracht tussen de frontend en de backend. De student heeft onderzoek gedaan naar de verschillende methoden voor dataoverdracht en de voor- en nadelen van elke methode.

Er is gekeken naar de performance van het ophalen van nodes in Neo4j met een Object Graph Mapper (OGM) en de standaard Neo4j-driver. Via tijdsmetingen en statistische analyse is geconcludeerd dat de OGM, in dit geval, een betere performance biedt.

2.2 Doel, resultaten en afbakening

Het doel van de implementatie is het vinden van de meest efficiënte manier om met de data om te gaan en het ontwikkelen van een stabiele en schaalbare architectuur. Het resultaat is een snellere en efficiëntere applicatie die eenvoudig kan worden uitgebreid. Dit implementatierapport is gelimitteerd tot de backend van de applicatie.

2.3 Randvoorwaarden

De volgende randvoorwaarden zijn gesteld voor de implementatie:

- **Beschikbaarheid van vakkennis**: De organisatie heeft voldoende kennis van de gebruikte technologieën.
- **Beschikbaarheid van medewerkers**: De organisatie heeft voldoende medewerkers met kennis van de gebruikte database, backend en frontend technologieën.
- **Beschikbaarheid van middelen:** De organisatie heeft voldoende middelen om de applicatie te hosten en te onderhouden.
- **Commitment van het management:** Het management is bereid om de applicatie te onderhouden en uit te breiden.

2.4 Overzicht van de structuur

1. **Samenvatting:** Dit deel biedt een beknopt overzicht van de belangrijkste punten van het rapport.
2. **Inleiding:** Hier wordt de context en het doel van het rapport uiteengezet.
3. **Implementatiestappen:** Dit hoofdstuk is opgedeeld in verschillende subsecties zoals voorbereiding, realisatie en de verschillende componenten van de applicatie zoals opstelling, configuratie, controllers, services en entiteiten.
4. **Communicatieplan:** Dit deel beschrijft hoe de communicatie tussen alle betrokken partijen wordt beheerd.
5. **Beheersplan:** Hier wordt het plan voor het effectief beheren van de applicatie gedurende haar levenscyclus uiteengezet.
6. **Conclusie:** Dit hoofdstuk biedt een samenvatting van de belangrijkste bevindingen en resultaten van het rapport.
7. **Bronnen:** Hier worden alle referenties en bronnen vermeld die zijn gebruikt tijdens het schrijven van het rapport.

3 Implementatiestappen

3.1 Voorbereiding

Voordat de implementatie kan beginnen zijn er een aantal stappen die moeten worden uitgevoerd. Er moet een serverruimte worden ingericht en de benodigde software moet worden geïnstalleerd. Testomgevingen moeten worden opgezet om de functionaliteit van de applicatie te testen. Tenslotte moet er een planning worden opgesteld om de voortgang van de implementatie te bewaken.

De stappenplan voor de realisatie is als volgt:

1. **SSH Verbinding maken met je VPS**
   - Gebruik een SSH-client om verbinding te maken met je VPS.
2. **Docker, Docker Compose en Git Installeren**
   - Update eerst je pakketlijst: `sudo apt update` (Debian-based distro's)
   - Installeer Docker: `sudo apt install docker`
   - Installeer Docker Compose: `sudo apt install docker-compose`
3. **Kopieer Bestanden van Github**
   - Gebruik `git` of een andere methode om de bestanden naar de VPS te kopiëren.
4. **Composer en NPM Stages Bouwen**
   - Voer `docker build --pull` uit om de containers te bouwen.
5. **Docker Compose Up**
   - Ga naar de directory waar je `docker-compose.yml` staat.
   - Voer `docker-compose up -d` uit om de containers te bouwen en te starten.
6. **Environment File**
   - Zorg ervoor dat je een `.env` file hebt met de benodigde environment variabelen. Plaats deze in dezelfde directory als je `docker-compose.yml`.
7. **Frontend (Native)**
   - De frontend draait op dit moment native. Dit zal in de toekomst worden gecontaineriseerd.
8. **Toegang**
   - Nu zou de API toegankelijk moeten zijn via `http://<je-VPS-ip>:9000` en Neo4j via `http://<je-VPS-ip>:7474`.
9. **Logs & Debugging**
   - Gebruik `docker logs <container_naam>` voor logs.
10. **Stop & Verwijder Containers**
    - Gebruik `docker-compose down` als je de containers wilt stoppen en verwijderen.

De opzet van de test- en productieomgeving is hetzelfde, met uitzondering van de environment variabelen. De testomgeving gebruikt een andere Neo4j-database en de productieomgeving gebruikt een andere Neo4j-database en een andere API-URL.

3.2 Realisatie

De realisatie van de applicatie bestaat uit het ontwikkelen van de backend en het optimalisatieproces.

3.2.1 Opstelling

De opstelling van de backend server maakt gebruik van Docker en docker-compose om een geïsoleerde omgeving te creëren waarin de Symfony-applicatie en Neo4j-database kunnen draaien. De `Dockerfile` bevat instructies om de Docker-image voor de Symfony-applicatie te bouwen. De `docker-compose.yml` bevat instructies om de containers te bouwen en bevat instructies hoe deze containers met andere services, zoals de Neo4j-database, samenwerken.

Dockerfile

- **Stages**:
  - **Composer Stage**: Gebruikt het `composer:latest` image om de PHP-afhankelijkheden te installeren.
  - **NPM Stage**: Gebruikt het `node:latest` image om de Node.js-afhankelijkheden te installeren.
  - **PHP Stage** Gebruikt het `php:latest` image als het uiteindelijke image.
- **Commando's**:
  - `composer install`: Installeert PHP-afhankelijkheden.
  - `npm install`: Installeert Node.js-afhankelijkheden.
  - `apt-get update`: Update de pakketlijst in het PHP-image.

docker-compose.yml

- **Services**:
  - `app`: De Symfony-applicatie.
    - Build: Vanuit de huidige directory.
    - Ports: 9000:9000
    - Command: Start de PHP-server en voert npm run watch uit.
  - `db`: De Neo4j-database.
    - Image: neo4j:latest
    - Ports: 7474:7474, 7687:7687
    - Volumes: Data en configuratiebestanden.
- **Volumes**:
  - `db`: Gedefinieerd voor het bewaren van Neo4j-data. De container is `delegated`, wat betekent dat de host in read-only mode is en niet direct kan schrijven naar het volume. Dit is een veiligheidsmaatregel om te voorkomen dat de host de database beschadigt.

Deze Docker-configuratie maakt het gemakkelijk om de hele stack (Symfony-app en Neo4j-database) lokaal te draaien voor ontwikkeling. Het maakt gebruik van meerdere stages in de `Dockerfile` om de build te optimaliseren en omvat alle benodigde configuratie in de `docker-compose.yml`.

3.2.1.1 Waarom Docker en Docker Compose?

Docker biedt een manier om de applicatie de afhankelijkheden in een geïsoleerde container te draaien. Dit maakt het gemakkelijker om de applicatie te verspreiden en uit te voeren op verschillende systemen. Docker Compose gaat nog een stapje verder door meerdere containers te orkestreren en te verbinden. Dit faciliteert het gebruik van meerdere services, bijvoorbeeld een webserver en een database, in één applicatie.

3.2.1.2 Voordelen

1. **Isolatie**: Docker containers zijn geïsoleerd van elkaar en van het host systeem. Dit betekent dat de applicatie in een gecontroleerde omgeving draait en dat de applicatie niet kan worden beïnvloed door andere applicaties of het host systeem.
2. **Draagbaarheid**: Docker containers zijn draagbaar. Dit betekent dat de applicatie op verschillende systemen kan worden uitgevoerd zonder dat er wijzigingen nodig zijn.
3. **Schaalbaarheid**: Docker containers zijn schaalbaar. Dit betekent dat de applicatie kan worden geschaald door meer containers toe te voegen.
4. **Versiebeheer**: Docker maakt het gemakkelijk om verschillende versies van de applicatie of database te beheren, wat handig is voor het testen van nieuwe functies.

Nadelen

1. **Complexiteit**: Docker containers zijn complexer dan traditionele applicaties. Dit betekent dat er meer kennis nodig is om de applicatie te beheren.
2. **Resourcegebruik**: Containers kunnen meer resources gebruiken dan traditionele applicaties.
3. **Leercurve**: Docker heeft een steile leercurve. Dit betekent dat het meer tijd kost om te leren hoe Docker werkt.

3.2.1.3 Conclusie

De opstelling van Docker en Docker Compose biedt een robuuste en flexibele omgeving voor de Symfony- en Neo4j-gebaseerde applicatie. Hoewel er enige complexiteit en overhead is, worden deze nadelen gecompenseerd door de voordelen op het gebied van isolatie, draagbaarheid en schaalbaarheid.

3.2.2 Configuratie

3.2.2.1 Configuratie in Symfony

De volgende configuratiebestanden zijn cruciaal voor het functioneren van de Symfony-applicatie. Deze bestanden bieden een gecentraliseerde plek om verschillende aspecten van de applicatie te beheren.

- `Bundles.php`:
  - Doel: Dit bestand bevat de lijst van alle Symfony-bundels die in de applicatie zijn geïnstalleerd.
  - Inhoud: Het bevat bundels zoals `FrameworkBundle`, `MakerBundle`, `TwigComponentBundle`, etc.
- `Framework.yaml`:
  - Doel: Basisconfiguratie voor de Symfony-applicatie.
  - Inhoud: Bevat instellingen zoals `secret`, `session`, `php_errors`, etc.
- `Routing.yaml`:
  - Doel: Configuratie voor de routering in de applicatie.
  - Inhoud: Bevat instellingen zoals `utf8: true` en `strict_requirements: null` voor productie.
- `Twig.yaml`:
  - Doel: Configuratie voor de Twig templating engine.
  - Inhoud: Bevat het pad naar de templates en instellingen voor testomgevingen.
- `Validator.yaml`:
  - Doel: Configuratie voor validatie van entiteiten in de applicatie.
  - Inhoud: Bevat instellingen zoals `email_validation_mode: html5`.
- `Routes.yaml`:
  - Doel: Definieert de routes voor de controllers.
  - Inhoud: Bevat de `resource` en `namespace` voor de controllers.
- `Services.yaml`:
  - Doel: Configuratie voor de services in de applicatie.
  - Inhoud: Bevat instellingen zoals `autowire`, `autoconfigure`, en `bind`.

3.2.2.2 Configuratie in Neo4j

Het Neo4j-configuratiebestand is te vinden in `./conf/neo4j.conf`. Dit bestand bevat de belangrijkste instellingen voor de Neo4j-database en de verbinding met de Symfony-applicatie.

- **Algemene Instellingen**:
  - `server.default_listen_address=0.0.0.0`: Hiermee kan de server verbindingen van elk IP-adres accepteren.
  - `dbms.security.auth_enabled=false`: Authenticatie is uitgeschakeld voor eenvoudige toegang. Dit moet worden ingeschakeld voor productie.
- **Geheugenbeheer**:
  - `server.memory.pagecache.size=1G`: Grootte van de pagina-cache.
  - `dbms.memory.heap.max_size=1G`: Maximale grootte van de Java heap.
- **Netwerkconnectoren**:
  - `server.bolt.enabled=true`: Bolt-connector is ingeschakeld.
  - `server.http.enabled=true`: HTTP-connector is ingeschakeld.
  - `server.https.enabled=false`: HTTPS is uitgeschakeld.
- **Logging**:
  - `server.directories.logs=/logs`: Pad naar de logbestanden.
- **Overig**:
  - `db.tx_log.rotation.retention_policy=100M size`: Retentiebeleid voor transactielogboeken.

Dit bestand biedt een basis voor de Neo4j-database en maakt het mogelijk om eenvoudig aanpassingen te doen voor specifieke behoeften.

3.2.3 Controllers

3.2.3.1 Algemene Architectuur

De Symfony Server maakt gebruik van het MVC (Model-View-Controller) model, wat een goede scheiding van verantwoordelijkheden biedt. De controllers zijn verantwoordelijk voor het ontvangen van de HTTP requests en het doorsturen van de requests naar de juiste services. De services zijn verantwoordelijk voor het verwerken van de requests en het terugsturen van de response.

3.2.3.2 Controllers en Endpoints

- **IndexController**: Deze controller is verantwoordelijk voor het renderen van de indexpagina. Het maakt gebruik van de `OGMClient` service om gegevens op te halen.
  Endpoints:
  1. `GET /`: Deze route haalt alle entiteiten op uit de Neo4j-database en rendert ze op de hoofdpagina. Het maakt gebruik van de OGMClient service om deze taak uit te voeren.
- **QueryController**: Deze controller heeft meerdere routes en is verantwoordelijk voor het uitvoeren van CRUD-operaties op de Neo4j-database. Het maakt gebruik van Symfony's `ValidatorInterface` voor validatie en de `OGMClient` service voor database-injectie.
  Endpoints:
  1. `GET /query`: Deze route haalt alle entiteiten op uit de Neo4j-database en rendert ze op de hoofdpagina. Het maakt gebruik van de OGMClient service om deze taak uit te voeren.
  2. `POST /insert`: Deze route is verantwoordelijk voor het toevoegen van een nieuwe node aan de Neo4j-database. Het maakt gebruik van Symfony's ValidatorInterface om de nieuwe node te valideren voordat deze wordt toegevoegd.
  3. `GET /api/nodes/{type}`: Deze route haalt alle nodes op van een bepaald type uit de Neo4j-database. Het retourneert een JSON-object met een lijst van nodes van het opgegeven type.
- **TestController**: Deze controller is meer een proof-of-concept en gebruikt de `Neo4jClient` service om alle entiteiten op te halen.
  Endpoints:
  1. `GET /test`: Deze route is meer een proof-of-concept en haalt alle entiteiten op uit de Neo4j-database. Het maakt gebruik van de Neo4jClient service om deze taak uit te voeren.

3.2.3.3 Conclusie

De API-endpoints zijn ontworpen om een reeks CRUD-operaties op de Neo4j-database uit te voeren. Ze maken gebruik van verschillende services en validatiemechanismen om ervoor te zorgen dat de integriteit van de applicatie behouden blijft en een consistente gebruikerservaring wordt geboden.

3.2.4 Services

3.2.4.1 Neo4jClient

Deze service is de brug tussen de Symfony-applicatie en de Neo4j-database. Het is ontworpen om CRUD-operaties te beheren. Kernelementen van de service zijn:

- **Constructor**:
  - Neemt parameters als `$alias`, `$url`, `$username` en `$password`. Deze parameters worden gebruikt om de gebruiker te authenticeren bij de Neo4j-database.
  - Maakt gebruik van de `ClientBuilder` om een Neo4j-client te initialiseren.
- **Methodes**:
  - `getClient()`:
    - Doel: Geeft de huidige Neo4j-client terug.
    - Return Type: ClientInterface
  - `setClient(ClientInterface $client)`:
    - Doel: Stelt een nieuwe Neo4j-client in.
    - Parameters: Een object dat voldoet aan ClientInterface.
    - Return Type: self
  - `getAllEntities()`:
    - Doel: Haalt alle entiteiten van het type Movie uit de database.
    - Return Type: CypherList
    - Cypher Query: MATCH (n: Movie) RETURN n

3.2.4.2 OGMClient

Deze service is bedoeld voor het beheren van entiteiten via Object-Graph Mapping (OGM), wat een meer abstracte manier is om met de Neo4j-database te werken.

- **Constructor**:
  - Neemt de `$url` als parameter. Deze parameter wordt gebruikt om de gebruiker te authenticeren bij de Neo4j-database.
  - Initialiseert `EntityManager` van `GraphAware\Neo4j\OGM`.
- **Methodes**:
  - `getEntityManager()`:
    - Doel: Geeft de huidige `EntityManager` terug.
    - Return Type: `EntityManager`
  - `getAllEntities()`:
    - Doel: Haalt alle `Node` entiteiten uit de database.
    - Return Type: `array`
  - `getEntity(string $id)`:
    - Doel: Haalt een specifieke `Node` entiteit op basis van de ID.
    - Parameters: `id` van het type `string`.
    - Return Type: `Node`

3.2.4.3 Conclusie

De twee belangrijkste services in de Symfony-applicatie zijn de `Neo4jClient` en de `OGMClient`. Deze dienen als ruggengraat voor alle interacties met de Neo4j-database. Ze bieden een goed gestructureerde en georganiseerde manier om de CRUD-operaties te beheren.

- `Neo4jClient`: Deze service is meer low-level en biedt de flexibiliteit om directe Cypher queries uit te voeren. Het is ideaal voor situaties waar je volledige controle wilt hebben over de database-interacties.
- `OGMClient`: Aan de andere kant biedt deze service een hoger niveau van abstractie via Object-Graph Mapping. Het is gebruiksvriendelijker en maakt het eenvoudiger om complexe relaties tussen entiteiten te beheren.

Kortom, de `Neo4jClient` en `OGMClient` bieden een goede balans tussen flexibiliteit en gebruiksvriendelijkheid. De `OGMClient` wordt voornamelijk gebruikt voor het ophalen van entiteiten voor de huidige frontend.

3.2.5 Entiteiten

3.2.5.1 Node

Algemene Informatie:
- Namespace: `App\Entity`
- Gebruikte Libraries: `GraphAware\Neo4j\OGM`, `Symfony\Component\Validator\Constraints`, `Doctrine\Common\Collections`
- Neo4j Label: `Node`

Attributen:

- `$id`:
  - Type: `string`
  - Annotaties: `@OGM\GraphId()`
  - Beschrijving: Unieke identificatie voor de node in de Neo4j-database.
- `$type`:
  - Type: `string`
  - Annotaties: `@OGM\Property(type="string")`, `@Assert\NotBlank()`
  - Beschrijving: Het type van de node. Kan niet leeg zijn.
- `$relatedNodes`:
  - Type: `Collection|Node[]`
  - Annotaties: `@OGM\Relationship(type="RELATES_TO", direction="OUTGOING", targetEntity="App\Entity\Node", collection=true)`
  - Beschrijving: Een collectie van gerelateerde nodes.

Methodes:
- `__construct()`:
  - Doel: Initialiseert de `$nodes` als een lege `ArrayCollection`.
- `addRelatedNode(Node $node): void`:
  - Doel: Voegt een gerelateerde node toe aan `$relatedNodes`.
  - Parameters: `Node $node`
- `removeRelatedNode(Node $node): void`:
  - Doel: Verwijdert een gerelateerde node uit `$relatedNodes`.
  - Parameters: `Node $node`
- `getRelatedNodes(): Collection|Node[]`:
  - Doel: Haalt de gerelateerde nodes op.
  - Return Type: `Collection|Node[]`
- `getId(): string` en `getType(): string`:
  - Doel: Haalt de waarden van `$id` en `$type` op.
  - Return Type: `string`
- `setType(string $type): self`:
  - Doel: Stelt de waarde van `$type` in.
  - Parameters: `string $type`
  - Return Type: `self`

De `Node` entiteit is vrij gestroomlijnd en biedt een goede basis voor het modelleren van nodes in een Neo4j-database. Het maakt gebruik van Symfony's validatie en Neo4j's OGM om een robuuste en flexibele entiteit te creëren.

3.3 Communicatieplan

Het is van groot belang dat alle betrokkenen bij de implementatie op de hoogte is van wat hij of zij moet doen, weet waarom de implementatie plaatsvindt en wanneer bepaalde activiteiten moeten worden uitgevoerd. De student heeft een communicatieplan opgesteld om ervoor te zorgen dat alle betrokkenen op de hoogte zijn van de implementatie.

Hiervoor is het volgende communicatieplan opgesteld:

Doelgroep	Doel	Boodschap	Communicatiemiddel
Collega	Collega's zijn op de hoogte van de voortgang van het product, openstaande taken, retrospective en planning.	De voortgang van het product en openstaande taken worden gedeeld via Jira en dagelijkse stand-ups. Retrospective en planning om de week	Dagelijkse stand-up met collega's en Jira.
Opdrachtgever	De opdrachtgever is op de hoogte van de voortgang van het product en kan feedback geven waar nodig.	De voortgang van het product en de mogelijkheid tot feedback.	Wekelijkse online meeting met de opdrachtgever.
Begeleiders	Begeleiders zijn op de hoogte van de voortgang van het product en kunnen feedback geven waar nodig.	De voortgang van het product en de mogelijkheid tot feedback.	E-mail, fysiek overleg of online meeting.

4 Beheersplan

4.1 Doel

Het volgende beheersplan is bedoeld om de levenscyclus van de applicatie te stroomlijnen, van ontwikkeling tot productie en onderhoud. Dit plan is bedoeld voor de opdrachtgever, de begeleiders en de toekomstige ontwikkelaars van de applicatie.

4.2 Ontwikkelingsfase

- **Versiebeheer**: Gebruik Git voor versiebeheer. Het hosten van de repository kan op Github.
- **Code Reviews**: Voer regelmatig code reviews uit om de codekwaliteit te waarborgen.

4.3 Testfase

- **Unit Tests**: Schrijf unit tests voor de belangrijkste functionaliteiten van de applicatie.
- **Testomgeving**: Gebruik een testomgeving om de applicatie te testen voordat deze in productie gaat.

4.4 Deploymentfase:

- **Deployment**: Gebruik een CI/CD pipeline om de applicatie te deployen. Dit kan bijvoorbeeld met Github Actions, GitLab CI of Jenkins.
- **Docker**: Gebruik Docker om de applicatie te containeriseren. Dit maakt het gemakkelijker om de applicatie te deployen en te schalen.

4.5 Onderhoudsfase:

- **Monitoring**: Gebruik een monitoring tool om de applicatie te monitoren. Dit kan bijvoorbeeld met Prometheus, Grafana of Zabbix.
- **Back-ups**: Maak regelmatig back-ups van de applicatie en database.

4.6 Toegangsbeheer: (Wanneer nodig)

- **Authenticatie**: Implementeer sterke authenticatiemechanismen.
- **Autorisatie**: Definieer rollen en toegangsrechten.

5 Conclusie

In dit rapport is een uitgebreide analyse uitgevoerd van de Symfony-Neo4j applicatie. De implementatie is gedetailleerd beschreven, van kerncomponenten en configuratie tot Docker-setup, om een volledig beeld te geven van de applicatiearchitectuur. Het communicatieplan heeft een cruciale rol gespeeld in het project, en het beheersplan biedt een raamwerk voor toekomstig beheer en onderhoud.

De documentatie in dit rapport dient als een uitgebreide handleiding voor het begrijpen van de huidige implementatie en biedt een solide basis voor toekomstige ontwikkeling en schaalbaarheid. Door de nadruk te leggen op best practices, is de applicatie ontworpen met onderhoudbaarheid en uitbreidbaarheid als kernprincipes.

Samenvattend, dit rapport toont aan dat de Symfony-Neo4j applicatie een robuust en flexibel systeem is, goed gepositioneerd voor toekomstige groei en het voldoen aan de behoeften van een uitbreidend gebruikersbestand.

6 Bronnen

Bijlagen

Hanzehogeschool Groningen logo	Hanzehogeschool Groningen	https://freebiesupply.com/logos/hanzehogeschool-groningen-logo/
Titelpagina figuur	DALL-E-3, OpenAI	bijlagen -> realisatierapport -> OIG.jpg
Symfony Server Source Code	L. J. J. Leuwol	https://github.com/Shambuwu/neo4j-symfony-app