|  |
| --- |
| CMD opdrachtenbox\_  Software Guidebook |

Auteurs: Eva Ketelaar, Pedro van Douveren, Stijn Appeldoorn, Stijn Welles, Yuillian Oorschot

Groepsnaam: Pittige Pinguïns

Versie: 1

17 januari 2025

INHOUDSOPGAVE

[1 Context 4](#_Toc187761796)

[1.1 Systeemoverzicht 4](#_Toc187761797)

[1.1.1 Overzichtsdiagram 4](#_Toc187761798)

[1.2 Belangrijke stakeholders 5](#_Toc187761799)

[1.2.1 Docent 5](#_Toc187761800)

[1.2.2 Opdrachtgever 6](#_Toc187761801)

[1.2.3 Student 6](#_Toc187761802)

[2 Functioneel overzicht 7](#_Toc187761803)

[2.1 Kernfunctionaliteit 7](#_Toc187761804)

[2.1.1 Docent 7](#_Toc187761805)

[2.1.2 Opdrachtgever 7](#_Toc187761806)

[2.1.3 Student 8](#_Toc187761807)

[2.2 Niet-functionele eisen 8](#_Toc187761808)

[2.3 Randvoorwaarden en beperkingen 8](#_Toc187761809)

[2.3.1 Technische randvoorwaarden 8](#_Toc187761810)

[2.3.2 Juridische randvoorwaarden 9](#_Toc187761811)

[3 Code 10](#_Toc187761812)

[3.1 Architectuurtype: Layered architecture 10](#_Toc187761813)

[3.1.1 Overzicht van de lagen 10](#_Toc187761814)

[3.2 Codecomponenten 15](#_Toc187761815)

[3.3 Belangrijke libraries en frameworks 15](#_Toc187761816)

[3.3.1 Security 15](#_Toc187761817)

[3.3.2 Mailing 17](#_Toc187761818)

[3.4 Innovaties en Technische Keuzes 19](#_Toc187761819)

[3.4.1 Het gebruik van Java-records 19](#_Toc187761820)

[3.4.2 Het gebruik van Java Optionals 19](#_Toc187761821)

[3.4.3 Het gebruik van Generics en interfaces 20](#_Toc187761822)

[3.4.4 Generieke exceptions 21](#_Toc187761823)

[3.4.5 Validatie en sanitatie 21](#_Toc187761824)

[4 Data 23](#_Toc187761825)

[4.1 Datamodellen 23](#_Toc187761826)

[4.2 Datastromen 23](#_Toc187761827)

[4.3 Integraties en externe gegevensbronnen 23](#_Toc187761828)

[4.4 Overige opmerkingen 24](#_Toc187761829)

[5 Deployment 25](#_Toc187761830)

[5.1 Infrastructuur 25](#_Toc187761831)

[5.2 Resourcebeheer 26](#_Toc187761832)

[5.3 CI/CD-pipeline 27](#_Toc187761833)

[5.4 Foutafhandeling en beschikbaarheid 29](#_Toc187761834)

[5.5 Configuratie en beheer 29](#_Toc187761835)

# Context

In dit hoofdstuk wordt de algemene context van het systeem beschreven en de relatie met externe factoren.

## Systeemoverzicht

Momenteel heeft de HAN geen makkelijke oplossing om CMD-opdrachten direct te koppelen aan een student. Op dit moment moet een bedrijf een docent van de HAN benaderen. Die docent moet dan een geschikte student vinden om aan die opdracht te koppelen. Dit kost veel tijd voor de docent, of kan ervoor zorgen dat het bedrijf ergens anders moet gaan zoeken om die opdracht uit te laten voeren.

Dit systeem functioneert als platform waar externe bedrijven projecten aan kunnen maken en hier taken aan kunnen koppelen, samen met de gewenste skills. Studenten kunnen een profiel aanmaken met hun skills en kunnen op basis daarvan onder andere een overzicht van projecten te zien krijgen die het beste bij hen passen. Studenten kunnen zich aanmelden voor projecten. Vervolgens kan een opdrachtgever uit de lijst van aanmeldingen een student accepteren of afwijzen voor de taak.

*Bron:* [*Plan van Aanpak*](https://hannl.sharepoint.com/teams/AIM-ICT-CNP-sep24-PittigePinguns/Gedeelde%20documenten/PvA%20pittige%20pinguins.docx)

### Overzichtsdiagram

Hieronder is een contextdiagram van het systeem weergegeven volgens het C4-model. Hierin worden de externe factoren en belangrijkste systeemgrenzen getoond.

A diagram of a group of people

Description automatically generated

Figuur : System context diagram voor CMD Opdrachtenbox

## Belangrijke stakeholders

In deze paragraaf worden de gebruikers van het systeem beschreven.

### Docent

De rol “docent” verwijst naar een docent aan de HAN. Deze rol heeft een beheerdersfunctie binnen de applicatie. Docent helpen om de integriteit van de applicatie te bewaren. Zij reguleren bijvoorbeeld welke bedrijven en skills aan het systeem worden toegevoegd. Als beheerder beschikt deze rol over bevoegdheden die opdrachtgevers en studenten niet hebben.

### Opdrachtgever

De rol “opdrachtgever” verwijst naar een medewerker van een bedrijf dat is aangemeld bij het systeem. Een supervisor wordt aan het systeem toegevoegd op uitnodiging van een docent. Opdrachtgevers kunnen hun eigen bedrijfsprofiel beheren, projecten en taken aanmaken en skills aan deze taken toevoegen. Daarnaast kunnen zij zelf nieuwe skills aanmaken. Verder kunnen zij studentaanmeldingen bekijken en deze accepteren of afwijzen. Hiervoor kunnen zij eventueel de profielpagina van een student bekijken, zodat zij meer informatie over de student op kunnen doen.

### Student

De rol “student” verwijst naar een CMD student aan de HAN. Studenten kunnen hun eigen profielpagina beheren. Aan deze profielpagina kunnen zij skills toevoegen. Hieraan kan een beschrijving gekoppeld worden, waarin bijvoorbeeld ervaringen en competenties kunnen worden toegelicht. Daarnaast kunnen zij bedrijfsprofielen, projecten en taken bekijken. Verder kunnen studenten zich aanmelden voor taken. Voor deze aanmeldingen kunnen zij een motivatiebrief schrijven om hun aanmelding te ondersteunen.

# Functioneel overzicht

In dit hoofdstuk worden de belangrijkste mogelijkheden van het systeem beschreven.

## Kernfunctionaliteit

In deze paragraaf worden de kernfunctionaliteiten van de stakeholders toegelicht. Zie voor de resterende user stories de [Issues pagina op GitHub](https://github.com/AIM-CNP-sep24/project-cnp-pittige-pinguins/labels/user%20story). User stories met het label “Niet goedgekeurd” zijn nog niet goedgekeurd door de product owner.

### Docent

#### Aanmaken van bedrijfspagina’s

Docenten kunnen nieuwe bedrijfspagina’s aanmaken op de beheerderspagina. Deze bedrijfspagina’s dienen als basisprofielpagina voor bedrijven die worden uitgenodigd.

#### Uitnodigen van opdrachtgevers

Na het aanmaken van de basisprofielpagina kan de docent een link genereren om een opdrachtgever van het bedrijf uit te nodigen. De docent moet deze link mailen naar de toekomstige opdrachtgever. Dit vindt plaats buiten de applicatie om.

#### Nieuwe skills beheren

Op de beheerderspagina worden nieuwe skills, die door opdrachtgevers zijn toegevoegd, getoond. Deze nieuwe skills moeten worden goedgekeurd door een docent. De docent heeft de mogelijkheid om deze skills af te wijzen, goed te keuren of aan te passen vóór goedkeuring.

### Opdrachtgever

#### Aanpassen van bedrijfspagina

Opdrachtgevers kunnen hun bedrijfspagina aanpassen door de profielfoto en beschrijving aan te passen.

#### Uitnodigen van collega’s

Opdrachtgevers kunnen collega’s uitnodigen om ook in het bedrijf projecten te maken. Deze krijgen dezelfde rechten als de originele opdrachtgever.

#### Toevoegen en beheren van projecten en taken

Opdrachtgevers kunnen projecten aanmaken en bewerken. Ze kunnen ook taken binnen een project aanmaken en de benodigde skills hiervoor aanpassen. Bij het maken van een project geven ze een titel, beschrijving en foto mee. Bij het aanmaken van een taak geven ze een titel, beschrijving en aantal benodigde mensen mee.

#### Beheren van studentaanmeldingen

Opdrachtgevers kunnen bij hun aangemaakte taken, aanmeldingen bekijken en deze goedkeuren of afkeuren. Het aantal beschikbare plekken wordt met één verminderd wanneer er een aanmelding wordt goedgekeurd.

#### Toevoegen van skills

Wanneer er een skill niet bestaat, kan een opdrachtgever deze aanmaken. Dit is een tijdelijke skill, aangeduid door een andere kleur op de pagina. Een docent moet deze goedkeuren, waarna de skill de normale kleur krijgt.

### Student

#### Aanpassen van persoonlijke pagina (toevoegen van skills en omschrijving)

Een student kan zijn beschrijving op de persoonlijke pagina aanpassen en skills aan zichzelf toevoegen.

#### Aanmelden voor taken

Een student kan zich aanmelden voor een taak binnen een project. De student kan een motivatie achterlaten, die de opdrachtgever kan zien.

## Niet-functionele eisen

Er is gevraagd om de applicatie zo makkelijk mogelijk te maken om te deployen. Wij gebruiken hiervoor Docker om in één keer alle services op te starten.

* De Product Owner heeft het inloggen via OAuth2 via Google en GitHub als beveiligingseis gesteld. De implementatie hiervan wordt verder toegelicht in [3.3.1 Security](#_Security);
* Volgens de Product Owner waren optimalisaties aan het ophalen van data voor nu niet aan de orde. Dit zal geïmplementeerd moeten worden als het aan de orde komt;
* De repository laag maakt gebruik van interface-lagen om de database van de rest van de applicatie te scheiden. Dit bevordert zowel onderhoud als schaalbaarheid. Hier is meer over te lezen in [3.1.1 Overzicht van de lagen](#_Repositorylaag).

## Randvoorwaarden en beperkingen

### Technische randvoorwaarden

* Binnen de applicatie wordt gewerkt met de programmeertalen HTML, CSS, JavaScript, Java en SQL;
* Voor de ontwikkeling van de front-end wordt React gebruikt, in combinatie met het framework Tailwind CSS voor de styling;
* De front-end wordt specifiek ontwikkeld voor de meest recente stabiele versies van de browsers Google Chrome, Edge, FireFox en Safari. Er wordt geen ondersteuning geboden voor oudere versies van de bovengenoemde browsers en Internet Explorer;
* De applicatie wordt ontworpen voor schermen met een minimale breedte van 320 pixels. Er wordt geen ondersteuning geboden voor apparaten met schermen kleiner dan 320 pixels.

### Juridische randvoorwaarden

* In overleg met de Product Owner houden wij tijdens de ontwikkeling geen rekening met juridische beperkingen. Dit wordt verder toegelicht in [4.4 Overige Opmerkingen](#_Overige_opmerkingen).

# Code

In dit hoofdstuk worden de belangrijkste structuren en patronen in de broncode beschreven.

## Architectuurtype: Layered architecture

De Opdrachtenbox maakt gebruik van een **Layered Architecture,** dit patroon deelt de software op in afzonderlijke lagen met elk hun eigen verantwoordelijkheid. Hier beschrijven we hoe deze lagen met elkaar samenwerken:

* De **presentatie laag** (frontend) verwerkt gebruikersinteracties en communiceert met de backend via RESTful API’s (endpoints);
* De **Controller laag** handelt inkomende verzoeken af, verifieert de input vanuit de presentatie laag en ligt tussen de frontend en service laag. Deze laag functioneert als een doorgeefluik;
* De **Service laag** bevat de logica van de applicatie en verwerkt de gegevens die het binnenkrijgt vanuit de controller laag;
* De **Repository laag** beheert de toegang tot de database.

### Overzicht van de lagen

De lagen in de backend werken samen volgens het **request**-**response**-principe. Dit werkt alsvolgt:

1. De **frontend** stuurt een HTTP-verzoek naar een endpoint in de **controllerlaag**;
2. De **controller** valideert het verzoek en roept de corresponderende methode(s) aan in de **servicelaag**;
3. De **servicelaag** voert de businesslogic uit en gebruikt de **repositorylaag** om gegevens op te halen, te verwijderen, te wijzigen of op te slaan. (Voor dit voorbeeld houden we ophalen aan);
4. De juiste **repository** haalt de gegevens op uit de database en retourneert deze aan de **servicelaag**;
5. De **servicelaag** retourneert het resultaat aan de **controllerlaag**, die het vervolgens terugstuurt naar de **frontend** in JSON-formaat.



Figuur : Flowchart van interactie tussen lagen

#### Presentatie laag

* Rol: de **presentatie laag** fungeert als een toegangspunt voor de gebruiker naar de website en de backend. Het verwerkt de inkomende gebruikersacties en stuurt deze naar de backend. Ook haalt het via de backend de relevante informatie op voor de gebruiker en toont deze data;
* Communicatie met de backend: De presentatie laag communiceert met de backend via een services laag op het front-end. Deze vraagt de nodige data via een functie op en maakt een ‘fetch call’ met (meestal JSON) geformatteerde data naar de juist URL op de backend. Deze services laag is te vinden in het bestand: ‘/frontend/src/services.js’.

#### Controllerlaag

* **Rol**: de **controllerlaag** fungeert als een toegangspunt naar de backend en verwerkt inkomende HTTP-verzoeken vanuit de **frontend**. De controllers roepen de juiste servicemethoden aan en retourneren een HTTP-response in het juiste formaat (meestal JSON);
* Endpoints.

Hieronder vind je een overzicht van de meest gebruikte endpoints binnen de applicatie. Voor een volledig overzicht van alle beschikbare endpoints kun je het API-contract raadplegen in [documentatie\openAPI.yaml](https://github.com/AIM-CNP-sep24/project-cnp-pittige-pinguins/blob/development/documentatie/openAPI.yaml).

**GET /projects/{projectId}**

**Beschrijving:**Haalt een specifiek project op basis van het meegeleverde projectId. Dit endpoint wordt veel gebruikt door de frontend en is beschikbaar voor elke ingelogde gebruiker.

**Locatie in de code:**

* Controller**:***backend/src/main/java/com/han/pwac/pinguins/backend/controller/ProjectController.java*
* Return Type:  
  **GetProjectDto**  
  *backend/src/main/java/com/han/pwac/pinguins/backend/domain/DTO/GetProjectDto.java*

**POST /tasks/{projectId}**

**Beschrijving:**Voegt een nieuwe taak toe aan een project. Dit endpoint is toegankelijk voor opdrachtgevers van een bedrijf.

**Locatie in de code:**

* **Controller:***backend/src/main/java/com/han/pwac/pinguins/backend/controller/TaskController.java*
* **Return Type:  
  ResponseEntity**
  + Body: "taak toegevoegd"
  + Statuscode: 201

**PATCH /registrations**

**Beschrijving:**Wijzigt de status van een registratie op basis van de doorgegeven gegevens. Het is toegankelijk voor opdrachtgevers en stuurt een e-mailnotificatie naar de betrokken student.

**Locatie in de code:**

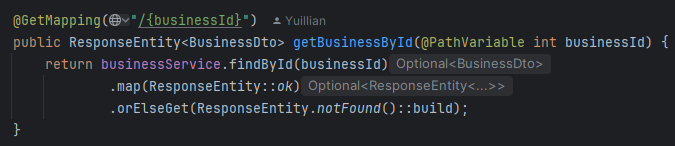
* **Controller:***backend/src/main/java/com/han/pwac/pinguins/backend/controller/RegistrationController.java*
* **Return Type:**  
  ResponseEntity<Object>

**Annotaties**:

* @OAuth2 – Zorgt voor OAuth2-authenticatie.
* @Sanitation – Verwerkt en valideert inkomende data.

**Request Body:**

* **Type:** PatchRegistrationDto
  + **userId**: ID van de gebruiker die de registratie wijzigt.
  + **taskId**: ID van de taak waaraan de registratie gekoppeld is.
  + **accepted**: Boolean, geeft aan of de registratie is geaccepteerd of afgewezen.
  + **response**: Optionele reactie of reden voor de wijziging.
* **Tech**: Spring Boot REST Controllers
* Belangrijkste componenten:
  + Annotaties zoals @RestController, @RequestMapping, @GetMapping, @Postmapping voor het definieren van **endpoints**.
  + Het ontvangen en valideren van inputparameters.
  + Het afhandelen van uitzonderingen en fouten vanuit de **frontend**.
* Voorbeeldcode:



Figuur 3: controllers/BusinessController.java. Regel:40.

In dit voorbeeld:

* De **controller** ontvangt HTTP-verzoeken (@GetMapping) en stuurt de aanvraag door naar de servicelaag. In dit geval ziet het endpoint er zo uit: {API\_URL}/business/1;
* De methode retourneert een **ResponseEntity** met het **BusinessDto** in JSON-formaat;
* Als het **BusinessDto** Optional object leeg is dan retourneert het een NotFound exceptiepagina.

#### Servicelaag

* **Rol**: de **servicelaag** bevat de businesslogic van de applicatie. Hier worden de regels en processen gedefinieerd die bepalen doe gegevens worden verwerkt. De service voert taken uit zoals validatie, berekeningen en het aanroepen van de **repositorylaag** voor interactie met de database;
* Tech: Spring Boot Services (@Service);
* Voorbeeldcode:

A computer screen shot of a program code

Description automatically generated

Figuur 4: services/UserService.java. Regel: 11.

In dit voorbeeld:

* De **servicelaag** haalt gegevens op via de **repositorylaag** en voert validaties uit voordat resultaten teruggegeven worden;
* Als de het Optional object User leeg is retourneert het weer een Not Found.

#### Repositorylaag

* **Rol**: de **repositorylaag** is verantwoordelijk voor de communicatie met de database. Hier worden gegevens opgehaald, opgeslagen, aangepast en verwijderd. In deze implementatie gebruiken wij **JdbcTemplate** voor de database interacties. De **repository** zorgt ervoor dat de applicatie niet direct afhankelijk is van de database, door middel van abstracties zoals DAO-interfaces;
* **Tech**: Spring JDBC met **JdbcTemplate**, het gebruik van **DAO** (Data Acces Object) interfaces voor gegevens beheer en RowMapper voor het mappen van database resultaten naar Java-objecten;
* Belangrijkste componenten:
  + Annotaties zoals @Repository
  + **JdbcTemplate**: Dit is de Spring-klasse die de verbinding met de database beheert en SQL-query’s uitvoert;
  + **RowMapper**: Een interface die wordt gebruikt om rijen uit de database om te zetten naar Java-objecten;
  + **DAO-interfaces**: de interfaces definiëren de contracten voor het handelen met de data en worden geïmplementeerd door de concrete DAO’s.
* Voorbeeldcode:

In dit voorbeeld zie je de implementatie van een DAO-klasse die de gegevens opvraagt uit de database met behulp van **JdbcTemplate** en een **Rowmapper** om de resultaten om te zetten naar een Java-object

A screenshot of a computer program

Description automatically generated

Figuur 5: repository/BusinessDao.java. Regel: 17.

In het volgende voorbeeld: De DAO-interfaces definiëren de contracten voor de repository operaties.

A black background with blue and orange text

Description automatically generated

Figuur 6: repository/contracts/IBusinessDao.java. Regel: 7.

Daarnaast is er nog een generieke basisinterface:

A screenshot of a computer program

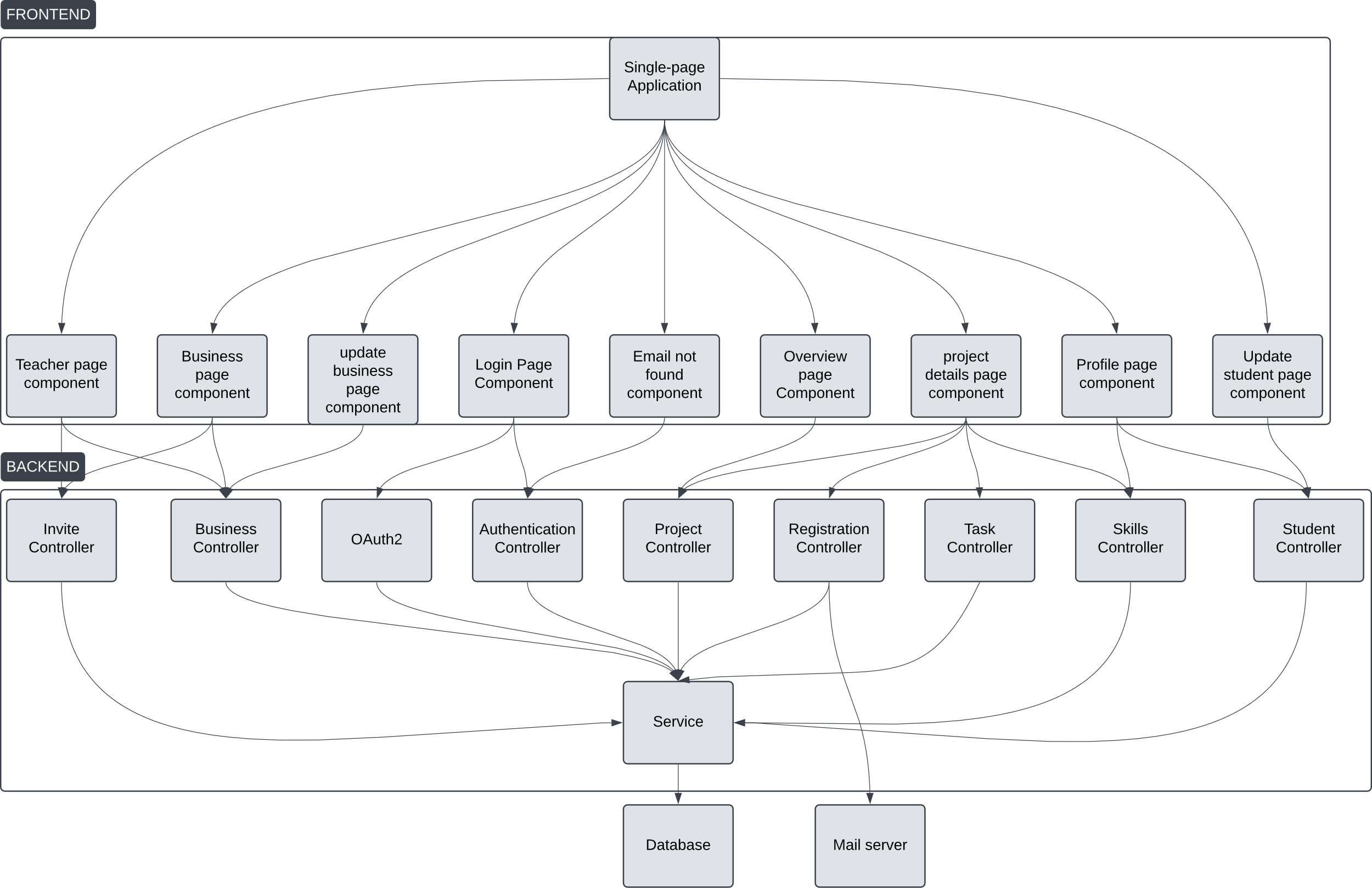
Description automatically generated

Figuur 7: repository/contracts/base/IBaseDao. Regel: 6.

* De **IBusinessDao** interface definieert de methoden die door de implementatie (**BusinessDao**) worden gebruikt om gegevens op te halen, bij te werken of te verwijderen. De interface erft van de generieke **IBaseDao** die basis CRUD-operaties biedt voor andere entiteiten;
* Als er geen resultaat wordt gevonden als de repository wordt geraadpleegd wordt er een **EmptyResultDataAccessException** opgevangen en een aangepaste uitzondering (**NotFoundException**) Gegooid.

## Codecomponenten

Hieronder staat een componenten diagram van de front-end, alle pagina’s, alle controllers, de database en de mailserver. Alle connecties worden weergegeven als pijlen.



Figuur : Component diagram

## Belangrijke libraries, frameworks en services

|  |  |  |  |
| --- | --- | --- | --- |
| Naam | Versie | Wat is het | Overige opmerkingen |
| Node | 20.17.0 | Een projectmanagement tool die dependencies beheert. Zorgt ervoor dat de front-end zijn dependencies ontvangt bij het opstarten. |  |
| React | 18.3.1 | Een framework voor het verwerken van JSX naar html |  |
| TailwindCSS | 3.4.15 | Een library die een aantal CSS-classes heeft om makkelijk CSS te schrijven in HTML. | Hiervoor hebben we gekozen omdat er veel kant en klare style classes beschikbaar zijn. |
| Storybook | 8.4.0034 | Een library om visuele tests te schrijven. Wordt ook gebruikt in de CI-pipeline om die tests te automatiseren |  |
| React Router | 6.28.0 | Een library voor React, zorgt ervoor dat je verschillende componenten toont op basis van de URL. |  |
| Java | 21 | Een taal die tijdens de lessen geleerd is |  |
| Maven | 4.0.0 | Een projectmanagement tool die dependencies beheert. Zorgt ervoor dat de backend zijn dependencies ontvangt bij het opstarten. |  |
| Spring boot | 3.3.5 | Een framework voor het afhandelen van http-verzoeken en zorgt ervoor dat dit snel en simpel is om met http te werken |  |

### Security

Wij gebruiken de spring security library om de beveiliging van de applicatie te regelen. Dit wordt in de SecurityConfig binnen de configuration folder afgehandeld. Hier wordt op basis van een specifieke endpoint gekeken of de gebruiker de juiste autoriteit heeft om bij deze endpoint te kunnen komen, zo niet krijgt de gebruiker een 403 te zien wat aan te passen is in de exceptionHandling functie. In het figuur hieronder is te zien hoe dat geïmplementeerd is binnen de applicatie.

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Figuur 9: configuration/SecurityConfig.java, regel: 66

Voor het login systeem gebruiken wij OAuth2. Dit zorgt voor een veilig inlog systeem en transfer van data. In de SessionFilter binnen de authentication folder wordt de autorisatie toegevoegd die uit de OAuth2AuthenticationToken gehaald die later gebruikt kan worden in de hasAuthority functie. Om de OAuth2 gebruiker info op te kunnen halen kun je de @OAuth2 annotatie gebruiken, deze kun je boven een functie van een controller plaatsen. Zo wordt de UserInfo klasse als parameter geïnjecteerd, dit heb je minimaal nodig om een gebruiker te identificeren.

Hieronder is te zien hoe de OAuth2 login wordt geïmplementeerd binnen spring boot. Er is hier een failureHandler functie voor wanneer er een error ontstaat bij het inloggen met OAuth2 en een successHandler functie voor wanneer een gebruiker succesvol is ingelogd. Binnen de successHandler wordt de gebruiker aangemaakt als dit account nog niet aangemaakt is.

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

Figuur 10: configuration/SecurityConfig.java, regel: 87

In het figuur hieronder is een voorbeeld te zien hoe je de informatie van een gebruiker kan ophalen met behulp van de @OAuth2 annotatie. Om de gebruiker zijn id op te halen kun je de getVerificationByProviderId functie gebruiken en het id veld uit de UserInfo hierbij gebruiken.

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Figuur 11: controller/AuthenticationController.java, regel: 37

Om nieuwe providers toe te voegen aan OAuth2 moet dit gedaan worden binnen het application.properties bestand. Hier kan het client-id en de client-secret van die specifieke provider aangegeven worden. Hierna kun je in de Provider enum, binnen de authentication folder de nieuwe provider naam toevoegen, de naam van de provider moet gelijk staan aan de client registration id van de provider en is niet hoofdletter gevoelig. Hierna kun je de data die je nodig hebt ophalen in de getUserInfo functie.

### Mailing

Voor mailing wordt de IMailService interface gebruikt. Hierin heb je een sendMail functie, hiermee kunnen mails verstuurd worden met behulp van de Mail klasse. De Mail klasse vraagt een onderwerp, ontvanger en de body van de mail. Bij het versturen kan er een of meerder mails in een batch verstuurd worden.

De Mail klasse vraagt een body wat een EmailBody klasse is. Deze EmailBody haalt binnen de constructor op basis van een boolean de template op, die te vinden is in de templates folder binnen de resources folder. Deze template bevat parts waarnaar gezocht kan worden, dit is zodat je makkelijk de template kan aanpassen zonder dat de mails die verstuurd worden er fout uit zullen zien. Een part wordt gedefinieerd door @part(“{part naam}”) in de html te zetten, op de plek waar je wil dat de code start. Een part kan geselecteerd worden door EmailBody.getPart(“{part naam}”) te gebruiken. Hier krijg je een EmailPart uit als de part gevonden kan worden. Deze EmailPart kan gebruikt worden om de email te vullen met voor die email specifieke tekst/html. Hieronder een voorbeeld van het versturen van een email.

Afbeelding met tekst, schermopname, Lettertype, software

Automatisch gegenereerde beschrijving

Figuur : email code voorbeeld

Bij het versturen komt er deze mail uit. Voor dit voorbeeld is Mailhog gebruikt. Dit wordt tijdelijk gebruikt voor het versturen van mails.

Afbeelding met tekst, Lettertype, schermopname, ontwerp

Automatisch gegenereerde beschrijving

Figuur : email voorbeeld

Voor de dagelijkse email samenvatting wordt een cronjob gebruikt. Cronjobs zijn te vinden in de package genaamd: ‘cron\_jobs’. Binnen deze package, staat een klasse genaamd MailCronJob, deze zorgt ervoor dat er dagelijks samenvattingen worden verstuurd naar gebruikers. Dit wordt gedaan door een lijst van ‘IMailCronService’ instanties op te halen met behulp van dependency injection. Elke keer als de cron job aangeroepen wordt, wordt de ‘getBatchedMails’ functie aangeroepen voor elke instantie in de lijst. De cronjob verwacht hier een map met als sleutel de identifiers van de gebruikers waar de mail naartoe gestuurd moet worden en een EmailPart. De cron job zal alle mails van individuele gebruikers bij elkaar samenvoegen om hiervan een samenvatting te maken.

Om zelf mails aan de samenvatting toe te voegen, moet je een service aanmaken die de ‘IMailCronJob’ interface implementeert. Binnen de functie die deze interface verwacht, kun je de data ophalen voor de mails. Na het ophalen van de data kan het EmailPart aangemaakt worden, deze part kun je eruit laten zien hoe je wilt door gebruikt te maken van de functies binnen de EmailPart klasse. Je hoeft geen begroeting of afsluiting toe te voegen dit wordt binnen de cronjob al gedaan. De part namen hiervan kunnen uit de template gehaald worden. Wat hier verder geretourneerd moet worden staat in de alinea hierboven.

Hieronder een voorbeeld implementatie van de IMailCronService interface.



Figuur : services/SkillService.java

## Innovaties en Technische Keuzes – Back-end

In dit hoofdstuk worden technologieën beschreven die afwijken van stand de behandelde lesstof, maar die bewust zijn gekozen vanwege hun voordelen voor onderhoud en efficiëntie van de code.

### Het gebruik van Java-records

Javarecords zijn een manier om gebruik te maken van immutable Java-objecten. Het voordeel aan records is ook dat het komt met minimale boilerplate-code. Ze worden toegepast voor het uitwisselen van data en het aanmaken van onze domeinen zoals in het volgende voorbeeld:



Figuur : domain/DTO/GetSkillDto.java

Voordelen:

* De getters en setters automatisch gegenereerd worden. Ook de toString methode wordt klaargezet;
* Garantie dat de objecten immutable zijn.

### Het gebruik van Java Optionals

We gebruiken Optionals om om te gaan met Null values, hiermee voorkomen wij NullPointer Excepties.

A computer screen shot of a program

Description automatically generated

Figuur 16:repository/BusinessDao.java.Regel: 62

Hier wordt Optional gebruikt om een null op te vangen mocht de query of de mapper geen resultaat kunnen vinden of maken.

Voordelen:

* Het dwingt ons om na te denken over de mogelijkheid dat waarden leeg kunnen zijn;
* Voorkomt runtime fouten zoals **NullPointerException** doordat je expliciet moet omgaan met de mogelijkheid dat een waarde ombreekt.

### Het gebruik van Generics en interfaces

We gebruiken Generics zodat we generieke oplossingen kunnen implementeren die inzetbaar zijn binnen verschillende onderdelen van onze applicatie. Generics worden gebruikt om typeparameters te introduceren in klassen. Dit biedt de volgende voordelen:

* Generieke klassen kunnen met verschillende datatypes worden gebruikt, zonder dat je code voor elk type opnieuw hoeft te schrijven.

Een voorbeeld uit onze applicatie:

A screenshot of a computer program

Description automatically generated

Figuur 17: repository/contracts/base/IBaseDao. Regel: 6.

In dit interface worden de generieke methoden gebruikt die kunnen worden geïmplementeerd door elke klasse die data moet beheren, ongeacht het type entiteit (**T**) of ID (**TId**).

Voordelen:

* Minimaliseert (duplicate) code;
* Door op het codevoorbeeld hierboven te extenden met een nieuwe implementatie wordt bijvoorbeeld findById ondersteund bij objecten naar keuze;
* De implementatie van een interface kan vervangen of aangepast worden zonder dat de servicelaag aangepast moet worden. Door deze losse koppeling is de code flexibel;
* Als er bijvoorbeeld een nieuwe manier van dataopslag komt kun je simpleweg een nieuwe implementatie toevoegen van de interface.

### Generieke exceptions

Voor excepties gebruiken wij een GlobalException klasse. Deze klasse erft over van RuntimeException en kan dus overal gebruikt worden. Door deze exceptie te gebruiken worden er minder exception handlers gemaakt om alle excepties af te handelen en wordt dit op één specifieke manier gedaan. Wanneer deze error ontstaat zal de error er bijna hetzelfde uit zien als de standaard errors van spring boot, alleen wordt er ook nog een bericht meegegeven. Dit bericht kan op het front-end opgevangen worden.

Op het front-end kunnen errors die uit de backend komen opgevangen worden met behulp van de fetchWithError functie die in het services.js bestand. Bij het gebruik van deze functie worden errors automatisch in de catch gezet als HttpError klasse, zo wordt het bericht ook automatisch meegegeven en kan dit op de front-end als bericht getoond worden. Bij het gebruik van deze functie wordt de json response van de server automatisch naar een javascript object geconverteerd.

### Validatie en sanitatie

Op de backend moeten waarden ook gevalideerd worden. Dit kun je doen door standard door spring geïmplementeerde validatie annotaties, zoals @Length voor string lengtes, @Min/Max voor nummers of @Valid voor validatie voor velden van klassen. Als een waarde niet gevalideerd kan worden zal er een error bericht ontstaan op basis van wat er niet gevalideerd kan worden. Dit bericht zal in de taal staan van de laptop waarop de server draait.

Waarden moeten ook gesanitiseerd worden. Dit kun je doen door de @Sanitize annotatie boven een controller te plaatsen. Zo wordt automatisch html uit tekst gehaald. Dit wordt ook automatisch binnen objecten gedaan.

## Innovaties en Technische Keuzes – Front-end

### Children prop

Binnen React maken wij ook gebruik van de children property. Dit zorgt ervoor dat alle child elements die je binnen een component plaatst met het children property automatisch meegegeven worden aan dat element. Zo kun je de children op een eigen plek in de html plaatsen. Hieronder een voorbeeld van een component met de children prop.



Figuur 18: frontend/src/components/Modal.jsx

Hieronder een voorbeeld van hoe de children mee worden gegeven aan de component:

Afbeelding met tekst, schermopname

Door AI gegenereerde inhoud is mogelijk onjuist.

Figuur 19: frontend/src/components/NewSkillsManagement.jsx

### Front-end: Notificatie systeem

Op het front-end is er ook een ingebouwde notificatie systeem. Zo’n notificatie kan overal aangemaakt worden op het front-end en ziet er zo uit:

Afbeelding met tekst, Lettertype, schermopname, ontwerp

Door AI gegenereerde inhoud is mogelijk onjuist.

Figuur : notificatie

Om zo’n notificatie aan te maken moet je ‘notification’ importeren uit het bestand: ‘/frontend/src/components/notifications/NotifySystem.jsx’. Om hiermee een notificatie aan te kunnen maken moet je een van de notificatie functies aanroepen zoals: success of error. Aan deze functie kun je een bericht meegeven. Dit bericht wordt dan getoond in de notificatie.

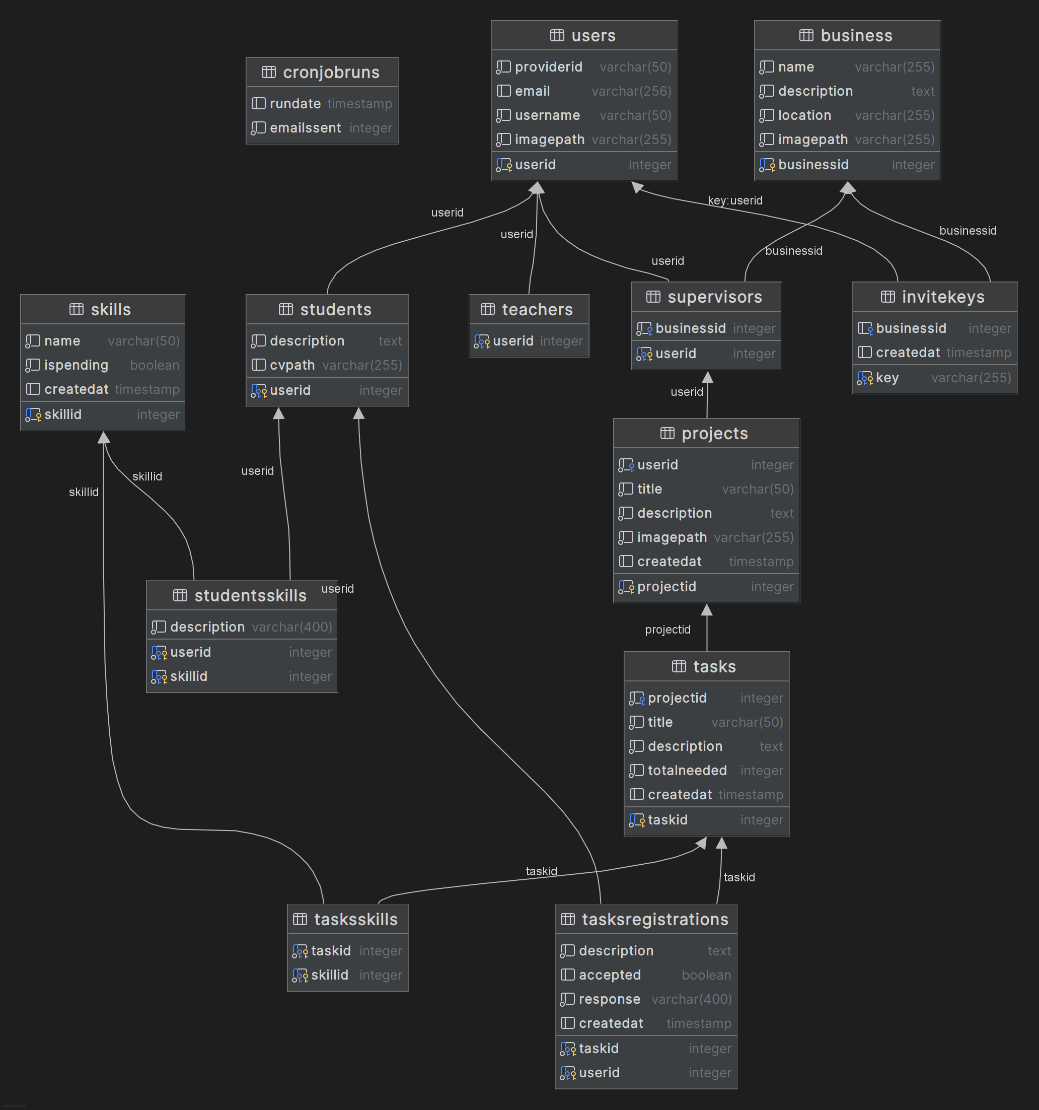
Ook kun je zelf luisteren of er een notificatie binnen komt. Je kan hiervoor de listen functie aanroepen en hieraan een callback functie meegeven. De argumenten van deze callback zijn dan het bericht en het type bericht wat gestuurd is zoals: success of error. Deze callback functie wordt bij elke notificatie aangeroepen.

# Data

In dit hoofdstuk staat beschreven welke gegevens het systeem verwerkt, hoe deze worden opgeslagen en hoe deze worden uitgewisseld.

## Datamodellen

Hieronder is de databasestructuur te zien in de vorm van een ERD. In het volgende hoofdstuk staat beschreven hoe deze database uitgevoerd kan worden.



Figuur : ERD

## Datastromen

Hieronder staan de verschillen beschreven tussen de databaseobjecten en bijbehorende domeinobjecten:

* Er bestaat geen Teacher domein object, de Teacher tabel bevat alleen een userid en daar was dus geen domein voor nodig. Het ophalen van of iemand een Teacher is, gebruik TeacherDao.findByUserId;
* Bij InviteKey is de naam in de database “createdAt” maar in het domein “dateTime”;
* Bij Projects wordt de userId niet gebruikt in de ProjectDto;
* Bij Tasks wordt de “createdAt” niet gebruikt in de TaskDto;
* De StudentDto bevat informatie uit de Students tabel, maar voegt ook informatie toe vanuit de Users en StudentSkills tabel toe. Om deze reden is er ook geen StudentSkills domein;
* De GetSkillDto bevat niet de “createdAt” informatie die in de Skills tabel staat;
* De TaskSkills tabel wordt gecombineerd met Tasks in de “TaskWithSkills” domeinobject. Dit domein bevat ook informatie over TaskRegistrations, namelijk hoeveel gebruikers er aangemeld zijn en hoeveel geaccepteerd zijn;
* Bij TaskRegistrations wordt de “createdAt” niet gebruikt in de GetRegistrationDto.

## Integraties en externe gegevensbronnen

* Het systeem maakt gebruik van OAuth. Het gebruikt hiervoor de spring-security-oauth2 library. Deze library zorgt voor de datastroom, hier hebben wij geen invloed op. Wanneer de gebruiker wil inloggen, stuurt de library de gebruiker naar de gekozen provider. Na het hele proces van inloggen krijgt de applicatie een id, naam, avatar en email terug. Dit is te zien in de code in authentication.Provider.
* Het toevoegen van een nieuwe provider doe je door van die provider een client-id en client-secret te verkrijgen. In het geval van Google, GitHub, facebook, okta en azure is dit het enige wat je nodig hebt. Voeg deze toe aan de applicatie volgens de [README.md](https://github.com/AIM-CNP-sep24/project-cnp-pittige-pinguins/blob/development/README.md).
  + Voor de overige providers heb je ook nog de scope, authorization-uri, token-uri, user-info-uri, user-name, client-authentication-method en redirect-uri-template nodig.
  + Voeg hierna aan de authentication.Provider de provider toe in de switch en pas de attributes aan naar wat de documentatie van de provider zegt:  
    Afbeelding met tekst, schermopname, software, Multimediasoftware

    Automatisch gegenereerde beschrijving

Figuur : Toevoegen nieuwe provider

## Overige opmerkingen

Er worden geen tekstbestanden of overige bestanden gebruikt voor het opslaan van data. Alle data staan in de database. De enige bestanden in het project die over data gaan, zijn de sql bestanden: schema.sql en testdata.sql. schema.sql is verplicht voor de werking van de applicatie, deze zorgt ervoor dat de juiste tables er zijn. testdata.sql zorgt ervoor dat er tijdens development al wat data aanwezig is om mee te testen. Het is aangeraden om tijdens development deze testdata in de database te laten staan.

Afbeeldingen worden in de static/files/ map opgeslagen, zodat de backend deze automatisch op de webserver toont. De referentie hiernaar wordt opgeslagen in de database.

We hebben met de PO afgesproken dat we met wettelijke verplichtingen geen rekening houden voor nu. Bijvoorbeeld het verwijderen van een account is op dit moment niet mogelijk. Er is ook geen rekening gehouden met back-up strategieën omdat er geen productieserver/-database aanwezig is.

# Deployment

In dit hoofdstuk wordt weergegeven hoe het systeem wordt geïnstalleerd en uitgevoerd in development.

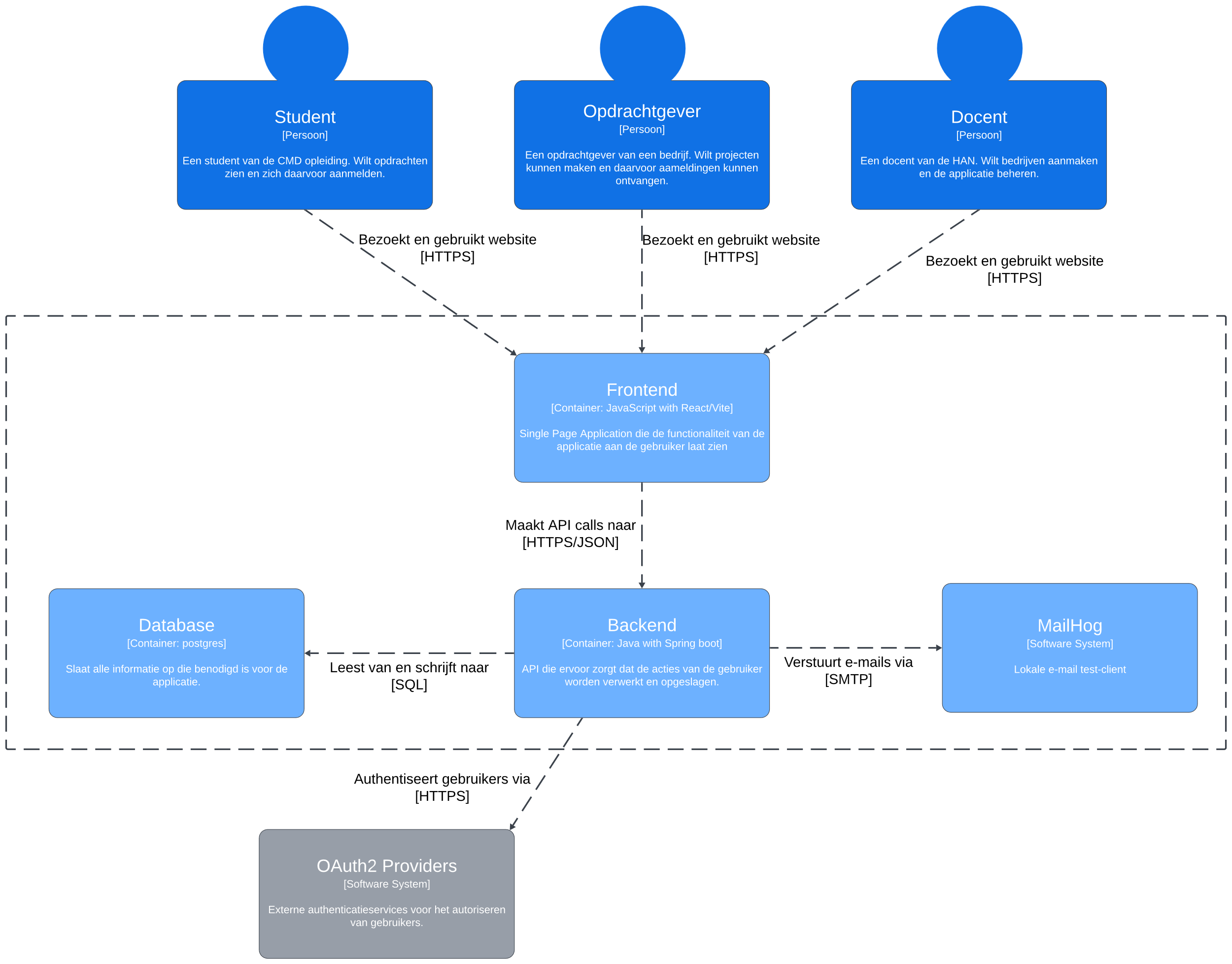
## Infrastructuur

De applicatie is verdeeld in vier onderdelen. Dit zijn de volgende:

* Frontend: JavaScript met React/vite;
* Backend: Java met Spring boot;
* Database: Postgres;
* Mailservice: Mailhog.

Alle vier de onderdelen staan in een Docker container. Dit zorgt ervoor dat je de applicatie kan uitvoeren met één commando (zie voor specifieke installatie instructies de [README.md op GitHub](https://github.com/AIM-CNP-sep24/project-cnp-pittige-pinguins/blob/development/README.md)). Daarnaast zorgt dit ervoor dat de fysieke infrastructuur niet uitmaakt, want Docker containers zijn vergelijkbaar met virtual machines: de hardware maakt niet uit.

Zie hieronder de containers (binnen de stippellijnen) en hoe die communiceren met de buitenwereld:



Figuur : Container Diagram (2e onderdeel van C4)

De MailHog container staat in dit diagram binnen de applicatie, omdat het een container is die binnen de applicatie wordt gedraaid. Het is een extern programma waarop wij geen invloed hebben op de kwaliteit of veiligheid, vandaar dat dit bij 1.1 als extern programma staat. Wanneer de applicatie in productie wordt genomen, is het de bedoeling dat MailHog wordt weggehaald als container en dat de e-mails naar de echte mail providers worden verstuurd.

Voor zowel de frontend als de backend geldt dat ze automatisch gecompileerd worden wanneer docker wordt opgestart. Na het compileren worden ze opgestart. De database wordt ook opgestart en de tabellen worden automatisch aangemaakt.

Voor het deployen op een server is het mogelijk het allemaal op één server te zetten, maar over het algemeen is het aangeraden om de database op een aparte server te zetten. De frontend kan ook met een Content Delivery Network (CDN) gehost worden.

De backend moet fysiek in de buurt zijn van de database. Als de database en de backend in verschillende regio’s staan, kunnen er grote vertragingen ontstaan omdat er veel database queries gebeuren. Als deze elke keer oplopen kan een reactie meerdere seconden duren.

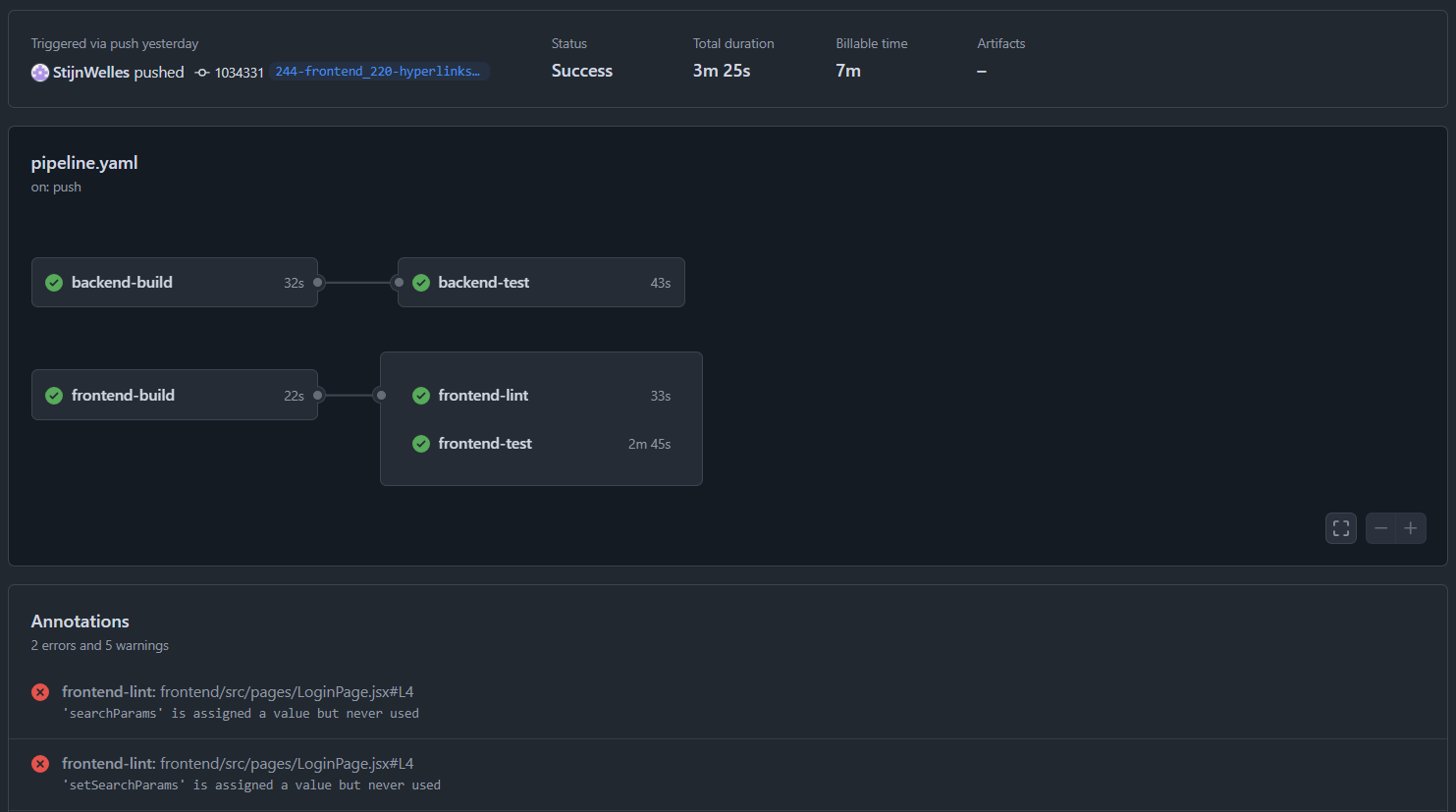
## Resourcebeheer

Resourcebeheer is niet van toepassing op dit project, omdat Docker dit allemaal vanzelf regelt. Er zijn geen servers aanwezig als fallback.

## CI/CD-pipeline

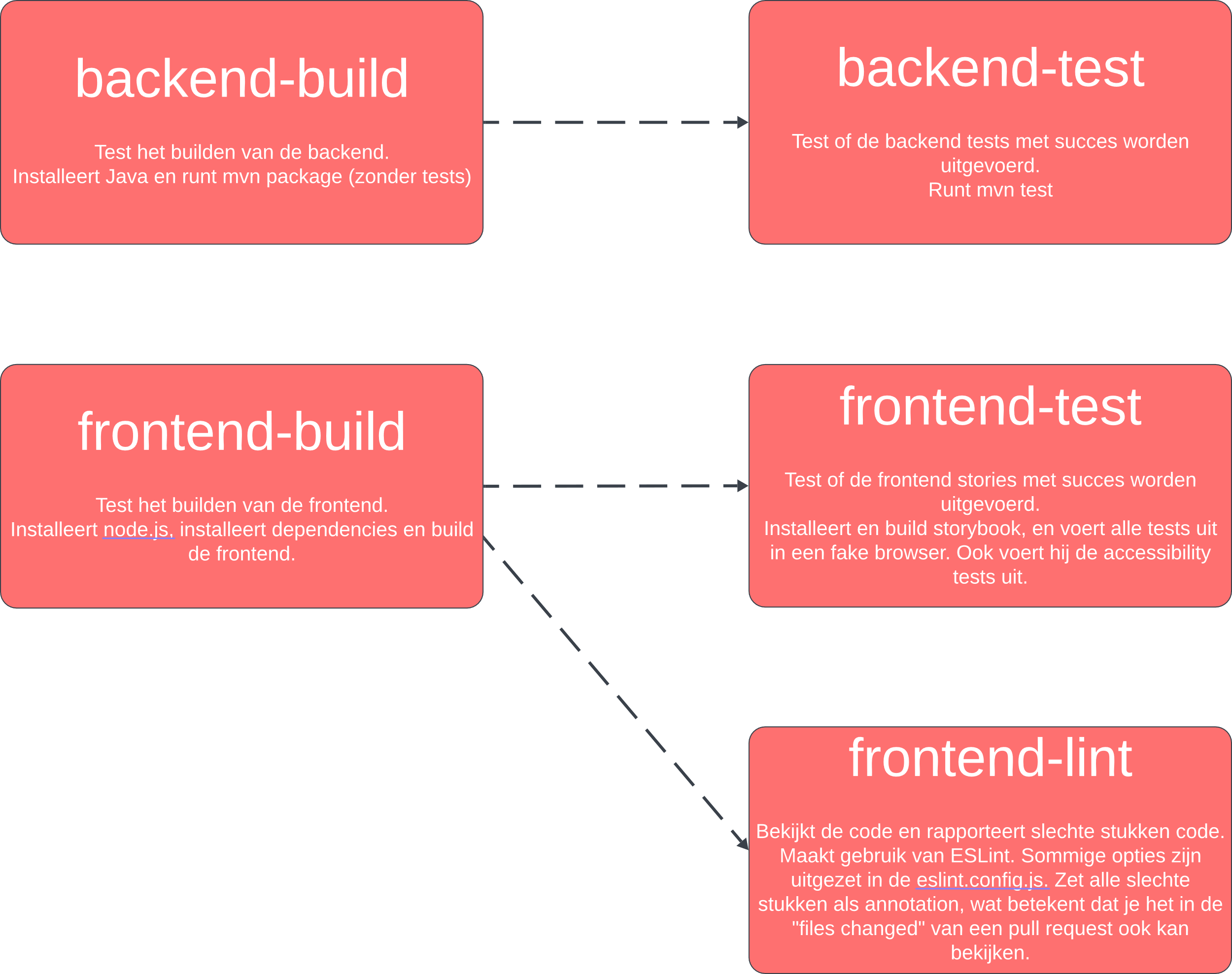
Om ervoor te zorgen dat nieuwe code van de gewenste kwaliteit is, is er een CI/CD pipeline ingesteld.

Op dit moment is het deployment onderdeel van de CI/CD niet gemaakt. Volgens de PO is dit niet nodig. De pipeline ziet er in GitHub als volgt uit:



Figuur : Screenshot van een pipeline uitvoering

Zoals hierboven te zien is bestaat deze pipeline uit verschillende stappen. Deze stappen zorgen er samen voor dat er geen niet-werkende code op development komt te staan. In onderstaand figuur staat per stap beschreven wat er getest wordt:



Figuur : CI/CD stappen uitgelegd

Afbeelding met tekst, software, Multimediasoftware, schermopname

Automatisch gegenereerde beschrijving

Figuur : Voorbeeld van hoe een ESLint annotation eruitziet bij de ‘files changed’ bij een pull request

Wij gebruiken een onderdeel van de Gitflow strategie als het gaat om branches. Dit werkt als volgt:

* Main (protected, minimaal 2 reviewers voor PR): deze branch wordt tijdens de sprint niet aangepast. Development wordt naar main gemerged na goedkeuring van de PO na de sprint review.
* Development (protected, minimaal 2 reviewers): dit is de branch waar alle nieuwe code terecht komt tijdens de sprint. Deze branch moet altijd werkende code hebben. Het is belangrijk dat de branches hieronder altijd up-to-date zijn met development om onnodige merge conflicts te voorkomen.
* Feature: Voor elke taak wordt er een feature branch aangemaakt. Er is gekozen om dit per taak te doen, zodat branches niet te ver gaat achterlopen op development. Nadat een feature branch gemerged is, is de bijhorende taak af en wordt de branch automatisch verwijderd. Het is de bedoeling dat er op een branch alleen wordt gewerkt aan de gelinkte taak. Voor andere wijzigingen dien je een nieuwe taak aan te maken, met bijbehorende branch.

Wanneer er iets fout gaat op de main, reverten we die commit via een nieuwe pull request.

## Foutafhandeling en beschikbaarheid

Bij een software- of infrastructuurfout gebeurt er niets, aangezien er geen productieserver bestaat. Data wordt ook niet gerepliceerd over verschillende locaties omdat er ook geen productiedatabase is. Er zijn ook geen hersteltijden gedefinieerd en wordt er geen actie ondernomen om downtime te minimaliseren.

## Configuratie en beheer

In het begin van de applicatie moet er één docent in het systeem staan. Vanuit daar kan die docent andere docenten toevoegen. Die docenten kunnen ook bedrijven aanmaken en opdrachtgevers toevoegen. Studenten kunnen zichzelf toevoegen via OAuth.

Docenten kunnen de applicatie ook beheren via de beheer pagina, zoals het goedkeuren van skills.

Configuratiewijzigingen, zoals OAuth, databaseinstellingen en het frontend adres zijn aan te passen in de application.properties.

