THOMAS MORE

Realisatie quizapplicatie Qurio

Lykios

Jan-Peter Dhallé
Amber Vranckx
Studenten Bachelor in de Toegepaste Informatica – Applicatieontwikkeling

2024 -

Inhoud

Lijst met figuren

Lijst met tabellen

Inleiding

Dit realisatiedocument omvat de uitwerking en realisaties die tijdens onze stage uitgevoerd zijn. Zoals in het Project Plan beschreven staat, gaat het om de uitwerking van een quizapplicatie.

In dit document wordt eerst de analyse van onze opdracht besproken aan de hand van een datamodel, usecasediagram en de gerealiseerde prototypes. Verder vermelden we de gebruikte tools en technologieën die we doorheen onze stageperiode gebruikt hebben. Vervolgens wordt het eindresultaat van onze stageopdracht besproken. Hierbij wordt eerst toegelicht hoe de setup van het project is gebeurd. Nadien bespreken we alle uitgewerkte functionaliteiten. Tot slot volgt er nog een besluit en een literatuurlijst.

1. Analyse

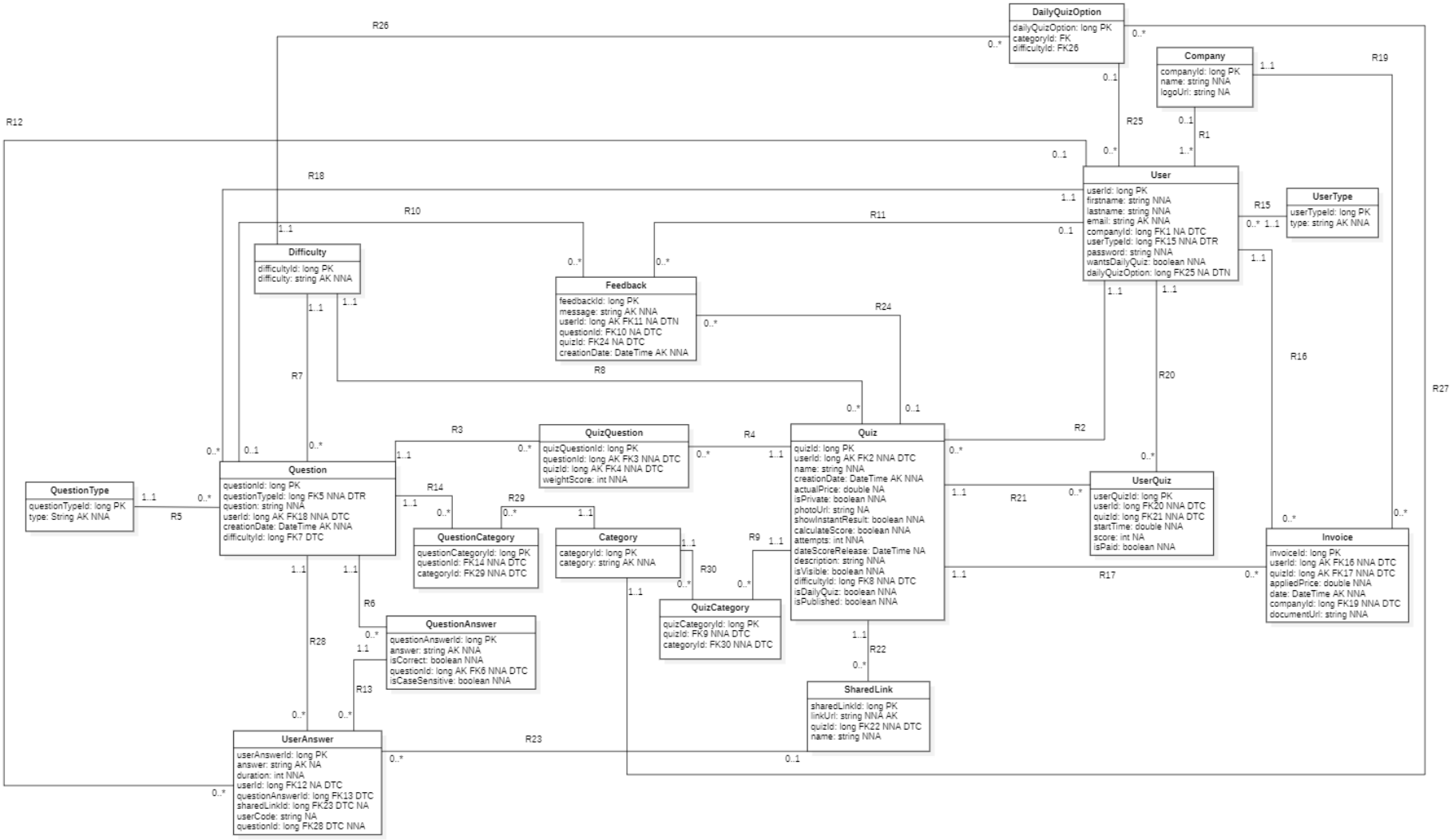
Hieronder wordt onze analyse toegelicht die we tijdens de eerste week van onze stage hebben gemaakt. Vervolgens worden de tools opgesomd waar we doorheen onze stage een beroep op hebben gedaan. Tot slot volgt een lijst van technologieën die we hebben gebruikt.

1.1. Opdracht

In dit onderdeel wordt de eerste analyse van de opdracht besproken. Aangezien er wordt gewerkt volgens de Scrum-methodiek, is deze analyse nog verder uitgewerkt en aangepast tijdens onze sprints. Hieronder wordt eerst het datamodel besproken. Hier wordt zowel de eerste versie als het eindresultaat getoond. Nadien volgt het usecasediagram met de bijhorende usecasebeschrijvingen.

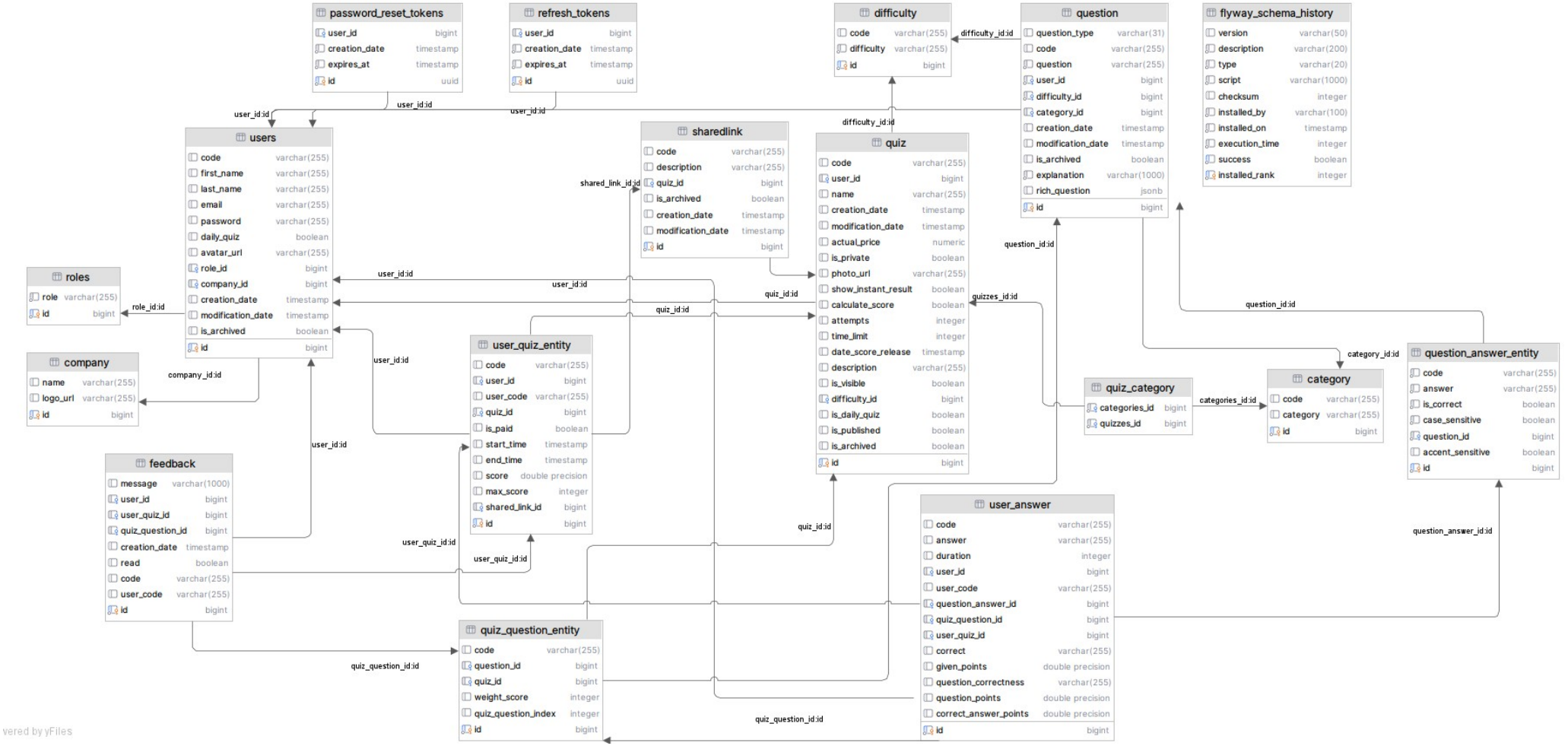
1.1.1. Datamodel

De eerste versie van ons datamodel werd gemaakt in StarUML. Het bestaat uit verschillende entiteiten. De grootste entiteiten zijn onder andere gebruiker, vraag en quiz.



Figuur 1: Eerste versie datamodel

Dit is ons uiteindelijke datamodel. Dit datamodel hebben we gegenereerd vanuit *IntelliJ IDEA.* Hier en daar is er een entiteit verschillend ten opzichte van onze eerste versie, maar over het algemeen zijn de belangrijkste entiteiten wel consistent.



Figuur 2: Eindresultaat datamodel

1.1.2. Usecasediagram

In ons usecasediagram zijn er vijf verschillende soorten gebruikers gedefinieerd met elk verschillende rechten.

Figuur 3: Usecasediagram

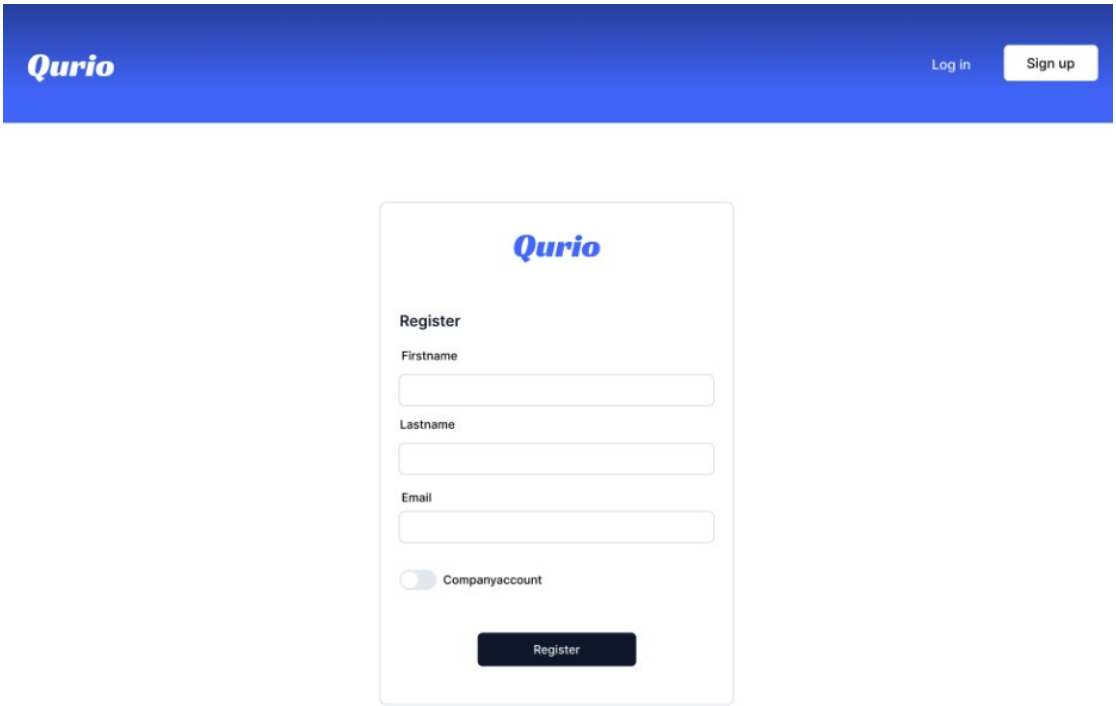
1.1.3. Usecasebeschrijvingen

Hieronder wordt voor elke usecase de usecasebeschrijving getoond aan de hand van de functionaliteit, de preconditie, het normaal verloop en de eventuele alternatieven. Bij de usecasebeschrijvingen worden de bijhorende prototypes getoond die werden gemaakt in Figma. Hier werd veel tijd aan gespendeerd zodat de styling tijdens het ontwikkelen van de applicatie vlotter zou verlopen.

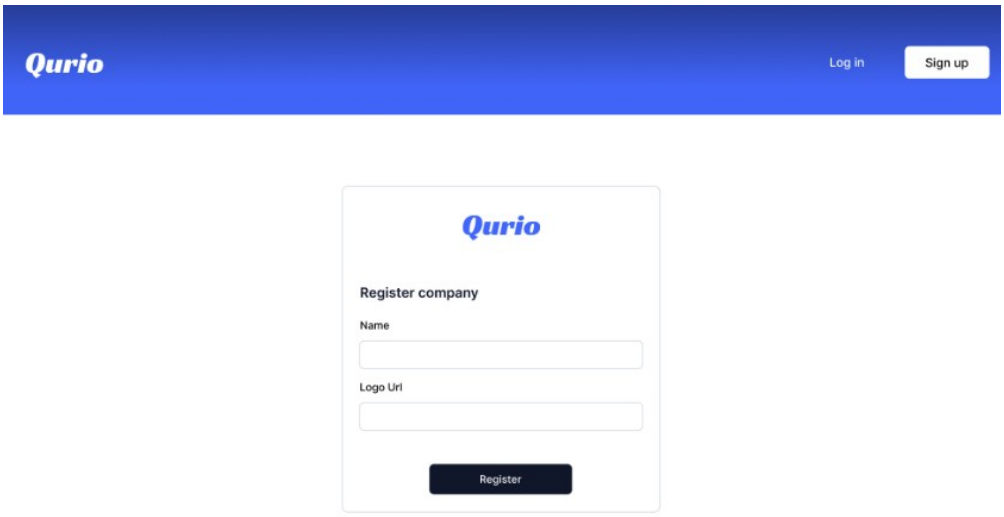
1.1.3.1. Algemene gebruiker

REGISTREREN

Functionaliteit: Als een algemene gebruiker, kan ik mezelf registreren
Normaal verloop: De actor vult zijn gegevens zoals: voornaam, achternaam en emailadres in en bevestigt deze.
Alternatieven:
Ontbrekende gegevens: Het systeem toont een foutmelding. De actor vult de ontbrekende gegevens in.
Gegevens zijn niet correct: Het systeem toont een foutmelding. De actor vult de foutieve gegevens correct aan.



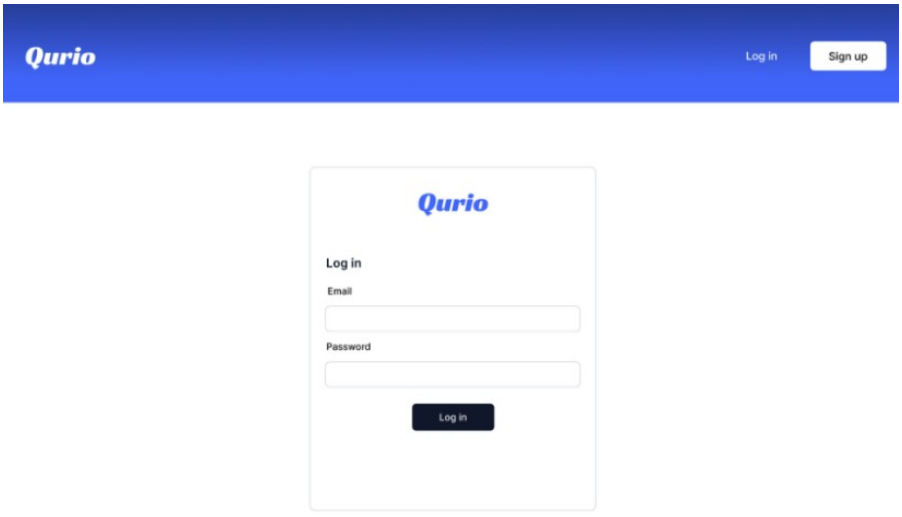
Figuur 4: Prototype registreren gebruiker



Figuur 5: Prototype registreren voor bedrijven

INLOGGEN

Functionaliteit: Als een algemene gebruiker, kan ik inloggen
Normaal verloop: De actor vult zijn emailadres en wachtwoord in. Het systeem controleert deze en toont het dashboard.
Alternatieven:
Ontbrekende gegevens: Het systeem toont een foutmelding. De actor vult de ontbrekende gegevens in.
Foutief wachtwoord: Het systeem toont een foutmelding. De actor vult zijn wachtwoord opnieuw in.



Figuur 6: Prototype login

QUIZ INVULLEN

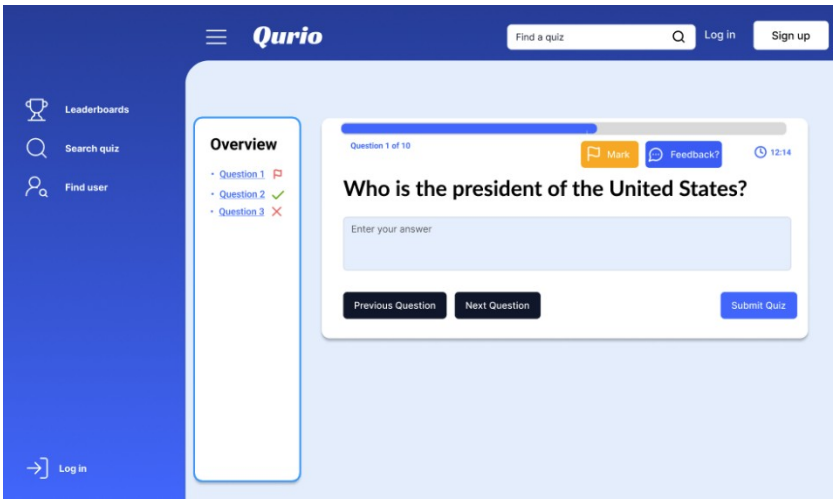
Functionaliteit: Als een algemene gebruiker, kan ik een quiz invullen.
Preconditie: Er is een quiz geopend.

Normaal verloop: De actor vult zijn antwoorden op de vragen in en verzendt deze.

Alternatieven:

Vraag markeren: Als de actor ervoor kiest om een vraag te markeren, switch het systeem naar de usecase “Vraag markeren”.

Antwoorden bekijken: Als de actor ervoor kiest om de antwoorden van een quiz te bekijken, switcht het systeem naar de usecase “Antwoorden bekijken”.



Figuur 7: Prototype quiz invullen

VRAAG MARKEREN

Functionaliteit: Als een algemene gebruiker, kan ik een vraag markeren.

Preconditie: Er is een quiz geopend.

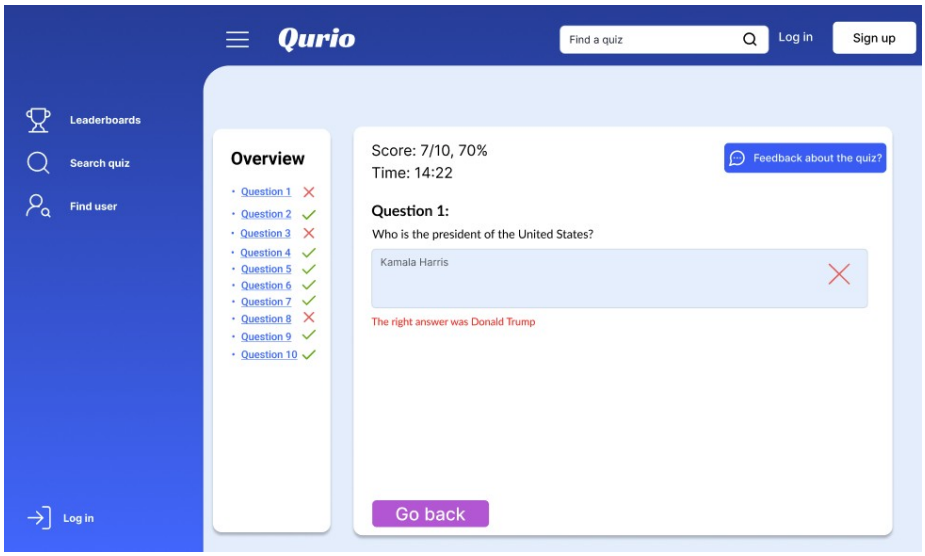
Normaal verloop: De actor duidt de vraag aan die hij later wil herbekijken. Het systeem markeert deze vraag.

ANTWOORDEN BEKIJKEN

Functionaliteit: Als een algemene gebruiker, kan ik de antwoorden van een quiz bekijken.

Preconditie: De quiz is ingediend.

Normaal verloop: Het systeem toont de juiste antwoorden.



Figuur 8: Prototype antwoorden bekijken

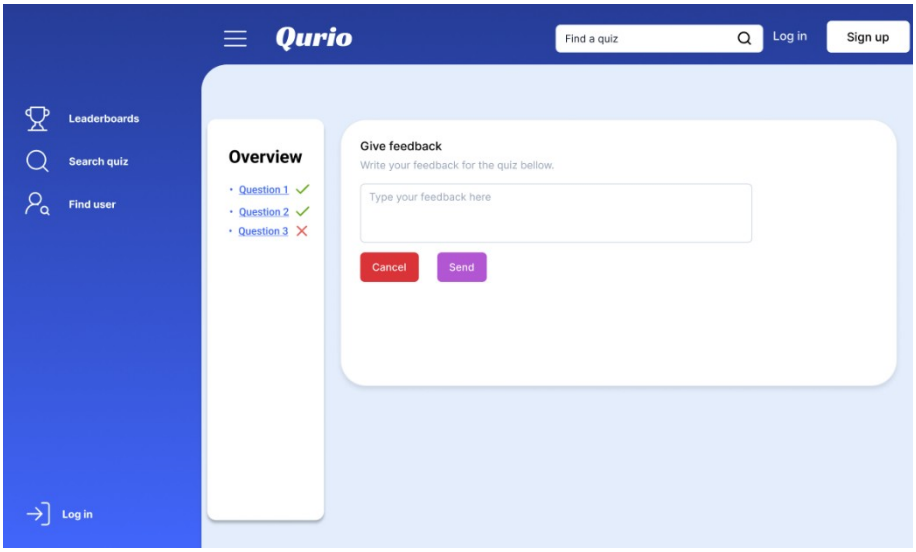
FEEDBACK VOOR QUIZ GEVEN

Functionaliteit: Als een algemene gebruiker, kan ik feedback voor een quiz geven.

Normaal verloop: De actor vult zijn feedback in en bevestigt deze.

Alternatieven:

Ontbrekende gegevens: Het systeem toont een foutmelding. De actor vult de ontbrekende gegevens in.



Figuur 9: Prototype feedback over quiz geven

FEEDBACK VOOR VRAAG GEVEN

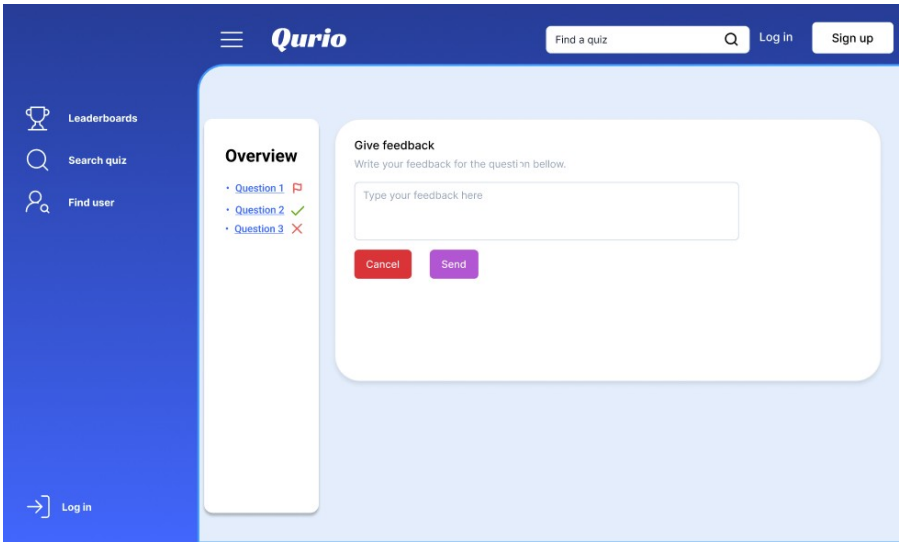
Functionaliteit: Als een algemene gebruiker, kan ik feedback voor een vraag geven.

Preconditie: Er is een quiz geopend en er is een vraag geselecteerd.

Normaal verloop: De actor vult zijn feedback in en bevestigt deze.

Alternatieven:

Ontbrekende gegevens: Het systeem toont een foutmelding. De actor vult de ontbrekende gegevens in.



Figuur 10: Prototype feedback over vraag geven

LEADERBOARD PER QUIZ BEKIJKEN

Functionaliteit: Als een algemene gebruiker, kan ik het leaderboard van een quiz bekijken.

Preconditie: De algemene gebruiker heeft een quiz geselecteerd.

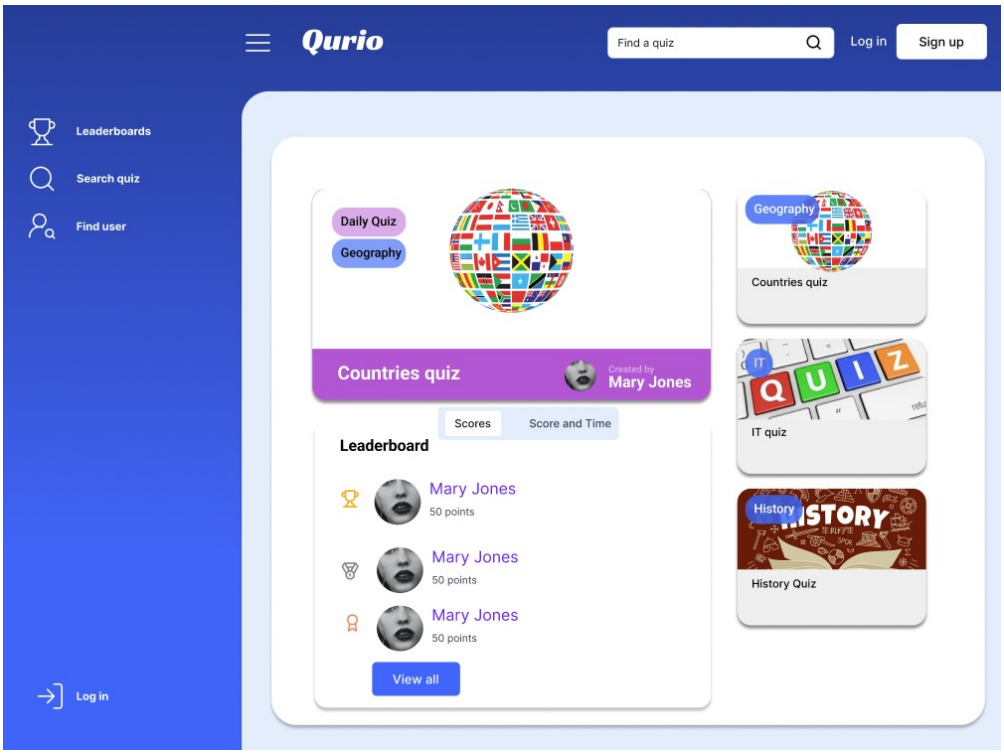
Normaal verloop: Het systeem toont het leaderboard van de geselecteerde quiz.

Figuur 11: Prototype startpagina quiz

ALGEMEEN LEADERBOARD BEKIJKEN

Functionaliteit: Als een algemene gebruiker, kan ik het algemene leaderboard opvragen.

Normaal verloop: Het systeem toont het algemene leaderboard.

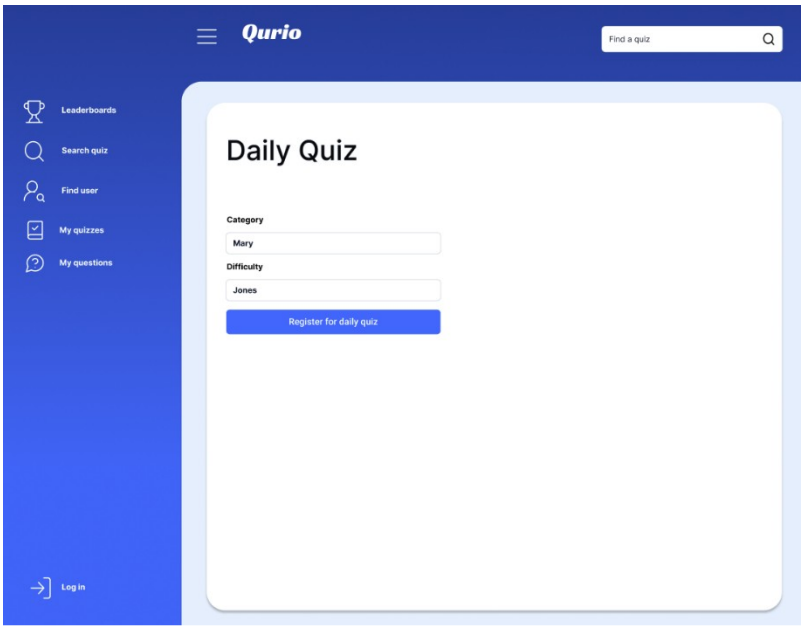


Figuur 12: Prototype algemeen leaderbord

1.1.3.2. Privégebruiker

INSCHRIJVEN VOOR KORTE DAGELIJKSE QUIZ

Functionaliteit: Als een privégebruiker, kan ik mij inschrijven voor een korte dagelijkse quiz.
Preconditie: De privégebruiker is ingelogd.
Normaal verloop: De actor kiest een categorie en een moeilijkheidsgraad.



Figuur 13: Prototype inschrijven voor dagelijkse quiz

HISTORIEK BEKIJKEN

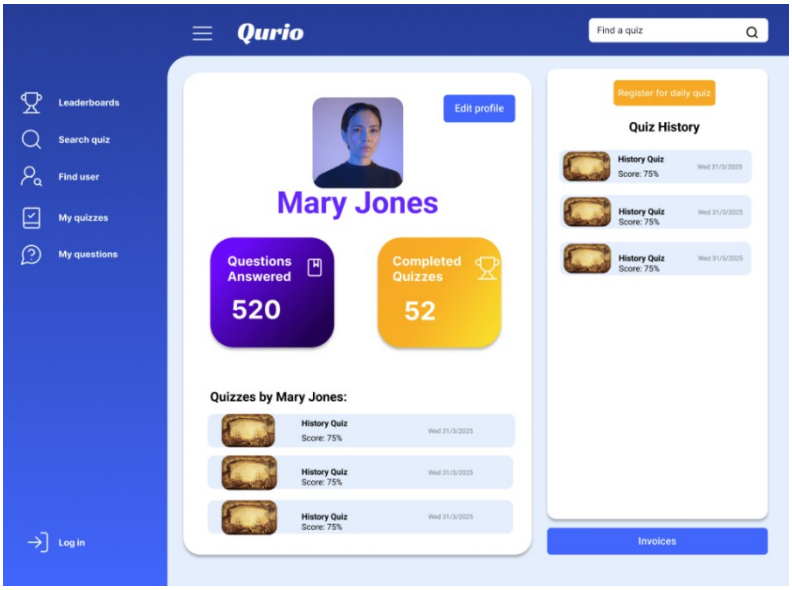
Functionaliteit: Als een privégebruiker, kan ik mijn historiek bekijken.

Preconditie: De privégebruiker is ingelogd.

Normaal verloop: Het systeem toont de quizhistoriek van de actor.

Alternatieven:

Nog geen historiek aanwezig: Het systeem toont een melding dat er nog geen quizhistoriek aanwezig is.



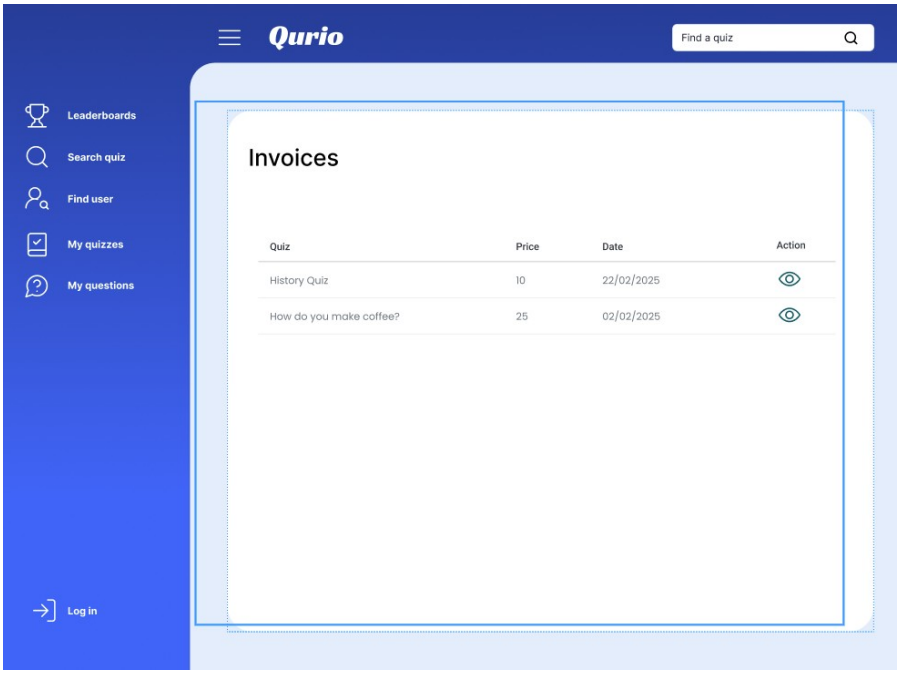
Figuur 14: Prototype profiel

FACTUREN BEKIJKEN

Functionaliteit: Als een privégebruiker, kan ik mijn facturen bekijken.

Preconditie: De privégebruiker is ingelogd.

Normaal verloop: Het systeem toont de alle facturen van de actor.



Figuur 15: Prototype facturen bekijken

BETALENDE QUIZ INVULLEN

Functionaliteit: Als een privégebruiker, kan ik een betalende quiz invullen.

Preconditie: De privégebruiker is ingelogd en heeft een quiz gekozen.

Normaal verloop: Het systeem vraagt bevestiging aan de actor omdat het een betalende quiz is. De actor gaat hiermee akkoord en vult zijn antwoorden op de vragen in en verzendt deze. Het systeem controleert deze antwoorden en toont een score en stuurt een email met de factuur naar de actor.

Alternatieven:

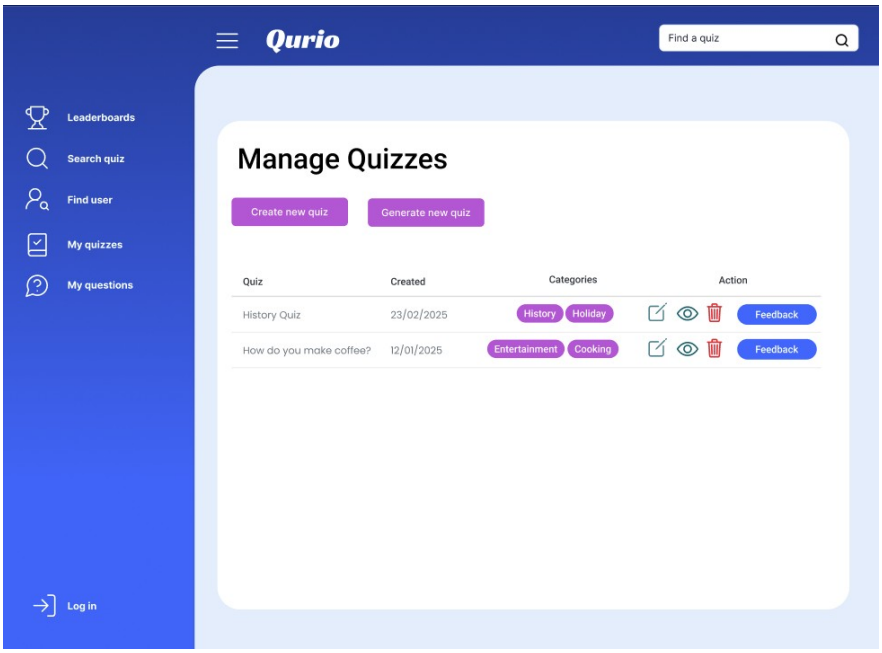
Actor gaat niet akkoord: Het systeem toont opnieuw alle quizzen.

QUIZZEN BEHEREN

Functionaliteit: Als een privégebruiker, kan ik mijn quizzen beheren.

Preconditie: De privégebruiker is ingelogd.

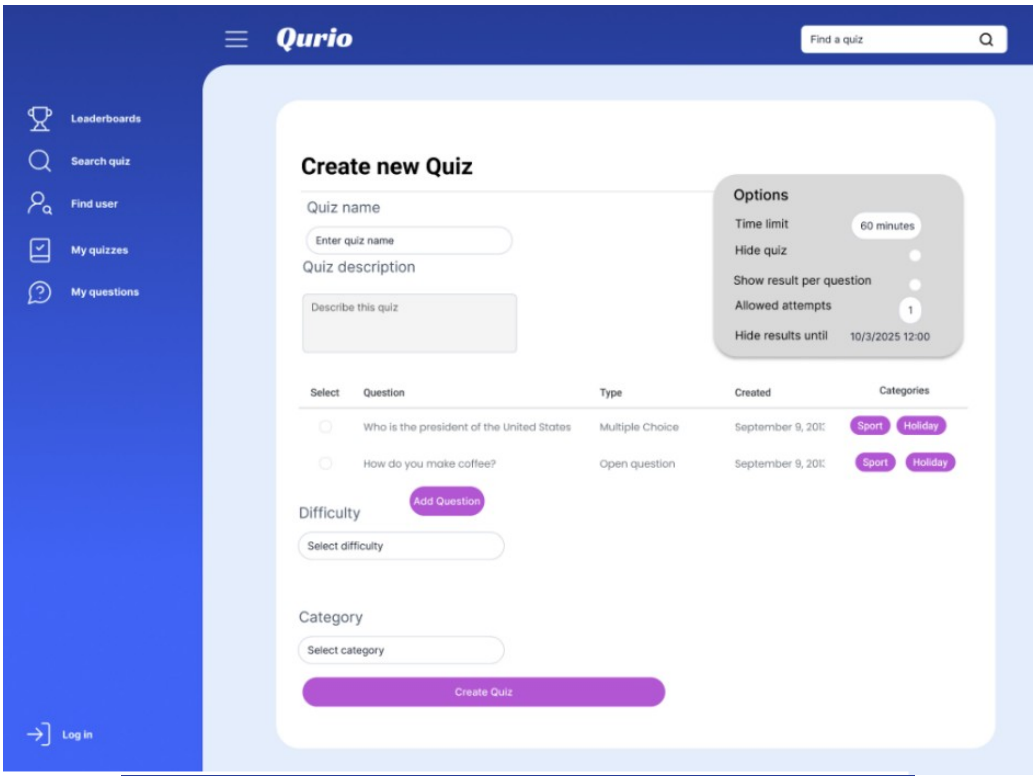
Normaal verloop: Het systeem toont een lijst van alle quizzen van de actor. De actor selecteert een quiz. Het systeem switcht naar de usecase “Quiz beheren”.



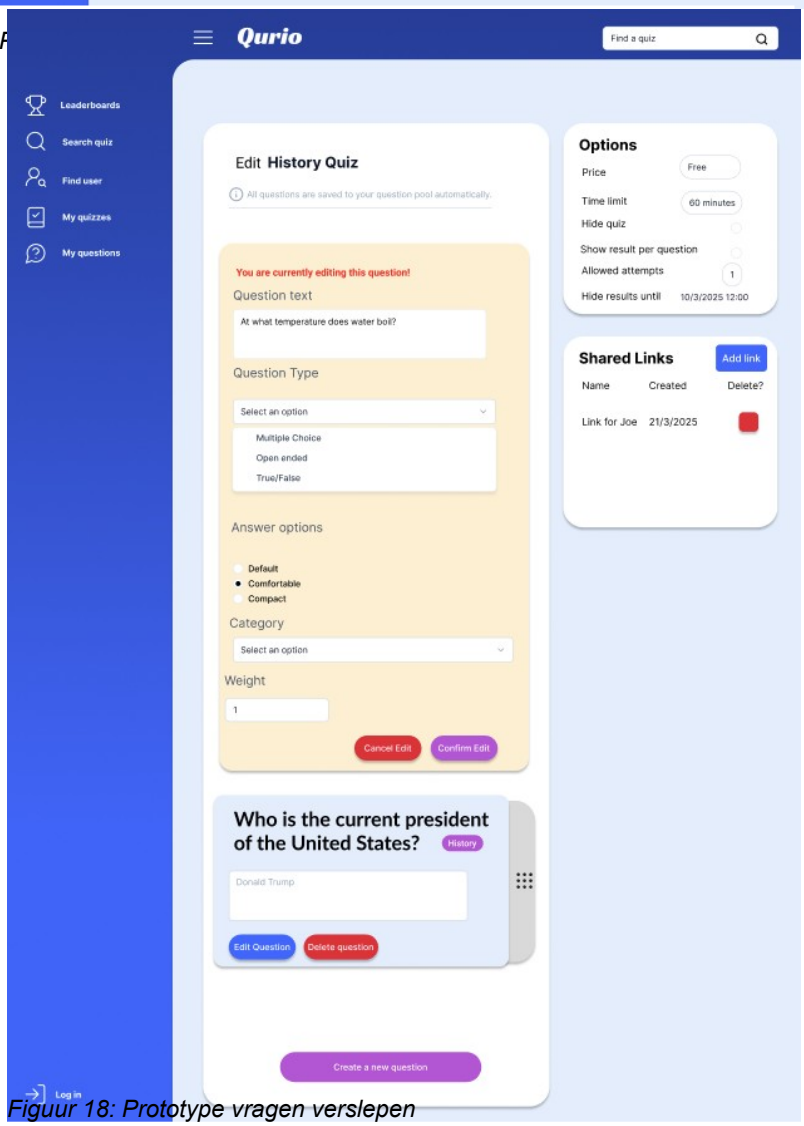
Figuur 16: Prototype quizzen bekijken

1.1.4. Quiz beheren

Functionaliteit: Als een privégebruiker, kan ik een quiz beheren.
Preconditie: De privégebruiker is ingelogd en heeft een quiz geselecteerd.
Normaal verloop: Het systeem toont een lijst met alle beschikbare vragen. De actor kiest een vraag uit de lijst of kiest om een vraag toe te voegen en vult het type, de categorie, de inhoud, de opties, het gewicht van de score en antwoord(en) in en bevestigt dit. Het systeem toont een lijst van alle vragen van de geselecteerde quiz inclusief de nieuwe vraag.
Alternatieven:
Opties wijzigen: Als de actor ervoor kiest om de opties te wijzigen, switcht het systeem naar de usecase “Opties quiz beheren”.
Wijzigen: De actor kiest om een vraag van de geselecteerde quiz te wijzigen.
Archiveren: De actor kiest om een vraag van de geselecteerde quiz te verwijderen. Het systeem vraagt een bevestiging aan de actor. De actor bevestigt dit. Het systeem archiveert de vraag van de geselecteerde quiz.



Figuur 17: P



Figuur 18: Prototype vragen verslepen

OPTIES QUIZ BEHEREN

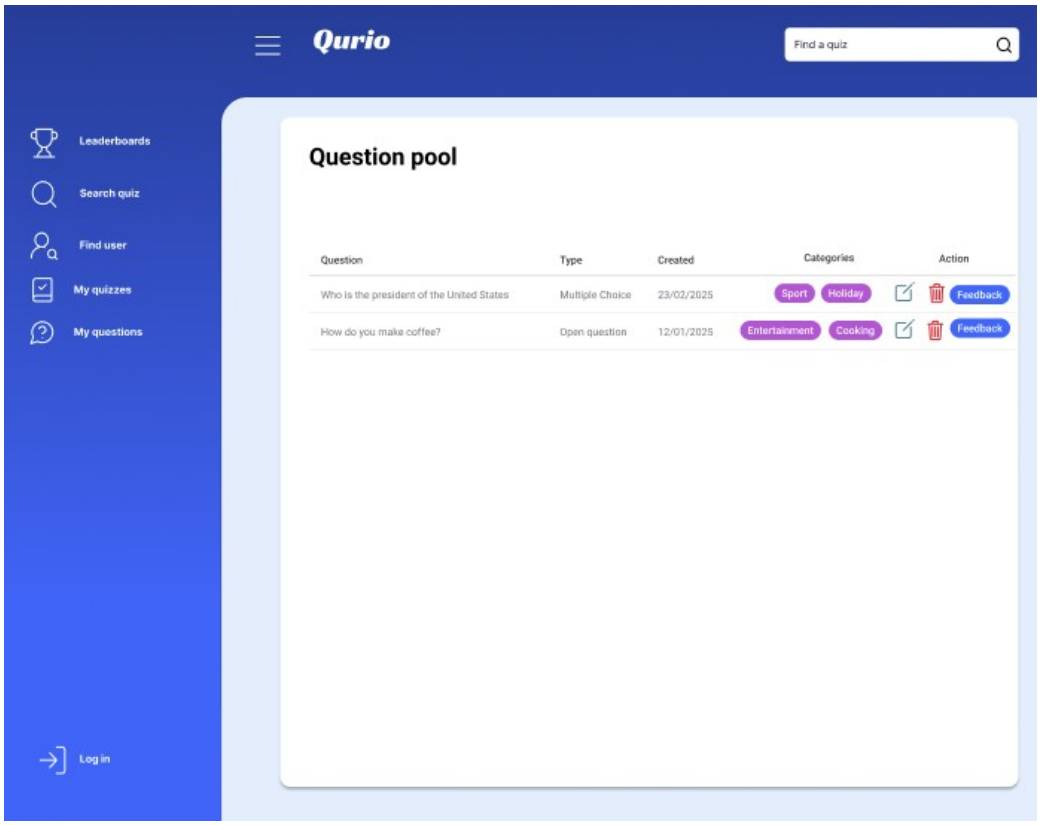
Functionaliteit: Als een privégebruiker, kan ik de opties van een quiz beheren.

Preconditie: De privégebruiker is ingelogd.

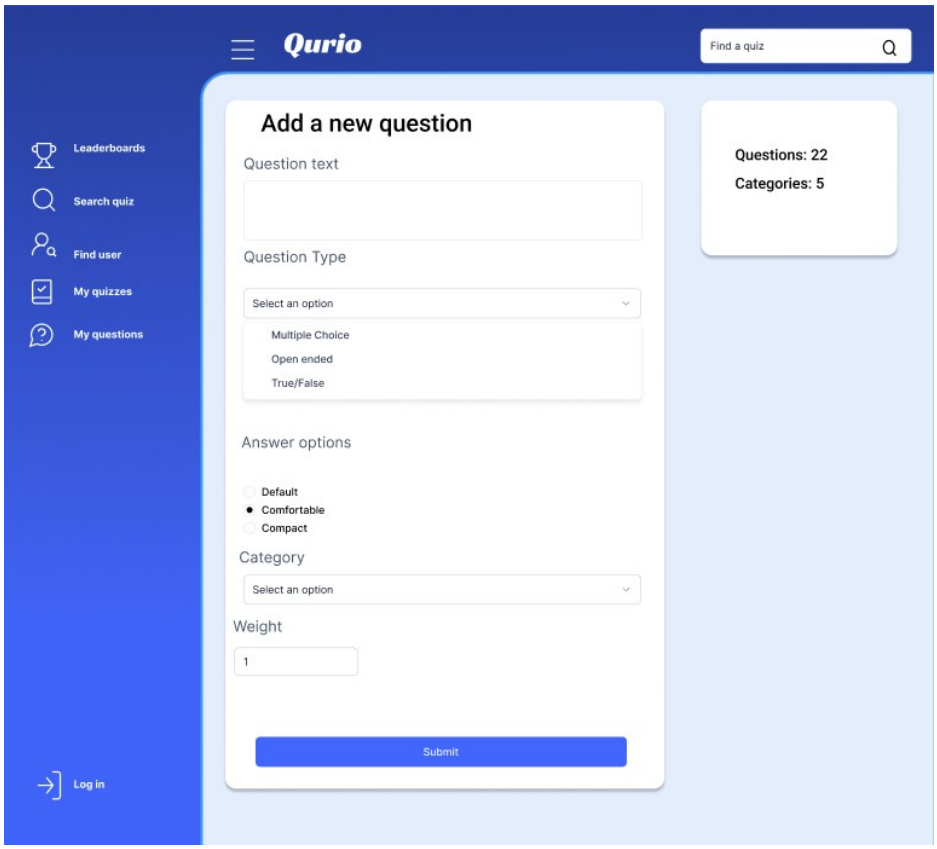
Normaal verloop: Het systeem toont een lijst van alle opties voor de quiz. De actor wijzigt deze opties.

1.1.5. Vragen beheren

Functionaliteit: Als een privégebruiker, kan ik de vragen beheren.
Preconditie: De privégebruiker is ingelogd.
Normaal verloop: Het systeem toont een lijst van alle vragen van de actor. De actor kiest om een vraag toe te voegen en vult het type, de categorie, de inhoud en antwoord(en) in en bevestigt dit. Het systeem toont opnieuw een lijst van alle vragen inclusief de nieuwe vraag.
Alternatieven:
Wijzigen: De actor kiest om een vraag te wijzigen.
Archiveren: De actor kiest om een vraag te archiveren. Het systeem vraagt een bevestiging aan de actor. De actor bevestigt dit. Het systeem archiveert de vraag.



Figuur 19: Prototype vragen bekijken



Figuur 20: Prototype nieuwe vraag toevoegen

LINKS BEHEREN

Functionaliteit: Als een privégebruiker, kan ik deelnemers uitnodigen.
Preconditie: De privégebruiker is ingelogd en heeft een quiz geselecteerd.
Normaal verloop: Het systeem toont een lijst van alle links van de geselecteerde quiz. De actor kiest om een link toe te voegen en vult een naam in en bevestigt dit. Het systeem toont opnieuw een lijst van alle links van de geselecteerde quiz.
Alternatieven:
Verwijderen: De actor kiest om een link te verwijderen. Het systeem vraagt een bevestiging aan de actor. De actor bevestigt dit. Het systeem verwijdert de link.

FEEDBACK QUIZ OPVRAGEN

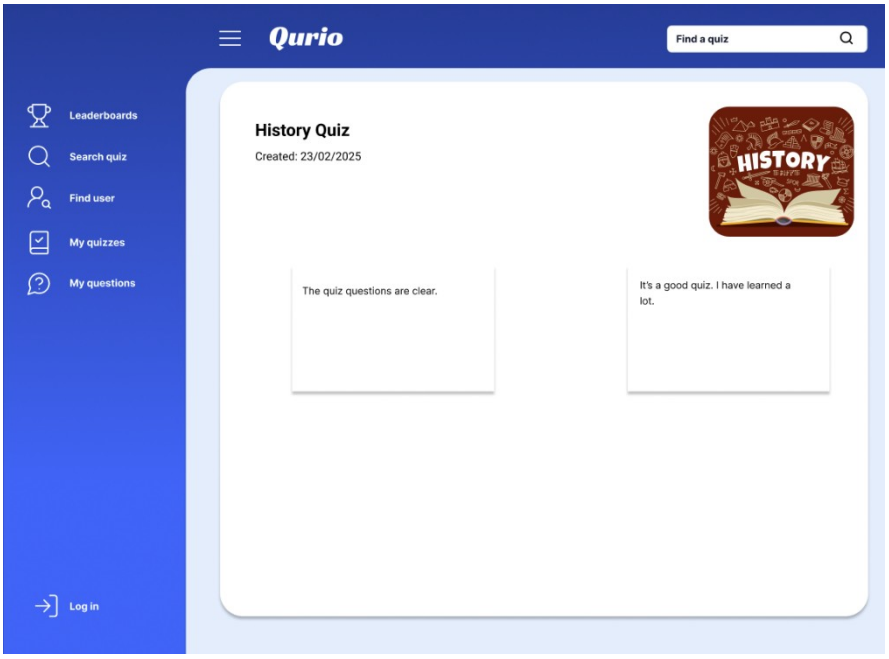
Functionaliteit: Als een privégebruiker, kan ik de feedback voor een quiz opvragen.

Preconditie: De privégebruiker is ingelogd en heeft een quiz geselecteerd.

Normaal verloop: Het systeem toont een lijst van de feedback voor de geselecteerde quiz.

Alternatieven:

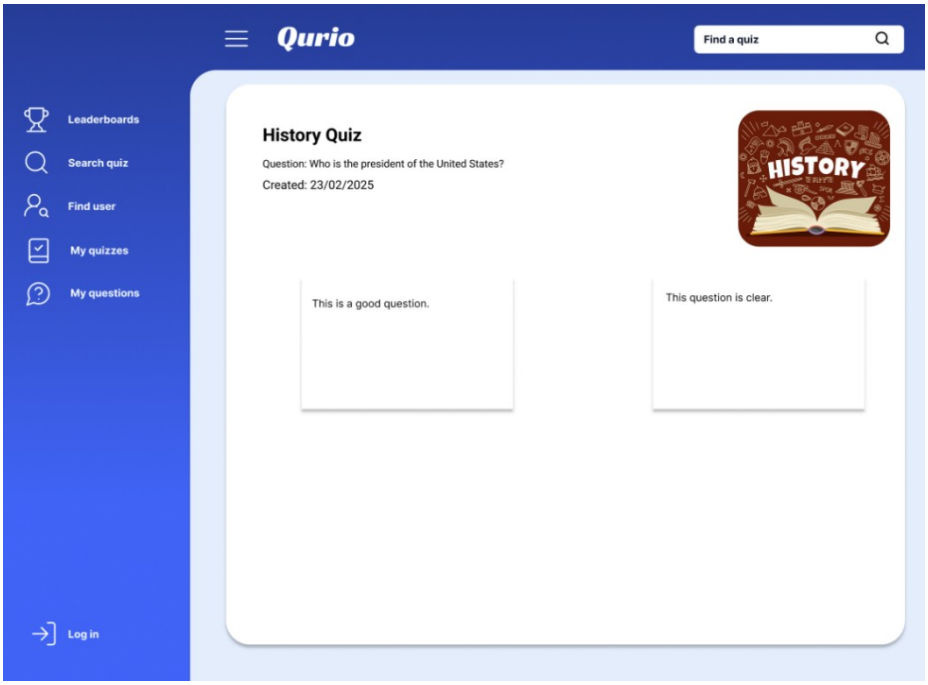
Geen feedback beschikbaar: Het systeem toont een bericht dat er nog geen feedback voor de geselecteerde quiz beschikbaar is.



Figuur 21: Prototype feedback over quiz opvragen

FEEDBACK VRAAG OPVRAGEN

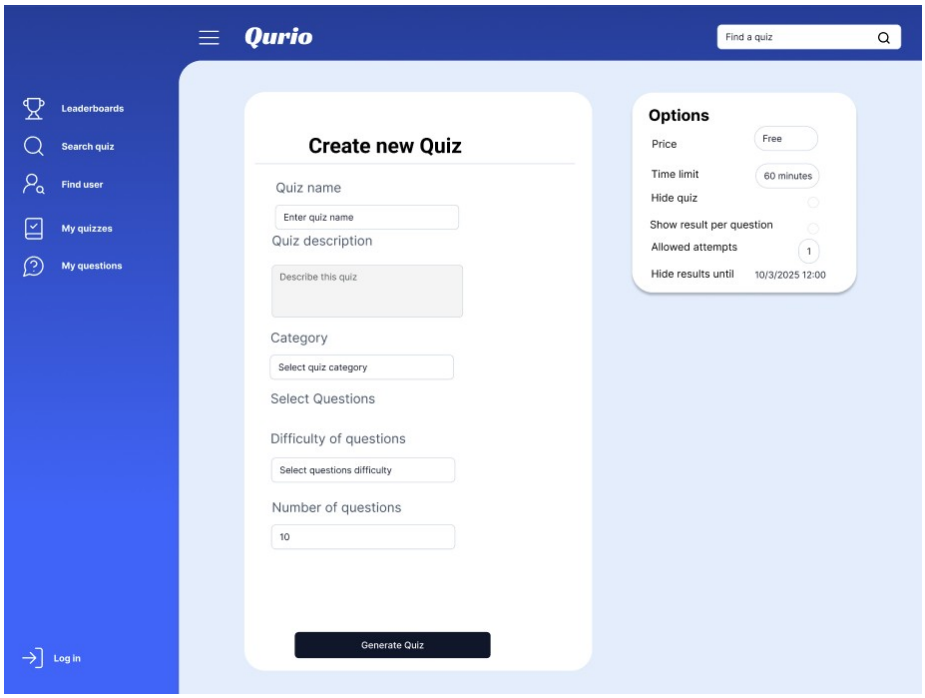
Functionaliteit: Als een privégebruiker, kan ik de feedback voor een vraag opvragen.
Preconditie: De privégebruiker is ingelogd en heeft een vraag geselecteerd.
Normaal verloop: Het systeem toont een lijst van de feedback voor de geselecteerde vraag.
Alternatieven:
Geen feedback beschikbaar: Het systeem toont een bericht dat er nog geen feedback voor de geselecteerde vraag beschikbaar is.



Figuur 22: Prototype feedback over vraag opvragen

QUIZ GENEREREN

Functionaliteit: Als een privégebruiker, kan ik automatisch een quiz genereren.
Preconditie: De privégebruiker is ingelogd.
Normaal verloop: Het systeem toont een lijst van alle beschikbare categorieën. De actor kiest een categorie, een moeilijkheidsgraad en het aantal vragen dat hij wil toevoegen. Het systeem genereert automatisch een quiz met het juiste aantal vragen over de gekozen categorie en met de juiste moeilijkheidsgraad.
Alternatieven:
Te weinig vragen: Het systeem toont een foutmelding dat er te weinig vragen beschikbaar zijn.
Te weinig vragen beschikbaar in categorie: Het systeem toont een foutmelding dat er te weinig vragen voor deze categorie beschikbaar zijn.
Te weinig vragen beschikbaar in moeilijkheidsgraad: Het systeem toont een foutmelding dat er te weinig vragen voor deze moeilijkheidsgraad zijn.



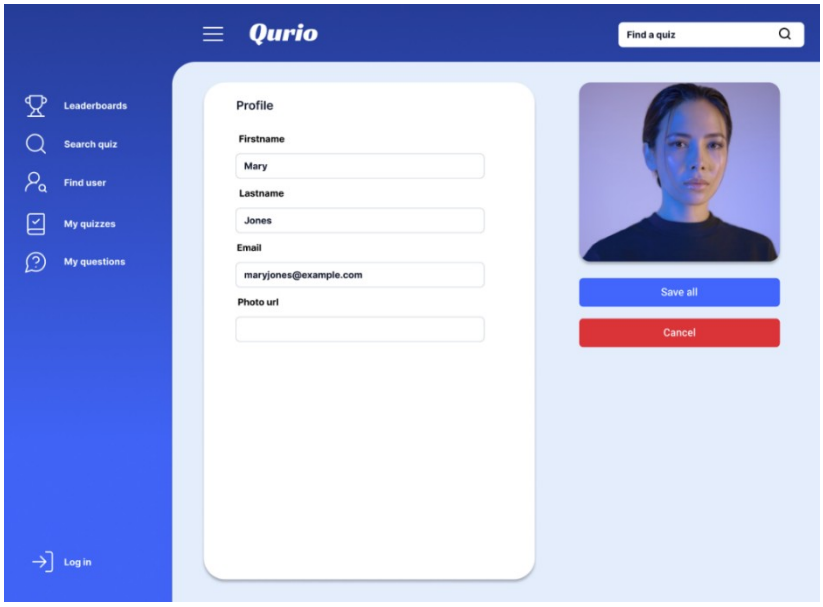
Figuur 23: Prototype quiz genereren

PROFIELGEGEVENS BEWERKEN

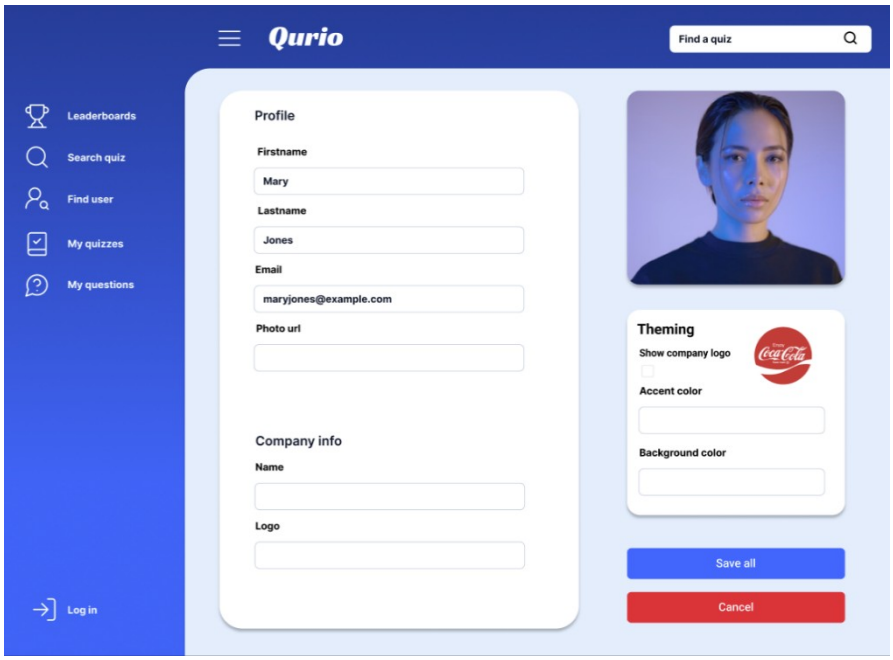
Functionaliteit: Als een privégebruiker, kan ik mijn profielgegevens bewerken.

Preconditie: De privégebruiker is ingelogd.

Normaal verloop: Het systeem toont de huidige profielgegevens van de actor. De actor wijzigt deze gegevens en bevestigt deze.



Figuur 24: Prototype profiel bewerken



Figuur 25: Prototype profiel bewerken als bedrijf

1.1.5.1. Bedrijfsklant

PRIJS BEHEREN

Functionaliteit: Als een bedrijfsklant, kan ik de prijs van een quiz beheren.
Preconditie: De bedrijfsklant is ingelogd en heeft een quiz geselecteerd.
Normaal verloop: Het systeem toont de huidige prijs. De actor past deze prijs aan.

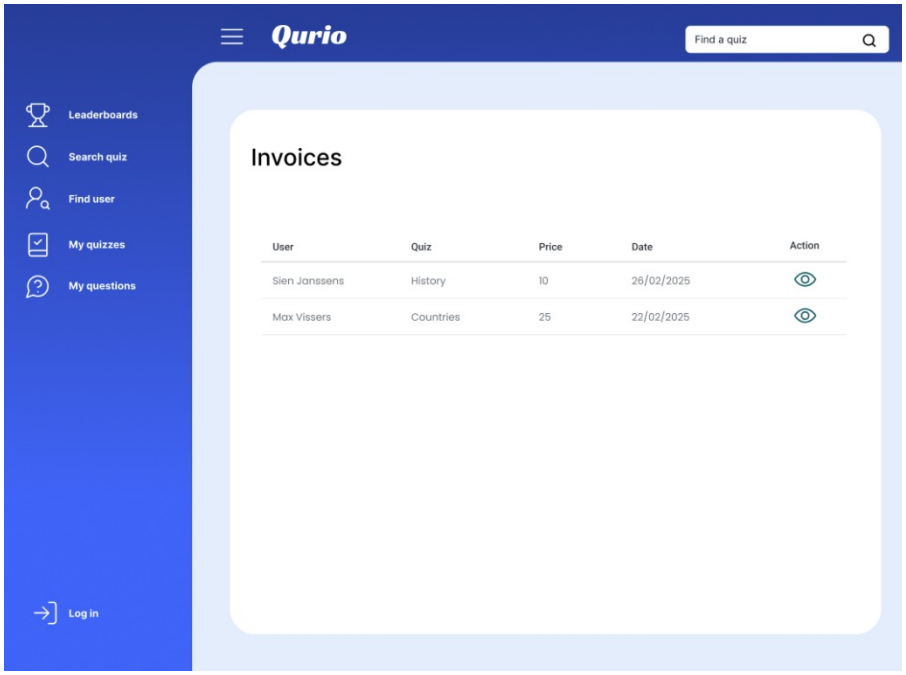
SKINNING BEHEREN

Functionaliteit: Als een bedrijfsklant, kan ik de skinning beheren.
Preconditie: De bedrijfsklant is ingelogd.
Normaal verloop: De actor kiest een achtergrondkleur en een accentkleur en geeft de url van het logo van het bedrijf in.

1.1.5.2. Bedrijfsadministrator

FACTUREN BEKIJKEN

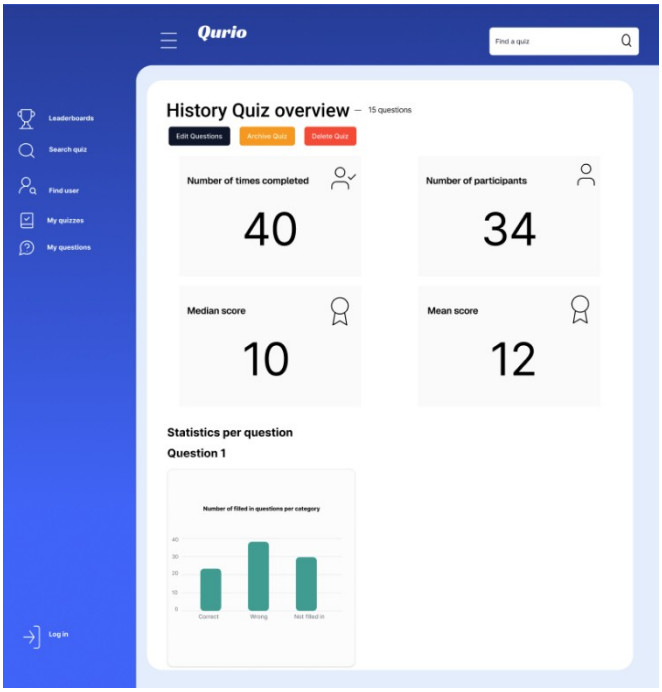
Functionaliteit: Als een bedrijfsadministrator, kan ik alle facturen voor mijn bedrijf bekijken.
Preconditie: De bedrijfsadministrator is ingelogd.
Normaal verloop: Het systeem toont alle facturen van dit bedrijf.



Figuur 26: Prototype facturen bekijken

STATISTIEKEN BEKIJKEN

Functionaliteit: Als een bedrijfsadministrator, kan ik de statistieken bekijken.
Preconditie: De bedrijfsadministrator is ingelogd en heeft een quiz geselecteerd.
Normaal verloop: Het systeem toont globale statistieken per quiz en per vraag.



Figuur 27: Prototype statistieken bekijken

RAPPORTEN BEKIJKEN

Functionaliteit: Als een bedrijfsadministrator, kan ik de rapporten bekijken.
Preconditie: De bedrijfsadministrator is ingelogd.
Normaal verloop: Het systeem toont een uitgebreid rapport van de globale statistieken over de categorieën.

1.1.5.3. Systeemadministrator

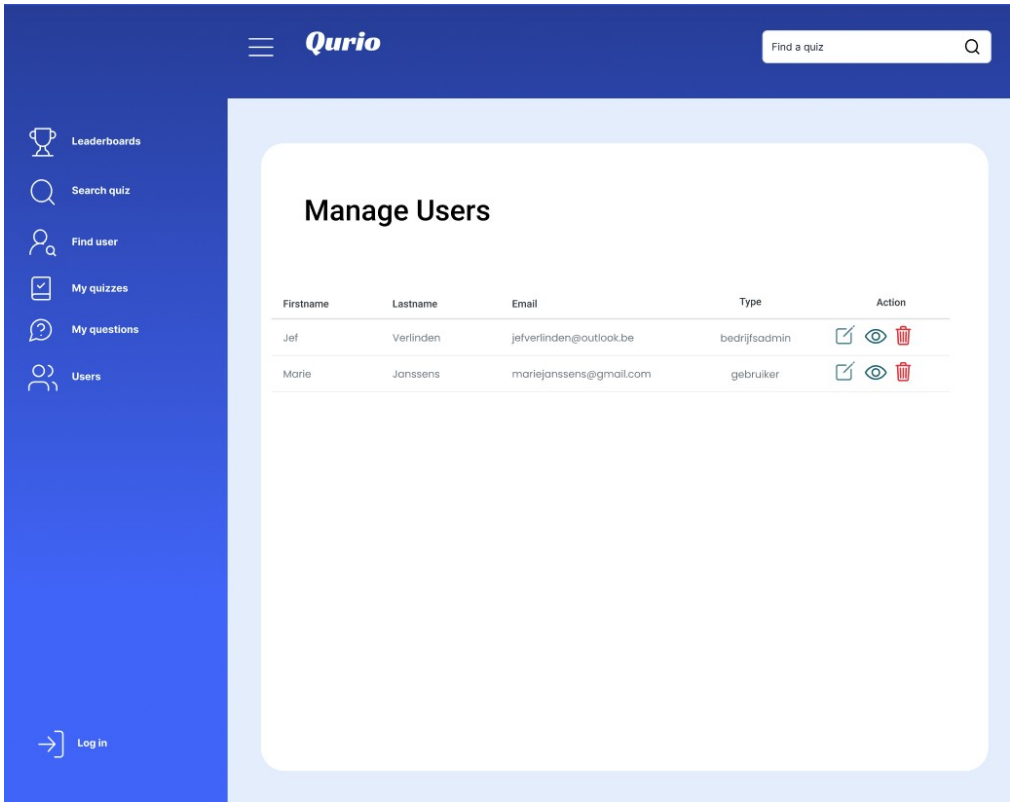
CATEGORIEËN BEHEREN

Functionaliteit: Als een systeemadministrator, kan ik de categorieën beheren.
Preconditie: De systeemadministrator is ingelogd.
Normaal verloop: Het systeem toont een lijst van alle categorieën. De actor kiest om een nieuwe categorie toe te voegen en vult de naam voor deze categorie in. Het systeem toont opnieuw een lijst met alle categorieën inclusief de nieuwe categorie.
Alternatieven:
Verwijderen: De actor kiest om een quizcategorie te verwijderen. Het systeem vraagt een bevestiging aan de actor. De actor bevestigt dit. Het systeem verwijdert de categorie.

GEBRUIKERS BEHEREN

Functionaliteit: Als een systeemadministrator, kan ik gebruikers beheren.
Preconditie: De systeemadministrator is ingelogd.

Normaal verloop: Het systeem toont een lijst van alle actieve gebruikers. De actor kiest om een gebruiker te archiveren. Het systeem toon opnieuw de lijst van alle actieve gebruikers, exclusief de gearchiveerde gebruiker.



Figuur 28: Prototype gebruikers beheren

1.2. Tools

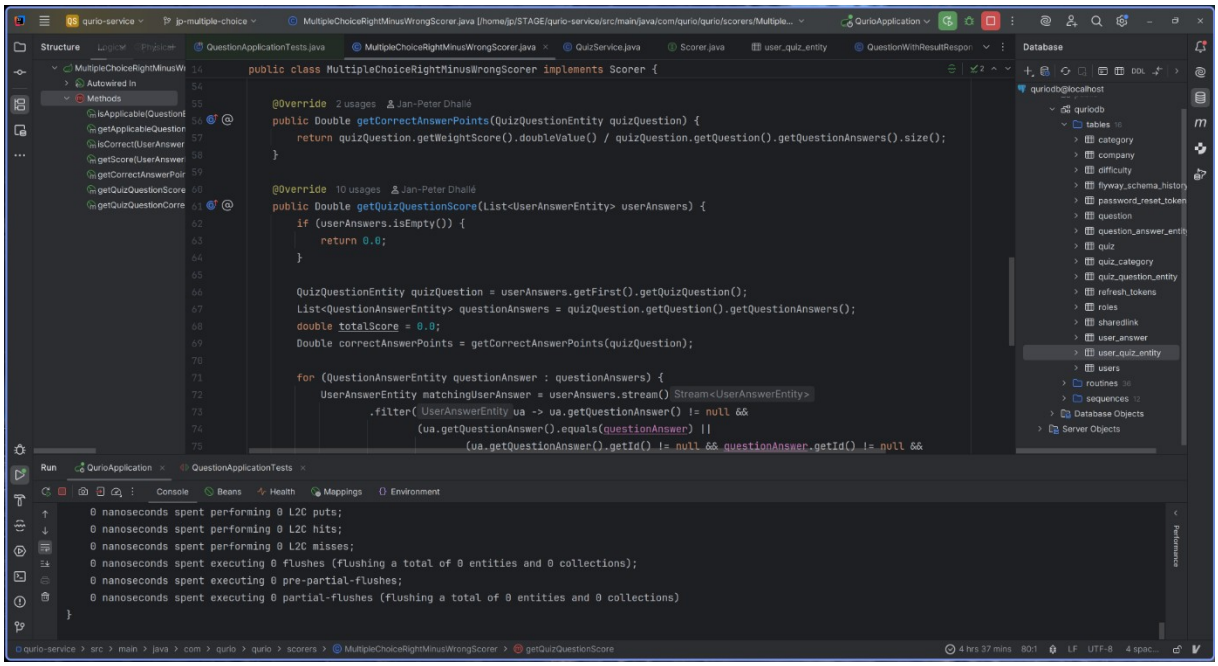
Bij het ontwikkelen van de applicatie werd er gebruik gemaakt van verschillende tools. In deze sectie bespreken we deze tools, waarom onze voorkeur uitging naar deze tools en waarvoor we ze gebruikt hebben.

1.2.1. IntelliJ IDEA Ultimate

Intellij IDEA is een *integrated development environment (IDE)* voor het ontwikkelen van Java-applicaties. Het is ontwikkeld door Jetbrains en is een van de meest populaire *IDE*s voor Java, ook voor professioneel gebruik. (IntelliJ IDEA overview | IntelliJ IDEA Documentation, 2025)

Binnen deze applicatie hebben we onze hele Java-backend ontwikkeld. Het biedt tools voor het schrijven van code, zoals een code-editor, maar ook veel andere tools voor applicatieontwikkeling. Voor dit project is er uitgebreid gebruik gemaakt van onder andere de inbegrepen testrunner, tools voor het berekenen van test *coverage*, automatische code-*formatting,* automatisch importeren van klassen, databaseviewer, functies voor het genereren van database-migratiescripts, terminal en git-functies.

Normaal gezien is een licentie vereist om toegang te krijgen tot alle functies van IDEA, maar voor dit project hebben we gebruik gemaakt van de onderwijslicentie van Jetbrains om hier toch toegang tot te krijgen.
Wij hebben voor deze *IDE* gekozen, om verschillende redenen. Om te beginnen hadden we er al ervaring mee, omdat we deze voor sommige vakken binnen de opleiding ook gebruikt hadden. Verder is deze applicatie ook de meest gebruikte editor binnen het stagebedrijf. Ten slotte is het ook een omgeving die de beste ondersteuning en de meeste tools biedt voor het ontwikkelen van een API met spring-boot.

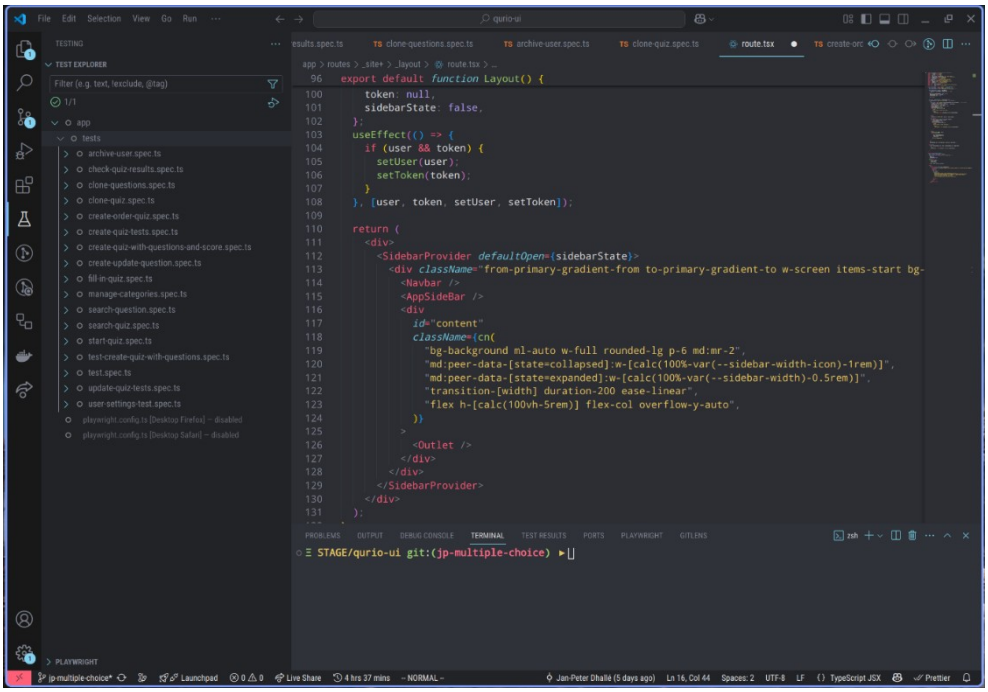


Figuur 29: IntelliJ IDEA Ultimate

1.2.2. Visual Studio Code

Visual Studio Code is een open-source text-editor ontwikkeld door Microsoft. Het frontend gedeelte van *Qurio* werd binnen deze editor ontwikkeld. Het is een van de meest gebruikte text-editor ter wereld en is erg populair onder ander omwille van het uitgebreide plugin-ecosysteem, zeker binnen frontend-ontwikkeling. (Technology | 2024 Stack Overflow Developer Survey, 2024)

Dit was ook een van de redenen waarom we voor Visual Studio Code gekozen hebben voor het ontwikkelen van onze frontend. Het heeft een extensie voor het automatisch ‘opnemen’ en uitvoeren van tests met Playwright, wat het testen van onze frontend eenvoudiger gemaakt heeft. Daarnaast was de extensie *prettier* erg nuttig voor het automatisch formatteren van onze code.

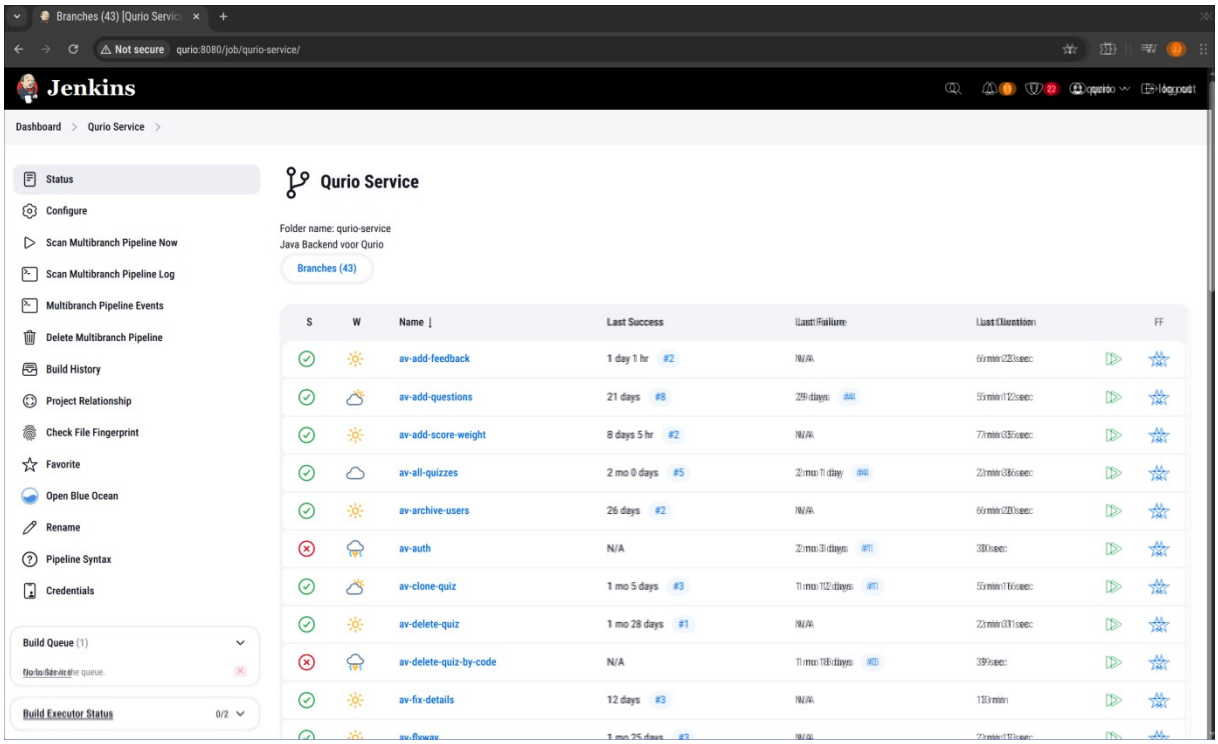


Figuur 30: Visual Studio Code

1.2.3. Jenkins

Het stagebedrijf vond het zeer belangrijk om de code die we schreven uitgebreid te testen. Op aanraden van onze stagementor, hebben we Jenkins gebruikt zodat het uitvoeren van deze tests geautomatiseerd werd.

Jenkins is een open-source CI/CD-platform waarin een gebruiker verschillende *pipelines* kan configureren. (Jenkins, sd) Het is ook mogelijk om deze te verbinden met Bitbucket, zodat de pipeline automatisch uitgevoerd wordt wanneer iemand nieuwe code toevoegt. Voor onze applicatie hebben we een pipeline geschreven in verschillende stappen. Later in het document wordt dit verder in detail besproken.

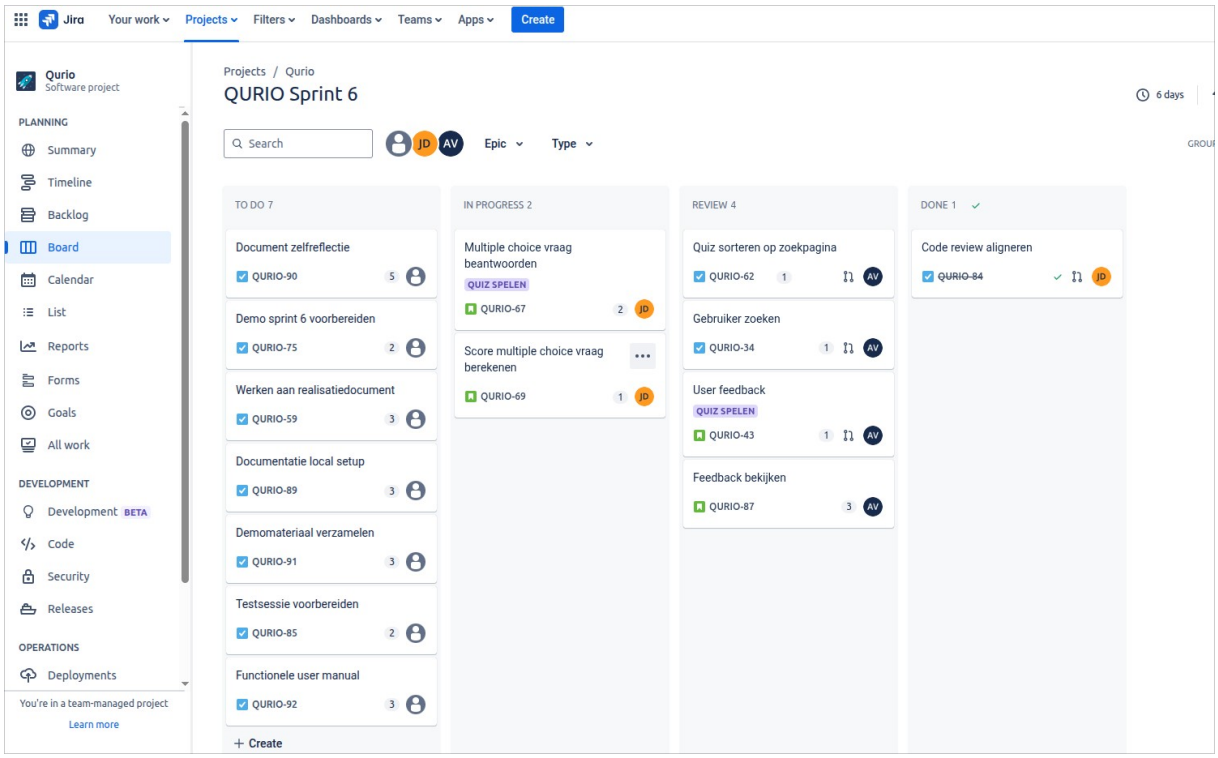


Figuur 31: Jenkins

1.2.4. Jira

Jira is een webapplicatie voor projectmanagement volgens de Agile-methodologie. Het werd ontwikkeld door Atlassian en is de de-facto standaard voor projectmanagement van softwareprojecten. (9 Best Scrum Tools To Master Project Management | Atlassian, sd)

Het stagebedrijf maakt hier tevens zelf gebruik van en zij hebben ons toegang gegeven tot ons eigen project binnen de omgeving. Jira was erg belangrijk voor de organisatie van het project. Het bood een goede structuur, aangezien alle taken hierin werden aangemaakt en verdeeld. Bovendien was de integratie met Bitbucket erg handig, omdat het een directe link maakt tussen onze code en de taken.

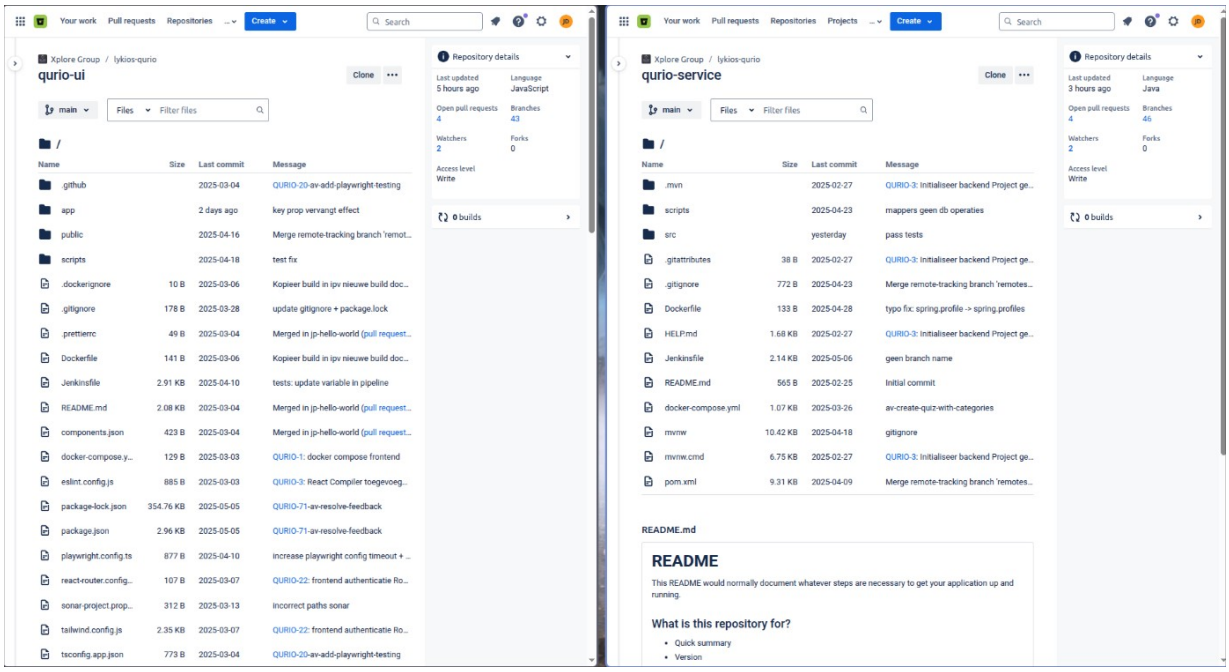


Figuur 32: Jira scrumbord

1.2.5. Bitbucket

Bitbucket is een ontwikkelingsplatform en versiebeheerplatform ontwikkeld door Atlassian. (Bitbucket Overview | Bitbucket, sd) Het biedt verschillende functies voor het samenwerken binnen een softwareproject zoals versiebeheer met git, nakijken van code en integratie met andere producten van Atlassian, waaronder Jira.

We hebben gekozen voor Bitbucket voor het hosten van onze code en versiebeheer, omdat het een goede integratie heeft met Jira en het expliciet aangeraden werd door de werknemers van ons stagebedrijf. Bovendien gebruikt ons stagebedrijf het platform zelf en heeft het een licentie voor verschillende producten van Atlassian.



Figuur 33: Bitbucket repositories

1.2.6. Confluence

Confluence is een online platform voor het onderhouden van een wiki. Dit platform is eveneens ontwikkeld door Atlassian. Het biedt tools voor het schrijven van documenten, projectmanagement en veel meer.

Binnen Confluence hebben we de bijkomstige documentatie geschreven van de applicatie. Daarnaast hebben we hierin verschillende andere documenten bijgehouden, zoals een document met gemaakte keuzes en een checklist die we afgaan voordat we een nieuwe *pull-request* gingen aanmaken. Om de twee weken, bij het afronden van een sprint, gebruikten we Confluence tevens om een sprint-review in te noteren. Ook dit platform werd sterk aangeraden door ons stagebedrijf.

1.2.7. Git

Git is een programma voor versiebeheer van code. Git kan een momentopname maken van een project. Zo bouwt het op termijn een hele historiek op van hoe de applicatie geëvolueerd is. Verder bevat het ook een aantal mechanismes en tools om in team samen te werken aan dezelfde code en de veranderingen uiteindelijk op de juiste manier samen te voegen.

Git is erg populair en wordt gebruikt voor bijna alle softwareprojecten wereldwijd. Git is eveneens open-source. Het werd oorspronkelijk ontwikkeld door de beroemde maker van het Linuxkernel-besturingssysteem, Linus Torvalds. (Git - A Short History of Git, sd)

Wij hebben git gebruikt voor dit project, omdat het de standaard is voor versiebeheer van moderne softwareprojecten. Daarnaast wordt het actief gebruikt door alle medewerkers van het stagebedrijf. In het verleden hebben we al veel gewerkt met git voor andere projecten. Toch hebben we doorheen de stage veel bijgeleerd over het samenwerken met meerdere ontwikkelaars binnen hetzelfde project met git.

1.2.8. Teams

Het houden van meetings en communicatie gebeurde doorheen de stage via Teams. Teams is een communicatieplatform voor het versturen van berichten en het houden van videoconferenties ontwikkeld door Microsoft. Het is erg populair in de bedrijfswereld en het stagebedrijf maakt bovendien intern erg veel gebruik van het platform. Wij hebben in onze opleiding eveneens gebruik gemaakt van Teams, dus dit was niets nieuws.

Binnen de applicatie had ons stagebedrijf een team aangemaakt waarin berichten naar alle bedrijfsleden en stagiairs verstuurd werden. Verder werd de planning gecommuniceerd via teams en werden alle meetings vastgezet in de kalender. Tijdens het thuiswerken, werd teams ook gebruikt voor het houden van deze meetings.

1.2.9. Postman

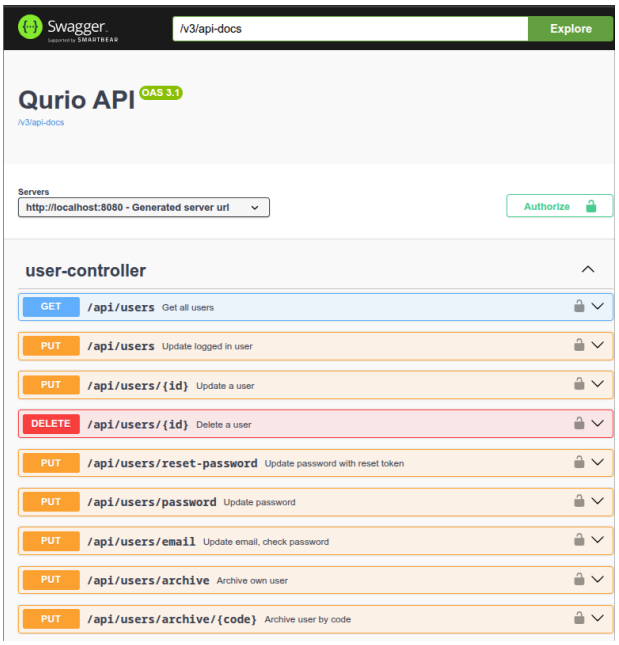
Postman is een populaire applicatie voor het vereenvoudigen van API-ontwikkeling. Het is gecreëerd door Indiase ontwikkelaars. Het wordt gebruikt voor het verzenden van verzoeken naar een API. (How We Built Postman - the Product and the Company | Postman Blog, 2021) We hebben deze applicatie gebruikt voor het ontwikkelen van onze backend.

We hebben voor Postman gekozen, omdat we er al eerder een beroep op hebben gedaan en omdat het ons ontwikkelingsproces vereenvoudigd.

1.2.10. Swagger

Swagger is een webapplicatie die via de OpenAPI-specificatie (oorspronkelijk de Swagger specificatie) een overzicht geeft van alle API-*endpoints*. (About, sd) Bovendien geeft het de datatypes weer voor elk *endpoint*, waardoor het dient als documentatie van onze API.

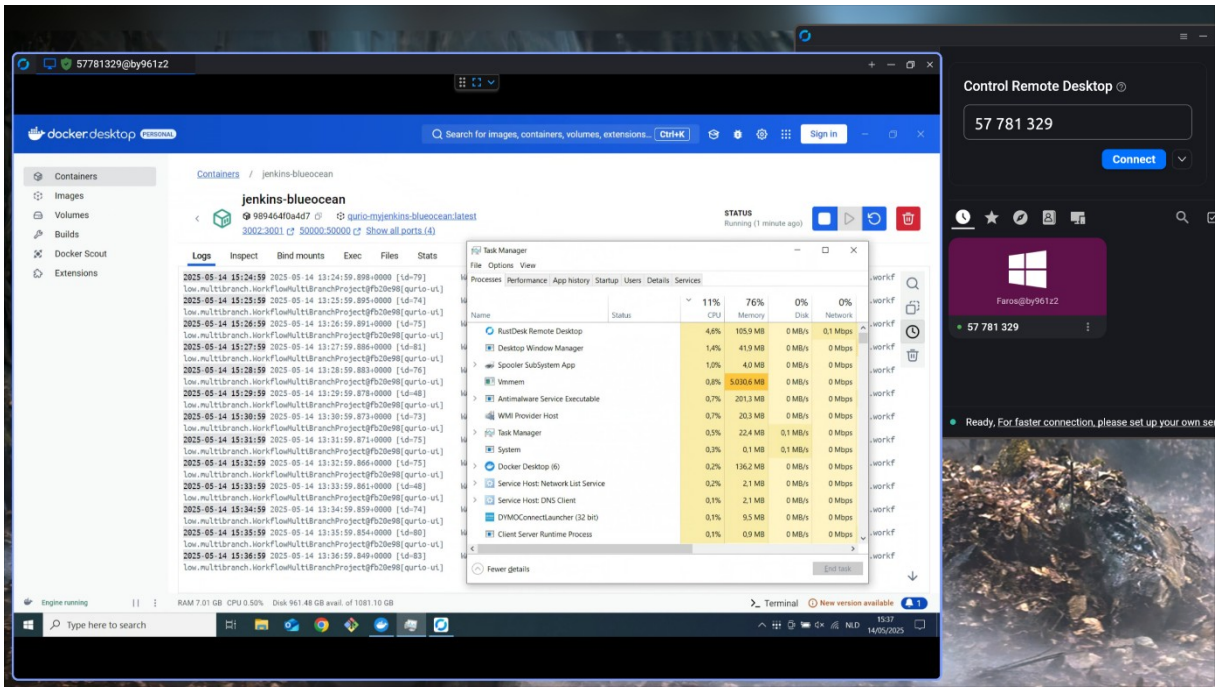
Binnen het ontwikkelproces hebben we gebruik gemaakt van Swagger om de ontwikkeling van de API te vereenvoudigen. Het is in gebruik erg gelijkaardig aan Postman, maar in Swagger is het niet mogelijk om gegevens op te slaan. Daarom hebben we vooral gebruik gemaakt van Postman voor verzoeken die we meerdere keren wilden uitvoeren.



Figuur 34: Swagger

1.2.11. RustDesk

RustDesk is een open-source applicatie voor remote-desktop. Via de applicatie kan er een verbinding gemaakt worden met een computer en kan deze vanop afstand bestuurd worden. Voor dit project was deze oplossing erg handig om onze laptop, die alle tests en de pipeline uitvoert, vanop afstand te beheren.



Figuur 35: Rustdesk

1.2.12. SonarQube

SonarQube is een open-source platform voor statische code analyse. Het controleert de veiligheid, de kwaliteit en de onderhoudbaarheid van de code. (Kashyap, 2024)

Wij hebben SonarQube gebruikt om een analyse te krijgen van onze geschreven code. Daarnaast werkten we in de backend met de SonarQube-extensie voor Intellij IDEA, die meteen aanbevelingen deed over de geschreven code.

1.2.13. Tailscale

Tailscale is een cross-platform VPN-oplossing, die gebruikmaakt van het open-source WireGuard-protocol om *peer-to-peer* verbindingen tussen apparaten tot stand te brengen. Het is gratis te gebruiken tot tien verschillende apparaten. Bovendien werkt Tailscale zonder dat er poorten in het netwerk geopend moeten worden. (What is Tailscale? Tailscale Docs, 2025)

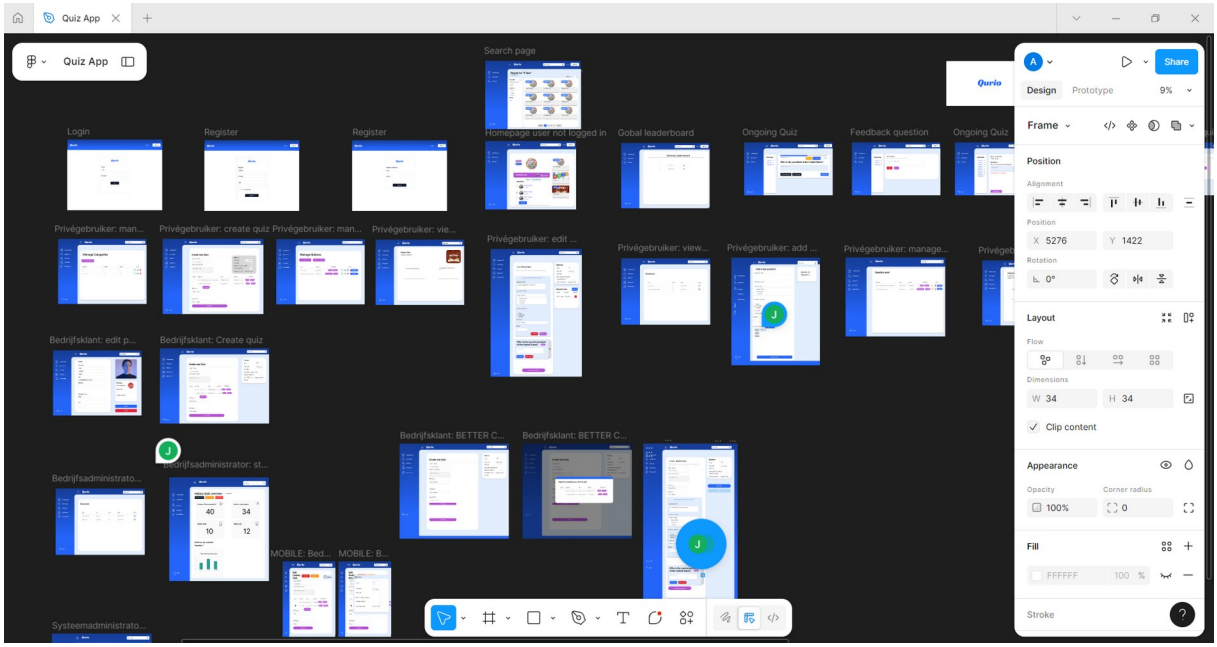
In ons project hebben we Tailscale ingezet om de laptop, die de pipeline en tests uitvoert, ook van buiten het kantoornetwerk toegankelijk te maken.

Figuur 36: Tailscale

1.2.14. Figma

Figma wordt gebruikt om te helpen bij het ontwikkelen van een user interface. Het biedt verschillende functies voor het maken van prototypes en functies die het samenwerken bevorderen. (Stevens, 2024)

We hebben Figma gebruikt voor het maken van onze prototypes. Hiervoor hebben we gebruik gemaakt van componenten van ShadCN/UI in Figma zodat we al een realistisch beeld van onze applicatie konden schetsen.



Figuur 37: Prototypes in Figma

1.2.15. Ngrok

Ngrok is een service waarmee lokale servers publiek toegankelijk gemaakt kunnen worden. Hierbij verkrijgt de gebruiker een domein van Ngrok, dat als een proxy naar de lokale service dient. (Soni, 2025)

We hebben dit gebruikt om onze applicatie toegankelijk te maken buiten het kantoornetwerk, zonder dat de medewerkers een VPN moesten installeren. Omdat er met een gratis account maar één statisch domein per gebruiker beschikbaar is, hebben we gebruik gemaakt van *nginx* om de frontend en backend te hosten.

1.2.16. nginx

Nginx is een open source HTTP-server, reverse proxy, cache, load-balancer, ... (nginx, sd) Het is wereldwijd de meest gebruikte webserver en meer dan 30% van het hele internet maakt gebruik van nginx. (Usage Statistics and Market Share of Web Servers, May 2025, 2025)

Binnen ons project hebben we het niet gebruikt als webserver, maar hebben we de reverse proxy-functionaliteit gebruikt om onze applicatie (frontend + backend) onder één domein beschikbaar te stellen.

1.3. Technologieën

Voor het ontwikkelen van de applicatie hebben we moeten beslissen welke technologieën we gingen gebruiken. Hieronder worden onze beslissingen en technologieën verder toegelicht.

1.3.1. Backend

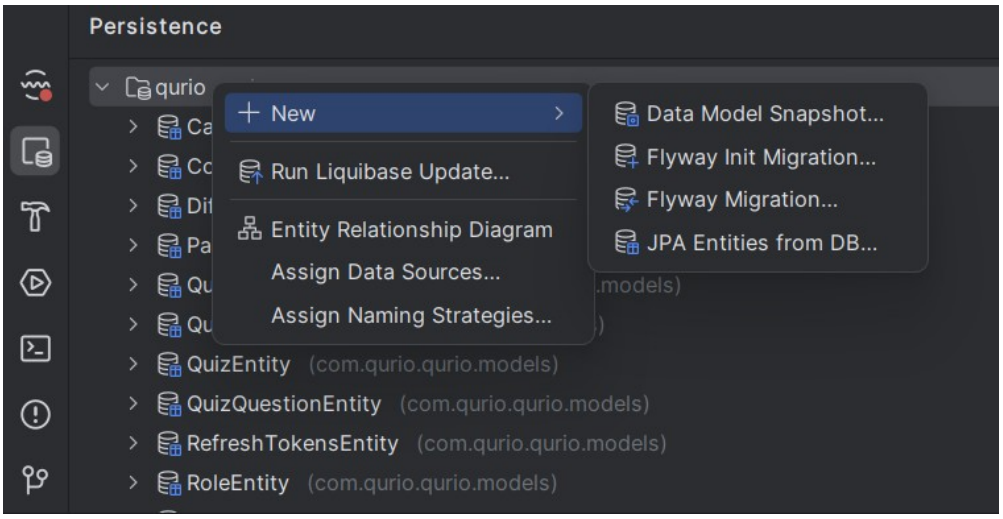
Voor onze opdracht moesten we in de backend gebruikmaken van Java en Spring Boot, aangezien het stagebedrijf hierin gespecialiseerd is. Spring Boot is een raamwerk voor het bouwen van APIs en webapplicaties. Het is gebaseerd op Spring, maar biedt al een hoop standaardconfiguratie en functionaliteiten aan, waardoor er veel minder code nodig is en het makkelijker is om te gebruiken.

Spring zelf is de kern van het framework, het is een *container* voor objecten (*modules)*. Spring gaat al deze objecten aanmaken, injecteren wanneer nodig en verwijderen. Bovendien maakt het veel gebruik van *aspecten*, waardoor er makkelijk standaardfunctionaliteiten aan een klasse toegevoegd kunnen worden door middel van annotaties.

Spring en Spring boot zijn echter niet de enige projecten binnen het ecosysteem. Het heeft verschillende bibliotheken om te werken met onder andere databases, microservices, security en cloud. Voor ons project hebben we uitgebreid gebruik gemaakt van *Spring data JPA* voor alle interactie met de database en *Spring security* voor de authenticatie en access-control. Daarnaast hebben we *Spring e-mail* gebruikt voor het verzenden van e-mails. Tot slot hebben we een beroep gedaan op *Spring-thymeleaf* voor de integratie met thymeleaf, een bibliotheek om Java-code te evalueren binnen HTML-*templates*.

Verder hebben we gebruik gemaakt van Lombok om de hoeveelheid *boilerplate*-code te verminderen. Lombok genereert automatisch code, zoals *getters*, *setters* en *constructors* op basis van annotaties.

Daarnaast hebben we een beroep gedaan op Flyway om de databasemigraties op te zetten. We hebben gebruik gemaakt van de Flyway-extensie in IntelliJ IDEA om onze initiële migratie te creëren. Verder hebben we deze plugin gebruikt om scripts te genereren van onze wijzigingen.



Figuur 38: Flyway plugin in IntelliJ

Tot slot hebben we Mapstruct gebruikt om automatisch *mappers* te genereren. *Mappers* zijn functies die objecten van een bepaald type kunnen omzetten naar een ander objecttype, meestal door gewoon de eigenschappen met dezelfde naam te kopiëren van het oude naar het nieuwe object. Binnen onze applicatie hebben we erg veel gebruik gemaakt van data transfer objecten (DTO's). Door deze objecten terug te sturen in plaats van objecten uit de database, hebben we meer controle over de data en kunnen we gevoelige en onnodige informatie weglaten. Het gebruik van *mappers* om objecten uit de database om te zetten naar deze DTO's bespaart veel herhalende code.

1.3.2. Frontend

De keuze voor de frontendtechnologie mochten we zelf maken. Hiervoor hadden we onderstaande gewogen beslissingsmatrix samengesteld. We hebben rekening gehouden met onze eigen ervaring, de ervaring van het stagebedrijf, de community support en de beschikbare bibliotheken die we kunnen gebruiken.

		Welk frontend-framework gaan we gebruiken?		
	Opties	React	Angular	Vue
Factoren	**Gewicht**			
Onze ervaring	5	3	2	0
Ervaring stagebedrijf	4	5	3	2
Community support	3	5	5	5
Beschikbare bibliotheken	2	4	4	4
		58	45	31

Tabel 1: Beslissingsmatrix frontend

Uiteindelijk hebben we besloten om React te gebruiken aangezien we hier beide de meeste ervaring mee hebben. Bovendien hebben de werknemers op de stage hier veel ervaring mee zodanig we eventuele vragen snel konden stellen. Verder heeft het voldoende community support en beschikbare bibliotheken voor ons stageproject.

Het ontwerpen van de applicatie hebben we gedaan met TailwindCSS en componenten van ShadCN/UI. Voor het verzenden van HTTP-verzoeken hebben we Axios gebruikt.

Op aanraden van een medewerker hebben we uiteindelijk gekozen om React Router in *framework-mode* te gebruiken. React Router was oorspronkelijk een bibliotheek voor *single-page applications (SPA)* om ook routing-functionaliteit via URL te voorzien. Dit is iets dat normaal gezien niet mogelijk is bij *SPAs*, aangezien de hele applicatie binnen een enkele pagina zit (een URL). De laatste jaren is het echter meegegroeid met de rest van het React-ecosysteem. In de versie die we gebruikt hebben, namelijk V7, heeft het tevens functionaliteiten die enkel op de server uitgevoerd worden. De voornaamste zijn *loaders,* om data op te halen op de server en later te injecteren in de componenten en *actions*, om formulieren te valideren en in te dienen. Daarnaast heeft het verschillende manieren om pagina's te *renderen*, namelijk *client-side rendering, Server Side Rendering en Static Pre-rendering.* Al deze functionaliteiten hebben een grote impact op de manier waarop de applicatie geschreven wordt, maar ook op de snelheid, prestaties, caching-strategieën en zoekmachineoptimalisatie *(SEO)*.

1.3.3. Database

Voor de keuze van de database hadden we eveneens een beslissingsmatrix opgesteld. We hebben hier Postgres, MySQL en MongoDB vergeleken. Hierbij hebben we rekening gehouden met onze eigen ervaring, de ervaring binnen ons stagebedrijf en de community support. We hebben uiteindelijk gekozen voor Postgres omdat we hier graag nog meer ervaring over wilden opdoen.

		Welke database gaan we gebruiken?		
	Opties	Postgres	MySQL	Mongodb
Factoren	**Gewicht**			
Onze ervaring	5	3	4	2
Ervaring stagebedrijf	4	4	4	3
Community support	3	4	5	3
JSON-opslag	2	4	1	5
Structuur (relationeel of niet)	3	4	4	2
Integratie met Java en Spring	3	4	4	3
		75	77	59

Tabel 2: Beslissingsmatrix database

2. Resultaat

In dit gedeelte wordt het eindresultaat van onze stageopdracht besproken. Eerst worden alle uitgewerkte functionaliteiten toegelicht. Nadien wordt er uitgelegd hoe wij onze applicatie hebben getest.

2.1. Uitgewerkte functionaliteiten

Hieronder wordt voor elke functionaliteit de werking kort toegelicht. De toegevoegde schermafbeeldingen zijn afbeeldingen van de applicatie zelf en niet van de prototypes zoals bij het analysegedeelte.

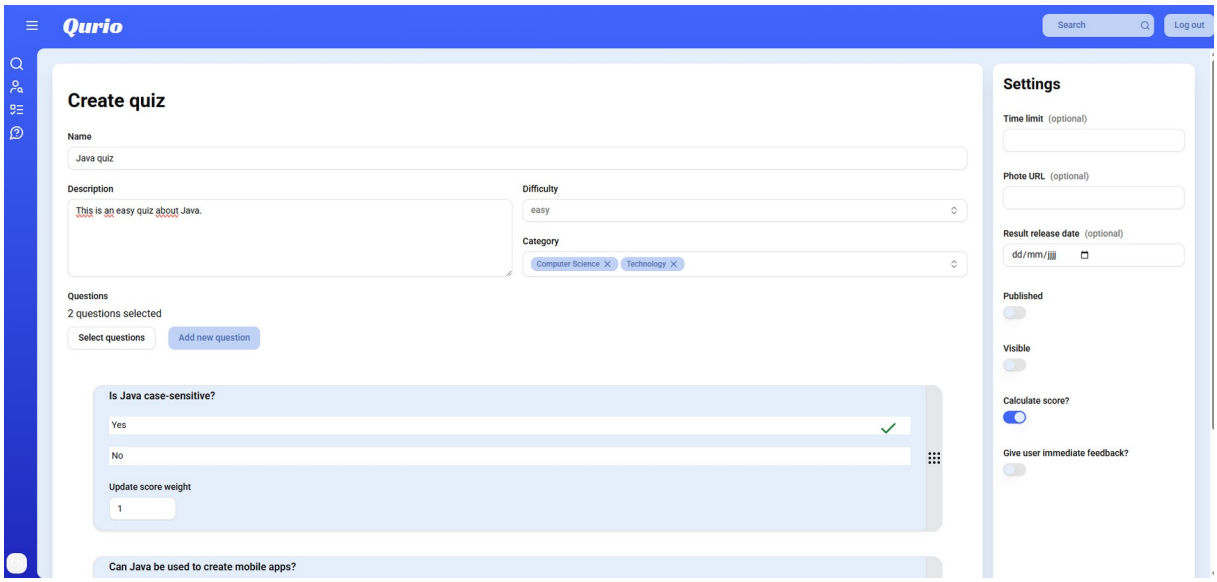
2.1.1. Authenticatie

De eerste functionaliteiten die we hebben uitgewerkt zijn het registreren en het inloggen. Dit is de basis van de applicatie en dit moet op een veilige manier kunnen gebeuren. In de backend hebben we hiervoor gebruik gemaakt van Spring Security en JWT-tokens. Bovendien maken we gebruik van *refreshtokens* zodat de gebruiker voor een langere periode ingelogd kan blijven.

2.1.2. Quiz beheren

De volgende functionaliteit die noodzakelijk was, was het beheren van de quizzen. Elke gebruiker kan zijn eigen quizzen bekijken en beheren. De gebruiker kan een nieuwe quiz aanmaken en kan eveneens een quiz wijzigen. Wanneer een quiz gepubliceerd is, kan deze niet meer gewijzigd worden omdat de andere gebruikers dan de mogelijkheid hebben om de quiz te spelen. De gebruiker heeft dan wel de optie om de quiz te klonen. In dat geval wordt er een nieuwe quiz aangemaakt met dezelfde vragen en eigenschappen als de vorige quiz waarbij de gebruiker wel de toelating heeft om deze quiz te wijzigen. Tot slot kan een gebruiker een quiz archiveren.

Alle data worden zowel in de frontend als op API-niveau gevalideerd. Op deze manier kunnen we garanderen dat er geen ongeldige of onverwachte data in onze database opgeslagen wordt.



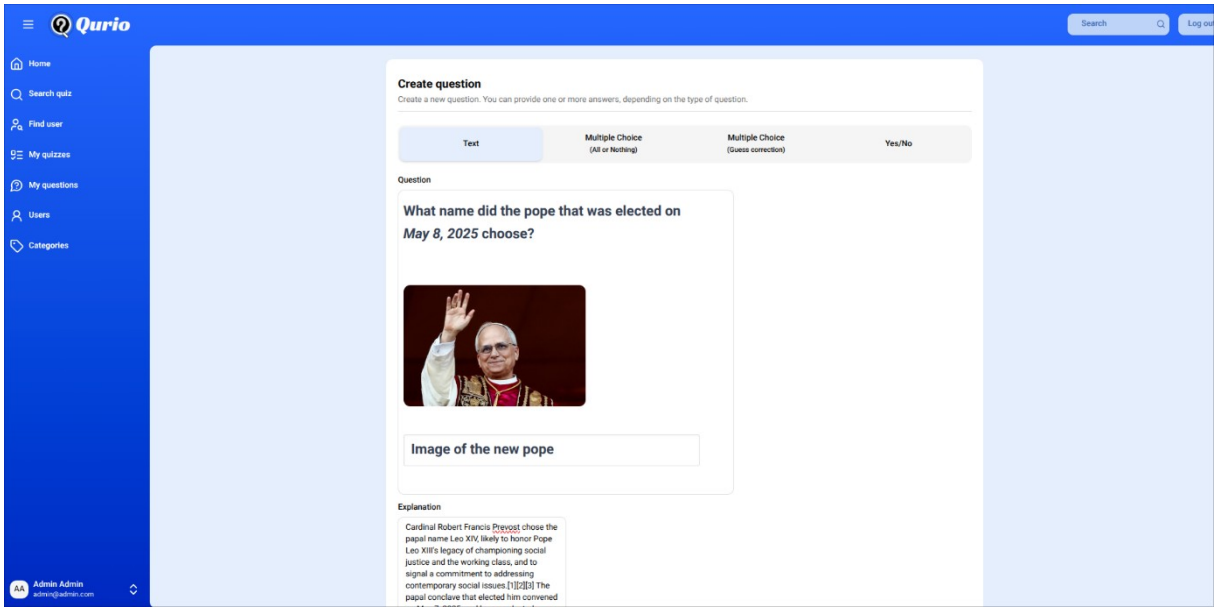
Figuur 39: Aanmaken van een quiz

2.1.3. Vragen beheren

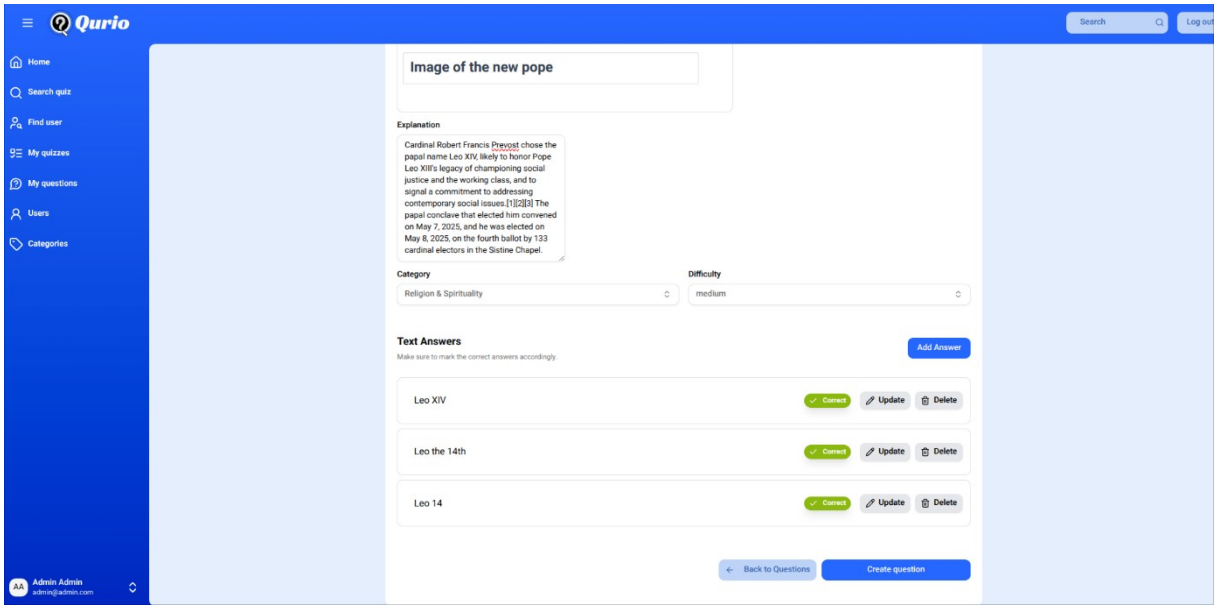
Tijdens onze analyse van ons project hadden we besloten dat we de mogelijkheid wilden om vragen in verschillende quizzen te kunnen hergebruiken. Dit betekent dat vragen en quizzen volledig los van elkaar bestaan. Om dit mogelijk te maken moet een gebruiker zijn vragen kunnen beheren.

Een gebruiker kan een vraag aanmaken, archiveren en klonen. Dit laatste is nuttig, want een vraag mag niet meer verwijderd of aangepast worden wanneer deze gelinkt is aan een gepubliceerde quiz. Dat betekent namelijk dat de quiz al gespeeld is. Deze beslissing hebben we gemaakt, omdat het wijzigen van vragen waar scores en antwoorden aan gelinkt zijn veel problemen zouden veroorzaken. Bovendien zou dit alle voordien gespeelde quizzen met deze vraag ongeldig maken.

Bij het maken van een vraag kan een gebruiker kiezen uit de verschillende vraagtypes. Hierbij moeten alle velden ingevuld worden en moet de gebruiker een of meerdere antwoorden op de vraag ingeven.



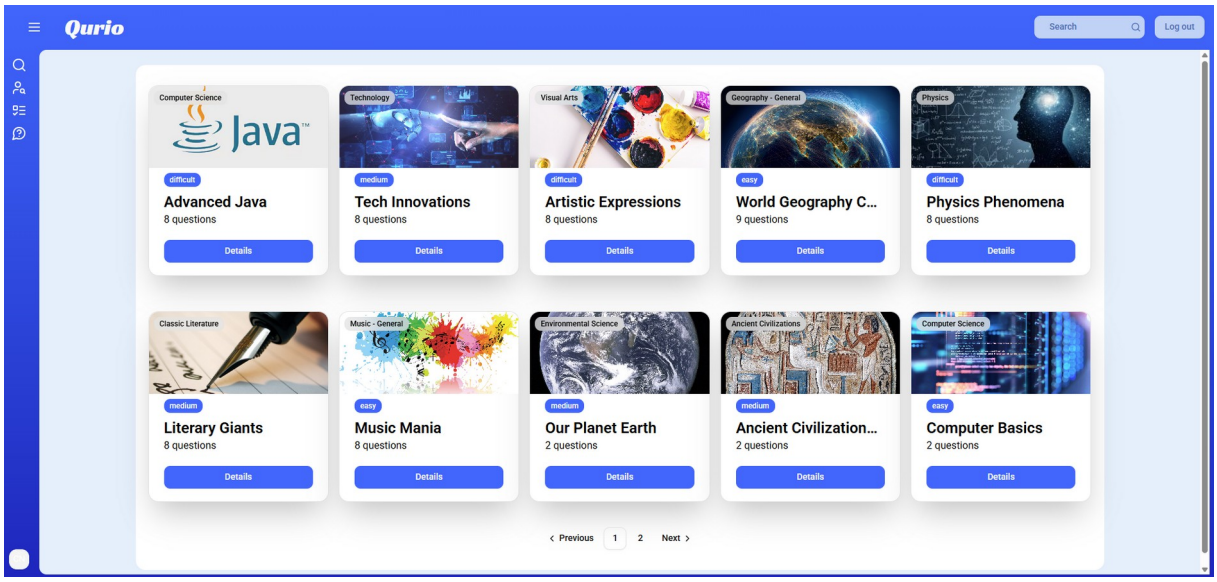
Figuur 40: Vraag aanmaken



Figuur 41: Antwoorden voor vraag aanmaken

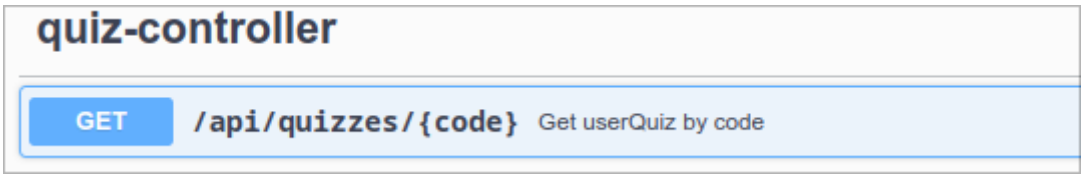
2.1.4. Quiz spelen

Op de homepagina staan alle quizzen. Elke quiz heeft een startpagina waarop de gebruiker meer info over de quiz kan vinden en hij de quiz kan beginnen.

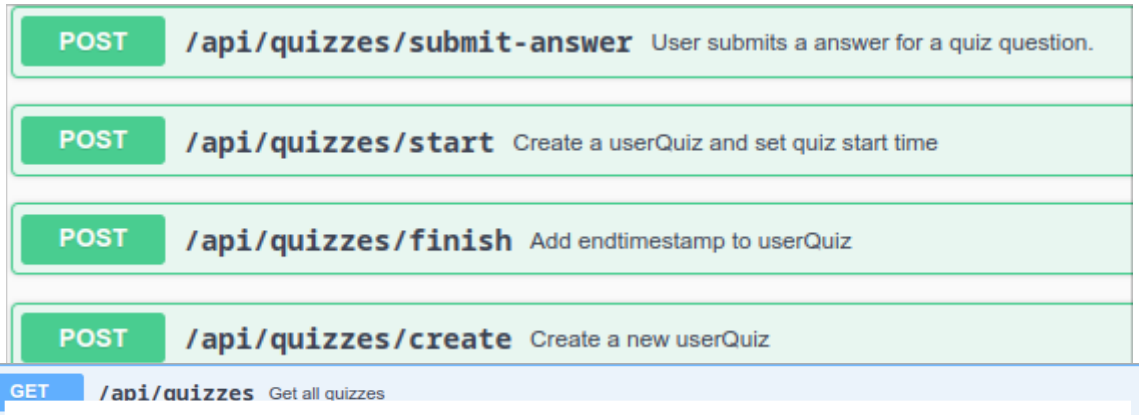


Figuur 42: Homepagina

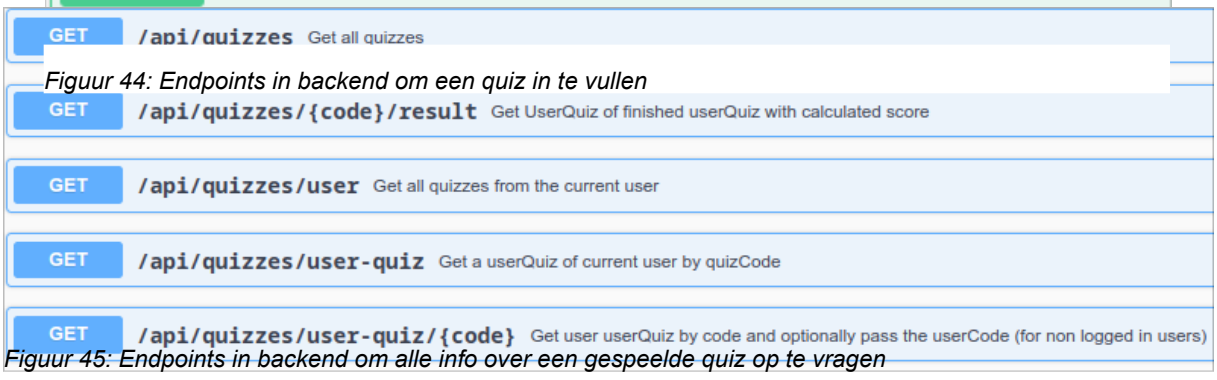
Om al deze acties mogelijk te maken hebben we eerst in onze API de volgende *endpoints* gedefinieerd.



Figuur 43: Endpoint in backend om een ingevulde quiz via de code op te vragen

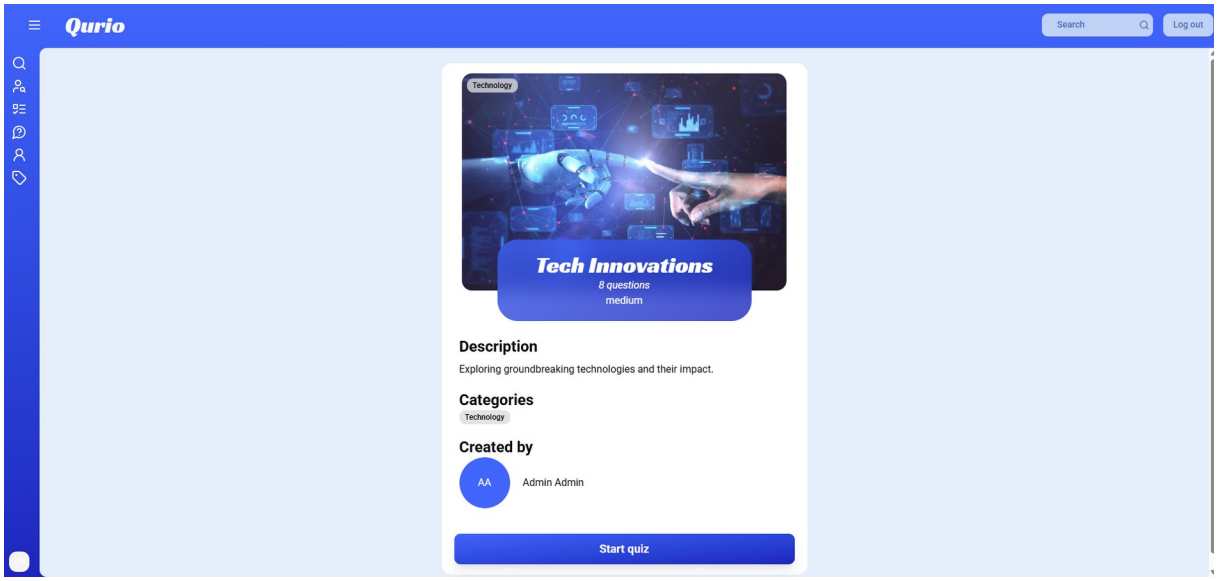


Figuur 44: Endpoints in backend om een quiz in te vullen



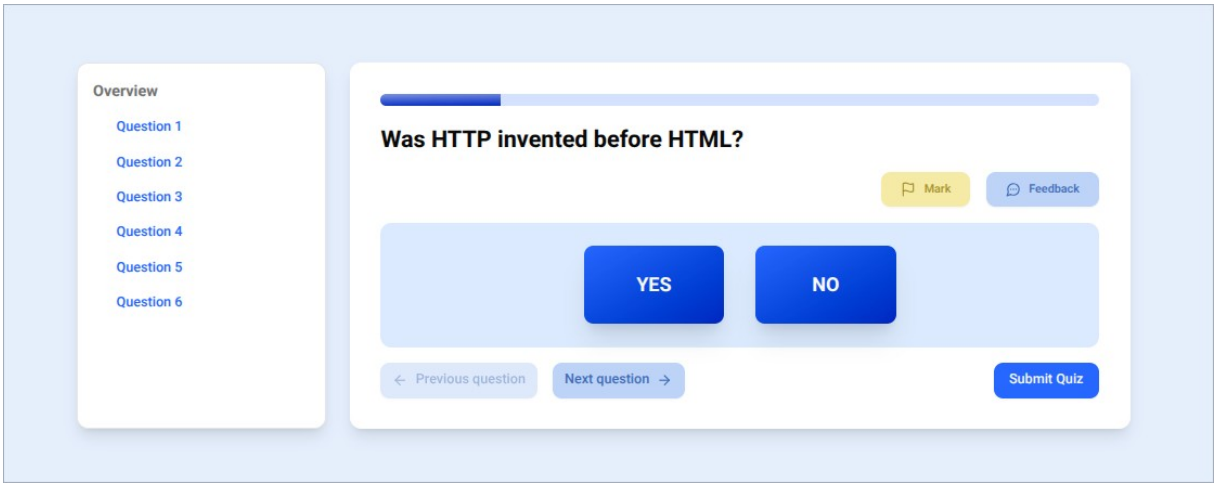
Figuur 45: Endpoints in backend om alle info over een gespeelde quiz op te vragen

De frontend gebruikt deze *endpoints* om data op te halen voor de verschillende pagina's. Wanneer een gebruiker na het bekijken van de detailpagina van een quiz besluit om deze te spelen, kan hij op de knop 'start quiz' klikken om de quiz te starten.



Figuur 46: Startpagina quiz

Hierna krijgt de gebruiker onmiddellijk de eerste vraag en een overzicht van alle vragen te zien. Het speelgedeelte van de applicatie is zo gemaakt, dat tijdens het invullen van de quiz alle antwoorden en de gespendeerde tijd per vraag gesynchroniseerd worden naar de server. Hierdoor kunnen we detecteren wanneer iemand een quiz afsluit voordat deze is afgerond. Verder kunnen we de validiteit van de data en de gespendeerde tijd controleren in de backend. De gebruiker kan er ook voor kiezen om vragen blanco te houden, of te markeren om later op terug te komen.



Figuur 47: Spelen van een quiz

De frontend maakt, zoals de rest van de applicatie, handig gebruik van de functionaliteiten van React Router. Bij elke vraag navigeren we naar een andere URL, overeenkomstig met de vraag. Hierdoor zal de

overeenkomstige *loader* uitgevoerd worden die de nodige data ophaalt. Om het efficiënt te houden cache we de quiz-data op server-niveau, zodat verschillende gebruikers deze kunnen gebruiken.

De staat van de quiz wordt bijgehouden via een *context* van React. Dit is een manier om de staat van een pagina te gebruiken binnen verschillende componenten. Dit is handig aangezien het invullen van een quiz bestaat uit verschillende componenten, die elk toegang nodig hebben tot dezelfde data (quizgegevens, vraaggegevens, etc.). Het timen van de gespendeerde tijd per vraag gebeurt binnen deze context, door middel van een *useEffect*. Deze wordt gestart het renderen van een component.

```
useEffect(() => {
  startTimerRef.current();
  const localStopTimer = stopTimerRef.current;
  return () => {
    localStopTimer();
  };
}, []);
```

Figuur 48: Timer

Navigatie in React Router is echter speciaal en gaat de component niet opnieuw aanmaken. Hierdoor blijft de staat wel behouden, zelfs wanneer we naar een andere URL navigeren. Wanneer we navigeren naar een andere pagina, wordt de vraagindex in de state ook geüpdatet. Wanneer dit gebeurt, wordt de timer voor de vorige vraag gestopt.

```
const setActiveQuestionIndex = (newIndex: number) => {
  stopTimerRef.current();
  dispatch({ type: "SET_ACTIVE_QUESTION_INDEX", payload: newIndex });
  startTimerRef.current();
};
```

Figuur 49: Timer stoppen

Zoals hierboven al vermeld, willen we ook de geselecteerde antwoorden al onmiddellijk synchroniseren met de backend. Dit is geïmplementeerd door middel van een *useEffect* die uitgevoerd wordt wanneer de vraagindex verandert.

```
useEffect(() => {
  if (state.activeQuestionIndex !== state.previousQuestionIndex)
    syncQuestionToServerRef.current(state.previousQuestionIndex);
}, [state.activeQuestionIndex, state.previousQuestionIndex]);
```

Figuur 50: Synchronisatie

Al de state wordt bijgehouden in een groot object. Hiervoor wordt er gebruikgemaakt van *useReducer*-hook van React. De state wordt dan aangemaakt in de *context*:

```
const [state, dispatch] = useReducer(quizReducer, initialState);
```

Figuur 51: useReducer-hook

Hierbij is *quizReducer* een functie die de huidige state accepteert. Er moet eveneens een actie meegegeven worden. De functie gaat dan over het type actie en voert code uit afhankelijk van deze actie.

```
const quizReducer = (state: QuizState, action: QuizAction): QuizState => {
  switch (action.type) {
```

```
export type QuizAction =
  | { type: "SET_QUIZ"; payload: QuizResponseDto | undefined }
  | { type: "SET_USER_CODE"; payload: string | null }
  | { type: "SET_USER_QUIZ_CODE"; payload: string }
  | { type: "TOGGLE_MARKED_QUESTION"; payload: string }
  | { type: "SET_ACTIVE_QUESTION_INDEX"; payload: number }
  | {
    type: "UPDATE_QUESTION_DURATION";
    payload: { questionCode: string; duration: number };
  }
  | { type: "SET_START_TIME"; payload: Date | null }
  | {
    type: "SET_QUESTION_ANSWER";
    payload: { questionCode: string; answer: string; code: string | null };
  }
  | { type: "MARK_QUESTION_SUBMITTED"; payload: string };
```

Figuur 53: Definiëren van acties

Figuur 52: acties

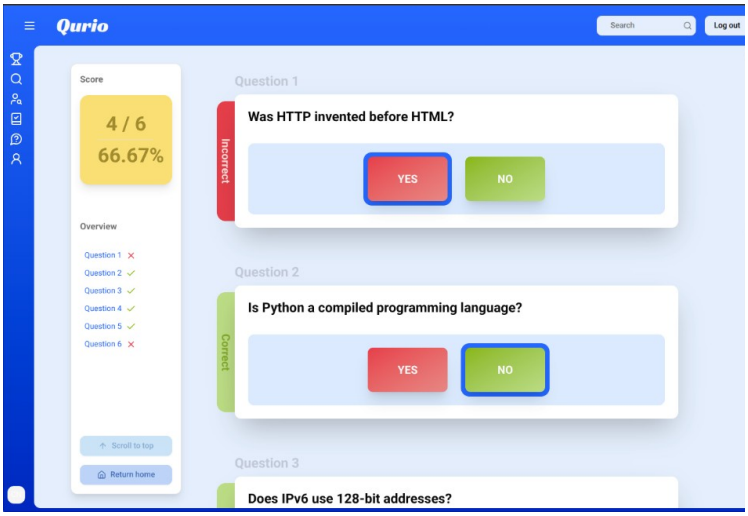
Alle actietypes zijn ook gedefinieerd, samen met de data die eraan meegegeven wordt (de *payload*).

Dit is heel het staatmechanisme voor het spelen van een quiz in onze applicatie. Op deze manier kunnen alle componenten die toegang nodig hebben tot de staat van het quizspel, de *useQuiz*-hook uit de *context* gebruiken om deze state te lezen of aan te passen.

```
const {
  quiz: contextQuiz,
  setQuiz,
  userQuizCode: contextUserQuizCode,
  setUserQuizCode,
  userCode: contextUserCode,
  setUserCode,
  activeQuestionIndex,
  setActiveQuestionIndex,
  submittedQuestions,
  questionAnswers,
  stopTimer,
} = useQuiz();
```

Figuur 54: useQuiz-hook

Bij het indienen van de quiz wordt de eindtijd opgeslagen en indien mogelijk het resultaat berekend. Dit resultaat wordt opgeslagen in de database, zodat we de berekening niet elke keer moeten herhalen. Als de maker van de quiz heeft ingesteld dat het resultaat meteen berekend wordt, wordt dit getoond aan de gebruiker.



Figuur 55: Resultatenpagina

Op deze resultatenpagina krijgt de gebruiker vervolgens een overzicht van alle vragen en de resultaten. De score wordt ook getoond.

De implementatie voor het berekenen van de score in de backend is ook interessant. Hiervoor hebben we gebruik gemaakt van het *strategy pattern* ontwerppatroon. Dit is een manier om functionaliteit te organiseren in objectgeoriënteerde programmeertalen, zodat de implementatie van een algoritme tijdens de uitvoering kan worden gekozen.

Het algoritme dat we veralgemeend hebben volgens dit ontwerppatroon, is het algoritme voor het controleren of een antwoord correct is of niet. Dit algoritme verschilt namelijk voor verschillende types van vragen. Voor elk type vraag hebben we een *Scorer*-klasse aangemaakt die een gemeenschappelijke interface implementeert.

```
public interface Scorer {  12 usages  1 implementation  Jan-Peter Dhallé *
    Boolean isApplicable(QuestionEntity question);  1 usage  1 implementation  Jan-Peter Dhallé

    Class<? extends QuestionEntity> getQuestionType();  2 usages  1 implementation  Jan-Peter Dhallé

    default Boolean isCorrect(QuestionEntity question, UserAnswerEntity userAnswer) {  2 usages  1 override
        throw new InvalidQuestionTypeException("Can not check if a question without type is correct.");
    }
}
```

Figuur 56: Interface Scorer

Het eenvoudigste type vraag is een ja/nee-vraag. Op onderstaande afbeelding wordt de implementatie van dit algoritme voor deze vraagsoort getoond.

```java
@Component  Jan-Peter Dhallé
public class YesNoScorer implements Scorer {
    @Override  1 usage  Jan-Peter Dhallé
    public Boolean isApplicable(QuestionEntity question) { return question.getQuestionType().equals("yesno"); }

    @Override  2 usages  Jan-Peter Dhallé
    public Class<? extends QuestionEntity> getQuestionType() { return YesNoQuestionEntity.class; }

    @Override  2 usages  Jan-Peter Dhallé
    public Boolean isCorrect(QuestionEntity question, UserAnswerEntity userAnswer) {
        if (Boolean.TRUE.equals(isApplicable(question))) {
            return userAnswer.getQuestionAnswer().isCorrect();
        }
        return false;
    }
}
```

Figuur 57: Implementatie interface

In de service-klasse wordt nadien een *scorer* gebruikt om de score van een quiz te berekenen. Om echter de juiste *scorer*-klasse te vinden voor een vraagtype, wordt er na het aanmaken van de service een map gebouwd met als sleutel de klasse van een vraagobject en als waarde de overeenkomstige *scorer*.

```java
private final Map<Class<? extends QuestionEntity>, Scorer> scorerMap = new HashMap<>();

@PostConstruct  Jan-Peter Dhallé
private void postConstruct() {
    // Register all scorers
    for (Scorer scorer : scorers) {
        scorerMap.put(scorer.getQuestionType(), scorer);
    }
}
```

Figuur 58: ScoreMap

Vervolgens is het eenvoudig om een algemene methode te schrijven voor het berekenen van het resultaat van een quiz, zonder rekening te moeten houden met de types van vragen. Dit betekent ook dat deze methode zal blijven werken, wanneer vraagtypes toegevoegd of verwijderd worden.

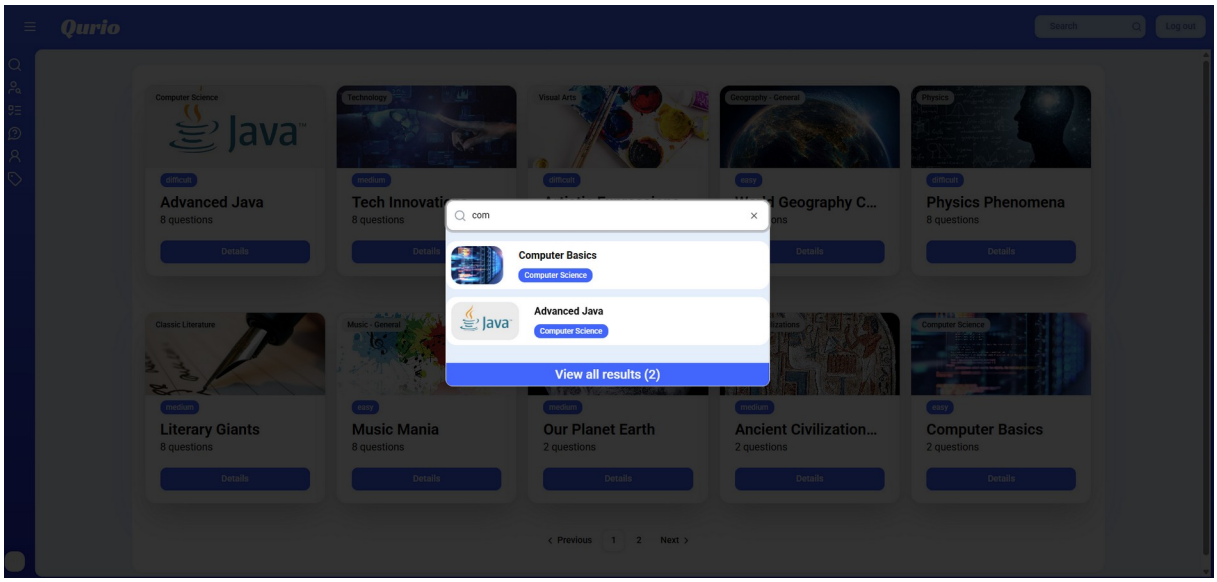
```java
public Integer calculateUserQuizScore(UserQuizEntity userQuiz) {  3 usages
    if (userQuiz.getEndTime() == null)
        throw new NoResultsBeforeSubmit("Should calculate user userQuiz score before end time is set.");
    return userQuiz.getUserAnswers().stream()  Stream<UserAnswerEntity>
            .filter( UserAnswerEntity a -> Optional.ofNullable(scorerMap.get(a.getQuizQuestion().getQuestion().getClass()))  Optional<Scorer>
                    .orElseThrow(() -> new InvalidQuestionTypeException("Invalid question type"))  Scorer
                    .isCorrect(a.getQuizQuestion().getQuestion(), a))
            .mapToInt( UserAnswerEntity userAnswer -> userAnswer.getQuizQuestion().getWeightScore())  IntStream
            .sum();
}
```

Figuur 59: Methode om score te berekenen

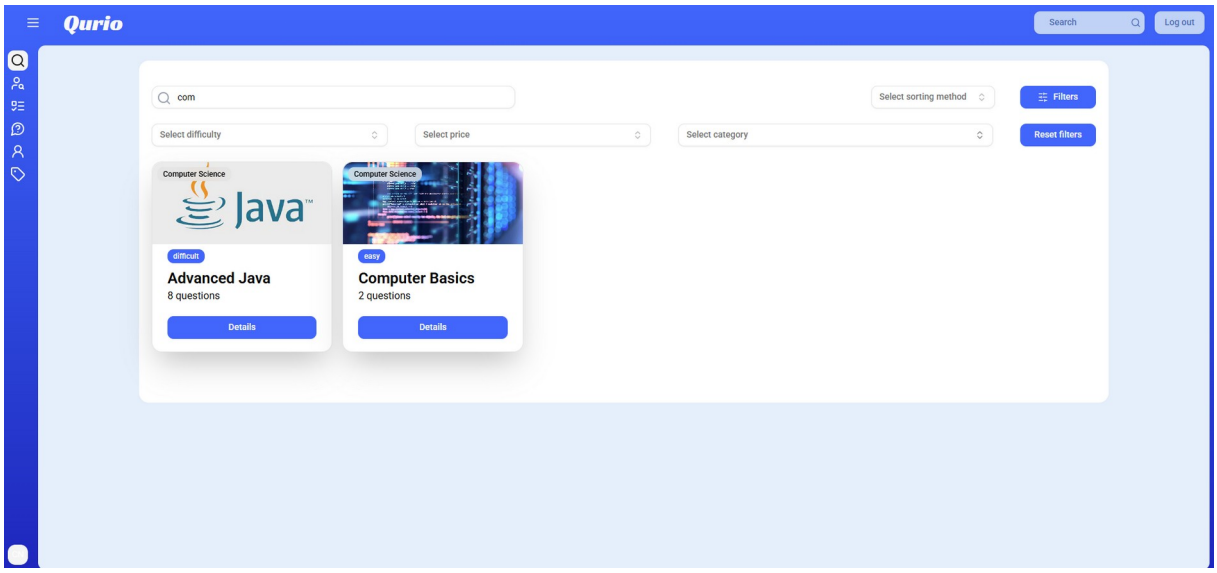
De methode controleert eerst of een quiz wel een eindtijd heeft, anders is die nog bezig en mag de score nog niet berekend worden. Vervolgens gaat het voor elk antwoord nakijken of deze correct is, door de overeenkomstige *scorer* uit de *map* te halen en hier vervolgens de *isCorrect* methode van op te roepen. Als het type van de vraag ongeldig is wordt er een *exception* gegooid. Hierna wordt er nog rekening gehouden met het gewicht van de vraag. Ten slotte wordt de som berekend. Het resultaat is de totale score van de gebruiker voor een bepaalde quiz.

2.1.5. Quiz zoeken

Aangezien het praktisch zou zijn om tussen alle quizzen een specifieke quiz te zoeken, hebben we zowel een zoekbalk, als een zoekpagina ontworpen. Via de zoekbalk kan een gebruiker een quiz zoeken via een naam, een beschrijving of een naam van een categorie. De zoekbalk toont standaard maximum vijf resultaten, maar toont onderaan wel een knop met het aantal effectieve resultaten. Wanneer de gebruiker op een resultaat klikt, wordt hij doorverwezen naar de startpagina van de gekozen quiz. Wanneer de gebruiker op de knop onderaan klikt, wordt hij doorverwezen naar doe zoekpagina. De filter blijft dan automatisch ingevuld wanneer er naar de zoekpagina genavigeerd wordt.



Figuur 60: Zoekbalk



Figuur 61: Zoekpagina

Op de zoekpagina zijn meerdere filters aanwezig. De gebruiker kan, zoals bij de zoekbalk, een quiz zoeken via de naam, de beschrijving of de naam van een categorie. Daarnaast is er de mogelijkheid om de moeilijkheidsgraad en/of een of meerdere categorieën aan te duiden. Tot slot is er nog een filter waarbij alle betalende of alle niet-betalende quizzen worden getoond.

Bovendien kan er op de zoekpagina gesorteerd worden. Dit kan zowel oplopend, als aflopend op datum en op populariteit.

Bij de zoekpagina wordt er gebruikgemaakt van URL-parameters. De *loader* gaat eerst controleren of er URL-parameters beschikbaar zijn. Als er parameters beschikbaar zijn gaat de *loader* deze gebruiken. Dit zorgt ervoor dat de URL gekopieerd en opnieuw geplakt kan worden in een ander tabblad, zonder dat de pagina zijn filters verliest. Het is dus eenvoudig om de URL te delen.

```
24  const url = new URL(request.url);
25  const text = url.searchParams.get("text") || "";
26  const difficulty = url.searchParams.get("difficulty") || "";
27  const category = url.searchParams.getAll("category") || [];
28  const paidQuiz = url.searchParams.get("paidQuiz") || "";
```

Figuur 62: URL-parameters

In de backend wordt er hierbij gebruik gemaakt van Criteria API. We hebben hiervoor gekozen op aanbevelen van onze stagementor, omdat we een efficiënte manier zochten om de optionele parameters allemaal samen te doen functioneren. Via Criteria API wordt er een grote query aangemaakt.

Hiervoor wordt er eerst een *CriteriaBuilder* gedefinieerd. Vervolgens wordt er een query aangemaakt en wordt de *root* en de *joins* aangemaakt. Daarna worden enkele condities toegevoegd voor de quiz. Zo moet de quiz zichtbaar zijn en mag het niet gearchiveerd of privé zijn. Nadien volgen er enkele methodes waarin de filters worden overlopen en eventueel nieuwe condities worden toegevoegd.

```java
@RequiredArgsConstructor  Amber Vranckx +1
@Slf4j
public class QuizSearchRepositoryImpl implements QuizSearchRepository {
    private final EntityManager entityManager;

    @Override  8 usages  Amber Vranckx
    public List<QuizEntity> searchQuiz(String sort, String direction, String text, List<String> categories, String difficulty, String paidQuiz) {
        CriteriaBuilder cb = entityManager.getCriteriaBuilder();
        CriteriaQuery<QuizEntity> query = cb.createQuery(QuizEntity.class);
        Root<QuizEntity> quiz = query.from(QuizEntity.class);
        Join<QuizEntity, CategoryEntity> categoryJoin = quiz.join( s: "categories", JoinType.LEFT);
        Join<QuizEntity, DifficultyEntity> difficultyJoin = quiz.join( s: "difficulty", JoinType.LEFT);
        Join<QuizEntity, UserQuizEntity> userQuizJoin = quiz.join( s: "playedQuizzes", JoinType.LEFT);
        query.select(quiz);
        List<Predicate> conditions = new ArrayList<>();
        conditions.add(cb.isTrue(quiz.get("visible")));
        conditions.add(cb.isFalse(quiz.get("archived")));
        conditions.add(cb.isFalse(quiz.get("privateQuiz")));

        searchByText(text, cb, quiz, categoryJoin, conditions, query);

        searchByCategory(categories, cb, categoryJoin, conditions);

        searchByDifficulty(difficulty, cb, difficultyJoin, conditions);

        searchByPrice(paidQuiz, cb, quiz, conditions);
```

Figuur 63: Criteria builder

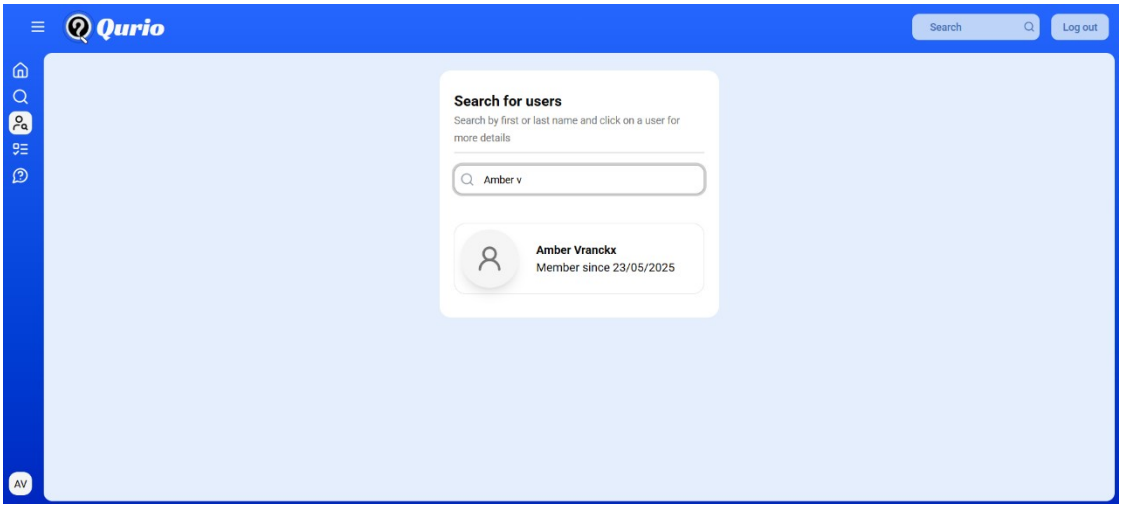
Na het overlopen van deze methodes, wordt er gecontroleerd of de lijst met condities niet leeg is. Vervolgens worden de filtercondities toegepast op de query. Daarna wordt er nog gecontroleerd of er gesorteerd moet worden. Tot slot wordt de query uitgevoerd.

```java
        if (!conditions.isEmpty()) {
            query.where(conditions.toArray(new Predicate[0]));
            try {
                if ("popularity".equals(sort)) {
                    query.groupBy(quiz.get("id"));
                    if ("desc".equals(direction)) {
                        query.orderBy(cb.desc(cb.count(userQuizJoin.get("quiz"))));
                    } else if ("asc".equals(direction)) {
                        query.orderBy(cb.asc(cb.count(userQuizJoin.get("quiz"))));
                    }
                } else {
                    if ("asc".equals(direction)) {
                        query.orderBy(cb.asc(quiz.get(sort)));
                    } else if ("desc".equals(direction)) {
                        query.orderBy(cb.desc(quiz.get(sort)));
                    }
                }

            } catch (IllegalArgumentException | IllegalStateException e) {
                log.error(e.getMessage());
                throw new IllegalArgumentException(e);
            }
        }

        TypedQuery<QuizEntity> typedQuery = entityManager.createQuery(query);
        return typedQuery.getResultList();
    }
```

Figuur 64: Toepassen van condities en sorteren

2.1.6. Gebruiker zoeken

Naast de zoekpagina voor de quiz, is er eveneens een zoekpagina voor de gebruikers voorzien. Via deze zoekpagina kan een gebruiker andere gebruikers opzoeken en hun profiel bekijken.

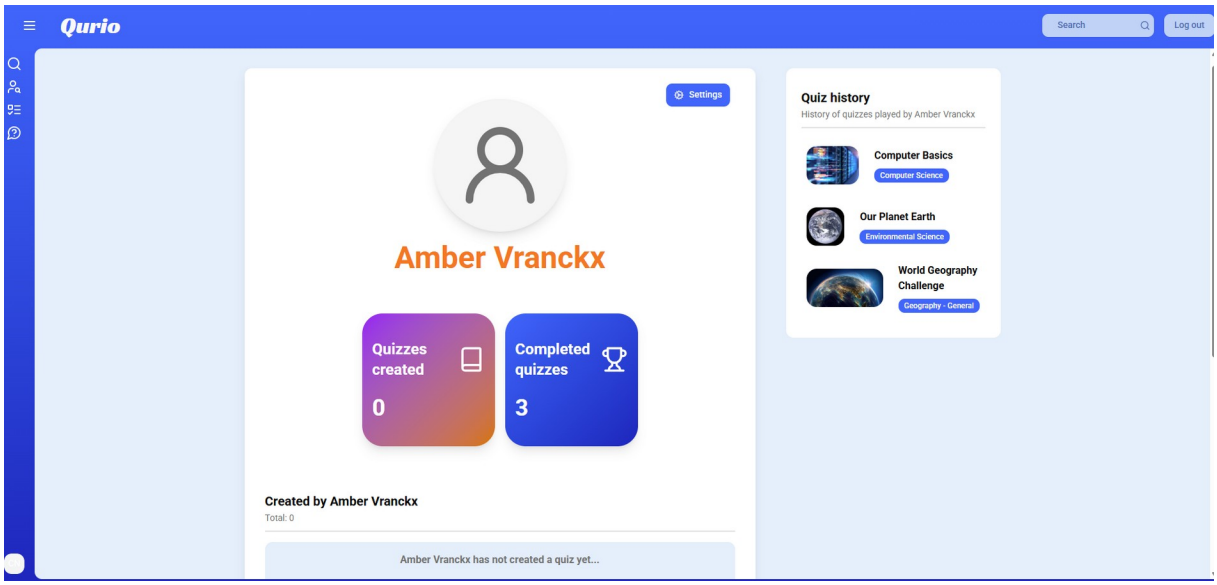


Figuur 65: Gebruiker zoeken

2.1.7. Profiel bekijken

Elke gebruiker heeft zijn eigen profiel. Hierop staan de quizzen die door die gebruiker gemaakt en gespeeld zijn. Als de gebruiker naar zijn eigen profiel navigeert, kan die naar de instellingen gaan. Hier kunnen alle persoonlijke gegevens beheerd worden. Verder kan de gebruiker hier zijn e-mailadres aanpassen, zijn wachtwoord veranderen en zijn account deactiveren.

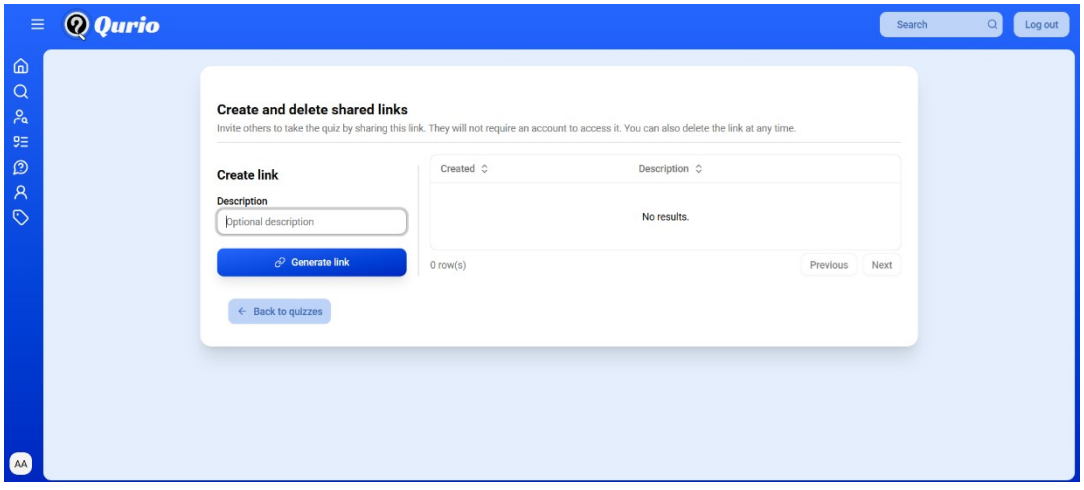
Om veiligheidsredenen hebben we besloten om bij het veranderen van de email ook het wachtwoord te vragen. Hetzelfde geldt bij het veranderen van het wachtwoord en het deactiveren van het account. Als de gebruiker zijn wachtwoord echt niet meer weet, moet deze gebruik maken van het wachtwoord-vergeten mechanisme.



Figuur 66: Profiel

2.1.8. Gebruikers uitnodigen voor een quiz

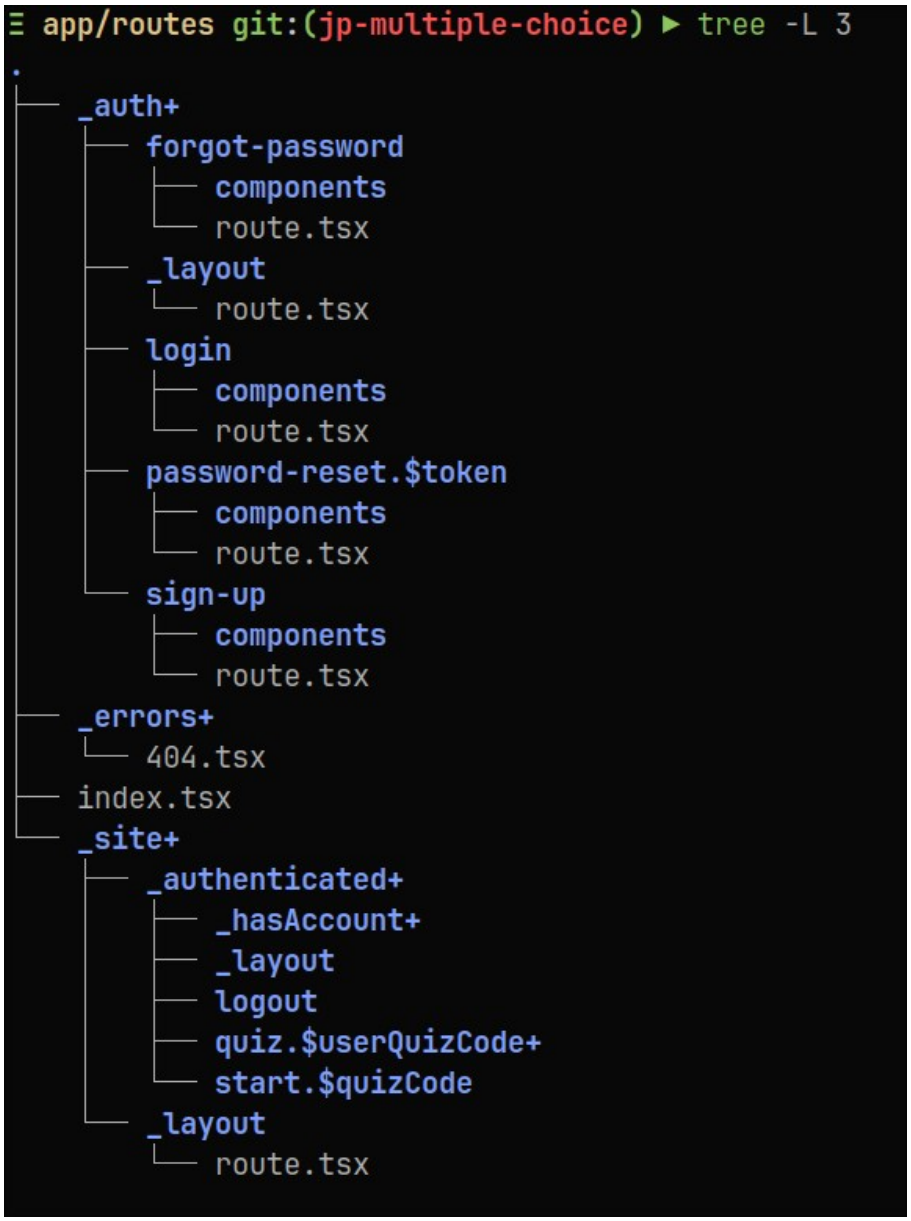
De quizapplicatie is toegankelijk voor mensen met een account. Er is echter ook een mechanisme dat toelaat dat mensen een quiz kunnen invullen, zonder een account aan te maken. Hiervoor moet de maker van de quiz een link genereren. Deze link heeft zijn eigen unieke UUID, die wordt opgeslagen in de database. Gebruikers krijgen met deze link toegang tot de quiz. Hoewel de gebruiker geen account heeft, willen we wel alle antwoorden kunnen linken aan elkaar. Daarom genereren we in de frontend per gebruiker nog een unieke code. Deze wordt bijgehouden voor ieder antwoord.



Figuur 67: Aanmaken van een link voor een quiz

Dit mechanisme voor de authenticatie hebben we in de frontend volledige geïmplementeerd via lay-out-componenten. Dit zijn componenten die hergebruikt worden voor verschillende pagina's. We hebben dus een deel van de applicatie dat niet toegankelijk is zonder authenticatie (alles onder de *authenticated+ map)*. Hierbij is zowel authenticatie via invite-link of account mogelijk. Hieronder hebben we dan nog een nieuwe lay-

out die alle pagina's bevat waar enkel personen met een account toegang tot hebben *(hasAccount+)*, dit is het grootste deel van de applicatie. Het handige aan deze geneste routes is dat we zowel het ontwerp als de toegang binnen zo'n lay-out kunnen hergebruiken. Zo krijgen alle *auth*-pagina's (*login, signup, etc.*) hetzelfde ontwerp en kunnen we de *sidebar* makkelijk voor heel de applicatie gebruiken.



Figuur 68: Routes

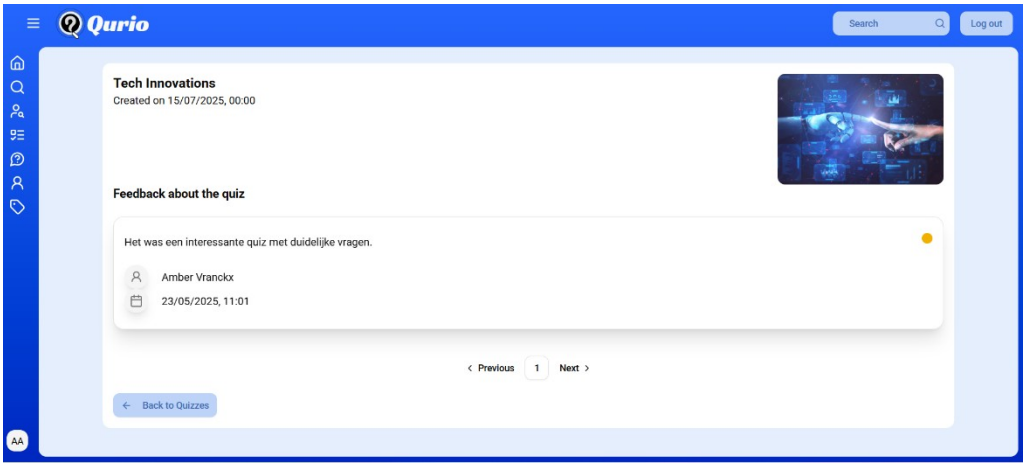
Het beheer van de gedeelde links gebeurt per quiz. Op de pagina kan de gebruiker links aanmaken en hiervoor een optionele beschrijving geven. Verder kunnen de links verwijderd worden of gekopieerd worden naar het klembord.

2.1.9. Feedback geven

Daarnaast is er een functionaliteit om feedback te geven voorzien. Er kan pas feedback gegeven worden als de gebruiker een quiz speelt of heeft gespeeld. De gebruiker kan feedback geven op een specifieke vraag van een quiz of op de quiz in zijn geheel.

2.1.10. Feedback raadplegen

Naast het geven van feedback is er een functionaliteit voorzien voor het raadplegen van deze feedback. Elke quiz heeft een pagina waarop de feedback wordt verzameld. Bovenaan staat de feedback die gegeven is op de volledige quiz. Onderaan wordt de feedback per vraag verzameld. Om het voor de maker van een quiz duidelijker te maken wanneer er nieuwe feedback is, hebben we een kleine notificatie badge voorzien.



Figuur 69: Feedback raadplegen

2.1.11. Statistieken raadplegen

Een eigenaar van een quiz kan de statistieken van een bepaalde quiz raadplegen. Op deze pagina vindt de eigenaar onder andere de gemiddelde score, de mediaan van de score, de hoogste score, de laagste score, het aantal deelnemers en de gemiddelde tijd. Bovendien staat er een tabel op deze pagina waarin de gebruiker kan bekijken wie zijn quiz heeft gespeeld en wat zijn score was. Er is tevens de mogelijkheid om de antwoorden van deze gebruiker te bekijken.

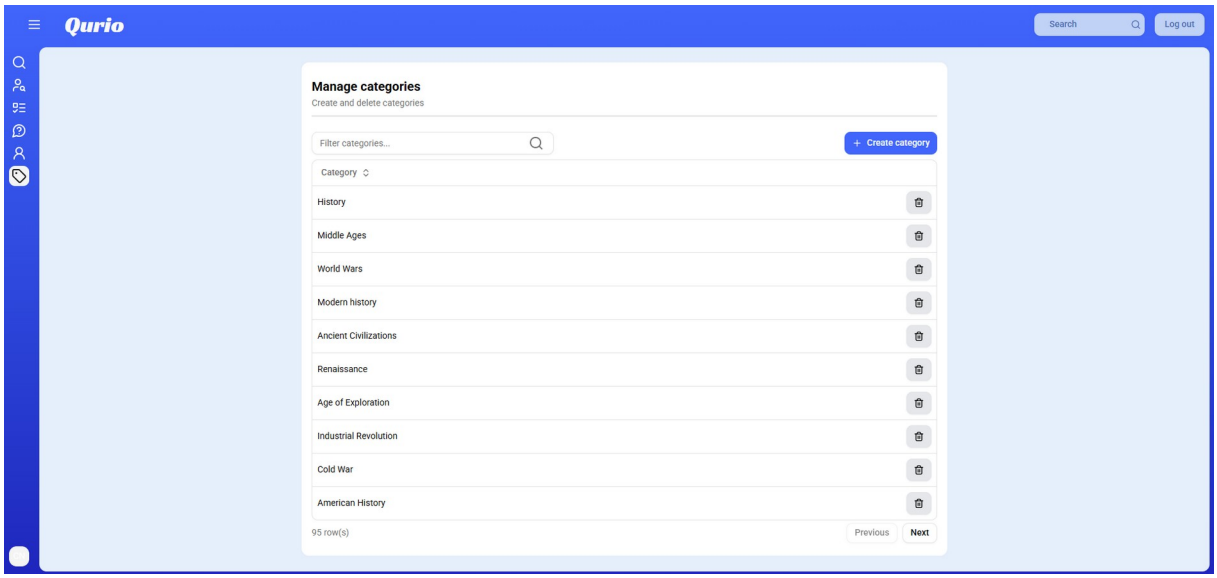
2.1.12. Gebruikers beheren

Voor een administrator is er de mogelijkheid om alle gebruikers van de applicatie op te vragen. Hier kan hij meer informatie terugvinden, zoals het e-mailadres, de voornaam, de achternaam en het type. Daarnaast kan de administrator een gebruiker zoeken via de voornaam, achternaam, e-mailadres of het type van gebruiker. Standaard worden enkel de actieve gebruikers getoond, maar de administrator kan gebruikmaken van de switch-knop op de pagina om alle gearchiveerde gebruikers te bekijken. Op deze pagina heeft de

administrator de mogelijkheid om een gebruiker te archiveren. Hierbij worden eveneens alle vragen en quizzen van die gebruiker gearchiveerd en verliest de gebruiker de mogelijkheid om in te loggen. Tot slot kan de administrator de gearchiveerde gebruikers opnieuw activeren. Ze krijgen dan opnieuw de mogelijkheid om in te loggen. De quizzen en vragen van deze gebruiker blijven echter wel gearchiveerd.

2.1.13. Categorieën beheren

Naast het beheren van de gebruikers, kan de administrator de categorieën beheren. Hierbij kan hij een nieuwe categorie aanmaken. Hiervoor moet er enkel een unieke naam voor de categorie ingegeven worden. Daarnaast heeft de administrator de mogelijkheid om een categorie te verwijderen.



Figuur 70: Categorieën beheren

2.2. Testen

Ons stagebedrijf vond het zeer belangrijk om al onze code uitvoerig te testen. Daarom hebben we zowel unittesten, integratietesten en end-to-end testen geschreven. Het eindtotaal waren **312 testen**, waaronder:

- 63 unittesten
- 187 integratietesten
- 62 end-to-end testen.

Spring boot komt gebundeld met een collectie van tools voor het schrijven en uitvoeren van testen. Voor unittesten gebruikt het *Junit 5* samen met *Mockito*. Aangezien unittesten enkel specifieke onderdelen van de code testen, moeten we alle interactie met externe systemen nabootsen. Het genereren van deze nagebootste antwoorden is de functie van *Mockito*. Dit hebben we ook gebruikt om al onze unittesten te schrijven.

Het doel van integratietesten daarentegen is om de interactie tussen verschillende systemen te testen. Hiervoor moet de hele applicatie gestart worden. Onze applicatie heeft natuurlijk een database nodig. Om een database te starten bij het runnen van de integratietesten, hebben we gebruikgemaakt van *TestContainers* in combinatie met *Junit 5*. *TestContainers* is een bibliotheek die lichte databases aanmaakt voor het uitvoeren van tests. Hierdoor kunnen we deze ook uitvoeren in de pipeline. (Getting started with Testcontainers for Java, sd)

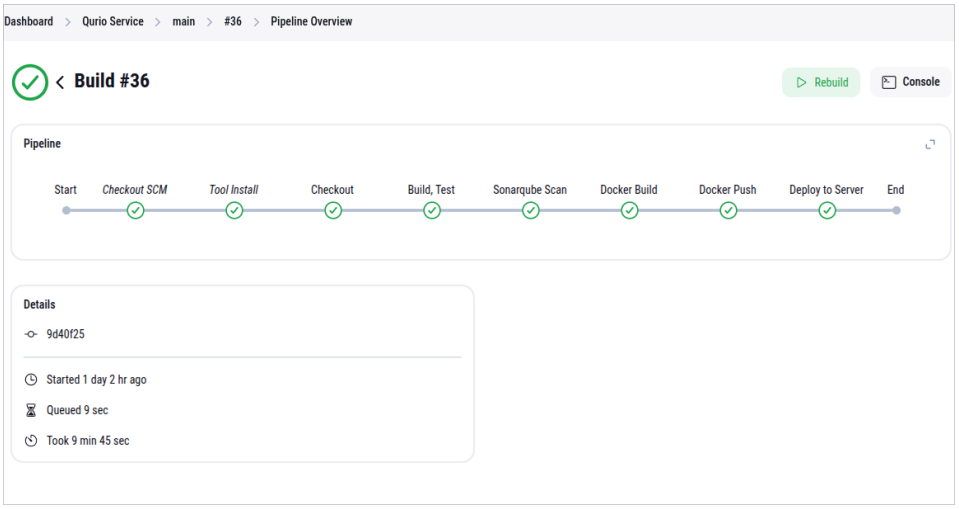
2.2.1. Pipeline

Al deze testen werden automatisch uitgevoerd in onze pipeline. De laptop die de testen uitvoerde had een versie van Jenkins. Deze voerde alle stappen uit die in de *jenkinsfile* van de repository gedefinieerd stonden.

Figuur 71: Laptop op kantoor

Onze pipeline voerde de volgende stappen uit in volgorde:

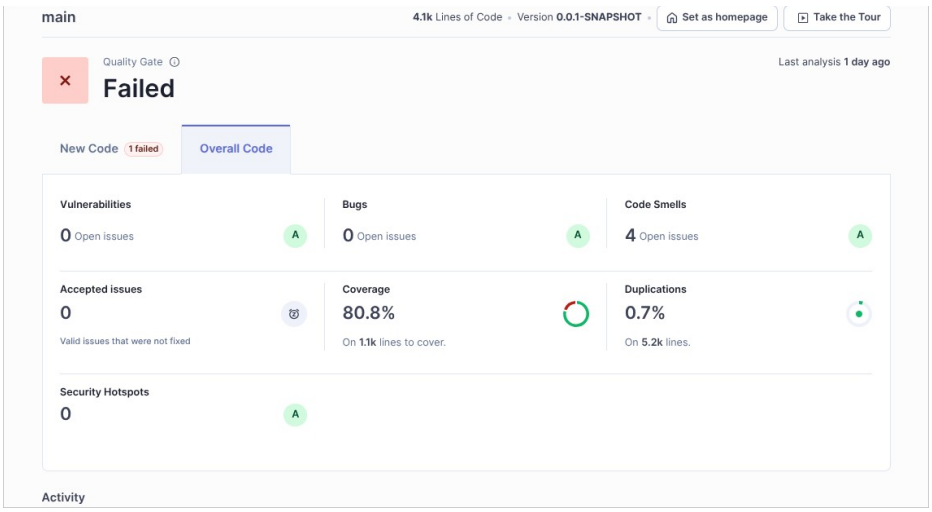
- Code clone
- Bouw het project
- Voer testen uit
- SonarQube analyse (als *main-branch)*
- Bouw container
- Push container naar Docker Hub (als *main-branch*)
- Herdeploy applicatie op laptop (als *main-branch)*



Figuur 72: Pipeline

Op de laptop hadden we niet de juiste permissies om Jenkins te installeren, daarom hadden we besloten om dit in een container uit te voeren. Achteraf bleek wel dat dit de situatie ingewikkelder zou maken, aangezien Jenkins zelf ook toegang moet hebben tot Docker. Dit probleem hebben we uiteindelijk opgelost via Docker-in-Docker (DinD). Hiermee wordt de docker-engine zelf uitgevoerd in een container. Vanuit de jenkins-container kunnen we dan verbinden met de docker-engine in de andere container door de *DOCKER_HOST* variabele in te stellen.

Zoals hierboven vermeld, wordt het project gescand in de pipeline met behulp van SonarQube. Dit is zelf ook een applicatie die op de laptop in een container opstaat. In onze projectconfiguratie hebben we *Jacoco* ingesteld. Dit is een tool die kan berekenen hoeveel procent van alle code getest is. SonarQube gebruikt de gegevens hiervan, en gaat zelf ook analyses uitvoeren om een overzicht te geven van de kwaliteit van de code. Tijdens het project hebben we ook tijd gespendeerd aan het oplossen van de fouten die gedetecteerd werden door SonarQube. Deze ‘fouten’ variëren van manieren om code te organiseren *(code smells*), tot veiligheidsproblemen.



Figuur 73: SonarQube

2.2.2. Testversie applicatie

Op de laptop zelf stond ook een versie van de applicatie, dit was onze *productie*-omgeving. Deze omgeving hebben we gebruikt voor het geven demo's en werd gebruikt door de medewerkers om te testen. Om dit uit te voeren hebben we gebruik gemaakt van *docker compose*. Ook hebben we ook een reverse-proxy ingesteld met *nginx*. Dit was nodig om de backend en frontend beiden op één domein te krijgen. Dit laatste was belangrijk omdat we maar één domein ter beschikking hadden. Dit domein hebben we via *ngrok* verkregen. Door enkel het adres van de proxy via *ngrok* publiek toegankelijk te maken en de proxy correct te configureren, hadden we een werkende versie van de applicatie die voor iedereen toegankelijk was. Dit alles zonder het openen van poorten of het aankopen van een cloudomgeving. Deze omgeving hebben we gebruikt voor het testen van de applicatie. Medewerkers konden dan bugs en feedback rapporteren in een document.

Het opstellen van deze testversie van de applicatie was eveneens een suggestie van een medewerker. Het zorgde ervoor dat we situaties met meerdere gebruikers konden simuleren en de efficiëntie van de applicatie konden evalueren.

3. Besluit

In dit document werd de stage bij Lykios besproken. Tijdens de stageperiode van dertien weken hebben we gewerkt aan een nieuw quizplatform, genaamd Qurio. Deze periode bood een uitgelezen kans om onze technische vaardigheden verder aan te scherpen, met name door de praktische toepassing in een realistische projectomgeving.

Een aspect dat tijdens de stage expliciet de aandacht kreeg, was de focus op het testen van onze code. Door het schrijven en herhaaldelijk uitvoeren van uitgebreide testen hebben we de functionaliteit en betrouwbaarheid van ons quizplatform verbeterd. In tegenstelling tot eerdere projecten, waar testen minder prominent aanwezig was, hebben we nu het cruciale belang van een robuust testraamwerk ten volle ondervonden.

Dankzij de hulp van de vriendelijke medewerkers bij Lykios hebben we een mooi resultaat kunnen afleveren en werd het leerproces versneld. De communicatie met de medewerkers was daarom ook een van de hoogtepunten van de stage.

Ten slotte hebben we met regelmatig overleg en het gezamenlijk overlopen van code, goed georganiseerde en consistente code afgeleverd. Bovendien hebben we alle belangrijke beslissingen en toekomstige functionaliteiten gedocumenteerd. Hierdoor kunnen andere ontwikkelaars in de toekomst voortbouwen aan de applicatie.

Terugblikkend op de stage zijn we erg blij met het behaalde resultaat. Hoewel het ontwikkelproces trager verliep dan op voorhand verwacht, hebben we meer ervaring opgedaan, feedback genoten en waardevolle lessen geleerd die ons in de toekomst zeker van pas zullen komen.

Daarnaast hebben we gedurende de stage gewerkt volgens de Scrum-methodiek. Deze agile aanpak, met korte sprints en frequente feedbackmomenten, stelde ons in staat om flexibel te blijven in onze ontwikkeling, snel onze functionaliteiten konden tonen aan de klant en terug te blikken op het geleverde werk om hieruit te leren. Het gebruik van projectmanagementtools zoals Jira was essentieel om overzicht te bewaren, prioriteiten te stellen en onze voortgang te monitoren, wat de efficiëntie van ons werk ten goede kwam.

De stage bij Lykios heeft een grote impact gehad op onze professionele voorbereiding. Het plannen en opstarten van een nieuw project, het navigeren door de dynamiek van een agile project, en de sterke nadruk op codekwaliteit met testen en onderhoudbaarheid, hebben ons waardevolle inzichten gegeven. We zijn erg enthousiast over de ervaring die we hebben opgedaan en zijn ervan overtuigd dat we de opgedane kennis en vaardigheden succesvol zullen kunnen inzetten in toekomstige functies en bij het grijpen van nieuwe professionele kansen.

Literatuurlijst

9 Best Scrum Tools To Master Project Management | Atlassian. (sd). Opgehaald van Atlassian: https://www.atlassian.com/agile/project-management/scrum-tools

About. (sd). Opgehaald van Swagger: https://swagger.io/about/

Bitbucket Overview | Bitbucket. (sd). Opgehaald van Bitbucket: https://bitbucket.org/product/guides/getting-started/overview#a-brief-overview-of-bitbucket

Getting started with Testcontainers for Java. (sd). Opgehaald van Testcontainers: https://testcontainers.com/guides/getting-started-with-testcontainers-for-java/

Git - A Short History of Git. (sd). Opgehaald van Git: https://git-scm.com/book/ms/v2/Getting-Started-A-Short-History-of-Git

How We Built Postman - the Product and the Company | Postman Blog. (2021, April 5). Opgehaald van Postman: https://blog.postman.com/how-we-built-postman-product-and-company/

IntelliJ IDEA overview | IntelliJ IDEA Documentation. (2025, April 7). Opgehaald van IntelliJ IDEA: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

Jenkins. (sd). Opgehaald van Jenkins: https://www.jenkins.io/

Kashyap, P. (2024, December 6). *Comprehensive Guide to SonarQube: Understanding, Benefits, Setup, and Code Quality Analysis with FastAPI*. Opgehaald van Medium: https://medium.com/@piyushkashyap045/comprehensive-guide-to-sonarqube-understanding-benefits-setup-and-code-quality-analysis-with-caffbc8afa0f

nginx. (sd). Opgehaald van nginx: https://nginx.org/en/

Soni, S. (2025, Januari 17). *What is Ngrok and how does it work?* Opgehaald van Requestly: https://requestly.com/blog/what-is-ngrok-and-how-does-it-work/#what-is-ngrok-1

Stevens, E. (2024, Augustus 23). *What is Figma - Uses, benefits, and Features | AND Academy*. Opgehaald van AND Academy: https://www.andacademy.com/resources/blog/graphic-design/what-is-figma/

Technology | 2024 Stack Overflow Developer Survey. (2024). Opgehaald van 2024 Developer Survey: https://survey.stackoverflow.co/2024/technology#1-integrated-development-environment

Usage Statistics and Market Share of Web Servers, May 2025. (2025, Mei 16). Opgehaald van W3Techs: https://w3techs.com/technologies/overview/web_server

What is Tailscale? Tailscale Docs. (2025, April 17). Opgehaald van Tailscale: https://tailscale.com/kb/1151/what-is-tailscale