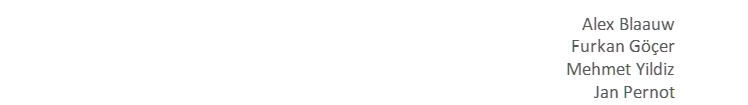
Afbeelding met tekst, buiten, heuvel, helling

Automatisch gegenereerde beschrijving​​

# Inhoudsopgave

Inhoud

[Inhoudsopgave 1](#_Toc125322406)

[Theater Laak 2](#_Toc125322407)

[Links 2](#_Toc125322408)

[​Ontwerp-rapport 2](#_Toc125322409)

[Inleiding 2](#_Toc125322410)

[Frontend 3](#_Toc125322411)

[Models 9](#_Toc125322412)

[​Controllers 10](#_Toc125322413)

[Privacy & Security 13](#_Toc125322414)

[Extra functionaliteit 14](#_Toc125322415)

# Theater Laak

Deze webapplicatie is ontworpen voor Theater Laak en biedt een gebruiksvriendelijke oplossing voor de theaterbezoekers om kaartjes te kopen en de programmering te bekijken. Daarnaast biedt de webapplicatie de mogelijkheid voor mensen of organisaties om zalen te reserveren voor eigen evenementen. Bezoekers kunnen een account aanmaken en zo makkelijker kaartjes kopen en inzien. Dit helpt Theater Laak om haar bezoekers beter te bedienen en hen te informeren over voorstellingen.

Daarnaast heeft Theater Laak een loyale groep donateurs die zorgen voor toegankelijkheid van het culturele programma voor mensen uit het Laakkwartier en omgeving. Voor hen is er de optie om het theater financieel te steunen. Die donaties zijn wat het theater draaiende houdt.

De programmering van Theater Laak is elk seizoen van x maand tot y maand. Theater Laak beschikt over 4 zalen, elk met een eigen grootte en capaciteit. Er zijn ook 10 ruimtes die vrij in te delen zijn met elk een capaciteit van maximaal 30 personen. De programmering wordt aangeleverd in een Excel bestand waarna deze wordt ingevoerd in de webapplicatie. Dit wordt dan automatisch omgezet naar de database en aan de bijbehorende onderdelen gekoppeld. Dit maakt het voor Theater Laak gemakkelijk om de programmering te beheren en de beschikbaarheid van de zalen te regelen.

# Links

​Website: <https://77.172.8.98:44469/>

​

Repo: <https://github.com/MehmetAliYildiz/WDPR2.git>

​Planning: <https://www.notion.so/7ea1aec9d5f8481185ac6eda86ff341d?v=b8e492a6834f4c1c8009cc7dfd672106>

​ ​

# ​Ontwerp-rapport

## Inleiding

Deze webapplicatie is ontworpen als een .NET MVC-applicatie, waarbij de Model-View-Controller-architectuur wordt gebruikt voor de opbouw van de applicatie. Hierdoor is de applicatie opgebouwd uit verschillende lagen, waaronder de model-laag voor de gegevens, de view-laag voor de presentatie en de controller-laag voor de logica.

Voor de opslag van gegevens maakt de applicatie gebruik van een MySQL-database, waarbij Entity Framework Core (EF Core) wordt gebruikt als ORM (Object-Relational Mapping). Dit maakt het mogelijk om gegevens in de database op een eenvoudige manier te beheren en te manipuleren vanuit de applicatie. Ook stelt dit ons in staat om code-first te werken, waarna het framework de database genereerd. Om de database efficiënt te kunnen benaderen, hebben we gebruik gemaakt van LINQ. LINQ maakt het mogelijk om verschillende data uit de database te filteren, ordenen of aanpassen middels eenvoudige syntax. Hierdoor is de brug tussen onze programmatuur en de database klein.

De front-end van de applicatie is gebouwd met React, een JavaScript-library voor het bouwen van gebruiksvriendelijke en interactieve gebruikersinterfaces.

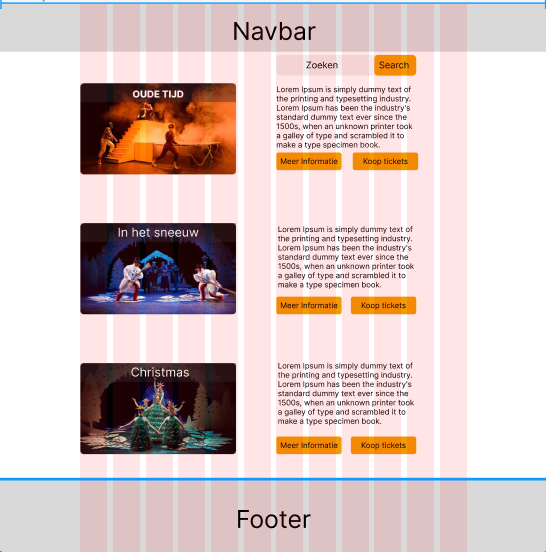
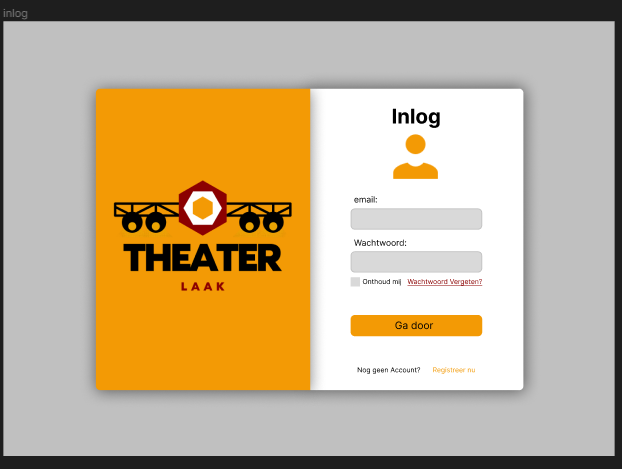
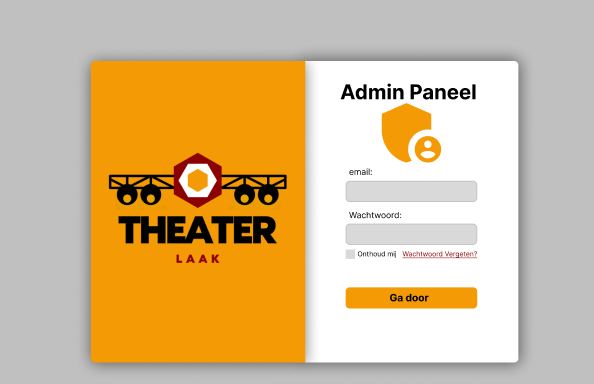
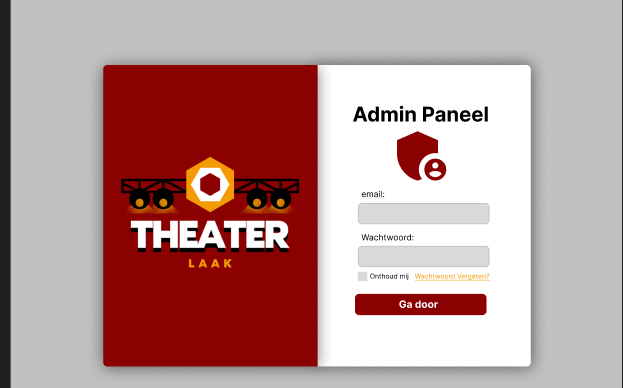
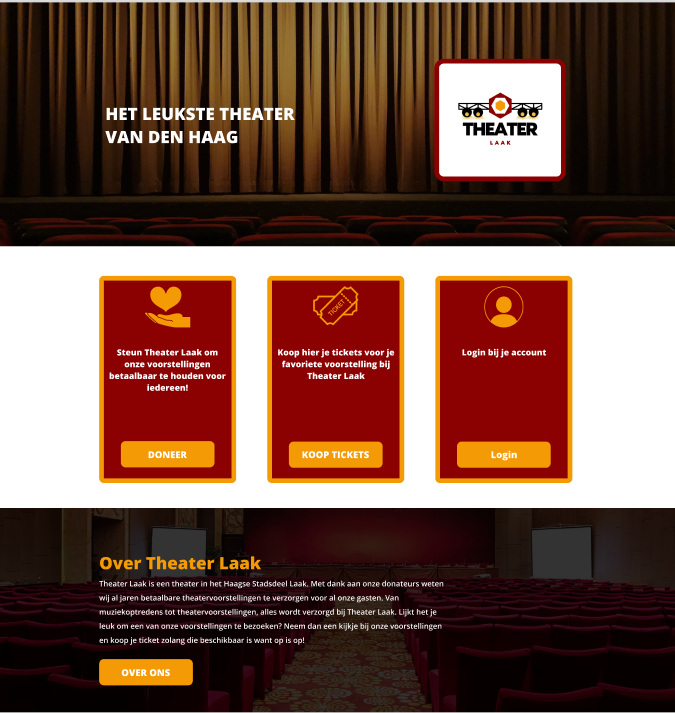
Deze verschillende libraries en frameworks maken de applicatie van Theater Laak een geheel.

## Frontend

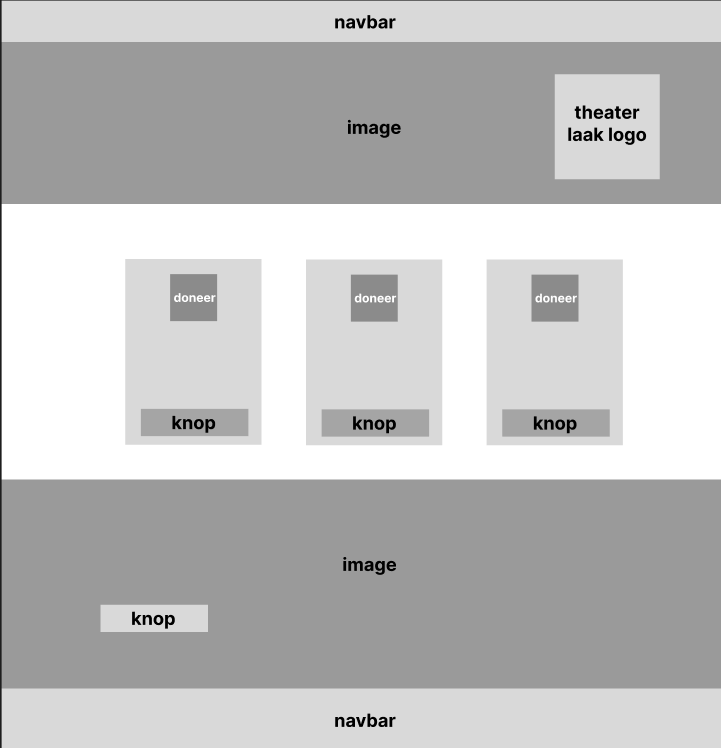
De applicatie van Theater Laak is responsief, waardoor deze toegankelijk is op verschillende apparaten.

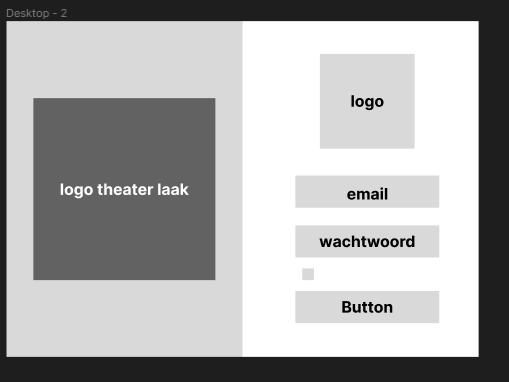
Daarnaast is onze applicatie ontworpen met toegankelijkheid voor mensen met een beperking in het achterhoofd. Dit betekent dat we gebruik maken van standaard best practices voor toegankelijkheid, zoals een heldere en eenvoudige navigatie, grotere tekst, en contrasterende kleuren. Op het hoofdmenu is bijvoorbeeld gezorgd voor goed contrast tussen de knoppen en de tekst in de knoppen. Zo kunnen mensen met een visuele beperking ook goed gebruik maken van de applicatie. Verder hebben we overal op de website genoeg white-space gebruikt zodat alle content goed op elkaar aansluit en de website een rustigere uitstraling heeft.

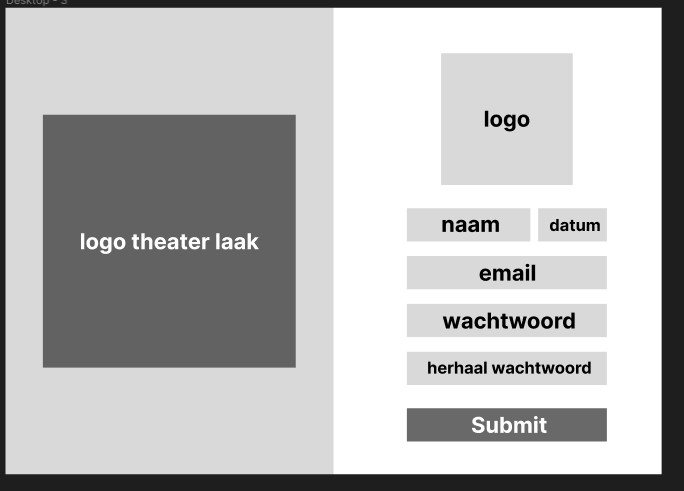
**Wireframe & Figma**











## Models

De data van de applicatie bestaat uit verschillende modellen. Deze modellen zijn het volgende:

1. Agenda: Een agenda bevat een Voorstelling en een Zaal. De agenda koppelt feitelijk deze twee aan elkaar en bevat een uniek Id om de Agenda te identificeren. De Agenda is een tussentabel tussen Voorstelling en Zaal. Er is voor een Agenda model gekozen omdat het belangrijk is dat dezelfde Voorstelling meerdere keren kan worden gegeven in verschillende zalen. Zonder een tussentabel zou dit onmogelijk zijn.
2. Artiest: Een artiest bevat de naam van een artiest. Een artiest is lid van een band. De band kan bestaan uit meerdere artiesten of uit slechts een enkel artiest. Er is voor een artiesten model gekozen omdat een artiest degene is die het optreden, alleen of met meerdere artiesten, verzorgd.
3. Band: De band treed op bij een voorstelling en bevat een naam en een lijst met artiesten die bij de band horen. Het band model is belangrijk omdat een Voorstelling kan worden gegeven door meerdere artiesten. Verder bevat een band ook een naam die moet worden opgeslagen in de applicatie.
4. Bestelling: Een bestelling bevat een bedrag, een waarde die checkt of de bestelling betaald is en de dag en tijd van de bestelling.
5. Gebruiker: De gebruiker is degene die gebruik maakt van de applicatie en die vrije ruimtes en kaartjes kan bestellen. In een gebruiker zit een naam, wachtwoord en email. De Gebruiker is een belangrijk model in de applicatie omdat dit de gegevens zijn van onze gebruikers. Hun data wordt gebruikt voor o.a. het toekennen van kaartjes aan de juiste gebruiker.
6. Gebruikerslogin: Een gebruiker logt in met een e-mail en wachtwoord en de gebruiker heeft de keuze om ingelogd te blijven. Wanneer een gebruiker een succesvolle inlogpoging heeft gedaan, wordt de JWT-token opgeslagen. Hierna kan de e-mail uit de JWT-token worden gehaald om een gebruiker te identificeren op andere pagina’s en zo data van een specifieke gebruiker op te vragen.
7. GebruikerMetWachtwoord: De GebruikerMetWachtwoord wordt gebruikt bij het registreren van een nieuw account.
8. Kaartje: Wordt aangemaakt op basis van een Agenda item. In een Kaartje zit een Agenda, een lijst met StoelKaartjes en een Bestelling die bij het kaartje hoort. Het Kaartje model is aangemaakt omdat we zo een Bestelling kunnen koppelen aan een Agenda en StoelKaartjes model.
9. Reservering: Een Reservering is een reservering van een VrijeRuimte. In de Reservering zit een ZaalId, de naam, de begin- en eindtijd van de Reservering en de bijbehorende Bestelling.
10. Stoel: Een Stoel is een stoel die in een Zaal staat. De Stoel heeft een status om bij te houden of deze bezet is of niet. Verder heeft de stoel een Row en een Rang die bij houden wat de rij en rang van de stoel is. De rang van een Stoel kan 1e, 2eof 3e rangs zijn. Verder zit bij een Stoel een lijst van StoelKaartjes om bij te houden welke StoelKaartjes bij deze Stoel horen.
11. StoelKaartje: Een StoelKaartje is een kaartje voor een Stoel. In een StoelKaartje zit een Kaartje en een Stoel. StoelKaartje fungeert als een tussentabel tussen deze twee. Het StoelKaartje stelt de applicatie in staat om een specifieke Stoel aan een Kaartje te koppelen.
12. Voorstelling: De Voorstelling is de voorstelling die wordt gegeven. In een Voorstelling zit een naam, beschrijving en plaatje. De voorstelling is een los model omdat er bij een Voorstelling specifieke data moet worden opgeslagen. Ook kunnen we een Voorstelling model gebruiken in een Agenda object zodat we de Voorstelling meerdere keren, in verschillende zalen kunnen geven.
13. VrijeRuimte: De VrijeRuimte is de ruimte die gebruikers kunnen huren. De VrijeRuimte is een apart model zodat gebruikers deze kunnen reserveren.
14. Zaal: Een Zaal is een model waar Voorstellingen in worden gegeven. In een Zaal zit een lijst met Stoelen. Hierdoor kan een zaal meerdere Stoelen hebben. De Zaal is een apart model omdat in een Zaal meerdere voorstellingen op verschillende tijdsstippen moeten kunnen worden gegeven.

​

## ​Controllers

Alle modellen worden beheerd door de controllers. De controller van elk model heeft specifieke logica die voldoen aan de requirements van Theater Laak.

1. **AccountController**

De AccountController is een onderdeel van de webapplicatie voor Theater Laak en is verantwoordelijk voor de gebruikersaccounts. De controller biedt functionaliteiten voor het inloggen van bestaande gebruikers en het registreren van nieuwe gebruikers.

De inlogmethode van de controller ontvangt een object van het type GebruikerLogin, waarmee een gebruiker zich kan aanmelden. Vervolgens wordt gecheckt of de e-mail en wachtwoord bij elkaar horen en als dat het geval is, wordt een JWT gegenereerd. Deze token bevat informatie over de gebruiker, zoals de e-mail en de rollen waar de gebruiker aan is toegevoegd.

De registreermethode van de controller ontvangt een object van het type GebruikerMetWachwoord, waarmee een nieuwe gebruiker kan worden geregistreerd.

Een JWT is een veilige en efficiënte oplossing voor het authenticeren van gebruikers. JWT's zijn compact, zelf-bevattend en kunnen gemakkelijk worden geverifieerd zonder dat er een verbinding met de server hoeft te worden gemaakt. Dit zorgt voor een snellere en efficiëntere verificatie van gebruikers en vermindert de belasting van de server. Dit is de reden waarom wij voor JWT’s hebben gekozen.

1. **AgendaController**

De AgendaController is verantwoordelijk voor het Agenda model van Theater Laak. Het biedt functionaliteiten voor het ophalen van agendapunten op basis van een voorstelling Id, het toevoegen van een nieuwe Agenda en het ophalen van de volledige Agenda.

De [HttpGet("voorstelling/{id}")] methode zorgt voor het ophalen van alle Agenda’s die horen bij de voorstelling met de ingevoerde Id. Hierdoor is het mogelijk om de programmering van een specifieke voorstelling te bekijken.

De [HttpPost] methode maakt het mogelijk om een Agenda toe te voegen met een Zaal en een Voorstelling. Hierbij wordt met LINQ functies gecontroleerd of de ingevoerde voorstelling en zaal bestaan en of de ingevoerde tijd niet al in gebruik is door een andere Voorstelling of Agenda. Dit voorkomt dat voorstellingen worden ingepland op momenten dat een zaal al in gebruik is en dat een voorstelling niet in twee zalen op hetzelfde moment kan worden gegeven.

1. **ArtiestController**

Deze controller biedt de functionaliteit voor het beheren van artiesten. Dit omvat het ophalen van alle artiesten, het ophalen van een specifieke artiest via een Id, het toevoegen van een nieuwe artiest en het verwijderen van een bestaande artiest.

1. **BandController**

De BandController is verantwoordelijk voor het beheren van bands in de applicatie. Dit wordt gedaan middels de volgende functionaliteit:

* Opvragen van een lijst met alle bands.
* Aanmaken van een nieuwe band.
* Wijzig een bestaande band.
* Voeg een artiest toe aan een band.
* Verwijder een specifieke band uit de database.

1. **GebruikerController**

De GebruikerController is verantwoordelijk voor het beheren van gebruikers in de applicatie. Dit wordt gedaan door middel van de volgende functionaliteit:

* Opvragen van een lijst met gebruikers.
* Opvragen van een specifieke gebruiker.
* Het aanpassen van een bestaande gebruiker.
* Het aanmaken van een nieuwe gebruiker.
* Het verwijderen van een specifieke gebruiker.

1. **KaartjeController**

In de KaartjeController wordt gebruik gemaakt van de \_boekingHubContext. Hier is voor gekozen omdat het zo mogelijk is om te zien welke gebruikers er op dit moment zijn verbonden met de websocket en dus op de stoelboeken pagina aanwezig zijn. Met deze lijst van verbonden gebruikers kan gecommuniceerd worden welke stoelen er al geboekt zijn. Hierdoor kunnen stoelen live worden geüpdatet.

Deze controller heeft verder nog de volgende functionaliteit:

* Het ophalen van een kaartje met een opgegeven Id.
* Het aanmaken van een nieuw kaartje met de opgegeven informatie.

1. **ReserveringController**

De ReserveringController is speciaal ontworpen voor het reserveren van een VrijeRuimte. In de controller worden twee endpoints gebruikt:

* De eerste endpoint geeft een lijst met reserveringen terug voor een specifieke zaal en datum. De id van de zaal en de datum worden meegegeven als route parameters.
* De tweede endpoint maakt een nieuwe Reservering van een VrijeRuimte aan. De gegevens van de reservering worden meegegeven als body van de request. Er wordt eerst met LINQ gecheckt of de starttijd vóór de eindtijd ligt. Daarna wordt gecontroleerd of er geen overlappende reserveringen zijn in de database. Als er geen overlappende reserveringen zijn, dan wordt de reservering toegevoegd aan de database.

1. **StoelController**

De StoelController wordt gebruikt voor het opvragen of aanmaken van stoelen. Dit gebeurt bijvoorbeeld bij het aanmaken van nieuwe Zalen. De StoelController omvat twee endpoints:

* Met de eerste endpoint kan een nieuwe stoel toegevoegd worden aan de database. Er wordt een Stoel object verwacht in de request body waarmee de nieuwe stoel kan worden aangemaakt.
* Met de tweede endpoint kan de beschikbaarheid van stoelen in een bepaalde zaal op een bepaalde datum opgevraagd worden. Er worden twee route parameters verwacht: zaalId en agendaId. Hiermee kan worden gecheckt welke stoelen nog vrij zijn. Er wordt een response teruggegeven met een lijst van stoelen, waarbij de status van elke stoel "Vrij" of "Bezet" is.

Er wordt gebruik gemaakt van LINQ om specifieke stoelen of beschikbare stoelen te selecteren uit de database. Ook wordt er gecheckt of de opgegeven Zaal wel bestaat.

De status van de stoelen wordt aangepast in de response, zodat de client direct kan zien welke stoelen beschikbaar zijn.

1. **VoorstellingController**

Met de VoorstellingController kunnen de voorstellingen beheerd worden. De Voorstelling wordt gebruikt bij het aanmaken van een Agenda en staat bewust los van een Zaal. Dit stelt ons in staat om een Voorstelling meerdere keren in verschillende zalen te laten plaatsvinden.

De VoorstellingController heeft de volgende functionaliteit.

* Het opvragen van alle voorstellingen uit de database.
* Het opvragen van een specifieke voorstelling uit de database.
* Het aanpassen van een specifieke voorstelling.
* Het verwijderen van een specifieke voorstelling uit de database.
* Het verwijderen van alle voorstellingen.

1. **VrijeRuimteController**

In de VrijeRuimteController kunnen VrijeRuimtes worden beheerd. Dit omvat het toevoegen en opvragen van VrijeRuimtes.

In de VrijeRuimteController zitten drie endpoints met de volgende functionaliteit:

* Het aanmaken van een nieuwe VrijeRuimte.
* Het opvragen van alle VrijeRuimtes.
* Het opvragen van een enkele VrijeRuimte.

1. **ZaalController**

In de ZaalController kunnen zalen worden beheerd. De ZaalController worden verschillende endpoints gebruikt om de data in de database toe te voegen of te bewerken. De endpoints hebben de volgende functionaliteit:

* Het opvragen van alle of een specifieke Zaal.
* Het toevoegen van een nieuwe zaal. Hierbij verwacht de nieuwe zaal een aantal stoelen. Dit kunnen 1e, 2e, of 3e rangs stoelen zijn. Hierdoor kan een admin makkelijk nieuwe zalen toevoegen en precies aangeven hoeveel en wat voor soort stoelen er in de zaal staan.

De ZaalController heeft ook een ZaalMetStoelnummers-klasse. Deze wordt gebruikt om een nieuwe zaal met stoelnummers te maken voordat deze in de database wordt opgeslagen.

## Privacy & Security

In de applicatie zouden rollen worden gebruikt door middel van de IdentityUser. In .NET Core wordt het "Identity" framework gebruikt om gebruikersauthenticatie en -autorisatie te behandelen. Het biedt een set van klassen en interfaces voor het beheren van gebruikers en rollen, evenals voor het uitvoeren van veelvoorkomende taken zoals wachtwoordhashing en gebruikersbeheer. Het "Identity" framework is ontworpen om flexibel en uitbreidbaar te zijn, zodat ontwikkelaars het gemakkelijk kunnen aanpassen aan hun specifieke behoeften.

Dit hebben we helaas niet helemaal kunnen implementeren zoals we het wilde. Echter was het idee om dat een gebruiker een user role kreeg die geupgrade kon worden naar een donateur als hij of zij meer dan 1000 euro gedoneerd had.

Natuurlijk hebben we ook een admin rol. De admin heeft verregaande rechten binnen de applicatie. Hij of zij kan bijvoorbeeld voorstellingen en agenda’s toevoegen. Uiteraard wordt de admin rol alleen toegekend aan gebruikers die daadwerkelijk toegang nodig hebben tot alles.

Verder hebben we gebruik gemaakt van JWT’s. Op het moment dat een gebruiker een succesvolle inlogpoging doet, krijgt hij of zij een JWT. Hiermee kan hij zichzelf authentiseren. Dit betekent dat de ingelogde gebruiker daadwerkelijk de rechten heeft tot de gegevens.

## Extra functionaliteit

Theater Laak is in staat om nieuwe programmering in een Excel sheet aan te leveren. Voorstellingen en Agenda items kunnen in dezelfde sheet worden aangeleverd en worden bij het uploaden automatisch toegevoegd aan de applicatie. Als een voorstelling al bestaat, moet een VoorstellingId worden aangeleverd zodat de applicatie weet welke Voorstelling er moet worden toegevoegd aan het Agenda item. Echter kunnen ook nieuwe Voorstellingen worden aangemaakt en meteen worden toegevoegd aan een Agenda item.

​

​Test-rapport

​Testen is belangrijk omdat het helpt om de kwaliteit van de software te verzekeren voordat deze in productie wordt genomen. Testen helpt bij het opsporen van fouten en problemen in de software voordat deze zich voordoen bij eindgebruikers. Hierdoor kan de kwaliteit van de software worden verbeterd en kan er worden voorkomen dat er problemen ontstaan bij gebruik.

We hebben gekozen voor het gebruik van Cypress als testframework. Cypress is een moderne e2e-test-framework dat speciaal is ontwikkeld voor webapplicaties. Het biedt een gebruiksvriendelijke interface en een intuïtieve API.

​

​​