Siebe Camerman

**Chargeroom for EEVEE-drivers** 🔋



Plan van aanpak

Inhoudsopgave

[Inleiding 3](#_Toc163724832)

[Stagebedrijf 3](#_Toc163724833)

[Probleemstelling 4](#_Toc163724834)

[Verwachte resultaat 5](#_Toc163724835)

[Vereisten: 5](#_Toc163724836)

[Frontend 6](#_Toc163724837)

[Framework 6](#_Toc163724838)

[Nummerplaat scanner 7](#_Toc163724839)

[Backend 8](#_Toc163724840)

[Chat Technologie 8](#_Toc163724841)

[Notificaties 8](#_Toc163724842)

[Files in chat verzenden 10](#_Toc163724843)

[Authenticatie 13](#_Toc163724844)

[Hosting 13](#_Toc163724845)

[Database 14](#_Toc163724846)

[Architectuur 15](#_Toc163724847)

[Uitbreiding: dynamische chat 16](#_Toc163724848)

[Risico’s 18](#_Toc163724849)

[Planning 19](#_Toc163724850)

[Referenties 20](#_Toc163724851)

# Inleiding

## Stagebedrijf

**Wisemen** [1] is een digital agency die je kunt opsplitsen in **3** grote takken:

* **Boost leads, verkoop én sollicitaties:**

Op zoek naar een voorspelbare instroom van meer leads en klanten? Uitdagingen met het opvolgen van leads en productiviteit van jouw dealers of distributeurs? Een complex product dat een product- of offerteconfigurator nodig heeft en niet zomaar in een basic webshop past? Samen bouwen we een digitaal, effectief marketing-, en verkoopsysteem.

* **Verhogen van digitale bedrijfsefficiëntie:**

Door het digitaliseren van bedrijfsprocessen kan je de efficiëntie aanzienlijk verhogen. Cloud software op maat maakt het mogelijk om gegevens centraal op te slaan en real-time toegang te bieden aan alle teamleden, en ook mobile kan je deze digitalisering doortrekken. Automatisering van routinetaken met Artificiële Intelligentie bespaart daarnaast aanzienlijk tijd en vermindert de kans op menselijke fouten.

* **Meer customer experience:**

Of je nu B2B of B2C georiënteerd bent, je klanten verwachten een vlekkeloze digitale dienstverlening en productaanbod waar digitaal centraal staat. Kan jouw productaanbod vandaag nog zonder bijhorende app of webportaal? De klanttevredenheid wordt direct beïnvloed door jouw digitale customer experience.

Daarnaast hebben ze een dochteronderneming **EEVEE mobility** [2]. EEVEE is een applicatie met als doel laadkosten van elektrische rijders in kaart brengen en zo in staat zijn terugbetalingen efficiënter uit te voeren zonder onnodige hardware. Is gestart als side project van Wisemen.

In eerste instantie bood EEVEE een mobiele app aan waarmee het volgen van laadsessies voor Tesla-auto's mogelijk was. Dit werd gemaakt om het probleem van Steffen op te lossen, namelijk het ontbreken van een duidelijk overzicht van zijn laadgedrag en de bijbehorende kosten. Ondertussen hebben ze meer dan 100.000 gebruikers en integraties met meer dan 15 automerken.

## Probleemstelling

Het probleem is dat elektrische auto's enige tijd nodig hebben om op te laden. Dit betekent dat bestuurders hun auto vaak aan de oplaadpaal laten hangen, zelfs als ze volledig zijn opgeladen. Hierdoor is het niet altijd gemakkelijk om een vrije oplaadpaal te vinden. Een ander probleem is dat sommige mensen bijvoorbeeld een volledig opgeladen auto nodig hebben voor een afspraak op locatie. Door het ontwikkelen van een chatroom applicatie kunnen we bestuurders in staat stellen om af te spreken om auto's te wisselen.

Mijn stageopdracht is om een chatroom applicatie voor EEVEE-gebruikers te bouwen:

Deze Chargeroom for EEVEE-drivers zal een applicatie zijn waarmee EEVEE-drivers andere elektrische drivers kunnen contacteren om van plaats te ruilen om te laden. Dit om efficiënter met laadruimte om te gaan en onnodige wachttijden en verlies van stroom te vermijden.

Hoe zal dit in zijn werk gaan? Heel eenvoudig:

1. Verbind je EEVEE-account en voeg je nummerplaat / VIN nummer toe

2. Scan een andere EV-driver zijn/haar nummerplaat / VIN nummer

3. Chat met de gebruiker indien hij/zij een account heeft ("ik zou graag laden bij deze laadpaal")

4. De EV-driver krijgt een melding bij een bericht en kan aangeven wanneer hij/zij klaar is;

## Verwachte resultaat

Het doel is om een Progressieve Web App (PWA) te ontwikkelen waarbij gebruikers de nummerplaat kunnen scannen met hun telefooncamera of handmatig kunnen invoeren. Nadat de nummerplaat is ingevoerd en een geregistreerde gebruiker is gevonden, kunnen ze een chat starten. Ook moet er een intern notificatiesysteem worden gemaakt dat het mogelijk maakt om notificaties te versturen naar gebruikers. Als uitbreiding op de MVP zou het systeem volledig dynamisch kunnen worden gemaakt en gemakkelijk te implementeren zijn, met ondersteuning voor verschillende soorten chats.

## Vereisten:

In dit project wil ik me voornamelijk richten op backend ontwikkeling. Het is daarom een vereiste om de backend in Node.js te ontwikkelen, aangezien dit ook wordt gebruikt bij Wisemen. Bovendien zal de backend worden gebouwd met behulp van het Nest.js framework want het template-project van Wisemen waarmee ik zal beginnen is in Nest.js is ontwikkeld.

# Frontend

## Framework

De grote vraag is wat framework ik ga gebruiken. Bij Wisemen gebruiken ze Vue.js voor de frontend. Ik heb hier nog niet mee gewerkt en omdat mijn focus voor mijn stage op backend ligt zat ik te denken aan Next.js waar ik al wel wat ervaring mee heb. Om de keuze tussen deze 2 te maken heb ik hieronder even een vergelijking gemaakt.

|  |  |  |
| --- | --- | --- |
| Feature | Vue.js  | Next.js  |
| Type | Progressive Framework  | React framework built on React |
| Learning Curve  | Easier | Steeper, requires React knowledge |
| Performance | Excellent, small bundle size, efficient virtual DOM  | Excellent, server-side rendering and static generation |
| Community & Ecosystem  | Large and active, vast ecosystem of libraries and tools  | Benefits from the extensive React ecosystem |
| Use Cases  | General web applications, SPAs, e-commerce  | Server-rendered and statically generated applications, complex routing |
| Ideal for  | Beginners, projects valuing ease of use and flexibility  | SEO-focused projects, complex routing, React familiarity |

In het kort zijn Next.js en Vue.js beide robuuste frameworks met verschillende voordelen. Projecten die server-side rendering en optimale prestaties vereisen, met name voor grotere toepassingen, zijn waar Next.js uitblinkt. Vue.js, vanwege zijn eenvoud en mogelijkheid tot incrementele adoptie, is perfect voor projecten en teams met verschillende niveaus van expertise, van klein tot middelgroot.

Vanwege het feit dat Next.js server-side rendering gebruikt, brengt het implementeren van websockets enige uitdaging met zich mee. Aangezien Vue.js daarentegen een eenvoudigere leercurve heeft en het integreren van websockets ook makkelijker is, heb ik besloten voor dit framework te kiezen. Ook omdat ze bij Wisemen dit framework ook gebruiken en ik hierdoor makkelijk om ondersteuning kan vragen als ik ergens vastzit en gebruik kan maken van hun templateproject.

Nummerplaat scanner

Voor de nummerplaat scanner zal ik gebruik maken van het Tesseract OCR-pakket. Tesseract.js is een JavaScript library die specifiek is ontworpen voor optische tekenherkenning (OCR) en in staat is om woorden in vrijwel elke taal uit afbeeldingen te halen. [3]

# Backend

De backend zal worden ontwikkeld in Node.js met behulp van het Nest.js framework. Door gebruik te maken van Nest.js kunnen we een schaalbare en onderhoudsvriendelijke applicatie maken. Bovendien wordt Nest.js ook gebruikt binnen Wisemen.

## Chat Technologie

Voor het implementeren van de chatfunctionaliteit zijn er verschillende opties beschikbaar. De backend fungeert als de server die de communicatie tussen gebruikers mogelijk maakt en berichten afhandelt. Hieronder worden twee veelgebruikte opties beschreven:

* **Socket.io:** Socket.IO is een library/framework die low-latency, bidirectionele en op gebeurtenissen gebaseerde communicatie mogelijk maakt tussen een client en een server. Het biedt een eenvoudige manier om real-time chatfunctionaliteit toe te voegen aan een webapplicatie. [4]

“It is a library or framework that abstracts away the underlying transport mechanism and provides a set of tools and utilities for building real-time, bidirectional communication between a client and server.” [5].

Het heeft MIT-licentie dit betekend dat we dit kunnen gebruiken voor commercial use.

* **Websocket (ws):** ws is een eenvoudig te gebruiken, razendsnelle en grondig geteste implementatie van een WebSocket-client en -server. [6]

Voor de keuze te maken heb ik een vergelijking gemaakt van beide opties op volgende pagina. Uit deze analyse is gebleken dat ik Socket.io zal gebruiken vanwege de eenvoudige implementatie. Bovendien biedt Socket.io automatisch fallbacks en reconnection-handling, waardoor de betrouwbaarheid van real-time communicatie wordt verbeterd. Ook beschikt het over een uitgebreide set features voor het beheren van verbindingen en het verzenden van berichten, wat goed aansluit bij de vereisten van ons project.

## Notificaties

In onze backend is het ook essentieel om functionaliteit toe te voegen voor het versturen van notificaties. Dit kan ook worden gerealiseerd met behulp van Socket.IO, waardoor we een geïntegreerde oplossing kunnen bieden voor realtime communicatie en meldingen binnen onze applicatie.

Vergelijking tussen websocket libraries:

|  |  |  |
| --- | --- | --- |
|  | Socket.io | Websocket |
| pros | * **Easier to set up:** Socket.io offers a higher-level abstraction over websockets, simplifying development and handling common tasks like connection management, message broadcasting, and event handling.
* **Built-in features:** Socket.io provides features like automatic reconnection, rooms, and namespaces out of the box, reducing the need for custom code.
* **Larger community:** Due to its popularity, socket.io has a larger community and more resources available for troubleshooting and finding solutions.
 | * **More control:** You have finer control over the websocket connection details and can customize the behavior as needed.
* **Potentially lower overhead:** By handling tasks like connection management and message broadcasting yourself, you might achieve slightly better performance compared to socket.io.
 |
| cons | * **Abstraction layer:** You might have less control over the underlying websocket connection compared to using the ws package directly.
 | * **More complex setup:** Implementing a complete websocket solution using the ws package requires writing more code and handling various potential edge cases compared to the higher-level features of socket.io.
 |

Files in chat verzenden

Voor dit onderdeel heb ik research moeten doen naar wat de best practices zijn om files door te sturen. Hieronder een aantal bevindingen die ik heb gedaan.

Er zijn in grote lijnen 3 verschillende manieren om aan file sharing te doen:

1. Save directly on the server
2. Saving the file’s binary data or base64 string date in a database
3. Using S3 buckets

Uit de vergelijking op volgende pagina die ik gedaan heb blijkt duidelijk dat het gebruik van cloudopslag de beste optie is wanneer je naar grotere toepassingen kijkt. Maar hoe doen we dit nu op een goede manier? Ik heb een aantal bronnen geraadpleegd om te achterhalen wat nu de best practice is. [7] [8] [9]

Dit is de aanbevolen werkwijze:

* 1. Ontvang een signed upload url voor opslaan in de cloud.

De cloudopslagbucket neemt de meeste beveiligingszorgen op zich en is losgekoppeld van de server. De server is alleen verantwoordelijk voor het genereren van presigned signed upload url en het opslaan van metadata over de bestanden. Het voordeel van het niet verzenden naar de server is dat deze de server niet kan uitschakelen als er bijvoorbeeld malware aanwezig is.

* 1. Splits het bestand in chunks aan de client side en upload het in chunks.

Hierdoor kunt u het uploaden onderbreken, hervatten en de mislukte chunks opnieuw proberen.

* 1. Opslaan in de bucket voor raw files en deze koppelen aan SNS.
	2. Processing
	3. Wanneer SNS wordt geactiveerd, kunt u de verwerking uitvoeren
	4. Upload naar final bucket
	5. Zodra we bevestiging hebben ontvangen dat het is geüpload, sturen we het bestand (link) vanuit de server naar de client.

Vergelijking van verschillende manieren om file sharing mogelijk te maken in de backend:

|  |  |  |
| --- | --- | --- |
|  | Pros | Cons |
| Save directly on the server | **Simplicity:** Relatively simple to implement, especially for smaller applications. **Direct access:** Straightforward to access files for processing or serving. | **Scalability limitations:** Can become challenging to manage and scale storage as the number of files or their sizes grow. **Performance considerations:** May impact server performance for large files or high traffic applications. **Security concerns:** Requires careful security measures to prevent unauthorized access or file corruption. |
| Base64 in db | **Compact storage:** Base64 encoding can reduce file size by roughly 33%, potentially saving database storage space for smaller files. | **Performance overhead:** Database operations, especially complex queries involving large files, can be slow and impact application performance. **Security concerns:** Requires ensuring data protection within the database, including encryption and access control mechanisms. -**Database suitability:** Not all databases are designed for efficient binary data storage, and database management can become complex with a large number of files. |
| S3 Bucket | **Scalability:** Highly scalable solution, easily accommodating large amounts of data and high-volume uploads. **Reliability:** Cloud storage services offer robust redundancy mechanisms and fault tolerance, ensuring data availability and protection against hardware failures. **Performance:** Optimized for storing and retrieving large objects, offering good performance for file downloads and access. **Cost-effectiveness:** Can be cost-effective for most applications, with pay-as-you-go models aligning with usage. **Security features:** Cloud storage providers typically offer robust security features like encryption at rest and in transit, access control, and auditing capabilities. | **Vendor lock-in:** Dependence on a specific cloud storage service provider might lead to potential vendor lock-in and associated implications. **Network latency:** Geographical factors or unstable internet connections can introduce slight latency in file access compared to directly storing on the server. **Cost considerations:** While often cost-effective, heavy usage or specific features might lead to higher costs with some cloud providers. |

# Authenticatie

Authenticatie zal worden geïmplementeerd in de Node.js backend met behulp van OAuth2, een protocol voor autorisatie dat veel wordt gebruikt voor het verlenen van beperkte toegang tot applicaties door externe services. Met OAuth2 kunnen gebruikers veilig en efficiënt worden geauthentiseerd en geautoriseerd voor het gebruik van onze applicatie. Aangezien ik start van het template-project van Wisemen is dit al reeds geïmplementeerd.

Voor authenticatie/autorisatie voor socket.io:

“There is no concept of user in Socket.IO.

It is the duty of your application to link a given Socket.IO connection to a user account.

For Node.js applications, you can for example:

* reuse the user context provided by [Passport](https://www.passportjs.org/) (check [this tutorial](https://socket.io/how-to/use-with-express-session))
* or use the [auth](https://socket.io/docs/v4/client-options/#auth) option on the client side to send the user credentials and validate them in a [middleware](https://socket.io/docs/v4/middlewares/)” [10]

Ik zal dus de JWT-token meesturen bij het openen van een verbinding en deze vervolgens verifiëren op de server via de Socket.IO-middleware.

# Hosting

Het project gaat volledig lokaal draaien. De enige vereiste die ik heb gekregen om het mogelijk te maken om te hosten is het in een docker container kunnen draaien.

# Database

Voor deze toepassing maak ik gebruik van twee databases.

**Redis** is een open-source, in-memory datastructuren-store die kan worden gebruikt als database, cache, message broker en streaming-engine. Ik zal Redis gebruiken als adapter om verschillende Socket.IO-servers met elkaar te laten communiceren via het pub/sub-systeem voor load balancing omdat nodejs single threaded is alsook voor bepaalde zaken op te slaan in de cache. [11]

Ik zal **PostgreSQL** gebruiken als relationele database om gebruikers, chatrooms en messages op te slaan en relaties te leggen met andere gegevens. Ik heb het databaseschema zorgvuldig ontworpen, met de nadruk op flexibiliteit voor toekomstige uitbreidingen. Dit zorgt ervoor dat het gemakkelijk aanpasbaar is als we extra gegevens willen opslaan of nieuwe functionaliteiten willen toevoegen.



Opmerking: 'seenBy' is een array van UUID's van de ‘user’ tabel. Er kon geen foreign key worden gemaakt vanwege het ontbreken van ondersteuning voor dit type relatie in PostgreSQL.

# Architectuur

Hieronder heb ik een schematische voorstelling gemaakt van hoe de applicatie er gaat uitzien.





# Uitbreiding: dynamische chat

Als uitbreiding op het Minimum Viable Product (MVP) wil ik de chatfunctionaliteit volledig dynamisch maken, zodat deze gemakkelijk te implementeren is in andere systemen. Bovendien wil ik functionaliteiten toevoegen om verschillende soorten chats te kunnen aanmaken, elk met hun eigen specifieke use case. Aangezien dit een uitbreiding betreft, heb ik het nog niet diepgaand uitgewerkt, maar ik heb wel voorzien dat het eenvoudig is om mijn systeem hiermee uit te breiden. Hieronder volgen enkele ideeën:

1. **Chatbot voor FAQ:** Gebruikers kunnen chatten met een chatbot (via de ChatGPT API) die is getraind om FAQ te beantwoorden. Beheerders kunnen de chatbot trainen door specifieke vraagstellingen in te voeren via een adminpaneel.
2. **Interne helpdesk (ticketingsysteem):** Gebruikers kunnen een chat starten om hulp te vragen bij problemen en kunnen indien nodig een ticket aanmaken. Het systeem initieert vervolgens een chat met een medewerker om verdere vragen te stellen. Medewerkers hebben andere rollen en rechten dan gebruikers.
3. **Feedback en enquêteverzameling:** Het systeem wil feedback verzamelen van gebruikers. Het maakt een chat aan met een chatbot die een vragenlijst op een gebruiksvriendelijke manier afhandelt.
4. **Chat tussen gebruikers:** Bijvoorbeeld, EEVEE-chauffeurs kunnen andere elektrische chauffeurs benaderen om van plaats te wisselen voor het opladen.

Om deze chats te beheren hebben we dus een adminpaneel nodig, waar we verschillende parameters kunnen instellen/aanpassen.

Mogelijke parameters:

* **Chat type**: Hiermee kan de admin kiezen welk type chat wordt gemaakt, zoals chatbot, helpdesk, enqueteverzameling of gebruikerschat.
* **Chatbot training**: Als het gaat om een chatbot, kan de admin hier vragen invoeren die moeten worden gebruikt om de chatbot te trainen.
* **Users**: Hiermee kan de admin de gebruikers selecteren waarmee de chat moet worden gestart, zoals specifieke gebruikers UUID's of groepen gebruikers.
* **Roles**: Als je verschillende gebruikersrollen hebt (bijv. beheerder, medewerker, klant), kan de admin hier instellen welke rollen toegang hebben tot de chat en welke acties ze kunnen uitvoeren.
* **Chat-beheer**: In sommige gevallen wil je misschien de mogelijkheid hebben om chats uit te schakelen of te archiveren. De admin kan hier opties hebben om chats aan te maken, beheren en te verwijderen.
* **Berichtsjablonen**: Voor gestandaardiseerde berichten, zoals initiële begroetingen of veelvoorkomende antwoorden in een chatbot, kan de admin hier berichtsjablonen definiëren.

* Taal- en regionalisatie-instellingen
* Nummerplaat scanner
* Dynamisch chat’s aanmaken
* Admin panel voor chat managing
* Verschillende soorten chat’s
* Dashboard (Analyse en rapportage)
* Chat joinen tussen 2 personen op basis van id(nummerplaat)
* Intern notificaties sturen

Won’t Have

Could Have

Should Have

Must Have

# Risico’s

Het grootste risico dat ik zie, is dat ik te maken heb met veel nieuwe technologieën en concepten die ik nog moet leren. Hierdoor kan het moeilijk zijn om de benodigde tijd voor ontwikkeling in te schatten. Als dit risico zich voordoet, kan dit leiden tot vertragingen in de ontwikkeling en mogelijk het niet kunnen voldoen aan mijn planning. Echter, ik heb veel vertrouwen in het project en maak voornamelijk gebruik van interne systemen, wat de kans op problemen op dat gebied verkleint.

Als ik merk dat ik toch niet genoeg tijd heb ingeschat, zal ik mijn planning aanpassen op basis van mijn MOSCOW-analyse om ervoor te zorgen dat mijn Must Have's worden voltooid. Mijn MvP is zeker haalbaar tijdens mijn stageperiode.

# Planning



# Referenties

[1] Wisemen, “Wisemen® | leading digital agency in Belgium,” *Wisemen*, Mar. 08, 2024. <https://wisemen.digital/>

[2] “EEVEE Mobility | Optimizing charge cost efficiency for EVs.” <https://www.eeveemobility.com/>

[3] Naptha, “GitHub - naptha/tesseract.js: Pure Javascript OCR for more than 100 Languages 📖🎉🖥,” *GitHub*. <https://github.com/naptha/tesseract.js>

[4] “Introduction | Socket.IO,” Feb. 20, 2024. <https://socket.io/docs/v4/>

[5] “What is Socket.IO?,” *PubNub*. <https://www.pubnub.com/guides/socket-io/>

[6] Websockets, “GitHub - websockets/ws: Simple to use, blazing fast and thoroughly tested WebSocket client and server for Node.js,” *GitHub*. <https://github.com/websockets/ws>

[7] “How can you handle file uploads in a back-end application?,” *www.linkedin.com*, Nov. 04, 2023. <https://www.linkedin.com/advice/0/how-can-you-handle-file-uploads-back-end-application-npbte#:~:text=The%20file%20upload%20process%20involves,to%20multipart%2Fform%2Ddata>

[8] “How to upload large files (1GB+) through a RestAPI,” *Reddit*, 2023. <https://www.reddit.com/r/softwarearchitecture/comments/10v7mo2/how_to_upload_large_files_1gb_through_a_restapi/>

[9] “How do I handle large file upload on node server so it doesn’t block the event loop?,” *Reddit*, 2022. <https://www.reddit.com/r/node/comments/w25ibg/how_do_i_handle_large_file_upload_on_node_server/>

[10] “FAQ | Socket.IO,” Feb. 20, 2024. <https://socket.io/docs/v4/faq/#how-to-identify-a-given-user>

[11] “Redis,” *Redis*. https://redis.io/