# TECHNISCH ONTWERP

API Documentatie

Autogarage "De Maker"

Albor Metaliaj

23 December 2025

HBO ICT/Backend
Software Development

# Inhoudsopgave

# 1 Inleiding

## 1.1 Probleembeschrijving

Autogarage ”De Maker”kampt met inefficiëntie in hun dagelijkse werkprocessen door het gebruik van papieren administratie. Uit onderzoek van automotive software providers blijkt dat garages met handmatige systemen gemiddeld 10-15 minuten per reparatieorder kwijt zijn aan administratieve taken—werk dat met digitale tools slechts 3-5 minuten duurt. Voor een gemiddelde garage met 300 reparaties per maand betekent dit een verlies van meer dan 50 uur per maand aan productieve werktijd.

Bij ”De Maker”worden klantgegevens, autopapieren, keuringsresultaten en reparatie-afspraken momenteel handmatig bijgehouden in fysieke mappen en spreadsheets.

Administratief medewerkers besteden dagelijks meerdere uren aan het zoeken naar klant-dossiers en autodocumentatie. Wanneer een klant belt over de status van een reparatie, moet eerst het juiste dossier opgezocht worden voordat er antwoord gegeven kan worden. Monteurs onderbreken regelmatig het kantoor met vragen als ”wat moet ik nu doen? waar zijn de papieren van kenteken XX-XX-XX?”, wat de workflow verstoort.

Belangrijke documenten zoals kentekenpapieren en verzekeringsgegevens worden in fysieke mappen bewaard.

Zonder centraal systeem weten medewerkers niet in welke fase een reparatie zich bevindt. Is de keuring al uitgevoerd? Heeft de klant akkoord gegeven? Welke onderdelen zijn besteld? Deze onduidelijkheid leidt tot miscommunicatie tussen administratie en monteurs, en zorgt voor ontevreden klanten die niet geïnformeerd worden over de voortgang.

Monteurs schrijven keuringsresultaten handmatig op papieren formulieren. Vervolgens moet een administratief medewerker deze informatie opnieuw invoeren in een digitaal systeem om een factuur te kunnen genereren. Deze dubbele invoer kost niet alleen extra tijd, maar introduceert ook extra kans op typefouten en miscommunicatie.

Deze web-API lost de genoemde problemen op door:

1. Centrale database voor alle klant- en autogegevens met snelle zoekfunctie
2. Digitale documentopslag van autopapieren in PDF-formaat met beveiligde toegang
3. Gestructureerde keuringsregistratie waarbij monteurs tekortkomingen direct kunnen invoeren
4. Traceerbare reparatie-administratie met klantakkoord en statusupdates
5. Rolgebaseerde toegang (Administratief Medewerker vs. Monteur) voor informatiebeveiliging

## 1.2 Doelgroep

Dit systeem is ontwikkeld voor twee primaire gebruikersgroepen binnen ”De Maker”:

Administratief Medewerkers (rol: ADMIN)

- Beheren van klantgegevens (naam, contactinformatie, gekoppelde auto’s)
- Registreren van nieuwe auto’s in het systeem
- Uploaden en downloaden van autopapieren (kentekenbewijs, APK-documenten)
- Registreren van klantakkoord voor reparaties
- Inzicht in alle lopende keuringen en reparaties
- Genereren van overzichten voor planning en financiële administratie

Monteurs (rol: MONTEUR)

- Inplannen van keuringen met klanten
- Registreren van gevonden tekortkomingen tijdens inspectie
- Vastleggen van geschatte kosten en veiligheidsrisico’s per tekortkoming
- Aanmaken en bijwerken van reparatie-opdrachten
- Bijhouden van uitgevoerde handelingen en gebruikte onderdelen
- Raadplegen van autopapieren tijdens werkzaamheden

Beide gebruikersgroepen hebben toegang tot dezelfde database, maar zien alleen de functionaliteiten die relevant zijn voor hun rol. Dit voorkomt dat monteurs bijvoorbeeld financiële rapportages kunnen inzien, terwijl administratief medewerkers wel toegang hebben tot alle systeem-onderdelen.

## 1.3 Systeemoverzicht

De Autogarage API ”De Makerïs een RESTful web-API gebouwd met Java Spring Boot. Het systeem biedt de volgende hoofdfunctionaliteiten:

Authenticatie en Autorisatie - Inloggen met gebruikersnaam en wachtwoord - JWT (JSON Web Token) authenticatie voor beveiligde communicatie - Rolgebaseerde toegangscontrole (ADMIN en MONTEUR) - Automatische token-verloop na 24 uur voor extra beveiliging

Klant- en Autobeheer

- CRUD operaties (Create, Read, Update, Delete) voor klantgegevens
- Registratie van auto’s met kenteken, merk, model en bouwjaar
- Koppeling van meerdere auto’s aan één klant
- Opvragen van complete klantgeschiedenis inclusief alle gekoppelde auto’s

Documentbeheer

- Upload van PDF autopapieren (max 10MB per document)
- Beveiligde opslag in database met metagegevens (bestandsnaam, uploaddatum)
- Download-functionaliteit voor administratief personeel en monteurs
- Automatische koppeling van documenten aan specifieke auto's

Keuringsproces

- Aanmaken van keuringen met geplande datum (intake-inspectie voor reparatie)
- Registratie van tekortkomingen met beschrijving, kosten en veiligheidsrisico
- Statusupdates (gepland/uitgevoerd/afgerond)
- Koppeling tussen gevonden tekortkomingen en uit te voeren reparaties

Reparatiebeheer

- Inplannen van reparaties op basis van keuringsresultaten
- Registratie van klantakkoord (wel/niet akkoord met voorgestelde werkzaamheden)
- Bijhouden van uitgevoerde datum en totale kosten
- Statusregistratie (gepland/in uitvoering/afgerond)

Technische Architectuur De API gebruikt een gelaagde architectuur:

- Controller-laag: Ontvangt HTTP-requests en stuurt responses terug
- Service-laag: Bevat business logica en valideert invoer
- Repository-laag: Communiceert met de database via JPA/Hibernate
- Database-laag: PostgreSQL voor data-persistentie

Alle endpoints zijn beveiligd via Spring Security, waarbij elke request gevalideerd wordt op basis van het meegestuurde JWT token en de bijbehorende gebruikersrol.

# 2 User Stories

## 2.1 User Story 01: Autopapieren uploaden

**Als** administratief medewerker
**Wil ik** autopapieren kunnen uploaden
**Zodat** de garage direct beschikt over alle benodigde informatie

Acceptatiecriteria:

- Alleen PDF-bestanden worden geaccepteerd
- Maximale bestandsgrootte is 10MB
- Systeem controleert of de auto bestaat in de database
- Bij registratie wordt de klant gekoppeld als contactpersoon voor deze auto
- Eén auto kan meerdere PDF's hebben (kentekenbewijs, APK, verzekering)
- Alleen administratief medewerker en monteur kunnen PDF's downloaden
- Bij upload wordt gelogd: wie, wanneer, welk bestand, bij welke auto
- Succesvolle upload toont bevestiging met bestandsnaam en uploaddatum

## 2.2 User Story 02a: Keuring inplannen

**Als** administratief medewerker
**Wil ik** een reparatie-keuring kunnen inplannen voor een auto van een klant
**Zodat** de monteur weet welke auto's vandaag gekeurd moeten worden

Acceptatiecriteria:

- Alleen administratief medewerker kan keuringen aanmaken
- Keuring is gekoppeld aan een bestaande auto (en dus klant)
- Geplande datum wordt vastgelegd
- Status wordt initieel gezet op ïngepland"
- Monteur wordt automatisch toegewezen bij openen van keuring
- Administratie kan lijst van ingeplande keuringen zien per dag

## 2.3 User Story 02b: Tekortkomingen

**Als** monteur
**Wil ik** tekortkomingen kunnen toevoegen aan een keuring
**Zodat** de administratie de klant kan informeren over benodigde reparaties

Acceptatiecriteria:

- Alleen monteur kan tekortkomingen toevoegen
- Monteur kan meerdere tekortkomingen per keuring registreren
- Bij elke tekortkoming wordt vastgelegd:
  - Beschrijving (verplicht, vrije tekst)
  - Geschatte kosten voor reparatie
  - Veiligheidsrisico (boolean: ja/nee)
- Monteur kan status van keuring wijzigen naar äfgerond”
- Keuringen zonder tekortkomingen zijn mogelijk (auto keurt goed)
- Systeem toont overzicht van alle tekortkomingen per keuring

## 2.4 User Story 03: Authenticatie - Inloggen en registreren

**Als** monteur of administratie
**Wil ik** een account kunnen aanmaken en inloggen
**Zodat** ik toegang heb tot het systeem met de juiste rechten voor mijn rol

Acceptatiecriteria - Registreren:

- Administratief medewerker kan nieuwe gebruikers registreren
- Bij registratie: gebruikersnaam, email, wachtwoord, rol (ADMIN/MONTEUR)
- Wachtwoord wordt versleuteld opgeslagen
- Email en gebruikersnaam moeten uniek zijn
- Wachtwoord minimaal 8 karakters

Acceptatiecriteria - Inloggen:

- Gebruiker logt in met gebruikersnaam/email en wachtwoord
- Bij succesvolle login ontvangt gebruiker een JWT token (geldig 24 uur)
- Token bevat gebruikersnaam en rol
- Bij ongeldige credentials krijgt gebruiker duidelijke foutmelding
- Bij verlopen token moet gebruiker opnieuw inloggen

Acceptatiecriteria - Autorisatie:

- Alleen monteur kan tekortkomingen toevoegen
- Alleen admin kan gebruikers registreren en keuringen inplannen
- PDF’s kunnen door beide rollen worden gedownload

## 2.5 User Story 04: Reparatie inplannen na klant-akkoord

**Als** administratief medewerker
**Wil ik** kunnen vastleggen welke tekortkomingen de klant wil laten repareren
**Zodat** alleen goedgekeurde reparaties worden uitgevoerd

Acceptatiecriteria:

- Administratie kan een keuring met tekortkomingen bekijken
- Bij elke tekortkoming kan worden aangegeven: ”Klant akkoord”(ja/nee)
- Als klant akkoord gaat, wordt een reparatie aangemaakt
- Reparatie is gekoppeld aan de geselecteerde tekortkomingen
- Bij reparatie wordt vastgelegd: geplande datum, status (gepland/bezig/klaar)
- Als klant niet akkoord gaat, wordt dit genoteerd maar geen reparatie aangemaakt
- Administratie kan lijst van geplande reparaties zien

# 3 Functionele en Niet-Functionele Eisen

## 3.1 Functionele Eisen

**FE01:** Als ADMIN kan ik nieuwe klanten registreren met voornaam, achternaam, email en telefoonnummer.

**FE02:** Als ADMIN kan ik auto's toevoegen aan het systeem met kenteken, merk, model en bouwjaar.

**FE03:** Als ADMIN kan ik een auto koppelen aan een bestaande klant.

**FE04:** Als ADMIN kan ik PDF autopapieren uploaden bij een auto met een maximale bestandsgrootte van 10MB.

**FE05:** Als ADMIN en MONTEUR kan ik geüploade autopapieren downloaden in PDF-formaat.

**FE06:** Als MONTEUR kan ik een keuring aanmaken voor een specifieke auto met een geplande datum.

**FE07:** Als MONTEUR kan ik tekortkomingen toevoegen aan een keuring met beschrijving, geschatte kosten en een indicatie voor veiligheidsrisico.

**FE08:** Als MONTEUR kan ik een reparatie inplannen die gekoppeld is aan één of meerdere gevonden tekortkomingen.

**FE09:** Als MONTEUR kan ik de status van een reparatie bijwerken naar gepland, in uitvoering of afgerond.

**FE10:** Als MONTEUR kan ik de uitgevoerde datum en totale kosten registreren bij een afgeronde reparatie.

**FE11:** Als ADMIN kan ik het klantakkoord registreren voor een reparatie (wel of niet akkoord met de voorgestelde werkzaamheden).

**FE12:** Als gebruiker kan ik inloggen met mijn gebruikersnaam en wachtwoord en ontvang ik een JWT token voor authenticatie.

**FE13:** Als systeem wijs ik automatisch een rol toe (ADMIN of MONTEUR) aan elke gebruiker op basis van hun account.

**FE14:** Als systeem blokkeer ik toegang tot endpoints op basis van de gebruikersrol zodat alleen geautoriseerde gebruikers bepaalde acties kunnen uitvoeren.

**FE15:** Als systeem valideer ik alle invoer op verplichte velden, correcte formats (zoals email en kenteken) en datatypen voordat data wordt opgeslagen.

**FE16:** Als systeem geef ik duidelijke JSON foutmeldingen terug bij ongeldige requests, inclusief de juiste HTTP-statuscodes (400, 401, 404, 500).

**FE17:** Als ADMIN en MONTEUR kan ik lijsten opvragen van alle klanten, auto's, keuringen en reparaties via GET endpoints.

## 3.2 Niet-Functionele Eisen

**NFE01:** Wachtwoorden worden versleuteld opgeslagen in de database met BCrypt hashing algoritme (minimaal cost factor 10).

**NFE02:** JWT tokens hebben een geldigheidsduur van 24 uur waarna de gebruiker opnieuw moet inloggen.

**NFE03:** PDF file uploads zijn gelimiteerd tot maximaal 10MB per bestand om database-opslag beheersbaar te houden.

**NFE04:** Alleen bestanden met content-type äpplication/pdf"worden geaccepteerd voor autopapieren uploads; andere bestandstypen worden geweigerd.

**NFE05:** De API volgt RESTful conventies met correcte HTTP methods (GET voor opvragen, POST voor aanmaken, PUT voor wijzigen, DELETE voor verwijderen) en statuscodescodes (200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error).

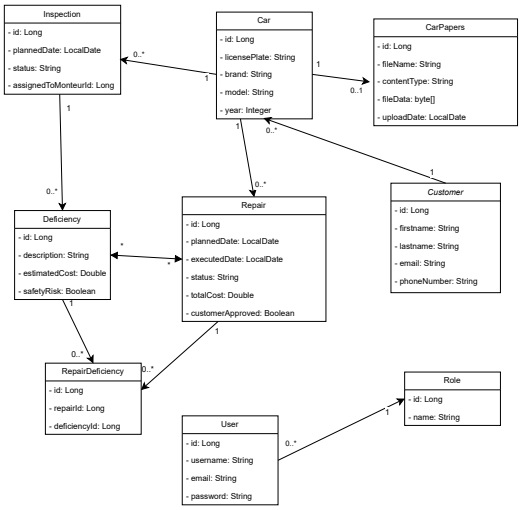
**NFE06:** De applicatie gebruikt PostgreSQL als database maar is via JPA/Hibernate database-agnostisch ontworpen, waardoor migratie naar andere relationele databases mogelijk is via configuratiewijziging.

**NFE07:** De code volgt SOLID principes en Clean Code richtlijnen: betekenisvolle namen, Single Responsibility Principle, Dependency Injection via constructors, en scheiding tussen Controller-Service-Repository lagen.

**NFE08:** De applicatie draait op Java 17 (LTS versie) met Spring Boot 3.x framework en Maven als dependency manager voor reproduceerbare builds.

# 4 Klassendiagram

## 4.1 Domeinmodel




Figuur 1: Demaker Klassendiagram

## 4.2 Relaties en Kardinaliteiten

Het domeinmodel van ”De Maker”bestaat uit 8 entiteiten met de volgende relaties:

Customer ↔ Car (One-to-Many)
Eén klant kan meerdere auto’s bezitten, maar elke auto hoort bij precies één klant. De relatie wordt vastgelegd via een foreign key `customer_id` in de `cars` tabel.

Car ↔ CarPapers (One-to-One)
Elke auto kan één set autopapieren hebben (PDF met kentekenbewijs, verzekering). Deze relatie is optioneel (0..1): niet elke auto heeft direct papieren geüpload. De `cars` tabel bevat een foreign key `car_papers_id`.

Car ↔ Inspection (One-to-Many)
Eén auto kan meerdere keuringen ondergaan (verschillende data), maar elke keuring hoort bij precies één auto. Foreign key `car_id` in de `inspections` tabel.

Inspection ↔ Deficiency (One-to-Many)
Eén keuring kan meerdere tekortkomingen bevatten, maar elke tekortkoming hoort bij precies één keuring. Foreign key `inspection_id` in de `deficiencies` tabel.

Deficiency ↔ Repair (Many-to-Many)
Dit is de meest complexe relatie: meerdere tekortkomingen kunnen samen in één reparatie worden opgelost, en één tekortkoming kan onderdeel zijn van meerdere reparaties

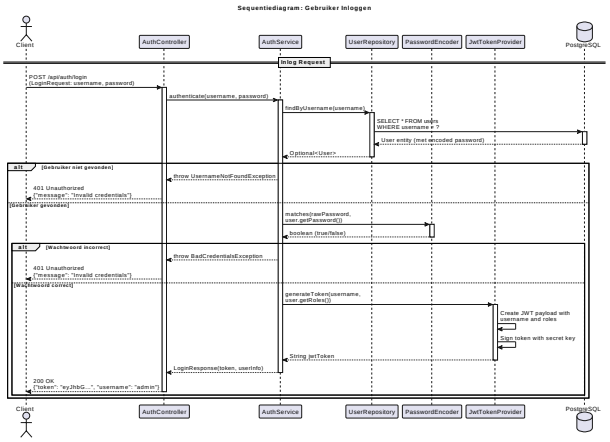
(bijvoorbeeld gespreid over meerdere werkzaamheden). Deze many-to-many relatie wordt geïmplementeerd via de koppeltabel `RepairDeficiency` met foreign keys naar beide entiteiten.

User ↔ Role (Many-to-Many)
Gebruikers kunnen meerdere rollen hebben (hoewel in deze casus meestal één: ADMIN of MONTEUR). Dit wordt geïmplementeerd via een standaard `user_roles` koppeltabel voor toekomstige uitbreiding.

# 5 Sequentiediagrammen

Dit diagram toont het complete authenticatieproces waarbij een gebruiker inlogt met gebruikersnaam en wachtwoord. Het systeem valideert de credentials, genereert een JWT token en geeft deze terug aan de client.

## 5.1 Sequentiediagram 1: Gebruiker inloggen




Figuur 2: Gebruiker inloggen

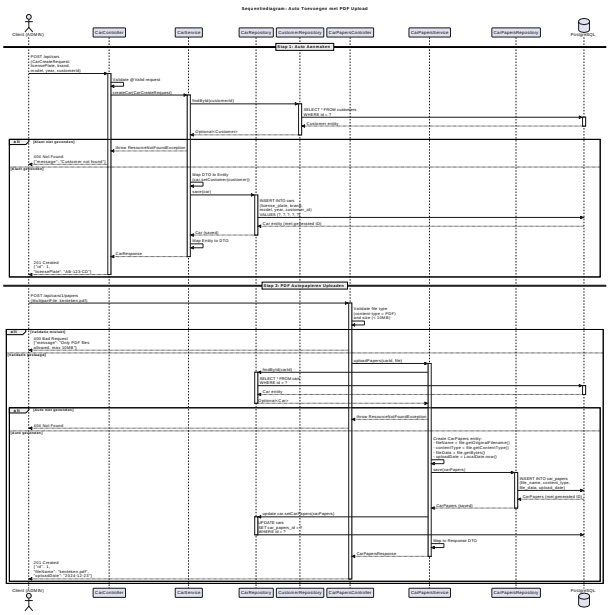
Uitleg van het diagram:

Dit sequentiediagram toont de drie architecturale lagen (Controller, Service, Repository) en hun interactie tijdens het inlogproces:

1. De client stuurt een POST request naar `/api/auth/login` met gebruikersnaam en wachtwoord in de request body.
2. `AuthController` ontvangt de request en delegeert de authenticatie naar de `AuthService`.
3. `AuthService` zoekt de gebruiker op via `UserRepository.findByUsername()`.
4. De repository voert een SQL SELECT query uit op de database en geeft de `User` entity terug (indien gevonden).
5. `AuthService` valideert het wachtwoord met `PasswordEncoder.matches()`, die het ingevoerde wachtwoord vergelijkt met de BCrypt hash uit de database.
6. Bij een correcte match genereert `JwtTokenProvider` een JWT token met de gebruikersnaam en rollen.
7. De token wordt teruggegeven via de service- en controller-laag naar de client met HTTP status 200 OK.
8. Bij ongeldige credentials (gebruiker niet gevonden of verkeerd wachtwoord) wordt een `BadCredentialsException` gegooid, wat resulteert in een 401 Unauthorized response.

De methodenamen in dit diagram komen exact overeen met de implementatie in de broncode.

## 5.2 Sequentiediagram 2: Auto toevoegen met PDF



Figuur 3: Auto toevoegen met PDF

Uitleg van het diagram:

Dit diagram toont twee opeenvolgende processen die samen de complete workflow van auto-registratie met documentatie illustreren:

Stap 1: Auto aanmaken

1. Client (ADMIN) stuurt een POST request naar `/api/cars` met autogegevens en een `customerId`.
2. `CarController` valideert de input met `@Valid` annotatie en roept `CarService.createCar()` aan.
3. `CarService` controleert eerst of de klant bestaat via `CustomerRepository.findById()`.
4. Bij een niet-bestaande klant wordt een `ResourceNotFoundException` gegooid (HTTP 404).
5. Bij een bestaande klant wordt de `Car` entity aangemaakt en gekoppeld aan de `Customer`.
6. `CarRepository.save()` voert een INSERT query uit, waarbij de database een auto-increment ID genereert.
7. De opgeslagen `Car` wordt geconverteerd naar een `CarResponse` DTO en teruggegeven met HTTP 201 Created.

Stap 2: PDF autopapieren uploaden

1. Client upload een PDF via POST naar `/api/cars/{id}/papers` als `multipart/form-data`.
2. `CarPapersController` valideert eerst het bestandstype (moet PDF zijn) en de grootte (max 10MB).
3. Bij ongeldige validatie wordt direct HTTP 400 Bad Request teruggegeven.
4. `CarPapersService` controleert of de auto bestaat via `CarRepository.findById()`.
5. De PDF wordt omgezet naar een byte array en opgeslagen in een `CarPapers` entity met metadata (bestandsnaam, content-type, uploaddatum).
6. `CarPapersRepository.save()` slaat de PDF op in de `car_papers` tabel als BLOB.
7. De `Car` entity wordt bijgewerkt met een foreign key naar de zojuist opgeslagen `CarPapers`.
8. De response bevat de PDF metadata (ID, bestandsnaam, uploaddatum) met HTTP 201 Created.

Dit diagram demonstreert het gebruik van DTOs (Data Transfer Objects) om database entities af te schermen van de API laag, en toont exception handling voor niet-bestaande resources.