

# DOCUMENTAȚIE VEHICHECK

Aplicație pentru Gestionarea și Monitorizarea Vehiculelor

**STUDENȚI**

Aruștei Andrei  
Petre Flaviu

23 iunie 2025

Universitatea Transilvania din Brașov  
Facultatea de Matematică și Informatică

# Cuprins

<b>1</b>	<b>Introducere și Obiective</b>	<b>3</b>
1.1	Ce își propune . . . . .	3
1.2	Ce probleme rezolvă . . . . .	3
<b>2</b>	<b>Tehnologiile Folosite</b>	<b>3</b>
2.1	Baza de Date . . . . .	3
<b>3</b>	<b>Arhitectura Bazei de Date</b>	<b>4</b>
3.1	Prezentarea Tabelelor și Relațiilor . . . . .	4
3.1.1	Users . . . . .	4
3.1.2	Cars . . . . .	5
3.1.3	CarModels . . . . .	5
3.1.4	CarManufacturers . . . . .	5
3.1.5	Components . . . . .	6
3.1.6	ComponentManufacturers . . . . .	6
3.1.7	CarModelsComponents . . . . .	6
3.1.8	Fixes . . . . .	6
3.1.9	ComponentsFixes . . . . .	7
<b>4</b>	<b>Prezentarea API-ului</b>	<b>7</b>
4.1	AuthController (api/auth) . . . . .	8
4.2	UsersController (api/users) . . . . .	8
4.3	CarsController (api/cars) . . . . .	8
4.4	CarModelsController (api/carmodels) . . . . .	8
4.5	CarManufacturersController (api/carmanufacturers) . . . . .	9
4.6	ComponentManufacturersController (api/componentmanufacturers) . . . . .	9
4.7	ComponentsController (api/components) . . . . .	9
4.8	FixesController (api/fixes) . . . . .	9
<b>5</b>	<b>Utilizarea Aplicației</b>	<b>10</b>
5.1	Tipuri de Utilizatori . . . . .	10
5.1.1	Administrator . . . . .	10
5.1.2	Utilizator Standard . . . . .	10
5.2	Procesul de Autentificare . . . . .	10
5.2.1	Înregistrare . . . . .	10
5.2.2	Autentificare . . . . .	10
5.2.3	Securitate . . . . .	10
<b>6</b>	<b>Concluzii și Contribuții</b>	<b>11</b>
6.1	Împărțirea Task-urilor . . . . .	11
6.1.1	Amândoi (Aruștei Andrei & Petre Flaviu) . . . . .	11
6.1.2	Aruștei Andrei . . . . .	11
6.1.3	Petre Flaviu . . . . .	11
6.2	Ce am Învățat . . . . .	11
6.3	Provocări Întâmpinate . . . . .	11

6.4 Îmbunătățiri Viitoare . . . . .	11
<b>7 Referințe</b>	<b>12</b>

# 1 Introducere și Obiective

## 1.1 Ce își propune

Vehicheck este o aplicație concepută pentru gestionarea și monitorizarea vehiculelor, oferind o platformă pentru urmărirea informațiilor despre autovehicule, întreținerea lor și diverse aspecte legate de operarea acestora. Aplicația își propune să digitalizeze și să eficientizeze procesele de administrare a flotelor de vehicule, oferind un sistem centralizat pentru stocarea și accesarea informațiilor relevante.

## 1.2 Ce probleme rezolvă

Aplicația Vehicheck adresează următoarele probleme principale:

- **Gestionarea ineficientă a informațiilor despre vehicule:** Aplicația centralizează toate datele într-un singur sistem, eliminând necesitatea utilizării documentelor fizice sau a multiplelor sisteme disparate.
- **Dificultatea urmăririi întreținerii:** Vehicheck permite programarea și urmărirea lucrărilor de întreținere, facilitând menținerea vehiculelor în stare optimă de funcționare.
- **Lipsa vizibilității asupra stării vehiculelor:** Aplicația oferă rapoarte și informații în timp real despre starea fiecărui vehicul din flotă.
- **Monitorizarea costurilor:** Aplicația oferă posibilitatea de a urmări și analiza costurile asociate fiecărui vehicul.

# 2 Tehnologiile Folosite

Dezvoltarea aplicației Vehicheck s-a bazat pe următoarele tehnologii moderne:

- **Framework:** ASP.NET Core 7.0
- **Limbaj de programare:** C#
- **Arhitectură:** Layered
- **ORM:** Entity Framework Core pentru accesarea bazei de date
- **Documentarea API-ului:** Swagger/OpenAPI
- **Autentificare/Autorizare:** JWT (JSON Web Tokens)
- **Dependency Injection:** Încorporat în ASP.NET Core

## 2.1 Baza de Date

- **Sistem de Management al Bazei de Date:** Microsoft SQL Server
- **Migrări:** Entity Framework Core Migrations

### 3 Arhitectura Bazei de Date

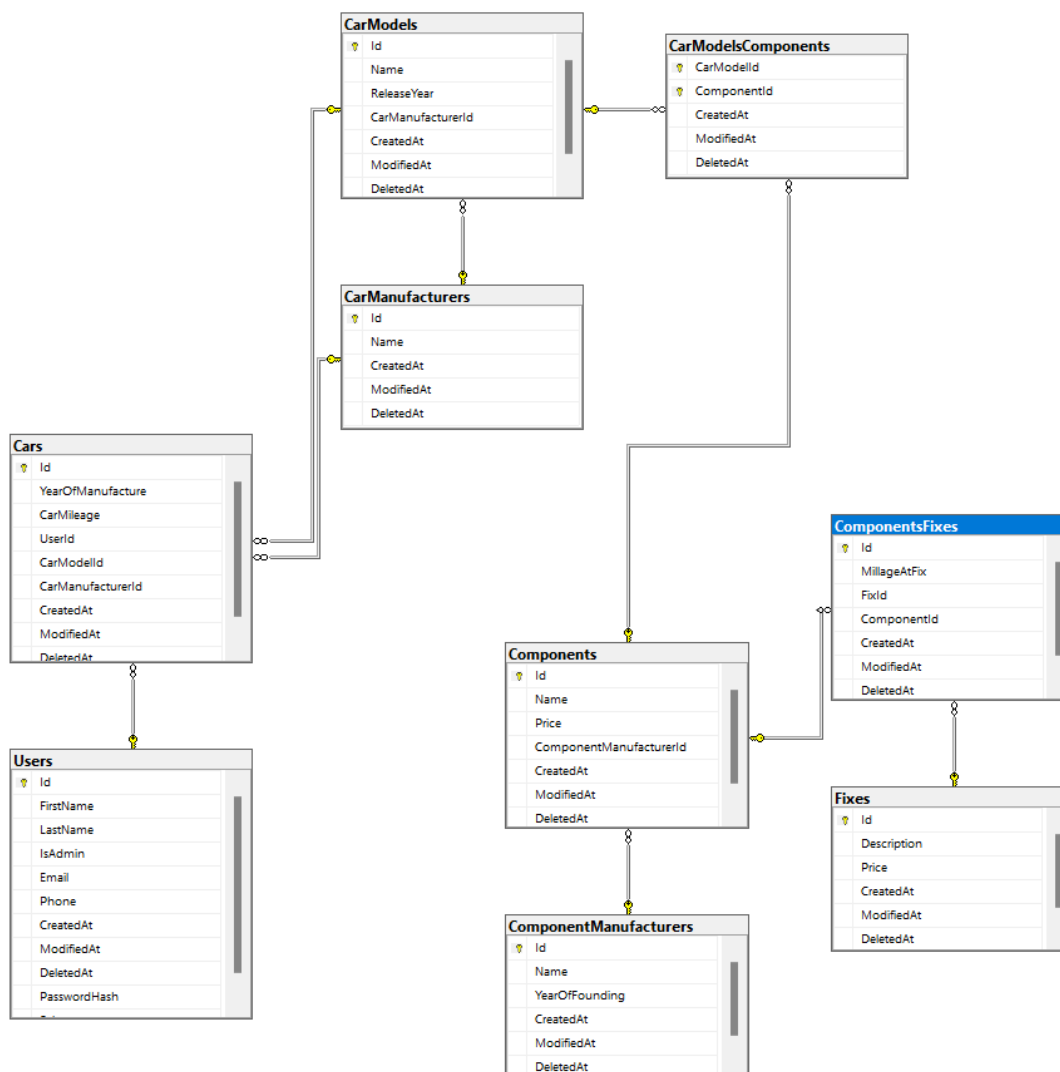


Figura 1: Diagrama entitate-relație a bazei de date Vehicheck

#### 3.1 Prezentarea Tabelelor și Relațiilor

##### 3.1.1 Users

- Conține informații despre utilizatorii sistemului
- **Cheie primară:** Id
- **Atribute principale:** FirstName, LastName, IsAdmin, Email, Phone, CreatedAt, ModifiedAt, DeletedAt, PasswordHash
- **Relații:** Un utilizator poate avea mai multe mașini (One-to-Many cu tabelul Cars)

### 3.1.2 Cars

- Stochează datele despre vehiculele utilizatorilor
- **Cheie primară:** Id
- **Chei străine:** UserId (referință la Users), CarModelId (referință la CarModels), CarManufacturerId (referință la CarManufacturers)
- **Atribute principale:** YearOfManufacture, CarMileage, CreatedAt, ModifiedAt, DeletedAt
- **Relații:**
  - O mașină aparține unui singur utilizator (Many-to-One cu Users)
  - O mașină este de un anumit model (Many-to-One cu CarModels)
  - O mașină este produsă de un anumit producător (Many-to-One cu CarManufacturers)

### 3.1.3 CarModels

- Definește modelele disponibile de vehicule
- **Cheie primară:** Id
- **Cheie străină:** CarManufacturerId (referință la CarManufacturers)
- **Atribute principale:** Name, ReleaseYear, CreatedAt, ModifiedAt, DeletedAt
- **Relații:**
  - Un model este produs de un singur producător (Many-to-One cu CarManufacturers)
  - Un model poate fi asociat cu mai multe mașini (One-to-Many cu Cars)
  - Un model poate avea mai multe componente asociate (One-to-Many cu CarModelsComponents)

### 3.1.4 CarManufacturers

- Conține informații despre producătorii de vehicule
- **Cheie primară:** Id
- **Atribute principale:** Name, CreatedAt, ModifiedAt, DeletedAt
- **Relații:**
  - Un producător poate avea mai multe modele de mașini (One-to-Many cu CarModels)
  - Un producător poate avea mai multe mașini produse (One-to-Many cu Cars)

### 3.1.5 Components

- Definește componentele disponibile pentru vehicule
- **Cheie primară:** Id
- **Cheie străină:** ComponentManufacturerId (referință la ComponentManufacturers)
- **Atribute principale:** Name, Price, CreatedAt, ModifiedAt, DeletedAt
- **Relații:**
  - O componentă este produsă de un singur producător de componente (Many-to-One cu ComponentManufacturers)
  - O componentă poate fi asociată cu mai multe modele de mașini (One-to-Many cu CarModelsComponents)
  - O componentă poate necesita mai multe reparații (One-to-Many cu ComponentsFixes)

### 3.1.6 ComponentManufacturers

- Conține informații despre producătorii de componente
- **Cheie primară:** Id
- **Atribute principale:** Name, YearOfFounding, CreatedAt, ModifiedAt, DeletedAt
- **Relații:** Un producător de componente poate produce mai multe componente (One-to-Many cu Components)

### 3.1.7 CarModelsComponents

- Tabel de legătură între modele de mașini și componente (relație Many-to-Many)
- **Chei primare compuse:** CarModelId, ComponentId
- **Atribute principale:** CreatedAt, ModifiedAt, DeletedAt
- **Relații:** Asociază modelele de mașini cu componentele compatibile

### 3.1.8 Fixes

- Conține informații despre reparații disponibile
- **Cheie primară:** Id
- **Atribute principale:** Description, Price, CreatedAt, ModifiedAt, DeletedAt
- **Relații:** O reparație poate fi necesară pentru mai multe componente (One-to-Many cu ComponentsFixes)

### 3.1.9 ComponentsFixes

- Tabel de legătură între componente și reparațiile asociate acestora
- **Cheie primară:** Id
- **Chei străine:** ComponentId (referință la Components), FixId (referință la Fixes)
- **Atribute principale:** MileageAtFix, FixId, CreatedAt, ModifiedAt, DeletedAt
- **Relații:** Asociază componentele cu reparațiile necesare și stochează informații suplimentare precum kilometrajul la momentul reparației

## 4 Prezentarea API-ului

Aplicația Vehicheck oferă o interfață REST API completă pentru toate operațiunile CRUD. Fiecare controller implementează operațiuni standard precum Create, Read, Update, Delete, precum și funcționalități avansate de filtrare și paginare.

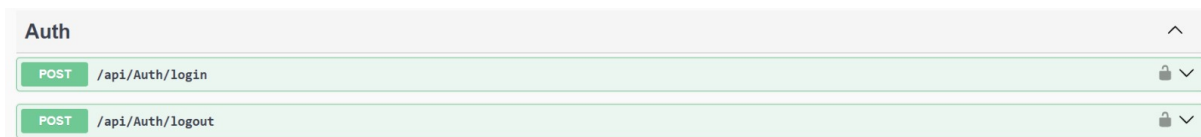


Figura 2: Endpoint-urile de autentificare din interfața Swagger

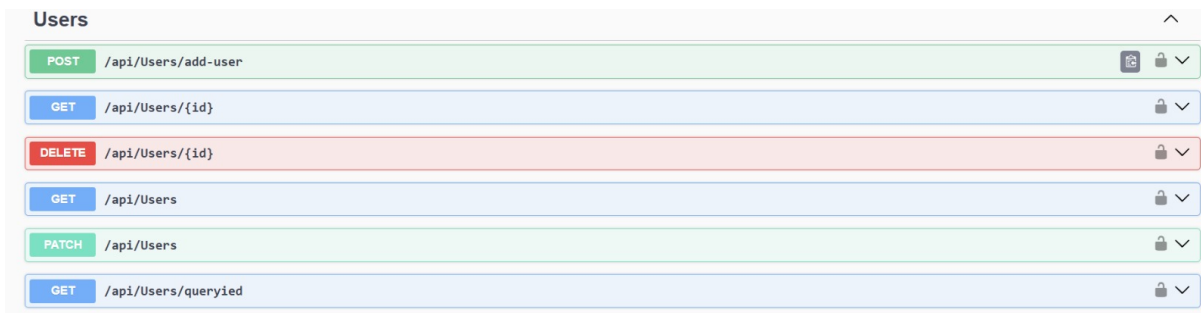
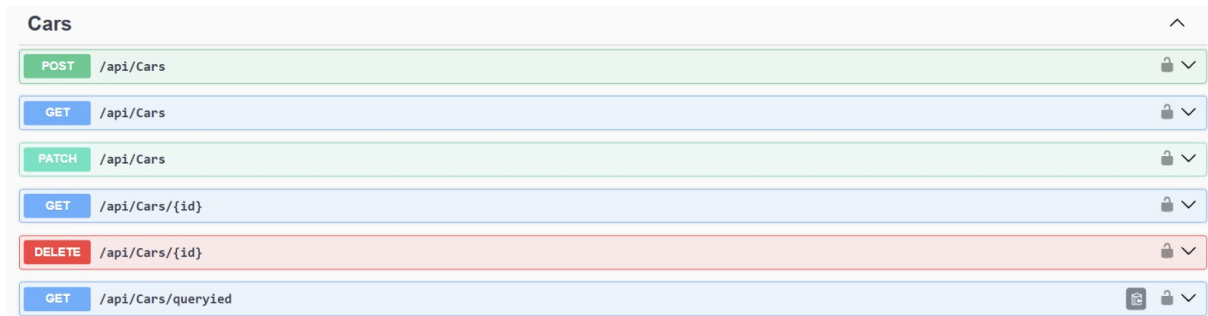


Figura 3: Endpoint-urile pentru gestionarea utilizatorilor





Cars		^
POST	/api/Cars	🔒
GET	/api/Cars	🔒
PATCH	/api/Cars	🔒
GET	/api/Cars/{id}	🔒
DELETE	/api/Cars/{id}	🔒
GET	/api/Cars/queryied	🔒

Figura 4: Endpoint-urile pentru gestionarea vehiculelor

#### 4.1 AuthController (api/auth)

- **POST** /login — Autentifică utilizatorul (Login)
- **POST** /logout — Deloghează utilizatorul (Logout)

#### 4.2 UsersController (api/users)

- **POST** /add-user — Creează un utilizator nou (Create)
- **GET** /{id} — Returnează datele unui utilizator după ID (Read)
- **GET** / — Returnează lista tuturor utilizatorilor (Read)
- **DELETE** /{id} — Șterge un utilizator după ID (Delete)
- **PATCH** / — Modifică datele unui utilizator (Update)
- **GET** /queryied — Returnează utilizatori filtrați sau paginați (Read avansat)

#### 4.3 CarsController (api/cars)

- **POST** / — Aduă o mașină nouă (Create)
- **GET** /{id} — Returnează datele unei mașini după ID (Read)
- **GET** / — Returnează toate mașinile (Read)
- **DELETE** /{id} — Șterge o mașină după ID (Delete)
- **PATCH** / — Modifică datele unei mașini (Update)
- **GET** /queryied — Mașini filtrate sau paginați (Read avansat)

#### 4.4 CarModelsController (api/carmodels)

- **POST** / — Creează un model de mașină (Create)
- **GET** /{id} — Detalii model după ID (Read)
- **GET** / — Toate modelele de mașini (Read)
- **DELETE** /{id} — Șterge un model de mașină după ID (Delete)

- **PATCH** / — Modifică un model de mașină (Update)
- **GET** /**queryied** — Modele filtrate sau paginate (Read avansat)

#### 4.5 CarManufacturersController (api/carmanufacturers)

- **POST** / — Creează un producător auto (Create)
- **GET** /{id} — Detalii producător după ID (Read)
- **GET** / — Toți producătorii auto (Read)
- **DELETE** /{id} — Șterge un producător auto după ID (Delete)
- **PATCH** / — Modifică un producător auto (Update)
- **GET** /**queryied** — Producători filtrați (Read avansat)

#### 4.6 ComponentManufacturersController (api/componentmanufacturers)

- **POST** / — Creează un producător de componente (Create)
- **GET** /{id} — Detalii producător de componente după ID (Read)
- **GET** / — Toți producătorii de componente (Read)
- **DELETE** /{id} — Șterge un producător de componente după ID (Delete)
- **PATCH** / — Modifică un producător de componente (Update)
- **GET** /**queryied** — Producători de componente filtrați (Read avansat)

#### 4.7 ComponentsController (api/components)

- **POST** / — Adaugă o componentă (Create)
- **GET** /{id} — Detalii componentă după ID (Read)
- **GET** / — Toate componentele (Read)
- **DELETE** /{id} — Șterge o componentă după ID (Delete)
- **PATCH** / — Modifică o componentă (Update)
- **GET** /**queryied** — Componente filtrate (Read avansat)

#### 4.8 FixesController (api/fixes)

- **POST** / — Adaugă o reparație (Create)
- **GET** /{id} — Detalii reparație după ID (Read)
- **GET** / — Toate reparațiile (Read)
- **DELETE** /{id} — Șterge o reparație după ID (Delete)

- **PATCH** / — Modifică o reparație (Update)
- **GET /queryied** — Reparații filtrate (Read avansat)

## 5 Utilizarea Aplicației

### 5.1 Tipuri de Utilizatori

#### 5.1.1 Administrator

- Acces complet la toate funcționalitățile sistemului
- Poate gestiona toți utilizatorii și vehiculele

#### 5.1.2 Utilizator Standard

- Poate gestiona propriile vehicule
- Poate adăuga, modifica și șterge informații despre vehiculele proprii
- Poate programa și urmări lucrările de întreținere

### 5.2 Procesul de Autentificare

#### 5.2.1 Înregistrare

1. Utilizatorul accesează pagina de înregistrare
2. Completează formularul cu informațiile personale (nume, email, parolă, număr de telefon)
3. Sistemul validează datele și creează contul
4. Utilizatorul primește confirmare prin email

#### 5.2.2 Autentificare

1. Utilizatorul introduce email-ul și parola
2. Sistemul verifică credențialele
3. La autentificare reușită, sistemul generează și returnează un JWT
4. Token-ul este stocat în local storage și utilizat pentru autentificarea ulterioară

#### 5.2.3 Securitate

Parolele sunt stocate criptat în baza de date, asigurând protecția datelor utilizatorilor.

## 6 Concluzii și Contribuții

### 6.1 Împărțirea Task-urilor

#### 6.1.1 Amândoi (Aruștei Andrei & Petre Flaviu)

- Arhitectura bazei de date
- Repository

#### 6.1.2 Aruștei Andrei

- Services
- Patch, Get, Post
- Paginare, sortare, filtrare

#### 6.1.3 Petre Flaviu

- Delete
- Controllers
- Middleware
- Autentificare, autorizare

### 6.2 Ce am Învățat

- Importanța planificării arhitecturii înainte de implementare
- Valoarea feedback-ului rapid și a iterațiilor
- Coordonarea eficientă între echipele de frontend și backend
- Gestionarea provocărilor de securitate în aplicațiile web moderne

### 6.3 Provocări Întâmpinate

- Asigurarea performanței sistemului cu volume mari de date
- Implementarea unui sistem robust de autentificare și autorizare
- Gestionarea concomitentă a mai multor versiuni de dezvoltare
- Asigurarea compatibilității între diferite medii

### 6.4 Îmbunătățiri Viitoare

- Implementarea unui sistem de notificări avansate
- Dezvoltarea unei aplicații mobile companion
- Adăugarea unui sistem de raportare și analiză avansată

## 7 Referințe

Link GitHub către codul proiectului:

<https://github.com/Andr3icutrei/Vehicheck>