# **Projekt: Aplikacja Wypożyczalni Samochodów**

Pełna aplikacja typu full-stack do zarządzania wypożyczalnią samochodów, obejmująca funkcjonalności dla klientów oraz panel administratora do zarządzania flotą, rezerwacjami i systemem lojalnościowym.

## **Spis Treści**

[**Projekt: Aplikacja Wypożyczalni Samochodów**](#_sy7vw3ad3575) **1**

[Spis Treści](#_1m9uvjl1w76) 1

[1. Architektura Aplikacji](#_ulh1vfaubzih) 1

[Frontend (Client-Side)](#_aho7qc5kvq2g) 1

[Backend (Server-Side)](#_odsywzdinsn9) 2

[Baza Danych](#_95fyley44lx8) 2

[2. Główne Funkcjonalności](#_1wsz7x9q6j8o) 2

[Funkcjonalności dla Użytkownika:](#_iqj7ld38fuke) 2

[Funkcjonalności dla Administratora:](#_ozyuj4wqqwa9) 2

[3. Struktura Bazy Danych (Schemat Prisma)](#_fmd9ll9wghfu) 2

[4. Instalacja i Uruchomienie](#_yqbd4gbc1xzr) 4

[4.1. Wymagania Wstępne](#_acijoc5ta4cp) 4

[4.2. Kroki Instalacji](#_g9hadcsmg54o) 4

[4.3. Uruchomienie Aplikacji](#_9lqez6tnc678) 5

[4.4. Logowanie jako Administrator](#_if1i39ornd17) 5

[4.5. Rozwiązywanie Najczęstszych Problemów](#_4j0jswkhz2y9) 5

[5. Kluczowe Endpointy API](#_8hjz6py1jgsu) 6

### **1. Architektura Aplikacji**

Aplikacja została zbudowana w architekturze client-server, składającej się z niezależnego frontendu (interfejs użytkownika) oraz backendu (serwer z logiką biznesową i API), które komunikują się ze sobą za pomocą REST API.

#### **Frontend (Client-Side)**

* **Język:** JavaScript (ES6+)
* **Framework:** React.js
* **Narzędzie budowania:** Vite
* **Routing:** React Router (react-router-dom)
* **Stylizacja:** Tailwind CSS
* **Zarządzanie stanem:** Zustand
* **Komunikacja z API:** Axios
* **Ikony:** Lucide React

#### **Backend (Server-Side)**

* **Język:** JavaScript
* **Środowisko uruchomieniowe:** Node.js
* **Framework:** Express.js
* **Interakcja z bazą danych (ORM):** Prisma
* **Uwierzytelnianie:** JWT (JSON Web Tokens)
* **Hashowanie haseł:** bcrypt.js
* **Wysyłka e-maili:** Nodemailer z transportem SendGrid
* **Zadania cykliczne:** node-cron

#### **Baza Danych**

* **System:** PostgreSQL
* **Narzędzie do zarządzania:** pgAdmin 4

### 2. Struktura Projektu i Plików

Projekt jest podzielony na dwa główne foldery: frontend i backend.

#### Struktura folderu backend

/backend

|-- /prisma # Konfiguracja i migracje bazy danych Prisma

| |-- schema.prisma # Główny plik definiujący strukturę bazy danych

| |-- seed.js # Skrypt do wypełniania bazy przykładowymi danymi

|-- /src # Główny folder z kodem źródłowym aplikacji

| |-- /controllers # Pliki z logiką dla każdego endpointu API

| |-- /middlewares # Funkcje pośredniczące (np. do weryfikacji tokenu JWT)

| |-- /routes # Pliki definiujące wszystkie ścieżki (endpointy) API

| |-- /services # Logika usług zewnętrznych (np. wysyłka e-maili)

| |-- index.js # Główny plik startowy serwera Express.js

|-- .env # Plik z poufnymi danymi (hasła, klucze API) - ignorowany przez Git

|-- package.json # Definicje projektu i lista zależności

#### Struktura folderu frontend

/frontend

|-- /public # Folder na statyczne zasoby (np. obrazki)

|-- /src # Główny folder z kodem źródłowym aplikacji

| |-- /api # Konfiguracja połączenia z backendem (Axios)

| |-- /assets # Zasoby (np. pliki CSS, dodatkowe obrazki)

| |-- /components # Małe, reużywalne komponenty (np. przycisk, nawigacja)

| |-- /hooks # Niestandardowe hooki React (np. do autoryzacji)

| |-- /pages # Komponenty reprezentujące całe strony/widoki

| |-- /store # Konfiguracja globalnego stanu (Zustand)

| |-- App.jsx # Główny komponent aplikacji, zawiera definicje routingu

| |-- main.jsx # Punkt wejścia aplikacji React

|-- index.html # Główny plik HTML, w którym renderowana jest aplikacja

|-- package.json # Definicje projektu i lista zależności

### **3. Główne Funkcjonalności**

Aplikacja realizuje wszystkie wymagania projektowe.

#### **Funkcjonalności dla Użytkownika:**

* Rejestracja konta z dwuetapową weryfikacją e-mail.
* Logowanie za pomocą e-maila lub nazwy użytkownika.
* Przeglądanie listy dostępnych samochodów.
* Rezerwacja samochodu na wybrane daty.
* System lojalnościowy: zdobywanie 10 punktów za każdy dzień wynajmu, możliwość wykorzystania punktów do uzyskania zniżki (1 pkt = 0.10 PLN, do 50% wartości rezerwacji).
* Podgląd historii swoich rezerwacji.
* Możliwość anulowania przyszłych rezerwacji (z poziomu aplikacji lub przez link w e-mailu).

#### **Funkcjonalności dla Administratora:**

* Dostęp do chronionego panelu administratora.
* Dodawanie, edycja i usuwanie samochodów z floty.
* Zarządzanie statusem pojazdów (Dostępny, Wypożyczony, Niedostępny).
* Dodawanie wpisów do historii serwisowej.1

### **4. Struktura Bazy Danych (Schemat Prisma)**

Struktura bazy danych jest zdefiniowana w pliku backend/prisma/schema.prisma.

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "postgresql"

url = env("DATABASE\_URL")

}

model User {

id String @id @default(cuid())

email String @unique

username String @unique

password String

role Role @default(USER)

points Int @default(0)

emailVerified Boolean? @default(false)

emailVerificationToken String? @unique

emailVerificationTokenExpires DateTime?

bookings Booking[]

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

}

model Car {

id String @id @default(cuid())

make String

model String

year Int

licensePlate String @unique

pricePerDay Float

status CarStatus @default(AVAILABLE)

imageUrl String?

bookings Booking[]

serviceHistory ServiceLog[]

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

}

model Booking {

id String @id @default(cuid())

userId String

carId String

startDate DateTime

endDate DateTime

totalPrice Float

status String @default("CONFIRMED") // np. CONFIRMED, CANCELLED, COMPLETED

pointsUsed Int @default(0)

cancellation\_token String? @unique // <-- NOWE POLE

user User @relation(fields: [userId], references: [id])

car Car @relation(fields: [carId], references: [id])

createdAt DateTime @default(now())

}

model ServiceLog {

id String @id @default(cuid())

carId String

date DateTime

description String

cost Float

car Car @relation(fields: [carId], references: [id])

createdAt DateTime @default(now())

}

enum Role {

USER

ADMIN

}

enum CarStatus {

AVAILABLE

RENTED

UNAVAILABLE

}

### **5. Instalacja i Uruchomienie**

#### **5.1. Wymagania Wstępne**

* **Node.js (wersja LTS):** [nodejs.org](https://nodejs.org/)
* **PostgreSQL (z pgAdmin):** [enterprisedb.com](https://www.enterprisedb.com/downloads/postgres-postgresql-downloads)

#### **5.2. Kroki Instalacji**

1. **Pobierz kod projektu**
2. **Stwórz bazę danych:** Uruchom pgAdmin, połącz się z lokalnym serwerem i stwórz nową, pustą bazę danych o nazwie wypozyczalnia.
3. **Skonfiguruj Backend:**
   * Przejdź do pliku .env i zmień hasło na swoje w odpowiedniej linii

DATABASE\_URL="postgresql://postgres:TWOJE\_HASLO\_DO\_BAZY@localhost:5432/wypozyczalnia"

* + Zainstaluj zależności: npm install
  + Uruchom migracje, aby stworzyć tabele: npx prisma migrate dev

1. **Skonfiguruj Frontend:**
   * Otwórz nowy terminal i przejdź do folderu frontend.
   * Zainstaluj zależności: npm install

#### **5.3. Uruchomienie Aplikacji**

1. Uruchom plik “uruchom strone”

#### **5.4. Logowanie jako Administrator**

1. Zarejestruj konto przez formularz w aplikacji i przejdź proces weryfikacji e-mail.
2. Użyj pgAdmin, aby w tabeli User zmienić wartość w kolumnie role dla swojego konta z USER na ADMIN.
3. Zaloguj się ponownie.

#### **5.5. Rozwiązywanie Najczęstszych Problemów**

* **Problem:** Błąd running scripts is disabled on this system w terminalu PowerShell.  
  + **Przyczyna:** Domyślne zabezpieczenia Windows blokujące skrypty.
  + **Rozwiązanie:** Uruchom PowerShell jako Administrator i wykonaj komendę: Set-ExecutionPolicy RemoteSigned -Scope CurrentUser. Następnie potwierdź, wpisując T.
* **Problem:** Błąd P1000: Authentication failed... podczas migracji Prismy.  
  + **Przyczyna:** Niepoprawne hasło do bazy danych w pliku backend/.env.
  + **Rozwiązanie:** Sprawdź dokładnie DATABASE\_URL w pliku .env. Upewnij się, że hasło jest identyczne z tym, które zostało ustawione podczas instalacji PostgreSQL.
* **Problem:** Błąd EPERM: operation not permitted, rename '...' podczas npm install lub komend prisma.  
  + **Przyczyna:** Inny program (najczęściej antywirus lub synchronizacja OneDrive) blokuje dostęp do plików w node\_modules.
  + **Rozwiązanie:** Przenieś projekt w lokalizację niesynchronizowaną z chmurą (np. C:\projekty). Dodaj folder projektu do listy wykluczeń w programie antywirusowym.
* **Problem:** E-maile nie dochodzą.  
  + **Przyczyna:** Błędny klucz API lub niezweryfikowany adres nadawcy w konfiguracji SendGrid.
  + **Rozwiązanie:** Sprawdź dokładnie zmienne SENDGRID\_API\_KEY i SENDER\_EMAIL w pliku .env. Upewnij się, że użyty e-mail został zweryfikowany jako "Single Sender" w panelu SendGrid.

### **6. Kluczowe Endpointy API**

*Wszystkie endpointy są dostępne pod prefiksem /api.*

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Ścieżka** | **Opis** |
| POST | /auth/register | Rejestracja nowego użytkownika. |
| POST | /auth/verify-email | Weryfikacja konta za pomocą kodu. |
| POST | /auth/login | Logowanie użytkownika. |
| GET | /cars | Pobiera listę dostępnych aut. |
| GET | /cars/:id | Pobiera szczegóły jednego auta. |
| GET | /bookings/my-bookings | Pobiera historię rezerwacji zalogowanego usera. |
| POST | /bookings | Tworzy nową rezerwację. |
| DELETE | /bookings/:id | Anuluje rezerwację (przez zalogowanego usera). |
| GET | /bookings/cancel/:id/:token | Anuluje rezerwację (przez link z e-maila). |
| POST | /admin/cars | Dodaje nowy samochód. |
| GET | /admin/cars | Pobiera listę wszystkich samochodów. |
| PATCH | /admin/cars/:id | Aktualizuje pełne dane samochodu. |
| DELETE | /admin/cars/:id | Usuwa samochód. |
| PATCH | /admin/cars/:id/status | Zmienia status samochodu. |
| POST | /admin/cars/service-log | Dodaje wpis serwisowy. |