

AI1	Dokumentacja projektu
Autor	Michał Pilecki, 125151
Kierunek, rok	Informatyka, II rok, st. stacjonarne (3,5-l)
Temat projektu	Portal aukcyjny (licytacja) samochodów używanych

1. Wstęp

Projekt skupia się na aplikacji internetowej dla przedsiębiorstwa zajmującego się licytacją samochodów używanych wykonanej przy użyciu frameworka Laravel w wersji 11. Aplikacja ma na celu pomóc i usprawnić licytację i dodawanie w łatwy i prosty sposób nowych ogłoszeń. W aplikacji są dwa poziomy kont: zwykła (user) oraz administrator (admin), dzięki temu aplikacja umożliwia sprawne funkcjonowanie takiej firmy. Aplikacja pozwala na przeglądanie różnych ofert nie tylko użytkownikom zalogowanym ale również nowym gościom, którzy po raz pierwszy odwiedzają ten serwis.

Zaimplementowane funkcjonalności:

- Logowanie: każdy użytkownik, który posiada konto może się zalogować i mieć dostęp do swoich danych i informacji na temat swoich aukcji, w których bierze udział albo które posiada może zawsze taką ofertę zmodyfikować lub usunąć.
- System licytacji: system, który jest podstawą tej aplikacji czyli walka w licytacji pomiędzy użytkownikami o wartościowe samochody oraz system, który w odpowiednim czasie zamknie licytację i powie kto wygrał.
- Zarządzanie danymi: czyli podstawowy system CRUD dla admina jest pełny dostęp, może edytować informację o każdym użytkowniku i samochodzie, może też usunąć, użytkownik za to może jedynie edytować zawartość, która sam posiada czyli tylko swoje oferty.
- Bezpieczeństwo: aplikacja posiada system hashowania haseł, hasła przechowywane w bazie danych są hashowane co sprawia że złamanie takiego hasła z bazy danych jest praktycznie nie możliwe

Założenia Techniczne:

- Framework: Laravel 11

- Baza danych: MySQL
- Frontend: Szablony Blade, CSS, JS
- Backend: PHP + Laravel
- Inne technologie: Bootstrap 5 zapewnia schludny wygląd interface'u na każdym urządzeniu

Aplikacja serwisu aukcyjnego została zaprojektowana tak aby wszystkie czynności, czyli system licytacji, dodawania i edytowania ogłoszeń był zautomatyzowany i wygodny dla użytkownika i administratora.

2. Narzędzia i technologie

Framework Laravel

Laravel to znany framework open-source do tworzenia aplikacji webowych w języku PHP. Charakteryzuje się czytelnym kodem i wykorzystaniem architektury Model-View-Controller, co ułatwia zarządzanie projektem. Zapewnia elastyczne zarządzanie trasami oraz narzędzia do pracy z bazą danych poprzez Eloquent ORM. Korzysta z Blade - silnika szablonów, ułatwiającego tworzenie interfejsów użytkownika. Dzięki middleware'om można przetwarzać żądania HTTP przed ich przekazaniem do kontrolera. Oferuje narzędzia do testowania aplikacji oraz gotowe biblioteki przyspieszające rozwój. Istnieje też szeroki ekosystem dodatkowych pakietów i rozszerzeń stworzonych przez społeczność. Podsumowując, Laravel to elastyczny framework ułatwiający tworzenie zaawansowanych aplikacji webowych w PHP.

Baza danych

MySQL to popularny system zarządzania bazą danych często stosowany w aplikacjach webowych. Charakteryzuje się skalowalnością, umożliwiając obsługę dużych ilości danych i użytkowników. Zapewnia zaawansowane mechanizmy bezpieczeństwa, takie jak kontrole dostępu i szyfrowanie danych. MySQL obsługuje transakcje, co pozwala na grupowanie operacji bazodanowych w logiczne jednostki, zapewniając spójność danych. Dodatkowo, oferuje wsparcie dla różnorodnych typów danych, w tym tekstowych, liczbowych i dat. Podsumowując, MySQL to niezawodny system bazodanowy, który sprawdza się doskonale w aplikacjach webowych

Konfiguracja połączenia z serwerem bazodanowym

Aby skonfigurować połączenia z bazą danych należy skonfigurować plik `config/database.php` oraz `.env`. w `database.php` należy mieć taki fragment kodu jak poniżej:

```

41
42     'mysql' => [
43         'driver' => 'mysql',
44         'url' => env('DB_URL'),
45         'host' => env('DB_HOST', '127.0.0.1'),
46         'port' => env('DB_PORT', '3306'),
47         'database' => env('DB_DATABASE', 'laravel'),
48         'username' => env('DB_USERNAME', 'root'),
49         'password' => env('DB_PASSWORD', ''),
50         'unix_socket' => env('DB_SOCKET', ''),
51         'charset' => env('DB_CHARSET', 'utf8mb4'),
52         'collation' => env('DB_COLLATION', 'utf8mb4_unicode_ci'),
53         'prefix' => '',
54         'prefix_indexes' => true,
55         'strict' => true,
56         'engine' => null,
57         'options' => extension_loaded('pdo_mysql') ? array_filter([
58             PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
59         ]) : [],
60     ],

```

A w pliku .env należy mieć taki kawałek kodu:

```

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3307
DB_DATABASE=auction_site
DB_USERNAME=root
DB_PASSWORD=

```

W każdym z tych kodów należy podać odpowiednie dane zależności jaką bazę stworzy, jaki port używa jego baza danych i jakie ma hasło i nazwę użytkownika w bazie danych.

Kontrolery

W Laravelu to kontrolery odpowiadają za całą logikę aplikacji, to właśnie tam w jest napisana cała logika logowania, dodawania, usuwania, edytowania danych oraz zarządzanie systemem licytacji. AuthController służy do obsługi logowania

```

class AuthController extends Controller
{
    // 👤 👤
    public function login()
    {
        if (Auth::check()) {
            return redirect()->route('cars.index');
        }
        return view('auth.login');
    }

    /**
     * Handle an authentication attempt.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
    public function authenticate(Request $request)
    {
        $credentials = $request->validate([
            'email' => ['required', 'email'],
            'password' => ['required'],
        ]);

        if (Auth::attempt($credentials)) {
            $request->session()->regenerate();
            return redirect()->route('cars.index');
        }

        return back()->withErrors([
            'email' => 'Nie prawidłowy email lub hasło.',
        ])->onlyInput('email');
    }
}

```

Modele

W Laravelu modele są odpowiedzialne za wygląd połączeń tabel w bazie oraz definicję jakich kolumn z danej tabeli można używać. Przykładowy model Annoucement, który odpowiada z tabelę z ogłoszeniami, definiuje połączenia z innymi tabelami.

```
class Announcement extends Model
{
    use HasFactory;

    protected $fillable = ['user_id', 'name', 'brand', 'year', 'mileage', 'description', 'end_date', 'is_end', 'min_price'];

    public $timestamps = false;

    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class);
    }

    public function bids(): HasMany
    {
        return $this->hasMany(Bid::class);
    }

    public function photos(): HasMany
    {
        return $this->hasMany(Photo::class);
    }

    public function randomPhoto()
    {
        return $this->photos()->inRandomOrder()->first();
    }
}
```

Walidacja Danych

Walidacja danych jest jednym z ważniejszych czynników gdyż należy uważać co użytkownik poda, nie zawsze użytkownik poda poprawną wartość, może być złośliwy i podać ujemną cenę lub rok, w takim przypadku należy przewidzieć taką możliwość i zabezpieczyć to w Laravelu można to zabezpieczyć na 2 sposoby po stronie backendu lub frontendu. Walidacja po stronie backendu jest bardziej bezpieczna na ingerencję użytkownika, gdyż nie może tak łatwo jej zmienić jak po stronie frontendu. Przykładowa walidacja zawiera informację o tym czy dana wartość jest wymagana i czy zawiera jakieś dodatkowe kryteria, które trzeba spełnić na przykład rok może być przyjmowany tylko pomiędzy 1976 a aktualnym czyli na czas pisania dokumentacji 2024. Dodatkowo można też dodać informację w razie wystąpienia problemów gdy użytkownik nie spełni konkretnego warunku

```

$request->validate([
    'name' => 'required|string|max:30',
    'brand' => 'required|string|max:30',
    'year' => 'required|integer|digits:4|min:1976|max:' . date('Y'),
    'mileage' => 'required|numeric|min:0',
    'description' => 'nullable|string',
    'min_price' => 'required|numeric|min:100',
    'end_date' => 'required|date|after:now',
    'images.*' => 'image|mimes:jpeg,png,jpg,gif,svg|max:2048',
], [
    'name.required' => 'Pole Marka jest wymagane.',
    'name.string' => 'Pole Marka musi być ciągiem znaków.',
    'name.max' => 'Pole Marka może mieć maksymalnie 30 znaków.',
    'brand.required' => 'Pole Model jest wymagane.',
    'brand.string' => 'Pole Model musi być ciągiem znaków.',
    'brand.max' => 'Pole Model może mieć maksymalnie 30 znaków.',
    'year.required' => 'Pole Rok produkcji jest wymagane.',
    'year.integer' => 'Pole Rok produkcji musi być liczbą całkowitą.',
    'year.digits' => 'Pole Rok produkcji musi mieć 4 cyfry.',
    'year.min' => 'Pole Rok produkcji nie może być wcześniejsze niż 1976.',
    'year.max' => 'Pole Rok produkcji nie może być późniejsze niż bieżący rok.',
    'mileage.required' => 'Pole Przebieg jest wymagane.',
    'mileage.numeric' => 'Pole Przebieg musi być liczbą.',
    'mileage.min' => 'Pole Przebieg nie może być mniejsze niż 0.',
    'description.string' => 'Pole Opis musi być ciągiem znaków.',
    'min_price.required' => 'Pole Cena jest wymagane.',
    'min_price.numeric' => 'Pole Cena musi być liczbą.',
    'min_price.min' => 'Pole Cena nie może być mniejsze niż 100.',
    'end_date.required' => 'Pole Data końcowa jest wymagane.',
    'end_date.date' => 'Pole Data końcowa musi być prawidłową datą.'
]);

```

Wstępne wypełnienie bazy danych

W celu uzupełnienia bazy danych Laravel wykorzystuje seedery, które automatycznie po wykonaniu komendy uzupełniają bazę danych. Seedery przydają się głównie do testowania zaimplementowanej bazy danych oraz testowaniu całej aplikacji w warunkach bardzo przypominających prawdziwe

```

12  /**
13   * Run the database seeds.
14   */
15  public function run(): void
16  {
17
18      Schema::withoutForeignKeyConstraints(function () {
19          Announcement::truncate();
20      });
21
22      Announcement::insert([[
23          'user_id' => 1,
24          'name' => 'Audi',
25          'brand' => 'A4',
26          'year' => 2000,
27          'mileage' => 321.23,
28          'description' => 'Samochód w bardzo dobrym stanie pierwszy właściciel.',
29          'end_date' => '2024-05-27 18:27',
30          'is_end' => false,
31          'min_price' => 200
32      ]],
33
34      [
35          'user_id' => 1,
36          'name' => 'BMW',
37          'brand' => 'M',
38          'year' => 2021,
39          'mileage' => 32.23,
40          'description' => 'Samochód w bardzo dobrym stanie pierwszy właściciel.',
41          'end_date' => '2024-06-08',

```

Przykład walidacji po stronie frontendu:

```

<div class="form-label">marka</label>
<input type="text" id="name" name="name" value="{{ old('name', $announcement->name) }}" required maxlength="255">

<div class="form-label">Model</label>
<input type="text" id="brand" name="brand" value="{{ old('brand', $announcement->brand) }}" required maxlength="255">

<div class="form-label">Rok produkcji</label>
<input type="text" id="year" name="year" value="{{ old('year', $announcement->year) }}" required min="1976" max="{{ date('Y') }}">

<div class="form-label">Przebieg</label>
<input type="text" id="mileage" name="mileage" value="{{ old('mileage', $announcement->mileage) }}" required min="0">

<div class="form-label">Cena</label>
<input type="text" id="price" name="min_price" value="{{ old('min_price', $announcement->min_price) }}" required min="0.01" step="0.01">

<div class="form-label">Opis</label>
<div class="form-textarea" id="description" name="description" rows="5" required maxlength="1000">{{ old('description', $announcement->description) }}</div>

<div class="form-buttons">
    <button type="button" class="btn btn-primary">Zapisz zmiany</button>
    <button type="button" class="btn btn-secondary">Anuluj</button>
</div>

```

IVI

Rok produkcji

2021

Przebieg

32

Cena

20000,00

Opis

Walidacja po stronie backendu:

Dodaj nowe ogłoszenie

- validation.min.numeric

Marka

Wprowadź markę samochodu

Model

Wprowadź model samochodu

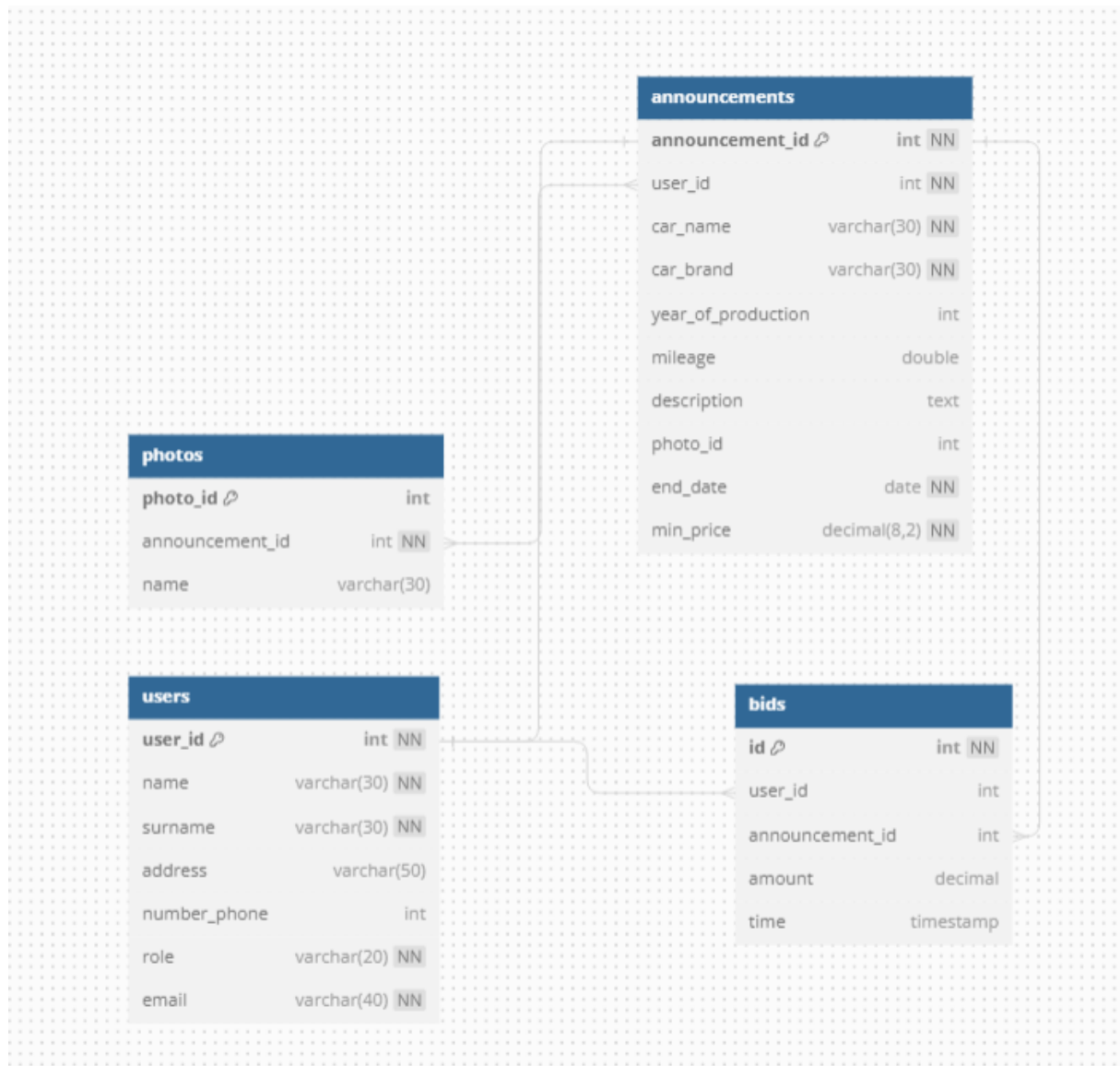
Rok produkcji

Wprowadź rok produkcji

3. Baza danych

Diagram

ERD:



Opis bazy danych

Diagram przedstawia bazę danych dla aplikacji obsługującej komis samochodowy, składającą się z czterech tabel: announcements (ogłoszenia), photos (zdjęcia), users (użytkownicy) i bids (oferty). Opis tabel i ich powiązań wygląda następująco:

Tabela announcements (ogłoszenia)

announcement_id (int, PK): Unikalny identyfikator ogłoszenia.

user_id (int, NN): Identyfikator użytkownika, który dodał ogłoszenie (odwołanie do tabeli users).

car_name (varchar(30), NN): Nazwa marki samochodu.

car_brand (varchar(30), NN): Model samochodu.

year_of_production (int, NN): Rok produkcji samochodu.

mileage (double, NN): Przebieg samochodu w kilometrach.

description (text, NN): Opis samochodu.

photo_id (int): Identyfikator zdjęcia (odwołanie do tabeli photos).

end_date (date, NN): Data zakończenia aukcji.

min_price (decimal(8,2), NN): Minimalna cena za samochód.

Tabela photos (zdjęcia)

photo_id (int, PK): Unikalny identyfikator zdjęcia.

announcement_id (int, NN): Identyfikator ogłoszenia, do którego przypisane jest zdjęcie (odwołanie do tabeli announcements).

name (varchar(30), NN): Nazwa pliku zdjęcia.

Tabela users (użytkownicy)

user_id (int, PK): Unikalny identyfikator użytkownika.

name (varchar(30), NN): Imię użytkownika.

surname (varchar(30), NN): Nazwisko użytkownika.

address (varchar(50), NN): Adres użytkownika.

number_phone (int, NN): Numer telefonu użytkownika.

role (varchar(20), NN): Rola użytkownika (np. admin, użytkownik).

email (varchar(40), NN): Adres email użytkownika.

Tabela bids (oferty)

id (int, PK): Unikalny identyfikator oferty.

user_id (int, NN): Identyfikator użytkownika, który złożył ofertę (odwołanie do tabeli users).

announcement_id (int, NN): Identyfikator ogłoszenia, na które złożono ofertę (odwołanie do tabeli announcements).

amount (decimal, NN): Kwota oferty.

time (timestamp, NN): Data i czas złożenia oferty.

Relacje między tabelami

Relacja jeden-do-wielu między users a announcements:

Jeden użytkownik może dodać wiele ogłoszeń.

announcements.user_id odwołuje się do users.user_id.

Relacja jeden-do-wielu między announcements a photos:

Jedno ogłoszenie może mieć wiele zdjęć.

photos.announcement_id odwołuje się do announcements.announcement_id.

Relacja jeden-do-wielu między users a bids:

Jeden użytkownik może złożyć wiele ofert.

bids.user_id odwołuje się do users.user_id.

Relacja jeden-do-wielu między announcements a bids:

Jedno ogłoszenie może mieć wiele ofert.

bids.announcement_id odwołuje się do announcements.announcement_id.

Migracje

W Laravelu służą do utworzenia tabel i kolumn, wskazaniu jakie dane kolumna będzie przechowywać typy oraz zdefiniowaniu kluczy obcych lub wartości unikatowych czy mogących być nullami. Przykład migracji announcement:

```
/*  
public function up(): void  
{  
    Schema::create('announcements', function (Blueprint $table) {  
        $table->id();  
        $table->foreignId('user_id')->constrained()->onDelete('cascade');  
        $table->string('name', 30);  
        $table->string('brand', 30);  
        $table->integer('year');  
        $table->integer('mileage');  
        $table->text('description')->nullable();  
        $table->timestamp('end_date');  
        $table->boolean('is_end');  
        $table->decimal('min_price', 10, 2);  
        $table->timestamps();  
    });  
}  
/**
```

Ten kod definiuje schemat tabeli 'announcements' w bazie danych. Zawiera on kolumny dla identyfikatora ogłoszenia ('id'), identyfikatora użytkownika ('user_id'), nazwy ('name'), marki ('brand'), roku ('year'), przebiegu ('mileage'), opisu ('description'), daty zakończenia ('end_date'), informacji czy ogłoszenie jest zakończone ('is_end'), minimalnej ceny ('min_price') oraz znaczników czasowych 'created_at' i 'updated_at' dla śledzenia czasu utworzenia i ostatniej aktualizacji rekordu.

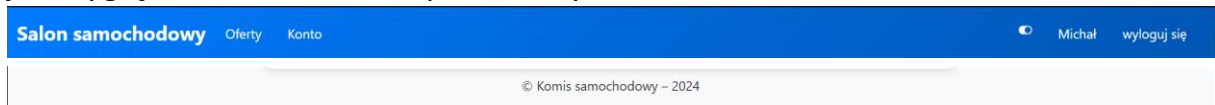
4. GUI

Opis Interfejsu użytkownika

Graficzny interfejs użytkownika pozwala aplikacji na łatwy i prosty sposób zarządzać aplikacją, a użytkownikom pozwala na łatwy sposób włączenia się do licytacji lub dodania własnego samochodu.

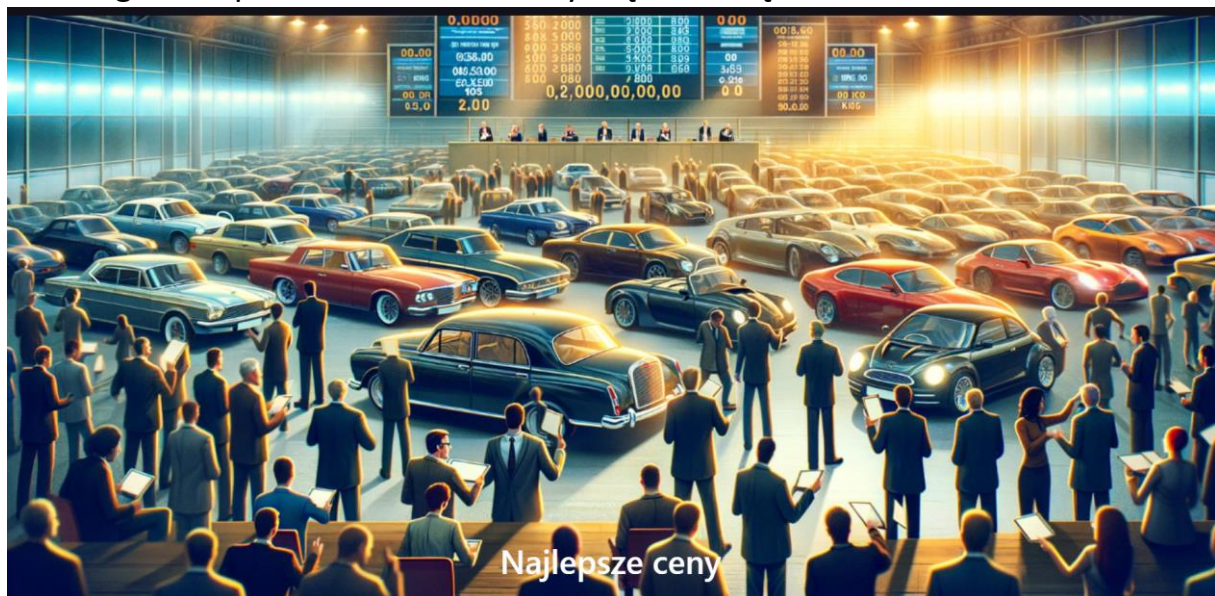
Wybrane widoki

GUI posiada szablon niektórych elementów, które są wykorzystywane wszędzie żeby utrzymać podobny poziom i użytkownik nie myślał, że przeszedł na inną stronę lub że aplikacja jest stroną phishingową. Poniżej przedstawiony jest jak wygląda navbar oraz stopka strony:



Navbar zmienia się w zależności od stanu, czy użytkownik jest zalogowany tak jak jest to pokazane, jeśli tak to jego imię jest obok wyloguj się. Również w zależności od uprawnień użytkownik może mieć więcej lub mniej opcji do wyboru na pasku i nie mogło przecież zabraknąć guzika do zmiany motywu aplikacji, pokazywana aplikacja jest w trybie jasnym, domyślnym trybem aplikacji jest ciemny.

Strona główna posiada również olbrzymią karuzelę:





Ostatnim z ważniejszych elementów na stronie to są karty, które wyświetlają najważniejszą część aplikacji czyli, samochody, licytację oraz szczegółowe informacje o samochodzie



Audi

Samochód w bardzo dobrym stanie
pierwszy właściciel.

[Więcej szczegółów...](#)

Toyota Corolla



Toyota Corolla

Opis:

Samochód w świetnym stanie, bezwypadkowy.

Rok produkcji:

2018

Przebieg:

40001 km

Data końca licytacji:

2024-06-15 00:00:00

Cena minimalna: 18000.00 zł

Aktualnie brak ofert

Licytuj

Ważną stroną jest zarówno strona profilowa użytkownika jak i panel admina, który wygląda następująco:

Michał Pilecki

Dane:

Imię: Michał

Nazwisko: Pilecki

Adres: Ulica

Numer telefonu: 123415092

Moje ogłoszenia

Marka: Audi

Model: A4

Rok produkcji: 2000

Edytuj

Usuń

Marka: BMW

Model: M

Rok produkcji: 2021

Edytuj

Usuń

Marka: Mercedes

Model: C-Class

Rok produkcji: 2015

Edytuj

Usuń

Marka: Volkswagen

Model: Golf

Rok produkcji: 2019

Edytuj

Usuń

Dodaj nowe ogłoszenie

Marka

Wprowadź markę samochodu

Model

Wprowadź model samochodu

Rok produkcji

Wprowadź rok produkcji

Przebieg (km)

Wprowadź przebieg samochodu

Opis

Dodaj opis samochodu



Data zakończenia

dd.mm.rrrr



Cena minimalna (PLN)

Wprowadź cenę minimalną

Zdjęcia

Wybierz pliki

Nie wybrano pliku

Dodaj ogłoszenie

Aukcje, w których biorę udział

Nie bierzesz udziału w żadnych aukcjach.

To właśnie na tej stronie użytkownik może zarządzać swoimi zasobami przeglądać co sam licytuje i wyświetlać samochody, które próbuje sprzedać.

Użytkownicy

ID	Nazwa	Email	Rola	Akcje
1	Michał	michal2@mail.com	user	<button>Edytuj</button> <button>Usuń</button>
2	Adam	nowak@mail.com	user	<button>Edytuj</button> <button>Usuń</button>
4	Maciej	kowal@mail.com	user	<button>Edytuj</button> <button>Usuń</button>

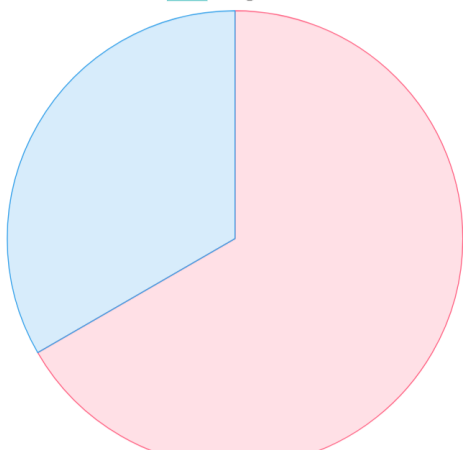
Oferty

ID	Marka	Model	Rok	Akcje
1	Audi	A4	2000	<button>Edytuj</button> <button>Usuń</button>
2	BMW	M	2021	<button>Edytuj</button> <button>Usuń</button>
3	Mercedes	C-Class	2015	<button>Edytuj</button> <button>Usuń</button>
4	Volkswagen	Golf	2019	<button>Edytuj</button> <button>Usuń</button>
5	Toyota	Corolla	2018	<button>Edytuj</button> <button>Usuń</button>
6	Ford	Focus	2017	<button>Edytuj</button> <button>Usuń</button>

Admin zaś posiada wgląd w dane wszystkich użytkowników, wszystkich ofert może je edytować i usunąć.

Liczba ogłoszeń użytkowników

■ michal2@mail.com ■ nowak@mail.com ■ admin@mail.com
■ kowal@mail.com



Statystyki

Średnia liczba ogłoszeń na użytkownika: 1.50

Średnia cena ofert: 3,213.00 zł

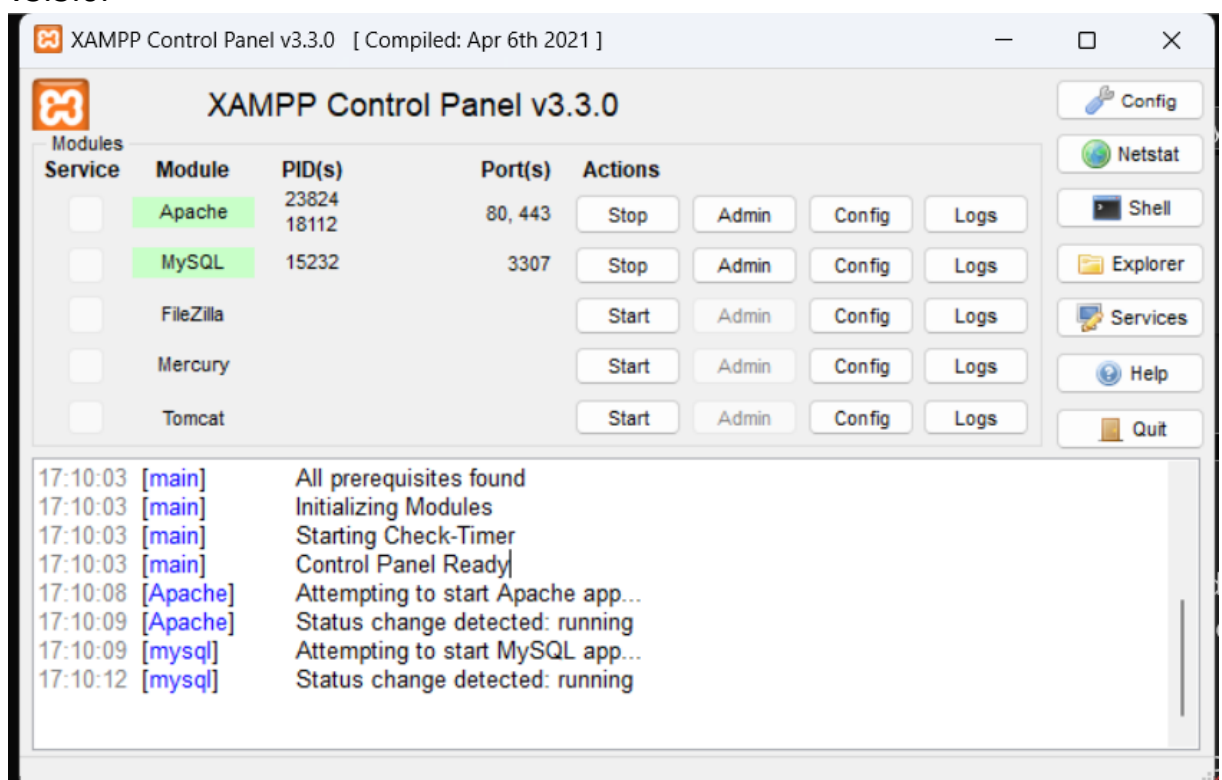
Dodatkowo admin może monitorować takie zasoby jak liczbę ogłoszeń widzi jaki użytkownik posiada najwięcej ogłoszeń oraz widzi jaka jest średnia cena oferty startowej za każdy samochód.

Wszystko jest wykonane w Bootstrapie więc problem responsywności jest załatwiony, gdyż twórcy Bootstrapa zapewniają, że strona będzie miała spójny i elastyczny wygląd.

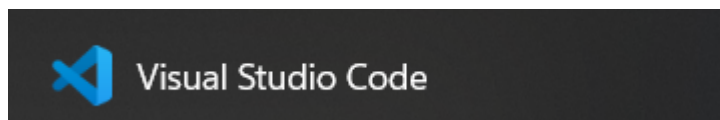
5. Uruchomienie aplikacji

Co trzeba przygotować?

Wymagany do uruchomienia aplikacji jest najnowsza wersja Xamppa v3.3.0:



Visual Studio Code:



Do przygotowania aplikacji został przygotowany skrypt który zainstaluje wszystko co jest wymagane, należy go odpalić 2 razy klikając na niego, Windows może ostrzec użytkownika że jest to niebezpieczny plik z nieznanego źródła jednak nie należy się tym martwić bo w każdej chwili można podejrzec co jest w środku bo nie jest to plik exe, który jest skompilowany i użytkownik nie ma pojęcia co jest w środku. Plik startowy został przygotowany na platformę Windows.

```
%systemDrive%\xampp\mysql\bin\mysql -uroot -e "CREATE DATABASE IF NOT EXISTS auction_site;"

if %errorlevel% neq 0 msg %username% "Nie udało się utworzyć bazy danych." && exit /b %errorlevel%

php -r "copy('.env.example', '.env');"

call composer install

call php artisan migrate:fresh --seed

call php artisan key:generate

call php artisan storage:link

code .
```

Jeśli to wszystko mamy zrobione czyli, Xampp został włączony Apache i MySQL i skrypt został odpalony to po czasie jak nie wywali błędu powinien uruchomić się Visual Studio Code, który będzie otwarty w miejscu projektu. To już praktycznie ostatni krok żeby odpalić aplikację zostaje tylko wpisać następującą komendę w konsoli: `php artisan serve`

```
Microsoft Windows [Version 10.0.22631.3672]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\micha\OneDrive - Uniwersytet Rzeszowski\Pulpit\strona_internetowa\nauka
C:\Users\micha\OneDrive - Uniwersytet Rzeszowski\Pulpit\strona_internetowa\nauka\AplikacjeInternetowe\Projekt>php artisan serve

[INFO] Server running on [http://127.0.0.1:8000].

Press Ctrl+C to stop the server
```

Efekt wpisania tej komendy powinien być następujący jeśli wszystko poszło dobrze. Aplikacja powinna być uruchomiona, żeby ją wyświetlić należy wpisać w przeglądarce <http://127.0.0.1:8000/home> żeby wyświetlić główną stronę aplikacji. W bazie danych są umieszczeni przykładowi użytkownicy:

```

User::insert([
  'name' => 'Michał',
  'surname' => 'Pilecki',
  'email' => 'michal2@mail.com',
  'password' => Hash::make('1234'),
  'address' => 'Ulica',
  'phone_number' => 123415092,
  'role' => 'user'
],
[
  'name' => 'Adam',
  'surname' => 'Nowak',
  'email' => 'nowak@mail.com',
  'password' => Hash::make('1234'),
  'address' => 'Ulica',
  'phone_number' => 123413359,
  'role' => 'user'
],
[
  'name' => 'Admin',
  'surname' => 'Admin',
  'email' => 'admin@mail.com',
  'password' => Hash::make('1234'),
  'address' => 'Admin',
  'phone_number' => 123413343,
  'role' => 'admin'
],
[
  'name' => 'Maciej',
  'surname' => 'Kowalski',
  'email' => 'kowal@mail.com',
  'password' => Hash::make('1234'),
  'address' => 'Dobra 6',
  'phone_number' => 123413323,
  'role' => 'user'
],

```

6. Funkcjonalności aplikacji

Aplikacja posiada system logowania:

Zaloguj się

Email

Hasło

Wyślij

Dzięki temu wiadomo jaki użytkownik przegląda stronę, licytuje samochód lub wystawia ogłoszenie.

System zakończenia licytacji:

Cena minimalna: 200.00 zł

Aktualnie oferowana cena: 3213.00 zł

Najwyższą ofertę złożył: nowak@mail.com

Licytacja zakończona

Zwycięzca licytacji: nowak@mail.com

Licytuj

Kończy licytację informując, kto wygrał jeśli ktoś licytował oraz uniemożliwia składanie kolejnych ofert.

Cena minimalna: 20000.00 zł

Aktualnie brak ofert


Nie możesz licytować własnego ogłoszenia.


Zabezpiecza to licytowanie swoich aukcji

Aplikacja posiada również walidację, która ma zapobiegać nadpisaniu niewłaściwymi danymi bazy danych. Obsługa podstawowych najczęściej spotykanych błędów na stronach internetowych:

 401.b

 403.b

 404.b

 405.b

 418.b

 500.b

500 Internal Server Error:

- Opis: Ten kod błędu sygnalizuje, że serwer napotkał niespodziewany błąd i nie jest w stanie zrealizować żądania.
- Przyczyny: Błąd może wynikać z różnych przyczyn, takich jak błędy konfiguracji serwera, błędy w kodzie aplikacji lub problem z bazą danych.

404 Not Found:

- Opis: Ten kod błędu oznacza, że serwer nie może znaleźć zasobu pod wskazanym adresem URL.
- Przyczyny: Może to być spowodowane tym, że zasób został usunięty, przeniesiony lub adres URL został źle wpisany.

418 I'm a teapot:

- Opis: Ten kod błędu jest humorystycznym kodem, który został zdefiniowany w dokumencie RFC 2324 dla protokołu kawowego. Oznacza to, że serwer nie jest w stanie zrealizować żądania, ponieważ jest urządzeniem typu "czajnik".

- Przyczyny: Ten kod błędu jest rzadko stosowany i zazwyczaj jest używany w celach testowych lub humorystycznych.

401 Unauthorized:

- Opis: Ten kod błędu oznacza, że serwer wymaga uwierzytelnienia użytkownika, ale użytkownik nie dostarczył odpowiednich danych uwierzytelniających lub dane uwierzytelniające są nieprawidłowe.
- Przyczyny: Użytkownik może próbować uzyskać dostęp do zasobu, który wymaga uwierzytelnienia, ale nie dostarczył odpowiednich danych uwierzytelniających lub dane są nieprawidłowe.

403 Forbidden:

- Opis: Ten kod błędu oznacza, że serwer odmawia spełnienia żądania, pomimo że jest prawidłowe. Oznacza to, że serwer rozpoznaje żądanie, ale nie zezwala na jego wykonanie.
- Przyczyny: To może być spowodowane ograniczeniami dostępu na serwerze, takimi jak błędna konfiguracja uprawnień dostępu do plików lub zasobów.

405 Method Not Allowed:

- Opis: Ten kod błędu sygnalizuje, że metoda żądania (np. GET, POST, PUT) nie jest obsługiwana przez zasób pod wskazanym adresem URL.
- Przyczyny: To może być spowodowane brakiem obsługi konkretnej metody HTTP przez serwer dla danego zasobu, na przykład próba wykonania metody POST na zasobie, który obsługuje tylko metody GET.