

**Politechnika Wrocławskas  
Wydział Informatyki i Telekomunikacji**

---

Kierunek: **ITE**

Specjalność: **IMT**

**PRACA DYPLOMOWA  
INŻYNIERSKA**

**Strona internetowa stajni**

**Szymon Hutnik**

Opiekun pracy  
**dr inż. Agata Kirjanów-Błażej**

Słowa kluczowe: Aplikacja webowa, React, Express

---

**WROCŁAW 2022**



# SKRÓTY

**REST** (ang. *Representational State Transfer*)

**API** (ang. *Application Programming Interface*)

**JSON** (ang. *JavaScript Object Notation*)

**JWT** (ang. *JSON Web Token*)

**HTTPS** (ang. *Hypertext Transfer Protocol Secure*)

**HTTP** (ang. *Hypertext Transfer Protocol*)

**HTML** (ang. *HyperText Markup Language*)

**JIT** (ang. *Just-in-Time (compilation)*)

**UI** (ang. *User Interface*)

**GUI** (ang. *Graphical User Interface*)

**URL** (ang. *Uniform Resource Locator*)

**URI** (ang. *Uniform Resource Identifier*)

# SPIS TREŚCI

<b>Skróty</b> . . . . .	1
<b>Wstęp</b> . . . . .	4
Wprowadzenie . . . . .	4
Cel pracy . . . . .	4
Struktura pracy . . . . .	4
<b>1. Wykorzystane technologie</b> . . . . .	5
1.1. JavaScript . . . . .	5
1.1.1. NodeJs . . . . .	5
1.1.2. Express . . . . .	5
1.1.3. React . . . . .	5
1.2. PostgreSQL . . . . .	5
1.3. Deta Drive . . . . .	6
1.4. Railway . . . . .	6
<b>2. Projekt aplikacji</b> . . . . .	7
2.1. Wymagania . . . . .	7
2.1.1. Wymagania funkcjonalne . . . . .	7
2.1.2. Wymagania niefunkcjonalne . . . . .	7
2.1.3. Diagram przypadków użycia . . . . .	8
2.2. Projekt bazy danych . . . . .	9
2.3. Projekt serwera . . . . .	10
2.4. Projekt strony internetowej . . . . .	11
2.5. Wdrożenie . . . . .	12
<b>3. Implementacja</b> . . . . .	13
3.1. Baza danych . . . . .	13
3.2. Serwer . . . . .	15
3.2.1. Routing . . . . .	15
3.2.2. REST API . . . . .	16
3.2.3. Modyfikacja danych . . . . .	17
3.2.4. Autentykacja i autoryzacja . . . . .	17
3.3. Strona internetowa . . . . .	19
3.3.1. Komponenty . . . . .	19
3.3.2. Komunikacja z serwerem . . . . .	19
3.3.3. Logowanie . . . . .	21
3.3.4. Edytory treści . . . . .	22
3.3.5. Widoki . . . . .	24
3.4. Wdrożenie projektu . . . . .	39
3.4.1. Uruchomienie lokalne . . . . .	39
3.4.2. Uruchomienie na platformie Railway . . . . .	40
<b>Podsumowanie</b> . . . . .	41
Podsumowanie pracy . . . . .	41

Dalsze możliwości rozwoju . . . . .	41
<b>Bibliografia</b> . . . . .	42
<b>Spis rysunków</b> . . . . .	43
<b>Spis listingów</b> . . . . .	44
<b>Dodatki</b> . . . . .	45
<b>A. Pendrive z kodem źródłowym</b> . . . . .	46

# **WSTĘP**

## **WPROWADZENIE**

Tematem pracy inżynierskiej jest strona internetowa stajni, którą właściciel obiektu, dalej zwany administratorem, będzie mógł zarządzać. W dzisiejszych czasach dostęp do informacji jest bardzo ułatwiony, ponad 90% gospodarstw domowych w Polsce ma dostęp do internetu [1]. Przekłada się to na wagę informacji dostępnych w sieci, jeżeli nie można ich znaleźć, potencjalny klient zazwyczaj wybiera inną firmę. Na decyzję o stworzeniu strony internetowej stajni wpłynął fakt jak niewiele obiektów tego typu ją posiada. Jako poczatkujący jeździec ciężko było mi wyszukać szczegółowe informacje w internecie, dlatego postanowiłem stworzyć taką aplikację.

## **CEL PRACY**

Celem pracy jest stworzenie nowoczesnej strony internetowej, która zachęci potencjalnych klientów do odwiedzenia stajni. Dane będą przechowywane w bazie danych i będą wysyłane oraz modyfikowane przy użyciu REST API. Strona internetowa ma pozwolić klientom na sprawdzenie informacji o stajni: koniach, trenerach, ofercie, cenach i informacjach kontaktowych, a także na przeglądanie galerii zdjęć. Ma również umożliwić właścielowi obiektu, administratorowi, możliwie najprostszy sposób edycji danych.

## **STRUKTURA PRACY**

Praca jest podzielona na: wstęp, 3 rozdziały i podsumowanie. Pierwszy rozdział opisuje wykorzystane technologie. W drugim opisano projekt aplikacji, podano wymagania oraz załączono diagramy mające ułatwić pracę. W trzecim rozdziale można przeczytać, jak zaimplementowano projekt, znajdują się tam fragmenty kodu oraz zrzuty ekranów z aplikacji, a także opis jak uruchomić projekt. Ostatnim rozdziałem jest podsumowanie.

# 1. WYKORZYSTANE TECHNOLOGIE

## 1.1. JAVASCRIPT

JavaScript jest lekkim, interpretowalnym, obiektowym, prototypowym językiem programowania z *first-class functions*. Stosowany do pisania skryptów, które najczęściej wykonują się jako część aplikacji webowej lub jako część serwera [2]. Menadżer paczek *npm* umożliwia dostęp do ogromnej ilości bibliotek i modułów, które można dodawać do swoich projektów [3]. Ilość gotowych rozwiązań znacznie przyspiesza czas tworzenia rozbudowanych aplikacji.

### 1.1.1. NodeJs

NodeJs jest asynchronicznymi, sterowanymi eventami środowiskiem Javascript, które pozwala na budowę skalujących się aplikacji webowych. Napędza je silnik V8 napisany w C++, którego używają również przeglądarki oparte na Chromium [4]. Stosuje nieblokującą architekturę oraz może przetwarzać wiele procesów jednocześnie, co jest bardzo ważne, gdy projektujemy serwer [5].

### 1.1.2. Express

Express jest szybkim, minimalistycznym webowym frameworkiem w NodeJs przystosowanym do tworzenia aplikacji webowych. Ogromna ilość wbudowanych lub łatwych do dodania modułów umożliwia tworzenie dopracowanych API. Jego minimalizm dostarcza tylko potrzebnych komponentów niezaśmiecających środowiska oraz nie zmniejszając wydajności NodeJs'a. Jego jakość potwierdza fakt, że stanowi bazę, na której zbudowane są inne frameworki [6].

### 1.1.3. React

React jest biblioteką napisaną przez Metę w języku Javascript, która pozwala na budowanie interaktywnego interfejsu użytkownika. Wyróżnia się użyciem niezależnych komponentów stworzonych przez dewelopera lub importowanych z innych bibliotek, każdy z nich zarządza własnym stanem oraz zwraca elementy napisane w html. React (re)renderuje komponenty, tylko gdy ich stany ulegną zmianie. Duże zmiany nastąpiły w 2018 roku, kiedy wprowadzono *React Hooks*. Nowa funkcjonalność pozwoliła na tworzenie komponentów bez pisania klas, kod stał się całkowicie funkcyjny, bardziej przejrzysty i przyjazny deweloperowi [7].

## 1.2. POSTGRESQL

PostgreSQL jest potężnym, open source'owym, obiektowym systemem bazodanowym. Jest jedną z najpopularniejszych baz danych, przekłada się to na dobrą dokumentację oraz duże wsparcie społeczności. Postgres wspiera klasyczny język SQL oraz pozwala na definiowanie własnych funkcji i typów danych [8]. Do pracy z bazą można używać dostarczonego przez twórców narzędzia psql, które uruchomione w terminalu zapewnia minimalistyczny interfejs.

Oprócz wykonywania zapytań można korzystać z dużej ilości meta-komend, które ułatwiają pracę [9].

### **1.3. DETA DRIVE**

Deta jest na zawsze darmową platformą udostępniającą 3 serwisy: Deta Base, Deta Drive i Deta Micros. W tym projekcie wykorzystany zostanie Deta Drive, czyli zarządzany, bezpieczny i skalujący się do 10GB system przechowywania plików. Dysk można obsługiwać poprzez przeglądarkowy interfejs lub za pomocą modułów w Pythonie lub JavaScript [10].

### **1.4. RAILWAY**

Railway jest platformą, na której można tworzyć i wdrażać infrastrukturę. Jest w tym podobny do Heroku jednak stosuje rozwiązania niewymagające od dewelopera dodatkowej wiedzy, aby móc uruchomić swój projekt. W kilku krokach można stworzyć bazę danych dostępną online, jak również bez dodatkowych konfiguracji uruchomić programy napisane w NodeJs. Dzięki zastosowaniu opłat na postawie zużycia oraz darmowego miesięcznego limitu utrzymanie infrastruktury może być całkowicie darmowe [11].

## **2. PROJEKT APLIKACJI**

### **2.1. WYMAGANIA**

Aplikacje można podzielić na części:

- baza danych,
- REST API,
- strona internetowa.

Razem tworzą cały system, dla którego sporządzono zbiór wymagań oraz diagram przypadków użycia przedstawiony na rysunku 2.1.

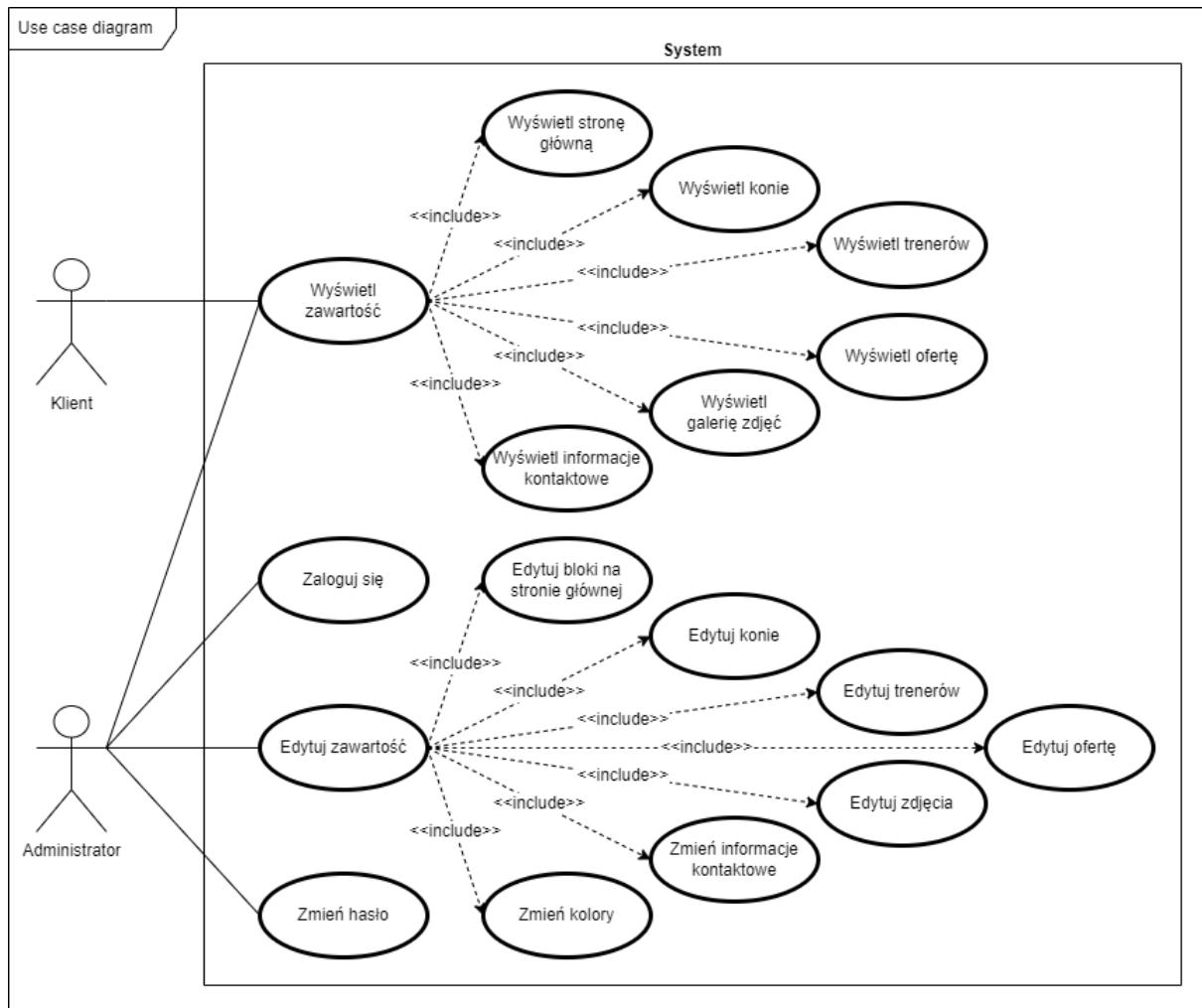
#### **2.1.1. Wymagania funkcjonalne**

1. Klient może wyświetlać stronę i jej podstrony.
2. Administrator może się zalogować hasłem.
3. Administrator może dodawać stronę główną, konie, trenerów, oferty i ceny w cenniku.
4. Administrator może modyfikować stronę główną, konie, trenerów, oferty i ceny w cenniku, dane kontaktowe i kolory strony.
5. Administrator może usuwać stronę główną, konie, trenerów, oferty i ceny w cenniku.
6. Każdy koń i trener mają zdjęcie profilowe.
7. Konie, trenerzy i oferty wyświetlają galerię zdjęć z nimi powiązanych.

#### **2.1.2. Wymagania niefunkcjonalne**

1. Każdy będzie mógł odwiedzić stronę internetową.
2. Strona oraz REST API będą działać pod jednym adresem.
3. Hasło będzie hashowane.
4. Baza danych jest backup'em, REST API nie powinno się z nią komunikować, jeżeli nie są wprowadzane modyfikacje.
5. Zdjęcia będą przechowywane w chmurze.

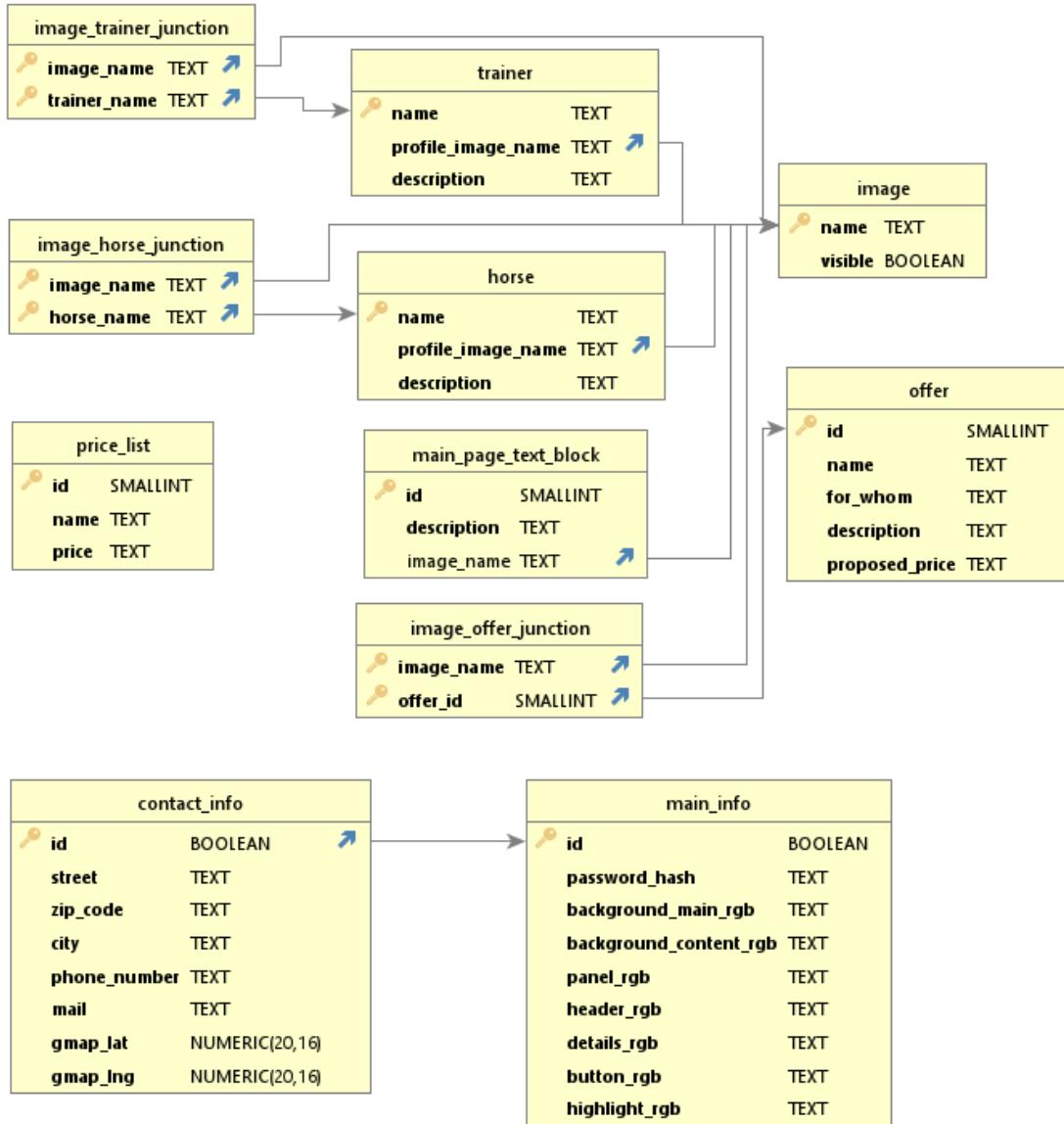
### 2.1.3. Diagram przypadków użycia



Rys. 2.1. Diagram przypadków użycia aplikacji

## 2.2. PROJEKT BAZY DANYCH

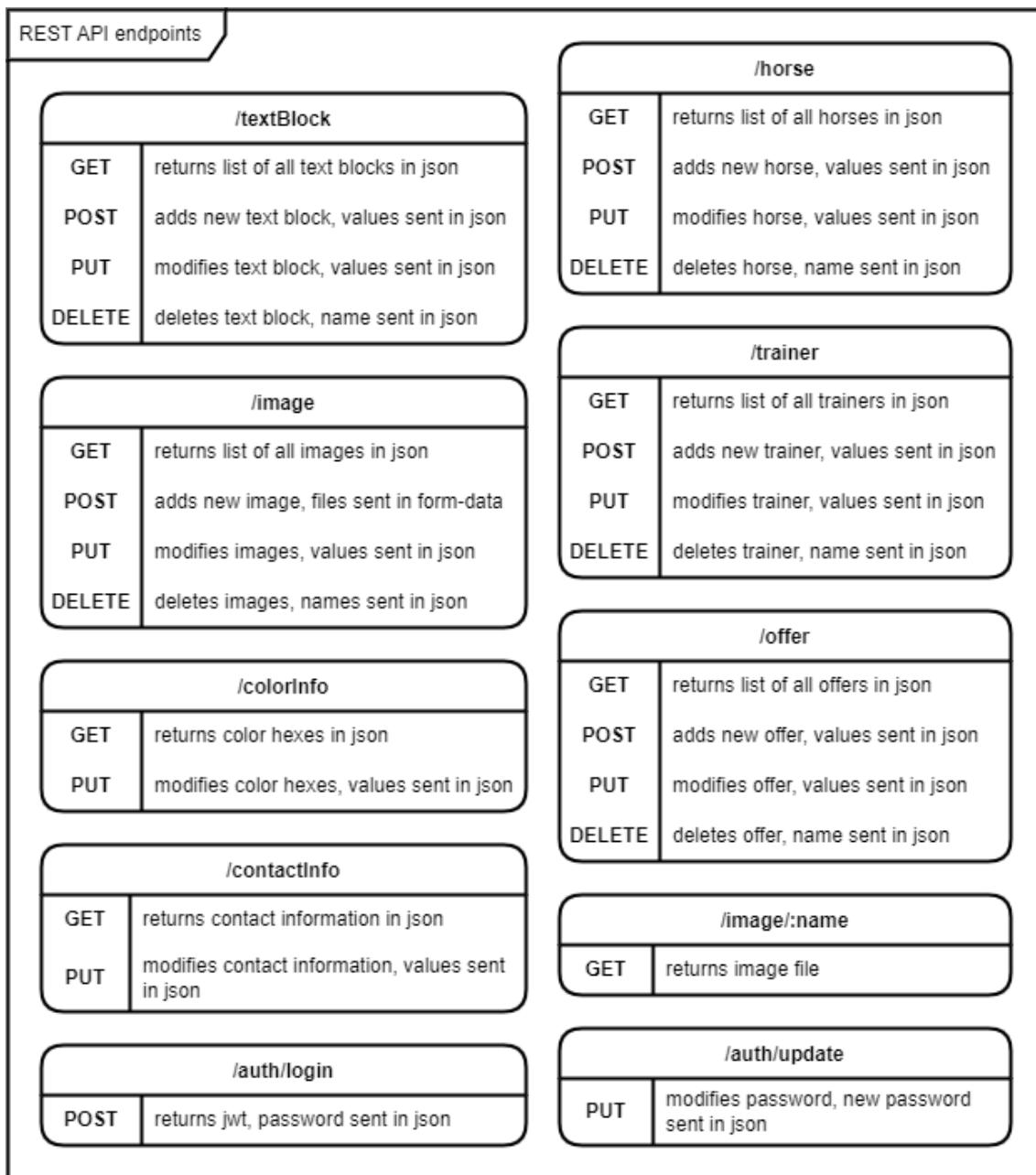
Baza danych będzie używana do przechowywania obiektów wyświetlanego na stronie internetowej. Obrazki będą przechowywane w Deta Drive, a ich dane trzymane w bazie danych. Jako system do zarządzania bazą danych wybrano PostgreSQL. Na rysunku 2.2 przedstawiono model bazy danych w formie diagramu klas. Dodatkowo zostaną przygotowane widoki, które będą umożliwiały łatwe pobranie wartości i zapisanie do formatu JSON.



Rys. 2.2. Diagram klas bazy danych

## 2.3. PROJEKT SERWERA

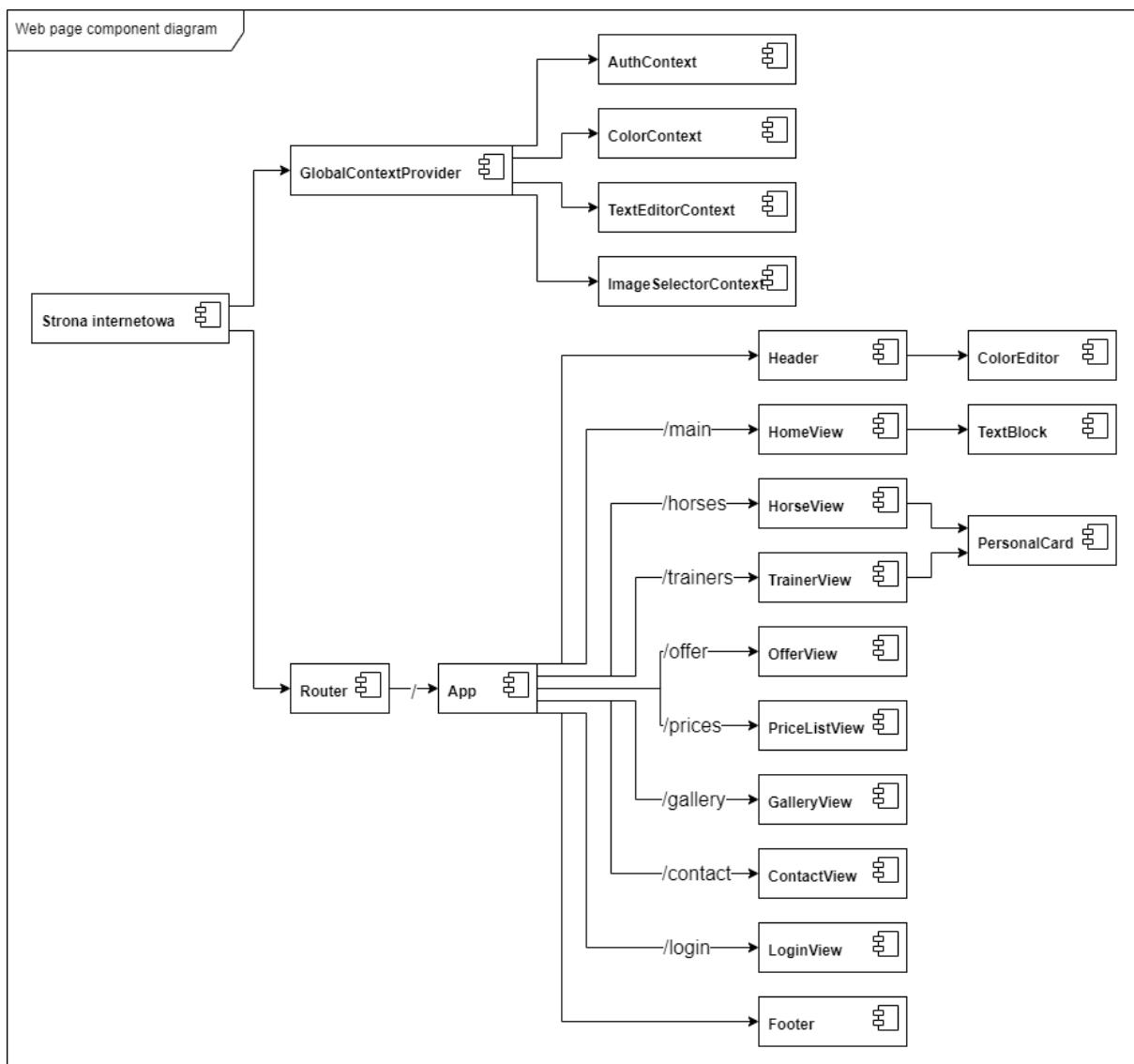
Serwer będzie wykonywać 2 czynności: serwować REST API oraz stronę internetową. REST API będzie wysyłać dane stronie internetowej oraz na żądanie modyfikować bazę danych. W trakcie działania serwer będzie wykorzystywał wartości, które pobrał w trakcie uruchamiania. Interakcje z bazą danych będą się odbywały, tylko gdy administrator zmodyfikuje informacje wyświetlane na stronie. Po zapisaniu zmian serwer zaktualizuje przechowywane wartości, aby nie musieć odpytywać bazy danych przy każdym żądaniu. Do jego stworzenia zostanie wykorzystany NodeJs oraz framework Express. Na rysunku 2.3 przedstawiono mapę endpointów REST API, znajdujących się pod ścieżką /api.



Rys. 2.3. Mapa endpointów REST API

## 2.4. PROJEKT STRONY INTERNETOWEJ

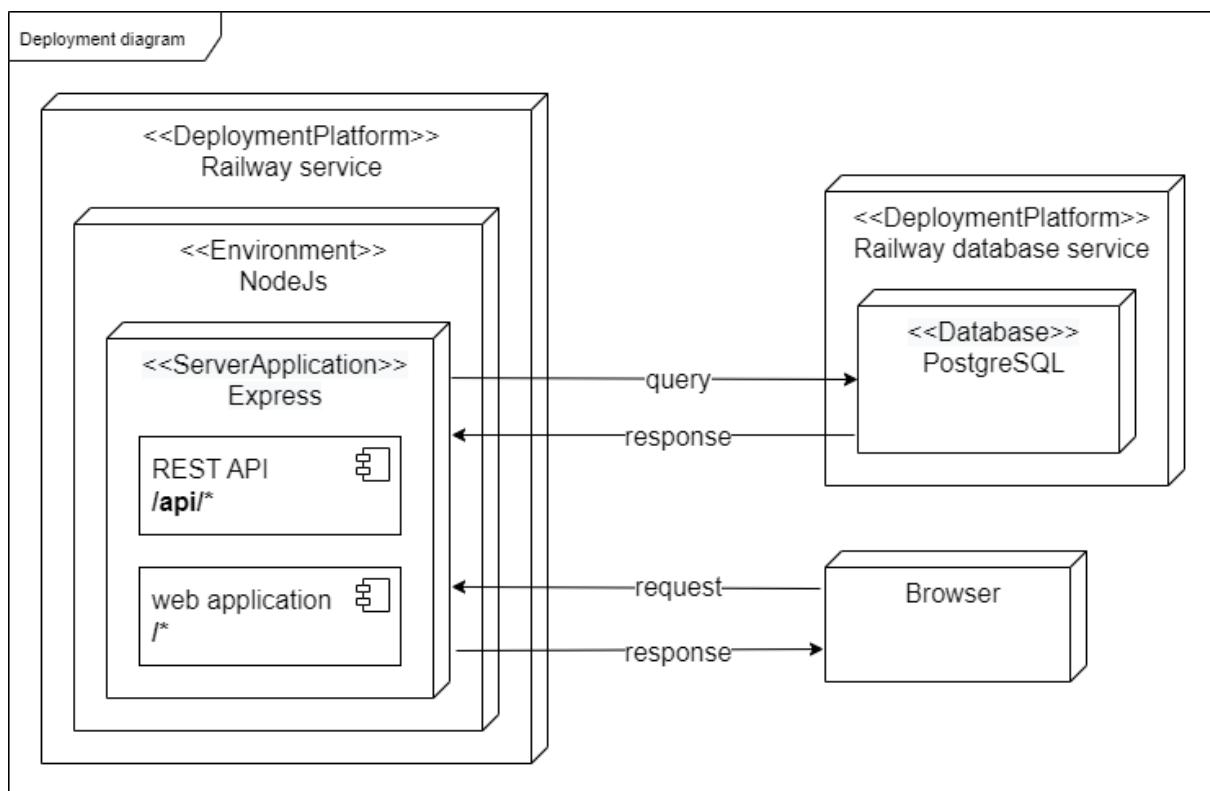
Strona internetowa jest główną częścią projektu, umożliwia klientowi przeglądanie informacji o stajni, a administratorowi zarządzanie jej zawartością. Dzięki implementacji interaktywnego interfejsu do zarządzania zasobami administrator strony nie będzie musiał posiadać żadnej wiedzy technicznej, aby wprowadzać zmiany. Strona zostanie napisana we framework'u React. Na rysunku 2.4 znajduje się diagram komponentów, który przedstawia podział strony na logiczne części. Każdy z nich odpowiada za inną funkcjonalność, posiada własną logikę i stany.



Rys. 2.4. Diagram komponentów strony internetowej

## 2.5. WDROŻENIE

Do wdrożenia aplikacji będzie potrzebna platforma, która pozwoli na hostowanie serwera oraz bazy danych. W tym projekcie zdecydowałem się na platformę Railway, ponieważ jest nowoczesnym, nieskomplikowanym i w pewnym zakresie darmowym rozwiązaniem. Na platformie zostaną stworzone 2 kontenery, w pierwszym będzie postawiona baza danych, a w drugim serwer. REST API znajdzie się pod adresem `/api/*`, a strona internetowa będzie wyświetlana dla pozostałych ścieżek. Plan wdrożenia przedstawia diagram wdrożenia na rysunku 2.5.



Rys. 2.5. Diagram wdrożenia aplikacji

# 3. IMPLEMENTACJA

## 3.1. BAZA DANYCH

Baza danych zgodna z zaprojektowanym modelem przedstawionym na rysunku 2.2 została zapisana w formie skryptu w języku SQL. Będzie no służyć do zainicjalizowania bazy, stworzy tabele, widoki i doda po rekordzie do tabel *main\_info*, *contact\_info* oraz *image*. Tabele *main\_info* i *contact\_info* są ograniczone do 1 rekordu i zawierają podstawowe informacje o kolorach strony oraz stajni. Pierwszy rekord w tabeli *image* zawiera dane o obrazku, który ma zostać wyświetlony w razie gdy inny obrazek nie zostanie znaleziony, nie można go zmienić. Poniżej znajdują się fragmenty skryptu inicjalizującego: listing 3.1 pokazuje tworzenie tabel, listing 3.2 pokazuje dodanie kluczy obcych do tabel, listing 3.3 pokazuje tworzenie widoków.

---

```
CREATE TABLE image (
    name          text NOT NULL,
    visible       boolean NOT NULL,
    CONSTRAINT pk_image PRIMARY KEY ( name )
);

CREATE TABLE horse (
    name          text NOT NULL,
    profile_image_name text DEFAULT 'dummyImage.jpg' NOT NULL,
    description    text NOT NULL,
    CONSTRAINT pk_horse PRIMARY KEY ( name )
);

CREATE TABLE image_horse_junction (
    image_name     text NOT NULL,
    horse_name     text NOT NULL,
    CONSTRAINT pk_image_horse_junction PRIMARY KEY ( image_name, horse_name
)
);
```

---

Listing 3.1: Tworzenie wybranych tabel: *image*, *horse*, *image\_horse\_junction*, fragment dbSchema.sql

---

```
ALTER TABLE horse ADD CONSTRAINT fk_horse_image FOREIGN KEY ( profile_image_name
↪ ) REFERENCES image( name ) ON DELETE SET DEFAULT ON UPDATE SET DEFAULT;

ALTER TABLE image_horse_junction ADD CONSTRAINT fk_image_horse_junction_1
↪ FOREIGN KEY ( image_name ) REFERENCES image( name ) ON DELETE CASCADE ON
↪ UPDATE CASCADE;

ALTER TABLE image_horse_junction ADD CONSTRAINT fk_image_horse_junction_2
↪ FOREIGN KEY ( horse_name ) REFERENCES horse( name ) ON DELETE CASCADE ON
↪ UPDATE CASCADE;
```

---

Listing 3.2: Dodawanie kluczy obcych do wybranych tabelach: *horse*, *image\_horse\_junction*, fragment dbSchema.sql

---

```
CREATE VIEW image_list_view AS
SELECT name as image,
       visible
FROM image
WHERE name != 'dummyImage.jpg';

CREATE VIEW horse_list_view AS
SELECT horse.name,
       horse.description,
       array_remove(array_prepend(horse.profile_image_name,
↪ array_agg(image_horse_junction.image_name)), NULL) as images
FROM horse
LEFT JOIN image_horse_junction ON horse.name = image_horse_junction.horse_name
GROUP BY name;
```

---

Listing 3.3: Tworzenie wybranych widoków: *image\_list\_view*, *horse\_list\_view*, fragment dbSchema.sql

## 3.2. SERWER

Serwer został napisany we framework'u Express w środowisku NodeJs. Oprócz bibliotek pobieranych przez **express-generator** podczas tworzenia podstawowego środowiska w projekcie znalazły się również:

- bcryptjs – biblioteka służąca do generowania i testowania hashy [12], wykorzystana do weryfikacji i aktualizacji hasła,
- deta – klasa służąca do komunikacji z serwisami Deta [13], wykorzystana do zapisu i odczytu obrazków z Deta Drive,
- express-fileupload – paczka dostarczająca middleware umożliwiający pobieranie plików [14],
- jsonwebtoken – paczka dostarczająca implementację JWT, służy do tworzenia i porównywania tokenów [15], wykorzystana w procesie logowania i autentykacji administratora,
- pg – kolekcja modułów umożliwiająca komunikację z bazą danych PostgreSQL [16],
- eslint – narzędzie do statycznej analizy kodu [17], pomaga utrzymać czysty kod i znaleźć potencjalne błędy.

### 3.2.1. Routing

Poprawnie działający serwer musi obsługiwać wszystkie endpointy zamodelowane na rysunku 2.3, a także serwować stronę internetową i zdjęcia. Aby spełnić dwa ostatnie wymagania zastosowano funkcje **static()**, listing 3.4. Pozwala ona na serwowanie plików statycznych znajdujących się w podanym katalogu oraz jego podfolderach. Serwowanie dzieje się automatycznie, jeśli plik na ścieżce podanej w url znajduje się w jednym z katalogów. Jeżeli nie zostanie on znaleziony, można przechwycić takie żądanie, aby przetworzyć je w dowolny sposób, na przykład przekierować je do właściwego miejsca, listing 3.5, lub wysłać odpowiedni plik bezpośrednio, listing 3.6.

---

```
app.use(express.static(path.join(__dirname, "public")));
app.use(express.static(path.join(__dirname, "/frontend/build")));
```

---

Listing 3.4: Express static routing, fragment src/app.js

---

```
router.get("/image/:name", (req, res) => {
  res.redirect(databaseConnector.DUMMY_IMAGE_PATH);
});
```

---

Listing 3.5: Express redirect, fragment src/routes/api.js

---

```
router.get("/*", (req, res) => {
  res.sendFile(path.join(__dirname, "/frontend/build/index.html"));
});
```

---

Listing 3.6: Express sendFile, fragment src/routes/index.js

Dzięki zastosowaniu Router'ów można rozdzielić zarządzanie ścieżkami na pliki, każdy router będzie zarządzał jedną z nich, podział zastosowany w projekcie widać na listingu 3.7.

---

```
const indexRouter = require("./routes/index");
const apiRouter = require("./routes/api");

app.use("/api", apiRouter);
app.use("/", indexRouter);
```

---

Listing 3.7: Express routing, fragment src/app.js

### 3.2.2. REST API

REST API będzie służyło do wysyłania danych o stajni stronie internetowej. Jest to most pomiędzy bazą danych, a GUI. Składać się będzie z pliku prototypu zarządzającego routingu, serwujące endpointy opisane na rysunku 2.3, i modułu odpowiadającego za łączenie się z bazami danych i przechowywanie informacji o stajni. Po otrzymaniu żądania na jeden z endpointów w przypadku metody GET zwracany obrazek lub plik json z odpowiednimi wartościami. Jeżeli metoda była inna, następuje weryfikacja poprawności przesłanych danych, następnie wywoływana jest odpowiednia funkcja z modułu komunikującego się z bazą danych, która wprowadza zmiany do bazy danych, a następnie aktualizuje przechowywane informacje. Przykładowo wysłanie żądania z metodą PUT na adres "/api/horse" wywoła funkcję pokazaną na listingu 3.8.

---

```
router.put("/horse", (req, res) => {
  const updatedHorse = req.body;
  if (
    !updatedHorse ||
    !updatedHorse.name ||
    !updatedHorse.description ||
    !updatedHorse.images ||
    updatedHorse.images.length === 0
  )
    return res.status(406).json("Mandatory fields not set");

  if ((err = checkName(updatedHorse.name, databaseConnector.getHorseList(),
    "Horse"))) return res.status(406).json(err);
  if ((err = checkImages(updatedHorse.images))) return
    res.status(406).json(err);

  databaseConnector.updateHorse(updatedHorse).then((err) => {
    return err ? res.status(500).json(err) : res.sendStatus(200);
  });
});
```

---

Listing 3.8: Express routing, fragment src/routes/api.js

### 3.2.3. Modyfikacja danych

Połączenie z bazą danych jest zrealizowane za pomocą klasy **Pool()** z kolekcji pg. Operacje na bazie danych są wykonywane poprzez wywołanie na niej funkcji **query()**, której podaje się zapytanie oraz listę zmienionych wartości [16]. Przykład funkcji dodającej rekord do tabeli *horse* w bazie danych można zobaczyć na listingu 3.9. Pobieranie i usuwanie zdjęć w Deta Drive odbywa się za pomocą modułu Deta, który łączy się z projektem za pomocą klucza projektu, znajdziemy go w panelu projektu na stronie <https://web.deta.sh/>. Następnie subklasa reprezentującą połączenie z wybranym dyskiem jest inicjalizowana [13], proces łączenia widać na listingu 3.10. Na zainicjalizowanej subklasie można wywoływać funkcje pozwalające modyfikację plików zapisanych na dysku.

---

```
const createHorse = async (newHorse) => {
  try {
    await pool.query("INSERT INTO horse VALUES ($1, $2, $3)", [newHorse.name,
      newHorse.image, newHorse.description]);
    return updateFromDatabase();
  } catch (err) {
    console.error(err);
    return INTERNAL_SERVER_ERROR_MESSAGE;
  }
};
```

---

Listing 3.9: Dodanie rekordu do tabeli *horse* w Javascript, fragment src/databaseConnector.js

---

```
const { Deta } = require("deta");
const detaIntance = Deta(process.env.DETA_KEY);
const detaImageDrive = detaIntance.Drive("images");
```

---

Listing 3.10: Inicjalizacja połączenia z Deta Drive *images*, fragment src/databaseConnector.js

### 3.2.4. Autentykacja i autoryzacja

Administrator strony musi mieć możliwość zalogowania się i modyfikacji treści, opcje te są zablokowane dla zwykłego użytkownika. Wymaga to zaimplementowania modułu autentykującego oraz autoryzującego użytkownika. Logowanie polega na wysłaniu na adres **/api/auth/login** żądania POST z hasłem, jeśli jest ono zgodne z hasłem przechowywanym w bazie danych serwer tworzy nowy jwt i wysyła go w odpowiedzi [18]; token jest ważny przez godzinę, listing 3.11. Następnie, gdy użytkownik chce zmodyfikować dane strony, musi przejść weryfikację. W tym celu żądanie najpierw przechodzi przez autoryzator, jeżeli token jest poprawny, żądanie przechodzi do następnej funkcji, listing 3.12.

---

```
router.post("/login", async (req, res) => {
  try {
    if (!req.body || !req.body.password) return res.status(406).json("Password
      ↵ not sent");
    const password = req.body.password;

    const passwordIsValid = bcrypt.compareSync(password, await
      ↵ databaseConnector.getPassword());
    if (!passwordIsValid) return res.status(401).json("Invalid Password!");

    const token = jwt.sign({}, process.env.PRIVATE_KEY, {
      expiresIn: EXPIRATION_TIME,
    });
    return res.status(200).json({
      accessToken: token,
      expiresIn: EXPIRATION_TIME,
    });
  } catch (err) {
    console.log(err);
    return res.status(500).json("Internal server error, contact maintainer");
  }
});
```

---

Listing 3.11: Autentykacja, fragment src/routes/authenticator.js

---

```
const verifyToken = (req, res, next) => {
  const token = req.headers["x-access-token"];

  if (!token) {
    return res.status(403).json("No token provided!");
  }

  jwt.verify(token, process.env.PRIVATE_KEY, (err) => {
    if (err) {
      return res.status(401).json("Unauthorized!");
    }
    next();
  });
};
```

---

Listing 3.12: Autoryzacja, fragment src/middleware/authorizer.js

### 3.3. STRONA INTERNETOWA

Strona internetowa została napisana we framework'u React. Oprócz bibliotek pobieranych przez **create-react-app** podczas tworzenia podstawowego środowiska w projekcie znalazły się również: primeflex, primeicons, primereact – ta kolekcja pozwala na użycie gotowych komponentów z biblioteki PrimeReact [19].

#### 3.3.1. Komponenty

Aplikacja została podzielona na zbiory komponentów, każdy z nich pełni inne funkcje:

- komponenty bez zbioru – w projekcie są 2 komponenty, które nie należą do żadnego zbioru: *src/index.js*, *src/App.js*. Stanowią one podstawę aplikacji, nie wykonują konkretnych funkcji, a jedynie renderują szkielet strony, który jest uzupełniany innymi komponentami.
- *views* (widoki) – każdy z nich jest innym widokiem, zakładką. Może być kompletnym elementem lub używać komponentów ze zbiorów layouts oraz components.
- *layouts* (układy) – są to fragmenty układu strony, które się powtarzają między widokami jak nagłówek.
- *contextProviders* – są to specjalne komponenty, które są dodawane renderowane przed resztą elementów. Zapewniają kontekst, który umożliwia wykorzystanie wartości i funkcji z tych komponentów. W tym projekcie do tego zbioru będą należeć komponenty obsługujące logowanie, kolory, edycję tekstu i obrazków.
- *components* (komponenty) – do tego zbioru trafiają komponenty, które nie mają specjalnego zastosowania.

#### 3.3.2. Komunikacja z serwerem

Aby strona mogła działać prawidłowo, musi pobierać dane z serwera. Do tego celu wykorzystano prostą funkcję **fetch()**, domyślną metodą jest GET. Po otrzymaniu odpowiedzi jest ona przetwarzana i jeśli status jest "ok" zmieniony zostaje stan komponentu, przykład użycia na listingu 3.13.

---

```
const fetchHorseList = () => {
  fetch(API_URL + "horse")
    .then((response) => checkResponseOk(response))
    .then((response) => setHorseList(response))
    .catch((err) => {
      console.error(`Server response: ${err}`);
    });
};
```

---

Listing 3.13: Przykład funkcji `fetch()`, fragment *src/views/HorseView/HorseView.js*

Aktualizowanie danych przebiega podobnie, jednak użytkownik musi być wcześniej zalogowany, aby uzyskać uprawnienia do edycji. Dlatego też funkcja aktualizująca została umieszczona w *contextProvider* odpowiedzialnym za logowanie. Przyjmuje w parametrach: adres url, metodę, obiekt, który ma zostać wysłany, callback, który ma zostać wywołany po udanej aktualizacji. Do żądania zostaje dołączony token, który umożliwia autoryzację. Funkcja wyświetla również komunikaty informujące użytkownika o jej wyniku, listing 3.14.

---

```
const performDataUpdate = (url, method, body, callback) => {
  const authToken = localStorage.getItem("authToken");
  if (!authToken) {
    toast.current.show({ severity: "error", summary: "Token lost", detail: "Nie
      ↵ znaleziono tokena, wylogowano!", life: 10000, });
    logoutUser();
    return;
  }
  fetch( API_URL + url, url === "image" && method === "POST"
    ? { method: "POST", headers: { "x-access-token": JSON.parse(authToken) },
      ↵ body: body, }
    : { method: method, headers: { "x-access-token": JSON.parse(authToken),
      ↵ "Content-Type": "application/json" }, body: JSON.stringify(body), }
  )
  .then((response) => checkResponseOk(response))
  .then(() => {
    callback();
    toast.current.show({ severity: "success", summary: "Sukces", detail:
      ↵ "Zmiany zostały zapisane", life: 2000 });
  })
  .catch((err) => {
    console.error(`Server response: ${err}`);
    if (err === "No token provided!") {
      toast.current.show({ severity: "error", summary: "Błąd", detail:
        ↵ `${err}`, token zgubiony, zostałeś wylogowany!`, life: 6000, });
      logoutUser();
      return;
    }
    if (err === "Unauthorized!") {
      toast.current.show({ severity: "warn", summary: "Brak uprawnień",
        ↵ detail: "Nie masz uprawnień, aby modyfikować ten zasób!", life:
        ↵ 6000, });
      return;
    }
    toast.current.show({ severity: "error", summary: "Błąd", detail: `Błąd
      ↵ serwera: ${err}`, zmiany nie zostały zapisane`, life: 6000, });
  });
};
```

---

Listing 3.14: Funkcja wysyłająca żądanie aktualizacji danych, fragment  
src/contextProviders/AuthContextProvider/AuthContextProvider.js

### 3.3.3. Logowanie

W celu zalogowania administrator musi wpisać hasło, następnie jest ono wysyłane do REST API w celu autentykacji. Jeżeli jest ono poprawne, w odpowiedzi otrzymuje token służący do potwierdzania tożsamości, gdy wprowadzane są zmiany danych. Token i czas jego ważności jest zapisywany w **localStorage**, które pozwala przechowywać wartości, nawet gdy strona jest odświeżana, dzięki temu nie trzeba się niepotrzebnie logować. Administrator może się wylogować, kiedy zechce lub zostanie wylogowany automatycznie, gdy czas ważności tokena upłynie. Na listingu 3.15 znajduje się funkcja logująca, a na listingu 3.16 wylogowująca.

---

```
const loginUser = (password) => {
  fetch(API_URL + "auth/login", { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify({ password: password }) })
    .then((response) => checkResponseOk(response))
    .then((response) => {
      localStorage.setItem("authToken", JSON.stringify(response.accessToken));
      localStorage.setItem("expirationTime", Date.now() + response.expiresIn - BUFFER_EXPIRATION_TIME);

      setAuthContext((prevState) => {
        prevState.isLoggedIn = true;
        return { ...prevState };
      });
      navigate("/");
    })
    .catch((err) => {
      console.error(`Server response: ${err}`);
      toast.current.show({ severity: "error", summary: "Błąd", detail: "Próba logowania nie powiodła się", life: 10000, });
    });
};
```

---

Listing 3.15: Funkcja logująca, fragment  
src/contextProviders/AuthContextProvider/AuthContextProvider.js

---

```

() => {
  console.warn("logout");
  localStorage.removeItem("authToken");
  localStorage.removeItem("expirationTime");

  setAuthContext((prevState) => {
    prevState.isLoggedIn = false;
    return { ...prevState };
  });
  navigate("/");
}

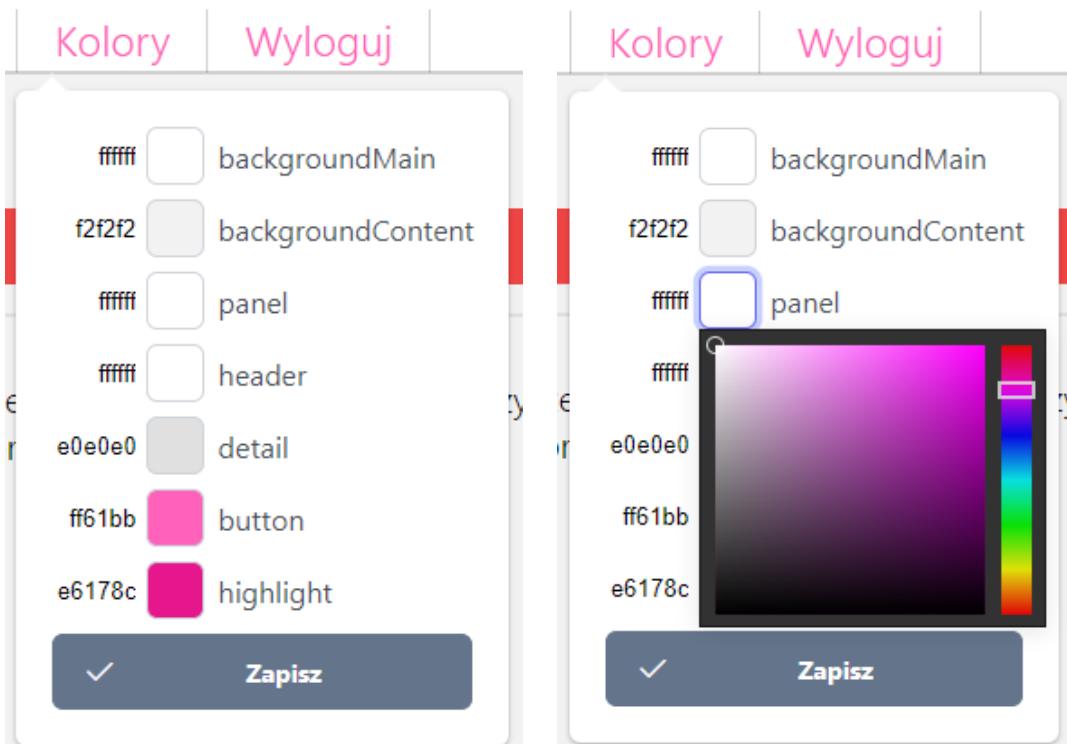
```

---

Listing 3.16: Funkcja wylogowująca, fragment  
src/contextProviders/AuthContextProvider/AuthContextProvider.js

### 3.3.4. Edytory treści

Administrator ma możliwość edycji treści wyświetlanej na stronie. Dokonuje tego za pomocą 3 edytorów: koloru, zdjęć i tekstu. Pierwszy z nich jest panelem, który pojawia się po kliknięciu przycisku w nagłówku, wyświetla listę kolorów, które mogą zostać zmienione, rysunek 3.1. Przy każdym z nich jest wyświetlony kod hex oraz podgląd nowego koloru. Edycji można dokonać poprzez ręczne wpisanie nowej wartości hex lub panel wybierania koloru, który otwiera się po kliknięciu kwadracika z podglądem, rysunek 3.2. Zapisanie zmian następuje dopiero po kliknięciu przycisku "Zapisz", zamknięcie panelu powoduje utratę wprowadzonych zmian.



Rys. 3.1. Edytor kolorów

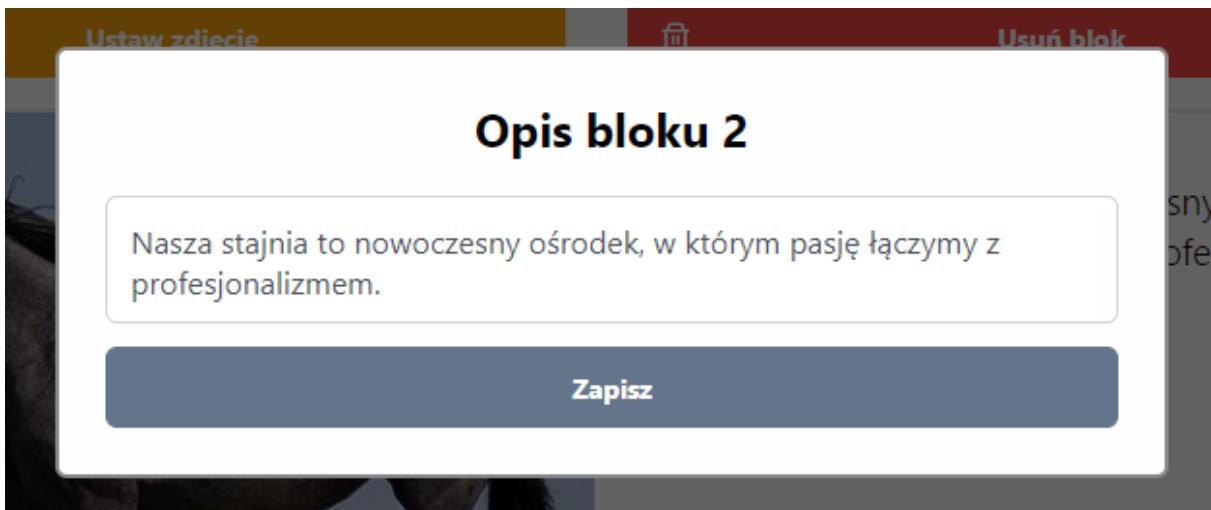
Rys. 3.2. Edytor kolorów, panel wyboru koloru

Komponent pozwalający na wybór obrazków również jest modelem. Pozwalać na wybranie kilku lub jednego zdjęcia, można podać maksymalnie jeden obrazek, który nie może zostać oznaczony. Zaznaczenia dokonuje się poprzez kliknięcie obrazka, zaznaczone zdjęcia mają szary filtr z gwiazdką. Wybrane obrazki należy zatwierdzić klikając przycisk "Zapisz", wywoła to funkcję, która zapisze wprowadzone zmiany, naciśnięcie szarego obszaru powoduje odrzucenie zmian.



Rys. 3.3. Wybieranie obrazków

Edytor tekstu jest modelem, który zostaje wywołany, gdy administrator modyfikuje lub wprowadza nowy tekst, rysunek 3.4. Nową wartość należy zatwierdzić kliknięciem przycisku, wywoła to funkcję, która zapisze wprowadzone zmiany, naciśnięcie szarego obszaru powoduje odrzucenie zmian.



Rys. 3.4. Edytor tekstu

### 3.3.5. Widoki

Strona internetowa będzie podzielona na 8 widoków, które będą różnić się w zależności, czy użytkownikiem jest klient, czy zalogowanym administratorem. Widoki klienta będą pokazywać stronę w formacie "read-only". Administratorowi będą wyświetlane dodatkowe przyciski służące do wprowadzania zmian w treści za pomocą edytorów omówionych w sekcji 3.3.4. Podstawowy układ strony składa się z 3 części: nagłówka, treści oraz stopki. Nagłówek zawiera logo oraz przyciski służące do nawigacji po stronie; po zalogowaniu powiększa się o 2 przyciski: do zmiany kolorów i do wylogowania. Nagłówek przed zalogowaniem pokazano na rysunku 3.5, a po zalogowaniu na rysunku 3.6. Stopka zawiera informację o prawach autorskich. Strona jest dostosowana urządzeń mobilnych, jednak zaleca się korzystanie z ekranów o standardowych rozdzielczościach w szczególności po zalogowaniu.



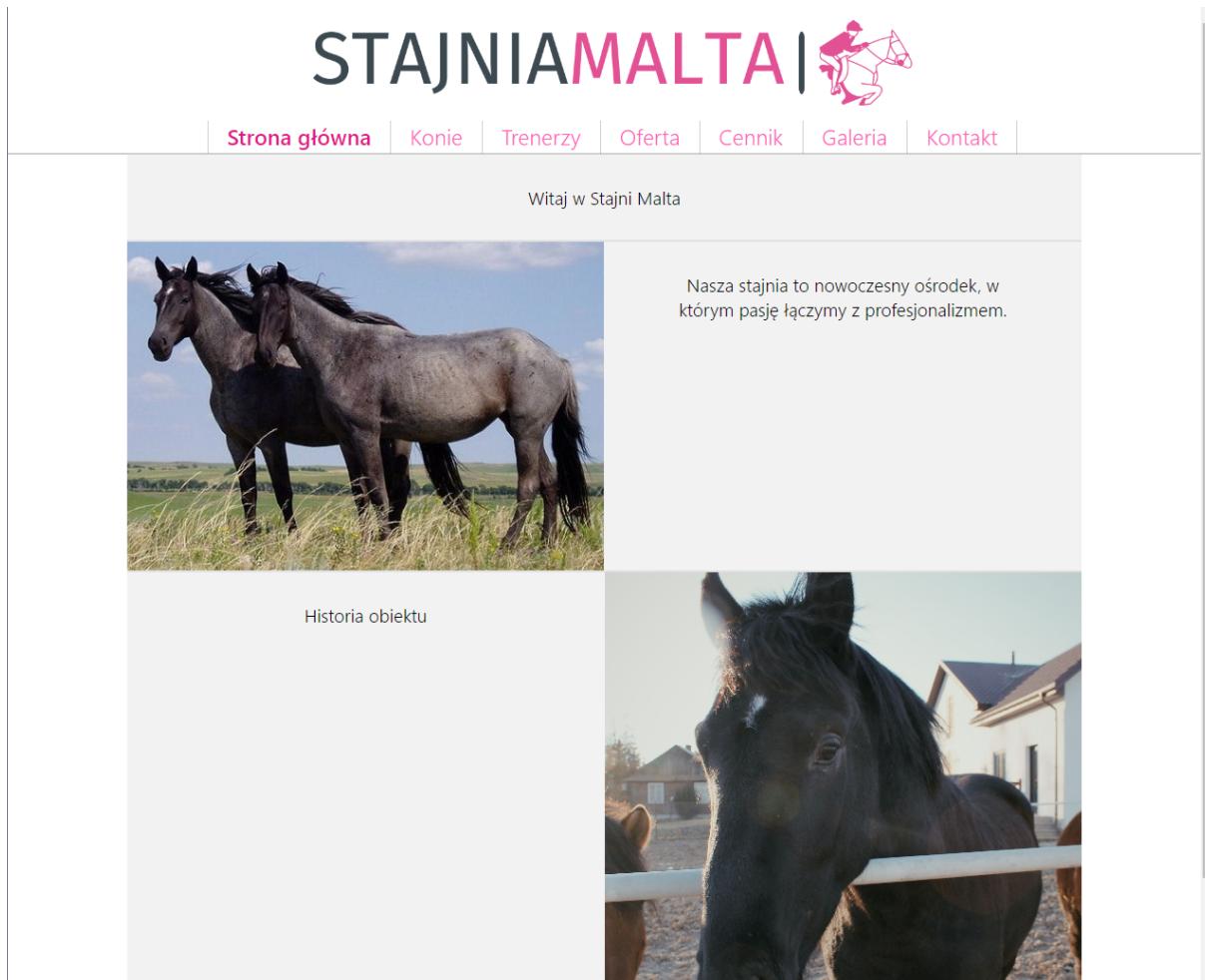
Rys. 3.5. Nagłówek klienta



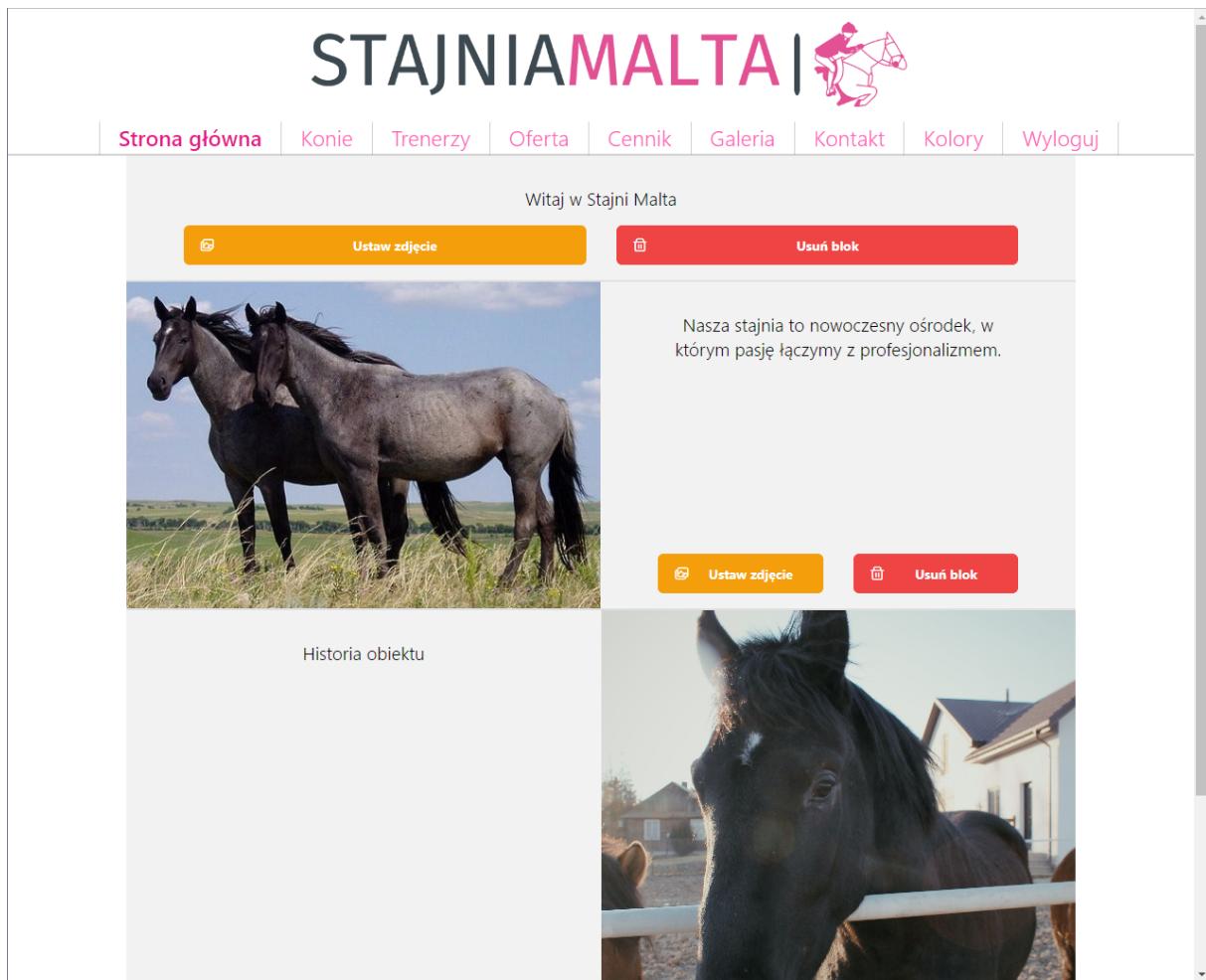
Rys. 3.6. Nagłówek administratora

### 3.3.5.1. Widok główny

Widok główny (domowy) jest zlokalizowany pod adresem /main, wszystkie adresy, które nie prowadzą do innych widoków, będą do niego przekierowywanie. Składa się z bloków z tekstem lub z tekstem i obrazem. Jego celem jest autoprezentacja i przekazanie informacji o stajni, opowiadzenie historii lub kilku ciekawostek o obiekcie. Administrator może kliknąć tekst, aby otworzyć edytor tekstu. Przykładowy wygląd widoku z perspektywy klienta, rysunek 3.7, oraz z perspektywy administratora, rysunek 3.8.



Rys. 3.7. Przykładowy wygląd widoku głównego z perspektywy klienta



Rys. 3.8. Przykładowy wygląd widoku głównego z perspektywy administratora

### 3.3.5.2. Widok koni

Widok koni jest zlokalizowany pod adresem `/horses`. Składa się z paneli z galerią obrazków oraz imieniem i opisem konia. Tutaj znajdują się informacje o koniach w stajni, a także galeria zdjęć związanych z każdym z nich. Administrator może edytować opis, profilowe oraz zdjęcia powiązane z koniem. Przykładowy wygląd widoku z perspektywy klienta, rysunek 3.9, oraz z perspektywy administratora, rysunek 3.10.

The screenshot shows a web page titled "STAJNIA MALTA" with a logo of a person riding a horse. Below the title is a navigation bar with links: "Strona główna", "Konie" (highlighted in pink), "Trenerzy", "Oferta", "Cennik", "Galeria", and "Kontakt". The main content area displays two horses. On the left is a dark brown horse named "Fryderyk" with a small "Edit" icon. On the right is a reddish-brown horse named "Malta" with a larger "Edit" icon. Both cards include a thumbnail image of the horse and a brief description below it.

Rys. 3.9. Przykładowy wygląd widoku koni z perspektywy klienta

**STAJNIA MALTA** | 

Strona główna Konie Trenerzy Oferta Cennik Galeria Kontakt Kolory Wyloguj



**Fryderyk**

Opis

 Ustaw zdjęcie profilowe

 Ustaw zdjęcia

 Usuń



**Malta**

Malta jest 10 letnim doświadczonym koniem. Jako oczko w głowie właścicielki jest bardzo zadbane oraz wytrenowana. Doskonały koń dla każdego jeźdźca

 Ustaw zdjęcie profilowe

 Ustaw zdjęcia

 Usuń

**Dodaj nowego konia**

© 2022 Stajnia Malta. All rights reserved

Rys. 3.10. Przykładowy wygląd widoku koni z perspektywy administratora

### 3.3.5.3. Widok trenerów

Widok trenerów jest zlokalizowany pod adresem [/trainers](#). Jest on analogiczny do widoku koni, ponieważ stosuje ten sam komponent do wyświetlania informacji. Przykładowy wygląd widoku z perspektywy klienta, rysunek 3.11, oraz z perspektywy administratora, rysunek 3.12.

The screenshot shows a website for 'STAJNIA MALTA'. The header features the logo 'STAJNIA MALTA' with a small horse icon next to it. Below the logo is a navigation bar with links: 'Strona główna', 'Konie', 'Trenerzy', 'Oferta', 'Cennik', 'Galeria', and 'Kontakt'. The main content area displays a card for a trainer named 'Szymon'. The card contains a small thumbnail image of Szymon wearing a fur-trimmed hood, followed by his name 'Szymon' in bold, and a short bio: 'Szymon jest naszym najbardziej doświadczonym trenerem. Jego umiejętności potwierdzają liczne nagrody zdobyte w zawodach.' At the bottom right of the page, there is a small copyright notice: '© 2022 Stajnia Malta. All rights reserved.'

Rys. 3.11. Przykładowy wygląd widoku trenerów z perspektywy klienta

# STAJNIA MALTA |

Strona główna | Konie | Trenerzy | Oferta | Cennik | Galeria | Kontakt | Kolory | Wyloguj



**Szymon**

Szymon jest naszym najbardziej doświadczonym trenerem. Jego umiejętności potwierdzają liczne nagrody zdobyte w zawodach.

[Dodaj nowego trenera](#)



[Ustaw zdjęcie profilowe](#)



[Ustaw zdjęcia](#)



[Usuń](#)

© 2022 Stajnia Malta. All rights reserved

Rys. 3.12. Przykładowy wygląd widoku trenerów z perspektywy administratora

### 3.3.5.4. Widok ofert

Widok ofert jest zlokalizowany pod adresem **/offer**. Jego celem jest zachęcenie potencjalnego klienta do skorzystania z jednej z usług oferowanych przez stajnię. Widok jest złożony z harmonijki, można rozwinąć maksymalnie jedną ofertę na raz. Każda z nich ma swoją nazwę, opis, informację do kogo jest skierowana oraz proponowaną cenę; jeśli powiązano z nią obrazki to zostaną one wyświetlane pod tekstem jako galeria. Administrator może edytować każdą z tych wartości. Przykładowy wygląd widoku z perspektywy klienta, rysunek 3.13, oraz z perspektywy administratora, rysunek 3.14.

The screenshot shows a website header for 'STAJNIA MALTA' with a logo of a horse and jockey. Below the header is a navigation menu with links: Strona główna, Konie, Trenerzy, Oferta (highlighted in pink), Cennik, Galeria, and Kontakt. The main content area features a section titled 'Jazda Indywidualna' (Individual Riding) with a sub-section 'Jazda terenowa' (Cross-Country Riding). The 'Jazda Indywidualna' section contains text about improving riding skills under experienced trainers, a 'Dla kogo:' note for everyone, a 'Propozycja ceny:' of 100, and a large central image of a dark horse's head and neck. The 'Jazda terenowa' section is partially visible below it.

Rys. 3.13. Przykładowy wygląd widoku ofert z perspektywy klienta

Strona główna Konie Trenerzy Oferta Cennik Galeria Kontakt Kolory Wyloguj

▼ Jazda indywidualna

Doskonały sposób na szlifowanie swoich umiejętności jeździeckich pod okiem doświadczonych trenerów.

Dla kogo: dla każdego  
Proponowana cena: 100



Zmień nazwę Ustaw zdjęcie Usuń

➤ Jazda terenowa

Dodaj nową ofertę

Rys. 3.14. Przykładowy wygląd widoku ofert z perspektywy administratora

### 3.3.5.5. Widok cennika

Widok cennika jest zlokalizowany pod adresem `/prices`. Wyświetla tabelę z nazwą oferty i ceną. Z założenia ma być uproszczonym widokiem ofert, tutaj klient uzyskuje konkretną informację dotyczącą ceny. Przykładowy wygląd widoku z perspektywy klienta, rysunek 3.15, oraz z perspektywy administratora, rysunek 3.16.

The screenshot shows the website header 'STAJNIA MALTA' with a horse icon. Below it is a navigation bar with links: Strona główna, Konie, Trenerzy, Oferta, Cennik, Galeria, Kontakt. The main content area displays a table:

Usługa	Cena
Jazda indywidualna - godzina	100

Rys. 3.15. Przykładowy wygląd widoku cennika z perspektywy klienta

The screenshot shows the website header 'STAJNIA MALTA' with a horse icon. Below it is a navigation bar with links: Strona główna, Konie, Trenerzy, Oferta, Cennik, Galeria, Kontakt, Kolory, Wyloguj. The main content area displays a table with a 'Usuń' (Delete) button and a 'Dodaj nowy przedmiot' (Add new item) button:

Usługa	Cena	Usuń
Jazda indywidualna - godzina	100	

**Dodaj nowy przedmiot**

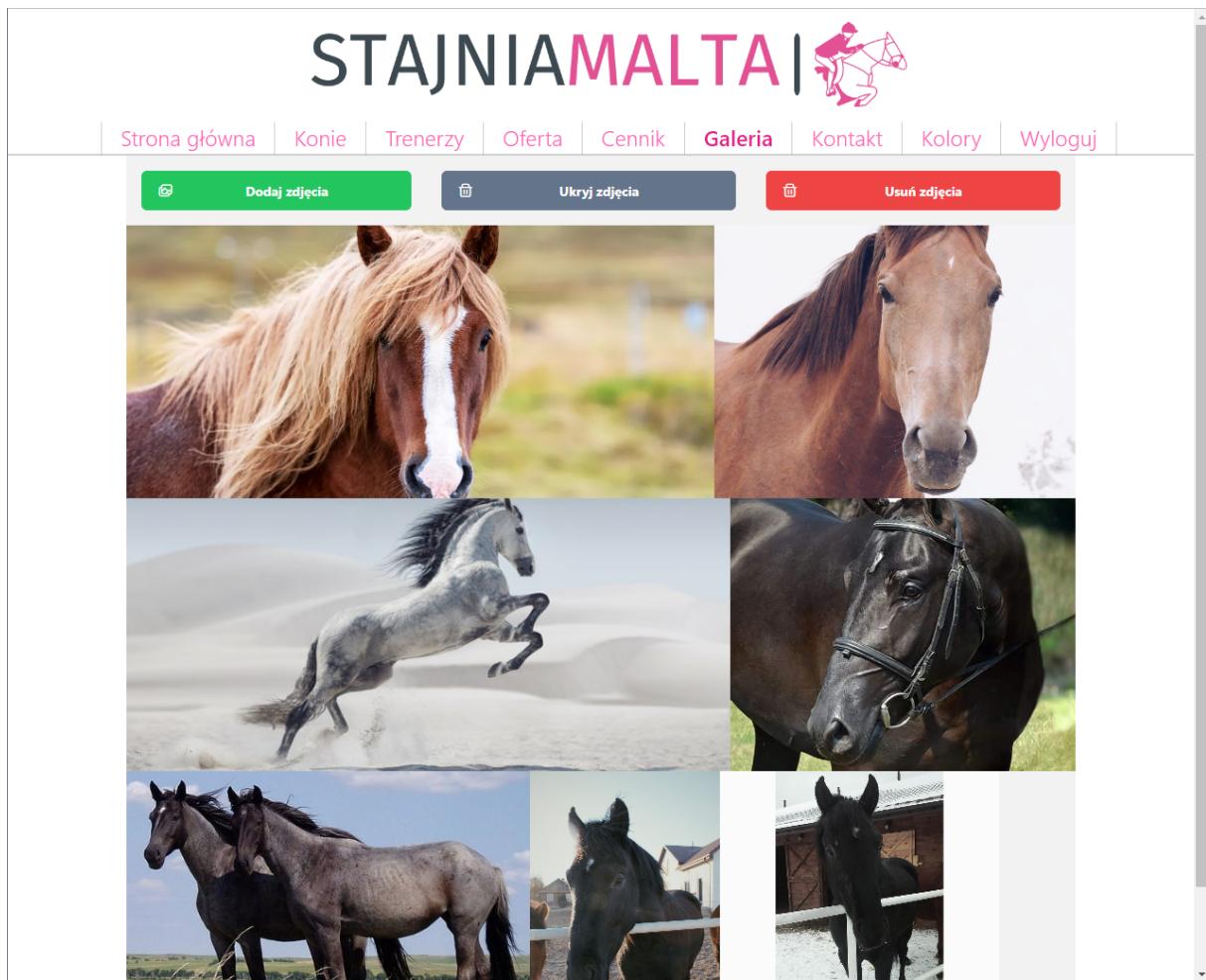
Rys. 3.16. Przykładowy wygląd widoku cennika z perspektywy administratora

### 3.3.5.6. Widok galerii

Widok galerii jest zlokalizowany pod adresem `/gallery`. Wyświetla galerię zdjęć, które są wgrane na stronę. W tym widoku administrator może dodawać i usuwać zdjęcia z bazy oraz zaznaczać, które z nich mają być niewyświetlane w galerii. Przykładowy wygląd widoku z perspektywy klienta, rysunek 3.17, oraz z perspektywy administratora, rysunek 3.18.



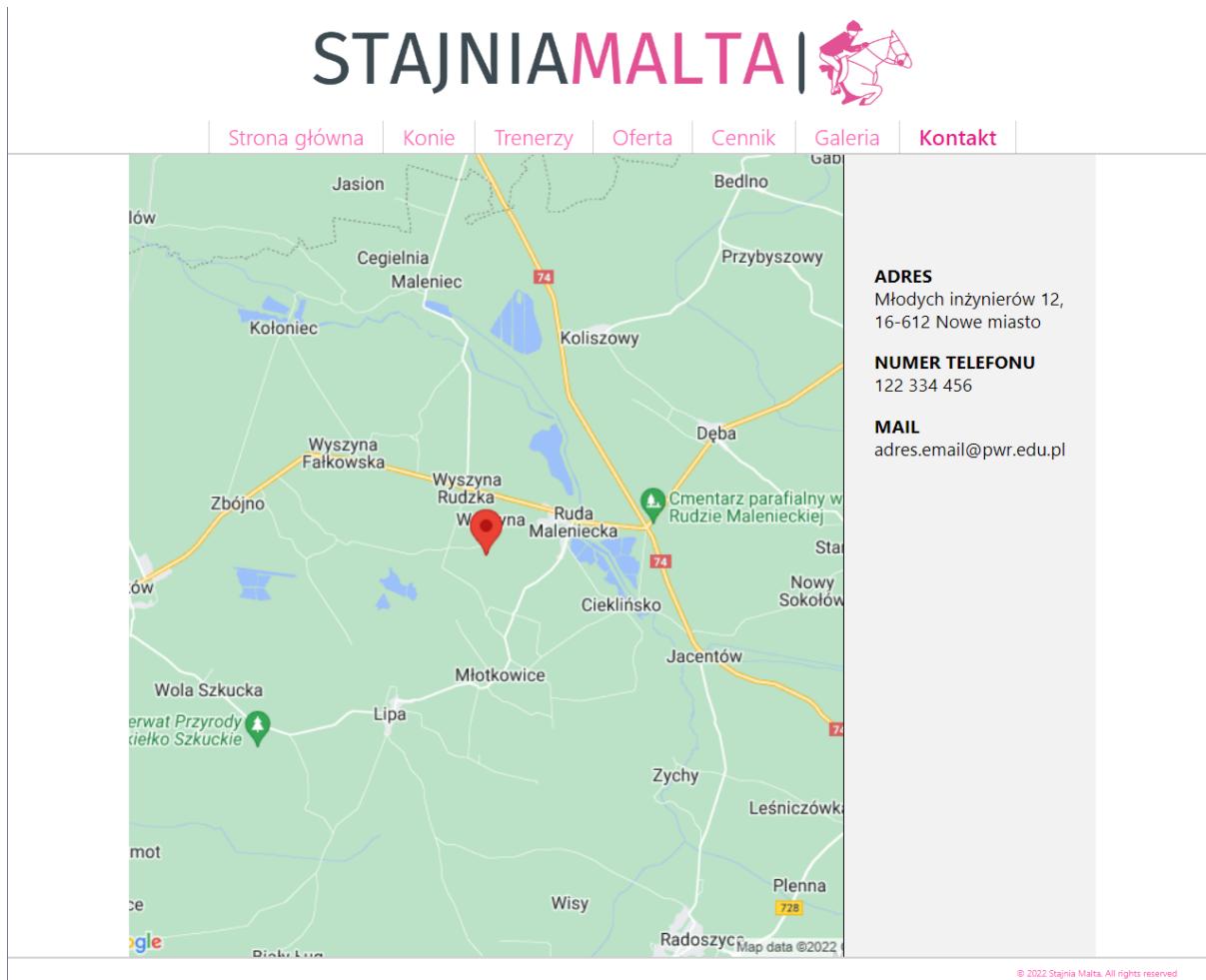
Rys. 3.17. Przykładowy wygląd widoku galerii z perspektywy klienta



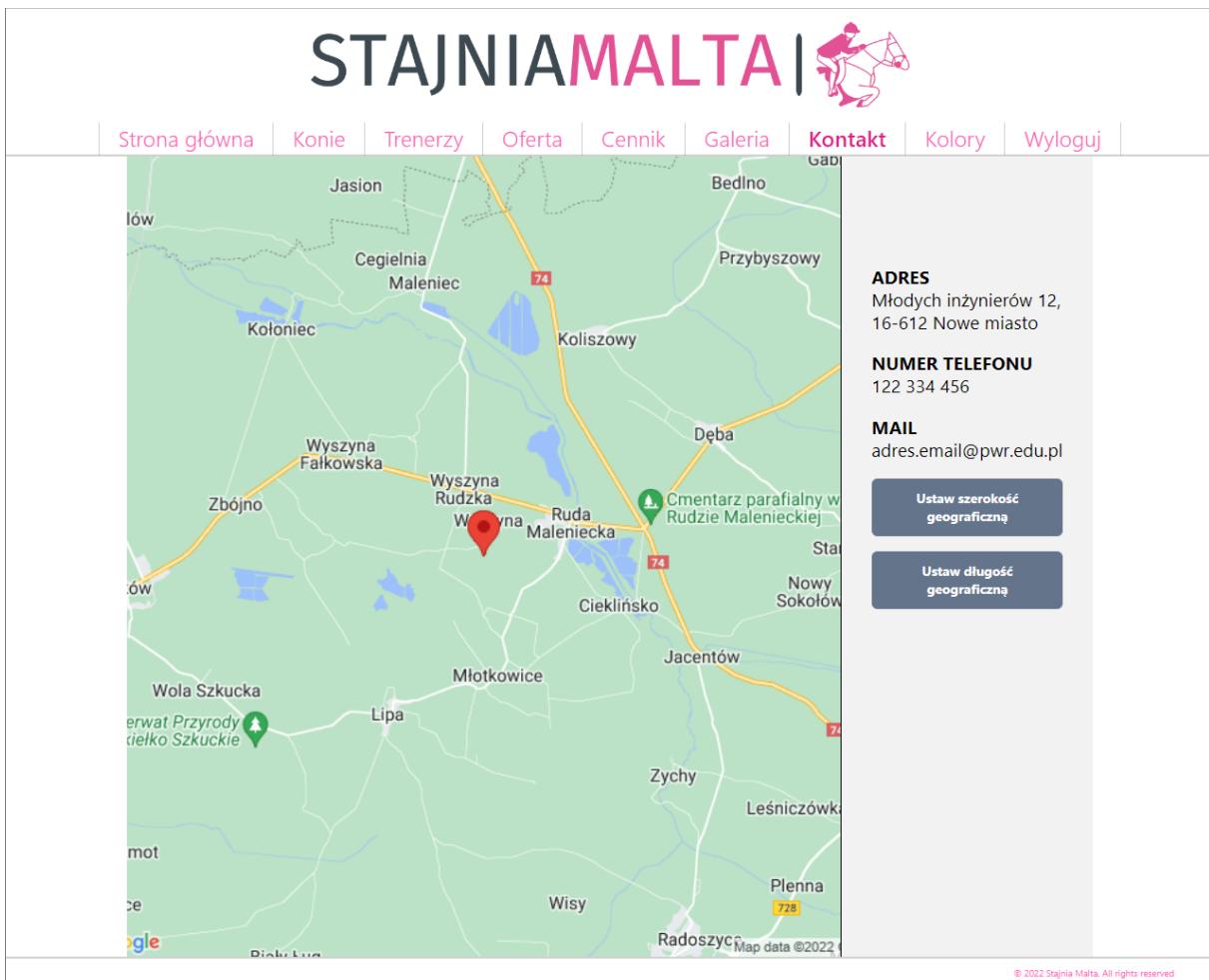
Rys. 3.18. Przykładowy wygląd widoku galerii z perspektywy administratora

### 3.3.5.7. Widok kontaktowy

Widok kontaktowy jest zlokalizowany pod adresem **/contact**. To w nim można znaleźć mapę, wykorzystującą Google Maps API [20], oraz informacje kontaktowe stajni. Administrator może edytować te informacje i zmieniać pozycję mapy podając nowe wartości długości i szerokości geograficznych. Przykładowy wygląd widoku z perspektywy klienta, rysunek 3.19, oraz z perspektywy administratora, rysunek 3.20.



Rys. 3.19. Przykładowy wygląd widoku kontaktowego z perspektywy klienta



Rys. 3.20. Przykładowy wygląd widoku kontaktowego z perspektywy administratora

### 3.3.5.8. Widok logowania i zmiany hasła

Widok logowania i zmiany hasła jest zlokalizowany pod adresem **/login**. Jest to jedyny widok, do którego nie można przejść przyciskiem, należy wpisać adres bezpośrednio w przeglądarkę. Wynika to z faktu, że ta funkcjonalność jest niedostępna dla zwykłego użytkownika. Strona logowania składa się z elementu do wpisania hasła i przycisku, rysunek 3.21. Strona zmiany hasła jest podobna, można do niej przejść tylko poprzez ręczne wpisanie adresu w przeglądarkę po uprzednim zalogowaniu. Również składa się z elementu do wpisania hasła i przycisku, rysunek 3.22. Na obu stronach, zamiast przycisku można posłużyć się klawiszem ENTER.

The screenshot shows the login page for 'STAJNIA MALTA'. At the top, there is a navigation bar with links: 'Strona główna' (Home), 'Konie' (Horses), 'Trenerzy' (Trainers), 'Oferta' (Offer), 'Cennik' (Price list), 'Galeria' (Gallery), and 'Kontakt' (Contact). Below the navigation bar is a logo featuring a horse and jockey. The main area contains a text input field labeled 'Podaj hasło' (Enter password) with a placeholder icon and a green 'Zaloguj' (Log in) button.

Rys. 3.21. Widok logowania

The screenshot shows the password change page for 'STAJNIA MALTA'. At the top, there is a navigation bar with links: 'Strona główna' (Home), 'Konie' (Horses), 'Trenerzy' (Trainers), 'Oferta' (Offer), 'Cennik' (Price list), 'Galeria' (Gallery), 'Kontakt' (Contact), 'Kolory' (Colors), and 'Wyloguj' (Logout). Below the navigation bar is a logo featuring a horse and jockey. The main area contains a text input field labeled 'Nowe hasło' (New password) with a placeholder icon and a dark blue 'Zmień hasło' (Change password) button.

Rys. 3.22. Widok zmiany hasła

## 3.4. WDROŻENIE PROJEKTU

Projekt można uruchomić lokalnie lub z użyciem infrastruktury opisanej na diagramie wdrożenia, rysunek 2.5. W obu przypadkach należy sprawdzić, czy dostępne będą poniższe programy:

- NodeJs, zalecana wersja 18.3.0 lub nowsza,
- npm, zalecana wersja 18.3.0 lub nowsza,
- PostgreSQL zalecana wersja 13.

Oprócz powyższych wymagań należy założyć konto na platformie Deta oraz Google Cloud Platform i uruchomić serwis umożliwiający korzystanie z Google Maps Static API [20]. Przed uruchomieniem projektu należy zgromadzić następujące dane:

- wartości służące do połączenia z bazą danych: PGDATABASE, PGHOST, PGPASSWORD, PGPORT, PGUSER,
- klucz prywatny służący do podpisywania jwt – PRIVATE\_KEY,
- klucz do projektu na platformie Deta – DETA\_KEY,
- klucz do Google Cloud Platform – REACT\_APP\_GMAP\_API\_KEY,
- adres REST API – REACT\_APP\_REST\_API\_URL

Wartości, które będą wykorzystywane we framework'u React muszą mieć przedrostek REACT\_APP. Można również podać PORT, na którym ma zostać uruchomiony serwer, domyślnie jest to 3001.

### 3.4.1. Uruchomienie lokalne

Przed przystąpieniem do uruchomienia serwera należy wgrać skrypt z pliku dbSchema.sql do bazy danych.

Zalecanym systemem jest Linux. W celu uruchomienia lokalnie należy przejść do katalogu, w którym znajduje się projekt i wprowadzić do lokalnego środowiska wartości wymienione w poprzednim punkcie, można to zrobić komendą **export**, przykład takiej operacji na listingu 3.17.

---

```
export PGDATABASE=database
export PGHOST=host
export PGPASSWORD=password
export PGPORT=port
export PGUSER=postgres
export PRIVATE_KEY=private_key
export DETA_KEY=deta_key

export REACT_APP_GMAP_API_KEY=gmap_key
export REACT_APP_REST_API_URL="http://localhost:3001/api/"
```

---

Listing 3.17: Przykład użycia komendy **export**

Następnie należy wpisać standardowe komendy, uruchomią one skrypty, które zainstalują wymagane paczki, zbudują stronę internetową i uruchomią serwer:

```
npm install
npm run build
npm start
```

Stronę internetową można uruchomić niezależnie od serwera w celach dewelopowania.

### **3.4.2. Uruchomienie na platformie Railway**

Platforma Railway jest docelowym miejscem, z którego aplikacja ma funkcjonować. Pierwszym krokiem powinno być sklonowanie projektu do repozytorium na platformie GitHub. Następnie należy założyć konto na platformie, może być ono całkowicie darmowe. Następnie należy stworzyć nowy projekt i w nim dodać bazę danych PostgreSQL. Udostępniony zostanie przejrzysty interfejs, którego możemy użyć do wgrania skryptu z pliku dbSchema.sql, jednak zaleca się skorzystanie z programu psql [9]. W zakładce widoczne są wartości służące do komunikacji z bazą danych, nie należy ich kopować, jeśli aplikacja serwera będzie uruchomiona w ramach tego samego projektu.

Kolejnym krokiem jest dodanie kontenera "GitHub Repo" i połączenie go z wcześniej utworzonym repozytorium. W zakładce "Variables" należy dodać: klucz prywatny, klucz do projektu na platformie Deta, klucz do Google Cloud Platform i adres REST API. Adres strony ustawiany jest w zakładce "Settings", można podać własną domenę lub podać część nazwy, która zostanie zakończona ".up.railway.app". W zakładce "Deployments" widoczny jest aktualnie działająca wersja aplikacji, domyślnie każda zmiana dodana do projektu na GitHub'ie powoduje przebudowanie aplikacji do najnowszej wersji. Dzięki skryptom znajdującym się w projekcie instalacja, budowanie i uruchomienie są automatyczne.

# **PODSUMOWANIE**

## **PODSUMOWANIE PRACY**

Celem pracy było stworzenie strony internetowej stajni. Założenia projektu zostały spełnione, został on zrealizowany w całości. Aplikacja będzie stanowić doskonały dodatek do każdej stajni, zapewni dostęp do informacji potencjalnym klientom i umożliwi właścielowi edycję wyświetlanej treści.

## **DALSZE MOŻLIWOŚCI ROZWOJU**

Mimo że aplikacja jest kompletna i nie wymaga dodatkowej pracy, aby spełniać zadane jej wymagania, można wprowadzić w niej kilka zmian. Jednak przed edycją projektu należałoby napisać do niego testy, pozwoli to wychwycić błędy, które mogą powstać podczas wprowadzania nowych funkcjonalności. Następnie w widoku kontaktu, można użyć interaktywnej mapy zamiast stałego obrazu, dzięki takiemu zabiegowi aplikacja będzie jeszcze bardziej responsywna. Na życzenie właściciela stajni należy również wprowadzać zmiany w wyglądzie strony.

# BIBLIOGRAFIA

- [1] Urząd Statystyczny w Szczecinie, *Społeczeństwo informacyjne w Polsce w 2022 r.*, <https://stat.gov.pl/obszary-tematyczne/nauka-i-technika-spoleczenstwo-informacyjne/spoleczenstwo-informacyjne/spoleczenstwo-informacyjne-w-polsce-w-2022-roku,2,12.html>.
- [2] Mozilla Foundation, *About javascript*, [https://developer.mozilla.org/en-US/docs/Web/JavaScript/About\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript). Stan na dzień 20.11.2022.
- [3] npm, Inc., *npm*, <https://docs.npmjs.com/cli/v8/commands/npm?v=true>. Stan na dzień 20.11.2022.
- [4] OpenJS Foundation, *What is v8?*, <https://nodejs.org/en/about/>. Stan na dzień 22.11.2022.
- [5] OpenJS Foundation, *About node.js*, <https://nodejs.org/en/about/>. Stan na dzień 22.11.2022.
- [6] OpenJS Foundation, *Express*, <https://expressjs.com/>. Stan na dzień 22.11.2022.
- [7] Meta Platforms, Inc., *Introducing hooks*, <https://reactjs.org/docs/hooks-intro.html>. Stan na dzień 20.11.2022.
- [8] The PostgreSQL Global Development Group, *About postgresql*, <https://www.postgresql.org/about/>. Stan na dzień 22.11.2022.
- [9] The PostgreSQL Global Development Group, *Psql documentation*, <https://www.postgresql.org/docs/current/app-psql.html>. Stan na dzień 11.11.2022.
- [10] Abstract Computing UG (haftungsbeschränkt), *Deta*, <https://docs.deta.sh/docs/home>. Stan na dzień 22.11.2022.
- [11] Railway Corp., *Railway documentation*, <https://docs.railway.app/>. Stan na dzień 08.10.2022.
- [12] *bcryptjs*, <https://www.npmjs.com/package/bcryptjs>. Stan na dzień 25.10.2022.
- [13] Abstract Computing UG (haftungsbeschränkt), *Deta drive documentation*, <https://docs.deta.sh/docs/drive/about>. Stan na dzień 08.10.2022.
- [14] *express-fileupload*, <https://www.npmjs.com/package/express-fileupload>. Stan na dzień 04.11.2022.
- [15] *jsonwebtoken*, <https://www.npmjs.com/package/jsonwebtoken>. Stan na dzień 25.10.2022.
- [16] Carlson, B., *Pg documentation*, <https://node-postgres.com/>. Stan na dzień 10.11.2022.
- [17] *Eslint*, <https://www.npmjs.com/package/eslint>. Stan na dzień 25.10.2022.
- [18] bezkoder, *React + node.js express: User authentication with jwt example*, <https://www.bezkoder.com/react-express-authentication-jwt/>. Stan na dzień 01.11.2022.
- [19] PrimeTek, *Primereact components - documentation*, <https://www.primefaces.org/primereact/setup/>. Stan na dzień 08.10.2022.
- [20] Alphabet Inc., *Google map api documentation*, <https://developers.google.com/maps/documentation/maps-static/start>. Stan na dzień 18.10.2022.

# SPIS RYSUNKÓW

2.1. Diagram przypadków użycia aplikacji . . . . .	8
2.2. Diagram klas bazy danych . . . . .	9
2.3. Mapa endpointów REST API . . . . .	10
2.4. Diagram komponentów strony internetowej . . . . .	11
2.5. Diagram wdrożenia aplikacji . . . . .	12
3.1. Edytor kolorów . . . . .	22
3.2. Edytor kolorów, panel wyboru koloru . . . . .	22
3.3. Wybieranie obrazków . . . . .	23
3.4. Edytor tekstu . . . . .	24
3.5. Nagłówek klienta . . . . .	24
3.6. Nagłówek administratora . . . . .	24
3.7. Przykładowy wygląd widoku głównego z perspektywy klienta . . . . .	25
3.8. Przykładowy wygląd widoku głównego z perspektywy administratora . . . . .	26
3.9. Przykładowy wygląd widoku koni z perspektywy klienta . . . . .	27
3.10. Przykładowy wygląd widoku koni z perspektywy administratora . . . . .	28
3.11. Przykładowy wygląd widoku trenerów z perspektywy klienta . . . . .	29
3.12. Przykładowy wygląd widoku trenerów z perspektywy administratora . . . . .	30
3.13. Przykładowy wygląd widoku ofert z perspektywy klienta . . . . .	31
3.14. Przykładowy wygląd widoku ofert z perspektywy administratora . . . . .	32
3.15. Przykładowy wygląd widoku cennika z perspektywy klienta . . . . .	33
3.16. Przykładowy wygląd widoku cennika z perspektywy administratora . . . . .	33
3.17. Przykładowy wygląd widoku galerii z perspektywy klienta . . . . .	34
3.18. Przykładowy wygląd widoku galerii z perspektywy administratora . . . . .	35
3.19. Przykładowy wygląd widoku kontaktowego z perspektywy klienta . . . . .	36
3.20. Przykładowy wygląd widoku kontaktowego z perspektywy administratora . . . . .	37
3.21. Widok logowania . . . . .	38
3.22. Widok zmiany hasła . . . . .	38

# SPIS LISTINGÓW

3.1	Tworzenie wybranych tabel: <i>image</i> , <i>horse</i> , <i>image_horse_junction</i> , fragment dbSchema.sql . . . . .	13
3.2	Dodawanie kluczów obcych do wybranych tabelach: <i>horse</i> , <i>image_horse_junction</i> , fragment dbSchema.sql . . . . .	14
3.3	Tworzenie wybranych widoków: <i>image_list_view</i> , <i>horse_list_view</i> , fragment dbSchema.sql . . . . .	14
3.4	Express static routing, fragment src/app.js . . . . .	15
3.5	Express redirect, fragment src/routes/api.js . . . . .	15
3.6	Express sendFile, fragment src/routes/index.js . . . . .	15
3.7	Express routing, fragment src/app.js . . . . .	16
3.8	Express routing, fragment src/routes/api.js . . . . .	16
3.9	Dodanie rekordu do tabeli <i>horse</i> w Javascript, fragment src/databaseConnector.js .	17
3.10	Inicjalizacja połączenia z Deta Drive <i>images</i> , fragment src/databaseConnector.js .	17
3.11	Autentykacja, fragment src/routes/authenticator.js . . . . .	18
3.12	Autoryzacja, fragment src/middleware/authorizer.js . . . . .	18
3.13	Przykład funkcji <code>fetch()</code> , fragment src/views/HorseView/HorseView.js . . . . .	19
3.14	Funkcja wysyłająca żądanie aktualizacji danych, fragment src/contextProviders/AuthContextProvider/AuthContextProvider.js . . . . .	20
3.15	Funkcja logująca, fragment src/contextProviders/AuthContextProvider/AuthContextProvider.js . . . . .	21
3.16	Funkcja wylogowująca, fragment src/contextProviders/AuthContextProvider/AuthContextProvider.js . . . . .	22
3.17	Przykład użycia komendy <b>export</b> . . . . .	39

## **Dodatki**

## **A. PENDRIVE Z KODEM ŹRÓDŁOWYM**

Do pracy jest dołączony pendrive, na którym znajdują się niniejsza praca oraz kod źródłowy projektu. Projekt można uruchomić stosując się do instrukcji opisanej w sekcji 3.4.