



POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

PROJEKT Z PRZEDMIOTU PROGRAMOWANIE APLIKACJI W CHMURZE OBLICZENIOWEJ

Zestawienie danych na temat wysokości stóp procentowych i cen mieszkań w okresie ostatnich 10 lat, z uwzględnieniem regionów i typów mieszkań

Zespół projektu:

- | | |
|---------------------------|-------|
| 1. Jakub Tadewicz, 7.14 | 99719 |
| 2. Maciej Targoński, 7.14 | 99720 |

Prowadzący:

Mgr. Dariusz Głuchowski

Lublin, 2025

1. Wprowadzenie

1.1 Opis projektu

Aplikacja ma zajmować się zbieraniem oraz analizą danych dotyczących cen mieszkań oraz wysokości stóp referencyjnych NBP w Polsce z podziałem na regiony i typy mieszkań w ciągu ostatnich 10 lat.

1.2 Cel projektu

Celem projektu jest umożliwienie użytkownikowi interaktywnego przeglądania danych historycznych przy pomocy wygenerowanych wykresów, porównywania oraz ich analizowania, a także eksportu do wybranego formatu.

1.3 Zakres projektu

- Pobieranie danych z zewnętrznych API (BDL, GUS)
- Wizualizacja danych w formie wykresów
- Filtrowanie danych według roku, regionu oraz typu mieszkania
- Eksport danych do formatu JSON/XML
- Przechowywanie pobranych danych w bazie MySQL
- Stworzenie interfejsu użytkownika
- Konteneryzacja poszczególnych części projektu

1.4 Wymagania funkcjonalne:

- Uwierzytelnianie użytkowników z wykorzystaniem tokenu JWT
 - rejestracja i logowanie użytkowników
- Buforowanie i zapisywanie danych z API
- Obsługa usługi REST dla pobierania danych
- Eksport danych w formacie XML lub JSON
- Zapis pozyskanych danych do bazy danych aplikacji oraz odczyt danych z bazy do aplikacji z wykorzystaniem ORM
- Implementacja mechanizmu transakcji w dostępie (odczyt zapis) do bazy danych
- Wyświetlanie danych w postaci wykresów
- Filtrowanie danych według wybranych kryteriów
- Eksportowanie danych do plików JSON/XML

1.5 Wymagania niefunkcjonalne

- Wykorzystanie warstwy graficznej
 - interfejs użytkownika
 - pobieranie danych
 - wyświetlanie wyników
- Funkcje usprawniające dostęp do współdzielonych zasobów (poziomy izolacji w bazach danych)
- Stworzenie kontenera Docker obejmującego stworzoną aplikację lub zespół aplikacji wraz ze wszystkimi koniecznymi do działania składnikami
- Możliwość rozbudowy aplikacji
- Wykorzystanie technologii Node.js (Express), Charts.js, React-chartjs-2, React.js, React Router, MySQL, JWT, Bcryptjs, Axios, Sequelize, Archiver, Json2xml, Xml2js, Docker
- System ma być dostępny na wszystkich systemach operacyjnych
- Kompatybilność z popularnymi przeglądarkami

1.6 Architektura projektu

- **Backend**
 - Node.js (Express) – serwer aplikacji
 - Sequelize – ORM do obsługi bazy danych
 - MySQL – baza danych (uruchamiana w kontenerze Docker)
 - JWT (jsonwebtoken) – autoryzacja użytkowników
 - bcryptjs – hashowanie haseł użytkowników
 - axios – pobieranie danych z zewnętrznych API
 - archiver, json2xml, xml2js – eksport/import danych w formatach XML/JSON
- **Frontend**
 - React – interfejs użytkownika
 - React Router – routing po stronie klienta
 - Chart.js + react-chartjs-2 – generowanie wykresów

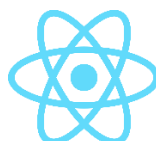


Chart.js



1.7 Plan realizacji projektu

1.7.1 Analiza i przygotowanie koncepcji

- Zidentyfikowanie wymagań funkcjonalnych i нефunkcjonalnych
- Wybór odpowiednich źródeł danych oraz przeanalizowanie działania API

1.7.2 Budowa warstwy serwerowej

- Implementacja logiki serwerowej przy użyciu poszczególnych bibliotek
- Stworzenie połączenia z API
- Utworzenie struktury bazy danych
- Konfiguracja ORM Sequelize

1.7.3 Budowa warstwy klienta

- Zaprojektowanie logicznego interfejsu
- Integracja warstwy serwerowej z klientem za pomocą fetch/http

1.7.4 Konteneryzacja aplikacji

- Utworzenie plików Dockerfile dla backendu i frontendu
- Utworzenie pliku docker-compose.yml
- Rozbicie systemu na cztery kontenery
 - MySQL
 - PHPMyAdmin
 - Server
 - Client

1.7.5 Testowanie i wdrażanie

- Przeprowadzenie testów manualnych
- Weryfikacja poprawności działania kontenerów i komunikacji
- Finalne wdrożenie aplikacji

1.7.6 Harmonogram

Tabela 2.1. Harmonogram prac w projekcie

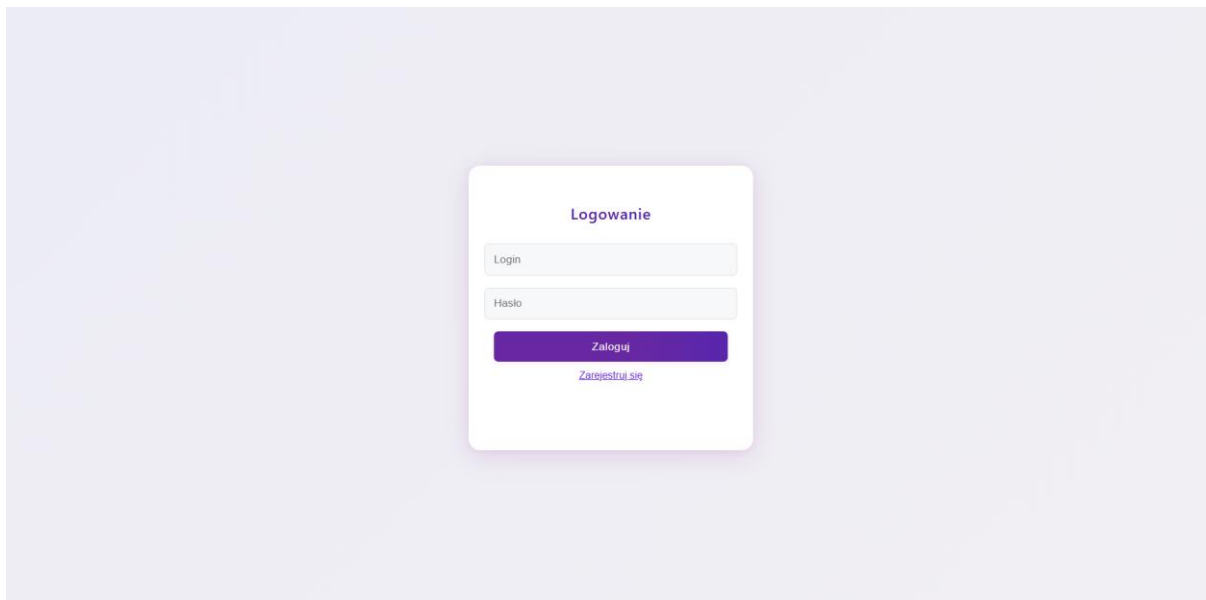
Lp.	Zadanie	Data rozpoczęcia	Data zakończenia	Osoba realizująca	Osoba odpowiedzialna
1.	Analiza i przygotowanie koncepcji	08.05.2025	18.05.2025	Maciej Targoński, Jakub Tadewicz	Jakub Tadewicz
2.	Budowa warstwy serwerowej	10.06.2025	12.06.2025	Maciej Targoński, Jakub Tadewicz	Jakub Tadewicz
3.	Budowa warstwy klienta	12.06.2025	13.06.2025	Maciej Targoński, Jakub Tadewicz	Maciej Targoński
4.	Konteneryzacja aplikacji	13.06.2025	14.06.2025	Maciej Targoński, Jakub Tadewicz	Maciej Targoński
5.	Testowanie i wdrażanie	14.06.2025	14.06.2025	Maciej Targoński, Jakub Tadewicz	Praca wspólna

1.8 Możliwe problemy podczas implementacji

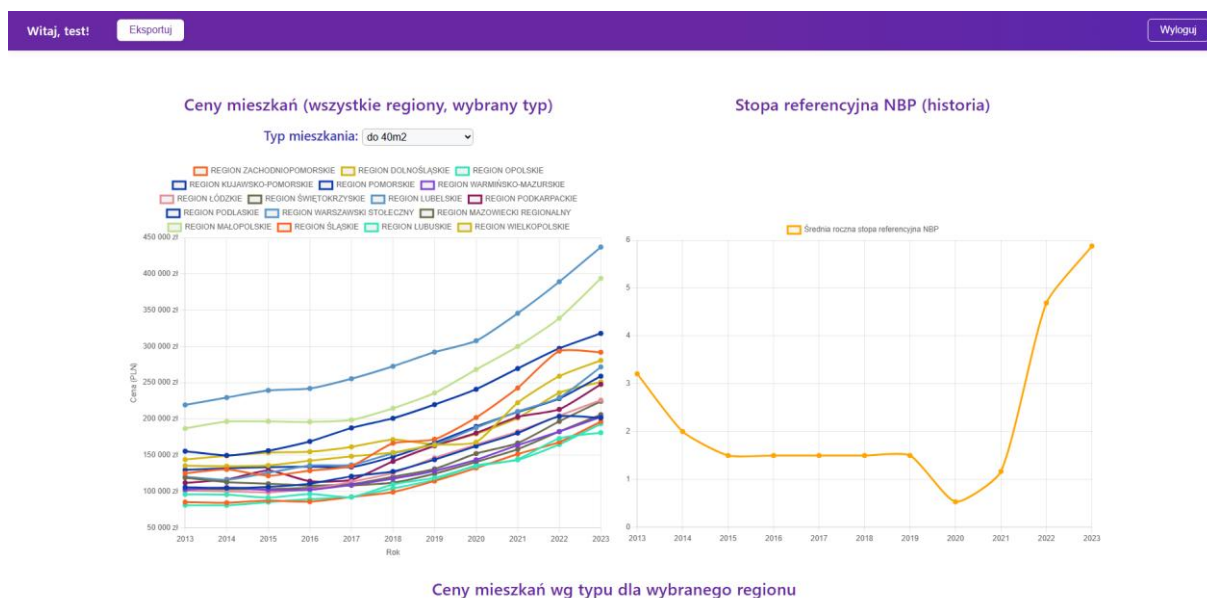
- Ograniczona ilość zapytań API
- Zbieg terminu realizacji projektu z Juwenaliami Politechniki, co może wpłynąć na dostępność i zaangażowanie członków zespołu.
- Problemy z oprogramowaniem systemowym
- Zrozumienie nowych technologii
- Problemy z integracją danych zewnętrznych
- Brak niektórych danych

2. Działanie aplikacji po stronie użytkownika

Aplikacja internetowa została zaprojektowana z myślą o osobach zainteresowanych rynkiem nieruchomości w Polsce. Po wejściu na stronę i założeniu konta użytkownik uzyskuje dostęp do strony głównej zawierającej wszystkie kluczowe funkcje.



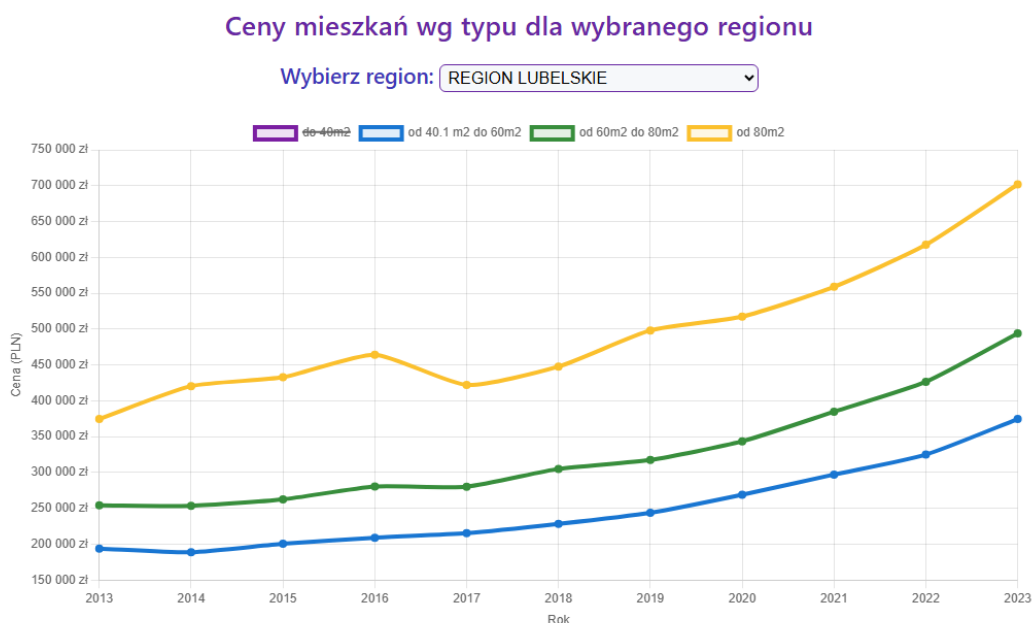
Rys. 1 Formularz logowania



Rys. 2 Widok główny

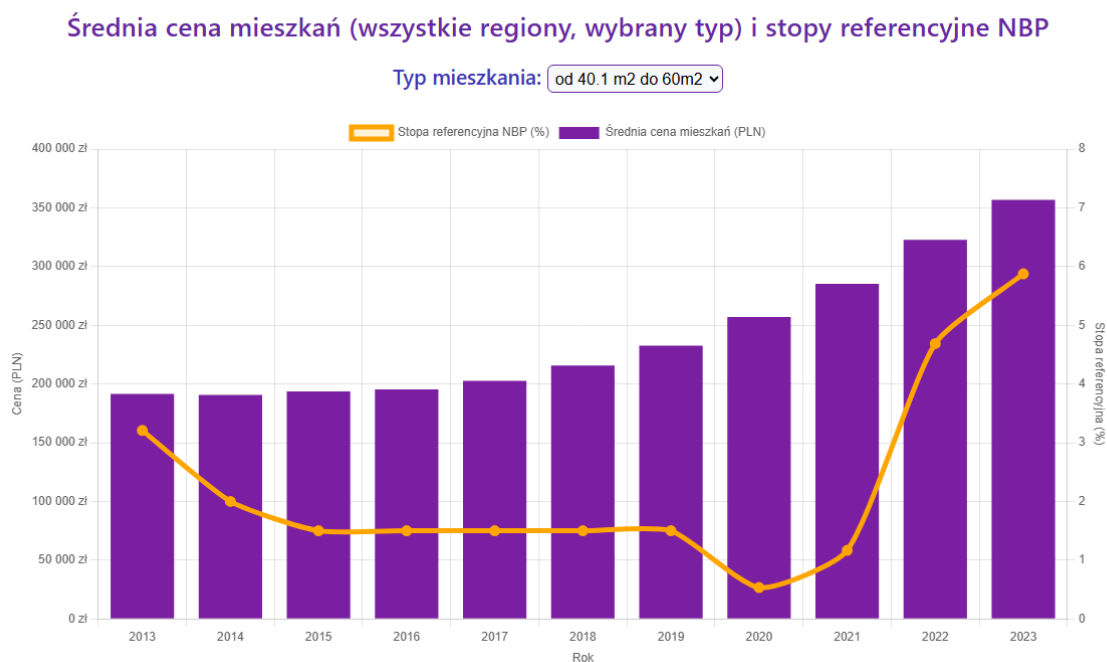
Wyniki prezentowane są w formie dynamicznych wykresów, podczas analizy których możliwe jest wybranie konkretnych regionów, bądź typów mieszkań. Wykres aktualizuje się w sposób automatyczny po zmianie wybranych filtrów.

W ten sposób użytkownik może z łatwością porównać nurtujące go ceny określonych typów mieszkań w danej lokalizacji.



Rys. 3 Wykres z użytymi filtrami

Dla bardziej dociekliwych użytkowników, którzy chcą poznać związek pomiędzy średnimi cenami wybranych typów mieszkań dla wszystkich regionów, a średnią wartością stopy procentowej dostępny jest wykres zbiorczy.



Rys. 4 Wykres "Średnia cena mieszkań (wszystkie regiony, wybrany typ) i stopy referencyjne NBP"

Użytkownik ma możliwość wyeksportowania poszczególnych danych w formacie JSON lub XML.

Witaj, test! Eksportuj Wyloguj

Eksport danych

Regiony:

- ☒ REGION ZACHODNIOPOMORSKIE
- ☒ REGION OPOLSKIE
- ☒ REGION POMORSKIE
- ☐ REGION ŁÓDZKIE
- ☐ REGION LUBELSKIE
- ☐ REGION PODLASKIE
- ☐ REGION MAZOWIECKI
- ☐ REGION ŚLĄSKIE
- ☐ REGION WIELKOPOLSKIE
- ☐ REGION DOLNOŚLĄSKIE
- ☐ REGION KUJAWSKO-POMORSKIE
- ☐ REGION WARMIŃSKO-MAZURSKIE
- ☐ REGION ŚWIĘTOKRZYSKIE
- ☐ REGION PODKARPACKIE
- ☐ REGION WARSZAWSKI
- ☐ REGION MAŁOPOLSKIE
- ☐ REGION LUBUSKIE

Typy mieszkań:

- ☒ do 40m2
- ☒ od 40.1 m2 do 60m2
- ☐ od 60m2 do 80m2
- ☐ od 80m2

Zakres lat: 2013 do 2023

Format eksportu:

- ☒ JSON
- ☐ XML

Eksportuj Anuluj

Rys. 5 Formularz eksportu danych

3. Opis działania aplikacji po stronie serwera

Serwer aplikacji zbudowany jest w oparciu o środowisko Node.js z wykorzystaniem framework'a Express.js. Jego działanie opiera się na obsłudze żądań przychodzących od klienta oraz zarządzaniu bazą danych i zewnętrznymi źródłami danych(API). Po otrzymaniu żądania serwer weryfikuje uprawnienia za pomocą token'a JWT, a następnie w zależności od typu, serwer pobiera odpowiednie dane z bazy MySQL lub jeśli to konieczne z zewnętrznych API. Dodatkowo dba o bezpieczeństwo użytkownika hashując hasła.

Serwer przetwarza i filtruje otrzymane dane zgodnie z przesłanymi wyborami użytkownika(np. wybrany region) i zwraca w odpowiednim formacie na wykresach. W przypadku eksportu danych generowane są pliki w wybranych formatach.

4. Struktura projektu:

Projekt składa się z dwóch głównych katalogów:

- **client** – zawiera aplikację frontendową stworzoną z wykorzystaniem React.js. Znajduje się tu folder src, który zawiera:
 - components – komponenty aplikacji takie jak formularze logowania (LoginForm.jsx).
 - pages – strony aplikacji, w tym dashboard (Dashboard.jsx).
 - Pliki główne aplikacji takie jak App.jsx, index.js oraz style (App.css, index.css).

Dodatkowo w katalogu client znajduje się plik Dockerfile, który określa środowisko uruchomieniowe aplikacji klienta.

- **server** – zawiera aplikację backendową napisaną w JavaScript (Node.js). Jest podzielona na:
 - models – modele danych, takie jak Housing.js, NBPRate.js, oraz User.js.
 - routes – ścieżki API, które obsługują zapytania klienta, w tym api.js, auth.js.

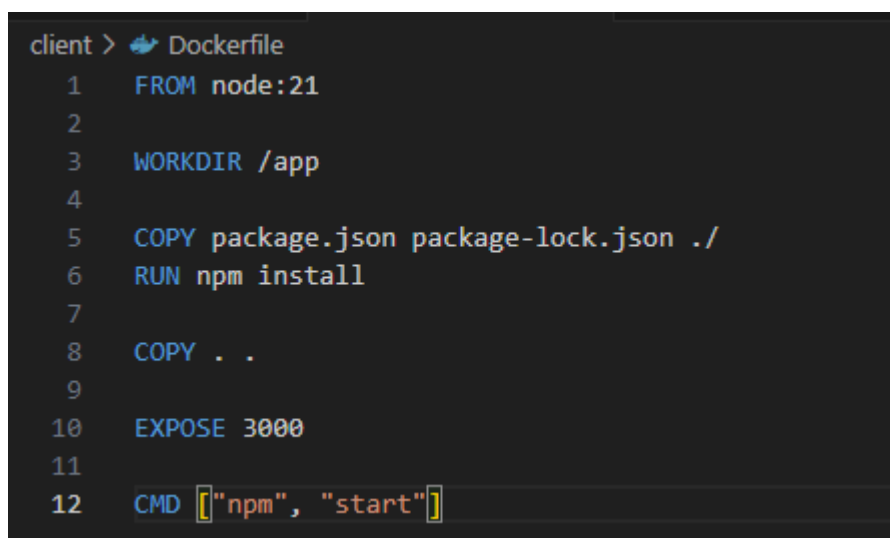
Pliki konfiguracyjne oraz pomocnicze jak połączenie z bazą danych db.js i plik uruchomieniowy index.js. Podobnie jak aplikacja klienta, backend zawiera plik Dockerfile określający jego środowisko uruchomieniowe.

W katalogu głównym projektu znajduje się plik docker-compose.yml, który służy do jednoczesnego uruchamiania aplikacji klienckiej i serwera, tworząc skoordynowane środowisko kontenerowe. Dodatkowo w katalogu głównym projektu znajduje się również plik README.md. Jego głównym celem jest dostarczenie użytkownikom podstawowych informacji dotyczących konfiguracji, uruchomienia aplikacji przy pomocy Docker Compose oraz opis wymaganych plików konfiguracyjnych, zmiennych środowiskowych, haseł oraz instrukcji uruchamiania i zatrzymywania całej aplikacji.

5. Dockerfile

Każdy kontener wystawia łatwo rozróżnialny port, co pozwala na łatwe powiązanie między kontenerami oraz hostem.

- **Dockerfile dla klienta** ustawia folder roboczy na /app i wystawia port 3000. Instalowane są zależności na podstawie plików package.json i package-lock.json, a następnie kopiowane są pozostałe pliki aplikacji. Kontener uruchamiany jest poleceniem npm start, co pozwala na uruchomienie serwera deweloperskiego React.
- **Dockerfile dla serwera** ustawia folder roboczy na /app i wystawia port 4000. Podobnie jak w przypadku klienta, instalowane są zależności z package.json, a następnie kopiowane są pozostałe pliki aplikacji. Kontener uruchamiany jest poleceniem npm start, co uruchamia backend Node.js/Express.



```
client > Dockerfile
1 FROM node:21
2
3 WORKDIR /app
4
5 COPY package.json package-lock.json ./
6 RUN npm install
7
8 COPY . .
9
10 EXPOSE 3000
11
12 CMD ["npm", "start"]
```

Rysunek 1 Plik Dockerfile dla aplikacji frontendowej (React).

Obraz bazowy: node:21 – zapewnia środowisko Node.js potrzebne do uruchomienia Reacta

WORKDIR /app – ustawia katalog roboczy na /app.

COPY package.json package-lock.json ./ oraz **RUN npm install** - kopiuje pliki zależności i instaluje wszystkie wymagane moduły npm

COPY . . – kopiuje całą zawartość katalogu projektu do obrazu

EXPOSE 3000 – udostępnia port 3000, na którym działa frontend

CMD ["npm", "start"] – uruchamia aplikację React

```

server > Dockerfile
1  FROM node:21
2
3  WORKDIR /app
4
5  COPY package.json ./
6  RUN npm install
7
8  COPY . .
9
10 EXPOSE 4000
11
12 CMD ["npm", "start"]

```

Rys. 6 Plik Dockerfile dla backendu (Node.js/Express)

Obraz bazowy: node:21 – środowisko Node.js dla backendu

WORKDIR /app – katalog roboczy /app

COPY package.json ./ oraz RUN npm install – kopiuje plik zależności i instaluje wymagane moduły npm

COPY . . – kopiuje całą zawartość katalogu backendu do obrazu

EXPOSE 4000 – udostępnia port 4000, na którym działa backend

CMD ["npm", "start"] – uruchamia serwer Express

Plik docker-compose.yml definiuje konfigurację usług, woluminów oraz sieci wykorzystywanych w projekcie. W projekcie zdefiniowane są cztery główne kontenery: backend (server), frontend (client), baza danych MySQL oraz phpMyAdmin. Każdy z nich pełni określoną rolę w architekturze aplikacji, a ich konfiguracja umożliwia współpracę i komunikację pomiędzy usługami. Kontener mysql obsługuje bazę danych MySQL. Mapowanie portów "3307:3306" umożliwia dostęp do bazy z hosta.

Kluczowe zmienne środowiskowe, takie jak MYSQL_DATABASE, MYSQL_USER, MYSQL_PASSWORD oraz MYSQL_ROOT_PASSWORD, konfigurowane są podczas uruchamiania bazy. Montowanie wolumenu wolumenu mysql_data zapewnia trwałość danych bazy między restartami kontenerów.

```

🔥 docker-compose.yml
1  version: '3.8'
2
3  services:
4  mysql:
5      image: mysql:8.0
6      container_name: projekt-mysql
7      restart: always
8      environment:
9          MYSQL_ROOT_PASSWORD: root
10         MYSQL_DATABASE: projekt_db
11         MYSQL_USER: user
12         MYSQL_PASSWORD: password
13     ports:
14         - "3307:3306"
15     volumes:
16         - mysql_data:/var/lib/mysql
58 volumes:
59     mysql_data:

```

Rys. 7 Kontener MySQL

Kontener phpmyadmin zapewnia graficzny interfejs do zarządzania bazą danych MySQL. Mapowanie portów "8080:80" umożliwia dostęp do panelu przez przeglądarkę. Zmienna środowiskowa PMA_HOST wskazuje na usługę mysql, a PMA_PORT na port bazy. Kontener ten zależy od uruchomienia bazy danych.

```

18  phpmyadmin:
19      image: phpmyadmin/phpmyadmin
20      container_name: projekt-phpmyadmin
21      restart: always
22      environment:
23          PMA_HOST: mysql
24          PMA_PORT: 3306
25          MYSQL_ROOT_PASSWORD: root
26      ports:
27          - "8080:80"
28      depends_on:
29          - mysql

```

Rys. 8 Kontener PHPMYAdmin

Kontener server zawiera aplikację backendową (Node.js/Express). Mapowanie portów "4000:4000" pozwala na dostęp do backendu z hosta. Zmienne środowiskowe konfigurowane są tak, aby backend mógł połączyć się z bazą danych. Kontener zależy od mysql, co gwarantuje, że backend uruchomi się dopiero po uruchomieniu bazy danych.

```

31  server:
32      build: ./server
33      container_name: projekt-server
34      restart: always
35      environment:
36          DB_HOST: mysql
37          DB_PORT: 3306
38          DB_NAME: projekt_db
39          DB_USER: user
40          DB_PASSWORD: password
41      ports:
42          - "4000:4000"
43      depends_on:
44          - mysql

```

Rys. 9 Kontener server

Kontener client odpowiada za aplikację frontendową (React). Mapowanie portów "3000:3000" umożliwia dostęp do aplikacji przez przeglądarkę. Zmienna środowiskowa CHOKIDAR_USEPOLLING poprawia działanie hot-reload w środowisku Docker. Kontener zależy od backendu, co zapewnia, że frontend uruchomi się dopiero po uruchomieniu backendu.

```

46  client:
47      build: ./client
48      container_name: projekt-client
49      restart: always
50      environment:
51          CHOKIDAR_USEPOLLING: "true"
52          BROWSER: "none"
53      ports:
54          - "3000:3000"
55      depends_on:
56          - server
57

```

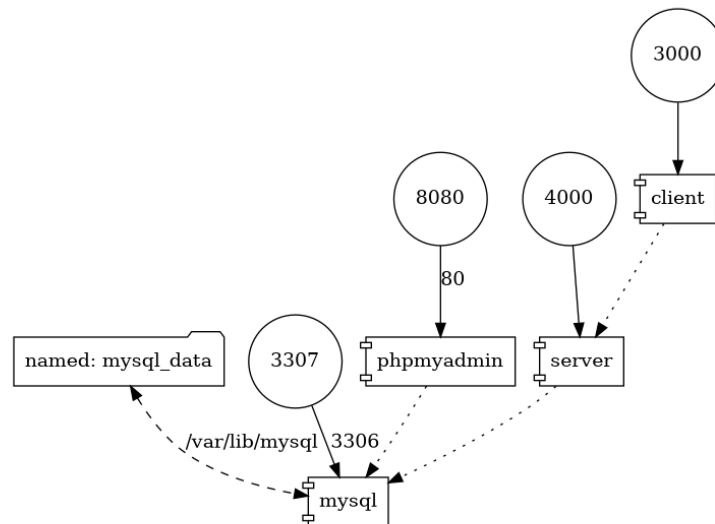
Rys. 10 Kontener client

Podsumowując, projekt składa się z kilku usług Docker, które współpracują ze sobą w określony sposób:

- Frontend (client) komunikuje się z backendem (server).
- Backend (server) komunikuje się z bazą danych MySQL (mysql).
- phpMyAdmin umożliwia zarządzanie bazą danych przez interfejs webowy.
- Każda usługa ma przypisane odpowiednie woluminy do przechowywania danych, co zapewnia trwałość danych między restartami kontenerów.

- Usługi są częścią wspólnej sieci Docker, co umożliwia komunikację między nimi.

Na podstawie plików Docker oraz wygenerowanego diagramu można zobaczyć, jak poszczególne usługi w projekcie współpracują ze sobą. Diagram został wygenerowany przy użyciu narzędzia `docker-compose-viz`, które umożliwia graficzną prezentację zależności pomiędzy kontenerami zdefiniowanymi w pliku `docker-compose.yml`.



Rys. 11 Diagram na podstawie `docker-compose.yml`

Na diagramie widoczne są cztery główne kontenery:

- **mysql** – baza danych, do której podłączony jest wolumen `mysql_data` zapewniający trwałość danych. Baza nasłuchuje na porcie 3306, a na zewnątrz udostępniana jest na porcie 3307.
- **phpmyadmin** – panel administracyjny do zarządzania bazą danych, dostępny na porcie 8080, komunikuje się bezpośrednio z kontenerem `mysql`.
- **server** – backend aplikacji (Node.js/Express), który łączy się z bazą danych `mysql` na porcie 3306 i udostępnia API na porcie 4000.
- **client** – frontend (React), który komunikuje się z backendem na porcie 4000 i jest dostępny na porcie 3000.

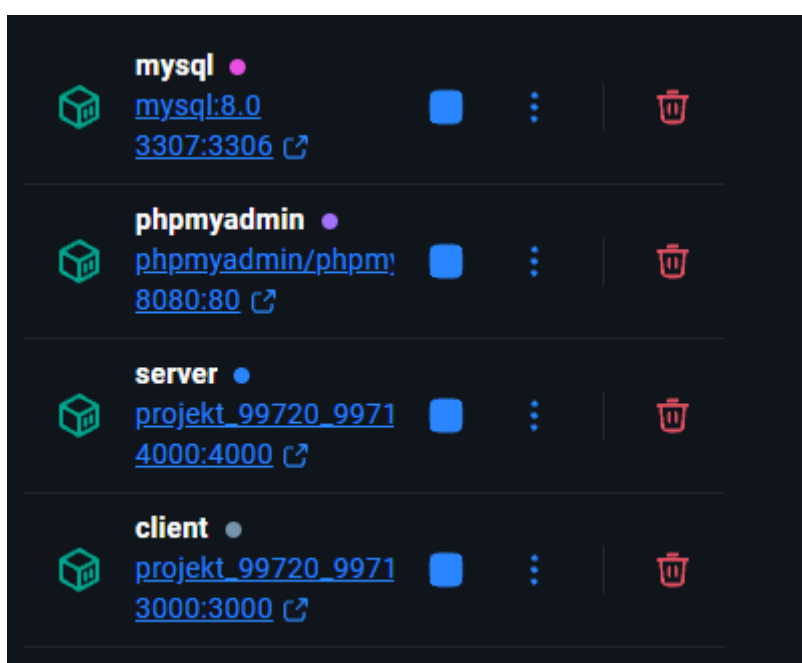
Strzałki na diagramie pokazują kierunki komunikacji pomiędzy usługami oraz mapowanie portów na hosta. Dodatkowo, wolumen `mysql_data` jest powiązany z kontenerem `mysql`, co zapewnia trwałość danych nawet po restarcie kontenerów. Diagram oraz konfiguracja w `docker-compose.yml` wspólnie ilustrują, jak usługi współpracują ze sobą, dzieląc zasoby i komunikując się

przez wyznaczone porty i wolumeny. Taka architektura umożliwia łatwe zarządzanie, skalowanie oraz monitorowanie poszczególnych komponentów aplikacji.

6. Działanie Dockera

W programie docker desktop widoczne są poszczególne działające w danym momencie kontenery:

- **mysql** – silnik bazy danych (port 3306)
- **phpmyadmin** – panel graficzny do zarządzania bazą (port 8080)
- **server** – backend Express.js obsługujący logikę aplikacji (port 4000)
- **client** – frontend Reacta odpowiedzialny za interfejs użytkownika (port 3000)



Rys. 12 Widok kontenerów w aplikacji docker desktop

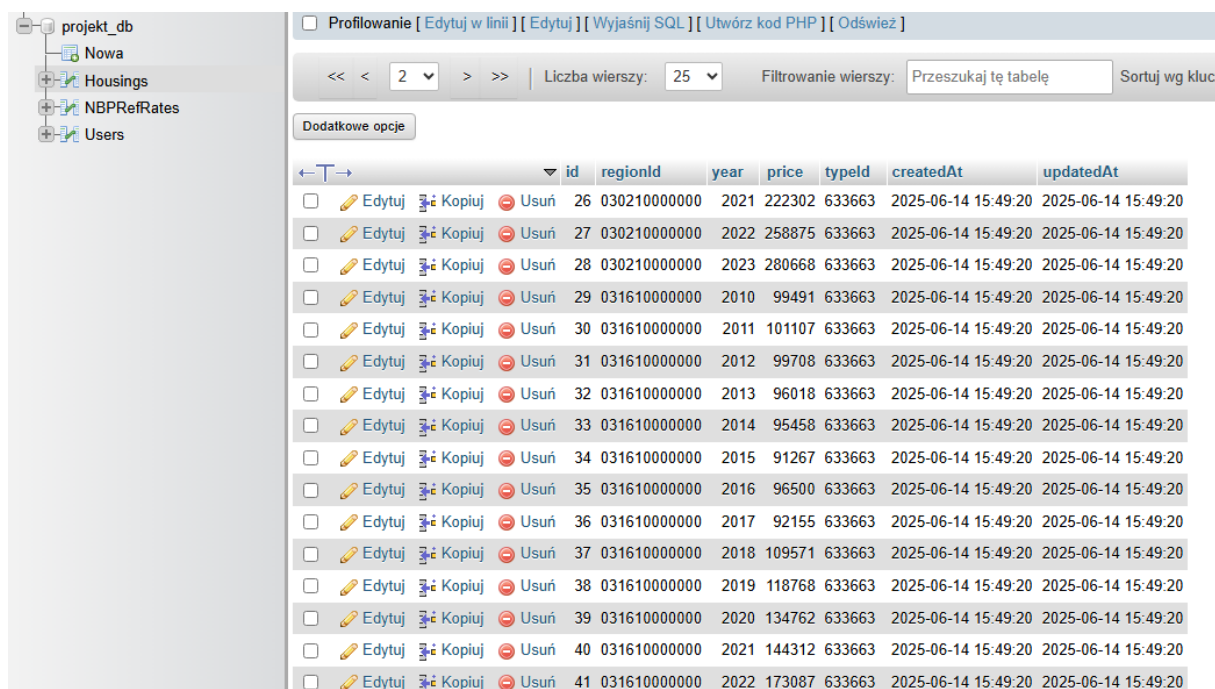
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
def836cde06c	projekt_99720_99719-client	"docker-entrypoint.s..."	5 hours ago	Up About a minute	0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp	projekt-client
dfdaa4abb08d	projekt_99720_99719-server	"docker-entrypoint.s..."	5 hours ago	Up About a minute	0.0.0.0:4000->4000/tcp, [::]:4000->4000/tcp	projekt-server
cb1940fd0ff1	phpmyadmin/phpmyadmin	"/docker-entrypoint.s..."	5 hours ago	Up About a minute	0.0.0.0:8080->80/tcp, [::]:8080->80/tcp	projekt-phpmyadmin
2423800ae0f0	mysql:8.0	"docker-entrypoint.s..."	5 hours ago	Up About a minute	0.0.0.0:3307->3306/tcp, [::]:3307->3306/tcp	projekt-mysql

Rys. 13 Wynik komendy docker ps -a

Jak widać każdy kontener działa poprawnie i posiada przypisane porty zarówno po stronie hosta, jak i kontenera, co zapewnia możliwość komunikacji między usługami oraz dostęp do przeglądarki.

7. Działanie serwisu PHPMyAdmin

W ramach projektu uruchomiono również usługę PHPMyAdmin w osobnym kontenerze. Jest to narzędzie służące do graficznego zarządzania bazą danych MySQL, które w dużym stopniu ułatwia wgląd w strukturę danych oraz ich edycję. Dzięki odpowiedniemu mapowaniu portów możliwy jest dostęp do interfejsu z poziomu przeglądarki pod adresem localhost:8080.



	id	regionId	year	price	typeId	createdAt	updatedAt
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	26	030210000000	2021	222302	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	27	030210000000	2022	258875	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	28	030210000000	2023	280668	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	29	031610000000	2010	99491	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	30	031610000000	2011	101107	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	31	031610000000	2012	99708	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	32	031610000000	2013	96018	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	33	031610000000	2014	95458	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	34	031610000000	2015	91267	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	35	031610000000	2016	96500	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	36	031610000000	2017	92155	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	37	031610000000	2018	109571	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	38	031610000000	2019	118768	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	39	031610000000	2020	134762	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	40	031610000000	2021	144312	633663	2025-06-14 15:49:20	2025-06-14 15:49:20
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	41	031610000000	2022	173087	633663	2025-06-14 15:49:20	2025-06-14 15:49:20

Rys. 14 Panel PHPMyAdmin

Po zalogowaniu do bazy MySQL użytkownik zyskuje możliwość m.in. podglądu tabeli, rekordów i ich struktury, edycji danych oraz importu i eksportu bazy danych.