Collegium Witelona Uczelnia Państwowa w Legnicy
Wydział Nauk Technicznych i Ekonomicznych
Kierunek: Informatyka

Projekt z przedmiotu "Projektowanie i programowanie systemów internetowych I"

Temat: Faily.

Autorzy
Mateusz Bogacz-Drewniak, nr. indeksu: 44491
Gabriela Grabarska, nr. indeksu: 43840
Mateusz Chimkowski, nr. indeksu: 43831

Prowadzący przedmiot
mgr inż. Krzysztof Rewak

Legnica, 2025

Spis treści

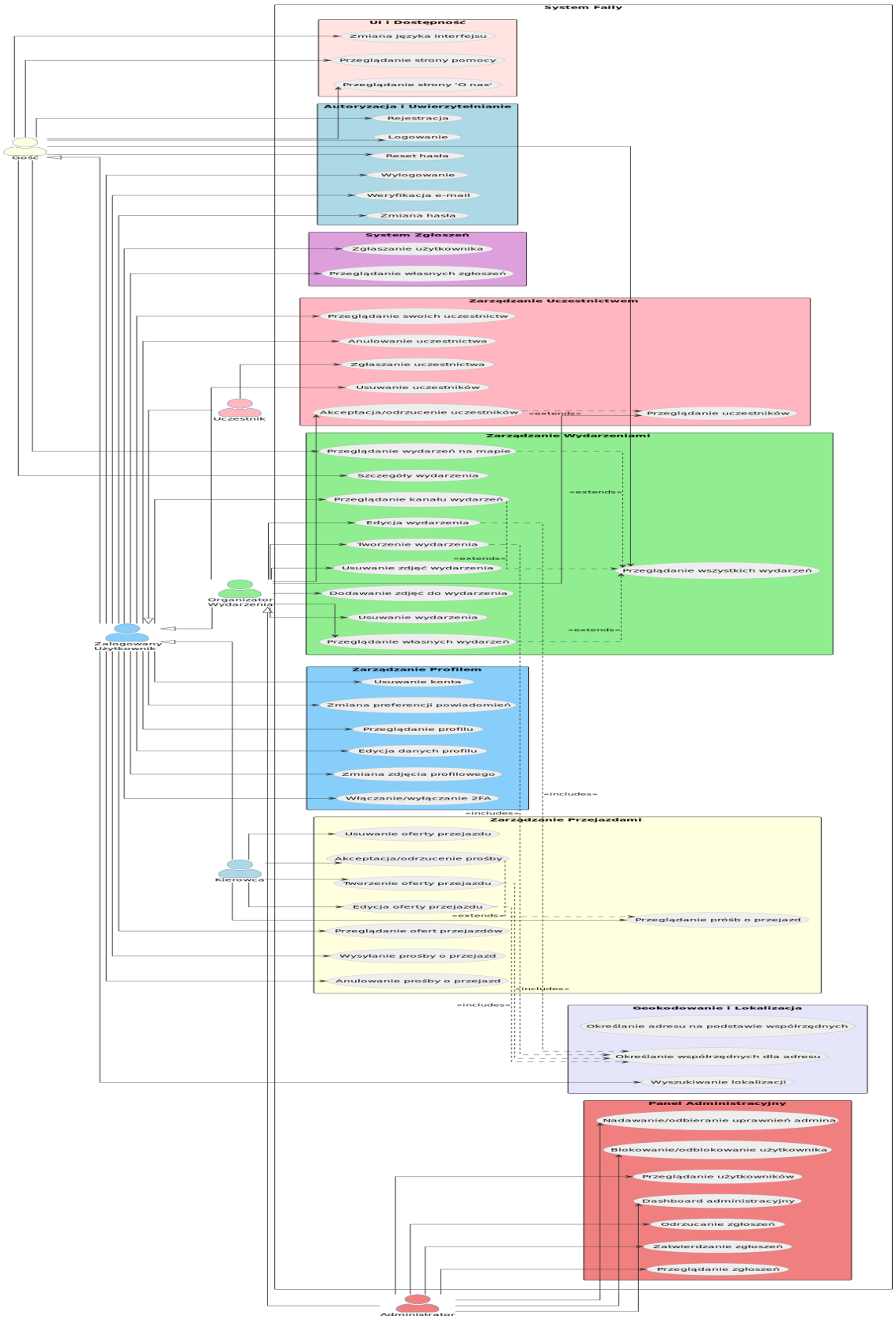
1 Cel i ogólna charakterystyka projektu

Projekt Faily to webowa aplikacja przygotowana jako zaliczenie przedmiotu „Projektowanie i programowanie systemów internetowych”na drugim roku studiów.

1.1 Główne cele projektu

Celem projektu jest umożliwienie użytkownikom:

- zawierania nowych znajomości,
- organizowania spotkań z poznanymi już osobami,
- zarządzania wydarzeniami i ofertami wspólnych eventów,
- organizowania wspólnych dojazdów do miejsc z konkretnymi wydarzeniami.



Rysunek 1: Diagram przypadków użycia(opracowanie własne)

1.2 Funkcjonalność systemu

System pozwala użytkownikom na:

- rejestrację w systemie,
- tworzenie wydarzeń,
- przeglądanie wydarzeń (na mapie, w panelu aktualności lub stronie postów),
- zgłaszanie udziału w wydarzeniach,
- tworzenie i wysyłanie próśb o wspólny dojazd na wydarzenia.

2 Wnioski projektowe

W trakcie realizacji projektu napotkano szereg wyzwań, które dostarczyły cennych doświadczeń dla przyszłych projektów.

2.1 Główne wyzwania

1. **Aspekty programistyczne** – Napotkano znaczące wyzwania związane z implementacją złożonych funkcjonalności systemu.
2. **Komunikacja zespołowa** – Komunikacja w zespole wymagała usprawnienia, szczególnie w zakresie koordynacji zadań.
3. **Harmonogram projektu** – Pierwotny harmonogram stwarzał nadmierne obciążenie dla członków zespołu.
4. **Podział kompetencji** – Podział zadań nie był optymalny względem kompetencji członków zespołu.
5. **Złożoność projektu** – Faktyczna złożoność projektu przewyższyła wstępne szacunki dotyczące wymaganych zasobów.
6. **Kompatybilność technologii** – Wybór frameworka frontend niezgodnego z domyślnym stosem Laravel spowodował znaczne trudności w integracji z gotowymi komponentami autoryzacji (Laravel Breeze), wymagając dodatkowych nakładów na dostosowanie interfejsu.

2.2 Możliwości ulepszenia działania

Na podstawie zdobytych doświadczeń, w przyszłości warto:

1. Wybierać technologie frontend kompatybilne z frameworkiem backend, aby uniknąć problemów integracyjnych i skrócić czas developmentu.
2. Lepiej podzielić role w zespole ze względu na wiedzę i doświadczenie członków.
3. Dokładniej oszacować złożoność projektu na etapie planowania.
4. Wdrożyć lepsze praktyki komunikacji zespołowej.

3 Struktura katalogów i plików projektu

Projekt został zorganizowany zgodnie z konwencjami Laravel, co ułatwia nawigację po kodzie i rozszerzanie funkcjonalności.

3.1 Główne pliki konfiguracyjne

- **compose.yaml** – główny plik Docker Compose opisujący usługi aplikacji (kontener aplikacji, bazy danych, Redis, Mailpit).
- **environment/dev/app/** – pliki konfiguracyjne dla kontenera aplikacji: Dockerfile, konfiguracje Nginx, PHP-FPM, Supervisor.

3.2 Struktura kodu aplikacji

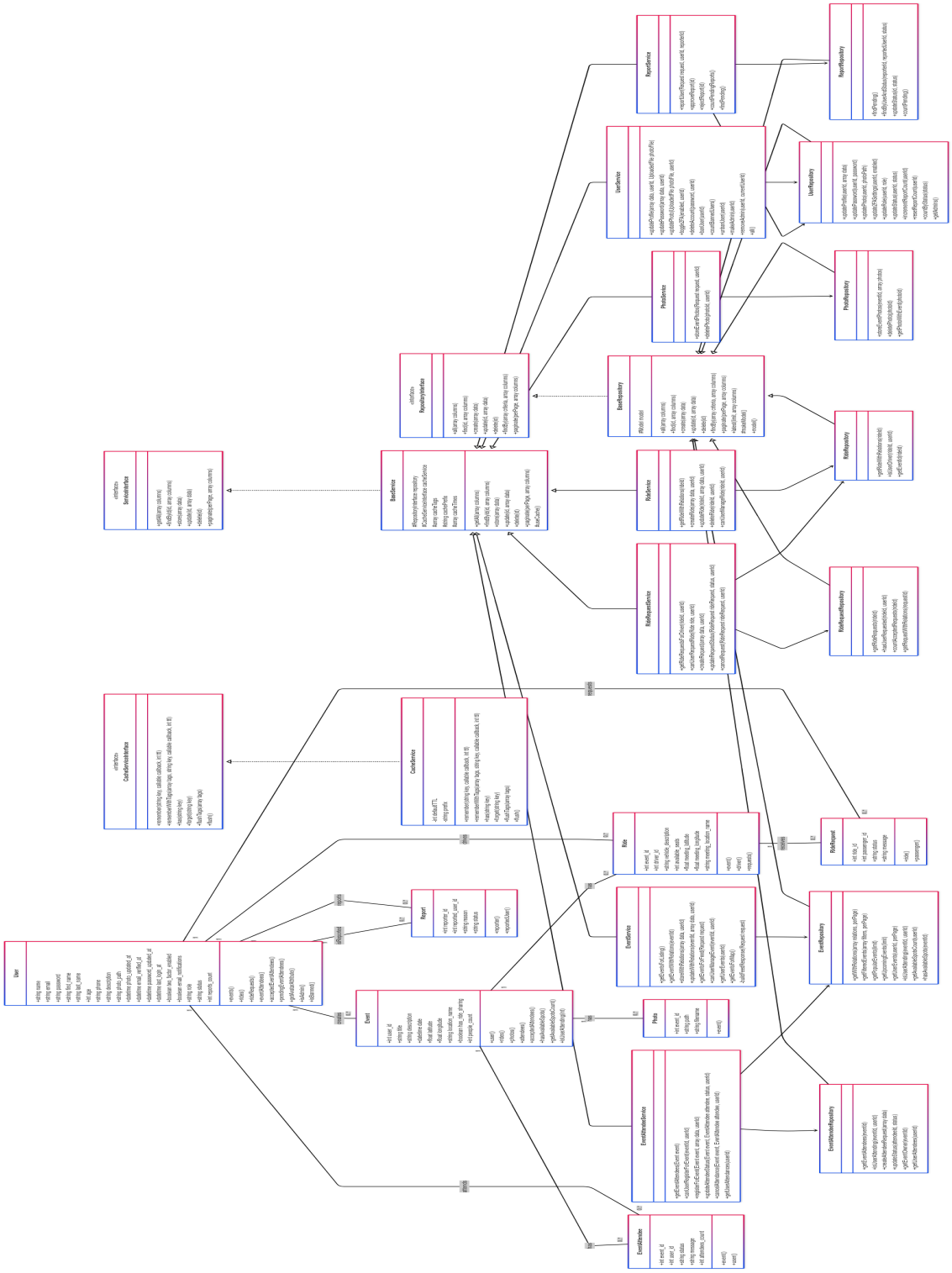
3.2.1 Katalog src/

Główny katalog kodu aplikacji Laravel zawiera:
src/app/ – kod aplikacji obejmujący:

- **Http/Controllers/** – kontrolery odpowiadające za logikę HTTP (kontroler eventów, uwierzytelniania, API, geokodowania)
- **Models/** – modele Eloquent (User, Event, EventAttendee, Ride, Photo)
- **Repositories/** – klasy dostępu do danych (EventRepository, UserRepository)
- **Services/** – warstwa serwisowa pośrednicząca między kontrolerami a repozytoriami
- **Notifications/** – klasy powiadomień (wysyłanie e-maili przypomnień)

3.2.2 Pozostałe katalogi

- **src/config/** – pliki konfiguracyjne Laravel (baza danych, cache, mail, autoryzacja)
- **src/database/** – migracje, seedy i fabryki bazy danych
- **src/public/** – publiczny katalog projektu (index.php, zasoby statyczne)
- **src/resources/** – zasoby frontendowe i widoki:
  - **views/** – szablony Blade (autoryzacja, panel użytkownika, formularze)
  - **css/** – główny plik CSS/SCSS importujący Bootstrap
  - **js/** – pliki JavaScript i Vue (komponenty, konfiguracja Vite)
- **src/routes/** – definicje tras (web.php dla tras webowych)
- **src/storage/** – pliki cache, logi i uploady użytkowników
- **src/tests/** – testy jednostkowe i funkcjonalne



Rysunek 2: Diagram klas systemu (opracowanie własne)

4 Użyte technologie i frameworki

Dokładne zestawienie wykorzystanych technologii przedstawia poniższa tabela:

Technologia	**Opis wykorzystania**
Laravel (backend)	Struktura MVC, Eloquent ORM
PHP	Wersja 8.3 w kontenerze Docker
Composer	Menedżer pakietów PHP
NPM	Node.js 18 + Vite 6.x do budowania frontendu
Bootstrap	Framework CSS do responsywnego UI
Vue.js	Interaktywne komponenty frontend (Vite, plugin Vue)
vue-i18n	Obsługa wielojęzyczności po stronie Vue
Leaflet	Wyświetlanie i interakcja z mapami
PostgreSQL	Relacyjna baza danych (kontener postgres:17-alpine)
Redis	Cache, sesje (kontener redis:7.4-alpine)
Laravel Sanctum	Uwierzytelnianie i autoryzacja API
Predis	Klient PHP do współpracy z Redis
Mailpit	Lokalny serwer SMTP do testów e-maili
Spatie ActivityLog	Logowanie aktywności użytkowników i modeli
Axios	Biblioteka HTTP do asynchronicznych wywołań API
Laravel Breeze	System autoryzacji (Blade + Bootstrap)
Docker Compose	Konfigurowanie środowiska wielokontenerowego

Tabela 1: Wykorzystane technologie i frameworki

Powyższe technologie zostały dobrane w celu kompleksowej realizacji funkcji webowej aplikacji. Laravel zapewnia solidne API i logikę serwera, Bootstrap i Vue odpowiadają za interaktywny, responsywny interfejs. Docker gwarantuje spójną konfigurację środowiska deweloperskiego.

5 Opis działania poszczególnych komponentów

5.1 Kontrolery (Controllers)

W katalogu `src/app/Http/Controllers/` znajdują się klasy obsługujące żądania HTTP:

5.1.1 Upsttrolery

- **EventController** – operacje CRUD dla wydarzeń, tworzenie i edycja wydarzeń z obsługą współdzielenia przejazdów, wyświetlanie feedu wydarzeń z filtrowaniem i paginacją
- **EventAttendeeController** – zarządzanie uczestnictwem w wydarzeniach, rejestracja uczestników, akceptacja/odrzucanie zgłoszeń, wyświetlanie listy uczestników dla organizatorów
- **UserAttendancesController** – wyświetlanie listy wydarzeń, w których użytkownik bierze udział
- **RideController** – operacje CRUD dla przejazdów, tworzenie ofert przejazdu dla wydarzeń z określeniem miejsca spotkania i liczby dostępnych miejsc
- **RideRequestController** – zarządzanie zgłoszeniami do przejazdów, składanie wniosków o przejazd, akceptacja/odrzucanie przez kierowców
- **ProfileController** – zarządzanie profilem użytkownika, edycja danych osobowych, zmiana zdjęcia profilowego, aktywacja 2FA, usuwanie konta
- **UserSettingsController** – ustawienia użytkownika, zmiana hasła, aktualizacja danych kontaktowych
- **BannedController** – obsługa zbanowanych użytkowników, wyświetlanie informacji o banie
- **AdminController** – panel administracyjny, zarządzanie użytkownikami (banowanie/odbanowywanie), nadawanie uprawnień administratora, przeglądanie statystyk
- **ReportController** – system zgłoszeń, moderacja zgłoszeń użytkowników, zatwierdzanie/odrzucanie raportów o niepożądanych zachowaniach
- **GeocodingController** – pobieranie adresów i lokalizacji za pomocą API Nominatim OpenStreetMap, wyszukiwanie miejsc i reverse geocoding
- **PhotoController** – zarządzanie zdjęciami wydarzeń, dodawanie i usuwanie zdjęć przez organizatorów
- **MainMapController** – wyświetlanie mapy z wydarzeniami, pobieranie danych o lokalizacjach wydarzeń
- **LanguageController** – zarządzanie językami interfejsu, przełączanie między dostępnymi lokalizacjami (pl, en, jpn, es, ua)

5.1.2 Kontrolery API

W katalogu `Api/` znajduje się zestaw kontrolerów udostępniających REST API pod prefiksem `/api`. Każdy kontroler obsługuje operacje CRUD oraz dodatkowe endpointy, zabezpieczone middlewarem `auth:sanctum`.

5.2 Kontlolery Auth

W katalogu `auth/` znajduje się zestaw kontrolerów dostarczonych przez Laravel Breeze, które umożliwiają kompletną obsługę uwierzytelniania użytkowników – rejestrację, logowanie, weryfikację email, resetowanie haseł oraz potwierdzanie tożsamości dla operacji wymagających dodatkowego bezpieczeństwa.

5.3 Modele (Models)

W katalogu `src/app/Models/` znajdują się klasy Eloquent odpowiadające tabelom bazy danych:

- **User** – model użytkownika z funkcją powiadomień
- **Event** – model wydarzenia
- **EventAttendee** – model uczestnictwa w wydarzeniu
- **Ride** – model oferty dojazdu
- **RideRequest** – model prośby o dojazd
- **Photo** – model zdjęć
- **Report** – model zgłoszeń użytkowników

Modele zawierają relacje i metody pomocnicze, wykorzystując Eloquent ORM do łatwego wykonywania operacji na bazie danych.

5.4 System tras (Routing)

5.4.1 Trasy webowe (web.php)

Generują widoki HTML, obejmują:

- logowanie i rejestrację,
- listę i widok pojedynczego wydarzenia,
- panel użytkownika,
- strony statyczne („O nas”, „Pomoc”).

5.4.2 Trasy API (api.php)

REST API z prefiksem `/api`, zawierające publiczne punkty końcowe do rejestracji i logowania. Pozostałe trasy zabezpieczone middlewarem `auth:sanctum`.

5.5 Widoki i szablony (Blade Templates)

W katalogu `resources/views/` znajdują się pliki Blade:

- **Autoryzacja** – formularze logowania, rejestracji, resetu hasła
- **Wydarzenia** – lista, tworzenie, edycja wydarzeń
- **Komponenty pomocnicze** – navbar, footer
- **Strony dodatkowe** – centrum pomocy, widok mapy

5.6 Komponenty Vue.js

W katalogu `resources/js/components/` znajdują się komponenty:

- **LeafletMap.vue** – interaktywna mapa do przeglądania wydarzeń z markerami, wyszukiwaniem miejsc, geolokalizacją i popup'ami ze szczegółami eventów
- **EventForm.vue** – kompleksowy formularz tworzenia wydarzeń z dwoma mapami (lokalizacja wydarzenia i miejsce spotkania dla car sharing), wyszukiwaniem adresów, upload'em zdjęć i geokodowaniem

5.7 Cache i kolejki

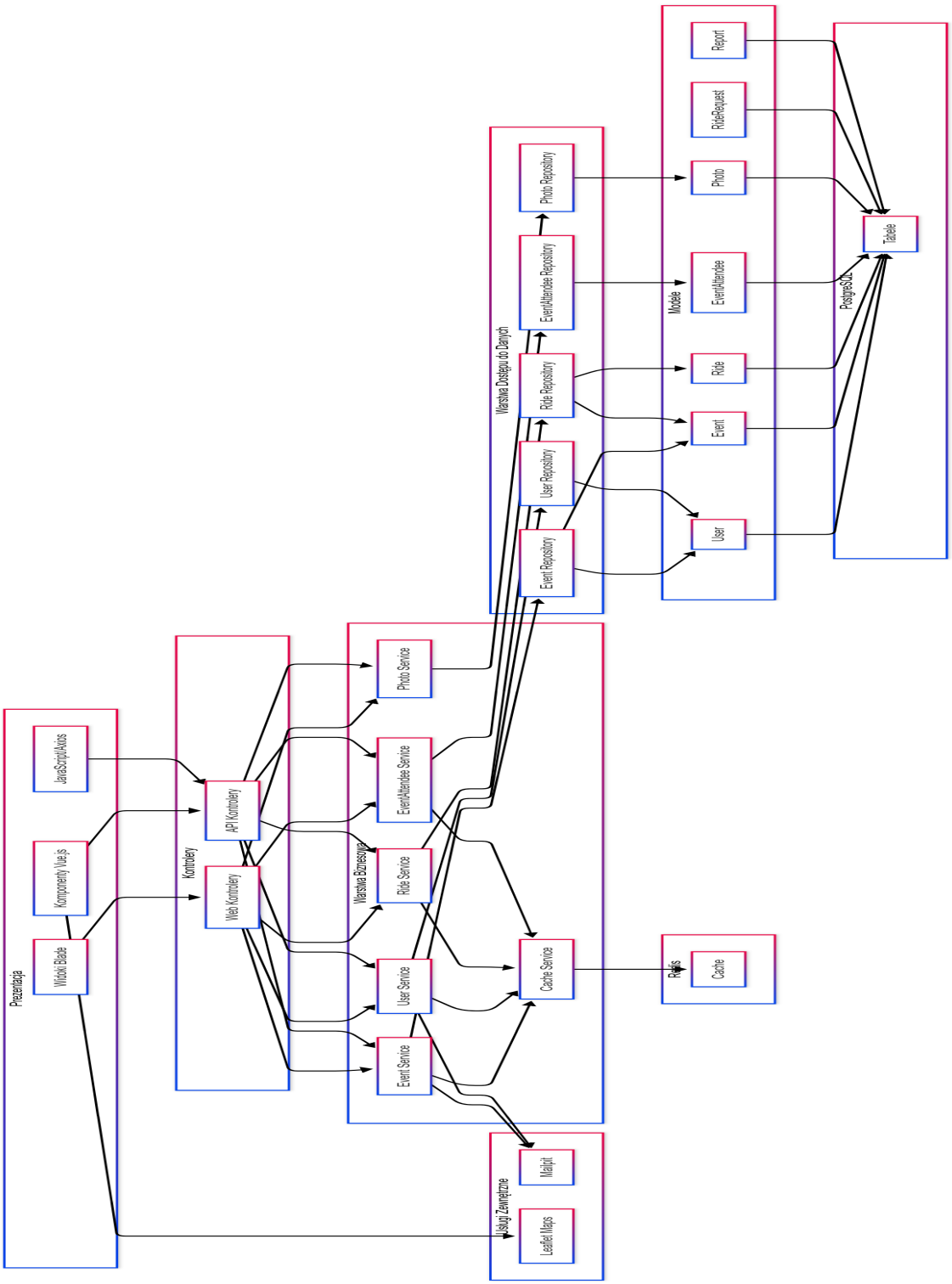
System wykorzystuje Redis jako domyślny store dla cache'u, sesji i kolejek. Powiadomienia e-mail są kolejkowane, a worker'y Supervisor obsługują kolejkę w tle.

5.8 System logowania

Oprócz standardowego logowania Laravel, projekt wykorzystuje pakiet Spatie Activitylog, który zapisuje aktywność użytkowników do osobnej tabeli. Umożliwia to śledzenie zdarzeń w systemie (np. tworzenie/usuwanie obiektów).

5.9 System mailingu

Konfiguracja SMTP wysyła e-maile do serwera Mailpit, który udostępnia interfejs webowy do podglądu wiadomości. Wykorzystano mechanizm powiadomień Laravel (np. EventReminderNotification).



Rysunek 3: Diagram komponentów (opracowanie własne)

6 Konfiguracja Docker, bazy danych i zależności

Projekt wykorzystuje Docker Compose z następującymi usługami:

6.1 Usługa app

Główny kontener z aplikacją Laravel, zbudowany na bazie obrazu PHP. Instalowane są dodatkowe rozszerzenia PHP (gd, pgsql, zip, redis, xdebug), Nginx i Supervisor. Port 80 mapowany na hosta (domyślnie 63851).

6.2 Usługa database

Kontener PostgreSQL 17 z trwałymi danymi dzięki wolumenowi `faily-postgres-data`. Port 5432 mapowany na hosta (domyślnie 63853).

6.3 Usługa redis

Kontener Redis używany jako cache i do przechowywania sesji Laravel. Dane trwałe dzięki wolumenowi `faily-redis-data`. Port 6379 mapowany na hosta (domyślnie 63852).

6.4 Usługa mailpit

Testowy serwer SMTP z interfejsem webowym na porcie 8025 (domyślnie host 63854). Aplikacja Laravel wysyła e-maile na port SMTP Mailpit (1025).

Plik `compose.yaml` definiuje wszystkie usługi oraz sieć `faily-dev`. Kontener aplikacji zależy od bazy danych i sprawdza jej stan przed startem.

Rysunek 4: Diagram ERD projektu Faily (opracowanie własne)

7 Analiza wdrożonych systemów

7.1 Założenia projektowe

Projekt realizuje następujące założenia techniczne:

1. **Framework MVC** – Aplikacja wykorzystuje Laravel 12 implementujący wzorzec Model-View-Controller.
2. **Framework CSS** – Bootstrap 5 zapewnia responsywny i spójny wygląd interfejsu.
3. **Baza danych** – PostgreSQL 17 jako relacyjna baza danych z dostępem przez Eloquent ORM.
4. **Cache** – Redis jako pamięć podręczna, sesje i kolejki.
5. **Dependency Manager** – Composer dla backendu (PHP), NPM dla frontendu (JavaScript).
6. **HTML** – Blade do generowania widoków HTML z wbudowanymi zmiennymi Laravel.
7. **CSS** – Bootstrap 5 uzupełniony własnymi stylami w `app.css`.
8. **JavaScript** – Vue.js jako główny framework z komponentami i Axios do komunikacji API.
9. **Routing** – Laravel dla tras serwera (web.php)
10. **ORM** – Eloquent ORM mapujący tabele na modele PHP z relacjami.
11. **Uwierzytelnianie** – Laravel Breeze + Laravel Sanctum dla API.
12. **lokalizacja** – System tłumaczeń Laravel z folderami `resources/lang/` oraz wykożustanie i18n dla tłumaczeń elementów Vue.
13. **Mailing** – Powiadomienia Laravel z integracją Mailpit do testów.
14. **Formularze** – Formularze HTML Blade z tokenami CSRF i walidacją.
15. **Interakcje Asynchroniczne** – Vue.js + Axios do operacji bez przeładowania strony.
16. **Konsumowanie API** – Integracja z Nominatim OSM do geokodowania.
17. **Publikacja API** – REST API pod prefiksem `/api` z kontrolerami CRUD.
18. **RWD** – Bootstrap + meta viewport dla wszystkich urządzeń.
19. **Logger** – Laravel logs + Spatie Activitylog do audytu.
20. **Deployment** – Wdrożenie zostało zrealizowane z wykorzystaniem Docker Compose, z przygotowanymi skryptami oraz szczegółową instrukcją zawartą w dokumentacji. Aplikacja została wdrożona w środowisku produkcyjnym w chmurze Oracle Cloud, z użyciem Laravel, Vue, Nginx oraz innych niezbędnych komponentów. Dodatkowo, w ramach wdrożenia wykorzystano: darmową wersję Upstash dla usługi Redis, darmową wersję Supabase opartą na PostgreSQL oraz bezpłatną wersję Brevo do obsługi wysyłki wiadomości e-mail.

8 Przebieg działania aplikacji

8.1 Przykładowe scenariusze użytkownika

8.1.1 Rejestracja i logowanie

Użytkownik wypełnia formularz rejestracji, Laravel tworzy nowego użytkownika i wysyła e-mail potwierdzający. Po potwierdzeniu można się zalogować – sesja jest utrzymywana po stronie serwera, a token Sanctum przyznawany przy logowaniu mobilnym.

8.1.2 Przeglądanie wydarzeń

Na stronie /event$_l istuytkownikwidzilistnadchodzcychwydarzewpostacikartBootstrap.Moefiltrowaw$

8.1.3 Tworzenie wydarzeń

Zalogowany użytkownik przechodzi do formularza tworzenia wydarzeń. Dzięki komponentowi mapy może wpisać adres i otrzymać sugestię lokalizacji (geokodowanie). Po zapisaniu dane trafiają do bazy, a autor może edytować wydarzenie.

8.1.4 Uczestnictwo w wydarzeniu

Na stronie wydarzenia znajduje się przycisk zgłoszenia udziału. Po wciśnięciu tworzony jest wpis w bazie. Organizator może zaakceptować lub odrzucić zgłoszenie przez interfejs aplikacji.

8.1.5 Prośby o wspólny dojazd

Użytkownik może przeglądać oferty przejazdów lub utworzyć własną. W widoku wydarzenia znajduje się sekcja „Dojazd"z formularzem. Po wpisaniu punktów początkowego i docelowego użytkownik wysyła prośbę. Właściciel może zaakceptować lub odrzucić prośbę.

8.1.6 Panel administracyjny

Przygotowano folder `resources/views/admin/` z plikami panelu administratora do zarządzania użytkownikami i eventami.

8.1.7 Zmiana języka

W navbarze znajduje się przełącznik języka. Po wyborze interfejs przeładowuje się z odpowiednimi tłumaczeniami, a Vue przełącza język komponentów.

9 Uruchomienie aplikacji lokalnielokalne

9.1 Wymagania wstępne

Do uruchomienia aplikacji wymagane są:

- Docker
- Docker Compose
- Git

9.2 Proces instalacji

9.2.1 Klonowanie repozytorium

```
git clone https://github.com/mateusz-bogacz-collegiumwitelona/Faily/
cd Faily
```

Listing 1: Pobranie kodu źródłowego

9.2.2 Przygotowanie środowiska

Upewnij się, że porty `63851`, `63852`, `63853`, `63854` oraz `5173` są dostępne. Następnie zmień nazwy plików `.env.example` na `.env` w katalogu głównym projektu oraz w katalogu **src**. Po wykonaniu tych czynności można przystąpić do budowania obrazu Dockera:

```
docker-compose build
docker-compose up
```

Listing 2: Uruchomienie kontenerów

9.2.3 Konfiguracja projektu

Opcja automatyczna (skrypt):

```
docker exec -it faily-app-dev bash
chmod +x deploy.sh
./deploy.sh
```

Listing 3: Użycie skryptu konfiguracyjnego

Opcja manualna:

```
docker exec -it faily-app-dev bash
composer install
npm install
npm run build
php artisan migrate
php artisan key:generate
php artisan db:seed
php artisan storage:link
```

Listing 4: Konfiguracja manualna

9.3 Dostęp do aplikacji

Po zakończeniu instalacji aplikacja będzie dostępna pod adresami:

- **Aplikacja Laravel**: `http://localhost:63851`
- **Panel Mailpit**: `http://localhost:63854`

10 Wdrożenie aplikacji na serwerze

10.1 Wprowadzenie

Instrukcja opisuje proces wdrożenia aplikacji Laravel+Vue.js na serwerze VPS z Ubuntu, wykorzystujący zewnętrzne usługi:

- **Supabase** – baza danych PostgreSQL
- **Upstash** – cache Redis
- **Brevo** – wysyłanie e-maili

10.2 Wymagania

- Serwer VPS z Ubuntu 24.04 (w projekcie: Oracle Cloud – 1 rdzeń, 2 wątki, 16GB RAM, 50GB dysk)
- Dostęp SSH do serwera
- Skonfigurowane domeny (w projekcie: faily.pl i faily.online z home.pl)
- Konta w zewnętrznych usługach: Supabase, Upstash, Brevo

10.3 Przygotowanie środowiska serwerowego

10.3.1 Aktualizacja systemu

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y curl git unzip nginx supervisor certbot python3-certbot-nginx
```

Listing 5: Aktualizacja Ubuntu i instalacja pakietów

10.3.2 Instalacja PHP 8.3

```
sudo apt install -y software-properties-common
sudo add-apt-repository ppa:ondrej/php -y
sudo apt update

sudo apt install -y php8.3-fpm php8.3-cli php8.3-common \
php8.3-pgsql php8.3-mbstring php8.3-xml php8.3-zip \
php8.3-bcmath php8.3-gd php8.3-curl php8.3-redis
```

Listing 6: Instalacja PHP i rozszerzeń

10.3.3 Instalacja Node.js

```
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs
```

Listing 7: Instalacja Node.js 18.x

10.3.4 Instalacja Composer

```
curl -sS https://getcomposer.org/installer | sudo php -- \
    --install-dir=/usr/local/bin --filename=composer
```

Listing 8: Instalacja Composer

10.3.5 Pobieranie kodu i ustawienie uprawnień

```
sudo mkdir /var/www/faily
cd /var/www/faily
git clone https://github.com/mateusz-bogacz-collegiumwitelona/Faily/ .

sudo mkdir -p storage/framework/{views,cache,sessions}

sudo chown ubuntu:www-data /var/www/faily
sudo chown -R www-data:www-data storage bootstrap/cache

sudo chmod 755 /var/www/faily
sudo chmod -R 775 storage bootstrap/cache
```

Listing 9: Przygotowanie katalogów i uprawnień

10.3.6 Konfiguracja Nginx

```
sudo nano /etc/nginx/sites-available/faily
```

Listing 10: Utworzenie konfiguracji Nginx

Zawartość pliku konfiguracyjnego:

```
server {
    listen 80;
    server_name faily.pl www.faily.pl faily.online www.faily.online;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name faily.pl www.faily.pl faily.online www.faily.online;

    ssl_certificate /etc/letsencrypt/live/faily.online/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/faily.online/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    root /var/www/faily/src/public;
    index index.php index.html;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/var/run/php/php8.3-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.(?!well-known).* {
        deny all;
    }
}
```

Listing 11: Konfiguracja Nginx z SSL

Aktywacja konfiguracji:

```
sudo ln -s /etc/nginx/sites-available/faily /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

Listing 12: Aktywacja konfiguracji Nginx

10.3.7 Konfiguracja PHP-FPM

Utworzenie dedykowanego pool dla aplikacji:

```
sudo nano /etc/php/8.3/fpm/pool.d/faily.conf
```

Listing 13: Konfiguracja PHP-FPM pool

Zawartość pliku konfiguracyjnego:

```
[faily]
user = www-data
group = www-data
listen = /var/run/php/php8.3-fpm.sock
listen.owner = www-data
listen.group = www-data
pm = dynamic
pm.max_children = 10
pm.start_servers = 3
pm.min_spare_servers = 2
pm.max_spare_servers = 5
pm.process_idle_timeout = 10s
pm.max_requests = 500
```

Listing 14: Konfiguracja pool PHP-FPM

Optymalizacja ustawień PHP:

```
sudo nano /etc/php/8.3/fpm/conf.d/99-laravel.ini
```

Listing 15: Optymalizacja PHP

```
upload_max_filesize = 100M
post_max_size = 100M
max_execution_time = 300
memory_limit = 512M
```

Listing 16: Ustawienia optymalizacji

Restart usługi PHP:

```
sudo systemctl restart php8.3-fpm
```

10.3.8 Rekordy DNS

W panelu zarządzania domeną należy dodać następujące rekordy:

Typ	**Nazwa**	**Wartość**
A	@	Adres IP serwera VPS
A	www	Adres IP serwera VPS

Tabela 2: Rekordy DNS do konfiguracji

10.3.9 Certyfikat SSL

Sprawdzenie propagacji DNS:

```
dig faily.pl
dig www.faily.pl
dig faily.online
dig www.faily.online
```

Listing 17: Weryfikacja DNS

Uzyskanie certyfikatu SSL:

```
sudo systemctl stop nginx
sudo certbot certonly --standalone \
    -d faily.pl -d www.faily.pl \
    -d faily.online -d www.faily.online
sudo systemctl start nginx
```

Listing 18: Instalacja certyfikatu Let's Encrypt

10.3.10 Instalacja zależności aplikacji

```
cd /var/www/faily/src
composer install --no-dev --optimize-autoloader
npm install
npm run build
```

Listing 19: Instalacja zależności

10.3.11 Przygotowanie pliku .env

```
cp .env.example .env
php artisan key:generate
```

Listing 20: Konfiguracja środowiska

Przykładowa konfiguracja .env

```
APP_NAME=Faily
APP_ENV=production
APP_KEY=base64:wygenerowany_klucz
APP_DEBUG=false
APP_URL=https://faily.pl

APP_LOCALE=pl
APP_FALLBACK_LOCALE=en

# Konfiguracja bazy danych (Supabase)
DB_CONNECTION=pgsql
DB_HOST=host_podany_w_supabase
DB_PORT=port_podany_w_supabase
DB_DATABASE=nazwa_bazy_supabase
DB_USERNAME=uzytkownik_supabase
DB_PASSWORD=haslo_supabase

# Konfiguracja Redis (Upstash)
REDIS_CLIENT=predis
REDIS_HOST=host_podany_przez_upstash
REDIS_PASSWORD=haslo_podane_przez_upstash
REDIS_PORT=port_podany_przez_upstash
CACHE_DRIVER=redis
SESSION_DRIVER=redis
QUEUE_CONNECTION=redis

# Konfiguracja poczty (Brevo)
MAIL_MAILER=smtp
MAIL_HOST=host_podany_przez_brevo
MAIL_PORT=port_podany_przez_brevo
MAIL_USERNAME=nazwa_uzytkownika_brevo
MAIL_PASSWORD=haslo_brevo
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=team@faily.pl
MAIL_FROM_NAME=Faily
```

Listing 21: Plik .env dla produkcji

10.3.12 Konfiguracja Supervisor dla kolejek

Utworzenie konfiguracji worker'a:

```
sudo nano /etc/supervisor/conf.d/faily-worker.conf
```

Listing 22: Konfiguracja Supervisor

```
[program:faily-worker]
process_name=%(program_name)s_%(process_num)02d
command=php /var/www/faily/src/artisan queue:work --sleep=3 --tries=3
autostart=true
autorestart=true
user=www-data
numprocs=2
redirect_stderr=true
stdout_logfile=/var/log/faily-worker.log
stopwaitsecs=3600
```

Listing 23: Konfiguracja Laravel worker

Aktywacja Supervisor:

```
sudo supervisorctl reread
sudo supervisorctl update
sudo supervisorctl start faily-worker:*
```

Listing 24: Uruchomienie worker'a

10.3.13 Optymalizacja aplikacji

```
php artisan migrate --force
php artisan optimize
php artisan config:cache
php artisan route:cache
php artisan view:cache
php artisan storage:link
```

Listing 25: Optymalizacja Laravel

10.3.14 Weryfikacja działania

```
sudo systemctl status nginx php8.3-fpm
curl https://faily.pl
```

Listing 26: Sprawdzenie statusu usług

10.4 Najczęstsze problemy i rozwiązania

- **10.4.1 Błędy uprawnień**

  Sprawdź uprawnienia katalogów `storage` i `bootstrap/cache`:

  ```
  sudo chown -R www-data:www-data storage bootstrap/cache
  sudo chmod -R 775 storage bootstrap/cache
  ```

- **10.4.2 Błędy SSL**

  Upewnij się, że porty 80 i 443 są otwarte. Sprawdź ustawienia VNIC lub firewall:

  ```
  sudo ufw allow 80
  sudo ufw allow 443
  ```

- **10.4.3 Błędy bazy danych**

  Zweryfikuj konfigurację Supabase w pliku `.env` oraz połączenie:

  ```
  php artisan tinker
  DB::connection()->getPdo();
  ```

- **10.4.4 Błędy cache**

  Sprawdź konfigurację Upstash Redis:

  ```
  php artisan cache:clear
  php artisan config:clear
  ```

- **10.4.5 Błędy e-mail**

  Zweryfikuj konfigurację Brevo i domenę nadawcy w pliku `.env`.

10.5 Monitoring i logowanie

Logi aplikacji znajdują się w:

- `/var/www/faily/src/storage/logs/` – logi Laravel
- `/var/log/nginx/` – logi Nginx
- `/var/log/faily-worker.log` – logi kolejek

11 Podsumowanie

Projekt Faily stanowi kompleksową aplikację webową do zarządzania wydarzeniami, zrealizowaną z wykorzystaniem nowoczesnych technologii i najlepszych praktyk programistycznych.

11.1 Osiągnięte cele

Aplikacja pomyślnie realizuje wszystkie założone cele:

- **Zarządzanie wydarzeniami** – pełny cykl życia eventów
- **System użytkowników** – rejestracja, autoryzacja, profile
- **Interakcje społeczne** – uczestnictwo, wspólne dojazdy
- **Geolokalizacja** – integracja z mapami i API geokodowania
- **Wielojęzyczność** – obsługa multiple języków
- **Responsywność** – działanie na różnych urządzeniach

11.2 Wartości edukacyjne

Projekt dostarczył zespołowi cennych doświadczeń w zakresie:

- pracy z frameworkiem Laravel i ekosystemem PHP,
- integracji frontendu (Vue.js) z backendem,
- konfiguracji środowiska Docker,
- wdrażania aplikacji na serwerze produkcyjnym,
- pracy zespołowej nad złożonym projektem.

11.3 Możliwości rozwoju

Aplikacja stanowi solidną podstawę do dalszego rozwoju. Potencjalne usprawnienia obejmują:

- implementację aplikacji mobilnej,
- rozszerzenie funkcji społecznościowych,
- integrację z zewnętrznymi serwisami społecznościowymi,
- zaawansowaną analitykę użytkowników,
- dodanie możliwości ocenianie wydarzeń i użytkowników,

Dzięki wykorzystaniu sprawdzonych technologii i architektury MVC, aplikacja jest przygotowana na skalowanie i dalszy rozwój funkcjonalności.

Spis rysunków

Spis kodów źródłowych