Sprawozdanie z projektu zespołowego

Strava Dream Team Edition (SDTE)

Przedmiot:	Projekt zespołowy Programowanie urządzeń mobilnych	
Prowadzący:	mgr inż. Krzysztof Rewak mgr inż. Kamil Piech	
Autorzy projektu:	Dominik Gwóźdź	43189
	Kamil Rudowski	43256
	Dawid Brożyna	42719

Data dokumentu: 18 stycznia 2026

Spis treści

1 Opis przedmiotu zamówienia

Uwaga: Poniższa treść rozdziału została dostarczona przez prowadzącego w ramach OPZ i nie stanowi autorskiego opisu zespołu.

Przedmiotem zamówienia jest system składający się z:

- aplikacji mobilnej (Android lub iOS),
- panelu administracyjnego dostępnego przez przeglądarkę,
- interfejsu REST API komunikującego się z bazą danych.

System służy do rejestrowania i analizy aktywności fizycznych użytkowników (takich jak bieganie, jazda na rowerze, marsz), z wykorzystaniem danych GPS urządzenia mobilnego.

1.1 Wymagania dotyczące aplikacji mobilnej

1. Użytkownik może pobrać aplikację na systemy Android lub iOS.

2. Użytkownik może zarejestrować konto i zalogować się w aplikacji.

3. Użytkownik może zresetować swoje hasło (np. gdy zapomni hasła).

4. Użytkownik po zalogowaniu może ustawić swoje dane profilowe takie jak: imię, nazwisko, data urodzenia, płeć, wzrost, waga i avatar.

5. Po zalogowaniu użytkownik może rozpocząć nową aktywność, pauzować i zakończyć ją oraz zapisać dane.

6. W trakcie aktywności aplikacja rejestruje czas, dystans, tempo, prędkość i ślad GPS.

7. Po zakończeniu aktywności użytkownik może nadać nazwę, dodać notatkę, zdjęcie i oznaczyć typ aktywności (rower, bieg, spacer).

8. Historia aktywności wyświetlana jest w formie listy.

9. Kliknięcie na aktywność otwiera szczegóły (trasa, czas, tempo, notatki, zdjęcie).

10. Użytkownik może filtrować i sortować swoje aktywności (np. po dacie, dystansie, typie).

11. Użytkownik może przeglądać zbiorczą listę treningów znajomych na stronie głównej - feed.

12. Aplikacja wyświetla podstawowe statystyki użytkownika (liczba treningów, łączny dystans, średnia prędkość).

13. Aplikacja obsługuje ranking użytkowników (np. sumaryczny dystans tygodniowy).

14. Aplikacja umożliwia synchronizację danych z serwerem (po zalogowaniu).

15. Aplikacja mobilna korzysta z REST API opisanym w punkcie 4.

16. Interfejs aplikacji dostępny jest w języku polskim i angielskim.

17. Wysyłanie i akceptowanie zaproszeń do znajomych.

18. Możliwość zablokowania użytkownika i/lub zgłoszenia nadużycia.

19. „Polubienia” (kudos) i proste komentarze pod zakończonym treningiem.

20. Powiadomienia typu PUSH dla nowych zaproszeń, komentarzy/polubień pod zakończonym treningiem.

1.2 Wymagania dotyczące panelu administracyjnego

21. Administrator może zalogować się do panelu webowego.

22. Panel umożliwia przeglądanie listy użytkowników i aktywności.

23. Administrator może filtrować (po użytkowniku, dacie dodania, długości trasy, typie aktywności), wyszukiwać i usuwać aktywności.

24. Administrator widzi globalne statystyki (liczba użytkowników, aktywności, łączny dystans).

1.3 Wymagania dotyczące API i backendu

25. Backend udostępnia REST API zgodne ze specyfikacją OpenAPI 3.0.

26. Dokumentacja API dostępna jest pod endpointem `/api/documentation` (OpenAPI-/Swagger).

27. API umożliwia eksport aktywności do pliku `.gpx`.

1.4 Dane testowe i konfiguracja początkowa

Po wdrożeniu systemu baza danych zawiera przykładowych 5 użytkowników oraz przykładowe aktywności testowe. Konto administratora tworzone jest przy pierwszym uruchomieniu systemu, a hasło ustalane przy pierwszym logowaniu.

1.5 Wymagania niefunkcjonalne

28. Interfejs przyjazny użytkownikowi (UX mobilny).

29. Komunikacja API zabezpieczona przez HTTPS.

30. Aplikacja mobilna działa w tle podczas aktywności.

1.6 Rozszerzenia opcjonalne

31. Eksport danych użytkownika do pliku CSV.

32. Tryb offline (synchronizacja po odzyskaniu sieci).

33. Funkcje AI (np. podsumowanie treningu).

2 Opis technologiczny

Projekt podzielony został na trzy główne komponenty:

1. **API (`C#, .NET`)**

   Repozytorium: `<https://github.com/dominog125/Strava_DreamTeam_Edition/tree/main/API>`

   Komponent odpowiedzialny za przechowywanie i udostępnianie danych, autoryzację użytkowników oraz obsługę endpointów dla aplikacji mobilnej i panelu webowego.

2. **Web Panel (PHP, Laravel)**

   Repozytorium: `<https://github.com/dominog125/Strava_DreamTeam_Edition/tree/main/AdminPanel>`

   Panel dostępny w przeglądarce, zawierający panel administracyjny do przeglądania użytkowników i aktywności, filtrowania danych oraz wykonywania operacji administracyjnych.

3. **Mobile (`<Dart, Flutter>`)**

   Repozytorium: `<https://github.com/dominog125/Strava_DreamTeam_Edition/tree/main/Mobile>`

   Aplikacja mobilna służąca do rejestrowania aktywności z użyciem GPS, prezentacji historii treningów i statystyk oraz synchronizacji danych z API.

3 Podział obowiązków i odpowiedzialności w zespole

W celu utrzymania przejrzystości prac i uniknięcia nakładania się zadań zdefiniowano role oraz zakres odpowiedzialności członków zespołu. Odpowiedzialności przypisano zgodnie z macierzą RACI (Responsible, Accountable, Consulted, Informed).

4 Podział obowiązków i odpowiedzialności

- **`Dominik Gwóźdź`** – backend
- **`Kamil Rudowski`** – web panel, skryba
- **`Dawid Brożyna`** – aplikacja mobilna

5 Rozpisane zadania zespołu wraz z przedstawioną czasochłonnością

Poniższa tabela przedstawia zadania z backlogu projektu oraz przybliżoną czasochłonność prac (w godzinach).

ID	Zadanie	Komponent	Estymata	Rzeczyw.
#1	Login Screen (Admin Panel)	`Web`	4h	4h
#2	Login Form (Admin Panel)	`Web`	6h	7h
#3	Database Structure	`API`	10h	12h
#4	Login Form (Mobile)	`Mobile`	8h	9h
#5	Login Screen (Mobile)	`Mobile`	6h	6h

ID	Zadanie	Komponent	Estymata	Rzeczyw.
#6	API Login	`API`	6h	6h
#7	API Registration	`API`	8h	8h
#10	Register with Email	`Mobile`	8h	9h
#13	Login with Email	`Mobile`	6h	6h
#15	Central API Integration	`Mobile`	12h	14h
#16	User Profile Setup	`Mobile`	12h	14h
#17	Start New Activity	`Mobile`	16h	18h
#18	Pause Activity	`Mobile`	4h	4h
#19	Finish & Save Activity	`Mobile`	12h	14h
#20	Activity Details on Finish	`Mobile`	8h	8h
#21	Sort Activities	`Mobile`	5h	5h
#22	Filter Activities	`Mobile`	6h	7h
#24	Friend requests + block/report user	`Mobile`	18h	20h
#25	Likes (kudos) & comments	`Mobile`	16h	18h
#26	Push notifications	`Mobile`	12h	14h
#27	Admin: user & activity list	`Web`	12h	14h
#28	Admin filters	`Web`	8h	9h
#29	Admin search	`Web`	4h	4h
#30	Admin delete activity	`Web`	4h	5h
#31	Global statistics view	`Web`	6h	7h
#32	Export user data to CSV	`API`	8h	10h
#35	Admin Panel PL/ENG	`Web`	5h	5h
#36	Friends feed	`Mobile`	10h	12h

6 Instrukcja uruchomienia systemu

6.1 Wymagania wstępne

Do uruchomienia systemu lokalnie wymagane są:

- Docker + Docker Compose (dla Web Panelu w trybie Sail oraz wdrożenia zdalnego),
- .NET SDK (dla uruchomienia API lokalnie bez Dockera),
- Flutter SDK (dla uruchomienia aplikacji mobilnej).

6.2 Uruchomienie lokalne

6.2.1 API (C#, .NET)

1. Sklonuj repozytorium API i przejdź do katalogu projektu.
2. Skonfiguruj zmienne środowiskowe / plik `appsettings.*` (np. connection string do bazy danych), zgodnie z repozytorium API.
3. Uruchom API:

```
dotnet restore
dotnet run
```

4. Po uruchomieniu API dokumentacja Swagger jest dostępna pod `/api/documentation`.

6.2.2 Web Panel (Laravel Sail)

1. Sklonuj repozytorium Web i przejdź do katalogu projektu.
2. Skopiuj plik konfiguracyjny:

```
cp .env.example .env
```

3. Ustaw w `.env` kluczowe wartości (przykład):

```
APP_ENV=local
APP_DEBUG=true

ADMINISTRATION_DATA_SOURCE=api
ADMINISTRATION_AUTH_MODE=jwt
ADMINISTRATION_API_BASE_URL=<URL_DO_API>
ADMINISTRATION_API_TIMEOUT_SECONDS=10
```

4. Uruchom kontenery Sail:

```
./vendor/bin/sail up -d
```

5. Wygeneruj klucz aplikacji i wyczyść cache (w kontenerze):

```
./vendor/bin/sail artisan key:generate
./vendor/bin/sail artisan config:clear
./vendor/bin/sail artisan view:clear
```

6. Zainstaluj zależności frontendu i uruchom Vite:

```
./vendor/bin/sail npm install
./vendor/bin/sail npm run dev
```

6.2.3 Mobile (Dart, Flutter)

1. Sklonuj repozytorium Mobile i przejdź do katalogu projektu.
2. Pobierz zależności:

```
flutter pub get
```

3. Uruchom aplikację na emulatorze/urządzeniu:

```
flutter run
```

4. W konfiguracji aplikacji ustaw adres bazowy API (`<URL_DO_API>`) zgodnie z wymaganiami repozytorium Mobile.

6.3 Uruchomienie zdalne

6.3.1 API (C#, .NET)

1. Na serwerze skonfiguruj zmienne środowiskowe (np. connection string do bazy, port, tryb środowiska).
2. Zbuduj paczkę produkcyjną:

```
dotnet publish -c Release -o out
```

3. Uruchom usługę (np. jako `systemd`) lub w kontenerze Docker zgodnie z konfiguracją serwera.

6.3.2 Web Panel (Docker Compose, `compose.production.yaml`)

1. Wgraj projekt na serwer i skonfiguruj `.env` (produkcyjne wartości `APP_ENV`, `APP_DEBUG=false`, `APP_URL`, oraz ustawienia `ADMINISTRATION_*`).
2. Uruchom usługi produkcyjne:

```
docker compose -f compose.production.yaml up -d --build
```

3. Wykonaj cache konfiguracji i widoków (w kontenerze aplikacji):

```
php artisan key:generate
php artisan config:cache
php artisan view:cache
```

4. Wystaw Web Panel przez reverse proxy (np. Nginx) oraz skonfiguruj HTTPS (np. certyfikat Let's Encrypt).

6.3.3 Mobile (wydanie aplikacji)

1. Zbuduj paczkę produkcyjną Android:

```
flutter build apk --release
```

2. (Opcjonalnie) App Bundle dla Google Play:

```
flutter build appbundle --release
```

3. Dla iOS przygotuj build release zgodnie z procesem publikacji (Xcode/TestFlight/App Store).

7 Wnioski z pracy projektowej

7.1 Co udało się zrealizować

W trakcie prac udało się zbudować spójny szkielet systemu z podziałem na trzy komponenty: API, Web Panel oraz Mobile, wraz z podstawową integracją pomiędzy nimi. Dostarczono kluczowe funkcje potrzebne do demonstracji przepływu end-to-end, w tym

logowanie oraz wybrane widoki list (z paginacją i filtrowaniem po stronie panelu administracyjnego), a także podstawowe scenariusze obsługi aktywności w aplikacji mobilnej. Zespół utrzymał porządek w zadaniach backlogu i konsekwentnie iterował po funkcjonalnościach, co ułatwiało planowanie i kontrolę postępu.

7.2 Co nie zostało zrealizowane lub było ograniczone

Nie wszystkie elementy backlogu udało się domknąć w planowanym czasie, a część funkcji została przesunięta do dalszych iteracji (np. elementy społecznościowe, powiadomienia PUSH, eksport danych lub rozszerzenia administracyjne). Wystąpiły także ograniczenia integracyjne wynikające z różnic w kontraktach danych pomiędzy komponentami (np. brak niektórych pól w jednym z endpointów), co wymusiło obejścia i dodatkowe mapowania po stronie Web Panelu.

7.3 Co można było zrobić lepiej

W kolejnej iteracji warto wcześniej ustalić i „zamrozić" kontrakty API oraz dodać proste testy kontraktowe/integracyjne, żeby szybciej wykrywać rozjazdy między API i panelem. Następnym usprawnieniem powinno być ujednolicenie strategii obsługi błędów i ograniczenie logowania do minimum potrzebnego do utrzymania, zgodnie z podejściem „blameless".

Dominik Gwóźdź

Nauczyłem się projektować REST API w .NET z opisem w OpenAPI/Swagger oraz utrzymywać spójne kontrakty danych pod aplikację mobilną i panel web. Najtrudniejsze było dopięcie autoryzacji i stabilnej integracji między komponentami, zwłaszcza gdy zmieniały się pola w DTO. Następnym razem wcześniej „zamroziłbym" kontrakty i dopiął testy kontraktowe oraz eksport GPX jako standardowy scenariusz

Kamil Rudowski

W web panelu nauczyłem się projektować warstwę dostępu do danych przez interfejsy i wstrzykiwanie zależności, tak żeby logika panelu nie była przywiązana do jednego źródła danych i dało się łatwo przełączać implementację między API a bazą danych. Najtrudniejsze było utrzymanie spójnych kontraktów i obsługi błędów

Dawid Brożyna

Nauczyłem się w Flutterze rejestrować aktywności z GPS i synchronizować je z API, uwzględniając uprawnienia i działanie w tle. Najtrudniejsze były ograniczenia pracy w tle i wpływ na baterię, szczególnie na iOS i nowszych Androidach. Następnym razem od razu zaplanowałbym tryb offline i strategię aktualizacji lokalizacji, żeby ograniczyć zużycie energii i liczbę requestów.