Imię: Kacper

Nazwisko: Kosuń
Numer indeksu:
48955

NOWATORSKI PROJEKT ZESPOŁOWY

Aplikacja do treningu pamięci

Link do repozytorium GitHub:
https://github.com/Kacpay/Fullstack_Projekt

Założenia projektowe

Cel główny

Celem projektu jest zaprojektowanie i implementacja mobilnej aplikacji wspierającej trening pamięci roboczej, opartej na zadaniu typu **dual n-back**. Projekt ma dostarczyć użytkownikom narzędzia umożliwiającego regularne wykonywanie ćwiczeń poznawczych, śledzenie postępów oraz porównywanie wyników z innymi użytkownikami.

Aplikacja ma pełnić funkcję zarówno praktyczną (codzienny trening), jak i analityczną (statystyki, porównania), umożliwiając użytkownikowi świadome monitorowanie procesu rozwoju poznawczego.

Zakres projektu

Projekt obejmuje realizację następujących elementów:

1. **Aplikacja mobilna**:
    - Umożliwiająca rejestrację i logowanie użytkowników,
    - Zawierająca interaktywny moduł gry **dual n-back** z zapisem wyników,
    - Udostępniająca historię wyników w formie statystyk i wykresów,
    - Umożliwiająca porównanie wyników z innymi użytkownikami po wpisaniu ich nazwy.

2. **System backendowy**:
    - Odpowiedzialny za zarządzanie danymi użytkowników i wynikami,
    - Udostępniający API do obsługi logiki aplikacji,
    - Zapewniający bezpieczne uwierzytelnianie i autoryzację.

3. **Baza danych**:
    - Przechowująca dane użytkowników oraz historię wyników sesji,
    - Zaprojektowana w sposób umożliwiający efektywne pobieranie i analizę danych.

Główne funkcjonalności

- Rejestracja i logowanie z obsługą sesji,
- Gra typu **dual n-back** z dynamicznym poziomem trudności,
- Zapis wyniku sesji raz dziennie oraz możliwość jego aktualizacji przy lepszym wyniku,
- Przegląd statystyk użytkownika: wykresy, ostanie 5 wyników,
- Możliwość porównania własnych wyników z wynikami innego użytkownika,

Oczekiwany rezultat

Efektem projektu ma być gotowa do użytku aplikacja mobilna, udostępniająca użytkownikowi:

- Przyjazny interfejs i intuicyjną obsługę,
- Możliwość regularnego wykonywania ćwiczeń,
- Funkcję śledzenia postępów i wyników w czasie,
- Funkcjonalność **porównania wyników** z innym użytkownikiem na podstawie jego nazwy.

Struktura aplikacji mobilnej

Aplikacja mobilna została zaprojektowana z zachowaniem zasad modularności oraz zgodnie z architekturą **MVVM (Model-View-ViewModel)**. Struktura projektu została podzielona na katalogi zgodne z odpowiedzialnością poszczególnych komponentów, co ułatwia rozwój, testowanie i utrzymanie aplikacji.

Plik główny

- **main.dart**
  Punkt wejścia aplikacji. Inicjalizuje podstawowe zależności, konfiguracje motywu oraz ustawia główny widget aplikacji (np. MaterialApp).

Katalog core/
Zawiera wspólne komponenty, narzędzia i definicje używane w całej aplikacji.

- **utils.dart** – Funkcje pomocnicze, np. formatowanie dat, sprawdzanie połączenia z internetem.
- **constants/server_constants.dart** – Stałe związane z adresami URL i endpointami API.
- **failure/failure.dart** – Definicje wyjątków i obsługi błędów w jednolity sposób.
- **providers/current_user_notifier.dart** – Provider zarządzający bieżącym

stanem zalogowanego użytkownika.

- **theme/** – Motyw aplikacji:
  - app_pallete.dart – Paleta kolorów,
  - theme.dart – Definicje stylów i motywów globalnych.
- **widgets/loader.dart** – Uniwersalny widget ładowania (spinner/progress bar).

Katalog features/
Zorganizowany według funkcjonalności. Zawiera kompletne moduły aplikacji: autoryzację oraz funkcje główne (ekran domowy, gra, statystyki).

auth/ – Moduł uwierzytelniania

- **model/user_model.dart** – Model użytkownika (id, imię, e-mail, token itp.).
- **repositories/** – Warstwa dostępu do danych:
  - auth_local_repository.dart – Obsługa lokalnego przechowywania (SharedPreferences),
  - auth_remote_repository.dart – Komunikacja z API.
- **view/pages/** – Ekrany logowania i rejestracji:
  - login_page.dart, signup_page.dart
- **view/widgets/** – Komponenty UI używane w auth:
  - auth_gradient_button.dart, custom_field.dart
- **viewmodel/auth_viewmodel.dart** – Logika biznesowa związana z logowaniem, rejestracją, obsługą błędów, sesji użytkownika.

home/ – Moduł główny

- **model/**
  - game_result_model.dart – Struktura danych sesji treningowych,
  - n_back_sequence.dart – Logika generowania sekwencji dla gry n-back,
  - settings_model.dart – Preferencje gry i użytkownika.

- **repositories/**
  - home_local_repository.dart – Odczyt/zapis danych lokalnych (np. najlepszy wynik dnia),
  - home_remote_repository.dart – Pobieranie i zapisywanie wyników do bazy przez API.

- **view/pages/**
  - home_page.dart – Ekran główny z dostępem do funkcji aplikacji,
  - n_back_game_page.dart – Interaktywny ekran gry dual n-back,
  - n_back_stats_page.dart – Strona z wizualizacją wyników użytkownika.

- **viewmodel/n_back_game_viewmodel.dart**
  - Odpowiada za logikę gry, zarządzanie stanem sesji, aktualizację wyników, obsługę zdarzeń z UI.

Architektura i zalety struktury

- **Podział per funkcjonalność (feature-first)** – Wszystkie pliki związane z daną funkcją są zorganizowane w jednym katalogu.
- **MVVM** – Ułatwia testowanie jednostkowe i zarządzanie stanem aplikacji.
- **Repository Pattern** – Oddziela logikę biznesową od warstwy danych.

Opis zastosowanych technologii

Flutter

Flutter to framework open-source stworzony przez Google, umożliwiający tworzenie aplikacji mobilnych, webowych i desktopowych z jednego wspólnego kodu źródłowego. W projekcie aplikacji do treningu pamięci Flutter został wykorzystany do budowy całego interfejsu użytkownika oraz logiki warstwy prezentacji.

Kluczowe zalety wykorzystania Fluttera:

- **Wieloplatformowość** – umożliwia uruchamianie aplikacji na Androidzie i iOS z jednej bazy kodu.
- **Szybki rozwój (Hot Reload)** – błyskawiczne wprowadzanie zmian bez konieczności ponownej kompilacji.
- **Bogaty ekosystem widgetów** – możliwość tworzenia nowoczesnych i responsywnych interfejsów użytkownika.
- **Integracja z narzędziami deweloperskimi** – obsługa debugowania, profilowania i testów jednostkowych.

W aplikacji Flutter został wykorzystany do:

- Projektowania ekranów i komponentów UI (np. graficzny interfejs gry n-back),
- Tworzenia nawigacji pomiędzy ekranami,
- Obsługi logiki interakcji użytkownika i połączeń z backendem.

Riverpod

Riverpod jest kluczowym narzędziem do zarządzania stanem i zależnościami w aplikacji. W projekcie pełni rolę warstwy pośredniej między widokiem a repozytoriami danych, umożliwiając czystą separację logiki biznesowej od UI i ułatwiając testowanie.

Obsługa zapisu i odczytu danych z wykorzystaniem Riverpod
W aplikacji Riverpod został wykorzystany do tworzenia instancji repozytoriów komunikujących się z backendem oraz lokalnym magazynem danych. Repozytoria te odpowiedzialne są za operacje takie jak rejestracja, logowanie, pobieranie aktualnych danych użytkownika, a także przechowywanie tokena sesji.

Repozytorium zdalne (AuthRemoteRepository)

- Zdefiniowane jako **provider Riverpod** (@riverpod), dzięki czemu instancja jest tworzona i zarządzana automatycznie przez framework.
- Metody signup, login oraz getCurrentUserData realizują zapytania HTTP do backendu (serwera FastAPI).

- Dane wysyłane są i odbierane w formacie JSON.
- Każda metoda zwraca wynik typu Either<AppFailure, UserModel>, co umożliwia elegancką obsługę sukcesów i błędów w stylu funkcyjnym.
- Przy rejestracji i logowaniu metoda zwraca model użytkownika wraz z tokenem JWT, który jest niezbędny do uwierzytelniania dalszych żądań.

Repozytorium lokalne (AuthLocalRepository)

- Również udostępnione jako provider Riverpod z adnotacją @Riverpod(keepAlive: true) — instancja jest utrzymywana tak długo, jak działa aplikacja.
- Używa biblioteki **Shared Preferences** do trwałego zapisu danych lokalnych na urządzeniu, w szczególności tokena JWT.
- Metody setToken i getToken odpowiadają za zapis i odczyt tokena, co pozwala na automatyczne utrzymywanie sesji użytkownika między uruchomieniami aplikacji.
- Inicjalizacja SharedPreferences odbywa się asynchronicznie w metodzie init(), która powinna być wywołana podczas startu aplikacji.

Zalety podejścia z Riverpod

- **Automatyczne zarządzanie cyklem życia** repozytoriów i ich zależności.
- **Łatwa integracja z UI** — widżety Fluttera mogą subskrybować zmiany stanów lub wywoływać metody repozytoriów przez provider.
- **Separation of concerns** — logika komunikacji sieciowej i lokalnego zapisu danych jest wyraźnie oddzielona od warstwy widoku.
- **Obsługa błędów i wyników w stylu funkcyjnym** — Either<AppFailure, UserModel> pozwala na spójną obsługę sukcesów i błędów bez rzucania wyjątków.
- **Testowalność** — dzięki niezależności repozytoriów i providera można łatwo pisać testy jednostkowe, mockując warstwę danych.

FastAPI

W części serwerowej projektu wykorzystano **FastAPI** jako główny framework do budowy REST API. FastAPI umożliwia szybkie tworzenie wydajnych, bezpiecznych i nowoczesnych serwisów webowych w języku Python.

Kluczowe elementy implementacji:

- **Struktura projektu**
  Projekt został podzielony na moduły odpowiadające za modele danych (models), schematy Pydantic (pydantic_schemas), logikę tras (routes) oraz middleware do autoryzacji.

- **Walidacja danych z Pydantic**
  FastAPI wykorzystuje Pydantic do automatycznej walidacji i serializacji danych wejściowych oraz wyjściowych. Przykładowo, podczas rejestracji użytkownika (/auth/signup) oraz logowania (/auth/login), dane są sprawdzane pod kątem poprawności typu i obecności wymaganych pól na podstawie klas UserCreate i UserLogin. Podobnie, przy operacjach na wynikach testu N-Back, dane są walidowane przez schemat NBackResult.

- **Bezpieczne hasła z bcrypt**
  Do przechowywania haseł użytkowników wykorzystywana jest biblioteka **bcrypt**, która zapewnia bezpieczne haszowanie haseł. Podczas rejestracji hasło jest haszowane przed zapisaniem do bazy, a przy logowaniu następuje porównanie hasła podanego przez użytkownika z hashem zapisanym w bazie.

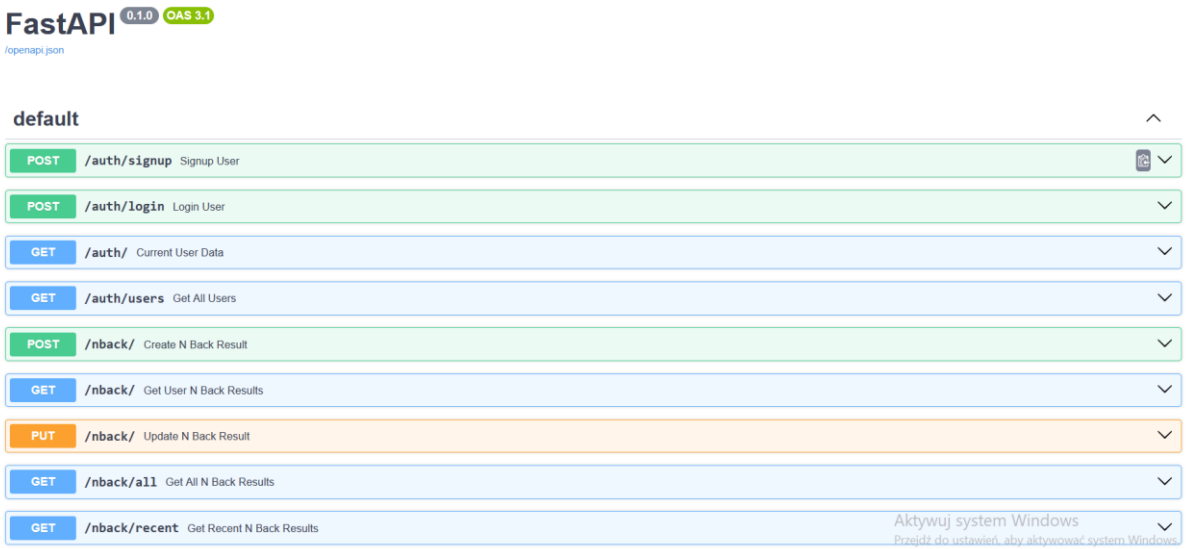
- **Autoryzacja JWT**
  Dostęp do chronionych zasobów wymaga przesłania tokena JWT w nagłówku. Middleware (auth_middleware) weryfikuje poprawność tokena i identyfikuje użytkownika.

- **Obsługa wyników testu N-Back**
  FastAPI udostępnia endpointy do zapisywania, pobierania i aktualizacji wyników testów N-Back. Dane są przechowywane w bazie PostgreSQL z wykorzystaniem SQLAlchemy.

- **Automatyczna dokumentacja**
  FastAPI automatycznie generuje dokumentację API dostępną pod /docs (Swagger UI) oraz /redoc.

Przykładowe endpointy:

- POST /auth/signup – rejestracja nowego użytkownika (walidacja danych przez Pydantic, haszowanie hasła przez bcrypt)
- POST /auth/login – logowanie i uzyskanie tokena JWT (walidacja danych przez Pydantic, weryfikacja hasła przez bcrypt)
- GET /auth/ – pobranie danych aktualnie zalogowanego użytkownika
- POST /nback/ – zapisanie wyniku testu N-Back (walidacja danych przez Pydantic)
- GET /nback/recent – pobranie ostatnich wyników użytkownika



Zalety zastosowania FastAPI, Pydantic i bcrypt:

- Wysoka wydajność i asynchroniczność
- Automatyczna walidacja i konwersja danych wejściowych/wyjściowych
- Bezpieczne przechowywanie haseł dzięki bcrypt
- Prosta integracja z bazą danych i JWT
- Automatyczna dokumentacja API
- Łatwość rozbudowy i testowania

PostgreSQL

W projekcie jako główną bazę danych zastosowano **PostgreSQL**. Jest to wydajny, bezpieczny i skalowalny system zarządzania relacyjnymi bazami danych, który doskonale współpracuje z aplikacjami opartymi o FastAPI.

Kluczowe aspekty integracji:

- **Połączenie z bazą**
  Połączenie z bazą PostgreSQL realizowane jest za pomocą SQLAlchemy, a dane dostępowe (adres, użytkownik, hasło, nazwa bazy) są skonfigurowane w pliku database.py w zmiennej DATABASE_URL.

- **SQLAlchemy jako ORM**
  SQLAlchemy umożliwia mapowanie obiektowo-relacyjne – modele Pythona (np. User, NBackResult) odpowiadają tabelom w bazie danych. Dzięki temu operacje na bazie (tworzenie, pobieranie, aktualizacja, usuwanie rekordów) są realizowane w sposób obiektowy.

- **Tworzenie tabel**
  Przy starcie aplikacji (w pliku main.py) wywoływana jest metoda Base.metadata.create_all(engine), która automatycznie tworzy wymagane tabele w bazie PostgreSQL na podstawie zdefiniowanych modeli.

- **Przechowywanie danych użytkowników i wyników**
  W bazie przechowywane są dane użytkowników (w tabeli users) oraz wyniki testów N-Back (w tabeli n_back_results). Relacje między tabelami są odwzorowane za pomocą kluczy obcych.

Schemat bazy danych

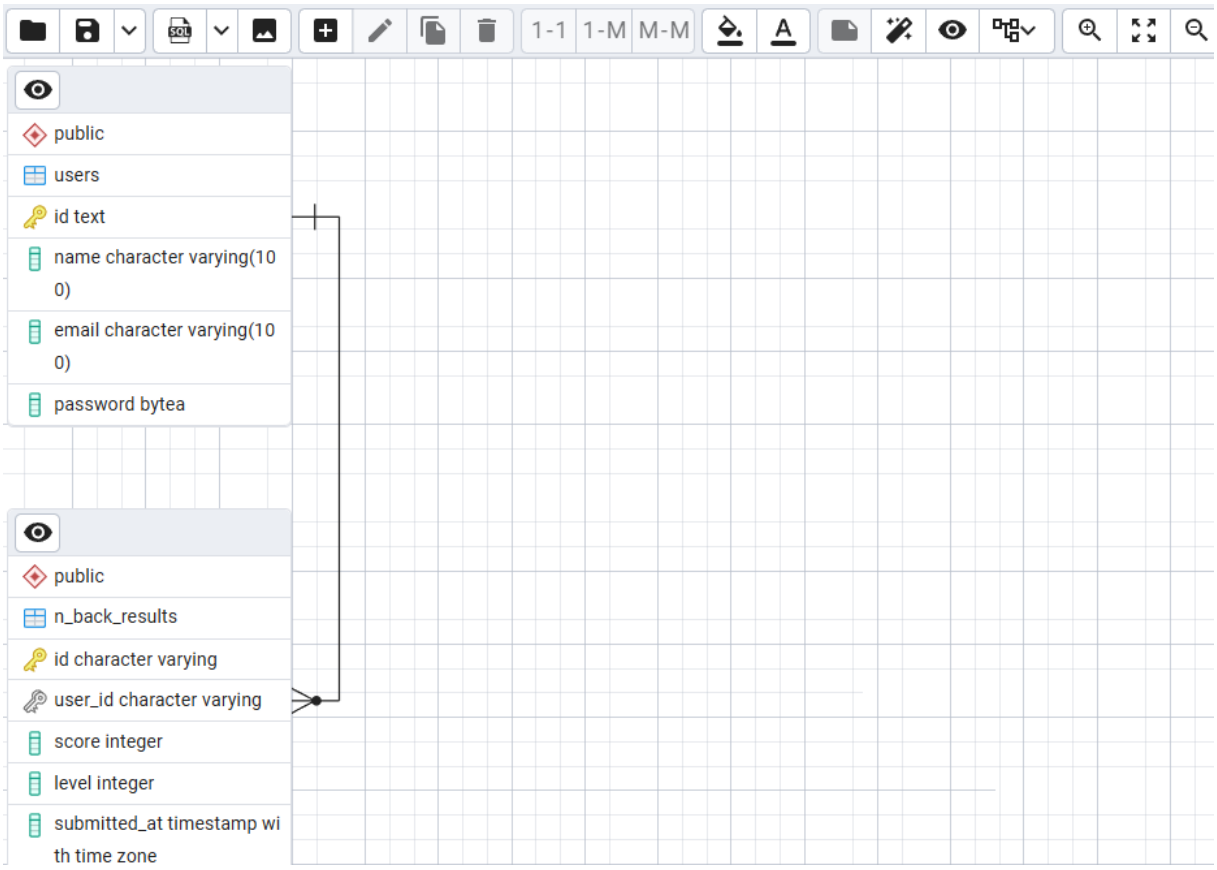


Tabela users

Tabela users przechowuje dane użytkowników aplikacji. Każdy użytkownik posiada unikalny identyfikator, imię, adres e-mail oraz zaszyfrowane hasło. Dane logowania są chronione przy użyciu bezpiecznego algorytmu haszującego.

Pola:

- **id** – unikalny identyfikator użytkownika (klucz główny),
- **name** – imię użytkownika,

- **email** – adres e-mail użytkownika,
- **password** – hasło użytkownika zapisane w postaci zaszyfrowanej.

Tabela n_back_results

Tabela n_back_results zawiera dane dotyczące wyników sesji treningowych w grze typu *dual n-back*. Każdy wpis w tej tabeli jest powiązany z jednym użytkownikiem za pomocą klucza obcego user_id.

Pola:

- **id** – unikalny identyfikator wyniku (klucz główny),
- **user_id** – identyfikator użytkownika powiązany z wynikiem (klucz obcy odnoszący się do users.id),
- **score** – wynik punktowy uzyskany podczas sesji,
- **level** – poziom trudności gry (np. 2-back, 3-back),
- **submitted_at** – data i czas przesłania wyniku (ustawiana automatycznie).

Relacje między tabelami

Schemat bazy danych opiera się na relacji **jeden-do-wielu** pomiędzy tabelami users i n_back_results:

- Jeden użytkownik może mieć wiele przypisanych wyników treningowych,
- Każdy wynik należy do dokładnie jednego użytkownika.

Implementacja

Proces implementacji aplikacji został przeprowadzony etapowo, zgodnie z założeniami architektury MVVM oraz z podziałem na odpowiednie warstwy: interfejs użytkownika, logikę biznesową i warstwę danych.

Etapy implementacji

- Stworzono projekt mobilny w technologii **Flutter** z zastosowaniem zarządzania stanem przy użyciu **Riverpod**.

- Backend został opracowany w języku Python z wykorzystaniem frameworka **FastAPI**, obsługując komunikację z bazą danych oraz autoryzację użytkowników.
- Do trwałego przechowywania danych zastosowano **PostgreSQL**, a do wizualnego zarządzania bazą wykorzystano narzędzie **pgAdmin**.

.

Użyte narzędzia

- **GitHub** – do kontroli wersji i zarządzania repozytorium projektu,
- **pgAdmin** – do wizualnego przeglądu struktury i zawartości bazy danych PostgreSQL,
- **Uvicorn** – jako serwer ASGI do uruchamiania aplikacji FastAPI w środowisku deweloperskim i produkcyjnym,
- **Flutter DevTools** – do debugowania i analizy działania aplikacji mobilnej,
- **Android Emulator** – testowanie działania aplikacji na urządzeniach wirtualnych,
- **Fizyczne urządzenie z Androidem** – do testów funkcjonalnych

Testowanie i uruchomienie

- Backend testowany był lokalnie przy użyciu automatycznie generowanej dokumentacji Swagger (dostępnej w FastAPI).
- Aplikacja mobilna była testowana na:
    - emulatorze systemu Android z poziomu Android Studio,
    - fizycznym smartfonie z systemem Android, co pozwoliło ocenić rzeczywistą wydajność i ergonomię interfejsu.
- Testy objęły logowanie, rejestrację, zapisywanie wyników, aktualizacje sesji treningowych oraz podgląd statystyk

Podsumowanie

W ramach projektu zrealizowano mobilną aplikację do treningu pamięci opartą na Flutterze i architekturze MVVM. Aplikacja oferuje rejestrację, logowanie, grę typu dual n-back, zapis wyników oraz porównywanie ich z innymi użytkownikami. Udało się zintegrować aplikację z backendem napisanym w FastAPI oraz bazą danych PostgreSQL. Projekt umożliwił praktyczne

wykorzystanie technologii takich jak Riverpod, JWT. Całość została przetestowana na emulatorze Android oraz fizycznym urządzeniu, a kod kontrolowano za pomocą systemu Git i platformy GitHub.

Wnioski i zdobyte umiejętności

1. **Projekt był utrwaleniem i poszerzeniem wiedzy z Fluttera**
   Realizacja aplikacji mobilnej w oparciu o Fluttera pozwoliła na praktyczne wykorzystanie wielu aspektów tej technologii, m.in. budowania dynamicznych interfejsów użytkownika, zarządzania stanem aplikacji przy użyciu Riverpod, obsługi nawigacji, pracy z formularzami, a także integracji z backendem. W trakcie pracy utrwalono również dobre praktyki kodowania, takie jak rozdzielanie logiki biznesowej od warstwy widoku (MVVM) oraz stosowanie modularnej struktury projektu.

2. **Utrwalenie wiedzy o bazach danych**
   Istotnym elementem projektu było zaprojektowanie i obsługa relacyjnej bazy danych PostgreSQL. Praca nad modelem danych, relacjami między tabelami oraz mechanizmami zapisu i odczytu danych pozwoliła zrozumieć praktyczne zastosowanie teorii baz danych. Narzędzie PGAdmin umożliwiło wygodne zarządzanie strukturą bazy, testowanie zapytań SQL oraz monitorowanie operacji CRUD wykonywanych z poziomu aplikacji.

3. **Nauka i zrozumienie programowania API**
   Backend aplikacji został stworzony w technologii FastAPI, co wymagało opanowania zasad tworzenia i projektowania nowoczesnych REST API. Podczas realizacji zadań backendowych szczególną uwagę poświęcono zarządzaniu żądaniami HTTP, obsłudze błędów oraz wdrożeniu mechanizmu uwierzytelniania przy użyciu tokenów JWT. Projekt pozwolił zrozumieć cykl komunikacji klient–serwer oraz sposoby zabezpieczania danych przesyłanych pomiędzy nimi

4. **Zrozumienie architektury i komunikacji między bazą danych a aplikacją mobilną**
   Całościowy proces tworzenia aplikacji – od warstwy prezentacji po backend i bazę danych – umożliwił pełne zrozumienie przepływu danych oraz współpracy poszczególnych komponentów systemu. Pozwoliło to prześledzić każdy etap komunikacji: od wysłania zapytania z poziomu interfejsu Fluttera, przez przetworzenie żądania w FastAPI, aż po zapis lub odczyt danych z bazy PostgreSQL i ich zwrot do aplikacji. Projekt ukazał zależności między technologiami.