# Dokumentacja Student Traffic

Krzysztof Gołuchowski
Mateusz Wala
Jakub Grześ
Jan Masternak
Krystian Sienkiewicz
Tomasz Smyda

11–13 kwietnia 2025

## Opis projektu

*Student Traffic* to mobilna aplikacja zaprojektowana z myślą o studentach mieszkających w akademikach Miasteczka Studenckiego AGH. Projekt ma na celu ułatwienie codziennego życia na kampusie poprzez rozwiązanie realnych problemów, z jakimi mierzą się mieszkańcy.

Aplikacja integruje kilka kluczowych funkcjonalności:

- umożliwia rezerwację pralek i suszarek wraz z podglądem ich dostępności oraz powiadomieniami o zakończeniu cyklu prania,
- oferuje tablicę ogłoszeń, gdzie można dzielić się jedzeniem lub przedmiotami,
- zawiera moduł do organizowania wydarzeń sportowych (np. gry we na boisku), umożliwiający tworzenie i dołączanie do wydarzeń.

Obecnie na kampusie nie istnieje żadne kompleksowe, dedykowane rozwiązanie, które wspierałoby studentów w tych kwestiach. Komunikacja odbywa się chaotycznie — głównie przez grupy na Messengerze, Facebooku czy plakaty, co jest niewystarczające, zawodne i niewygodne.

# Zastosowane technologie

- **Frontend – React + Next.js + shadcn/ui**
  Warstwa frontendowa aplikacji została zrealizowana z wykorzystaniem biblioteki **React**, która umożliwia tworzenie dynamicznych i responsywnych interfejsów użytkownika w formie aplikacji webowej. Do budowy interfejsu graficznego użyty został nowoczesny system komponentów **shadcn/ui**. Do routingu wewnątrz aplikacji użyto frameworka **Next.js**, który wykorzystuje router oparty na strukturze plików.

- **Backend – Java + Spring**
  Część serwerowa aplikacji została napisana w języku **Java** z wykorzystaniem frameworka **Spring**, który jest jednym z najpopularniejszych rozwiązań do tworzenia nowoczesnych aplikacji webowych. Aplikacja udostępnia funkcjonalności poprzez **REST API** i odpowiada za obsługę logiki biznesowej, zarządzanie sesjami użytkowników oraz komunikację z bazą danych.

- **Baza danych – Firebase**
  Do przechowywania danych użytkowników, ogłoszeń oraz stanu rezerwacji wykorzystano **Firebase** – platformę chmurową firmy Google. Dzięki integracji z Firebase możliwe było szybkie wdrożenie autoryzacji, przechowywania danych w czasie rzeczywistym oraz wykorzystania skalowalnych funkcji backendowych bez konieczności konfiguracji własnego serwera bazodanowego.

# Instrukcja uruchomienia aplikacji

Należy zainstalować następujące narzędzia:

- Node.js (wraz z `npm`) – przetestowane na wersji `v22.14.0 (LTS)`

Następnie należy przejść do katalogu `frontend` i wykonać tam następujące polecenia:

```
npm install --legacy-peer-deps
npm run dev
```

Aplikacja powinna być dostępna pod adresem `http://localhost:3000`.

Naszym oczom ukaże się napis „This app is only available on mobile.” – należy zmniejszyć okno przeglądarki, aby wymusić widok mobilny.

Jeśli wszystko poszło dobrze, powinniśmy zobaczyć ekran logowania. Jako nazwę użytkownika należy wpisać `test`, a jako hasło `test1234`.

Ze względu na darmowy hosting, pierwsze wywołanie API może zająć nawet 90 sekund.

# Napotkane problemy

W trakcie realizacji projektu napotkaliśmy na kilka istotnych wyzwań i problemów, które wymagały szybkiej adaptacji oraz współpracy w zespole:

- **Problemy z integracją frontendu i backendu** – Na początku projektu ustalenie wspólnego formatu komunikacji pomiędzy frontendem (React) a backendem (Java) nie było jednoznaczne. Problem został rozwiązany poprzez wspólne spotkanie zespołu i utworzenie dokumentu opisującego API.
- **Zarządzanie czasem** – Czas trwania hackathonu był ograniczony, a zakres funkcjonalności rozbudowany. Udało nam się poradzić sobie z tym problemem dzięki dobrej organizacji pracy, podziale zadań oraz codziennym spotkaniom synchronizacyjnym (standupy).
- **Problemy z autoryzacją użytkowników** – Początkowo implementacja JWT napotkała błędy w przekazywaniu tokenów. Finalnie zastosowano bibliotekę `spring-security`, a błędy debugowano wspólnie na podstawie logów backendu.
- **Błędne decyzje dotyczące modelu bazy danych** – Już w trakcie pracy zauważono, że pierwotny projekt bazy danych nie odpowiada naszym potrzebom. Spowodowało to konieczność usunięcia części kodu i jego refaktoryzacji.
- **Trudności z integracją bazy danych z backendem** – Problemy wynikały z ograniczonego doświadczenia jednej z osób ze Spring Boot oraz początkowych niejasności w komunikacji z osobą wspierającą ten fragment.

# Elementy technicznej dokumentacji

elementy technicznej dokumentacji - np. architektura systemu, opis komponentów czy zastosowanych pomysłów/wzorców. Warto umieścić tu diagramy (np. klas, przepływu sterowania)

## Frontend

Warstwa frontendowa została zrealizowana w bibliotece **React** z wykorzystaniem systemu komponentów **shadcn/ui**. Interfejs użytkownika jest responsywny, modularny i oparty na koncepcji wielokrotnego użycia komponentów (*reusable components*). Do organizacji routingu wewnątrz aplikacji wykorzystano framework **Next.js**, który oferuje intuicyjny i wydajny system routingu oparty na strukturze plików, umożliwiający łatwe zarządzanie ścieżkami i widokami. Komunikacja z backendem odbywa się poprzez zapytania HTTP do udostępnionego REST API.

## Backend

Część serwerowa została zaimplementowana w języku **Java** z użyciem frameworka **Spring**. Projekt backendu został podzielony na klasyczne warstwy:

- **Controller** – obsługuje przychodzące żądania HTTP i przekazuje je do odpowiednich serwisów,
- **Service** – zawiera logikę biznesową,
- **Repository** – odpowiada za komunikację z bazą danych.

Dzięki udostępnionym mechanizmie *komponentów*, zastosowano wzorzec **Dependency Injection** (DI), co zwiększa testowalność i elastyczność systemu. Do odwzorowania danych między warstwą aplikacyjną a bazą danych wykorzystano proste obiekty **DTO (Data Transfer Object)**. Backend udostępnia funkcjonalności w formie **REST API**.

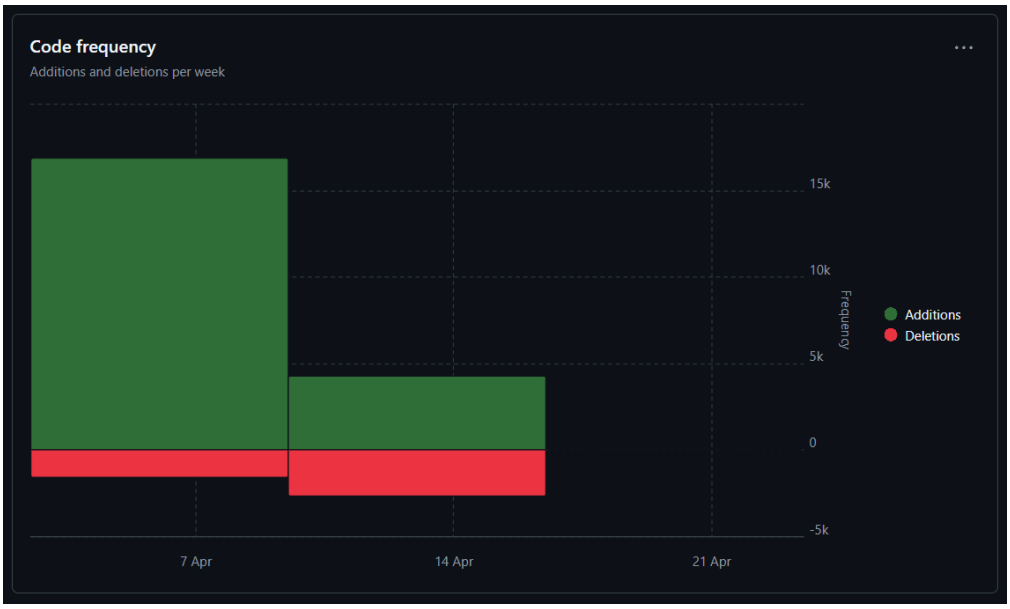
**Lista dostępnych endpointów:**

- `GET /announcements` - zwraca wszystkie aktualne ogłoszenia
- `POST /announcements/post` - dodawanie nowego ogłoszenia
- `POST /damage/report` - oznaczenie danego obiektu jako zepsuty
- `POST /damage/fix` - oznaczenie danego obiektu jako sprawny
- `POST /events/add_event` - dodanie nowego wydarzenia
- `POST /events` - zwraca wydarzenia danej kategorii, danego dnia
- `POST /login/auth` - autoryzuje dane użytkownika
- `POST /objects/laundry` - zwraca stany dostępności pralek/suszarek, na danym piętrze, danego dnia
- `POST /reservation` - tworzy nową rezerwację obiektu

## Baza danych

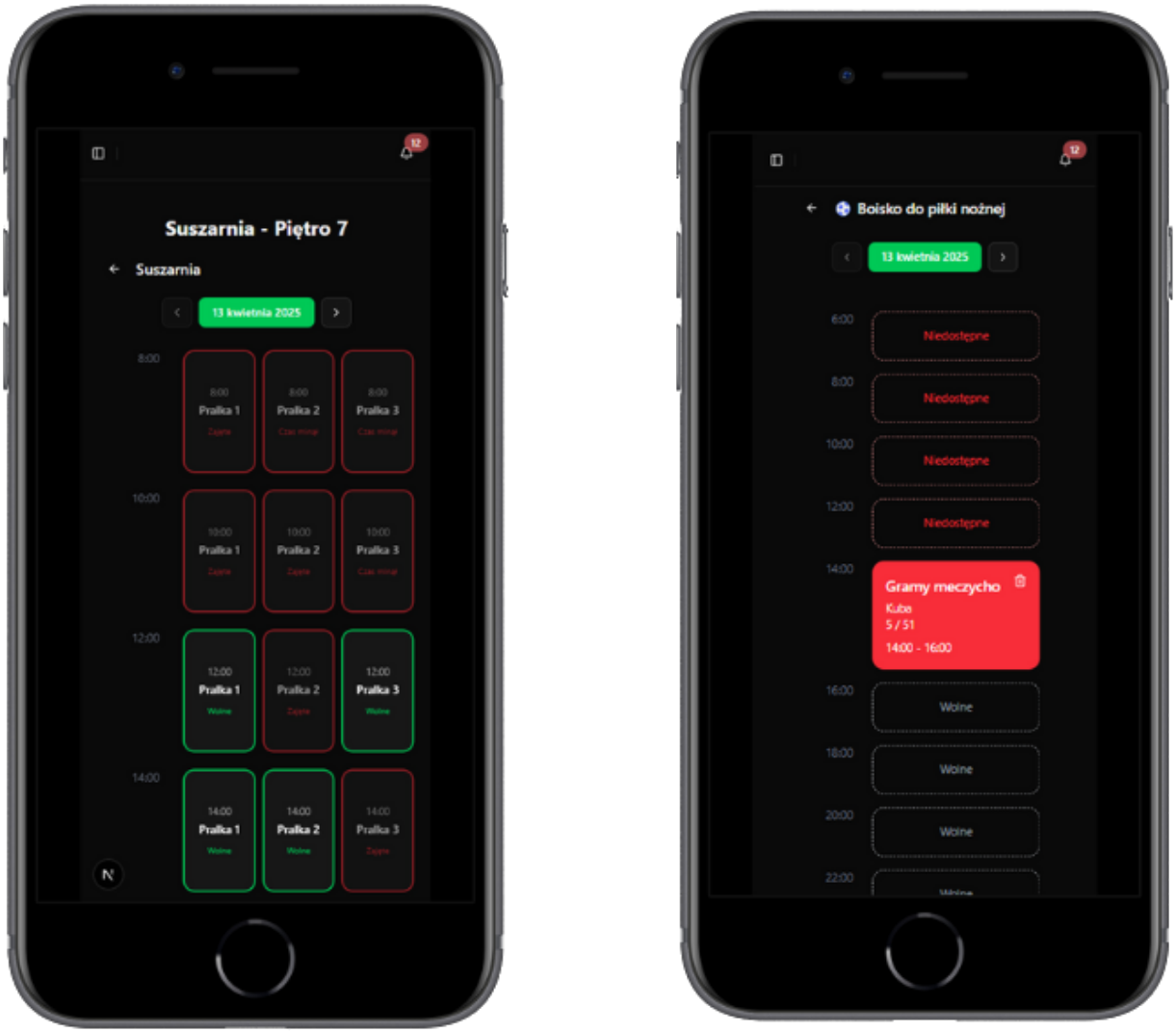
Do przechowywania danych wykorzystano platformę chmurową **Firebase**. Zapewnia ona zarówno bazę danych (w czasie rzeczywistym), dzięki temu każdy członek naszego zespołu mógł mieć dostęp do jednej, aktualnej wersji bazy.

# Metryki z GitHuba




Widać tutaj bardzo dużą rozbieżność ilości dodanego kodu pomiędzy dwoma dniami pracy. Pokazuje nam to jak dużą część pisanego kodu stanowi samo stworzenie projektu

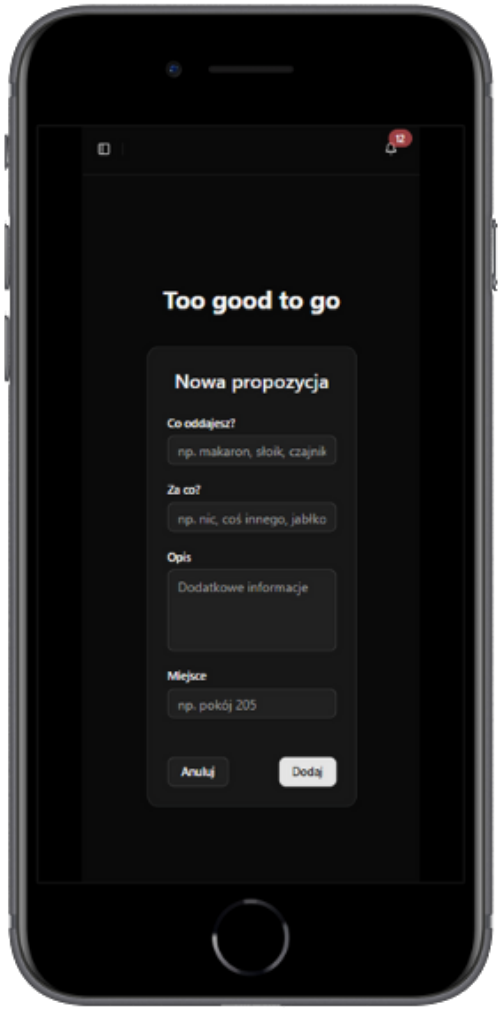
# Finalny wygląd aplikacji



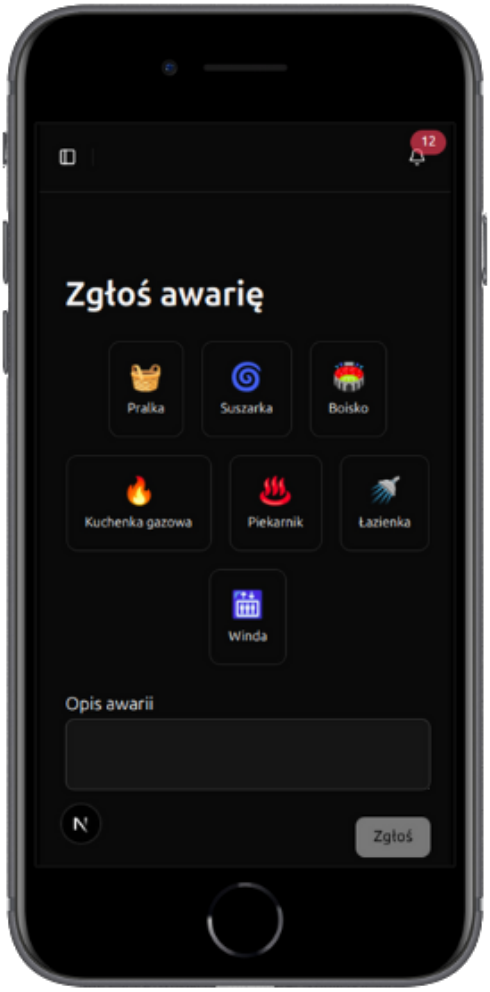

(a) Rezerwacja pralek/suszarek

(b) Organizacja wydarzeń

Zrzuty ekranu naszej aplikacji (cz. 1)




(a) Wymiana




(b) Zgłaszanie awarii

Zrzuty ekranu naszej aplikacji (cz. 2)