

**Politechnika Śląska**

**Wydział Automatyki, Elektroniki i Informatyki**

**Kierunek Informatyka**

##### Projekt inżynierski

System zarządzania siecią siłowni

Autor: Dawid Danel

Kierujący pracą: dr inż. Ewa Płuciennik

Gliwice, Styczeń 2021

Spis treści

[Streszczenie 3](#_Toc92213906)

[Rozdział 1 Wstęp 4](#_Toc92213907)

[1.1 Tematyka pracy 4](#_Toc92213908)

[1.2 Cel pracy 4](#_Toc92213909)

[1.3 Rozdziały i podział prac 4](#_Toc92213910)

[Rozdział 2 Analiza tematu 6](#_Toc92213911)

[2.1 Analiza problemu pracy 6](#_Toc92213912)

[2.2 Potrzeby i wymagania 8](#_Toc92213913)

[2.3 Opis istniejących rozwiązań 10](#_Toc92213914)

[Rozdział 3 Wymagania i narzędzia 17](#_Toc92213915)

[3.1 Wymagania niefunkcjonalne 17](#_Toc92213916)

[3.2 Wymagania funkcjonalne 19](#_Toc92213917)

[3.4 Wykorzystywane narzędzia 23](#_Toc92213918)

[3.5 Metodyka pracy nad projektem 24](#_Toc92213919)

[Rozdział 4 Specyfikacja zewnętrzna 27](#_Toc92213920)

[4.1 Wymagania sprzętowe i programowe 28](#_Toc92213921)

[4.2 Sposób instalacji i uruchamianie 29](#_Toc92213922)

[4.3 Użytkownicy i bezpieczeństwo 32](#_Toc92213923)

[4.4 Interfejs użytkownika 33](#_Toc92213924)

[4.5 Administracja i sposób działania 38](#_Toc92213925)

[Rozdział 5 Specyfikacja wewnętrzna 43](#_Toc92213926)

[5.1 Funkcjonalność systemu 43](#_Toc92213927)

[5.2 Warstwa prezentacji 44](#_Toc92213928)

[5.3 Warstwa serwera 54](#_Toc92213929)

[5.4 Struktury danych i organizacja bazy danych 62](#_Toc92213930)

[Rozdział 6 Weryfikacja i walidacja 66](#_Toc92213931)

[6.1 Weryfikacja działania systemu 66](#_Toc92213932)

[6.2 Przypadki testowe 67](#_Toc92213933)

[6.3 Testy jednostkowe 68](#_Toc92213934)

[6.4 Testy integracyjne 71](#_Toc92213935)

[6.5 Testy manualne 74](#_Toc92213936)

[Rozdział 7 Podsumowanie i wnioski 75](#_Toc92213937)

[7.1 Wyniki pracy 75](#_Toc92213938)

[7.2 Dalsza rozbudowa systemu 76](#_Toc92213939)

[7.3 Problemy i wyzwania 77](#_Toc92213940)

[Bibliografia i](#_Toc92213941)

[Spis skrótów i symboli iii](#_Toc92213942)

[Lista dodatkowych plików, uzupełniających tekst pracy v](#_Toc92213943)

[Spis rysunków vi](#_Toc92213944)

[Spis kodów vii](#_Toc92213945)

# Streszczenie

Celem pracy jest opracowanie aplikacji internetowej będącej systemem zarządzania siecią siłowni. Powinna ona pozwalać na użytkownie zarówno przez pracowników takiej sieci, jak również klientów zewnętrznych rejestrujących się przez stronę internetową. Aplikacja pozwala na przeglądanie dostępnych siłowni, ich wyposażenia czy grafików zajęć, jak również umożliwiać zapisywanie się na nie i śledzenie swojej historii treningów. W pracy przedstawione zostało gruntowne omówienie tematyki projektu, jego wymagania funkcjonalne oraz niefunkcjonalne jak również sposób implementacji i weryfikacji dostarczonego rozwiązania.

**Słowa kluczowe**: aplikacja internetowa, Java, Angular

# Rozdział 1 Wstęp

W niniejszej pracy przedstawiono projekt systemu do zarządzania siecią siłowni. W ramach pracy przedstawiono analizę, projektowanie, implementację oraz weryfikację poprawności dostarczonego rozwiązania. Zawarte są również wynikające z pracy wnioski oraz możliwości dalszego rozwoju systemu.

## 1.1 Tematyka pracy

Tematyką pracy jest utworzenie systemu do zarządzania siecią siłowni, w postaci aplikacji internetowej. Powinien on dostarczać wszystkie funkcjonalności wymagane do kompleksowej obsługi sieci siłowni. System przeznaczony jest zarówno dla użytkowników technicznych, czyli pracowników i właścicieli obiektów, jak również klientów którzy będą korzystać z oferowanych przez obiekty sportowe aktywności.

## 1.2 Cel pracy

Celem pracy jest utworzenie systemu w postaci aplikacji internetowej z wykorzystaniem relacyjnej bazy danych oraz nowoczesnych, popularnych technologii wykorzystywanych w tworzeniu aplikacji w architekturze mikro serwisowej.

## 1.3 Rozdziały i podział prac

Projekt przedstawiony został za pomocą następujących rozdziałów:

* **Analiza tematu** – zawiera sformułowanie problemu pracy oraz dokładny opis celów, wymagań i przeznaczenia powstającego systemu. Dodatkowo opisano istniejące systemu spełniające podobne funkcjonalności
* **Wymagania i narzędzia** – przedstawia wymagania funkcjonalne oraz niefunkcjonalne implementowane przez system oraz narzędzia wykorzystywane do ich stworzenia
* **Specyfikacja zewnętrzna** – określa sposób przygotowania środowiska do uruchomienia aplikacji, wymagań sprzętowych oraz potrzebnych narzędzi. Zawiera również instrukcję instalacji i uruchomienia systemu
* **Specyfikacja wewnętrzna** – omawia implementację systemu, sposób działania poszczególnych warstw i części. Zawiera przykładowe implementacje oraz opis struktury systemu
* **Weryfikacja i walidacja** – przedstawia sposób weryfikacji poprawności działania systemu, wykorzystane metodyki oraz sposoby testów. Dodatkowo przytoczone są konkretne przykłady testów oraz błędy który udało się dzięki nim wykryć

Całość prac nad systemem podzielono na poszczególne części przeplatające się ze sobą wraz upływem czasu. Wszystkie elementy zostały w pełni wykonane przez twórcę pracy. Podział można przedstawić następująco:

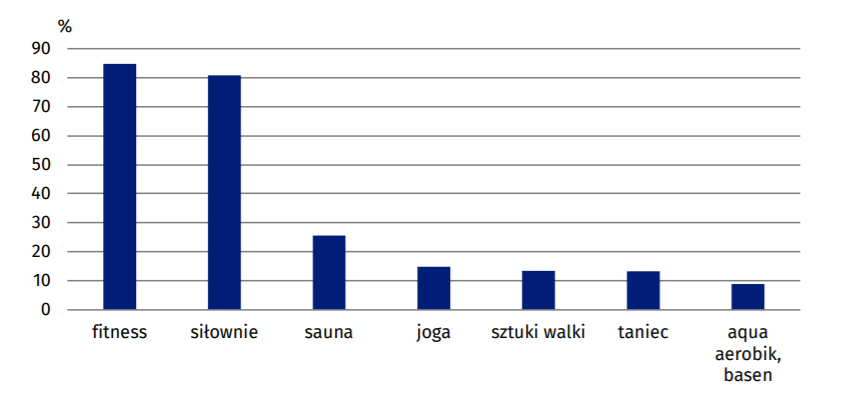
* **Analiza** – faza pierwotna, opierająca się na analizie wymagań i potrzeb systemu, jak również sprawdzeniu aktualnie istniejących rozwiązań
* **Projektowanie** – faza poprzedzająca implementację, zawierająca wstępne szkice wykorzystywanych obiektów czy struktury bazy danych
* **Implementacja** – faza rzeczywistego tworzenia kodu aplikacji oraz jego testowanie
* **Dokumentacja** – faza zawierająca w sobie tworzenie specyfikacji zewnętrznej i wewnętrznej oraz przygotowywanie diagramów

# Rozdział 2 Analiza tematu

W rozdziale przedstawiono szczegółową analizę tematu pracy inżynierskiej, w tym gruntowne omówienie problematyki oraz rozwiązań dostarczanych przez powstały projekt. Zawiera on także opisy istniejących i aktualnie używanych systemów spełniających częściowo lub w podobnym zakresie wymagania postawione sobie przy realizacji projektu.

## 2.1 Analiza problemu pracy

Zdrowy styl życia jest obecnie tematyką bardzo powszechną i interesującą dla dużej ilości konsumentów. Według danych Głównego Urzędu Statystycznego w Polsce w roku 2020 istniało 3900 obiektów służących do poprawi kondycji fizycznej, przez co rozumiane są między innymi miejsca oferujące zajęcia fitness czy treningi na siłowniach[[1]](#footnote-1). Na przestrzeni ostatnich 5 lat każdego roku odnotowano wzrost liczby zarejestrowanych i aktywnych podmiotów zajmujących się poprawą kondycji fizycznej. Nawet w roku 2020, w którym większość takich obiektów zamknięta była z powodu obostrzeń, wystąpił niewielki, 2% wzrost.



Rysunek 2.1 Najpopularniejsze usługi oferowane w obiektach służących poprawie kondycji fizycznej.

Na Rysunku 2.1 (Źródło danych Główny Urząd Statystyczny) przedstawione zostały usługi oferowane przez poszczególne obiekty służące poprawie kondycji fizycznej. Jasno wynika z niego fakt, iż największym zapotrzebowaniem klientów cieszą się zajęcia fitness jak i tradycyjne treningi na siłowni. Rodzi to naturalne zapotrzebowanie na nowoczesne rozwiązania dostosowane do świadomego użytkownika smartfonów czy samego Internetu.

Tak duża ilość obiektów powstających na przestrzeni ostatnich lat powoduje pewien naturalny chaos w aktualnie używanych przez nie systemach komputerowych (czy nawet takich nie wykorzystujących żadnych nowoczesnych rozwiązań). Wiele placówek czy sieci posiada swoje własne, często przeznaczone tylko dla pracowników systemy służące do kontroli karnetów czy regulacji płatności. Niektóre z nich korzystają również z mediów społecznościowych jako form kontaktu z klientami, gdzie w postaci infografik czy wiadomości przedstawiane są najbliższe grafiki zajęć, kontakt do trenerów osobistych czy ceny karnetów.

Odnalezienie się przez docelowego klienta w takim natłoku form komunikacji i ich niespójności może odrzucać część osób już na samym początku. Właśnie z tych powodów zaprojektowany został system zarządzania siecią siłowni opisany w tej pracy. Jest on odpowiedzią na problemy opisane wcześniej, a jednocześnie stworzony z myślą o użytkownikach spędzających dużo czasu z ich mobilnymi urządzeniami elektronicznymi, pozwalający na szybkie i intuicyjne sprawdzanie dostępnych informacji czy zapisywanie się na zajęcia.

Istotnym elementem systemu jest również jego jednoczesne przeznaczenia dla pracowników jak i klientów. Dzięki takiemu rozwiązaniu możliwa jest szybsza wymiana informacji oraz docelowo ograniczenie ilości wymaganych osób zajmujących się systemem poprzez automatyzację wielu procesów. Dodatkowe funkcje umożliwiające śledzenie swoich wizyt na siłowni czy aktywności dodatkowo pomagają klientom w obserwowaniu na bieżąco swoich postępów i docelowo na dłużej zatrzymują ich w powstałym środowisku.

Nowoczesna, responsywna aplikacja internetowa jest rozwiązaniem na miarę aktualnych czasów. Jej zastosowanie i jednoczesna automatyzacja wielu części sprawia, że jest przystępna zarówno dla użytkowników aktywnie korzystających z samej aplikacji, jak również bardziej konwencjonalnych, preferujących tradycyjny metody kontaktów z wcześniejszych lat. Zastosowanie nowoczesnych technologii i wysoki standard zabezpieczeń gwarantuje bezpieczeństwo, wydajność i trwałość powstałego systemu.

## 2.2 Potrzeby i wymagania

Aby powstały system przyciągnął uwagę właścicieli potencjalnych placówek zaimplementowane muszą być konkretne funkcjonalności spełniające potrzeby zarówno pracowników jak i klientów takich miejsc. Określenie podstawowego zakresu wymagań umożliwiło utworzenie systemu kompletnego, ale również otwartego na dalsze usprawnienia i modyfikacje.

Podstawowym problemem było określenie najważniejszych części systemu z perspektywy użytkownika docelowego. Jeżeli spełnione zostaną jego potrzeby, będzie on chciał wracać i nadal korzystać w powstałej aplikacji. W przeciwnym wypadku nawet najbardziej wyrafinowane funkcjonalności dostępne dla pracowników nie przyciągną uwagi klienta odstraszonego topornością systemu, szczególnie przy tak wysokiej ilości konkurencji.

Określone zostały zatem 3 najważniejsze czynniki przyciągające uwagę klientów:

* Wyposażenie
* Zajęcia
* Kadra

Pierwszym, podstawową możliwością systemu musi być jasne i czytelne przedstawienie dostępnych obiektów, wraz z ich położeniem, inwentarzem czy dostępnymi zajęciami. Istotną funkcjonalnością byłoby również sprawdzanie aktualnego obłożenia każdej z placówek, tak by móc zaplanować o której godzinie wyjście na trening nie spowoduje konieczności unikania zatłoczonych miejsc siłowni. Aktualnie niewielka liczba systemów oferuje taka możliwość, użytkownik więc musi posiłkować się intuicją czy na przykład danymi oferowanymi przez wyszukiwarkę Google, która informuje nas o średnim ruchu w danych godzinach. Implementacja takich udogodnień do samego systemu powoduje zwiększenie jego atrakcyjności dla użytkownika.

Każdy dostępny obiekt powinien zawierać także kompletny spis dostępnego wyposażenia. Większość siłowni zawiera dziesiątki czy nawet setki różnych rodzajów maszyn, bieżni, sztang czy hantli. Decyzja o wyborze placówki na którą klient będzie chciał uczęszczać będzie również dostępnością sprzętu który jest mu potrzebny w realizacji określonych celów treningowych. Jeżeli użytkownik nie otrzyma w żaden sposób informacji o dostępnym wyposażeniu może odczuć rozczarowanie po pierwszej wizycie, która bardzo często poprzedzona jest już zakupem karnetu czy opłaceniem opłaty wstępnej.

Kolejnym istotnym punktem jest wygodny w użyciu i przeglądaniu widok grafiku zajęć. Robienie zdjęć wywieszonym tabelkom czy przepisywanie danych z wpisu na mediach społecznościowych jest zajęciem czasochłonnym i niewygodnym. Samo zapisywanie się na zajęcia często musi być poprzedzone konsultacją z pracownikiem bądź trenerem, czy znajdzie się na nich miejsce i odpowiedni sprzęt. Łatwo dostępny grafik na stronie internetowej, z możliwością przeglądania i zapisania się na konkretne zajęcia ułatwia użytkownikowi znalezienie odpowiedniej dla niego aktywności bez konieczności udziału osób trzecich.

Ostatnim istotnym elementem powstającego systemu byłaby kadra dostępna w placówkach, czyli trenerzy prowadzący same zajęcia. Możliwość wcześniejszego sprawdzenia prowadzących aktywności, przejrzenie ich poprzednich treningów czy nawet wyszukanie o nich dodatkowych informacji ułatwia podjęcie decyzji o wyborze odpowiedniego miejsca ćwiczeń.

Dodatkowymi ważnymi funkcjonalnościami byłyby również:

* Obsługa karnetów
* Obecności na zajęciach i aktywnościach

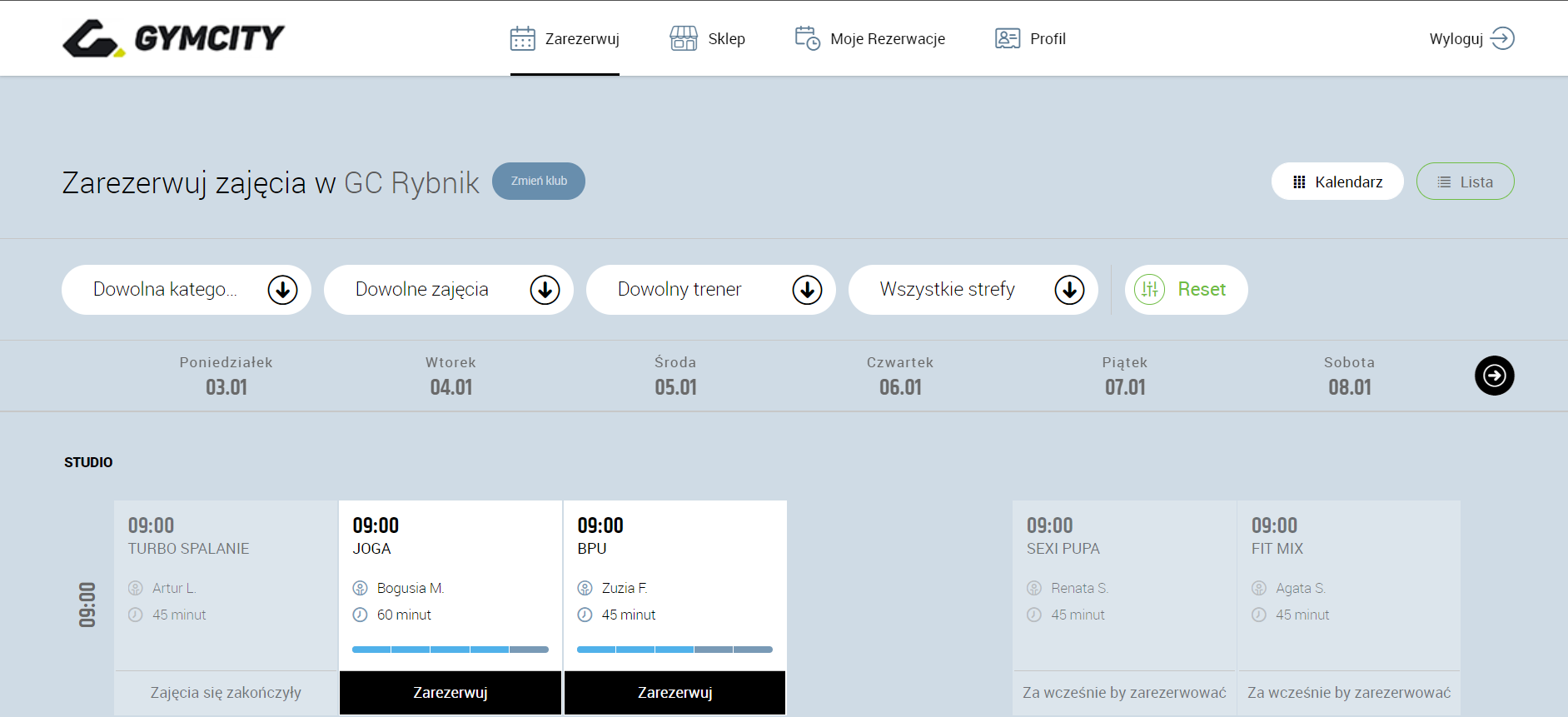
Użytkownik po rejestracji na stronie internetowej może dowolnie przeglądać cały system, sprawdzać dostępne w placówkach wyposażenie czy zajęcia. Kiedy dochodzi jednak do momentu wyrażanie chęci uczestnictwa w wybranych zajęciach czy zwykłym treningu na siłowni, wymagane jest zakupienie karnetu. Zakup karnetu obywać się musi poprzez profil użytkownika, tak by było to możliwie jak najszybsze i najprostsze w realizacji.

W samym profilu użytkownik powinien móc przeglądać również zajęcia na których uczestniczył czy same wizyty na siłowni, wraz z ich czasem trwania. Dostarcza to nie tylko cennych informacji dla klientów o ilości czy ich konsekwencji, ale również dostarcza pozytywnych wrażeń po dłuższych okresach regularnej aktywności fizycznej. Spełniony klient będzie chciał wracać na kolejne treningi, co docelowo doprowadzi do przedłużania karnetu i dalszego korzystanie z wybranej siłowni i systemu.

Wszystkie opisane powyżej funkcjonalności stworzone zostały z myślą o użytkowniku docelowym, czyli osobie korzystając z usług siłowni. Drugą stroną aplikacji są jednak pracownicy siłowni, obsługujący cały system. Ich zadaniem będzie inwentaryzacja siłowni, aktualizowanie grafików czy potwierdzanie obecności użytkowników na zajęciach. Dzięki korzystaniu z tego samego systemu co klienci będą oni w stanie lepiej zrozumieć ich potrzeby oraz rozwiązywać i wyłapywać powstałe problemy. Część pracownicza aplikacji również musi zachowywać w pełni profesjonalne standardy wykonania, tak by zarządzanie systemem było proste i przystępne. W tym celu powstaną specjalne formularze umożliwiające dodawanie wyposażenia, aktywności czy całych obiektów siłowni, będące jednocześnie odporne na błędy i możliwe nadużycia.

## 2.3 Opis istniejących rozwiązań

Oczywistym wnioskiem wynikającym z przytoczonej wcześniej liczby istniejących już teraz placówek jest to, że systemu do zarządzania siłowniami istnieją i są aktualnie używane. W kolejnej części pracy przedstawione zostaną niektóre rozwiązania stosowane obecnie przez siłownie w celu zarządzania nimi czy komunikacji z samym klientem.



Rysunek 2.2 Widok grafiku aplikacji GymCity

Na Rysunku 2.2 przedstawiony został ekran widoku grafiku zajęć w aplikacji GymCity. Istnieje tutaj również możliwość zmiany wybranego klubu dla którego sprawdzamy zajęcia czy filtrowanie po wybranych polach. Dodatkowymi zakładkami dostępnymi dla użytkownika są tutaj

* Sklep - zawierające produkty takie jak koszulki czy suplementy
* Moje Rezerwacje - zawierające zajęcia na które zapisaliśmy się
* Profil - informacje o danych osobowych czy karnecie i płatnościach

System GymCity dostępny jest aktualnie w 3 placówkach tej siłowni, obecnych w Bełchatowie, Rybniku i Wodzisławiu Śląskim. Styl aplikacji jest minimalistyczny i zawierający wszystkie potrzebne informacje, istotną zaletą tego systemu jest również jego nieustanny rozwój. Kilka miesięcy temu widok grafiku był nieinteraktywną tabelą, aktualnie natomiast umożliwia zapisanie się na zajęcia czy informacje o ilości wolnych miejsc. Jako minusy czy raczej brakujące elementy tego rozwiązania można wymienić

* Brak informacji o zajęciach i wizytach na siłowni które wydarzyły się w przeszłości
* Brak informacji o wyposażeniu konkretnych placówek

Dostęp do systemu GymCity jest również ograniczony poprzez konieczność założenia konta w placówce a następnie uiszczenia opłaty wstępnej przed rozpoczęciem korzystanie z serwisu. Widok grafiku zajęć posłużył za podwaliny systemu zaimplementowanego w docelowym projekcie.

Jak wspomniano już wcześniej niektóre kluby poza autorskimi aplikacjami i stronami internetowymi korzystają również z innych form kontaktu. Jedną z takich sieci siłowni jest Top Fitness.

Obraz zawierający tekst, osoba, wewnątrz

Opis wygenerowany automatycznie

Rysunek 2.3 Wpis informujący o zajęciach w klubie Top Fitness, Facebook

Obraz zawierający tekst, monitor, sprzęt elektroniczny, wyświetlanie

Opis wygenerowany automatycznie

Rysunek 2.4 Wpis informujący o odwołanych zajęciach, Facebook

Rysunki 2.3 oraz 2.4 przedstawiają sposób informowania o zajęciach czy ogólnych informacji o siłowni poprzez udział mediów społecznościowych, w tym wypadku serwisu Facebook. Informacje przedstawiane w taki sposób trafiają do ograniczonej liczby odbiorców, wymagają bowiem sprawdzania kilku miejsc jednocześnie aby zdobyć je wszystkie. Klub Top Fitness przedstawiony na powyższych rysunkach posiada również rozbudowaną stronę internetową z strefą klienta, grafikiem czy możliwością zakupu karnetu przez Internet.

Obraz zawierający tekst, zrzut ekranu, wewnątrz

Opis wygenerowany automatycznie

Rysunek 2.5 Strona Top Fitness, ekran zakupu karnetu

Formularz zakupu karnetu dostępny jest dla wszystkich nowych klientów siłowni. Istnieją możliwości wyboru czasu startu czy różne typy karnetów (czasowe, na ilość wejść, otwarte). Ekran grafiku jest przejrzysty i zawiera możliwość zapisania się na konkretne zajęcia, nie udostępnia jednak informacji o ilości uczestników. Sama strona wielokrotnie wymagała również długich czasów ładowania, co znacząco wpływa na komfort użytkowania i obsługi. Same ekrany dostępne dla klienta często zawierały przytłaczającą ilość informacji i utrudniały tym samym nawigację po systemie.

Jako najmniejsza z omawianych tutaj sieci siłowni Top Fitness mimo niewielkich zasięgów posiada kompleksową i podobną do konkurencji stronę internetową. Przy dalszym rozwoju aplikacji mogłaby się stać kompletną i spełniającą wszystkie potrzeby użytkowników.

Obraz zawierający tekst, zrzut ekranu, ekran

Opis wygenerowany automatycznie

Rysunek 2.6 Strona startowa, Smart Gym

Największym z przytoczonych tutaj przykładów systemów będzie Smart Gym, oferujący placówki w Zabrzu, Gliwicach czy Katowicach. Strona internetowa zawiera bardzo dużo informacji o dostępnych klubach, ich wystroju i wyposażeniu (galerie zdjęć) czy cennikach i sposobach kontaktu. Po zalogowaniu się istnieje także możliwość przeglądania i zapisywania się na zajęcia. Widok grafiku przedstawiono na Rysunku 2.7

Obraz zawierający stół

Opis wygenerowany automatycznie

Rysunek 2.7 Widok grafiku zajęć, Smart Gym

Aplikacja wykorzystywana przez sieć siłowni Smart Gym spełnia sporą część wymagań postawionych przed systemem tworzonym w ramach pracy inżynierskiej, nie zawiera ona jednak wszystkich wykorzystanych innowacji.

Wszystkie z powyższych systemów łączy również rozdzielenie na część zewnętrzną w formie strony internetowej, jak również strefę klienta dostępna po zalogowaniu, zazwyczaj ograniczonym wcześniejszym zakupem karnetu.

Przedstawione powyżej systemy łączy wiele wspólnych cech, wynikających głównie z funkcjonalności jakie potrzebne są do zarządzania siłownią. Porównując przejrzane więc systemu z analizą tematu przedstawioną powyżej, jako elementy wspólne systemów można wyszczególnić:

* Widok grafiku zajęć z możliwością zapisywania się na zajęcia
* Możliwość zakupu karnetu

Jako elementy niewystępujące w sprawdzonych systemach wymienić można:

* Możliwość przeglądania wyposażenia siłowni
* Sprawdzanie aktualnej ilości ćwiczących
* Wyświetlenie wszystkich wizyt i odbytych treningów na siłowni

Analiza systemów będących aktualnie w użyciu umożliwiła poszerzenie perspektyw oraz odnalezienie brakujących im elementów. To właśnie te dodatkowe funkcjonalności nieobecne w większości istniejących systemów mają docelowo dać przewagę tworzonej aplikacji oraz przyciągnąć do niej, a w konsekwencji do obsługiwanych siłowni, większą ilość użytkowników.

Dodatkowymi istotnymi czynnikami przy tworzeniu aplikacji będzie jej wydajność, bezpieczeństwo oraz stabilność. Część testowanych systemów miała trudności z utrzymaniem ciągłości działania, zdarzały się niespodziewane błędy czy długie czasy oczekiwania na załadowanie się stron. Intuicyjność samych warstw prezentacji jest równie istotną kwestią, styl graficzny powstałego systemu musi być spójny i łatwo przekazujący wszystkie potrzebne informacje do odbiorcy.

Konkurencja w obrębie systemów do zarządzania siłowniami jest bardzo duża, tak samo jak rynek i zapotrzebowanie. Wykorzystanie najnowszych technologii oraz zarówno sprawdzonych jak i nowych innowacyjnych rozwiązań może zapewnić potrzebną siłę przebicia i powiew świeżości na rynku fitness.

# Rozdział 3 Wymagania i narzędzia

W rozdziale przedstawiono wymagania funkcjonalne (w tym diagram przypadków użycia) oraz niefunkcjonalne systemu, wraz z opisem wykorzystywanych narzędzi i metodologii w jego tworzeniu. Dokładny opis założeń postawionych w procesie projektowania w pełni umożliwia zrozumienie potrzeby powstania aplikacji.

## 3.1 Wymagania niefunkcjonalne

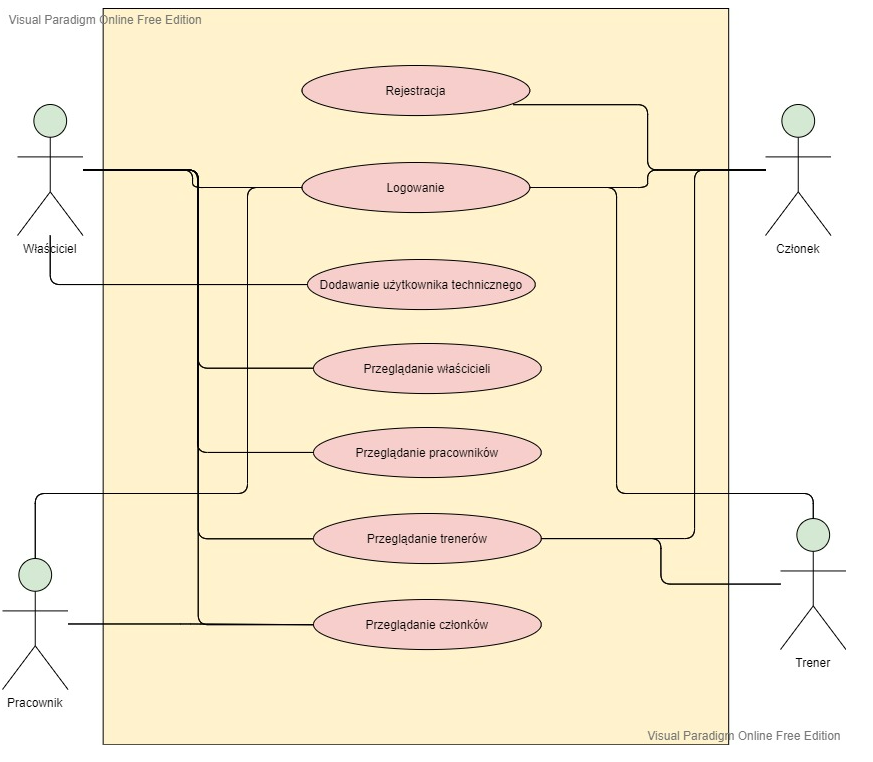
Wymaganie niefunkcjonalne aplikacji określają głównie oczekiwania co do samej jakości działania tworzonego systemu jak również jego pożądane cechy. Opisują warunki, w jakich powinien działać system. Określanie wymagań niefunkcjonalnych odgrywa bardzo ważną rolę w procesie tworzenia aplikacji.

Określono następujące wymagania niefunkcjonalne systemu:

* **Dostępność** – system powinien być dostępny całodobowo, przez siedem dni w tygodniu. Jedyne przerwy w działaniu systemu przeznaczone są na wdrożenia nowych wersji oprogramowania, w godzinach nocnych. Dostęp z poziomu każdej obecnie obsługiwanej przeglądarki internetowej, również z urządzeń mobilnych.
* **Wydajność** – system musi radzić sobie z dużą liczbą użytkowników korzystających z aplikacji jednocześnie. Zaprojektowana architektura musi umożliwiać skalowanie aplikacji by obsłużyć rosnącą ilość osób korzystających z serwisu. Założenia startowe aplikacji przedstawiają się następująco
  + Maksymalna liczba użytkowników korzystających jednocześnie z aplikacji: 1000
  + Maksymalny czas na odpowiedź serwera na zapytanie użytkownika: 5 sekund
* **Wsparcie** – w ramach awarii systemu użytkownicy muszą mieć możliwość zgłaszania znajdowanych błędów, określony powinien być również sposób monitorowania oraz maksymalny czas do naprawy znalezionych nieprawidłowości. Określono wymagania:
  + Email jako forma kontaktu do zgłaszania błędów przez użytkowników
  + Monitorowanie aplikacji w formie codziennego przeglądania logów aplikacji
  + Maksymalny czas na naprawę błędu aplikacji wynika z poziomu jego krytyczności. Czas liczony jest od momentu przyjęcia zgłoszenia przez dostawcę, który wynosi 1 dzień roboczy.
    - Niski (7 dni roboczych)
    - Średni (5 dni roboczych)
    - Wysoki (3 dni robocze)
    - Krytyczny (1 dzień roboczy)
* **Bezpieczeństwo** – całość funkcjonalności aplikacji zabezpieczony jest poprzez użycie systemów bezpieczeństwa, w tym tokenów dostępu oraz bezpiecznych haseł aplikacji, szyfrowanych w bazie danych. Dane osobowe zawarte w aplikacji powinny być zabezpieczone przed nieautoryzowanym dostępem
* **Wdrożenie** – w przypadku dalszego rozwoju aplikacji nowe funkcjonalności dostarczane będą wykorzystując metodyki zwinne w formie iteracyjnej. Wdrożenia odbywać się będą w godzinach nocnych, w celu zminimalizowania niedostępności systemu dla użytkowników.
* **Użyteczność** – wymagania dotyczące wyglądu interfejsu użytkownika, jego wygląd i sposobach interakcji z użytkownikiem. Określono wymagania:
  + Responsywność, strona wyglądająca poprawnie zarówno na urządzeniach stacjonarnych jak i mobilnych
  + Interfejs użytkownika przejrzysty i łatwy w obsłudze
  + Stonowana i nowoczesna kolorystyka i projekt.

## 3.2 Wymagania funkcjonalne

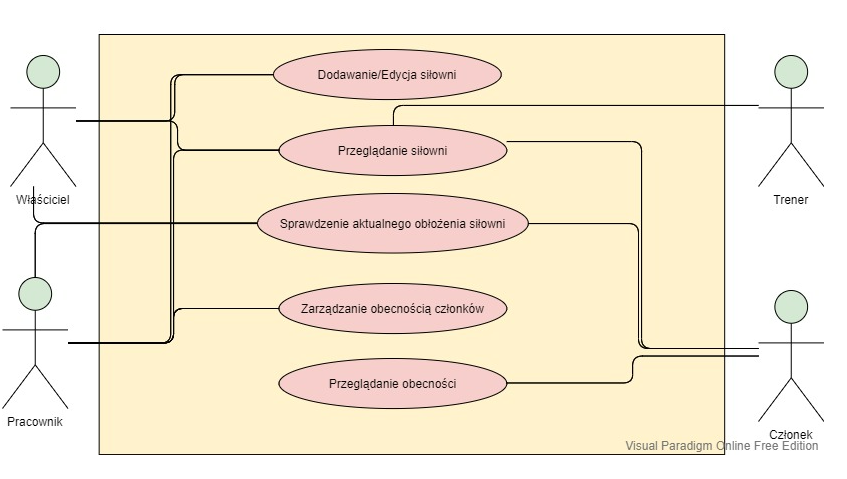
Wymagania funkcjonalne definiują funkcjonalności dostępne w systemie. Ich zarys utworzony został w początkowym okresie tworzenia systemu, by następnie ewoluować i zmieniać się wraz z postępem prac. W dalszej części rozdziału przedstawione zostaną diagramy przypadków użycia dla poszczególnych części systemu, wraz z dokładniejszym opisem wymaganego sposobu działania.



Rysunek 3.1 Diagram przypadków użycia, domena użytkowników

Domena użytkowników widoczna na rysunku 3.1 zawiera w sobie informacje odnośnie kont użytkowników, ich zakładania oraz dostępu do nich. Poszczególne wymagania to:

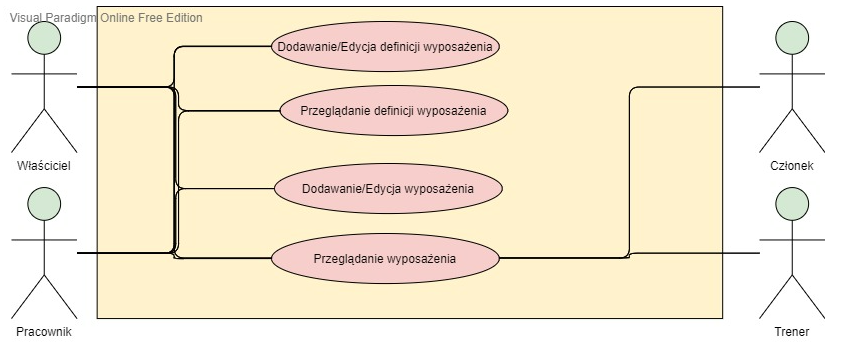
* **Rejestracja** – możliwość rejestracji w systemie poprzez dedykowany formularz. Konta tworzone poprzez proces rejestracji mogą mieć tylko rolę Członka.
* **Logowanie** – możliwość autentykacji w systemie za pomocą ustalonej wcześniej nazwy użytkownika i hasła.
* **Dodawanie użytkownika technicznego** – możliwość utworzenia kont użytkowników z inna rola niż Członek.
* **Przeglądanie właścicieli** – możliwość wyświetlenia listy wszystkich użytkowników w roli właściciela.
* **Przeglądanie pracowników** – możliwość wyświetlenia listy wszystkich użytkowników w roli pracownika.
* **Przeglądanie trenerów** – możliwość wyświetlenia listy wszystkich użytkowników w roli trenera.
* **Przeglądanie członków** – możliwość wyświetlenia listy wszystkich użytkowników w roli członka.



Rysunek 3.2 Diagram przypadków użycia, domena siłowni

Domena siłowni widoczna na Rysunku 3.2 zawiera w sobie podstawowe informacje oraz operacje o zarządzaniu obiektami siłowni oraz obecności członków na niej. Poszczególne wymagania to:

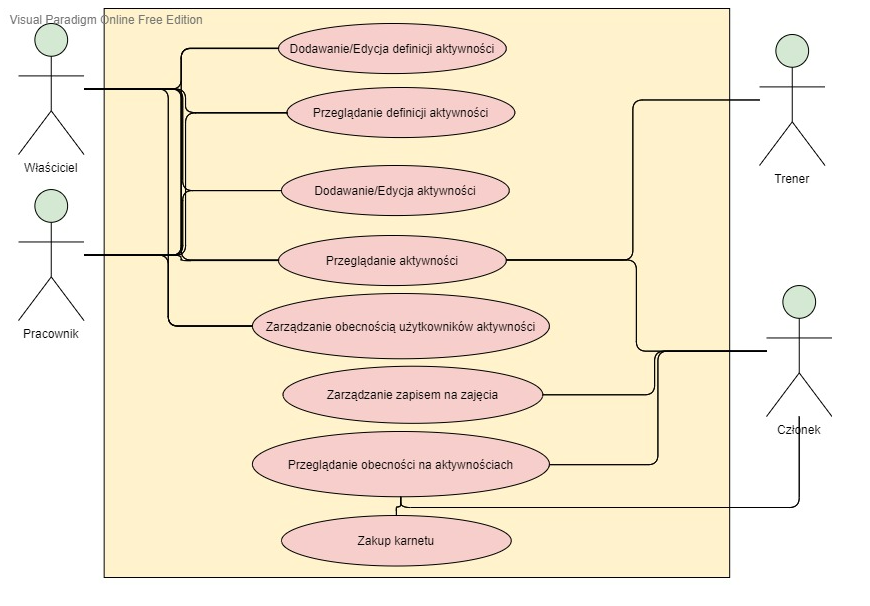
* **Dodawanie/Edycja siłowni** – poprzez odpowiednie formularze istnieje możliwość utworzenia i późniejszej edycji nowych obiektów siłowni
* **Przeglądanie siłowni** – możliwość przeglądanie wszystkich dostępnych w systemie siłowni
* **Sprawdzenie aktualnego obłożenia siłowni** – możliwość sprawdzenia aktualnej liczby osób ćwiczących na siłowni
* **Zarządzanie obecnością użytkowników** – możliwość odnotowania chwili przybycia członka na siłownię a następnie momentu wyjścia.
* **Przeglądanie obecności** – członkowie mogą sprawdzić listę swoich odnotowanych obecności na siłowni



Rysunek 3.3 Diagram przypadków użycia, domena wyposażenia

Domena wyposażenia widoczna na Rysunku 3.3 zawiera informacje o dostępnych w systemie rodzajach wyposażenia, ich definicjach oraz ilości na konkretnych siłowniach. Poszczególne wymagania to:

* **Dodawanie/Edycja definicji wyposażenia** – poprzez dedykowane formularze istnieje możliwość dodawania i późniejszej edycji definicji wyposażenia
* **Przeglądanie definicji wyposażenia** – możliwość wyświetlenia i przeglądania wszystkich definicji wyposażenia dostępnych w systemie
* **Dodawanie/Edycja wyposażenia** – możliwość przypisywania do konkretnych siłowni
* **Przeglądanie wyposażenia** – możliwość przeglądania dostępnego wyposażenia na konkretnych siłowniach



Rysunek 3.4 Diagram przypadków użycia, domena aktywności

Domena aktywności widoczna na Rysunku 3.4 zawiera w sobie informacje o definicjach aktywności istniejących w systemie oraz ich organizację w grafiki zajęć przypisane do konkretnych siłowni. Częścią tej domeny są również informacje o karnecie użytkownika oraz odnotowywanie obecności użytkowników na aktywnościach. Poszczególne wymagania to:

* **Dodawanie/Edycja definicji aktywności** – poprzez dedykowane formularze istnieje możliwość dodawania i późniejszej edycji definicji aktywności
* **Przeglądanie definicji aktywności** – możliwość wyświetlenia i przeglądania wszystkich definicji aktywności dostępnych w systemie
* **Dodawanie/Edycja aktywności** – możliwość przypisywania aktywności do konkretnych siłowni, ustalając moment ich rozpoczęcia oraz zakończenia czy ilość osób mogących brać w nich udział.
* **Przeglądanie aktywności** – możliwość przeglądania dostępnych aktywności na konkretnych siłowniach
* **Zarządzanie obecnością użytkowników aktywności** – możliwość odnotowania przybycia lub jego braku na zajęcia przez zapisanego na nie użytkownika
* **Zarządzanie zapisem na zajęcia** – możliwość zapisania oraz wypisania się z dowolnych nierozpoczętych zajęć
* **Przeglądanie obecności na aktywnościach** - członkowie mogą sprawdzić listę swoich odnotowanych obecności na aktywnościach
* **Zakup karnetu** – możliwość zakupu karnetu przez członków klubu. Dzięki zakupie karnetu istnieje możliwość zapisu na zajęcia

Wyszczególnione wymagania funkcjonalne umożliwiły stworzenie systemu w pełni spełniającego wstępne założenia, jak również pozwalające na skuteczne wdrożenie i używanie systemu w praktyce. Wszystkie podstawowe potrzeby oraz oczekiwania użytkowników względem systemu zarządzania siecią siłowni zostały spełnione. Wymagania funkcjonalne zostały również zaprojektowane z myślą o ich przyszłej rozbudowie i udoskonalaniu.

## 3.4 Wykorzystywane narzędzia

Do utworzenia projektu wykorzystywanych zostało wiele narzędzi ułatwiających i przyspieszających pracę. Najważniejsze z nich wymieniono poniżej.

* **Git** – rozproszony system kontroli wersji. Korzystanie z tego narzędzia umożliwiło ustrukturyzowanie pracy nad systemem. Każda nowa funkcjonalność tworzona była na nowej gałęzi, tak by jednocześnie móc pracować nad kilkoma funkcjonalnościami i w razie potrzeby wycofywać błędne zmiany. Śledzenie wszystkich zmian kodu systemu umożliwia również swobodne przeglądanie historii powstawania aplikacji
* **IntelliJ IDEA** – zintegrowane środowisko programistyczne (IDE) umożliwiło łatwiejsze zarządzanie powstającym kodem, jego automatyczna organizację czy uproszczenia w uruchamianiu i testowaniu aplikacji.
* **Google Chrome** – przeglądarka internetowa, wykorzystywana w celu uruchamiania oraz testowania warstwy prezentacji powstałego systemu. Dostęp do narzędzi developerskich ułatwiał również znajdowanie i naprawę błędów.
* **GitHub** – hosting umożliwiający zarządzanie repozytoriami Git. GitHub przechowywał repozytorium kodów zawierające całość systemu. Dodatkowo korzystając z funkcji strony powstała tam również tablica Kanban oraz wszelkie notatki wymagane do rozwoju systemu.
* **pgAdmin** – program do zarządzania i pracy z bazami danych PostgreSQL. Umożliwiał utworzenie i analizę utworzonej docelowo przez aplikację bazy danych.
* **VisualParadigm Online** – narzędzie w formie aplikacji internetowej używane w celu utworzenia diagramów UML

## 3.5 Metodyka pracy nad projektem

Na przestrzeni lat w tworzeniu oprogramowania powstało wiele metodyk i sposobów rozwoju systemów, tak by maksymalnie zwiększać efektywność samej pracy, jak również gwarantować stabilność i odporność na błędu tworzonego kodu. Przy tworzeniu systemu wykorzystywane były następujące metodyki i podejścia do programowania.

* **DDD** (ang. *Domain Driven Design),* domenowe podejście do programowania
* **TDD**, (ang. *Test Driven Development),* programowanie sterowane testami
* **Agile**, zwinne metodyki programowania (w tym Kanban)

Z wymienionych powyżej metodyk wybrane zostały konkretne części umożliwiające łatwiejszą i bardziej zorganizowaną pracę nad projektem. Pełne zaadoptowanie wybranego podejścia było niemożliwe z powodu ram czasowych powstawania systemu oraz jednoosobowego zespołu tworzącego go.

Domenowe podejście do programowania zakłada tworzenie oprogramowania w oparciu o dziedzinę. Istotą programu według podejścia domenowego jest jego zdolność do rozwiązywania problemów powiązanych oraz wynikających z obranej przez niego dziedziny[[2]](#footnote-2). Sama dziedzina jest odzwierciedleniem wybranego fragmentu rzeczywistości, zaprojektowana tak by wiernie oddawała jego strukturę i potrzeby. Im bardziej skomplikowana i złożona dziedzina, tym większą pracę będą musieli wykonać programiści by zdobyć odpowiednią wiedzę biznesową. Poszczególne techniki zapożyczone z podejścia domenowego opisane zostały w rozdziale 5.

Programowanie sterowane testami (TDD) zakłada pierwszeństwo tworzenia testów przed właściwą logiką systemu. Istotnymi zasadami takiego podejścia jest pisanie nowego kody tylko wtedy, kiedy poprzednio utworzony nie przechodzi pomyślnie utworzonych już wcześniej testów, a następnie usuwaniu ewentualnie powstałych w tym procesie duplikacji (czyli proces refaktoryzacji kodu)[[3]](#footnote-3). Wykorzystywana jest również zasada czerwone-zielone-refaktoryzacja, czyli najpierw utworzenie testu który nie ma szans na powodzenie, następnie stworzenie kodu w najprostszej postaci tak by test zakończył się pomyślnie i w ostatnim kroku refaktoryzacja powstałego w ten sposób kodu do postaci zgodnej z wymaganiami odnośnie struktury systemu. Techniki wykorzystane z podejścia TDD zostały szczegółowo opisane w rozdziale 6.

Struktura prac prowadzonych nad systemem zapożyczała techniki z zwinnych metodyk tworzenia oprogramowania (Agile), oraz Kanban. Zadania wymagane do przejścia do kolejnych faz tworzenia systemu dzielone były na mniejsze podzadania które następnie z użyciem wirtualnej tablicy zmieniane były na kartki przenoszone pomiędzy różnymi kolumnami oznaczającymi statusy. Z podejścia zwinnego zapożyczony został również ogólny sposób tworzenia projektu. Prace rozpoczęte zostały analizą dziedziny systemu, określone zostały wymagania oraz założenia. W zwinnym programowaniu analiza jednak nigdy się nie kończy[[4]](#footnote-4). Prace podzielone zostały na trwające jeden tydzień iteracje. Iteracja zerowa służyła wspomnianej wcześniej wstępnej analizie, która została następnie przetworzona do zwizualizowanych form używanych przez Kanban. Każda kolejna iteracja zakładała implementację części wybranych funkcjonalności, z jednoczesnym zachowaniem marginesu błędu na dodatkową analizę czy zmianę konkretnych funkcjonalności.

Wykorzystanie podejścia zwinnego umożliwiło iteracyjny rozwój systemu z nowszymi wersjami zawierającymi kolejne funkcjonalności, a w konsekwencji stały kontakt i sprzężenie zwrotne z docelowymi odbiorcami aplikacji, tak by móc dynamicznie dostosowywać projekt do ich potrzeb oraz oczekiwań.

Pod względem technicznym tworzone oprogramowanie wykonywane było z uwzględnieniem dobrych praktyk zarówno w odniesieniu do tworzonego kodu jak i samej architektury systemu. Celem dobrze wybranej i zaimplementowanej architektury systemu jest efektywne zmniejszenie liczby ludzi potrzebnych do zbudowania a następnie rozwoju i utrzymania systemu[[5]](#footnote-5). Odpowiednie wyznaczanie odpowiedzialności komponentów, granic systemu oraz interfejsów i ich wykorzystania gwarantuje przejrzystość i stabilność systemu na poziomie jego struktury. Stosowanie się również do zasad pisania dobrego kodu (w tym mneomnika SOLID stworzonego przez Roberta C. Martina) oraz wykorzystanie wzorców projektowych umożliwiło stworzenie systemu który w przyszłości mógłby być nadal rozwijany przez dowolnego programistę. Istnieje wiele definicji czystego kodu jedną z nich jest określenie go jako prostego i eleganckiego. Taki kod można czytać jak dobrze napisaną prozę, nie zaciemnia od zamiarów projektanta a dzięki zastosowaniu trafnych abstrakcji w pełni przekazuje jego intencje[[6]](#footnote-6).

# Rozdział 4 Specyfikacja zewnętrzna

W rozdziale przedstawiono specyfikację zewnętrzną aplikacji. Zawiera on informacje o wymaganiach sprzętowych i programowych, wymaganych narzędziach oraz sposobie instalacji i uruchomienia. Rozdział wypełniają również przykłady działania systemu, wymagane akcje i wyniki dla przykładowych operacji. Zawarto również informacje dotyczące bezpieczeństwa aplikacji, rolach i uprawnieniach użytkowników oraz zarządzania systemem w czasie jego działania.

## 4.1 Wymagania sprzętowe i programowe

Przy tworzeniu aplikacji webowych z wykorzystaniem technologii takich jak Java i JavaScript, czy szkieletów programistycznych określenie jednoznacznie minimalnych wymagań sprzętowych jest często niemożliwe. Obciążenie systemu zależy od rozmiarów aplikacji, natężenia ruchu czy nawet wykorzystywanych bibliotek i argumentów. Przedstawione poniżej wymagania sprzętowe oparte są więc o zalecenia dotyczące bazy danych PostgreSQL[[7]](#footnote-7), oraz miejsce zajmowane przez aplikację wraz z wszystkimi wymaganymi bibliotekami.

* System operacyjny Windows, Mac lub Linux
* Procesor o pamięci co najmniej 1 GHz
* 2 GB pamięci RAM
* 5 GB pamięci HDD

Wymaganie programowe przedstawiają się następująco:

* Java JDK w wersji 11
* Apache Maven w wersji 3
* PostgreSQL w wersji 14
* Node.js w wersji 16.13.0
* Npm w wersji 8.1.0
* Angular w wersji 13.0
* Dowolona przeglądarka internetowa z obsługą JavaScript (domyślnie Google Chrome)

Wykorzystany stos technologiczny jest jednym z najpopularniejszych obecnie dostępnych rozwiązań do tworzenia systemów internetowych, dokumentacja jest szeroko i łatwo dostępna co upraszcza konfigurację i uruchomienie środowiska.

## 4.2 Sposób instalacji i uruchamianie

Uruchomienie całego systemu wymaga wykonania najpierw kilku niezbędnych kroków przygotowawczych. Na stacji roboczej na której chcemy uruchomić aplikację powinny być zainstalowane i dostępne wszystkie wymienione wcześniej programy w odpowiednich wersjach. Z punktu widzenia wygody instalacji i operowania na aplikacji przydatne będzie również ustawienie zmiennych środowiskowych systemu. Poniższa instrukcja opisuje sposób instalacji i uruchomienia na środowisku wyposażonym w system operacyjny Windows. W przypadku innego systemu proces instalacji może się różnić.

Na samym początku procesu instalacji warto zadbać o odpowiednio skonfigurowane zmienne środowiskowe. Ułatwią one dalszą pracę i uruchamianie wszystkich części systemu.

* **JAVA\_HOME** – kierująca do katalogu z instalacją JDK Javy w wersji 11
* **M2\_HOME** – kierująca do miejsca instalacji oprogramowania Apache Maven
* **MAVEN\_HOME** – j. w.

Dodatkowo, aby móc uruchamiać potrzebne programy z linii poleceń bez konieczności wskazywania konkretnych ścieżek do plików wykonywalnych pomocne powinno być ustawienie powyższych zmiennych do zmiennej środowiskowej PATH. Poza wymienionymi dotychczas zmiennymi powinny znaleźć się tam odniesienia do Node.js oraz menadżera pakietów.

Kolejnym krokiem przygotowawczym będzie utworzenie bazy danych PostgreSQL, do której aplikacja będzie mogła się połączyć i uruchomić odpowiednie skrypty tworzące i uzupełniające strukturę. Aplikacja w dostarczonej konfiguracji oczekuje połączenia do bazy danych dostępnej z następującymi parametrami:

* **Url** – *jdbc:postgresql://localhost:5432/gymex*
* **Nazwa użytkownika** – *postgres*
* **Hasło** – *root*

Utworzona baza danych musi więc zostać zapisana pod tym adresem. Najprostszym sposobem utworzenia odpowiedniej bazy danych jest uruchomienie narzędzia *pgAdmin* instalowanego razem z całym pakietem narzędzi PostgreSQL, ustawienie hasła *root* dla bazowego użytkownika (*postgres)* a następnie utworzenie bazy danych na lokalnej maszynie (stąd adres *localhost).* Proces dodawania nowej bazy danych widoczny jest na Rysunku 4.1.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rys. 4.1 Utworzenie nowej bazy danych w narzędziu pgAdmin

Z utworzoną już bazą danych można przejść do kolejnego kroku, którym będzie uruchomienie warstwy serwera. Samodzielne skompilowanie kodów Kodwych umożliwia lepsze zapoznanie się z systemem oraz zmianę niektórych parametrów konfiguracji, jak np. adresy, nazwy użytkownika i hasło do bazy danych). W kolejnych krokach przedstawione zostaną wszystkie wymagane operacje do przeprowadzenia zarówno kompilacji jak i startu systemu.

W głównym katalogu aplikacji (znajduje się w nim plik *pom.xml)* należy uruchomić następujące polecenia.

* ***mvn -v*** – wypisuje na ekran używana wersję oprogramowania Apache Maven, jak również wykorzystywaną wersję Javy. Jest to polecenie kontrolne, dzięki któremu wiemy czy skonfigurowaliśmy tą część poprawnie
* ***mvn clean install*** – polecenie najpierw wykonuje czyszczenie jakichkolwiek zbudowanych plików aplikacji, a następnie kompiluje kody programu, wykonuje wszystkie testy oraz tworzy plik .jar służący do uruchomienia aplikacji. W przypadku błędu testów bądź kompilacji plik nie zostanie wygenerowany a użytkownik zostanie poinformowany o błędzie
* ***java -jar -Dspring.profiles.active=runtime target/gymex-1.jar***– polecenie uruchamia aplikację Spring Boot zapisaną w pliku jar wskazanym na końcu polecenia. Podany profil *runtime* gwarantuje, że aplikacja podłączy się pod bazę danych dodaną przez nas wcześniej w PostgreSQL. Drugim dostępnym profilem jest *integration*, który uruchamia system z bazą danych H2, zapisującą dane w pamięci które tracone są po wyłączeniu aplikacji

Jeżeli wszystko zostało skonfigurowane pomyślnie warstwa serwera uruchomi się i następnie połączy z bazą danych. Wykorzystane w systemie ustawienia Hibernate za każdym razem po połączeniu z bazą danych usuną wszystkie jej obiekty i następnie ponownie od zera utworzą pusty schemat. Po jego utworzeniu aplikacja automatycznie uzupełni bazę danych danymi testowymi (tylko dla profilu *runtime)*. Bazowym portem uruchomienia warstwy serwera jest port 8085.

Ostatnim krokiem instalacji systemu jest uruchomienie warstwy prezentacji. Poniższe kroki zakładają, że użytkownik znajduje się w katalogu *src/gymex-ui*. Do instalacji wymagana jest instalacja Node.js oraz menadżera pakietów Npm, którym następnie można zainstalować bilbiotekę Angular w odpowiedniej wersji. W przypadku korzystania z gotowego pliku należy od razu przejść doDo uruchomienia warstwy prezentacji potrzebne będą następujące kroki.

* ***node -v*** *–* polecenie kontrolne sprawdzające poprawność skonfigurowania Node.js
* ***npm -*v** – polecenie kontrolne sprawdzające poprawność skonfigurowania npm
* ***npm ci*** *–* polecenie bliźniacze do poprzednio używanego *mvn clean install*. Wykonuje ono najpierw czyszczenie wszystkich zainstalowanych pakietów wykorzystywanych przez warstwę prezentacji a następnie pobranie odpowiednich bibliotek używanych przez system
* ***ng serve*** *–* polecenie budujące i następnie uruchamiające warstwę prezentacji na porcie 4200. W przypadku błędów kompilacji aplikacja nie uruchomi się.

Po wykonaniu wszystkich powyższych kroków cały system powinien być już gotowy do działania. Trzeba tutaj zaznaczyć, że przedstawiona instrukcja instalacji i uruchamiania pokazuje system gotowy do odpalenia na lokalnie użytkowanej maszynie. Aplikacja dostępna będzie pod adresem URL *localhost:4200* w przeglądarce. Rozwiązanie to ma swoje oczywiste plusy w postaci szybkich modyfikacji i ogólnemu sprzyjaniu rozwojowi aplikacji, jednak docelowo system musiałby być uruchomiony na zdalnym serwerze dostępnym dla wszystkich użytkowników Internetu. Przy takich potrzebach aplikacja zostałaby skonteneryzowana, podpięte zostałyby również mechanizmy ciągłej integracji i ciągłego cyklu produkcyjnego (ang. *continious integration, continious deployment CI/CD)*, opisanych szerzej w przyszłych planach rozwoju aplikacji.

## Użytkownicy i bezpieczeństwo

W systemie istnieją cztery główne typy użytkowników, dzielących ich zarówno na użytkowników technicznych (zarządzających systemem) jak i korzystających z niego. Zastosowany podział naturalnie wynika z wymaganych funkcjonalności systemu i ułatwia zarządzanie kwestiami bezpieczeństwa danych oraz dostępnych akcji.

Role techniczne dostępne w systemie to:

* **Właściciel** (ang. *owner)* – właściwie konto posiadające najwyższe uprawnienia w systemie, mogące być nazywane również administratorem. Umożliwia pełne zarządzanie systemem jak również dodawanie innych użytkowników.
* **Pracownik** (ang. *employee)* – konto techniczne przeznaczone dla pracowników siłowni. Umożliwia zarządzanie system w zakresie wyposażenia, grafików czy potwierdzania obecności na zajęciach. Wszystkie uprawnienia pracownika dostępne są również dla właściciela

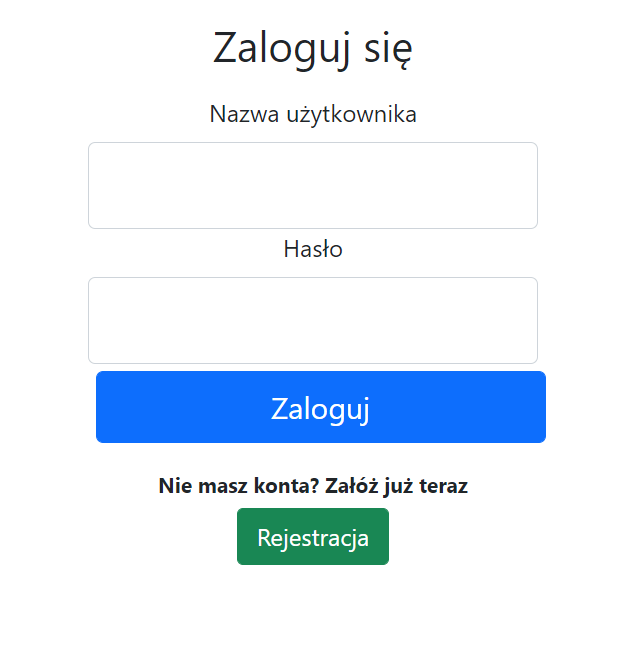
Role klientów docelowych to:

* **Trener** (ang. *Trainer) –* konto przeznaczone dla trenerów przypisywanych do poszczególnych zajęć na siłowni. Jest to konto posiadające najmniejsze uprawnienia, zmieniające portal w Kod informacji o nadchodzących zajęciach osoby będącej trenerem.
* **Członek** (ang. *Member)* – konto użytkownika przeznaczone dla klienta siłowni. Zakładane poprzez samodzielną rejestrację w serwisie. Konto to umożliwia dostęp do wszystkich wymaganych funkcjonalności systemu dla klientów, w tym przeglądanie i zapisywanie się na zajęcia, dostęp do historii swoich treningów czy sprawdzanie aktualnego stanu obłożenia siłowni.

Wymieniony wyżej zestaw ról dostępnych dla użytkowników skutecznie umożliwił podział systemu na części dostępne dla odpowiednich użytkowników. Szczegóły systemu zabezpieczeń i ich implementacje szerzej zostały opisane w rozdziale 5. W dalszej części specyfikacji zewnętrznej wyszczególnione zostaną poszczególne ekrany aplikacji wraz z informacją jacy użytkownicy mają do nich dostęp. Ułatwi to osadzenie wyżej wspomnianych typów kont i ról w obrazie całości systemu.

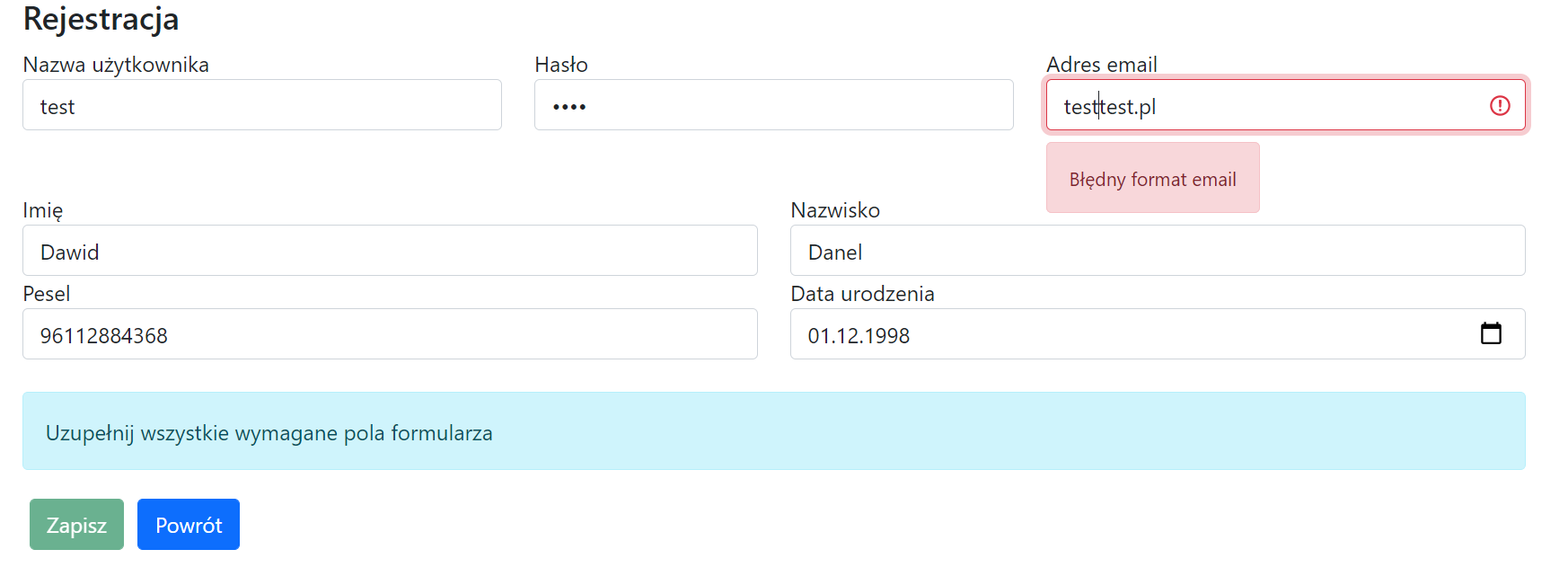
## Interfejs użytkownika

Interfejs użytkownika zaprojektowany został od podstaw z myślą o prostym, przystępnych i responsywnym systemie, gwarantującym szybkość i stabilność działania na dowolnych urządzeniu z opcją uruchomienia przeglądarki internetowej. Wykorzystanie szkieletu programistycznego Angular ułatwiło całość implementacji poprzez zamknięcie konkretnych ekranów w przystosowanych do tego celu komponentach. W dalszej części przedstawione zostaną wybrane ekrany oraz ich cele.



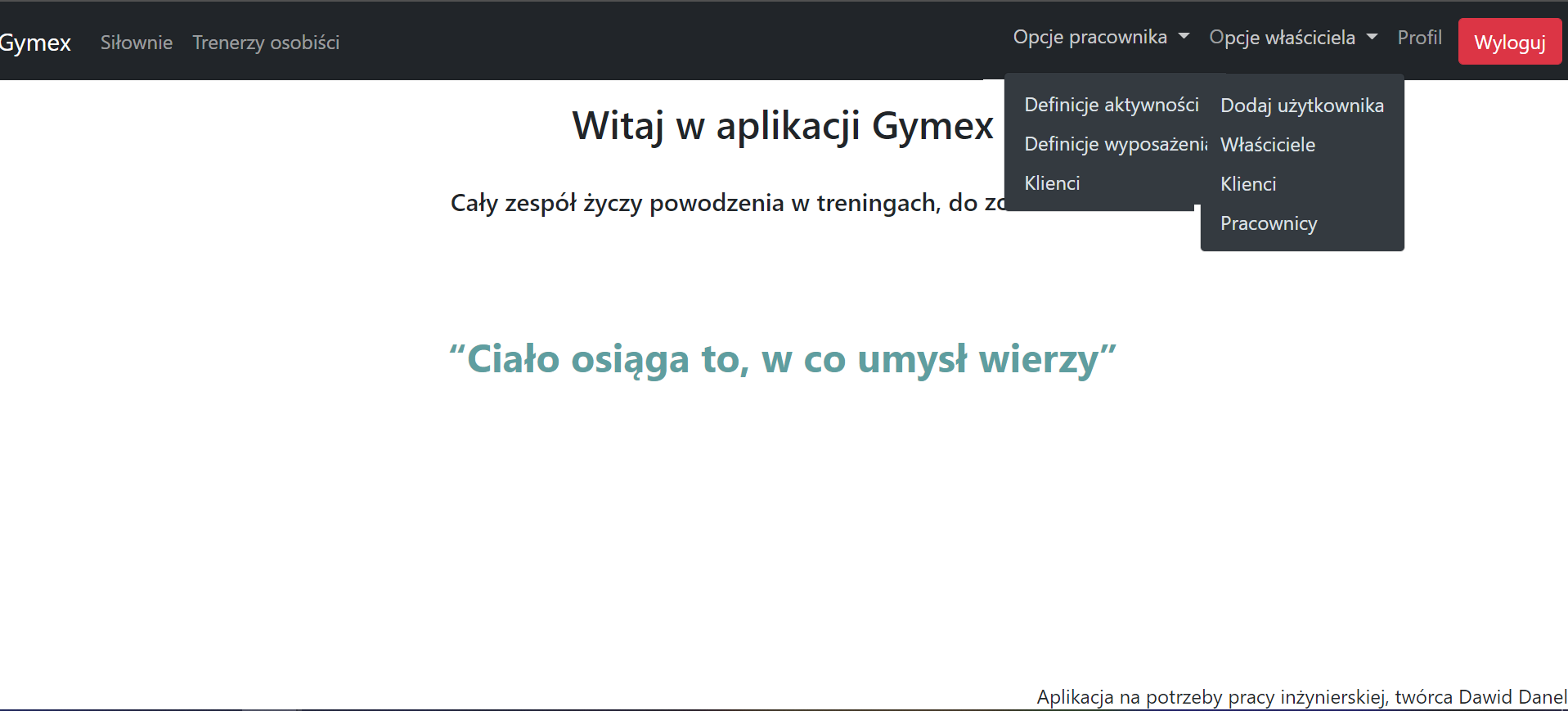
Rysunek 4.2 Ekran logowania

Ekran logowania widoczny na Rysunku 4.2 jest jednym punktem wejścia do systemu. Wymaga on podania nazwy użytkownika oraz hasła, a następnie poprzez przycisk *Zaloguj* w przypadku podania prawidłowych danych zautoryzuje użytkownika i przeniesie na stronę startową aplikacji. Drugą możliwością jest zarejestrowanie się w systemie dla nowych użytkowników. Po użyciu przycisku *Rejestracja* otworzy się odpowiedni formularz umożliwiający założenie konta.



Rysunek 4.3 Ekran rejestracji

Ekran rejestracji wymaga podania podstawowych danych osobowych oraz technicznych potrzebnych do założenia konta. Na Rysunku 4.3 widoczny jest również błąd walidacji formularza, w tym przypadku oznaczający błędny format adresu email. Wszystkie formularze aplikacji są w pełni walidowane, tak aby dane przekazywane do warstwy serwera były poprawne. Przycisk *Zapisz* jest w tym momencie zablokowany, do momentu w którym wszystkie pola formularza wypełnione zostaną prawidłowymi danymi.



Rysunek 4.4 Ekran startowy

Ekran startowy (Rysunek 4.4) aplikacji zawiera w sobie zmieniający się w czasie cytat motywacyjny oraz odniesienia do wszystkich części aplikacji. Widoczny w górnej części ekranu pasek nawigacyjny dostosowuje swoją treść w zależności od typu zalogowane użytkownika. Idąc od lewej strony ekranu odpowiednie opcje prowadzą do ekranów:

* **Siłownie** – ekran z widoczną listą wszystkich siłowni
* **Trenerzy osobiści** – wykaz trenerów dostępnych na wszystkich siłowniach, wraz z informacjami o typach prowadzonych zajęć
* **Opcje pracownika** – widoczna tylko dla typów kont *Pracownik* i *Właściciel*. Dostęp do komponentów z rozwijanej listy również jest ograniczony dla podanych wcześniej ról. Opcje dostępne dla *Pracownika* to
  + **Definicje aktywności** – zarządzanie definicjami aktywności systemu, czyli dodawanie i edycję
  + **Definicje wyposażenia** – zarządzanie definicjami wyposażenia
  + **Klienci** – dostęp do listy wszystkich klientów, czyli kont w roli *Członka*
* **Opcje właściciela** – widoczna tylko dla konta *Właściciel*. Dostęp do wszystkich komponentów z tej rozwijanej listy dostępny jest też tylko dla tego typu użytkownika. Dostępne opcje to
  + **Dodaj użytkownika** – formularz umożliwiający dodanie nowego typu użytkownika, podobny do formularza rejestracji ale z dodatkową opcją wyboru typu konta
  + **Właściciele** – dostęp do listy wszystkich osób w roli właściciela
  + **Klienci** – dostęp do listy wszystkich klientów, czyli kont w roli *Członka*
  + **Pracownicy** – dostęp do listy wszystkich pracowników, czyli kont w roli *Pracownika*
* **Profil** – ekran zawierający dla wszystkich typów kont dane osobowe i dane o koncie. Dodatkowo dla kont w roli *Członka* zawiera dodatkowo listę zajęć na które zapisany był dany użytkownik oraz wszystkie odbyte wizyty na siłowni.
* **Wyloguj** – przycisk umożliwiający wylogowanie z serwisu, przenoszący na stronę startową.

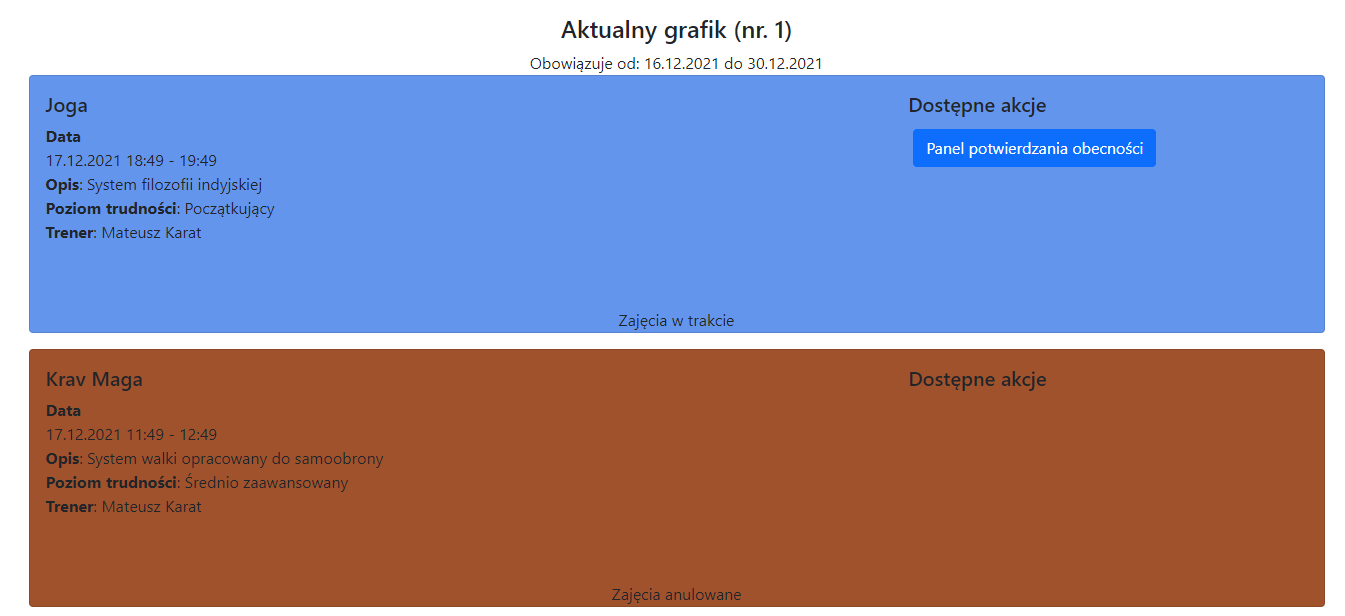
Poniżej przedstawione zostaną niektóre z wymienionych powyżej ekranów. Cały system zaprojektowany został z myślą o spójności, niewidoczne tutaj części zachowują taki sam styl graficzny oraz funkcjonalności. Widoczne ekrany to:

* Widok listy siłowni (Rysunek 4.5)
* Widok ekranu grafiku (Rysunek 4.6)
* Widok dodawania definicji aktywności (Rysunek 4.7)
* Widok dodawania definicji wyposażanie (Rysunek 4.8)
* Widok listy definicji aktywności (Rysunek 4.9)

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 4.5 Ekran widoku listy siłowni



Rysunek 4.6 Ekran widoku grafiku

Obraz zawierający tekst

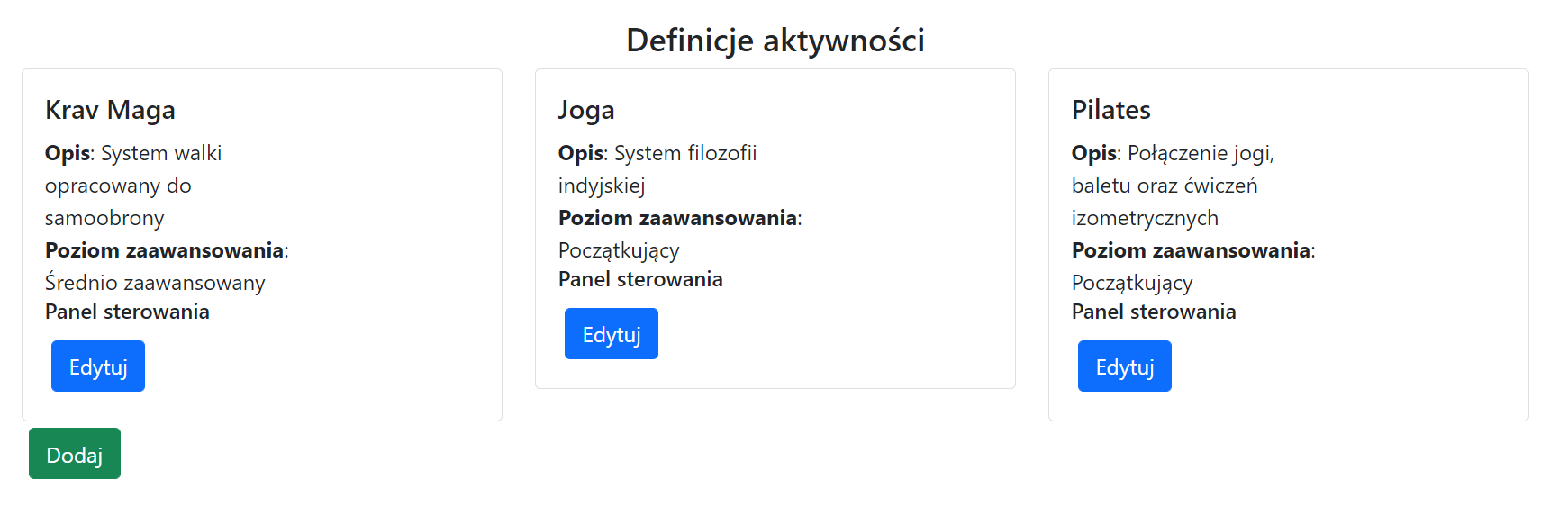
Opis wygenerowany automatycznie

Rysunek 4.7 Ekran dodawania definicji aktywności

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 4.8 Ekran dodawania definicji wyposażenia



Rysunek 4.9 Ekran listy definicji aktywności

## Administracja i sposób działania

W celu przedstawienia sposobu działania systemu w następnej części przedstawiona zostanie operacja dodawania nowych zajęć, zapisanie się na nie przez użytkownika oraz następnie ich automatyczne rozpoczęcie oraz potwierdzenie jego obecności. Proces wymagał będzie używania dwóch kont w rolach *Pracownik* i *Członek.* Dla uproszczenia procesu wymagane jest wykonanie kilku wcześniejszych kroków:

* Dostępna chociaż jedna definicja aktywności
* Dostępny chociaż jeden trener
* Dostępne aktywne konto użytkownika
* Dostępne aktywne konto pracownika
* Dostępna siłownia z aktywnym grafikiem

Pierwszą operacją będzie z poziomu konta *Pracownika* dodanie nowej aktywności do grafiku zajęć, widoczną na Rysunku 4.10.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 4.0.10 Ekran dodawania aktywności

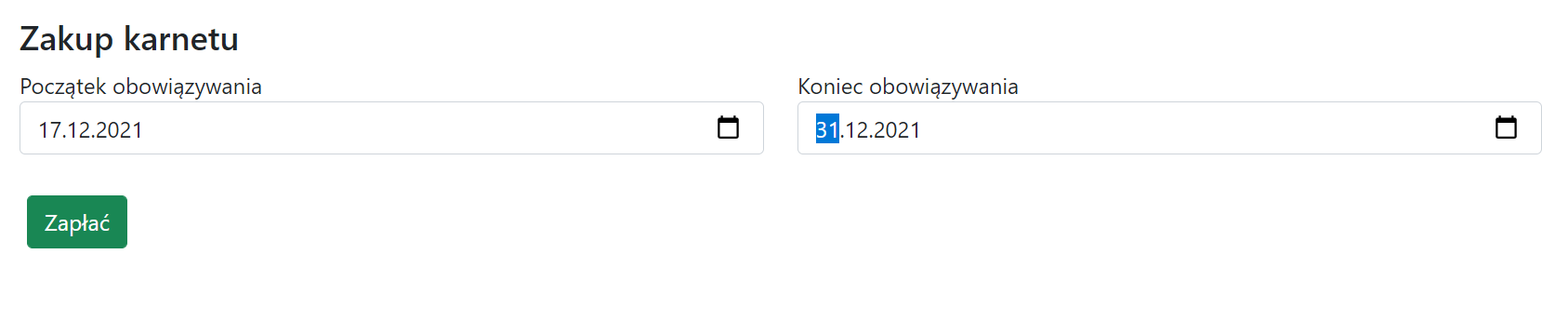
Zajęcia muszą być planowane z co najmniej godzinnym wyprzedzeniem, użytkownik ma więc teraz dość czasu by zapisać się na zajęcia. Aby kontynuować proces należy z konta użytkownika przez ekran siłowni przejść do grafiku zajęć oraz zapisać się na wybrane zajęcia. Ekran grafiku widoczny jest na Rysunku 4.11

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 4.11 Widok dodanych zajęć

Użytkownik po prawej stronie pod napisem *Dostępne akcje* nie widzi jednak żadnych przycisków, ponieważ nie zakupił on jeszcze karnetu. Bez posiadania aktywnego karnetu nie jest możliwe zapisanie się na zajęcia. Zakup karnetu odbywa się w zakładce *Profil*, co widoczne jest na Rysunku 4.12.



Rysunek 4.12 Ekran zakupu karnetu

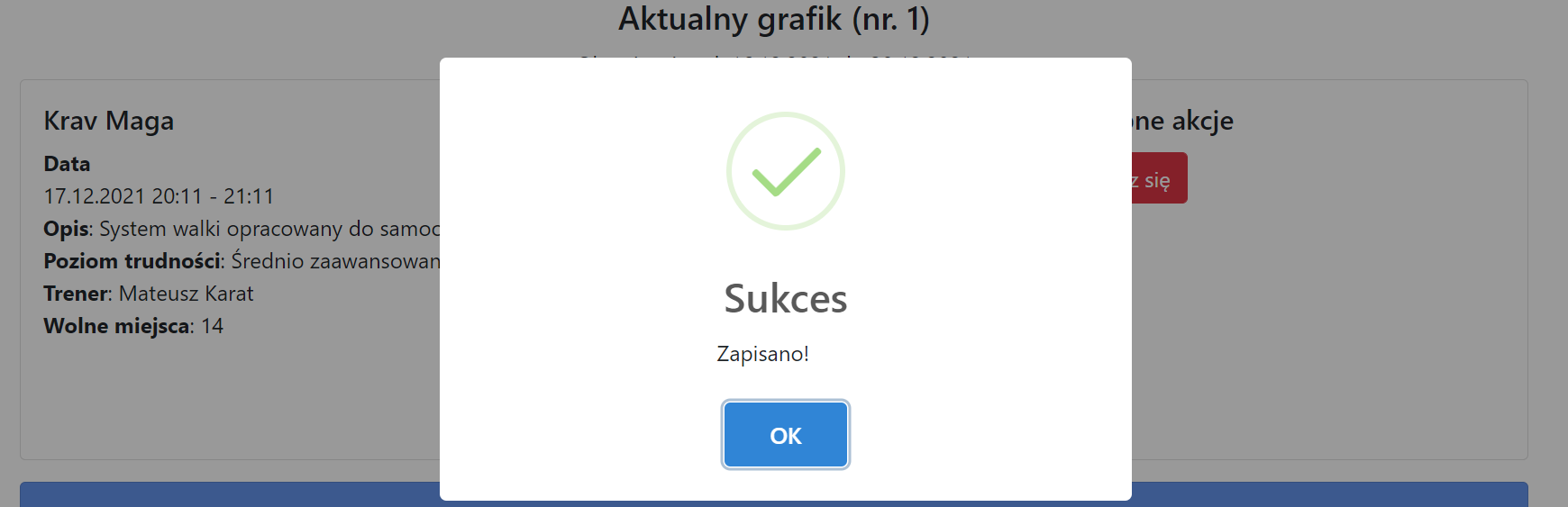
Płatności nie są zaimplementowane w aktualnej iteracji utworzonego systemu, za proces zakupu wystarczy więc wciśnięcie przycisku i podanie odpowiednich dat obowiązywania.

Obraz zawierający tekst

Opis wygenerowany automatycznie

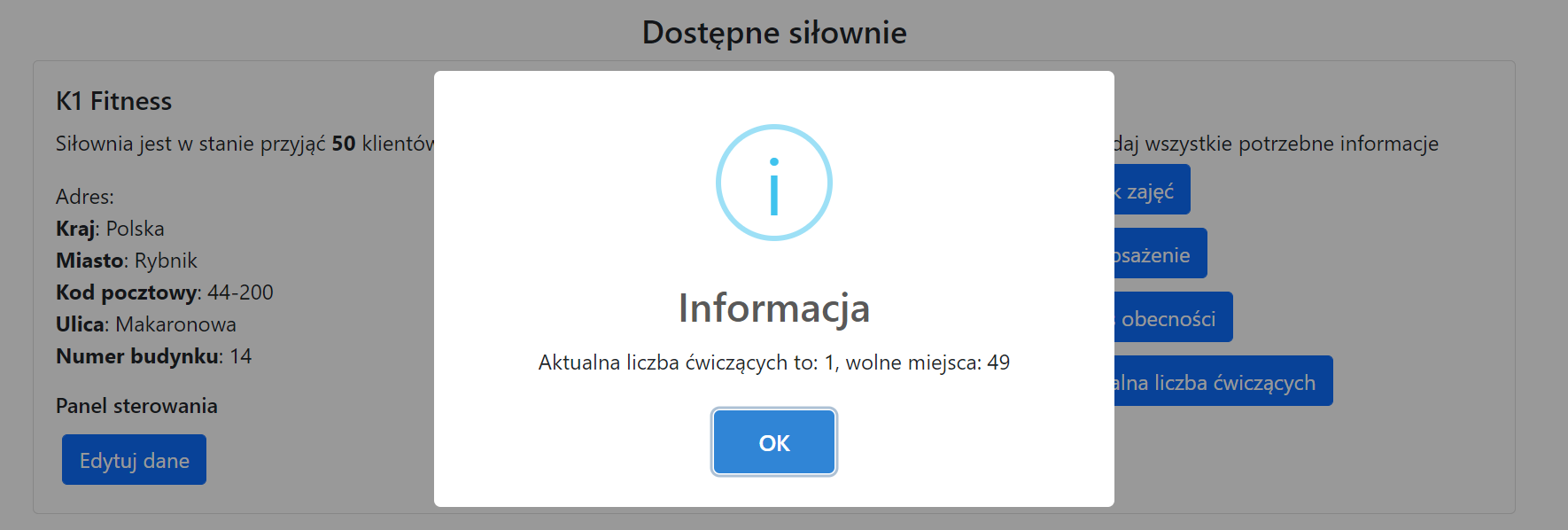
Rysunek 4.13 Widok aktywności z dostępna opcją zapisania się

Po zakupie karnetu pojawiła się możliwość zapisania się użytkownika na utworzone wcześniej zajęcia, co widoczne jest na Rysunku 4.13.



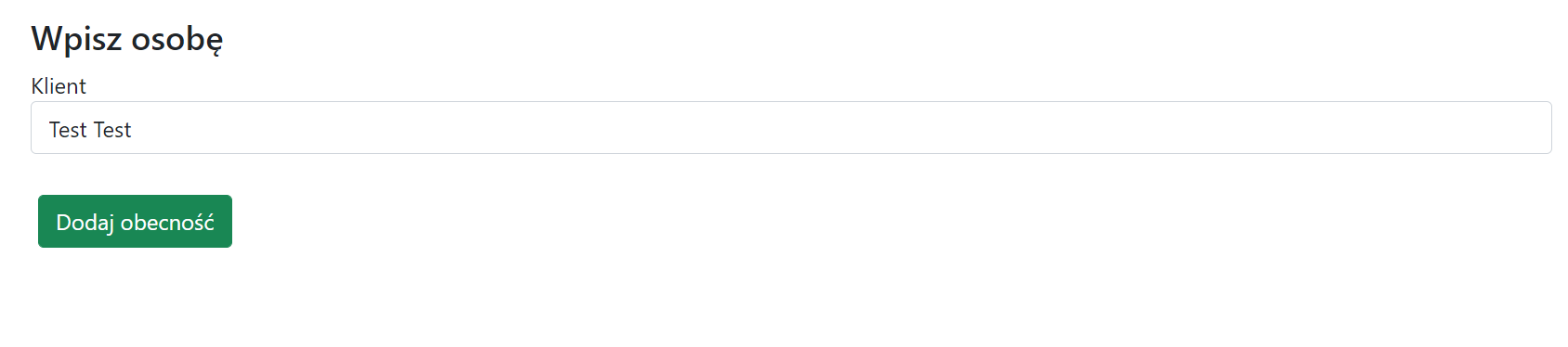
Rysunek 4.14 Udany zapis na zajęcia

Po wykonaniu akcji *Zapisz się* użytkownik otrzymuje informację o pomyślnym dopisaniu się na do listy uczestniczących w zajęciach, co widoczne jest na Rysunku 4.14. Następnie w celu dalszego procesowania zakładamy upływ czasu aż do startu zajęć i przybycie klienta na siłownię. Przed wyjściem z domu klient może również sprawdzić aktualną ilość osób ćwiczących na siłowni, co umożliwia opcja *Aktualna liczba ćwiczących*, zaprezentowana na Rysunku 4.15.

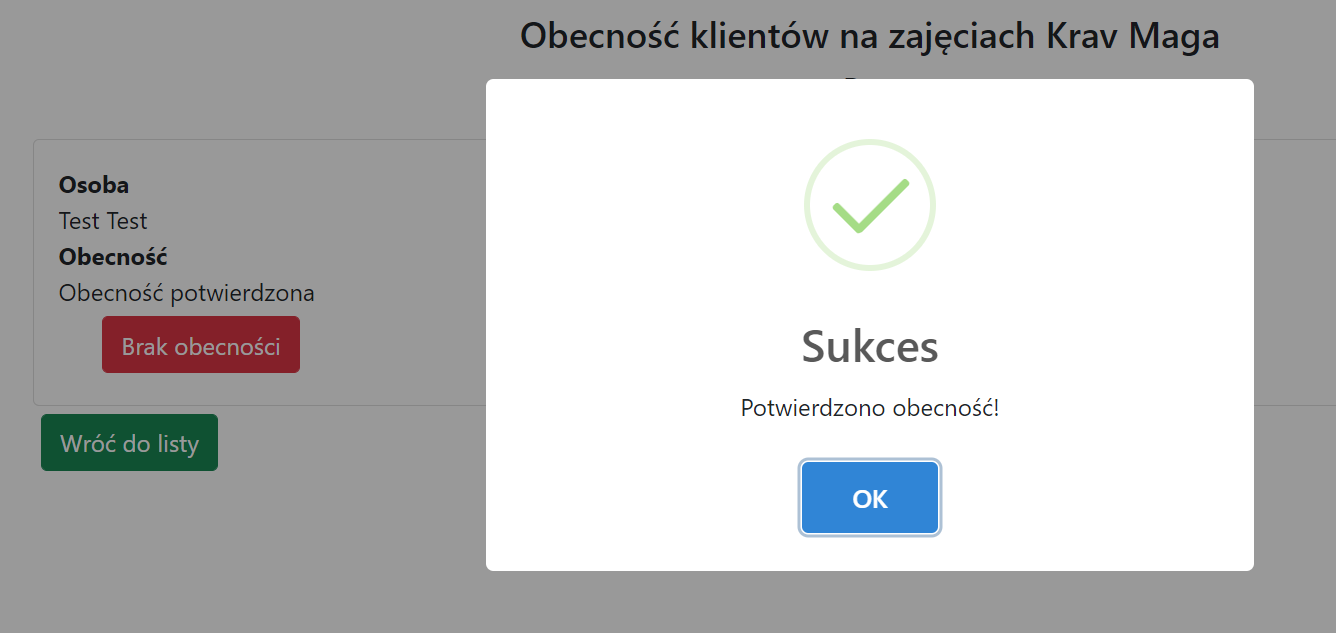


Rysunek 4.15 Ekran informujący o aktualnym obłożeniu siłowni

Następne kroki wykonywane są z poziomu konta pracownika. Wpuszczając klienta na siłownię najpierw wpisuje go na listę obecności (Rysunek 4.16) i potwierdza jednocześnie jego obecność na zajęciach na które się zapisał (Rysunek 4.17). Przy wyjściu klienta z siłowni kończy on również jego aktualny pobyt.



Rysunek 4.16 Ekran zapisania wizyty klienta na siłowni



Rysunek 4.17 Ekran potwierdzania obecności na zajęciach

Klient po zalogowaniu się na swoje konto może sprawdzić wszystkie aktywności i wizyty na siłowni poprzez zakładkę profilu (Rysunek 4.18).

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 4.18 Ekran profilu użytkownika

Powyżej przedstawiony proces jest tylko jednym z wielu możliwych do wykonania w systemie, pokazuje on jednak sposób działania systemu oraz zazębianie się akcji wymaganych przez różnych użytkowników. Dzięki takiej konstrukcji systemu osoby zarządzające nim korzystają z tych samych narzędzi co klienci, zmniejszając tym samym możliwość wystąpienia błędów wynikających z potrzeby synchronizacji wielu systemów. Odpowiednio ustalone granice i zabezpieczenia czynią aplikację łatwą w obsłudze a jednocześnie posiadającą kompleksowe funkcjonalności w pełni umożliwiające sterowanie siecią siłowni.

# Rozdział 5 Specyfikacja wewnętrzna

W rozdziale przedstawiono specyfikację wewnętrzną aplikacji. Zawiera on wszystkie informacje o architekturze, wykorzystanych strukturach danych czy organizacji bazy danych. Znajdują się tutaj również informacje o sposobie działania systemu, wykorzystanych algorytmach, modułach jak i poszczególnych komponentach. Do przedstawienia informacji wykorzystywane są zarówno diagramy UML jak i wycinki kodu.

## 5.1 Funkcjonalność systemu

Głównym celem aplikacji było stworzenie systemu zarządzania siecią siłowni spełniającego potrzeby zarówno użytkownika końcowego, czyli w tym przypadku klientów siłowni, jak również osób zarządzających danymi siłowniami. Projektowanie takiego typu systemu niesie za sobą zarówno ograniczenia jak i uproszczenia. W każdym momencie tworzenia aplikacji należy kłaść duży nacisk na odpowiednie podziały i zabezpieczenia, umożliwiające odpowiednie wydzielanie logiki odpowiedniej dla poziomu uprawnień użytkownika. Każde naruszenie posiadanych uprawnień mogłoby prowadzić do nieprzewidzianych sytuacji i w konsekwencji błędów systemu. Należy również pamiętać o kwestii bezpieczeństwa danych osobowych, nie można pozwolić by użytkownik mógł przeglądać dane do których nie jest uprawniony.

Powstała architektura wpisuje się w wzorzec projektowy MVC, czyli Model-Widok-Kontroler (ang. *Model-View-Controller)* wykorzystywanego w aplikacjach z interfejsami graficznymi. Wzorzec ten zakłada podział aplikacji na trzy główne części, czyli:

* **Model** – reprezentujący logikę aplikacji
* **Widok** – reprezentujący interfejs użytkownika
* **Kontroler** – przyjmujący dane użytkownika, reagujący na jego polecenia i aktualizujący model oraz widok.

Dodatkowo wykorzystane zostały niektóre elementy z domenowego podejścia do programowania DDD. Dzięki temu podział aplikacji nie odpowiada jednoznacznie strukturze proponowanej przez MVC a rozwija go i umożliwia lepsze przedstawienie wycinka rzeczywistości w kodzie. Z modelu domenowego wdrożone zostały m.in. agregaty, język zunifikowany, obiekty wartości czy asercje, opisane szczegółowo w dalszych częściach rozdziału.

Powstały system składa się z dwóch niezależnych lecz wysoko powiązanych ze sobą części. Wyszczególnić możemy więc warstwę prezentacji (ang*. frontend)* oraz warstwę serwera (ang. *backend)*.

## 5.2 Warstwa prezentacji

Warstwa prezentacji zawiera całą logikę danych wyświetlanych użytkownikowi, zajmuje się również wstępną walidacją danych przez niego przekazywanych (wszystkie pola formularzy zabezpieczono odpowiednimi walidacjami). Dane aplikacja zdobywa poprzez komunikację HTTP z warstwa serwera, trzymając się zasad REST. Ważniejsze technologie i rozwiązania wykorzystane przez tą część aplikacji to:

* **Angular** – otwarty szkielet programistyczny (ang. *framework)* wykorzystywany do tworzenia SPA[[8]](#footnote-8), użyty w wersji 13. Ogromna baza dostępnych rozwiązań, bibliotek i kompatybilnych modułów, wysoka popularność i ciągły rozwój i wsparcie naprowadzają na wybór Angulara jako rozwiązania dla warstwy prezentacji.
* **TypeScript** – język programowania, nadzbiór języka JavaScript[[9]](#footnote-9). Wybór ten podyktowany jest faktem, iż TypeScript jest głównym językiem używanym przy tworzeniu aplikacji z wykorzystaniem szkieletu programistycznego Angular. Umożliwia on w przeciwieństwie do JavaScript typowanie obiektów, co może ułatwiać komunikację z warstwą serwera poprzez jawnie określone zawartości obiektów.
* **Node.js** – wieloplatformowe środowisko uruchomieniowe dla tworzenia aplikacji w języku JavaScript (oraz w tym przypadku jego nadzbioru, TypeScript).
* **Npm** – główny manager pakietów platformy Node.js wykorzystywany do zarządzania modułami i zależnościami. Mnogość bibliotek warstw prezentacji nie sprzyja budowaniu odpowiednich rozwiązań, Npm jednak umożliwia zapanowanie nad ciągle zmieniającym się środowiskiem i jego zależnościami.
* **Bootstrap** – biblioteka CSS, zawierająca gotowe, łatwe w użyciu i dostosowaniu style dla wszystkich elementów warstwy prezentacji, rozpoczynając od pojedynczych elementów jak np. przyciski, kończąc na całych strukturach przedstawianych stron[[10]](#footnote-10). Możliwość korzystania z tak bardzo rozwiniętej biblioteki znacząco przyśpiesza pracę, umożliwiając większe skupienie nad pozostałymi częściami systemu.

Wykorzystanie szkieletu programistycznego Angular umożliwia utworzenie aplikacji składając się z komponentów wielokrotnego użytku, pozwalające budować responsywne i przyjazne użytkownikowi aplikacje webowe. Struktura warstwy prezentacji została przedstawiona na rysunku 5.1

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 5.1 Struktura katalogów warstwy prezentacji

Struktura katalogów została utworzona z myślą o łatwej identyfikacji poszczególnych części systemu. Niektóre elementy są również narzucone przez szkielet programistyczny Angular wykorzystywany do tworzenia tej warstwy, zaliczyć do nich możemy m.in.:

* ***app.component.html*** *-* główny plik HTML całej aplikacji, zawierający pojedynczą linię kodu jako odniesienie do wyświetlanych w niej komponentów
* ***app.module.ts*** *-* plik zawierający informacje o głównym module aplikacji Angularowej, umożliwia importowanie innych modułów i zapewnia ich dostępność w pozostałych komponentach systemu. Aplikacja może posiadać więcej niż jeden moduł, w obecnym stanie systemu wykorzystywany jest tylko ten główny
* ***app.component.ts***-główny, podstawowy komponent aplikacji Angularowej. Tworzony automatycznie przy generowaniu nowego projektu
* ***app.routing-module.ts*** *-* plik zawierający informację o Routingu aplikacji. Zawiera dane o adresach URL i odpowiadających im komponentach na które powinna przekierować aplikacja.

*1 const routes: Routes = [  
2 {  
3 path: '',  
4 component: MainComponent,  
5 canActivate: [AuthGuard],  
6 children: [  
7 {path: '', component: QuoteComponent},  
8 {path: 'gyms', component: GymListComponent},  
9 {path: 'gyms/create', component: GymAddComponent, canActivate: [OwnerEmployeeRoleGuard]}*

*,*

Kod 5.1, Fragment klasy *app-routing-module.ts*

Wycinek kodu (Źrodło 5.1) przedstawia strukturę pliku. W linijce 1 tworzona jest lista obiektów typu Routes, zawierająca mapowanie pomiędzy konkretnymi komponentami a ścieżką dostępu do nich. W linijkach 5 oraz 9 widać również przekazanie do parametru *canActivate*obiektu typu *Guard*, odpowiadającego za zabezpieczenia dostępu do danej części systemu.

* ***package.json*** *-* plik w którym zapisane są metadane projektu, używany przez Node.js oraz npm w celu odpowiedniego zarządzania i instalacji zależności czy uruchamiania aplikacji. W tym pliku znajdują się wersje wszystkich używanych przez aplikację bibliotek

Dalszy podział katalogów warstwy prezentacji wynika z podziału domenowego samego systemu jak i wymaganych przez niego funkcjonalności. Poszczególne pakiety systemu (podane w kolejności alfabetycznej) odpowiadają za:

* ***assortment*** *–* zarządzanie wyposażeniem siłowni
* ***default*** *–* pakiet zawierający główne komponenty systemu nie wchodzące w skład innych wyszczególnionych części. W tym pakiecie znajdują się m.in. zaprojektowane walidacje formularzy, klasy ułatwiające formatowanie tekstu czy komponenty służące za stronę tytułową i pasek nawigacji
* ***gyms*** – pakiet zawierający wszystkie komponenty i serwisy powiązane z wyświetlaniem, edycją i dodawaniem obiektów siłowni.
* ***person***– pakiet zawierający komponenty powiązane z kontami użytkowników, profilami, wyświetlaniem danych osobowych
* ***presence***– pakiet zawierający komponenty powiązane z wyświetlaniem aktualnego obłożenia siłowni oraz notowania obecności klientów
* ***security***– pakiet techniczny, zawierający formularze rejestracji i logowania jak również wszystkie serwisy potrzebne do zarządzania bezpieczeństwem aplikacji w warstwie prezentacji.
* ***timetable***– pakiet zawierający komponenty powiązane z grafikiem zajęć siłowni

Struktura większości z powyższych pakietów operuje na tych samych zasadach budowy, dlatego przybliżenie jednej z nich (komponenty powiązane z siłownią) powinno dać dobry obraz sposobu działania systemu.

Jak widoczne było na Rysunku 5.1 katalogu *gyms*dzieli się na 4 podkatalogi oraz dodatkowo zawiera jeden plik nazwany *gym.service.ts*. Plik ten jest wykorzystywany w większości komponentów tego pakietu do komunikacji z warstwą serwera.

*01 export class GymService {  
02   
03 constructor(private api: ApiService) {  
04 }  
05   
06 getAll(): Observable<Gym[]> {  
07 return this.api.get('gym/all')  
08 }  
09   
10 get(id: any): Observable<Gym> {  
11 return this.api.get(`gym/${id}`)  
12 }  
13   
14 create(data: CreateGym): Observable<Gym> {  
15 return this.api.post(`gym`, data)  
16 }*

Kod 5.2, Fragment klasy *GymService*

Kod 5.2 zawiera część z metod dostępnych w serwisie. Są to odpowiednio:

* linijka 6 – pobieranie wszystkich dostępnych siłowni
* linijka 10 – pobieranie konkretnej siłowni poprzez parametr ID
* linijka 14 – utworzenie nowej siłowni poprzez przesłanie odpowiednich danych do warstwy serwera.

Serwis ten korzysta z *ApiService* do komunikacji poprzez HTTP z warstwą serwera. Wyniki zwracane są w formie *Observable****,*** czyli obiektów implementujących wzorzec projektowy Obserwatora, umożliwiających asynchroniczną komunikację i oczekiwanie na otrzymanie danych[[11]](#footnote-11). Dzięki temu komponenty korzystające np. z pobierania danych z tego serwisu mogą zostać załadowane i oczekiwać na odpowiedź serwera, nie wymuszając przy tym ekranów ładowania czy przestojów w użytkowaniu aplikacji.

Jak wspomniano już wcześniej, pakiet *gyms* posiada jest 4 podkatalogi. Są to odpowiednio:

* ***gym*** – katalog zawierający komponent wyświetlający pojedynczą siłownię
* ***gym*-add** – katalog zawierający komponent formularza dodającego nową siłownię
* ***gym*-edit** – katalog zawierający komponent formularza edytującego istniejącą już siłownię
* ***gym*-*list*** – katalog zawierający komponent wyświetlający listę siłowni

Powyższa struktura używana jest przez 90% całej warstwy prezentacji, przechodząc przez taki sam schemat katalogów odpowiednio dla definicji wyposażenia, aktywności, obecności czy samych zajęć. Utrzymanie podobnej struktury w większości systemu umożliwiło łatwiejsze skalowanie i rozwój. Raz użyte rozwiązania do walidacji czy pobierania danych do wyświetlania można było następnie dostosowywać do nowych komponentów i dzięki temu oszczędzać czas.

Komponenty mają również podobną strukturę wewnątrz samych plików. Na początku każdy z komponentów wymaga zainicjalizowania używanych przez niego serwisów. Efekt ten uzyskiwany jest za pomocą konstruktora komponentu widocznego w Kodzie 5.3

*1 constructor(  
2 private gymService: GymService,  
3 readonly userService: SecurityUserService,  
4 private toasts: ToastsService  
5 ) {  
6 }*

Kod 5.3, Konstruktor komponentu

W linijkach 2-4 wyszczególnione są przykładowe serwisy używane przez komponent. *GymService*został już wcześniej omówiony, *ToastsService* zapewnia prostą funkcjonalność wyskakujących wiadomości dla użytkownika, oznaczających sukces lub niepowodzenie w przeprowadzanej operacji. *SecurityUserService* odpowiada za odpowiednie ukrywane i zabezpieczanie opcji systemu, co zostanie omówione w dalszej części rozdziału.

Kolejnym istotnym elementem wykorzystywanych komponentów są metody pozwalające pobierać i zapisywać dane korzystając z warstwy serwera.

*01 ngOnInit(): void {  
02 this.retrieveGyms();  
03 }  
04   
05 retrieveGyms(): void {  
06 this.gymService.getAll()  
07 .subscribe({  
08 next: value => {  
09 this.gyms = value  
10 },  
11 error: err => {  
12 this.toasts.showErrorToast(`Błąd przy pobieraniu listy`)  
13 }  
14 })  
15 }*

Kod 5.4, Kod pobierający listę siłowni

Kod z Źródła 5.4 ilustruje operację pobierania pełnej listy dostępnych siłowni. W linijce 1 wykorzystana jest metoda *ngOnInit,* będąca częścią cyklu życia komponentów w frameworku Angular[[12]](#footnote-12). Uruchamiana jest przy inicjalizacji komponentu, dzięki czemu od razu wywołujemy funkcję pobierającą listę siłowni z linijki 5.

Funkcja pobierająca listę siłowni operuje na wspominanym wcześniej obiekcie typu *Observable*, dzięki czemu po zasubskrybowaniu wyczekujemy otrzymania danych i w przypadku otrzymania wartości (linijka 8) przypisujemy odpowiedź do zmiennej komponentu. Tak samo obsłużyć możemy błąd przy pobieraniu (linijka 11) i w takim wypadku wyświetlić monit o błędzie przy próbie pobierania danych.

Podobnie do funkcji pobierającej prezentuje się funkcja zapisująca nowo dodaną siłownię, widoczna w Kodzie 5.5.

*01 saveGym(): void {  
02 this.gymService.create(this.gymCommand)  
03 .subscribe({  
04 next: response => {  
05 this.toasts.showSuccessToast(`Siłownia poprawnie dodana`)  
06 this.router.navigate(['/gyms']);  
07 },  
08 error: err => {  
09 this.toasts.showErrorToast(`Błąd przy dodawaniu siłowni`)  
10 }  
11 })  
12 }*

Kod 5.5, Metoda zapisująca obiekt siłowni

W linijce 2 wywoływana jest metoda serwisu kontaktująca się z warstwą serwera, następnie linijki 4 i 8 definiują efekt którego oczekujemy w zależności od udanego wykonania akcji. W linijce 6 widać także przekierowanie po udanym dodaniu nowego obiektu do komponentu zawierającego listę wszystkich siłowni. Obiekt przesyłany do serwisu *gymCommand* jest powiązany z formularzem dostępnym dla użytkownika, zapisanym w pliku HTML. Przykładowe zdefiniowane pole takiego formularza zaprezentowano w Kodzie 5.6

*01 <div class="col">  
02 <label for="timetableStartDate">Data startu pierwszego grafiku zajęć</label>  
03 <input  
04 type="date"  
05 class="form-control"  
06 id="timetableStartDate"  
07 [(ngModel)]="gymCommand.timetableStartDate"  
08 #timetableStartDate="ngModel"  
09 name="timetableStartDate"  
10 required  
11 futureDate  
12 [ngClass]="{'is-invalid': (timetableStartDate.touched || timetableStartDate.dirty) && !timetableStartDate.valid }"  
13 />  
14 <span class="invalid-feedback"> <br/>  
15 <span \*ngIf="timetableStartDate.hasError('required')" class="alert alert-danger">  
16 Pole wymagane  
17 </span>  
18 <span \*ngIf="timetableStartDate.hasError('futureDate')" class="alert alert-danger">  
19 Data musi być dzisiejsza lub z przyszłości  
20 </span>  
21 </span>  
22 </div>*

Kod 5.6, Przykładowe pole formularza

W linijce 13 rozpoczyna się deklaracja pola do którego użytkownik będzie wpisywał dane. Linijki 10 i 11 (oraz bezpośrednio powiązane z nimi linijki 15 i 18) oznaczają zastosowane walidacje dla tego konkretnego pola. Korzystając z możliwości frameworka Angular (oraz dyrektyw warunkowych typu *ngIf)* możliwe jest dodanie walidacji do każdego pola formularza oraz w przypadku podania błędnych danych zablokowanie procesu i poinformowanie użytkownika odpowiednim komunikatem.

Istotną częścią warstwy prezentacji są również odpowiednie zabezpieczenia. O ile na warstwie serwera spoczywa odpowiedzialność za ostateczną walidację uprawnień użytkownika, tak odpowiednie użycie dostępnych narzędzi dodatkowo podnosi poziom bezpieczeństwa systemu.

Utworzona aplikacja korzysta z zabezpieczeń w formie tokenu JWT. Jest to obecnie jedno z najpopularniejszych rozwiązań w zabezpieczeniach aplikacji webowych. Każda akcja wywoływana w warstwie serwera musi autoryzować się poprzez przekazanie w nagłówku zapytania HTTP właśnie tego tokena JWT. Sam mechanizm generowania i przydzielania tokena omówiony zostanie w warstwie serwera, warto natomiast przyjrzeć się implementacji wykorzystanej po stronie frontendu.

Każdorazowo dostęp do aplikacji poprzedza formularz logowania. Po podaniu danych użytkownika (nazwa, hasło) warstwa serwera sprawdza czy podane dane zgadzają się z kontem użytkownika zapisanym w bazie danych, jeżeli tak to zostaje wygenerowany i zwrócony do użytkownika token JWT. Token ten następnie przechowywany jest w pamięci *localStorage* przeglądarki, umożliwiającej przechowywanie informacji w postaci struktury danych mapy. W momencie wylogowania lub próby wykonania akcji niedozwolonej dla danego użytkownika, token zostanie usunięty i użytkownik będzie zmuszony do ponownego zalogowania się. Token ten jest następnie przekazywany przy każdym zapytaniu do warstwy serwera jako nagłówek zapytania http.

Poza przechowywaniem w pamięci tokena, pobierane i zapisywane są dane użytkownika, zawierające między innymi rolę jaką posiada. Posiadana rola ma kluczowe znaczenie w ukrywaniu odpowiednich części systemu. Dzięki temu w aplikacji można korzystać z metod takich jak widoczna w Kodzie 5.7 metoda sprawdzająca rolę aktualnie zalogowane użytkownika.

*1 public isOwner(): boolean {  
2 return this.getUser().role === Role.OWNER  
3 }*

Kod 5.7, Metoda sprawdzająca rolę użytkownika

Użycie powyższej metody w połączeniu z dyrektywą *ngIf (*Kod 5.8) umożliwiają ukrywanie konkretnych części systemu, jak np. edycji wyposażenia siłowni

*1 <div class="container" \*ngIf="userService.isOwnerOrEmployee()">  
2 <h6 class="card-title">Panel sterowania</h6>  
3 <a class="btn btn-primary" routerLink="/assortment/equipment/{{ gymId }}/{{ equipment.id }}">Edytuj</a>  
4 </div>*

Kod 5.8, Użycie metody sprawdzającej rolę w komponencie

Kolejnymi istotnymi elementami korzystającymi z tokena zapisanego w pamięci są wspomniane już wcześniej *Guard’y.* Na podstawie roli zapisanego użytkownika definiują one dostęp do całych ekranów systemu.

*01 public canActivate(  
02 route: ActivatedRouteSnapshot,  
03 state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {  
04   
05 if (this.securityUserService.isOwner()) {  
06 return true;  
07 }  
08   
09 this.jwtService.clearToken()  
10 this.securityUserService.clearUser()  
11 this.router.navigate(['/login']);  
12 return false;  
13 }*

Kod 5.9, AuthGuard dla roli Właściciela

Guard utworzony dla roli OWNER widoczny jest w Kodzie 5.9. W linijce 5 widoczny jest warunek sprawdzający czy dana osoba posiada rolę typu OWNER. Jeżeli tak, zwracana jest wartość *true* umożliwiająca dostęp do strony. W przeciwnym wypadku od linijki 9 kolejno usunięte zostają z pamięci dane tokena i użytkownika oraz następuje przekierowanie do strony logowania.

## 5.3 Warstwa serwera

Warstwa serwera ma za zadanie obsługę zapytań HTTP wysyłanych przez warstwę prezentacji, przetwarzanie tych danych i ostatecznie zapis do bazy danych. Cała logika aplikacyjna również znajduje się w tej warstwie. Istotne technologie użyte w budowie tej warstwy to:

* **Java** – obiektowy język programowania, kompilowany do kodu bajtowego wykonywanego przez maszynę wirtualną. Wybór Javy jako języka warstwy serwera podyktowany jest jego wysoką popularnością, stabilnością i ciągłym wsparciem twórców. Tworzone systemy i aplikacje są skalowalne i proste w rozwoju oraz utrzymaniu. Szeroki dostęp do dodatkowych bibliotek, frameworków i dokumentacji cementuje wybór Javy. W utworzonym systemie używana w wersji 11.
* **Spring Boot** – Spring Framework jest najpopularniejszym szkieletem programistycznym języka Java, gdzie Spring Boot jest narzędziem rozwijającym go. Umożliwia on szybkie utworzenie aplikacji gotowej do uruchomienia już od momentu wygenerowania początkowego kodu. Spring Boot zapewnia narzędzia automatycznej konfiguracji systemu, nadpisywanej przez użytkownika tylko w przypadku indywidualnej potrzeby[[13]](#footnote-13). Dodatkowo, gotowe implementacje i pakiety zależności (nazywane *starterami*) przyśpieszają pracę nad aplikacją i implementację nowych funkcji. W systemie używany w wersji 2.5.5
* **Hibernate** – Framework będący jedną z najpopularniejszych implementacji JPA w Javie. Służy do mapowania obiektowo-relacyjnego, czyli w uproszczeniu translację danych pomiędzy relacyjną bazą danych a obiektami aplikacji[[14]](#footnote-14).
* **Maven** – Narzędzie automatyzujące budowę oprogramowania opartego o platformę Java[[15]](#footnote-15). Pozwala na zarządzanie zależnościami, kompilacją, budowaniem czy tworzeniem raportów na temat aplikacji. Działanie maven określone jest przez główny plik konfiguracyjny POM.

Dzięki wykorzystaniu popularnych, dobrze udokumentowanych rozwiązań rozwój warstwy serwera systemu przebiegał pomyślnie od samego startu prac. Wykorzystanie Spring Boot umożliwiło błyskawiczne uruchomienie pierwszej wersji aplikacji, którą następnie rozwijano inkrementacyjnie. Warstwa prezentacji została utworzona w terminie późniejszym, kiedy wszystkie rozwiązania warstwy serwera były już utworzone.

Struktura pakietów warstwy serwera zaprezentowana została na Rysunku 5.2.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rys. 5.2 Struktura pakietów warstwy serwera

Struktura pakietów warstwy serwera dzieli aplikację na kilka części współpracujących ze sobą, wydzielające jednak konkretne części systemu z ich obowiązkami i potrzebami. Poszczególne pakiety posiadają następujące odpowiedzialności:

* ***adapters/rest/resource*** *–* pakiet zawierający klasy kontrolerów, odpowiadające za odbieranie komunikatów HTTP i delegowanie ich do konkretnych implementacji serwisów odpowiadający za daną część systemu
* ***application***– warstwa aplikacyjna, zawierająca wszystkie serwisy, obiekty DTO przesyłane do warstwy prezentacji oraz klasy mapujące encje do tych obiektów. Serwisy działają na zasadzie wzorca programowania fasady, zbierając wszystkie potrzebne informacje do wykonania danej akcji a następnie delegowanie czynności do warstwy domenowej. Na tym etapie następuje także sprawdzanie uprawnień użytkownika
* ***config***– pakiet zawierający wszystkie klasy konfiguracyjne systemu, w tym konfigurację zabezpieczeń
* ***domain*** *–* pakiet zawierający warstwę domenową aplikacji. Znajdują się tutaj wszystkie encje systemu oraz repozytoria umożliwiające zapis i odczytanie ich z bazy danych. To w tym pakiecie wykonywane są również wszystkie wymagane walidacje.
* ***infrastructure*** *–* pakiet zawierający klasy pomocnicze i techniczne

Najlepszy wgląd w architekturę warstwy serwera osiągnąć można poprzez prześledzenie całości procesu zapytania wysłanego przez warstwę prezentacji aż do ostatecznego zapisania w bazie danych.

Proces potwierdzania obecności klienta na zajęciach dostępny jest tylko dla właścicieli i pracowników siłowni. Rozpoczyna się od metody klasy kontrolera (Kod 5.10)

*1 @PostMapping("/{gymId}/{timetableId}/activity/{activityId}/confirm/{userId}")  
2 public ActivityDto confirmAttendance(@PathVariable Long gymId,  
3 @PathVariable Long timetableId,  
4 @PathVariable Long activityId,  
5 @PathVariable Long userId) {  
6 return timetableService.confirmAttendance(gymId, timetableId, activityId, userId);  
7 }*

Kod 5.10 Metoda klasy kontrolera

W linijce 1 widoczna jest ścieżka URL wykorzystywanego do wywołania danej akcji. Wszystkie parametry zmienne w takiej ścieżce oznaczone są poprzez zamknięcie w klamrach {}, następnie w linijkach 2-5 dowiązywane są one do parametrów metody. Adnotacja *@PostMapping* oznacza, że dana metoda obsłuży właśnie zapytanie POST wysłane pod ten adres.

W linijce 6 widoczne jest wywołanie metody z klasy *TimetableService*, zaprezentowaną w Kodzie 5.11.

*01 @Transactional  
02 @PreAuthorize("hasAnyRole('ROLE\_OWNER','ROLE\_EMPLOYEE')")  
03 public ActivityDto confirmAttendance(Long gymId, Long timetableId, Long activityId, Long userId) {  
04 Member member = personService.memberById(userId);  
05   
06 Timetable timetable = getTimetableById(gymId, timetableId);  
07   
08 Activity activity = timetable.activityById(activityId);  
09   
10 ConfirmAttendance confirmAttendance = timetableCommandMapper.confirmAttendance(member);  
11 activity.confirmAttendance(confirmAttendance);  
12   
13 timetable = timetableRepository.save(timetable);  
14 return timetableMapper.activity(timetable.activityById(activityId));  
15 }*

Kod 5.11, Metoda klasy serwisu

Ważniejsze kroki tej metody można zawrzeć w następujących punktach.

* linijka 1 – adnotacja *Transactional* gwarantuje, że cały blok metody wykonany zostanie w transakcji
* linijka 2 – adnotacja *@PreAuthorize* sprawdza uprawnienia użytkownika i dopuszcza do wykonania tylko role OWNER i EMPLOYEE.
* linijka 4, 6 i 8 – pobieranie odpowiednich danych do wykonania akcji, korzystając z parametrów przekazanych w zapytaniu POST
* linijka 10 – utworzenie obiektu który przekazany zostanie do warstwy domenowej systemu
* linijka 11 – wywołanie akcji potwierdzenia obecności danego użytkownika, dokładnie omówiona w kolejnym akapicie
* linijka 13 – zapis danych agregatu *Timetable* do bazy danych.
* linijka 14 – zwrócenie danych przemapowanych na obiekt DTO do warstwy kontrolera, skąd następnie przekazane zostaną do warstwy prezentacji

Na powyższym przykładzie widoczne są wszystkie obowiązki warstwy aplikacyjnej systemu. Serwisy na podstawie danych przekazanych przez kontrolery agregują odpowiednie obiekty, tworzą obiekty polecenia przekazywane do warstwy domenowej, zajmują się również zapisem, propagacją i zwróceniem danych w przystępnej formie dla warstwy prezentacji.

Warstwa domenowa jest najniżej położoną częścią systemu. W warstwie domenowej zawierają się encje będące w tym wypadku także agregatami danych, czyli obiektami będącymi zbiorem skojarzonych ze sobą obiektów w którym wszelkie zmiany czy całkowite usunięcie agregatu będzie miało także wpływ na podległe mu obiekty[[16]](#footnote-16).

Encje zbudowane są z relacji z innymi encjami, prostych pól oraz wartości (ang. *value object)*. Wartości to obiekty opisujące fragment domeny bez tożsamości, czyli chodzi tutaj o elementy mające dla nas znaczenie oparte tylko o to jakie są, a nie kim lub czym są[[17]](#footnote-17). Ostatnim elementem wartym wyszczególnienia w warstwie domenowej są asercje. Asercje umożliwiają ujawnienia, a następnie obsługę wszelkich nieprawidłowości danych przekazanych do encji[[18]](#footnote-18). Użycie wszystkich tych elementów gwarantuje utworzenie spójnej i odpornej na błędy domeny, a w konsekwencji całego systemu.

Przykłademobiektu wartości z systemu może być pole zawierające kod pocztowy.

*1 @Embedded  
2 @AttributeOverride(name = "value", column = @Column(name = "POSTAL\_CODE"))  
3 private PostalCode postalCode;*

Kod 5.12, Użycie obiektu wartości

Powyższy kod (Kod 5.12) zawarty jest w klasie *Address*, gdzie zamiast użycia obiekty typu String dane o kodzie pocztowym zapisywane są w klasie *PostalCode*. Adnotacje w linijkach 1 oraz 2 są częścią JPA i umożliwiają odpowiedni zapis i odczytanie do bazy danych.

*1 private static final String POSTAL\_CODE\_REGEX = "\\d{2}-\\d{3}";  
2   
3 private void initValue(String value) {  
4 DomainAsserts.assertArgumentNotNull(value, "Postal code is null");  
5 DomainAsserts.assertArgumentMatchesPattern(POSTAL\_CODE\_REGEX, value, "Postal code is malmorfed");  
6 this.value = value;  
7 }*

Kod 5.13, Metoda inicjalizująca obiekt wartości

W samej klasie *PostalCode* istotną metodą jest *initValue* (Kod 5.13). Jest to jedyna możliwość utworzenia obiektu tego typu. Kody pocztowe w Polsce mają wartości typu 43-523, więc w linijce 1 zapisane zostało wyrażenie regularne (ang. *regular expression)* sprawdzające dokładnie taki wzór danych. Następnie linijki 4 i 5 sprawdzają wartość przekazaną do konstruktora obiektu *PostalCode* i w przypadku niezgodności z wymaganą formą przerywają proces poprzez zatrzymanie aplikacji wyjątkiem (ang. *exception)*.

Wracając do przykładu potwierdzania obecności klienta na zajęciach kolejnym korkiem jest proces wywoływanej wewnątrz domeny (Kod 5.14)

*1 public void confirmAttendance(ConfirmAttendance command) {  
2 DomainAsserts.assertArgumentNotNull(command, "command cannot be null");  
3 DomainAsserts.assertArgumentNotNull(command.getMember(), "member cannot be null");  
4   
5 Attendance attendance = attendanceByMember(command.getMember());  
6 attendance.confirmAttendance();  
7 }*

Kod 5.14, Metoda domenowa potwierdzania obecności obiektu Activity

Linijki 2 oraz 3 to wspomniane wyżej asercje. Upewniają się one, że dane przekazane do warstwy domenowej są poprawne i dalsze procesowanie jest możliwe. W przypadku niespełnienia danych warunków operacja została by przerwana a użytkownik poinformowany o określonym błędzie.

Linijka 5 umożliwia wyszukanie odpowiedniego obiektu Obecności (ang. *Attendance)* z listy, a następnie wywołanie metody w linijce 6. Metoda widoczna jest w Kodzie 5.15.

*1 public void confirmAttendance() {  
2 DomainAsserts.assertState(this.attended == null || !this.attended, "wrong attendance parameter");  
3 this.attended = true;  
4 }*

Kod 5.15, Metoda domenowa potwierdzania obecności obiektu Attendance

W linijce 2 po raz kolejny widoczna jest asercja, w tym wypadku upewniająca się, że nie próbujemy ponownie zmienić parametru obecności jeżeli została już ona potwierdzona.

Dodatkowym elementem wartym omówienia w warstwie serwera będą odpowiednie zabezpieczenia. W wcześniejszym przekroju procesu potwierdzania obecności klienta na zajęciach można było zauważyć wykorzystanie metody *@PreAuthorize* wykonującej sprawdzenie posiadania odpowiedniej roli przez użytkownika wykonującego dane zapytanie. Za całość bezpieczeństwa aplikacji odpowiada implementacja oparta na tokenie JWT.

Proces rozpoczyna się tak jak już wcześniej wspomniano w warstwie prezentacji przy próbie zalogowanie się. Wywoływana jest wtedy w serwisie aplikacyjnym odpowiedzialnym za ten proces metoda *authenticate*, widoczna w Kodzie 5.16.

*01 public String authenticate(SignInCommand command) throws Exception {  
02 try {  
03 authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(command.getUsername(), command.getPassword()));  
04 } catch (DisabledException e) {  
05 throw new Exception("User disabled", e);  
06 } catch (BadCredentialsException e) {  
07 throw new Exception("Invalid credentials", e);  
08 }  
09   
10 final UserDetails userDetails = userDetailsService.loadUserByUsername(command.getUsername());  
11 return tokenUtil.generateToken(userDetails);  
12 }*

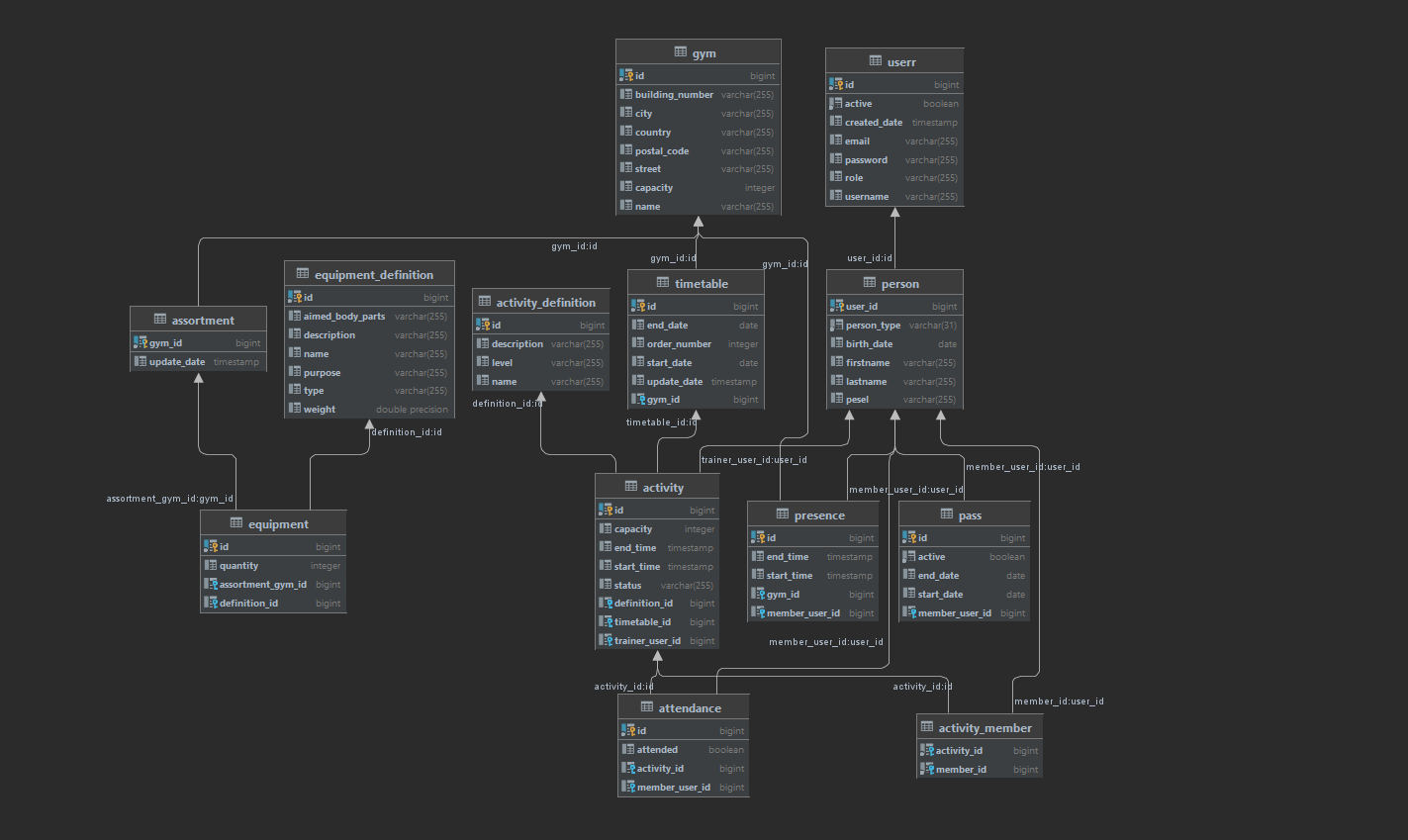
Kod 5.16, Metoda autoryzująca użytkownika w systemie

Do metody przekazywany jest obiekt SignInCommand, zawierający nazwę użytkownika i hasło potrzebne w procesie logowania. Następnie w linijce 3 następuje próba sprawdzenia czy użytkownik z podanym hasłem istnieje w bazie danych. Jeżeli wszystkie sprawdzenia zostają wykonane poprawnie w linijce 10 następuje pobranie szczegółowych danych użytkownika, a linijka 11 zawiera odwołanie do klasy pomocniczej generującej token JWT zwracany w odpowiedzi do warstwy prezentacji. Konfiguracja tej części aplikacji ma miejsce w odpowiedniej klasie konfiguracyjnej SecurityConfig, gdzie wybierane jest m.in. Kod z którego powinny być pobierane dane do autoryzacji, wybranie odpowiedniego szyfrowania haseł czy konfiguracja odpowiednich ścieżek URL systemu które powinny być zabezpieczone.

Warstwa prezentacji wysyła otrzymany token JWT jako nagłówek każdego połączenia HTTP. Jest on następnie przechwytywane przez klasę JwtRequestFilter, która dla każdego zapytania wyszukuje odpowiedni nagłówek, sprawdza poprawność tokena poprzez jego odszyfrowanie a następnie sprawdzenie poprawności danych z tymi dostępnymi w systemie. W przypadku braku tokena lub jego niepoprawnej strukturze zapytanie zostanie odrzucone.

## 5.4 Struktury danych i organizacja bazy danych

Struktura bazy danych (Rysunek 5.3) utworzona została automatycznie z wykorzystaniem narzędzie Hibernate, w oparciu o utworzone encje i relacje pomiędzy nimi. System miał za zadanie odwzorować fragment rzeczywistości powiązany z zarządzaniem siecią siłowni, utworzone encje zatem w większości powiązane są z tym zagadnieniem. Końcowy schemat bazy danych przedstawia się następująco.



Rys. 5.3 Schemat bazy danych

Encje utworzone w systemie, w kolejności od najszerszego agregatu, opisano w ogólnym zakresie w kolejnych podpunktach.

* ***Gym*** *–* Główny agregat systemu, najwyżej położony obiekt w hierarchii systemu. Zawiera informacje o konkretnej siłowni takie jak nazwa, adres czy dozwolona ilość ćwiczących, jak również dowiązania do innych istotnych elementów powiązanych z samą siłownią. Jest to odpowiednio wyposażenie siłowni (*Assortment)*, grafiki zajęć (*Timetable),* oraz listy wizyt trenujących (*Presence).*
* ***Assortment*** *–* Encja powiązana relacją jeden do jednego z siłownią. Zawiera w sobie informację o wyposażeniu siłowni (lista encji *Equipment)*.
* ***Timetable*** *–* Encja zawierająca informacje o grafiku zajęć. Powiązana z siłownią w relacji wiele do jednego. Grafik ma określony czas startu i końca, zawierając w sobie listę zajęć (*Activity*) w danym okresie. Aktualny może być tylko jeden grafik, reszta przechowywana jest jako archiwum
* ***Presence*** *–* Encja zawierająca powiązanie jeden do jednego do klienta (*Member),* oraz siłowni w relacji wiele do jednego. Zawiera informacje o dacie rozpoczęcia i zakończenia treningu użytkownika na danej siłowni.
* ***Equipment*** *–* Encja opisująca wyposażenie dostępne na danej siłowni. Powiązana jeden do wielu z encją opisującą definicję wyposażenia (*Equipment Definition)*, zawiera informację o ilości sztuk danego wyposażenia na siłowni.
* ***Activity*** *–* Encja opisująca zajęcia, posiadająca w sobie odwołanie jeden do jednego z definicją aktywności (*ActivityDefinition)*, trenerem (*Trainer)*, listą zapisanych klientów oraz listą obecności na zajęciach tych klientów (*Attendance).* Dodatkowo przekazuje informację o czasie początku i końca zajęć, ich statusie oraz ilości klientów mogących z nich skorzystać.
* ***EquipmentDefinitoon*** *–* Encja opisująca wyposażenie w szczegółach takich jak nazwa, opis, zastosowanie, typ czy waga
* ***ActivityDefinition*** *–* Encja opisująca szczegół aktywności, takie jak opis czy poziom trudności
* ***Attendance*** *–* Encja opisująca obecność klienta na zajęciach, poprzez powiązania do zajęć, klienta oraz flagę przyjmująca wartość w zależności od faktycznego stanu zajęć
* ***Person*** *–* Encja opisująca dane osobowe klientów, właścicieli, pracowników i trenerów. W strukturze bazy danych zapisana w jednej tabeli w aplikacji dzieli się na obiekty typu:
  + ***Member*** *–* dane klienta, dodatkowo zawierające dane karnetu (*Pass*) oraz aktywności na które uczęszczał (powiązanie wiele do wielu pomiędzy klientem a aktywnością)
  + ***Trainer*** *–* dane trenera, dodatkowo zawierające listę zajęć w których był prowadzącym
  + ***Owner*** *–* dane właściciela
  + ***Employee*** *–* dane pracownika
* ***Pass*** *–* Encja zawierająca dane o karnecie klienta. Karnet wymagany jest do uczestniczenia w zajęciach. Informacje przechowywane przez karnet to data początku i końca obowiązywania karnetu. Posiada powiązanie wiele do jednego z klientem. Tylko jeden karnet może być w danej chwili aktualny
* ***User*** *–* Encja zawierająca dane o koncie użytkownika, parametry takie jak nazwa użytkownika, hasło, email czy rola.

Utworzona struktura encji pozwoliła na odwzorowanie potrzeb użytkowników korzystających z powstałej aplikacji, jednocześnie będąc dostatecznie otwartą na modyfikacje by umożliwić w przyszłości dalszy rozwój systemu. Wykorzystanie podejścia domenowego zagwarantowało odpowiednie i wystarczające zaimplementowanie potrzeb użytkowników aplikacji, a następnie również spójność danych przechowywanych i przetwarzanych przez system.

# Rozdział 6 Weryfikacja i walidacja

W rozdziale przedstawiono sposób weryfikacji i walidacji poprawności działania systemu, opisano zastosowane wzorce, rodzaje oraz strategie testowania oprogramowania. Tekst rozwijają przykłady utworzonych testów jak również problemu których udało się uniknąć dzięki ich zastosowaniu.

## 6.1 Weryfikacja działania systemu

Utworzony i przedstawiony w powyższych rozdziałach system spełnia wszystkie założone funkcjonalności wymagane do zarządzania siecią siłowni. Powstała jednak aplikacja wymaga również sprawdzenia poprawności wykonanych implementacji oraz upewnienia się że wszystkie procesy działają prawidłowo. W celu zdobycia takiej pewności system został sprawdzony korzystając z kilku metod testowania oprogramowania, tak by wyeliminować możliwe jak największą liczbę błędów oraz uodpornić system na wszelkie nieoczekiwane sytuacje.

Testowanie aplikacji inaczej nazwać można jej ewaluacją. W początkowym stadium tworzenia systemu testy (najlepiej wykonywane w sposób zautomatyzowany) gwarantują poprawne spełnianie wybranych założeń. Przy wszelkich modyfikacjach i rozwinięciach już istniejącego systemu wykonywanie testów po wprowadzeniu najmniejszych możliwych zmian informuje programistę o ewentualnych błędach wywoływanych przez nowe funkcjonalności.

Przy tworzeniu systemu wykorzystywana była cześć zasad i pryncypiów wynikających z podejścia sterowanego testami, głównie jednak w samej strukturze i formach utworzonych testów. Pełne zaadoptowanie TDD uniemożliwiał brak zespołu odpowiedzialnego za tworzony system. Przy jednym twórcy, odpowiedzialnym zarówno za analizę rozwiązań, tworzenie założeń i ich późniejszą implementację zmiany wymagań są naturalną konsekwencją procesu twórczego i niwelują korzyści wykorzystania TDD.

Pomimo dużej ilości wykonanych testów należy nadal zaznaczyć że testowanie systemu jest niepełne. Nie wykonane zostały m.in. testy wydajnościowe czy testy bezpieczeństwa, których przeprowadzenie byłoby obowiązek przed wprowadzeniem systemu do użycia przez użytkowników zewnętrznych. Obecna ilość testów systemu gwarantuje jednakże w pełni stabilne działanie podstawowych funkcjonalności systemu oraz umożliwia łatwy dalszy rozwój i utrzymanie.

Do testowania warstwy serwera systemu użyta została biblioteka JUnit, czyli jednego z najpopularniejszych narzędzi do tworzenia testów oprogramowania pisanego w języku Java.

W dalszej części rozdziału opisane zostaną typy utworzonych testów aplikacji, ich sposób implementacji oraz przykładu wyjęte z kodu. Przedstawione zostaną również problemy których udało się uniknąć poprzez wykorzystanie danego typu testowania.

## 6.2 Przypadki testowe

Do testowania aplikacji poprzez testy jednostkowe i integracyjne wykorzystane zostały dane testowe tworzone na potrzeby każdego z testów w indywidualnym zakresie, korzystając z metod dostępnych w aplikacji, tak by w pełni wiarygodnie oddać sposób działania aplikacji. Do testów manualnych wykorzystane zostały dane testowe opisane w Rozdziale 4, czyli automatycznie ładowane do bazy danych podstawowe informacje umożliwiające szybszą weryfikację działania systemu. Do danych testowych wchodziło

* Jeden obiekt siłowni, z przypisanym wyposażeniem
* Trzy definicje aktywności
* Cztery konta użytkowników, po jednym na każdą rolę (poniżej przedstawione dane do logowania w postaci login:hasło
  + Właściciel, (*owner:owner)*
  + Pracownik (*employee:employee)*
  + Trener (*trainer:trainer)*
  + Członek (*member:member)*, z wykupionym karnetem, obecnościami na zajęciach oraz wizytach na siłowni
* Trzy definicje wyposażenia
* Trzy aktywności w różnych statusach, ich daty rozpoczęcia i końca oparte o aktualną datę systemową, tak by można było skutecznie przetestować funkcjonalności niezależnie od momentu uruchomienia systemu

Utworzone dane testowe były wystarczające do manualnego sprawdzenia wszystkich wymagań funkcjonalnych systemu, jednocześnie poprzez ich automatyczne dodawania przy każdym starcie aplikacji przyśpieszyły znacząco proces naprawy błędów.

## 6.3 Testy jednostkowe

Testy jednostkowe wykorzystane zostały do testowania warstwy serwera powstałego systemu, opisywanej szerzej w rozdziale 5.3. Testy jednostkowe są sposobem testowania oprogramowania w którym weryfikowana jest poprawność działania najmniejszych, pojedynczych jednostek systemu, czyli w przypadku systemu opartego o programowanie obiektowe metod lub samych obiektów.

Przykładem wyjętym z systemu będą testy związane z klasą *Address*, odpowiedzialną za przechowywanie informacji o pełnym adresie. Klasa ta zawiera w sobie wcześniej opisywane obiekty wartości, posiadające swoje własne metody tworzące i walidujące obecne w nich dane.

Pierwszym testem będzie samo utworzenie obiektu i sprawdzenie poprawności przypisywania wszystkich danych. Test przedstawiono w Kodzie 6.1

*01 @Test  
02 public void testCreateAddress() {  
03 //given  
04 CreateAddress command = Fixtures.createAddress()  
05 .build();  
06   
07 //when  
08 Address address = Address.create(command);  
09   
10 //then  
11 assertEquals(address.getCountry().getValue(), command.getCountry());  
12 assertEquals(address.getCity().getValue(), command.getCity());  
13 assertEquals(address.getStreet().getValue(), command.getStreet());  
14 assertEquals(address.getPostalCode().getValue(), command.getPostalCode());  
15 assertEquals(address.getBuildingNumber().getValue(), command.getBuildingNumber());  
16 }*

Kod 6.1, Test jednostkowy tworzenia adresu

W linijce 4 widoczne jest utworzenie obiektu *CreateAddress* za pomocą statycznej metody klasy *Fixtures*. Klasa ta zawiera w sobie pomocnicze metody generujące w pełni poprawne i wypełnione obiekty przekazywane do konstruktorów poszczególnych klas w celu ich utworzenia. Obiekty przekazywane są z wykorzystaniem wzorca programistycznego Budowniczy, umożliwiając łatwą zmianę parametrów. W linijce 8 tworzony jest obiekt *Address* z wykorzystaniem wcześniej utworzonego obiektu polecenia, a następnie w linijkach 11-15 asercje sprawdzają poprawne przypisanie poszczególnych wartości do pól obiektu. Komentarze widoczne w linijkach 3,4 oraz 10 są pomocniczą informacją dla programisty określające miejsca testu w którym

* ***given­*** *–* podawane są parametry wejściowe, ustalane środowisko danego testu
* ***when*** *–* wykonywana jest operacja którą ewaluujemy
* ***then*** *–* sprawdzenie wyników i określenie rezultatu testu

Drugim przykładem testu jednostkowego wykorzystywanego w aplikacji będzie symulowana katastrofa, czyli sprawdzenie działania systemu w sytuacji wyjątkowej, której zaistnienie w naturalnych warunek jest bardzo mało prawdopodobne[[19]](#footnote-19). Przykład takiego testu widoczny jest w Kodzie 6.2

*01 @Test  
02 public void testCreateAddressPostalCodeError() {  
03 //given  
04 CreateAddress command = Fixtures.createAddress()  
05 .postalCode("123")  
06 .build();  
07   
08 //when  
09 InvalidStateException exception = assertThrows(InvalidStateException.class, () -> {  
10 Address.create(command);  
11 });  
12   
13 //then  
14 assertEquals("Postal code is malmorfed", exception.getMessage());  
15 }*

Kod 6.2, Test jednostkowy symulowana katastrofa

W przedstawionym powyżej teście w linijce 5 do obiektu tworzącego *Address* przekazany jest kod pocztowy w formie 123. Obiekt wartość użyty w *Address* do przechowywania informacji o kodzie pocztowym oczekuje obiektu w formie xx-xxx, gdzie x to dowolna liczba z zakresu 0-9. W tym wypadku system powinien uniemożliwić utworzenie obiektu i zastosować odpowiedni wyjątek z informacją dla użytkownika. W linijce 9 wykorzystany jest mechanizm do przechwytywania błędów programu, a następnie w linijce 14 sprawdzana jest wiadomość z przechwyconego wyjątku. W tym wypadku wiadomość *Postal code is malmorfed* oznacza błędny format kodu pocztowego, test więc kończy się rezultatem pomyślnym.

Cała warstwa domenowa została przetestowana w dokładnie tak sam sposób jak kod adresu, co umożliwiło wychwycenie i naprawę niskopoziomowych błędów systemu trudnych do wychwycenia z poziomu wyższych warstw. Jednym z naprawionych w ten sposób błędów było błędne ustawianie flagi obecności w procesie potwierdzania przybycia klienta na zajęcia.

*1 ConfirmAttendance confirmAttendance = Fixtures.confirmAttendance()  
2 .member(member)  
3 .build();  
4 //when  
5 activity.confirmAttendance(confirmAttendance);  
6   
7 //then  
8 Attendance attendance = activity.attendanceByMember(member);  
9 assertTrue(attendance.getAttended());*

Kod 6.3, Test jednostkowy metody confirmAttendance

W Kodzie 6.3 przedstawiono test *testConfirmAttendanceParticipant*. W linijce 5 wywoływana jest metoda *confirmAttendance* dla określonego klienta zapisanego na zajęcia. Implementacja metody błędnie przypisywała flagę *attended* wartością *false*, co wynikało z kopiowania kodu przy tworzeniu metod. Asercja w linijce 9 wychwyciła błędną wartość flagi i w skutku pozwoliła na poprawę błędu.

## 6.4 Testy integracyjne

Testy integracyjne wykorzystywane są w celu wykrycia błędów i nieprawidłowości w komunikacji pomiędzy poszczególnymi modułami systemu lub całymi systemami. W utworzonym systemie testy integracyjne wykorzystane zostały do testowania warstwy aplikacyjnej oraz kontrolerów.

W implementacja testów integracyjnych zastosowano testowanie z użyciem obiektów atrap (ang. *mock-objects)*. Obiekt taki jest zastąpieniem rzeczywistego zasobu na spreparowany, zwracający określone wcześniej wartości[[20]](#footnote-20). Atrapy cechują się wysoką wydajnością oraz czytelnością działania. Umożliwiają one testowanie logiki aplikacji bez kosztownych operacji uruchamiania baz danych czy łączenia się z zewnętrznymi serwisami. W połączeniu z uruchamianiem cały kontekstu Spring z wykorzystaniem adnotacji *SpringBootTest* wykonywane testy skutecznie sprawdzały wybrane fragmenty logiki systemu bez konieczności każdorazowego przygotowywania danych dla wszystkich warstw systemu.

Przykładem testu integracyjnego warstwy kontrolera będzie operacja utworzenia siłowni.

*01 @Test  
02 @WithMockUser(roles = "OWNER")  
03 void createGym() throws Exception {  
04 //given  
05 CreateGymCommand command = Fixtures.createGymCommand();  
06   
07 //when  
08 MvcResult result = mockMvc.perform(post("/gym")  
09 .contentType("application/json")  
10 .content(objectMapper.writeValueAsString(command)))  
11 .andExpect(status().isOk())  
12 .andReturn();  
13   
14 //then  
15 String content = result.getResponse().getContentAsString();  
16 GymDto gym = objectMapper.readValue(content, GymDto.class);  
17   
18 assertEquals("Test", gym.getName());  
19 }*

Kod 6.4, Test integracyjny tworzenia siłowni

Test przedstawiono w Kodzie 6.4. W linijce 2 oznaczona jest rola użytkownika z jaką sprawdzamy przypadek testowy. W linijce 8 wykonywane jest połączenie do warstwy serwera pod URL tworzący siłownię, przekazywane są również odpowiednie dane. Odpowiedź serwera mapowana ulega ekstrakcji a następnie zmapowanie na obiekt transferu danych w linijkach 15 i 16, a następnie w linijce 18 następuje sprawdzenie poprawności danych.

Testy kontrolerów wychwyciły przede wszystkim błędne ustalenia autoryzacji dla konkretnych miejsc systemu. Niektóre metody dostępne tylko dla *Właścicieli* czy *Pracowników* dostępne były również dla trenerów i członków, co udało się wyeliminować dzięki błędnym wynikom testów.

Przykładem testu integracyjnego z wykorzystaniem obiektów atrap będzie automatyczna zmiana statusów aktywności przez cykliczne zadania systemu. Omawiany test przedstawiono w Kodzie 6.5

*01 @Test  
02 public void testStartActivityIfShouldStart() {  
03 //given  
04 Activity activity1 = Activity.create(Fixtures.createActivity().build());  
05 activity1.setId(1L);  
06 activity1.setStatus(ActivityStatus.CREATED);  
07 activity1.setStartTime(LocalDateTime.now().minusMinutes(10));  
08 activity1.setEndTime(LocalDateTime.now().plusHours(1));  
09   
10 Activity activity2 = Activity.create(Fixtures.createActivity().build());  
11 activity2.setId(2L);  
12 activity2.setStatus(ActivityStatus.CREATED);  
13 activity2.setStartTime(LocalDateTime.now().plusMinutes(10));  
14 activity2.setEndTime(LocalDateTime.now().plusHours(1));  
15   
16 when(timetableService.activitiesForStatus(any())).thenReturn(List.of(  
17 activity1, activity2  
18 ));  
19   
20 //when  
21 activityJob.changeActivityStatus();  
22   
23 //then  
24 verify(timetableService).startActivity(1L);  
25 }*

Kod 6.5, Test integracyjny zmiany statusu aktywności

Linijki 4-14 tworzą dane testowe wykorzystywane w dalszej części testu. Są to dwie aktywności w statusie *CREATED*, jedna z nich gotowa do uruchomienia przez system. W linijce 16 widoczne jest użycie obiektu atrapy *timetableService* tak by bez wykonywania jego logiki zwrócić utworzone wcześniej aktywności. Następnie linijka 21 to wywołanie automatycznej zmiany statusów, by ostatecznie w linijce 24 sprawdzić czy wykonana została akcja *startActivity* tylko dla jednej, gotowej do rozpoczęcia aktywności.

Jednym z błędów wykrytych przez testy integracyjne była błędna instrukcja wykorzystywana przez wspomniany już wcześniej *activityJob*, czyli klasę uruchamiającą zadanie automatycznej zmiany statusów w aplikacji. Po rozpoczęciu odpowiednich zajęć następnym krokiem zadania jest zakończenie zadań w statusie *IN\_PROGRESS*, o ile czas ich końca przekroczył czas aktualny. Po raz kolejny z powodu kopiowania wcześniej napisanego kodu zamiast metody do zakończenia aktywności wywoływana była ta służąca do ich rozpoczynania. Z tego powodu aktywności trafiały w nieskończoną pętlę i nigdy nie zmieniały statusu na końcowy.

## 6.5 Testy manualne

Poza testami warstwy serwera wykonywanymi automatycznie przy każdym budowaniu aplikacji aplikacja została przetestowana manualnie, poprzez wielokrotne sprawdzanie funkcjonalności systemu. Testy manualne opierały się na przechodzeniu procesów z poziomu warstwy prezentacji z użyciem różnych kont użytkowników we wszystkich dostępnych rolach. Ograniczenia wynikające z testów manualnych wynikają głównie z ich struktury. Zasoby ludzie są podstawowym czynnikiem zwiększającym lub zmniejszającym skuteczność takich testów. Im większa liczba i doświadczenie zespołu testerów tym dokładniejsze sprawdzenie całości aplikacji. Zróżnicowanie testów umożliwia także ocenę przystępności systemu oraz jego działanie w różnych konfiguracjach programowo systemowych.

Testy manualne wykonywane były zarówno w czasie tworzenia aplikacji jak i po wstępnym zakończeniu i spełnieniu oczekiwanych założeń. Wszystkie dodatkowe funkcjonalności były gruntownie sprawdzane pod kątem działania, wyświetlania oraz efektów. Naprawa znalezionych błędów była przeprowadzana natychmiast, co wynikało z odpowiedzialności tej samej osoby za tworzenie i następnie testowanie oprogramowania. Z tego samego powodu nie stosowanego też żadnej formy raportowania takich błędów.

Problemy wychwycone przez tą część testów były zdecydowanie najliczniejsze w całej aplikacji. O ile warstwa serwera zawiera w sobie całą najważniejszą logikę systemu, tak w warstwie prezentacji jest więcej miejsca na błędy i pomyłki. Wielokrotnych poprawek wymagał sposób prezentacji przycisków, tabel czy ekranów. Naprawiane były również walidacje formularzy oraz informowanie o błędach wprowadzanych do nich danych. Istotnym skutkiem testów manualnych było również zaobserwowanie i zabezpieczenie wszystkich błędów wynikających z uprawnień w systemie. Ekrany czy same odnośniki do nich były często widoczne dla użytkowników w rolach wykluczających ich z dostępu do danych części systemu.

# 

# Rozdział 7 Podsumowanie i wnioski

Rozdział zawiera podsumowanie całości prac nad systemem, w tym połączenie otrzymanych wyników względem postawionych celów i założeń. Dodatkowo zawarte są możliwości dalszego rozwoju aplikacji oraz problemy które napotkano przy implementacji, a następnie sposób ich rozwiązania.

## 7.1 Wyniki pracy

System utworzony w ramach pracy inżynierskiej spełniał wszystkie postawione przed nim pierwotnie wymagania. Funkcjonalności zostały zaimplementowane i sprawdzone, tak aby podstawa systemu działała w pełni poprawnie i całość była gotowa na dalsze usprawnienia i rozwój.

Dodatkowa weryfikacja działania systemu nastąpiłaby w momencie oddania go do użytku, wtedy na światło dzienne mogłyby wyjść różnego rodzaju błędy wynikające z jednoczesnego wykorzystania systemu przez wielu użytkowników czy problemy wydajnościowe. W rozdziale o dalszej rozbudowie systemu przedstawione zostały sposoby na zaradzenie ewentualnie powstałym problemom.

Projekt spełnił swoją podstawową określona na początku funkcję bycia systemem do zarządzania siecią siłowni. Określone wymagania okazały się wystarczające do zarządzania obiektem w podstawowej formie, a jednocześnie stanowią solidną podstawę pod dalsze prace.

## 7.2 Dalsza rozbudowa systemu

Utworzony system spełnia wszystkie postawione przy jego projektowaniu wymagania, nie oznacza to jednak, że jest on systemem w pełni ukończonym i zamkniętym na dalsze modyfikacje. Dalsze usprawnienia i dodatkowe funkcjonalności byłyby naturalnym dopełnieniem systemu i polepszały wrażenia z jego wykorzystywania.

Elementem znacząco zwiększającym użyteczność systemu byłoby wprowadzenie funkcjonalności powiadomień. Użytkownicy mogliby wtedy otrzymywać na wybrany przez siebie sposób informacje o zbliżających się zajęciach czy konieczności przedłużenia karnetu. Do implementacji po stronie warstwy serwera mogłoby posłużyć do tego oprogramowanie RabbitMQ, czyli broker wiadomości wykorzystujący protokół AMQP. Prace w tym kierunku wymagałyby również określenia form komunikacji, ewentualnych szablonów dla wiadomości email lub tekstowych.

Aplikacja zyskałaby również na bezpieczeństwie jeżeli zastosowano by zewnętrzny system uwierzytelniania, tak aby hasła i dane wrażliwe użytkowników nie znajdowały się w samej aplikacji a w sprawdzonym, rozbudowanym narzędziu dedykowanym do takich zadań. Jedną z możliwości byłoby oprogramowanie Keycloak, które wykorzystuje system jednokrotnego logowania i następnie umożliwia zabezpieczenie aplikacji za pomocą tokenów JWT, wykorzystywanych już w aktualnym rozwiązaniu. Dzięki takiemu przejściu system nie wymaga gruntownej zmiany, ponieważ mechanizm obsługi tokenów JWT jest już zaimplementowany, a jedynie zwiększa bezpieczeństwo całego systemu.

Wraz z rozwojem aplikacji naturalnym krokiem byłoby również przejście z jednego utworzonego serwisu w kilka spełniających odpowiednie wycinki funkcjonalności. Podejście z wykorzystaniem wielu mikro serwisów w warstwie serwera i jednej warstwy prezentacji jest obecnie najpopularniejszym sposobem implementacji, ze względu na łatwość rozwoju pojedynczych części aplikacji bez ingerowania w całość systemu. Wdrożenia nowych wersji systemu znacząco się wtedy upraszczają.

Wraz z przejściem w architekturę mikro serwisową istotne byłoby wykorzystanie konteneryzacji oraz mechanizmów CI/CD, za pomocą oprogramowania takiego jak Kubernetes czy Jenkins. Jeszcze bardziej usprawniło by to zarządzanie systemem, umożliwiło łatwiejszą integrację z zewnętrznymi usługami czy zautomatyzowało testowanie i wdrożenia.

## 7.3 Problemy i wyzwania

W czasie tworzenia systemu napotkano wiele mniejszych i większych problemów, zarówno implementacyjnych jak i analitycznych. Tworzenie systemów o dużej skali, jednocześnie troszcząc się o potrzeby użytkowników jak i przejrzystość, stabilność i wydajność systemu niesie za sobą wiele wyzwań. W dalszej części przedstawiono kilka z nich.

Pierwszym napotkanym wyzwaniem okazała się analiza istniejących rozwiązań oraz poznanie wymagań klientów i właścicieli obiektów sportowych dla których system powstawał. Większość aktualnie istniejących systemów wymaga przed skorzystaniem z nich podpisania umowy z siecią siłowni wraz z podaniem wszystkich danych osobowych. Takie umowy wymagają również uiszczenia wielu opłat wstępnych, a następnie muszą być wypowiadane na zasadach tradycyjnych umów, z miesięcznym okresem wypowiedzenia. Zdobycie dostępu do narzędzi dostępnych dla pracowników czy właścicieli siłowni okazało się również niemożliwe dla osoby z zewnątrz. Ten problem udało się rozwiązać poprzez osobisty kontakt z sieciami siłowni oraz otrzymanie dzięki ich uprzejmości informacji o potrzebach i co ważniejsze brakujących elementach ich systemów. Potrzeby samych klientów takich siłowni udało się wywnioskować na podstawie rozmów, ankiet oraz analizy najczęściej pojawiających się pytań w kontekście konkretnych placówek.

Wyzwaniem związanym z samym projektowaniem aplikacji okazało się również poprawne określenie wymagań niefunkcjonalnych systemu. Ich poprawne ustalenie jest jednym z najważniejszych części całego procesu tworzenia aplikacji, tak aby można było stworzyć system wydajny oraz stabilny, zdolny do obsłużenia wystarczającej liczby akcji użytkowników. Do określenia poprawnych wartości ponownie wykorzystane zostały dane otrzymane od konkretnych placówek siłowni, takich jak na przykład codzienna liczba klientów. Poza tym wykorzystane została statystyki publicznie dostępne w takich serwisach jak Facebook czy Google, oraz matematyczna analiza liczby zajęć oraz miejsc dostępnych na nich w konkretnych placówkach.

Kolejną napotkaną trudnością była sama warstwa prezentacji systemu, a raczej wygląd ekranów dostępnych w systemie. Tworzenie styli aplikacji od podstaw okazało się bardzo czasochłonne i trudne, a jednocześnie efekty często były niewymierne do ilości poświęconego na nie czasu. Ostatecznie rozwiązaniem okazało się wykorzystanie gotowych, kompletnych pakietów jak na przykład Bootstrap. Dzięki jego wykorzystaniu tworzenie kolejnych ekranów było szybsze i sprawniejsze, pozwalając bardziej skupić się nad samym wrażeniu użytkownika niż detalach implementacyjnych. Określenie spójnej kolorystyki i wykorzystywanych komponentów również pozytywnie wpłynęło na odbiór aplikacji przez użytkowników.

Wyzwaniem które pojawiło się w ostatniej części tworzenia systemu będzie proces weryfikacji poprawności działania systemu. W rozdziale Weryfikacja i walidacja opisane zostały wykorzystywane metody testowania aplikacji, w tym testowanie manualne. To właśnie ta metoda testowania była najbardziej czasochłonną i niewygodną częścią weryfikacji systemu. Przy wielu wychwyconych błędach wymagane było ponowne uruchomienie aplikacji, co poprzez wykorzystywanie bazy danej H2 przechowywanej w pamięci oznaczało usunięcie wszystkich istniejących wpisów. Rozwiązaniem które choć częściowo odciążyło programistę i zmniejszyło ilość pracy było utworzenie skryptów ładujących do bazy danych dane startowe, uruchamiane przy każdym starcie aplikacji. Wykorzystano w nich mechanizmy ustawiania aktualnej daty do wielu wpisów, tak aby na przykład zajęcia rozpoczynały się zawsze 5 minut po starcie aplikacji, co ułatwiało testy.

Dodatkowo, w czasie całej pracy nad systemem napotkano wiele błędów i problemów natury programistycznej. Wykorzystywanie niektórych konstrukcji języka Java takich jak strumienie, czy elementy szkieletu programistycznego Angular takie jak obiekty obserwatorzy okazywało się w wielu miejscach trudne i skomplikowane. Wykorzystanie zintegrowanego środowiska programistycznego pomagało z analizą i wychwytywaniem takich błędów, tak samo jak rozbudowane i kompletne dokumentacje wszystkich wykorzystywanych w projekcie bibliotek.

# Bibliografia

1. Urząd Statystyczny w Rzeszowie. Obiekty służące poprawie kondycji fizycznej w 2020 r, 2021
2. Eric Evans. Domain-Driven-Design. Helion, Gliwice, 2015
3. Kent Beck. TDD. Sztuka tworzenia dobrego kodu. Helion, Gliwice 2020
4. Robert C. Martin. Czysty Agile. Powrót do podstaw. Helion, Gliwice 2020
5. Robert C. Martin. Czysta architektura. Helion, Gliwice 2018
6. Robert C. Martin. Czysty kod. Helion, Gliwice 2014
7. Dokumentacja PostgreSQL, wymagania sprzętowe <https://www.enterprisedb.com/docs/supported-open-source/postgresql/installer/01_requirements_overview/> (dostęp 15.12.2021)
8. Oficjalna dokumentacja Angular <https://angular.io/docs> (dostęp 15.12.2021)
9. Oficjalna dokumentacja TypeScript <https://www.typescriptlang.org/docs/> (dostęp 17.12.2021)
10. Oficjalna dokumentacja biblioteki Bootstrap w wersji 5 <https://getbootstrap.com/docs/5.0/getting-started/introduction/> (dostęp: 19.12.2021)
11. Dokumentacja Spring Boot <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/> dostęp 03.01.2022)
12. Oficjalna strona Hibernate <https://hibernate.org/> (dostęp 03.01.2022)
13. Oficjalna strona Apache Maven <https://maven.apache.org/> (dostęp 03.01.2022)

Dodatki

# Spis skrótów i symboli

***DDD***domenowe podejście do programowania (ang. *domain driven design*)

**HTTP** protokół przesyłania dokumentów hipertekstowych (ang. *Hypertext Transfer Protocol*)

**REST** styl architektury oprogramowaniaprzy użyciu HTTP (ang. *Representational state transfer*)

**SPA** jednostronicowa aplikacja internetowa (ang. *Single Page Application*)

**CSS** kaskadowe arkusze stylów (ang. *Cascading Style Sheets)*

**HTML** hipertekstowy język znaczników (ang. *HyperText Markup Language)*

**URL** ujednolicony format adresowania zasobów (ang. *Uniform Resource Locator)*

**JWT** JSON Web Token

**JPA** Java Persistence API

**ORM** Object Relational Mapping

**POM** Project Object Model

**MVC** model-widok-kontroler (ang. *Model-View-Controller)*

**DTO** Data Transfer Object

**JDK** Java Development Kit

**SQL** strukturalny język zapytań (ang. *Structured Query Language*)

**RAM** pamięć o dostępie swobodnym (ang*. Random Access Memory*)

**HDD** dysk komputerowy (ang. *hard disk drive*)

**TDD** podejście do programowania w oparciu o testy (ang. *Test Driven Development)*

**IDE** zintegrowane środowisko programistyczne (ang. *Integrated Development Environment)*

**UML** zunifikowany język modelowania (ang. *Unified Modeling Language)*

**AMQP** (ang. *Advanced Message Queuing Protocol)*

# Lista dodatkowych plików, uzupełniających tekst pracy

W systemie, do pracy dołączono dodatkowe pliki zawierające:

* źródła programu,
* wygenerowaną dokumentację
* film pokazujący działanie opracowanego oprogramowania

# Spis rysunków

2.1 Najpopularniejsze usługi oferowane w obiektach służących poprawie kondycji fizycznej

2.2 Widok grafiku aplikacji GymCity

2.3 Wpis informujący o zajęciach w klubie Top Fitness, Facebook

2.4 Wpis informujący o odwołanych zajęciach, Facebook

2.5 Strona Top Fitness, ekran zakupu karnetu

2.6 Strona startowa, Smart Gym

2.7 Widok grafiku zajęć, Smart Gym

3.1 Diagram przypadków użycia, domena użytkowników

3.2 Diagram przypadków użycia, domena wyposażenia

3.3 Diagram przypadków użycia, domena siłowni

3.4 Diagram przypadków użycia, domena aktywności

4.1 Utworzenie nowej bazy danych w narzędziu pgAdmin

4.2 Ekran logowania

4.3 Ekran rejestracji

4.4 Ekran startowy

4.5 Ekran widoku listy siłowni

4.6 Ekran widoku grafiku

4.7 Ekran dodawania aktywności

4.8 Ekran dodawania definicji wyposażenia

4.9 Ekran listy definicji aktywności

4.10 Ekran dodawania aktywności

4.11 Widok dodanych zajęć

4.12 Ekran zakupu karnetu

4.13 Widok aktywności z dostępną opcją zapisania się

4.14 Udany zapis na zajęcia

4.15 Ekran informujący o aktualnym obłożeniu siłowni

4.16 Ekran zapisania wizyty klienta na siłowni

4.17 Ekran potwierdzania obecności na zajęciach

4.18 Ekran profilu użytkownika

5.1 Struktura pakietów warstwy prezentacji

5.2 Struktura pakietów warstwy serwera

5.3 Struktura bazy danych

# Spis kodów

5.1 Fragment klasy *app-routing-module.ts*

5.2 Fragment klasy *GymService*

5.3 Przykładowy konstruktor komponentu

5.4 Kod pobierający listę siłowni

5.5 Metoda zapisująca obiekt siłowni

5.6 Przykładowe pole formularza

5.7 Metoda sprawdzająca rolę użytkownika

5.8 Użycie metody sprawdzającej rolę w komponencie

5.9 *AuthGuard* dla roli Właściciela

5.10 Metoda klasy kontrolera

5.11 Metoda klasy serwisu

5.12 Użycie obiektu wartości

5.13 Metoda inicjalizująca obiekt wartości

5.14 Metoda domenowa potwierdzania obecności obiektu *Activity*

5.15 Metoda domenowa potwierdzania obecności obiektu *Attendance*

5.16 Metoda autoryzująca użytkownika w systemie

6.1 Test jednostkowy tworzenia adresu

6.2 Test jednostkowy symulowana katastrofa

6.3 Test jednostkowy metody *confirmAttendance*

6.4 Test integracyjny tworzenia siłowni

6.5 Test integracyjny zmiany statusu aktywności

1. Urząd Statystyczny w Rzeszowie, 2021, s. 1 [↑](#footnote-ref-1)
2. Eric Evans. *Domain-Driven-Design.* Helion, Gliwice 2015, s. 32 [↑](#footnote-ref-2)
3. Kent Beck. *TDD. Sztuka tworzenia dobrego kodu.* Helion, Gliwice 2020, s. 7 [↑](#footnote-ref-3)
4. Robert C. Martin, *Czysty Agile. Powrót do podstaw*. Helion, Gliwice 2020, s. 42 [↑](#footnote-ref-4)
5. Robert C. Martin. *Czysta architektura.* Helion, Gliwice 2018, s. 28 [↑](#footnote-ref-5)
6. Robert C. Martin. *Czysty kod*. Helion, Gliwice 2014, s. 30 [↑](#footnote-ref-6)
7. Dokumentacja PostgreSQL, wymagania sprzętowe https://www.enterprisedb.com/docs/supported-open-source/postgresql/installer/01\_requirements\_overview/ [↑](#footnote-ref-7)
8. Oficjalna dokumentacja Angular https://angular.io/docs [↑](#footnote-ref-8)
9. Oficjalna dokumentacja TypeScript https://www.typescriptlang.org/docs/ [↑](#footnote-ref-9)
10. Oficjalna dokumentacja biblioteki Bootstrap w wersji 5 https://getbootstrap.com/docs/5.0/getting-started/introduction/ [↑](#footnote-ref-10)
11. Dokumentacja Angular, rozdział o obiekcie Observable https://angular.io/guide/observables-in-angular [↑](#footnote-ref-11)
12. Dokumentacja Angular, rozdział OnInit https://angular.io/api/core/OnInit [↑](#footnote-ref-12)
13. Dokumentacja Spring Boot, https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ [↑](#footnote-ref-13)
14. Oficjalna strona Hibernate https://hibernate.org/ [↑](#footnote-ref-14)
15. Oficjalna strona Apache Maven https://maven.apache.org/ [↑](#footnote-ref-15)
16. Eric Evans. *Domain-Driven-Design.* s. 155 [↑](#footnote-ref-16)
17. Eric Evans. *Domain-Driven-Design.* s. 126 [↑](#footnote-ref-17)
18. Eric Evans. *Domain-Driven-Design.* s. 293 [↑](#footnote-ref-18)
19. Kent Beck. *TDD. Sztuka tworzenia dobrego kodu.* s. 157 [↑](#footnote-ref-19)
20. Kent Beck. *TDD. Sztuka tworzenia dobrego kodu.* s 154 [↑](#footnote-ref-20)