AKADEMIA TARNOWSKA
Wydział Politechniczny

Kierunek: ***Informatyka***
Specjalność: Informatyka Stosowana
Specjalizacja: Inżynieria oprogramowania

2023/2024

Damian Majka

PRACA INŻYNIERSKA

Aplikacja do zarządzania grafikami pracy w firmie

Promotor pracy:
mgr inż. Tomasz Gądek

Tarnów, 2024

Spis treści

1. Wstęp

1.1. Cel pracy

Głównym celem pracy było opracowanie rozwiązania informatycznego umożliwiającego bezproblemowe tworzenie oraz zarządzanie grafikami czasu pracy w firmie oraz automatyzację tego procesu. Zaimplementowana aplikacja będzie miała możliwość generowania grafiku pracy na podstawie zadanych parametrów lub zostanie wyposażona w narzędzie do tworzenia takiego grafiku samodzielnie. Będzie umożliwiała obsługę wniosków urlopowych oraz wniosków o zastępstwa. Poprzez wdrożenie takiego sposobu zarządzania grafikiem potencjalna firma korzystająca z systemu będzie mogła efektywniej planować czas pracy pracowników, minimalizować błędy w planowaniu oraz lepiej wykorzystywać zasoby ludzkie. Praca ta ma na celu dostarczenie narzędzia, które przyczyni się do zwiększenia efektywności firmy i usprawnienia zarządzania personelem.

1.2. Zakres pracy

W ramach wykonania pracy została zaimplementowana:

- Aplikacja desktopowa,
- Baza danych.

1.3. Motywacja

Obecnie wiele firm i instytucji nie posiada oddzielnego systemu do układania grafiku czasu pracy, czy też zarządzania swoimi pracownikami. W dalszym ciągu w wielu mniejszych firmach spotykany jest sposób tworzenia i przekazywania do pracowników grafiku w postaci kartek papieru, co jest zarówno nieekologiczne jak i zajmuję dłuższy czas, który można poświęcić na realizację innych zadań. W jeszcze większej części do zarządzania grafikami i pracownikami używa się narzędzia Google, czy Excel. Takie rozwiązanie jest dużo bardziej ekologiczne, natomiast nie jest ono optymalne. Odpowiednia konfiguracja takich arkuszy oraz późniejsze układanie grafiku szczególnie dla większej ilości pracowników jest jeszcze bardziej czasochłonne i narażone na pomyłki. Uwzględniając tylko te kwestie można zauważyć, że już w przypadku, tworzenia takiego grafiku może nastąpić spadek efektywności

samego managera, czy też innymi słowy przełożonego danej grupy ludzi. Taka osoba mogłaby w czasie, który poświęca na tworzenie grafiku monitorować wyniki swoich pracowników, weryfikować raporty, czy też zajmować się ważniejszymi kwestiami. Problem przekłada się również na samych pracowników, którzy mogą mieć utrudnioną komunikację z przełożonym, czy też mogą nie zawsze mieć dostęp do swojego grafiku, szczególnie jeśli jest on tworzony dynamicznie, na przykład z tygodnia na tydzień. Problematyczny jest również fakt braku automatycznego powiadamiania pracownika o utworzeniu dla niego grafiku.

Można zauważyć jak ważne jest odpowiednie zarządzanie pracownikami i w jaki sposób to umożliwić. Ważne jest, aby maksymalizować wyniki pracownika, czy też całego zespołu. Bez odpowiedniego zarządzania nimi, prawidłowego określania ich godzin pracy i komunikacji odnośnie takich rzeczy jest to niemożliwe.

W takim przypadku konieczne jest zastosowanie aplikacji do zarządzania grafikami czasu pracy w firmie, aby dokonać maksymalizacji spełnienia każdego z parametrów, na podstawie których funkcjonuje firma.

2. Opis wykorzystanych technologii

2.1. Java

Java to język powstały w 1995 roku, który wprowadziła na rynek firma Sun Microsystems. Był to język składniowo oparty na znanym i sprawdzonym języku C++. W przeciwieństwie jednak do swego pierwowzoru język Java nie musiał zachowywać kompatybilności wstecznej z wcześniejszymi językami. Dzięki temu jest on bardziej spójny i elastyczny od C++.
Najważniejszymi cechami charakterystycznymi Javy są:

- przenośność,
- obiektowość,
- bezpieczeństwo,
- sieciowość,
- otwartość [3].

2.2. Maven

Apache Maven to narzędzie do zarządzania projektami oprogramowania. Bazując na koncepcji modelu obiektu projektu (POM), Maven może zarządzać procesem kompilacji projektu, tworzeniem raportów oraz dokumentacją na podstawie centralnego źródła informacji [4].

2.3. Spring i Spring Boot

Spring jest frameworkiem open source, zaproponowanym przez Roda Johnsona i opisanym w jego książce Expert One-on-One: J2EE Design and Development [1]. Spring ma za zadanie sprawić, aby programowanie aplikacji w Javie było łatwiejsze, szybsze i bezpieczniejsze. Powstał on jako alternatywa do EJB - Enterprise JavaBeans. Programowanie w Spring zapewnia dużo większa swobodę w tworzeniu rozwiązań, jednocześnie posiadając wbudowane rozwiązania wielu zagadnień.

Spring Boot, którego można również nazwać swego rodzaju frameworkiem ułatwia tworzenie samodzielnych aplikacji. Aplikacje budowane przy użyciu Spring Boot, można znacznie szybciej uruchomić oraz nie wymagają one znacznej konfiguracji Spring.

Możliwości Spring Boot:

- Tworzenie samodzielnych aplikacji Spring.
- Bezpośrednie osadzanie serwera Tomcat, Jetty oraz Undertow.
- Udostępnianie zależności typu „starter" o zdaniach, które upraszczają konfigurację budowy.
- Automatyczna konfiguracja Springa oraz bibliotek firm trzecich, jeżeli to możliwe.
- Udostępnianie gotowych do użycia funkcji produkcyjnych, takich jak metryki, testy stanu aplikacji (health checks) i konfiguracja zewnętrzna.
- Brak wymogu konfiguracji w formacie XML [6].

2.4 JavaFX

JavaFX jest technologią przeznaczoną do tworzenia graficznych interfejsów użytkownika. W języku Java były wprowadzone początkowo ciężkie komponenty AWT (ang. *Abstract Window Toolkit*), szybko przekształcone w nieatrakcyjne komponenty Swing. JavaFX wprowadził szereg nowych kontrolek oraz możliwość ich ozdabiania przy użyciu języka CSS. Wprowadzono cały szereg nowych rozwiązań, takich jak obserwowalne kolekcje, wiązanie nisko i wysokopoziomowe, nowa obsługa mediów, w tym internetu, itp. JavaFX jest elementem ciągle rozwijanym i adaptującym, aby sprostać coraz to nowszy wymaganiom.

Obecnie JavaFX jest częścią języka Java i pozwala na wykorzystanie wszystkich jego możliwości, przede wszystkim wielowątkowości, programowania uogólnionego i programowania funkcyjnego, w tym wyrażeń lambda oraz modularyzacji. Istnieje kilka sposobów wykorzystywania tego elementu. Jednym z nich jest wykorzystanie w językach używających maszyny wirtualnej Java np. Groovy, Visage, Scala. Może być użyte w postaci XML przy użyciu języka FXML (także z wykorzystaniem wizualnego edytora, np. aplikacji Scene Builder) [2].

2.5 Spring Security

Aby zadbać o bezpieczeństwo danych przechowywanych przez aplikację, czyli jeden z najistotniejszych elementów aplikacji jest używany bardzo często Spring Security. Został zaimplementowany za pomocą Spring AOP i filtrów serwletów, bazującym na frameworku Spring wykorzystującym w pełni wstrzykiwanie zależności oraz techniki aspektowe. Spring Security dba i zapewnia bezpieczeństwo aplikacji bazujących na Springu, poprzez dostarczanie znacznej ilości rozwiązań w zakresie bezpieczeństwa zarówno na poziomie żądania sieciowego jak i na poziomie wywoływania metod. Używany jest głównie do autoryzacji użytkowników poprzez określanie uprawnień do uruchomienia zapytań, czy też metod osadzonych bezpośrednio w kodzie. Przy jego pomocy można zarówno zabezpieczyć hasło użytkownika poprzez stosowanie odpowiedniego rodzaju szyfrowania [8].

2.6 PostgreSQL

PostgreSQL jest systemem zarządzania bazą danych (ang. *Database Management System)* utrzymywanym na zasadach licencji open source. Jest jednym z najbardziej powszechnych i najczęściej używanych DBMS (ang. *Database Management System*). Został on zaprojektowany na Uniwersytecie Kalifornijskim w Berkeley początkowo pod nazwą „*The Berkeley POSTGRES Project*", następnie od 1994 pod naszą „*Postgres95"*. Od 1996 roku znany jest pod nazwą„*PostgreSQL*".

Wskazany DBMS jest jednym z liderów w zakresie wydajności zarządzania bazą danych. Porównując go z innym tego typu systemem czyli MySQL możemy dojść do wniosku, że dla operacji dodawania, aktualizacji i usuwania danych PostgreSQL jest dwukrotnie szybszy niż jego konkurent [9]. Relacyjna baza danych w całości składa się z tabel, które są rozbite na kolumny, a w nich znajdują się atrybuty. Tabele mogą być ze sobą połączone relacjami. PostgreSQL natomiast jest bardziej rozbudowany, XML, key-value (HStore). Możemy w nim również używać funkcji składowych w językach proceduralnych: PL/pgSQL, Perl, Python, Tcl. Dostępne są również inne języki za pomocą rozszerzeń, takie jak Java, JavaScript (V8), R, Lua i Rust. Zaimplementowano w nim również wiele rodzajów indeksów jak na przykład: B-tree, Multicolumn, Expressions, Partial. Posiada zaimplementowaną możliwość partycjonowania tabel, czy też kompilacje wyrażeń just-in-time (JIT) [5].

2.7. Java Persistence API

Java Persistence API jest to narzędzie, które dostarcza mechanizm mapowania obiektowo - relacyjnego. Narzędzie języka Java służące do przekształcania istniejących w języku Java klas, na encje (ang. *Entity)* bazodanowe. Dane są przekształcane poprzez zastosowanie odpowiednich adnotacji dla klasy w języku Java. Przykładowe adnotacje w Javie zaprezentowano w tabeli 2.1 [10].

Adnotacja	Opis
@Entity	Adnotacja mówiąca o tym, że klasa jest encją - reprezentuje tabele bazy danych.
@Table	Adnotacja pozwalająca skonfigurować ustawienia tabel. Może to być na przykład nazwa tabeli, wtedy zapis będzie przedstawiony w następujący sposób: @Table (name = "nazwa_tabeli")
@Id	Adnotacja oznaczająca dane jako identyfikator, czyli klucz główny tabeli.
@GeneratedValue	Adnotacja powiązana z @Id określa w jaki sposób wartość identyfikatora jest generowana, @GeneratedValue(strategy = GenerationType.IDENTITY)
@Column	Adnotacja pozwalająca skonfigurować ustawienia kolumn z tabeli. Może to być nazwa kolumny. Wtedy zapis będzie przedstawiony w następujący sposób: @Column(name = "product_id")

Tabela 2.1. Przykładowe adnotacje Java Persistence API [Opracowanie własne]

Oprócz adnotacji odnośnie samej tabeli, istnieją adnotacje określające relacje w bazie danych takie jak zaprezentowano w tabeli 2.2.

Adnotacja relacji	Opis adnotacji
@OneToOne	Adnotacja reprezentująca bazodanową relacje 1:1
@OneToMany	Adnotacja reprezentująca bazodanową relacje 1:∞
@ManyToOne	Adnotacja reprezentująca bazodanową relacje ∞ :1
@ManyToMany	Adnotacja reprezentująca bazodanową relacje ∞:∞

Tabela 2.2. Adnotacje relacji Java Persistence API [Opracowanie własne]

2.8. Spring Data & Spring Data JPA

Spring Data ma za zadanie dostarczenie opartego na Springu modelu programowania dostępu do danych. Aby wciąż zachowywać cechy bazodanowego magazynu danych. Spring Data ułatwia korzystanie z dostępu do danych różnego rodzaju, na przykład baz relacyjnych, jak i nierelacyjnych, czy też rozwiązań chmurowych. Głównymi zaletami korzystania ze Spring Data są:

- Wykonywanie zapytań bazodanowych na podstawie nazwy metody.
- Zaawansowana integracja z Spring MVC.
- Możliwość integracji niestandardowego kodu repozytorium.
- Łatwa konfiguracja z Spring przy użyciu JavaConfig.

Spring Data posiada również wiele modułów:

- Spring Data JPA,
- Spring Data JDBC,
- Spring Data MongoDB,
- Spring Data REST.

Spring Data JPA jest narzędziem Spring Data, które ma za zadanie ułatwić implementację repozytoriów w oparciu o JPA (Java Persistence API). Spełnia to zadanie poprzez

automatyczne konfiguracje repozytoriów, posiadanie metod, na przykład pozwalających na paginację oraz sortowanie danych, czy też dynamiczne wykonywanie zapytań [7].

2.9. Spring Email

Framework Spring Boot dostarcza jedno z bardziej użytecznych narzędzi podczas implementacji aplikacji, czyli moduł Spring Email. Służy on do zarządzania procesem wysyłki wiadomości email. Umożliwia efektywne i elastyczne zarządzanie procesem wysyłania e-maili z aplikacji. W wielu przypadkach jest to kluczowe w nowoczesnych systemach [6]. Spring Email jest używany w aplikacjach biznesowych między innymi do realizacji takich funkcjonalności jak:

- Wysyłanie potwierdzeń, np. potwierdzenie pomyślnej rejestracji.
- Wysyłanie powiadomień, np. w celu powiadomienia użytkowników o planowanych pracach serwisowych.
- Raportowanie, np. automatyczne wysyłanie raportów, innych danych generowanych przez użytkownika.
- Zarządzanie subskrypcją, np. wysyłanie ofert dla aplikacji biznesowych posiadających newsletter.

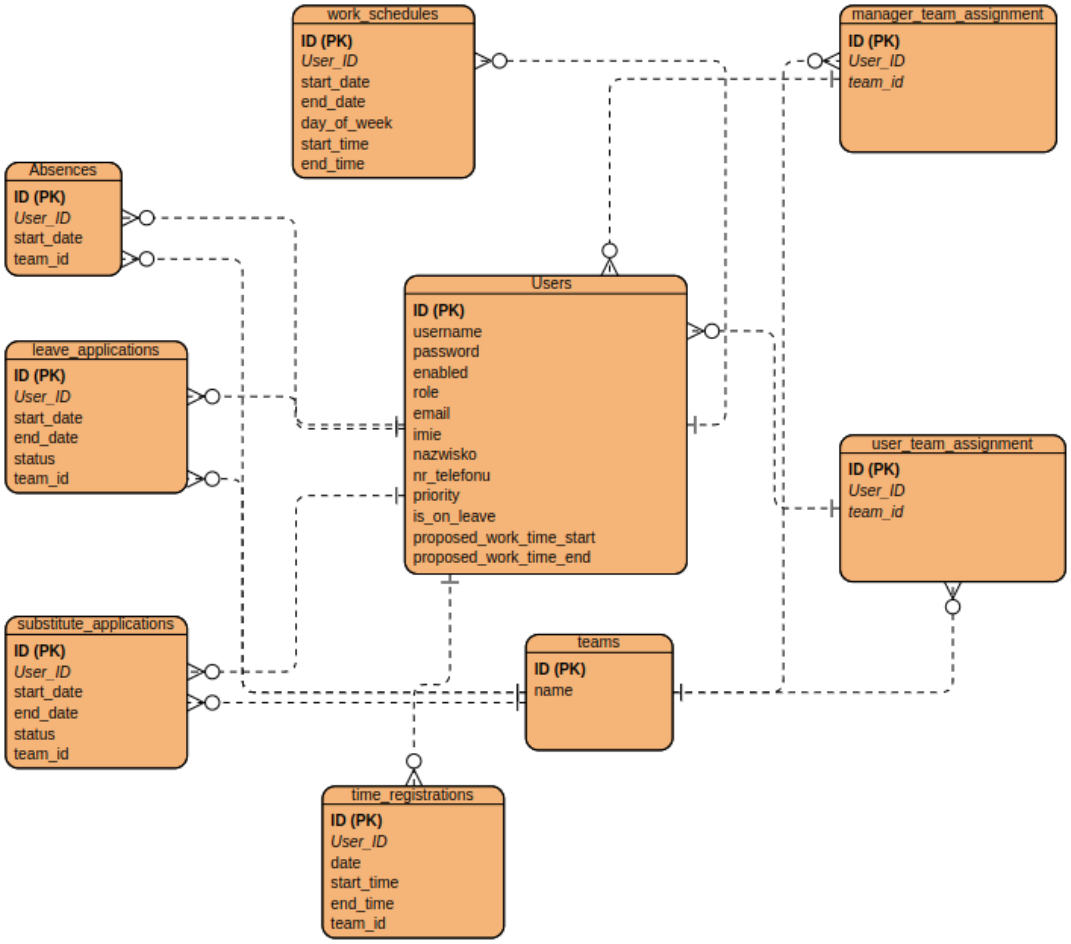
3. Diagram ERD

Diagram ERD czyli Model Związków Encji (ang. *Entity Relationship Diagram*) jest jedną z najpopularniejszych metod przedstawiania modelowania danych. Diagram przedstawia obiekty (encje) szerzej znane pod nazewnictwem tabele, wraz z określającymi je atrybutami. Jednym z nich jest zawsze klucz główny, czyli tak zwany identyfikator. Może zawierać również klucze obce czyli odniesienia do identyfikatora wiersza z innej tabel. Przedstawia również relacje między nimi. Wyróżniane jest kilka rodzajów relacji. Przedstawiono je w tabeli 3.1.

Rodzaje relacji	Opis relacji
Jeden do jednego	Przedstawia powiązanie jednego elementu danej encji do tylko i wyłącznie jednego elementu, na przykład powiązanie (relacje) obiektu samochód do obiektu miejsce parkingowe. Gdzie jeden samochód może zajmować tylko jedno miejsce parkingowe
Jeden do wielu	Przedstawia powiązanie jednego elementu danej encji do nieskończonej możliwości odwołań innego elementu, na przykład powiązanie obiektu firma do obiektu pracownik. Gdzie jedna firma może mieć nieskończenie wiele pracowników.
Wiele do wielu	Przedstawia powiązanie wielu elementów danej encji do nieskończonej możliwości odwołań do innego obiektu, na przykład relacja między obiektami pracowników i grup w firmie. Gdzie dany pracownik może pracować w kilku działach, jak i każdy dział może mieć kilku pracowników

Tabela 3.1. Typy relacji w relacyjnej bazie danych [Opracowanie własne]

Na rysunku 3.1. przedstawiono zaprojektowaną bazę danych do realizacji projektu aplikacji w formie Diagramu ERD.



Rys. 3.1 Diagram ERD [Opracowanie własne]

Główną uwagę w zaprojektowanej bazie danych należy zwrócić na tabelę „users", w której przechowywane są dane użytkownika wraz z jego rolą oraz informacje dotyczące logowania, roli, wraz z podstawowymi informacjami osobowymi o użytkowniku. Każdy użytkownik aplikacji posiadający utworzone konto, jest zapisany do wskazanej tabeli, a następnie to właśnie z tej tabeli są pobierane wszelkie dane o użytkowniku.

Drugą z tabel, jest „Teams". Mimo jej pozornie małych rozmiarów, ponieważ składa się jedynie z identyfikatora - klucza głównego oraz nazwy zespołu, pełni ona kluczową rolę w funkcjonalności aplikacji.

W celu przypisania użytkowników do zespołów w zależności od ich roli wykorzystano tabele „user_team_assignment" dla aktorów o roli users oraz

„manager_team_assignment" dla aktorów o roli manager, które przyjmują wartości kluczy obcych użytkownika z tabeli users oraz zespołu z tabeli „teams”.

W celu obsługi wniosków urlopowych pracowników oraz wniosków o zastępstwo zostały zaprojektowane dwie osobne tabele „leave_application" dla wniosków urlopowych oraz „substitute_application" dla wniosków o zastępstwo. Nie są one przypisywane do obsługi bezpośrednio przez przełożonego tylko do zespołu, aby nawet w przypadku nieplanowanej zmiany przełożonego, możliwa była obsługa wniosków złożonych przed zmianą managera.

Kolejną z tabel jest tabela „absences", gdzie są zapisywane dane odnośnie zgłoszonych przez pracowników nieobecności.

Przedostatnią, odpowiadającą za jedną z kluczowych funkcjonalności w działaniu aplikacji tabelą jest tabela „time_registrations", w której zapisywane są dane dotyczące rejestracji czasu pracy pracowników.

Ostatnią, siódmą tabelą w przedstawionej bazie danych jest tabela „work_schedules", która to odpowiada za przechowywanie danych w harmonogramie pracy każdego z pracowników. Jest to tabela, która w czasie żywotności aplikacji przechowuje największą ilość danych i to na niej skupia się główna funkcjonalność aplikacji.

4. Funkcjonalność aplikacji

Wykonany system posiada podstawowych czterech aktorów jakimi są:

- Gość,
- Użytkownik zalogowany - pracownik,
- Użytkownik zalogowany - manager,
- Użytkownik zalogowany - admin.

Rola Gość	Przypadki użycia jako gość
Logowanie	• Możliwość uzupełnienia formularzu logowania celem zalogowania.
Przypomnienie hasła	• Możliwość uzupełnienia formularza wysyłającego jednorazowe hasło na adres e-mail.

Tabela 4.1 Przypadki użycia jako gość [Opracowanie własne]

Rola pracownik	Przypadki użycia jako pracownik
Grafik	● Możliwość podglądu swojego grafiku. ● Możliwość uzupełnienia danych odnośnie proponowanych godzin pracy.
Zgłoszenie zastępstwa	● Możliwość uzupełnienia wniosku o zastępstwo.
Zgłoszenie urlopu	● Możliwość uzupełnienia formularza urlopowego.
Zgłoszenie nieobecności	● Możliwość zgłoszenia nieobecności.
Czas pracy	● Możliwość zapisanie czasu pracy w danym dniu.
Ustawienia profilu	● Wyświetlanie swoich danych. ● Zmiana hasła. ● Podgląd oraz zmiana danych kontaktowych (nr telefonu / adres e-mail).

Tabela 4.2 Przypadki użycia jako pracownik [Opracowanie własne]

Rola managera	Przypadki użycia jako manager
Podgląd grafiku	● Możliwość wyboru grupy oraz użytkownika do podglądu grafiku.
Automatyczne tworzenie grafiku:	● Określenie godziny otwarcia oraz ilości pracowników pracujących w danym momencie oraz maksymalnej ilości godzin pracy jednego pracownika i ułożenie grafiku automatycznie.
Ręczne tworzenie grafiku	● wyświetlenie informacji o priorytecie użytkownika oraz jego proponowanych godzinach pracy. ● Uzupełnienie danych dla danego pracownika i ułożenie grafiku.
Wnioski o urlop i zastępstwo	● Wyświetlenie wniosków. ● Akceptacja lub odrzucenie wybranego wniosku.
Zgłaszanie nieobecności	● Podgląd zgłoszonych informacji o nieobecnościach.
Lista pracowników	● Dostęp do listy pracowników. ● Zmiana priorytetu. ● Modyfikacja danych kontaktowych. ● Wysłanie wiadomości email.

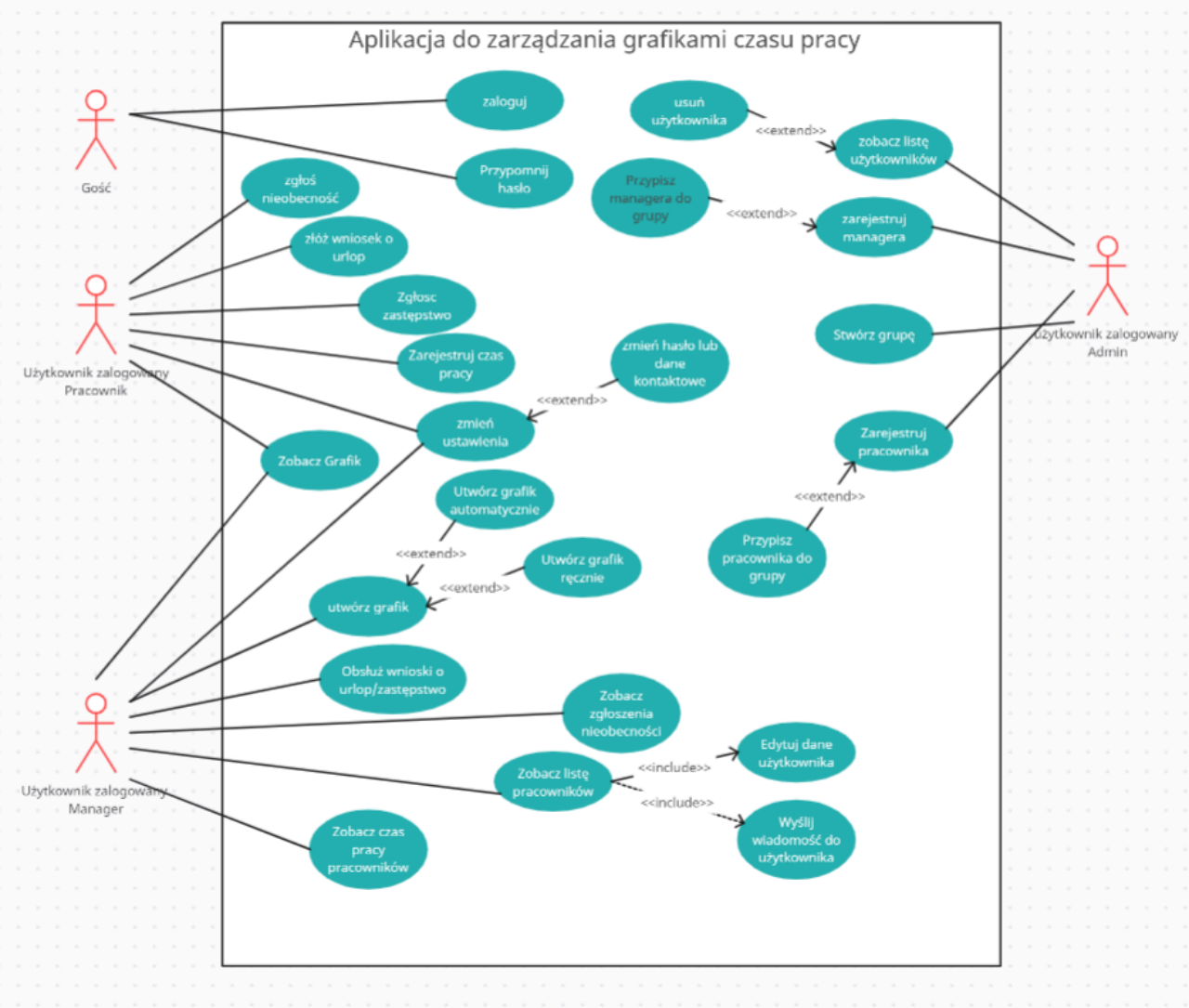
Czas pracy	• Wybór grupy. Wyświetlanie wszystkich godzin pracy zarejestrowanych przez użytkowników w grupie. • Wybór grupy oraz użytkownika. Wyświetlenie szczegółowych informacji na temat każdorazowej rejestracji czasu pracy.
Ustawienia profilu	• Wyświetlanie swoich danych. • Zmiana hasła. • Podgląd oraz zmiana danych kontaktowych (nr telefonu / mail).

Tabela 4.3 Przypadki użycia przez Managera [Opracowanie własne]

Rola Admin	Przypadki użycia jako admin
Wyświetlenie listy pracowników	• Możliwość usunięcia pracownika lub managera. • Usunięcie dowolnego użytkownika.
Dodanie pracownika	• Uzupełnienie formularza i założenie konta pracownikowi. • Przypisanie pracowników do grupy. • Uzupełnienie formularza i założenie konta managerowi. • Przypisanie managera do grupy.
Grupy	• Wyświetlanie listy grup. • Utworzenie nowej grupy.

Tabela 4.4 Przypadki użycia przez admina [Opracowanie własne]

Diagram przypadków użycia są rodzajem diagramów UML (*Unified Modeling Language*). Służą one do modelowania funkcji systemu z perspektywy użytkowników (aktorów). Taki diagram przypadków użycia został zaprezentowany na rysunku 4.1. Rysunek ten przedstawia w formie skróconych opisów funkcje każdej roli. Aktorzy zostali przedstawieni za pomocą odpowiednich ikon. Możemy dzięki takiej formie przedstawienia, z dużo większą łatwością zaobserwować różnicę w funkcjonalnościach, które odpowiedni aktorzy powinni posiadać.



Rys. 4.1 Diagram przypadków użycia [Opracowanie własne]

Na diagramie (rysunek 4.1) możemy zaobserwować jakie funkcjonalności posiada każdy z aktorów. Zostali oni opisani za pomocą odpowiednich ikon na skrajnych częściach diagramu. Od aktorów zostały poprowadzone linie wskazujące możliwe funkcjonalności dostępne dla nich. Na diagramie można zaobserwować, że niektóre funkcjonalności, jak na przykład opcja „Zobacz Grafik" są dostępne dla większej ilości aktorów, a niektóre, jak na

przykład „Stwórz grupę”, czy też „Złóż wniosek o urlop” są unikalne dla jednego z aktorów. Na diagramie można również zaobserwować, które funkcjonalności wynikają z pozostałych. Oznacza to, że aktor o roli manager, będzie mieć możliwość edycji użytkowników dopiero po uruchomieniu ich podglądu. Analogicznie wygląda sytuacja rozpatrywania wniosków przez managera, gdzie może to zrobić dopiero w momencie, gdy w pierwszej kolejności zostanie uruchomiona możliwość podglądu wniosku.

4.1. Wymagania niefunkcjonalne aplikacji

Do realizacji projekt konieczne było zadbanie o kilka czynników wpływających na komfort użytkowania aplikacji, przez każdego z użytkowników niezależnie od jego roli.

4.1.1. Bezpieczeństwo

Podstawowym elementem w realizacji projektu jest bezpieczeństwo. W celu jego zapewnienia konieczne było zaimplementowanie odpowiedniego mechanizmu autoryzacji i uwierzytelniania. Uwierzytelnianie jest to weryfikacja tożsamości użytkownika, logującego się do systemu. Hasło użytkownika powinno być szyfrowane odpowiednim algorytmem. W realizacji pracy została użyta funkcja haszująca „BCrypt". Autoryzacja natomiast jest to określenie czy dany użytkownik powinien posiadać uprawnienia do danego zasobu, w tym celu użyty został framework Spring Security.

4.1.2. Wydajność

Baza danych w przypadku dłuższego funkcjonowania aplikacji zacznie gromadzić dużą ilość danych, a co za tym idzie szybkość wykonywania każdego zapytania z czasem by malała. Stąd też ważne jest, aby zadbać o optymalizację przechowywania wszelkich danych, na przykład poprzez automatyczne usuwanie nieużywanych wierszy z tabel bazy danych.

4.1.3. Interfejs użytkownika

Kolejną ważna kwestią jest zaprojektowanie przyjaznego i przejrzystego dla użytkownika interfejsu graficznego. Użytkownik powinien mieć bezproblemowy dostęp do każdej z funkcjonalności. Interfejs powinien zostać zaimplementowany w taki sposób, aby nie generował on długiego czasu oczekiwania na wczytanie elementów. Powinien być

pozbawiony nadmiarowej ilości wszelkiego rodzaju niepotrzebnych animacji, czy też dużych plików multimedialnych.

4.2. Baza danych

Do implementacji projektu inżynierskiego konieczne było zaprojektowanie i zaimplementowanie bazy danych, spełniającej odpowiednie wymogi. Baza danych została zaprojektowana z myślą o późniejszym rozwoju, bez zbędnych pozycji, które nie są wymagane do działania aplikacji, ani nigdy nie zostaną użyte. Baza danych została odpowiednio znormalizowana, aby nie była redundantna. Zaprojektowana baza danych składa się z 9 tabel:

- users,
- teams,
- substitute_applications,
- leave_applications,
- manager_team_assignment,
- users_team_assignment,
- work_schedules,
- time_registrations
- absences.

Tabele zostały przedstawione i opisane dokładnie w rozdziale poświęconym diagramom ERD (ang. *Entity-Relationship Diagram*)

Na każdą z tabel nałożono odpowiednie ograniczenia. Przykładowe zastosowane ograniczenia dla wartości kolumn w bazie danych przedstawiono w tabeli 4.5

Ograniczenia wartości	Opis ograniczenia
UNIQUE	Wymaga, aby wartość wprowadzona była unikalna. Używana, np. przy tworzeniu użytkowników, aby nie było możliwości zarejestrowania dwóch użytkowników o tym samym loginie.
NOT NULL	Wymaga, aby wartość dla tej kolumny nie była pusta. Używana, np. przy rejestracji użytkownika, dla kolumny password, aby nie utworzyć użytkownika, który nie będzie posiadał hasła
DEFAULT	Pozwala określić domyślna wartość, w przypadku jeśli nie została podana wartość dla kolumny, dla której jest zastosowana. Używana w projekcie, np. do ustawienia domyślnego priorytetu.
CHECK	Pozwala dodać kryteria, np. aby wprowadzony wiek nie mógł być niższy niż 18.
AUTO_INCREMENT	Używane najczęściej do wartości ID danej kolumny, w celu automatycznego jej inkrementowania.

Tabela 4.5 Przykłady ograniczeń dla wartości wprowadzanych do tabeli
[Opracowanie własne]

Dodatkowo, w celu odpowiedniej optymalizacji aplikacji, część zadań zostało zaimplementowanych po stronie bazy danych. Utworzono wyzwalacz, czyli odpowiednie funkcje wywołujące się podczas wykonywania określonej czynności, na przykład operacji

dodawania danych do tabeli, usuwania czy też aktualizacji. Przykład zastosowania takiej funkcji zaprezentowano na listingu 4.1.

```
CREATE OR REPLACE FUNCTION set_default_team_id()
RETURNS TRIGGER AS $$
BEGIN
  NEW.id_teamu := (
    SELECT team_id
    FROM user_team_assignment
    WHERE user_id = NEW.user_id
  );
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER set_team_id_trigger
BEFORE INSERT ON leave_applications
FOR EACH ROW
EXECUTE FUNCTION set_default_team_id();
```

Listing 4.1 Wyzwalacz set_default_team_id [Opracowanie własne]

W celu dokładnego opisania przedstawionej funkcji, należałoby w pierwszej kolejności przyglądnąć się działaniu funkcjonalności wniosków urlopowych. Wniosek taki jest składany przez pracownika poprzez odpowiedni formularz. Następnie wniosek jest obsługiwany przez managera przypisanego jako przełożony do zespołu, w którym znajduje się dany pracownik. Po utworzeniu wniosku funkcja ta ma za zadanie przypisania pracownika do odpowiedniego zespołu. Bez użycia takiej funkcji konieczne byłoby wykonanie kolejnego zapytania, realizującego tę funkcję lub napisanie mniej optymalnego zapytania SQL.

Przy implementacji obsługi bazy danych należało pamiętać o odpowiednich zabezpieczeniach i zagrożeniach płynących z ich braku. Głównym takim zagrożeniem, jest narażenie danych na atak typu SQL Injection. Ten sposób polega na umieszczaniu fragmentu kodu SQL w miejscu, gdzie przykładowo powinna być wpisana nazwa użytkownika. Przykładem użycia takiego sposobu jest wywołanie zapytania.

```
SELECT  * FROM User WHERE username = przykladowyUzytkownik;
```

Listing 4.2 Przykładowe zapytanie SQL [Opracowanie własne]

Zapytanie wydaje się bezpieczne natomiast ważne jest aby dokonać poprawnej jego implementacji w kodzie. W przypadku użycia np. połączenia dwóch stringów jak to zaprezentowano na listingu 4.3

```
String Sql = "SELECT * FROM User WHERE UserId = ";
String Username = "przykladowyUżytkownik";
String Query = Sql + Username;
```

Listing 4.3 Przykładowe utworzenie zapytania SQL [Opracowanie własne]

użytkownik mógłby podać nazwę użytkownika jako „username OR 1=1”, co sprawiłoby, że zostanie zwrócona lista wszystkich użytkowników. W celu zapobiegania takim działaniom należało użyć odpowiedniego sposobu przesyłania zapytań, np:

```
@Query("SELECT * FROM User WHERE username = :username")
Long getUserByUsername(@Param("username") String username);
```

Listing 4.4 Przykładowe utworzenie zapytania SQL [Opracowanie własne]

4.3. Zabezpieczenia

W celu odpowiedniego zabezpieczenia przed dostępem przez nieuprawnione osoby, została wdrożona do aplikacji ścisła polityka silnych haseł oraz zaawansowane techniki szyfrowania. Implementacja tych metod obecnie jest właściwie obowiązkowa, w celu zabezpieczenia przed atakami cybernetycznymi takimi jak brute force, czy metody słownikowe.

4.3.1. Polityka Silnych Haseł

W aplikacji została zastosowana polityka silnych haseł. W ramach tej polityki, każdy użytkownik ma obowiązek ustawienia hasła, które będzie spełniało odpowiednie kryteria:

- Długość - ustalono, aby hasło użytkownika nie było krótsze niż 12 znaków. Taka ilość została ustalona ze względu na ilość kombinacji, które trzeba brać pod uwagę w celu złamania hasła.
- Złożoność hasła - ustalono, ze względów bezpieczeństwa, konieczność używania różnych znaków. Hasło nie może składać się tylko z cyfr, lub samych liter. Konieczne

jest ustalenie hasła, które będzie posiadało zarówno małą i dużą literę, cyfrę oraz znak specjalny. Taki zabieg jest konieczny, w celu zapewnienia bezpieczeństwa. Na przykładzie porównania ilości kombinacji dla samych cyfr, jak i cyfr oraz małych liter. Różnicę tę przedstawiono w tabeli 4.6

Długość hasła	Ilość kombinacji
4 cyfry	10 000
8 cyfr	100 000 000
12 cyfr	1 000 000 000 000
4 cyfry i litery	1 679 616
8 cyfr i liter	2 821 109 907 000
12 cyfr i liter	4 738 381 338 000 000 000

Tabela 4.6 ilości kombinacji dla różnej długości haseł [Opracowanie własne]

Na przykładzie tabeli 4.6 możemy zaobserwować, że już dla hasła składającego się z 4 znaków jest znaczna rozbieżność w ilości kombinacji, gdyż z samych cyfr można utworzyć 10 000, natomiast zakładając umieszczenie w haśle liter, na przykładzie alfabetu angielskiego, już dla 4 znaków jest to ilość możliwych kombinacji 1 679 616.

4.3.2. Szyfrowanie haseł

Przed zapisaniem do bazy danych, hasła użytkowników są szyfrowane przy użyciu algorytmu BCrypt. BCrypt jest zaawansowanym algorytmem hashującym, który jest szczególnie skuteczny w zabezpieczaniu haseł ze względu na jego zdolność do oporu wobec ataków siłowych i słownikowych. Kluczowe cechy algorytmu BCrypt obejmują:

- Soling - BCrypt automatycznie generuje i stosuje unikalną sól do każdego hasła przed jego haszowaniem. Dzięki temu, nawet identyczne hasła po zahashowaniu będą miały różne wartości, co znacząco utrudnia ataki typu „rainbow table”.
- Adaptacyjność - BCrypt pozwala na regulowanie „kosztu” operacji hashowania, co oznacza, że można zwiększyć ilość pracy potrzebnej do wygenerowania każdego

hasha. W miarę rozwoju technologii obliczeniowych, koszt ten może być zwiększany, aby utrzymać wysoki poziom bezpieczeństwa.

4.4. Komunikacja

W aplikacji konieczne było zaimplementowanie metody komunikacji, aby użytkownik na bieżąco otrzymywał informację, gdy:

- zostało założone konto użytkownika,
- złożył wniosek o urlop, bądź zastępstwo,
- została wysłana prośba o urlop,
- jego wniosek został rozpatrzony,
- użytkownik zapomniał hasła i zostało ono zresetowane.

W tym celu wykorzystano moduł Spring Email. Moduł ten pozwala na kompleksową obsługę wysyłania wiadomości mailowych przy użyciu protokołu SMTP. Aby użyć modułu Spring Email konieczna jest jego wcześniejsza konfiguracja do obsługi odpowiedniego serwisu pocztowego. W aplikacji projektowej, został on skonfigurowany do obsługi poczty Outlook. Konfiguracja polegała na wartości w pliku „application.properties”. Do poprawnej konfiguracji konieczne było ustawienie następujących wartości:

- Adres serwera SMTP,
- Port,
- Dane uwierzytelniające,
- Szyfrowanie.

Dla zaimplementowanej aplikacji zostały one przedstawione na listingu 4.5

```
spring.application.ui.title=exampleNameOfProject
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=examplemail@gmail.com
spring.mail.password=examplepassword
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Listing 4.5 Konfiguracja Spring Email [Opracowanie własne]

Dzięki takiej implementacji można wysłać e-mail za pomocą metody „sendSimpleMessage” (listing 4.6).

```
@Autowired
private JavaMailSender mailSender;

public void sendSimpleMessage(String to, String subject, String text) {
    SimpleMailMessage message = new SimpleMailMessage();
    message.setFrom("noreply@example.com");
    message.setTo(to);
    message.setSubject(subject);
    message.setText(text);
    mailSender.send(message);
}
```

Listing 4.6 Użycie metody wysyłającej wiadomośc mailową [Opracowanie własne]

4.5. Implementacja wybranych funkcjonalności

4.5.1. Automatyczne tworzenie grafiku.

Do zaimplementowania funkcjonalności automatycznego przypisywania grafików czasu pracy została zaimplementowana dodatkowa klasa „ScheduleGenerator”. Właśnie w tej klasie w połączeniu z serwisem „WorkScheduleService”, następuje przypisanie użytkownikom ich grafiku czasu pracy. Klasa ta składa się z 7 metod oraz konstruktora. Pierwsza z metod:

```
public void generateSchedule() {
        LocalDate currentDate = startDate;
        while (!currentDate.isAfter(endDate)) {
            generateDailySchedule(currentDate);
            currentDate = currentDate.plusDays(1);
        }
    }
```

Listing 4.7 metoda „generateSchedule” [Opracowanie własne]

Generuje ona harmonogram pracy na każdy dzień od „currentDate” do „endDate”. Następną z metod użytych w tej klasie to „generateDailySchedule”, która ma za zadanie iterować po każdej godzinie i przypisuje pracownika do danej godziny.

```
private void generateDailySchedule(LocalDate date) {
        LocalTime currentTime = openingTime;
        while (currentTime.isBefore(closingTime.minusHours(1))) {
            currentTime = currentTime.plusHours(1);
            assignEmployeesToHour(date, currentTime);
        }
    }

    private void assignEmployeesToHour(LocalDate date, LocalTime time) {
        List<User> availableUsers = getSortedAvailableUsers(date);
        Set<Long> assignedUserIds = new HashSet<>();

        for (User user : availableUsers) {
            if (assignedUserIds.size() >= requiredEmployeesPerHour) {
                break;
            }

            if (isWithinProposedHours(user, time) && canBeAssigned(user, date)) {
                assignUserToSchedule(date, time, user, assignedUserIds);
            }
        }
        if (assignedUserIds.size() < requiredEmployeesPerHour) {
            for (User user : availableUsers) {
                if (assignedUserIds.size() >= requiredEmployeesPerHour) {
                    break;
                }
        if (!isWithinProposedHours(user, time) && canBeAssigned(user, date)) {
               assignUserToSchedule(date, time, user, assignedUserIds);
        }
            }
        }
    }
```

Listing 4.8 metoda „generateDailySchedule” oraz „assignEmplyeesToHour” [Opracowanie własne]

Wskazany na listingu 4.8 fragment kodu zawiera dwie metody, pierwsza, która ma za zadanie iterować po każdej godzinie danego dnia, w celu przypisania do niej pracowników przy pomocy drugiej metody, czyli „assignEmployeesToHour”. W Metodzie „assignEmplyeesToHour”, w pierwszej kolejności pobierana jest lista odfiltrowanych i posortowanych pod względem priorytetu użytkowników, a następnie przypisywane są godziny pracy, które mieszczą się w obrębie proponowanych godzin pracownika. Później przypisywane są pozostałe godziny.

```
private void assignUserToSchedule(LocalDate date, LocalTime time, User user, Set<Long>
assignedUserIds) {
        createAndAddWorkSchedule(date, time, user);
        user.addAssignedHours(date, 1);
        assignedUserIds.add(user.getId());
    }
```

Listing 4.9 metoda „assignUserToSchedule” [Opracowanie własne]

W przedstawionym na listingu 4.9 fragmencie kodu przedstawiona została metoda „assignUserToSchedule”. Odpowiedzialna jest za kontynuacje przydzielania pracownikowi godzin pracy, przypisanie danych bezpośrednio do encji, natomiast ta wartość nie jest przesyłana do bazy danych ze względu na użycie w niej adnotacji `@Transient`.

4.5.1. Dodawanie czasu pracy

Na listingu 4.10 przedstawiono metodę odpowiadającą za rejestrowanie czasu pracy, metoda ta przyjmuje argument daty oraz dwie liczby całkowite odpowiadające za reprezentacje godziny rozpoczęcia i zakończenia pracy.

```
public Boolean addTimeRegistration(LocalDate date, Integer startTime, Integer endTime) {
        if (!date.isBefore(LocalDate.now()) || startTime == null || endTime == null ||
            endTime <= startTime) {
            return false;
        }
        String loggedInUsername =
        SecurityContextHolder.getContext().getAuthentication().getName();
        User loggedInUser = userRepository.findByUsername(loggedInUsername);

        TimeRegistration timeRegistration = new TimeRegistration();
        timeRegistration.setUserId(loggedInUser);
        timeRegistration.setDate(date);
        timeRegistration.setStartTime(startTime);
        timeRegistration.setEndTime(endTime);
        timeRegistrationRepository.save(timeRegistration);

        return true;
    }
```

Listing 4.10 metoda „addTimeRegistration” [Opracowanie własne]

4.5.3. Składanie wniosków o urlop

Na listingu 4.11 przedstawiono metodę pozwalającą na składanie wniosków o urlop. Metoda ta otrzymuje parametry takie jak data początku oraz końca urlopu. Tworzy ona instancję obiektu „LeaveApplication” oraz pobiera dane odnośnie aktualnie zalogowanego użytkownika, aby później zapisać dane rejestracji. Jeśli rejestracja się powiedzie, metoda zwraca „True”, jeśli się nie powiedzie zwraca „False”. Następnie wynik jest obsługiwany w kontrolerze do wyświetlenia stosownego komunikatu użytkownikowi.

```
public Boolean applyForLeave(LocalDate startData, LocalDate endData) {
        LeaveApplication leaveApplication = new LeaveApplication();
        leaveApplication.setStatus("oczekujący");
        leaveApplication.setStartDate(startData);
        leaveApplication.setEndDate(endData);
        String loggedInUsername = SecurityContextHolder
                                  .getContext()
                                  .getAuthentication()
                                  .getName();
        User loggedInUser = userRepository.findByUsername(loggedInUsername);
        leaveApplication.setUser(loggedInUser);
        LeaveApplication result = leaveApplicationRepository.save(leaveApplication);
        if(result == null){
            return false;
        }else{
            return true;
        }
    }
```

Listing 4.11 metoda „applyForLeave” [Opracowanie własne]

4.5.5. Dodanie nieobecności

Metoda przedstawiona na listingu 4.12 realizuje dodanie zgłoszenia nieobecności do metody w serwisie, przekazywana jest data wybrana przez użytkownika, kiedy zostaje zgłoszona nieobecność. Tworzona jest instancja obiektu „Absences” i ustawiane są jego zmienne, w tym aktualnie zalogowany użytkownik. Wynikiem funkcji jest zmienna typu „Boolean”, na podstawie której później jest wyświetlany stosowny komunikat dla użytkownika.

```
public Boolean addAbsences(LocalDate absenceDate) {
        Absences absences = new Absences();
        String loggedInUsername =
        SecurityContextHolder.getContext().getAuthentication().getName();
        User loggedInUser = userRepository.findByUsername(loggedInUsername);
        absences.setUser(loggedInUser);
        absences.setDate(absenceDate);
        Absences result = absencesRepository.save(absences);
        if(result == null){
            return false;
        }else{
            return true;
        }
    }
```

Listing 4.12 metoda „addAbsences” [Opracowanie własne]

5. Interfejs użytkownika

W niniejszym rozdziale zostanie opisany interfejs użytkownika (*ang. UI - user interface*) zaprojektowany na potrzeby implementacji projektu zostaną opisane zasady, którymi kierowano się podczas jego projektowania oraz różne aspekty dotyczące tego elementu.

System nie składa się jedynie z samych funkcjonalności działających na bazie danych, czy też komunikacji przy użyciu różnych protokołów, a użytkownik nie jest w stanie operować tylko na takich danych. Taka osoba komunikuje się z systemem poprzez interfejs użytkownika. Oprócz samej komunikacji z użytkownikiem UI odpowiada również za inne operacje. Przykładowe role opisano w tabeli 5.1.

Rola interfejsu	Opis roli
Prezentacja informacji	Interfejs pełni rolę wizualnej prezentacji danych i informacji zawartych w systemie bądź bazie danych. Dostarcza dane w sposób czytelny i zrozumiały dla użytkownika.
Wspomaganie nawigacji	Interfejs wspomaga w nawigacji po systemie, jasno wskazuje, gdzie użytkownik znajdzie daną funkcjonalność.
Zarządzanie danymi	Interfejs umożliwia użytkownikom wprowadzanie danych, ich modyfikację, czy też usuwanie poprzez zastosowanie. np. formularzy przycisków.
Zabezpieczenie i kontrolowanie dostępu	Interfejs w zależności od uprawnień użytkownika może się różnić, co zabezpiecza dodatkowo przed uruchomieniem danej funkcji przez nieuprawnionego użytkownika [14].

Tabela 5.1 Przykładowe role UI [Opracowanie własne]

5.1. Technologie do projektowania interfejsu aplikacji

Obecnie do tworzenia interfejsu użytkownika dostępnych jest wiele różnych technologii. Do zaimplementowania aplikacji użytkownika użytko JavaFX, które jak sama nazwa wskazuje, jest narzędziem do tworzenia interfejsu użytkownika w języku Java. Technologia ta umożliwia tworzenie zarówno interfejsów dla aplikacji desktopowych, jak i mobilnych. Wykorzystano następujące elementy JavaFX:

- Języka FXML (*FXML Markup Language*), który pozwala na definiowanie elementów w formie plików XML,
- Kontrolery, które zostały zaimplementowane do obsług interakcji użytkownika oraz współpracy z warstwą logiki biznesowej.

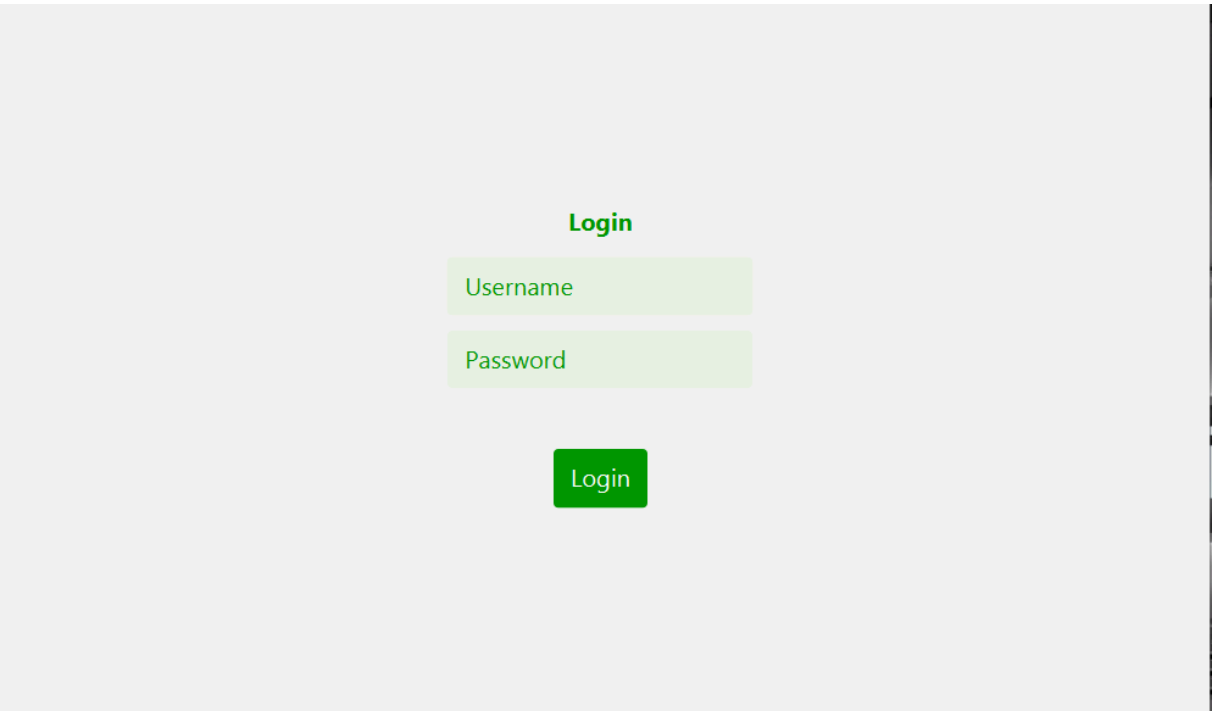
5.2. Projektowanie Interfejsu

Przy projektowaniu interfejsu użytkownika należało pamiętać o spełnieniu wszystkich z przedstawionych w tabeli 5.1 ról interfejsu oraz o doborze odpowiedniej kolorystyki. Biorąc pod uwagę wszystkie aspekty zdecydowano się na zaprojektowanie UI w kolorystyce odcieni zieleni. Badania pokazują, że kolory odpowiadają za nastrój, a także odczucia ludzi. Dlatego też ze względu na charakter implementowanej aplikacji, która miałaby służyć na potrzeby zarządzaniu grafikami w firmie, czyli miejscu zatrudnienia, wybrano kolorystykę w odcieniach zieleni. Badania pokazują, że ten kolor najczęściej kojarzony jest z naturą, spokojem i równowagą. Zastosowano tą kolorystykę, aby wykreować przyjazne i relaksujące środowisko dla użytkowników, co mogłoby mieć przełożenie, na późniejszą efektywności pracowników. Kolorystyka ta została konsekwentnie zastosowana w każdym elemencie aplikacji, zarówno na ekranie logowania, jak i podczas korzystania z systemu [15]. Przy projektowaniu interfejsu użytkownika postawiono również na prostotę użytkowania, aby zarówno pracownik o większej, jak i mniejszej wiedzy technicznej był w stanie używać systemu. Całość została zaprojektowana tak, aby każdy element był dla użytkownika odpowiednio wyróżniony, przejrzysty oraz dokładnie opisany.

5.3. Interfejs użytkownika

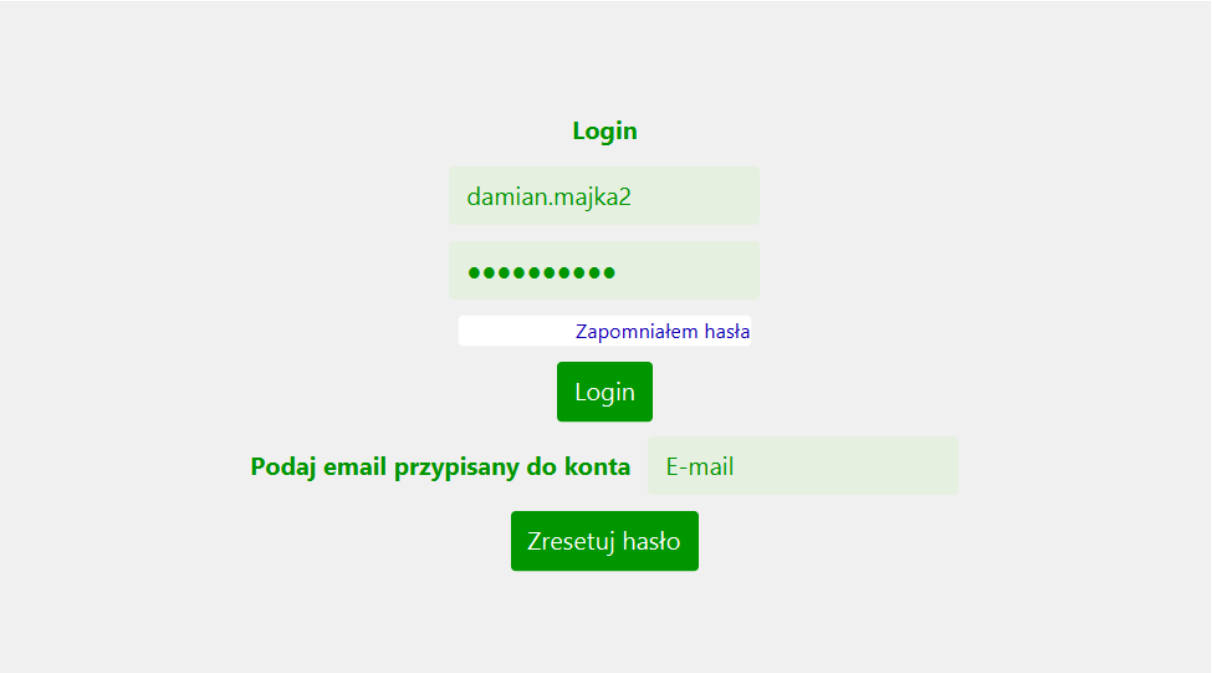
Na podstawie wszystkich wymagań został zaprojektowany zaprezentowany w niniejszym rozdziale interfejs użytkownika. Poruszanie się po interfejsie zostało zrealizowane w intuicyjne, opisane zakładki, z których każda odpowiada za jedną funkcjonalność. Każdy z aktorów posiada różne funkcjonalności, dlatego też został przygotowany osobny interfejs dla osoby o roli użytkownika, managera, czy też admina. Pierwsza przedstawia użytkownika o roli użytkownik (zwykły pracownik), który jest najmniej rozbudowany, ze względu na mniejszą ilość funkcjonalności.

Na rysunku 5.1 przedstawiono wygląd ekranu logowania, na którym użytkownik ma możliwość uzupełniania formularza, dotyczącego ekranu logowania, bądź po wpisaniu błędnego hasła uruchomienia opcji „Zapomniałem hasła”.



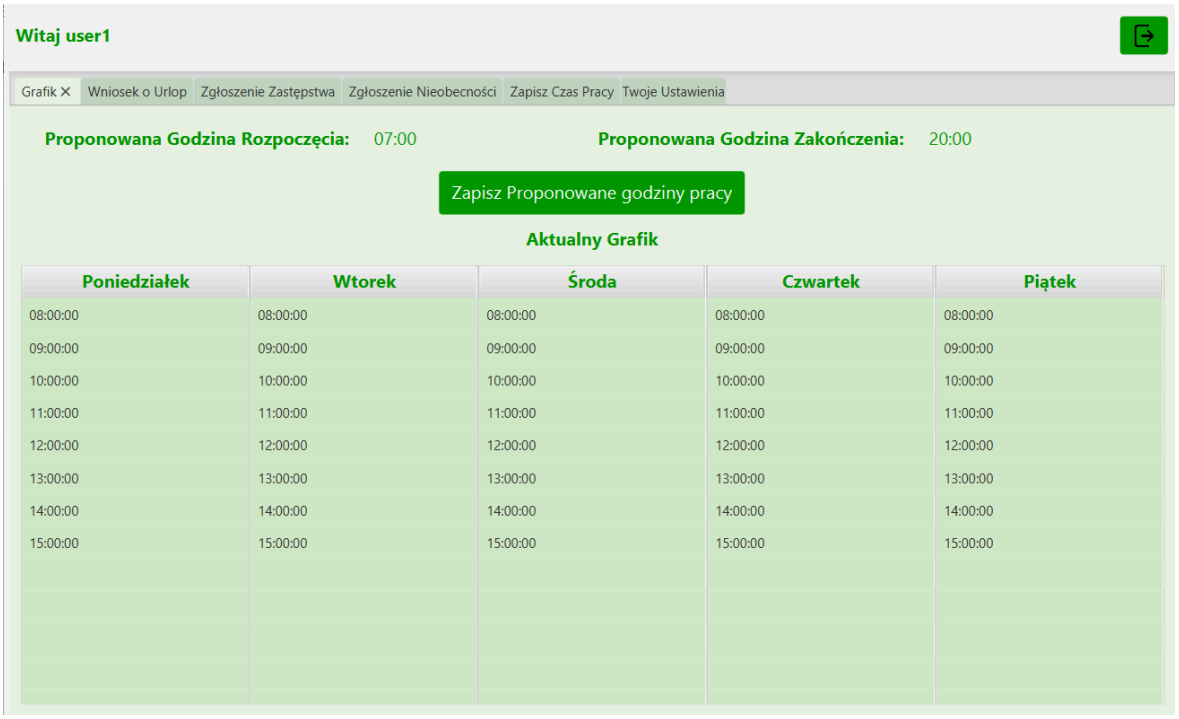
Rys. 5.1. Ekran logowania [Opracowanie własne]

W przypadku błędnie wpisanego hasła pojawia się przycisk „Zapomniałem hasła”, po naciśnięciu którego, pojawia się pole do wpisania adres E-mail oraz przycisk „Zresetuj hasło” (rysunek 5.2)



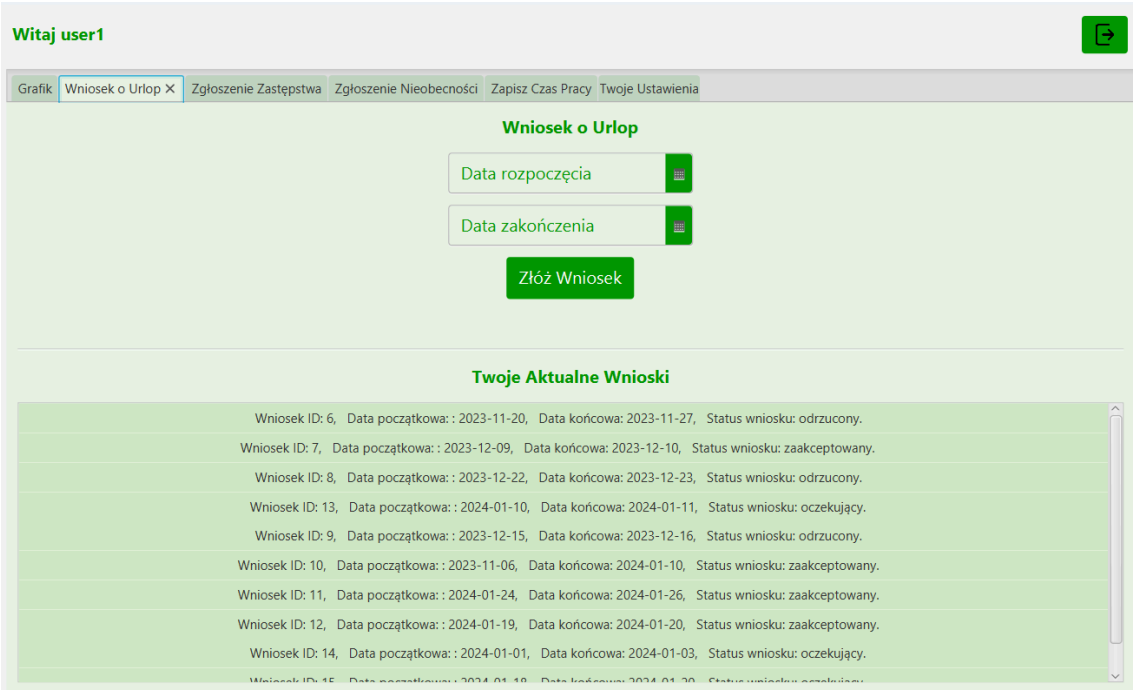
Rys. 5.2. Ekran logowania - przypomnienie hasła [Opracowanie własne]

Na rysunku 5.3. przedstawiono wygląd dostępny dla użytkownika po poprawnym zalogowaniu się. Zdecydowano się, aby po zalogowaniu wyświetlała się funkcjonalność, do której najczęściej zostanie używana aplikacja, czym jest podgląd aktualnego grafiku czasu pracy na najbliższy tydzień. W lewym górnym rogu, wyświetlana jest nazwa użytkownika, w tym wypadku użytko użytkownika testowego o nazwie „user1”. W lewym rogu utworzono przycisk dotyczący wylogowania, na którym umieszczono stosowną ikonę. Poniżej znajdują się zakładki opisane jaką funkcjonalność mają pracować. W centralnej części znajduje się komunikat odnośnie aktualnego grafiku oraz tabela, na której zaznaczony zostaje grafik zalogowanego pracownika w przypadku, gdy ten został już utworzony. Ponadto, na tym ekranie użytkownik ma również możliwość uzupełnienia swoich proponowanych godzin pracy.



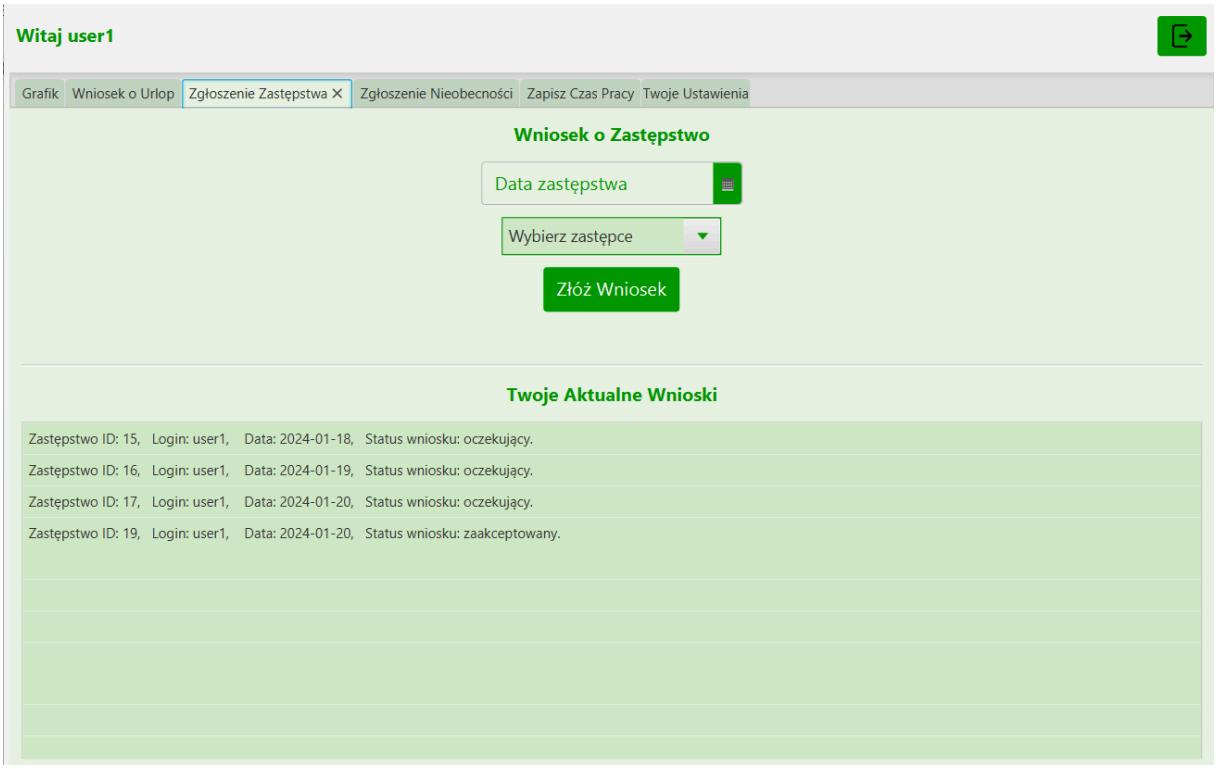
Rys. 5.3. Interfejs podglądu grafiku czasu pracy [Opracowanie własne]

Na rysunku 5.4. Przedstawiono kolejną z zakładek dla osobnej funkcjonalności, jaką jest możliwość złożenia przez użytkownika wniosku urlopowego. Użytkownik składa wniosek poprzez uzupełnienia daty startu oraz daty zakończenia planowanego urlopu, a także poniżej ma możliwość zobaczenia statusu jego aktualnych oraz historycznych wniosków.



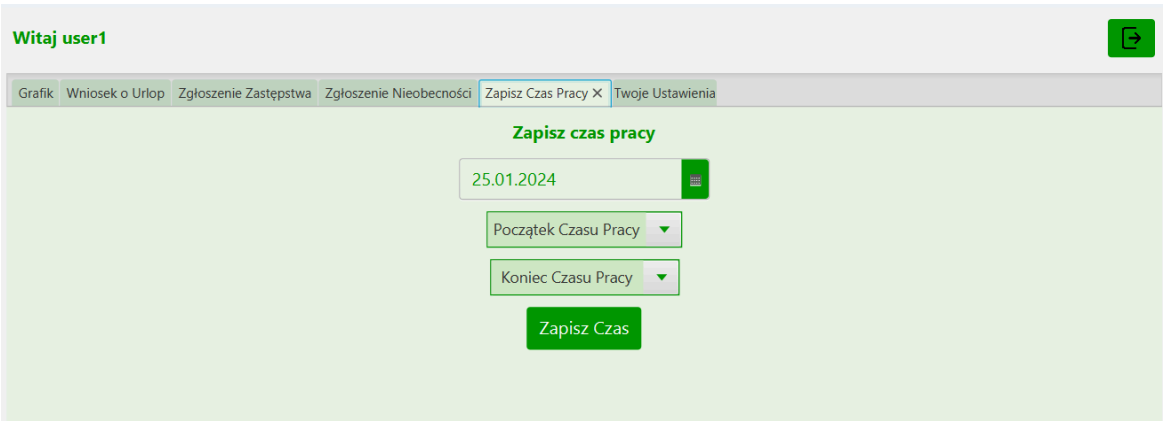
Rys. 5.4. Interfejs wniosku urlopowego [Opracowanie własne]

Interfejs dla zakładki „Zgłoszenie Zastępstwa” jest bardzo zbliżony do tego z zakładki wniosku urlopowego. Zastępstwo określane jest jako jednodniowa nieobecność, dlatego też, użytkownik wybiera jedynie datę zastępstwa. Posiada jedynie możliwość wyboru dnia zastępstwa oraz wyboru z listy osoby z tej samej grupy co użytkownik, który jest aktualnie zalogowany (rysunek 5.5).



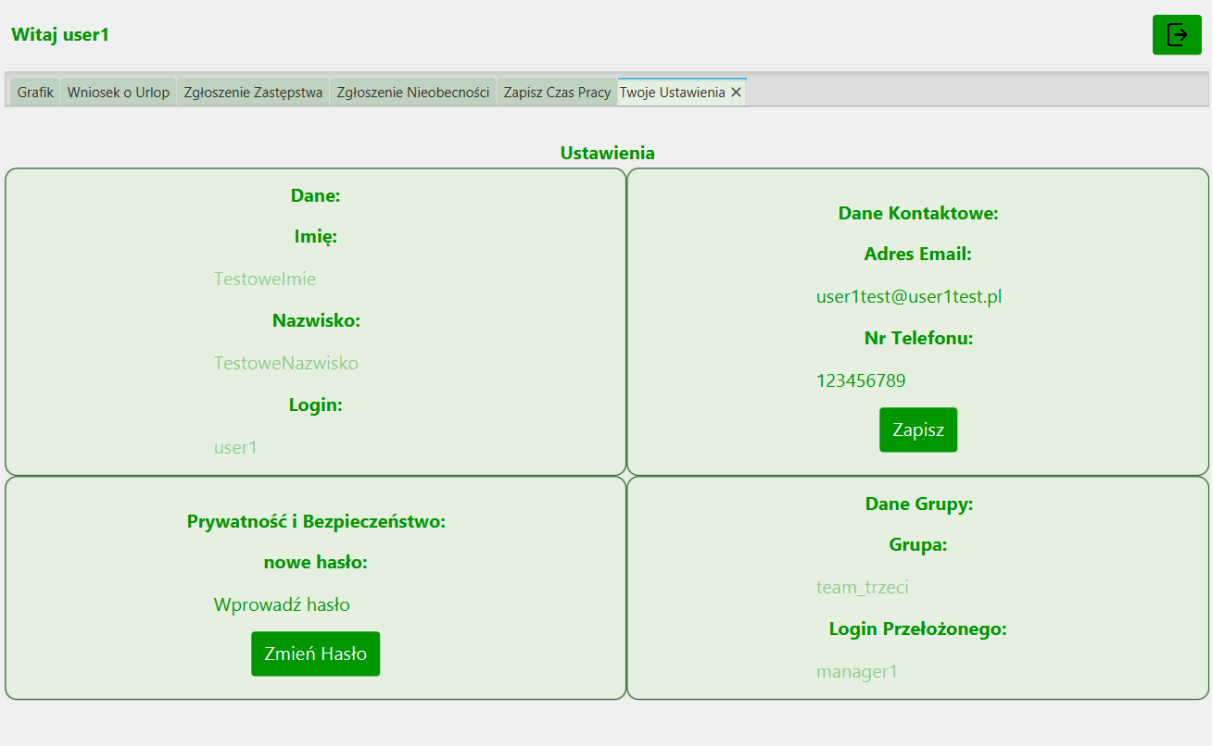
Rys. 5.5. Interfejs zakładki zgłoszenia zastępstwa [Opracowanie własne]

Następna w kolejności jest zakładka dotycząca czasu pracy, gdzie użytkownik ma możliwość rejestrowania czasu pracy w określonym przedziale (rysunek 5.6).



Rys. 5.6 Interfejs zapisu czasu pracy [Opracowanie własne]

Na rysunku 5.7. została przedstawiona zakładka ustawienia dla użytkownika. Zakładka ta została podzielona na 4 sekcje, każda przedstawiająca osobną część ustawień i informacji o użytkowniku. Pola, które pełnią funkcję informacyjną zostały wyłączone i delikatnie wyszarzone, w celu podkreślenia braku możliwości ich modyfikacji.



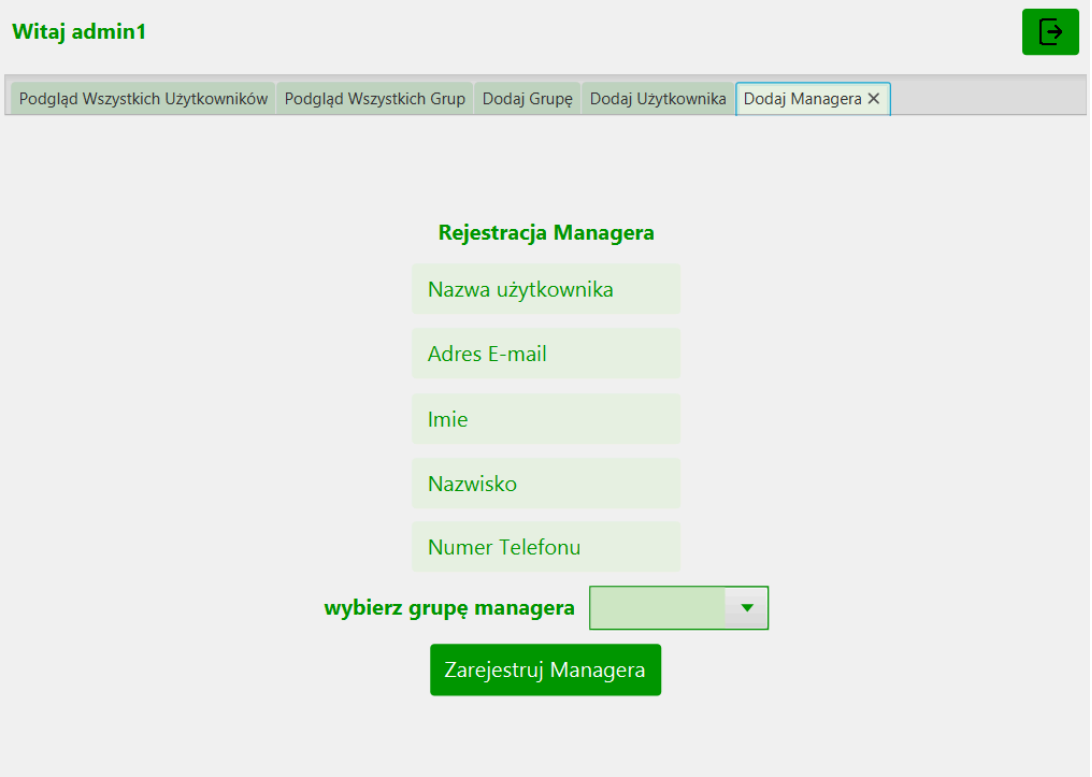
Rys. 5.7. Interfejs ustawień [Opracowanie własne]

Drugim w kolejności jest interfejs dla admina, który posiada funkcje głównie administracyjne takie jak tworzenie grup, czy też użytkowników oraz przypisywanie użytkowników do grup. Przedstawione funkcjonalności są opisane za pomocą zakładek, jak na rysunku 5.8.

Rys. 5.8. Zakładki dla interfejsu admina [Opracowanie własne]

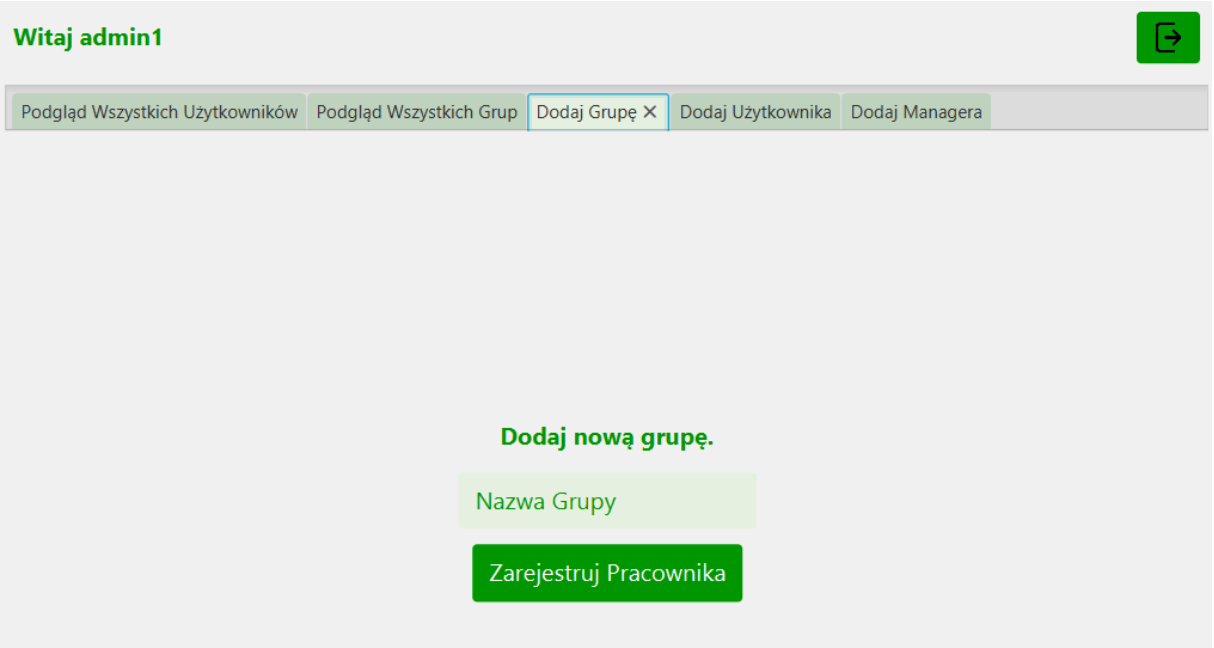
Admin jest jedyną osobą, która ma możliwość dodawania użytkowników i managerów. W celu zarejestrowania nowego managera, admin uruchamia zakładkę „Dodaj Managera", gdzie kolejno uzupełnia nazwę użytkownika, adres e-mail, imię, nazwisko oraz numer telefonu tej osoby. Również wybiera z tego poziomu od razu grupę, którą zarządzać

będzie dany manager. Dokładny wygląd zakładki „Dodaj Managera” został zaprezentowany na rysunku 5.9. Podobnie wygląda procedura dodawania nowego użytkownika o roli pracownik, jedyną różnicą jaka istnieje to różne etykiety takie jak „Wybierz grupę użytkownika”, czy też „Zarejestruj użytkownika”.



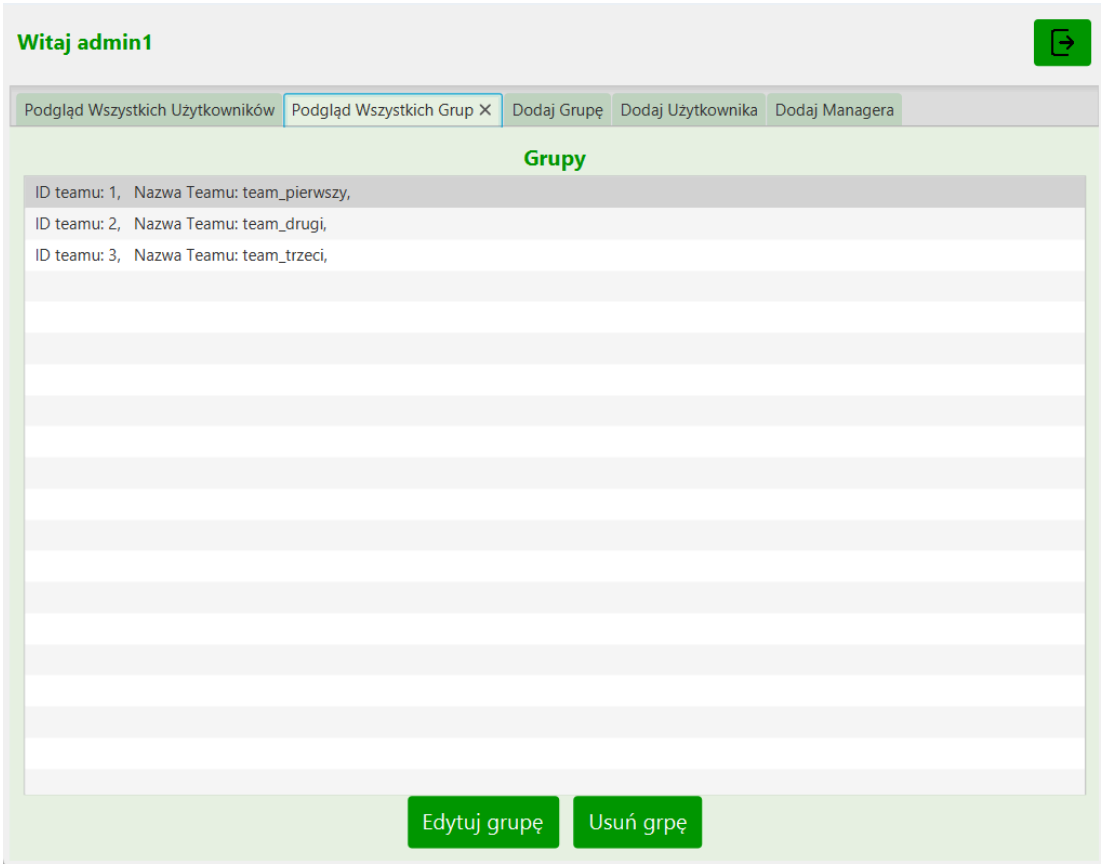
Rys. 5.9 Rejestracja nowego managera. [Opracowanie własne]

Dla zaprezentowanej funkcjonalności (Rysunek 5.9) konieczny jest wybór grupy. Została dodana osobna zakładka zatytułowana „Dodaj Grupę”. Jest ona jedną z najprostszych możliwych zakładek, gdyż składa się z jedynie tytułu, pola do wprowadzenia nazwy grupy oraz przycisku realizującego utworzenie grupy. Dokładny jej wygląd zaprezentowano na rysunku 5.10.



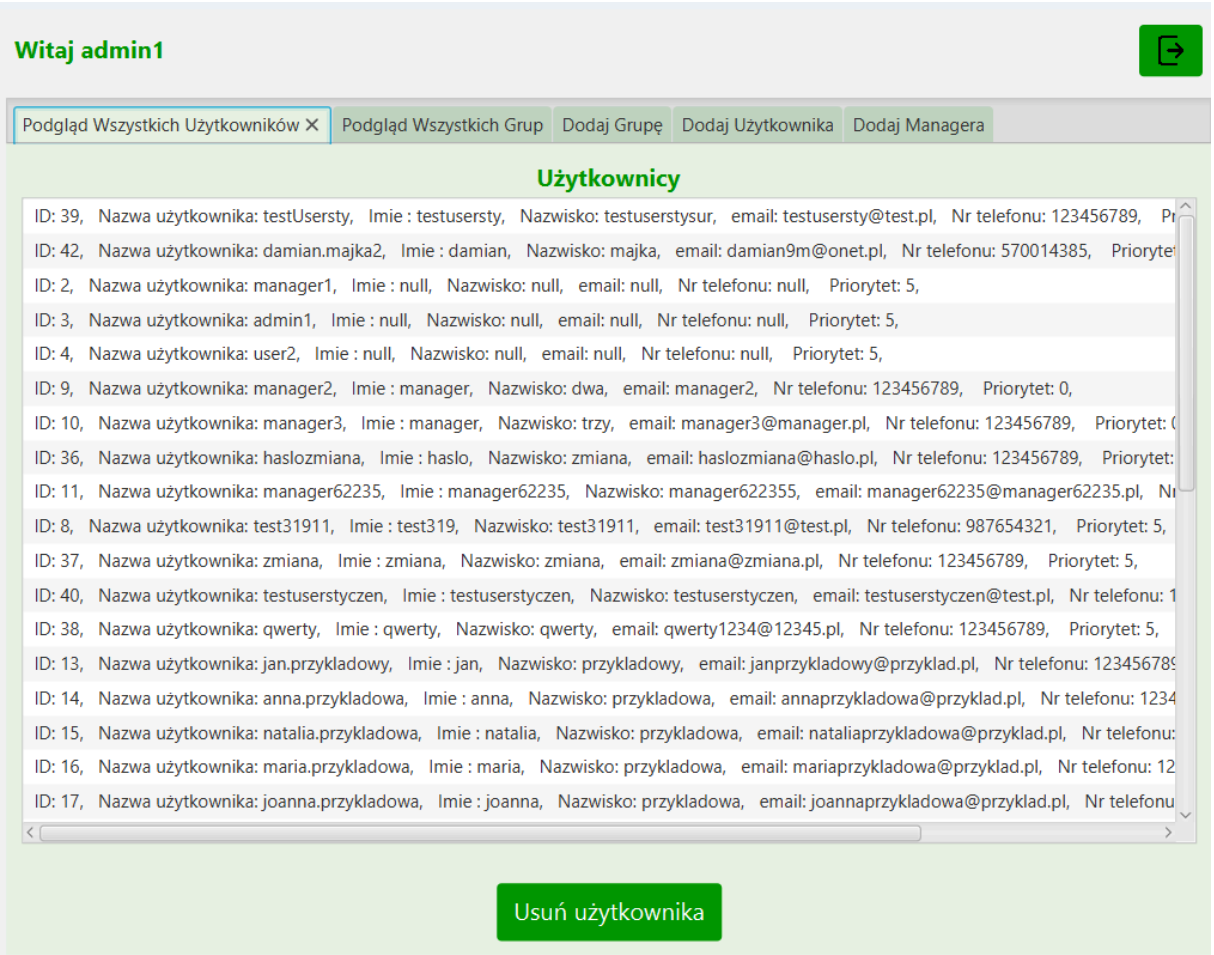
Rys. 5.10. Interfejs dodawania nowej grupy [Opracowanie własne]

Admin ma również możliwość edytowania grupy, poprzez zakładkę „Podgląd Wszystkich Grup" wyświetla się na niej lista wszystkich grup oraz na podstawie wybranej aktualnie grupy istnieje możliwość jej edytowania lub usunięcia (rysunek 5.11).



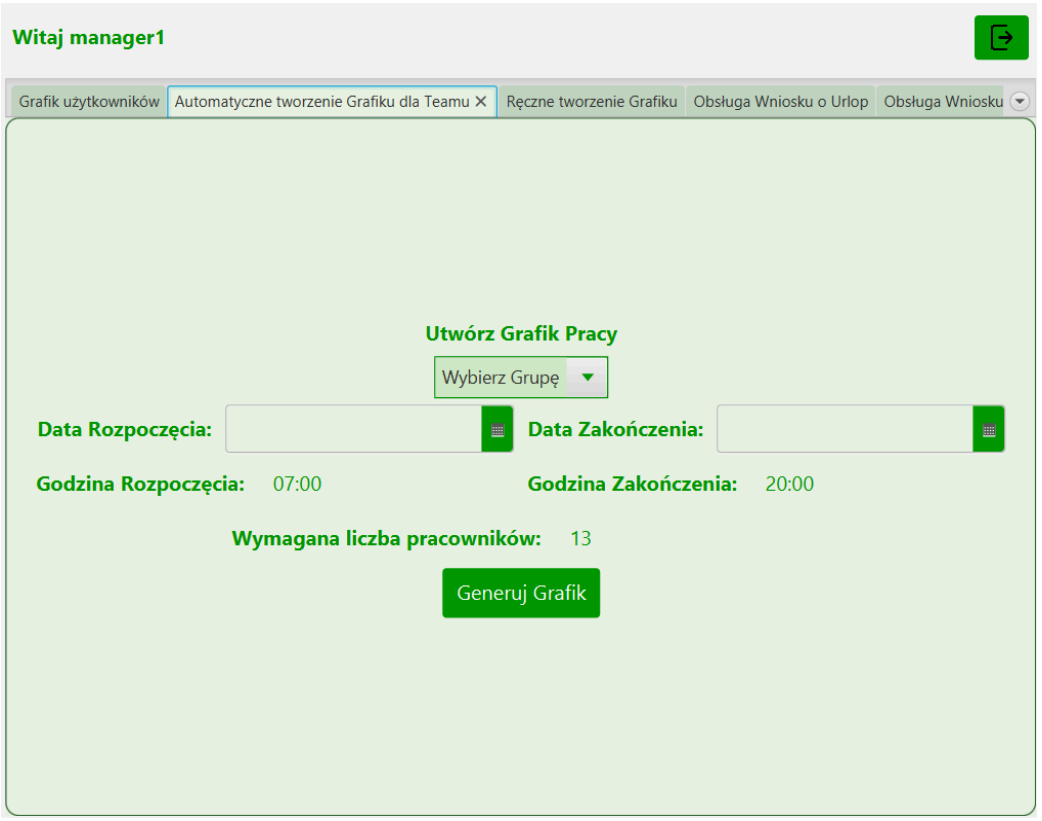
Rys. 5.11 Interfejs podglądu wszystkich grup [Opracowanie własne]

Analogicznie do poprzedniej zakładki, została utworzona zakładka „Podgląd Wszystkich Użytkowników”, w której admin ma możliwość zarządzania wszystkimi użytkownikami, zarówno tymi o roli managera, jak i tymi o roli użytkownika.



Rys. 5.12 Interfejs podglądu wszystkich użytkowników [Opracowanie własne]

Użytkownik o roli managera posiada uprawnienia służące do zarządzania użytkownikami i ich harmonogramem pracy, jak i weryfikacji wniosków, czy też czasu pracy. Główną z nich jest automatyczne tworzenie grafiku dla pracowników na podstawie określonych parametrów oraz określonych wcześniej priorytetów, mających na celu ustalenie kolejności, który pracownik w pierwszej kolejności otrzyma przypisane godziny. Wygląd interfejsu odnośnie tej zakładki został zaprezentowany na rysunku 5.13.

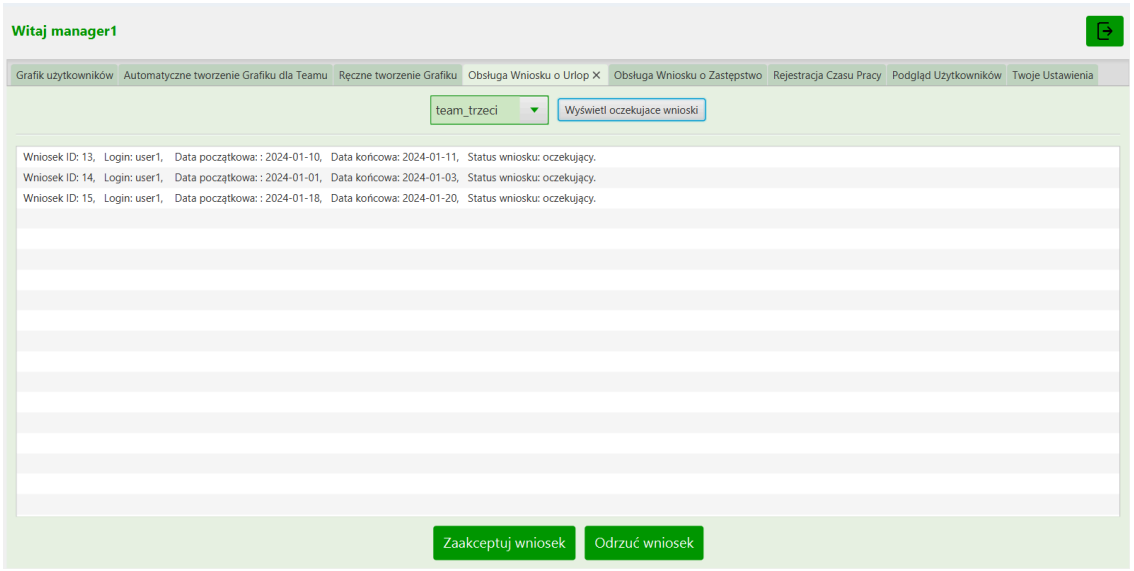


Rys. 5.13 Interfejs Tworzenie grafiku czasu pracy dla pracowników [Opracowanie własne]

Drugą wersją tworzenia grafiku jest ręczne jego tworzenie, przy użyciu zakładki do tego przeznaczonej. Manager wybiera jednego spośród wszystkich użytkowników, po czym wyświetlane są dane użytkownika takie jak priorytet oraz proponowane przez pracownika godziny pracy (Rysunek 5.14).

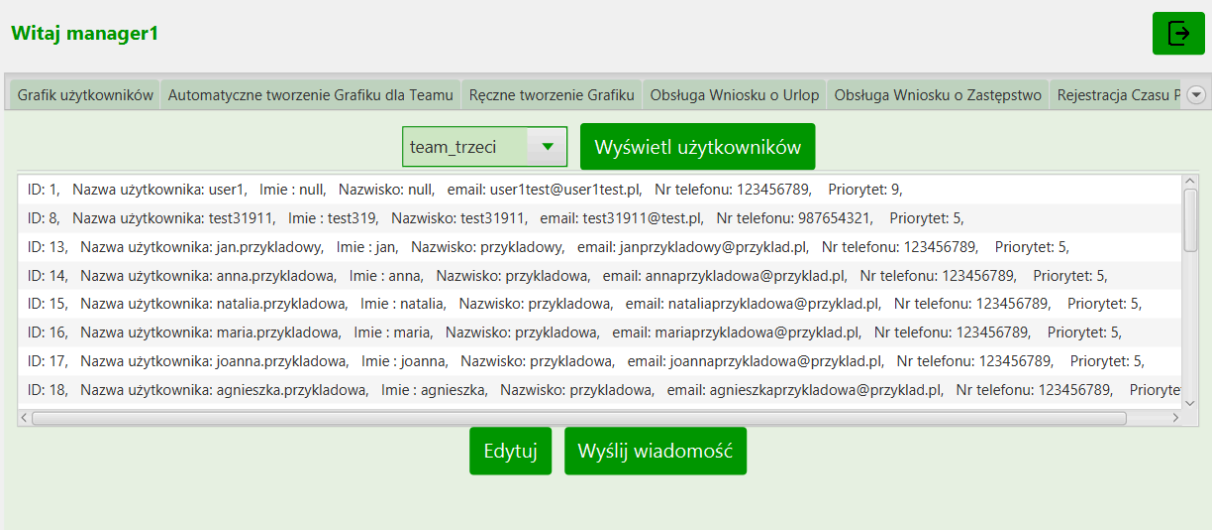
Rys. 5.14 Interfejs ręcznego tworzenia grafiku czasu pracy [Opracowanie własne]

Manager ma możliwość obsługiwania wniosków o urlop oraz wniosków o zastępstwo. Te funkcjonalności zostały zaprojektowane w odpowiadającym im zakładkom. Manager powinien w pierwszej kolejności wybrać grupę, dla której dany manager chce zweryfikować wnioski urlopowe, czy też zgłoszenia zastępstw.



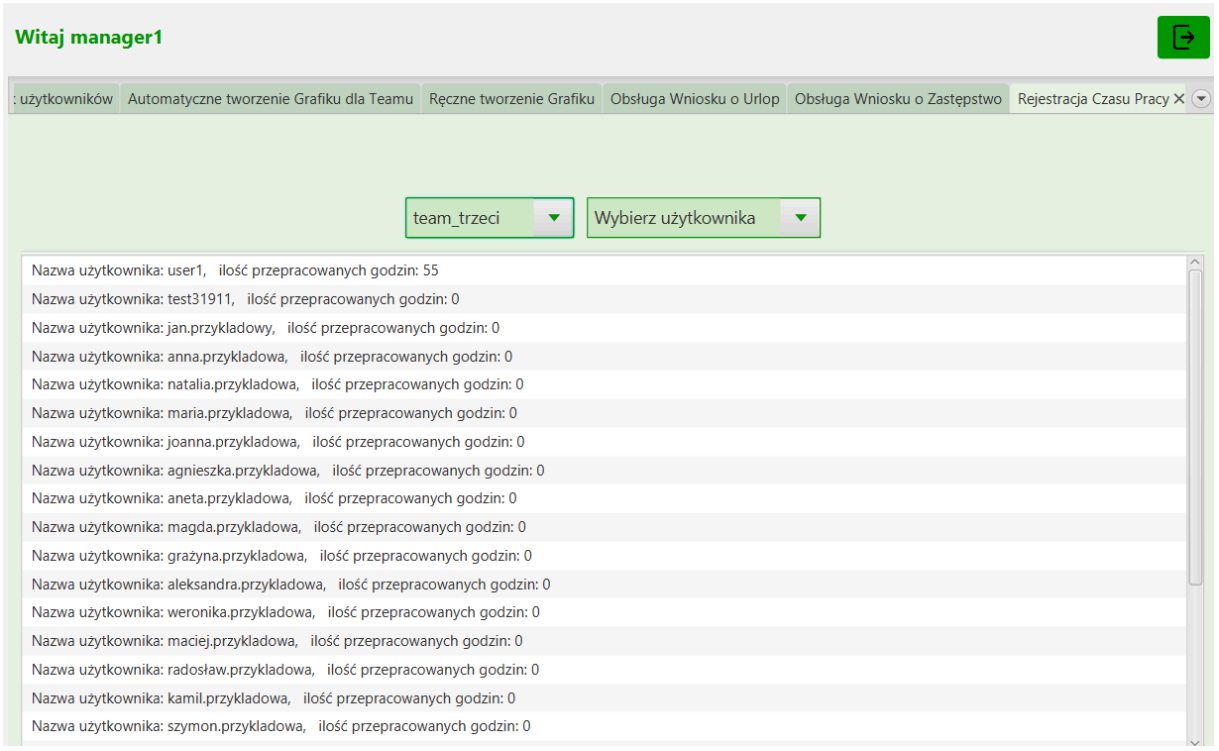
Rys. 5.15 Interfejs obsługi wniosków urlopowych [Opracowanie własne]

Użytkownik o roli manager posiada również możliwość podglądu i edycji wszystkich użytkowników z grup, do których jest przypisany jako przełożony:

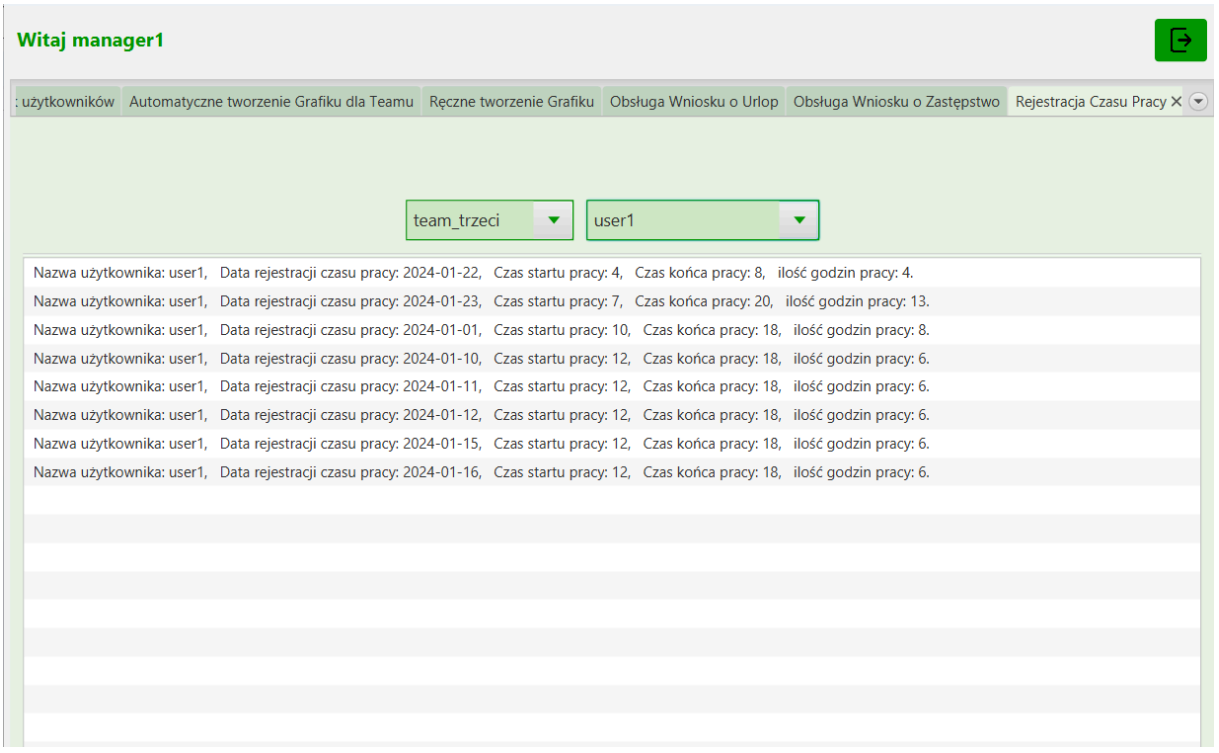


Rys. 5.16 Interfejs podglądu użytkowników [Opracowanie własne]

Menadżer ma również możliwość weryfikacji czasu pracy zarejestrowanego przez pracowników. Może wyświetlić łączną ilość godzin wszystkich użytkowników (rysunek 5.17) lub szczegółowe informacje na temat każdorazowej rejestracji czasu pracy wybranego pracownika (rysunek 5.18).



Rys. 5.17 Interfejs podglądu wszystkich godzin pracy [Opracowanie własne]



Rys. 5.18 Interfejs podglądu szczegółowych informacji dotyczących godzin pracy [Opracowanie własne]

6. Testy

W niniejszym rozdziale zostaną opisane aspekty testowania oprogramowania, obejmujące między innymi wprowadzenie do testów, czyli określenie czym są. Przedstawione zostaną cele takiego procesu. Zostaną również przedstawione różne podziały testów oraz ich przykłady.

6.1. Czym są testy ?

Testowanie jest kluczowym elementem pracy przy implementacji każdego oprogramowania, systemu, aplikacji mobilnej, desktopowej, czy też webowej. Stosuje się je celem weryfikacji istnienia potencjalnych błędów, niedociągnięć oraz każdego z problemów w działaniu, który uniemożliwiłby prawidłowe funkcjonowanie aplikacji. Jest to jeden z najważniejszych etapów przed wydaniem oprogramowania na rynek, ponieważ sprawdza takie etapy, jak działanie każdej z funkcjonalności, wydajność aplikacji, szybkość, przyjazność UI/UX, bezpieczeństwo danych oraz wiele innych.

Testowanie w dużej ilości idące w parze z prawidłowo i kompleksowo wykonanymi testami są szczególnie ważne, ponieważ w przypadku nieprawidłowego przetestowania danego oprogramowania, jedną z najbardziej łagodnych rzeczy jakie mogą się wydarzyć to nie wyświetlająca się ikona po kliknięciu w daną zakładkę. Konsekwencje braku testów mogą być dużo poważniejsze. W historii odnotowane zostało wiele wycieków danych, spowodowanych niedopatrzeniem w zakresie bezpieczeństwa, które prawdopodobnie zostałoby wykryte, gdyby dane oprogramowanie zostało prawidłowo przetestowane.
W historii zdarzały się również poważniejsze konsekwencje braku poprawnego testowania. Jednym z przykładów, gdzie niepoprawne przetestowanie oprogramowania mogło doprowadzić do poważniejszych konsekwencji była awaria oprogramowania poduszek powietrznych zamontowanych w przeszłości w samochodach marki Nissan. Ta nieprzetestowana usterka, mogła nieść za sobą konsekwencje nawet w postaci obrażeń cielesnych [11].

Aby testy mogły być prawidłowo przeprowadzone należy dotrzymywać standardów określonych przez ISTQB czyli *International Software Testing Qualifications Board.* Dokładne wytyczne odnośnie przeprowadzanych testów zostały opisane w standardzie IEEE Std 829-2008 [12].

6.2. Rodzaje testów

Wyróżnia się kilka rodzajów testów oprogramowania, natomiast główny podział jest pomiędzy testami manualnymi i automatycznymi.

Rys. 6.1 Podział testowania oprogramowania [Opracowanie własne]

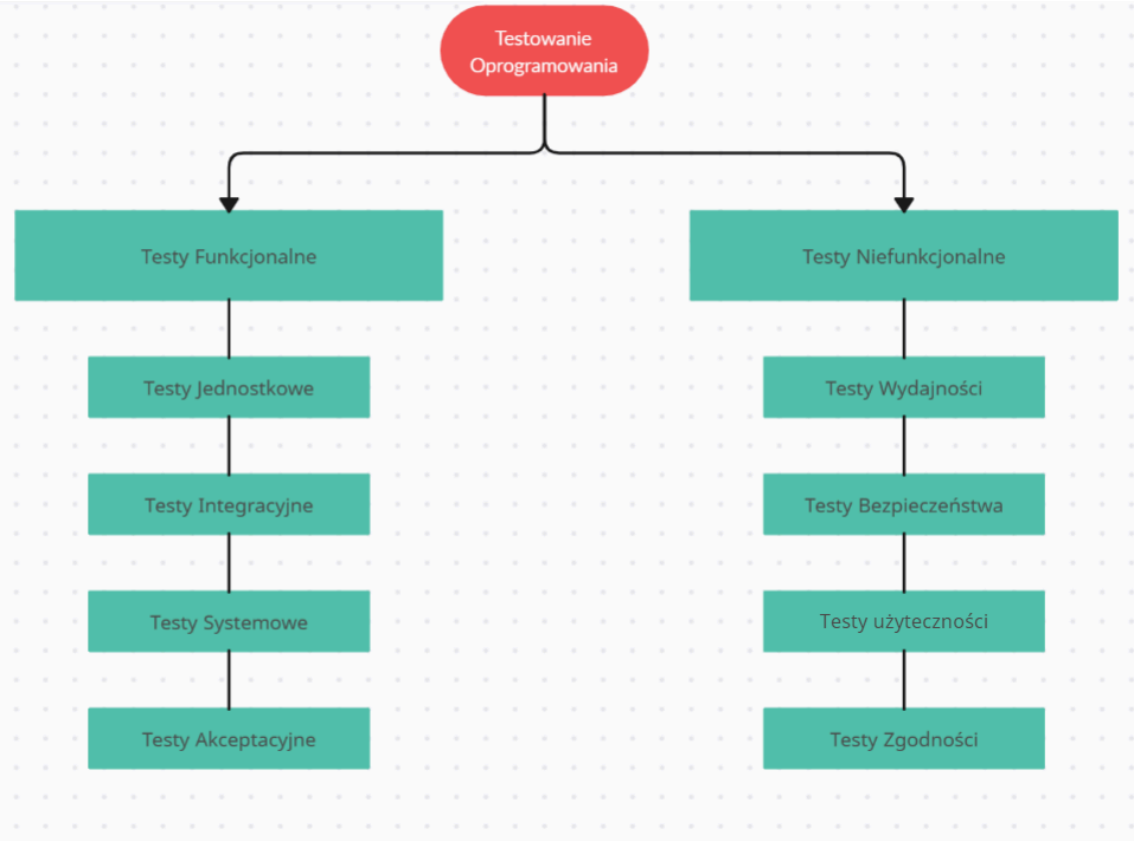
Testy manualne są szczególnie ważne, ponieważ, pozwalają określić:

- Czy UI/UX jest przejrzyste i przyjazne dla użytkownika?
- Czy użytkownik może nacisnąć każdy z elementów przeznaczony do takiego użycia?
- Czy grafiki wyświetlają się prawidłowo?
- Czy aplikacja webowa działa prawidłowo na każdej przeglądarce?

Polegają na ręcznym sprawdzaniu danych funkcjonalności, poprzez uruchomienie aplikacji i przeglądnięcie, czy każda wszystkie elementy interfejsu poprawnie się załadowały.

Testy Automatyczne wykonywane są przy pomocy specjalnych narzędzi oraz przygotowanych skryptów, mają za zadanie określenie czy zaimplementowane metody oraz algorytmy działają prawidłowo. Obejmują one zarówno pojedyncze funkcje, jak i całą serię zdarzeń. Mimo to, że ich jakość jest zależna od tego jak dobrze zostaną one napisane, to są bardziej niezawodne niż testy manualne. Dzieje się tak, ponieważ skrypt automatycznie sprawdza, czy dany algorytm lub metoda zwraca określony wynik i w tym zakresie nie ma możliwości pomyłki.

Oprócz takiego rozróżnienia testów istnieje również podział testów ze względu na ich przeznaczenie. Uwzględniając taki podział można przyjąć, że testy dzielimy na testy funkcjonalne oraz testy niefunkcjonalne, jak to przedstawiono na rysunku 6.2.



Rys. 6.2. Testowanie Oprogramowania - podział [Opracowanie własne]

Testy funkcjonalne mają za zadanie sprawdzić, czy zostały spełnione wszystkie wcześniej ustalone kryteria. Zostały poprawnie zaimplementowane wszystkie zadane funkcjonalności. Głównym celem testów funkcjonalnych jest sprawdzenie, czy oprogramowanie działa zgodnie z założonymi wymaganiami funkcjonalnymi. Oceniają one, czy aplikacja spełnia oczekiwania względem tego jak powinna działać.

Testy niefunkcjonalne są to testy skupiające się na jakości oprogramowania. Mają za zadanie sprawdzić, czy oprogramowanie jest wystarczająco szybkie, bezpieczne, dostosowane do używania w różnych środowiskach oraz przez różnych użytkowników. Są testami wykonywanymi najczęściej po testach funkcjonalnych [13].

6.3. Frameworki testowe

Istnieje wiele różnych frameworków testowych. Jednymi z najpopularniejszych frameworków do pisania testów jednostkowych i integracyjnych są:

- JUnit,
- Spock.

JUnit jest to jeden z najpopularniejszych i najstarszych frameworków testowych w języku Java, obecnie występuje w wersji 5 .

Spock jest to framework testowy dla aplikacji pisanych w języku Java, Groovy oraz Kotlin. Obecnie w najnowszej wersji 2.3, Spock korzysta z Groovy opartego na JVM. Do zalet Spock można zaliczyć między innymi dynamiczne typowanie, zwięzłość kodu, czy też w samym procesie pisania testów gotowe etykiety, dla metodyki Given, When, Then.
Niektóre z różnic między tymi frameworkami opisano w tabeli 6.1 [13].

JUnit	Spock
Test definiuje się poprzez dodanie adnotacji @Test do metody.	Test definiuje się przy użyciu słowa kluczowego **def** oraz nazwy testu na przykład: `def "nazwa metody testującej"()`
Porównanie odbywa się za pomocą asercji (assertEqual)	Porównanie odbywa się za pomocą wyrażenia boolean.
Brak etykiet Given | When | Then	Zaimplementowane etykiety Given | When | Then.
Testy parametryzowane tworzone za pomocą adnotacji @ParameterizedTest	Testy parametryzowane tworzone z użyciem tabeli gdzie dane parametry oddzielone są znakiem “|”.

Tabela 6.1 Różnice między JUnit i Spock [opracowanie własne]

6.4. Testy manualne

Testy manualne wydają się najmniej obszerne, natomiast nie każdy element można zweryfikować automatycznie na podstawie kodu, czy też pisząc skrypt. Człowiek może okazać się potrzebny przy testowaniu UI/UX. System nie zweryfikuje i nie zdecyduje za człowieka, czy dany element prezentuje się w odpowiedni sposób, czy jest on odpowiednio wyrównany, aby wyglądał odpowiednio. Dlatego też, w testowaniu bierze udział również człowiek, który weryfikuje pewne elementy ręcznie. W celu dokonania takich testów przygotowuje się wcześniej scenariusze testowe. Scenariusze testowe mogą przyjmować

większy zakres czynności do wykonania lub też przykładem takiego scenariuszu może być zapis czynności koniecznych do wykonania przez testera. Oto praktyczny przykład:

Test uzupełnienia formularza do założenia konta dla managera:

- Uruchomienie formularza rejestracji,
- Wpisanie wymaganych danych - błędne uzupełnienie,
- Naciśnięcie przycisku „Zarejestruj Managera”,
- Obserwacja, czy pojawił się komunikat o błędnych danych,
- Obserwacja, czy komunikat jest wystarczająco widoczny i odpowiednio wskazuje, w którym miejscu dokonano błędu,
- Poprawa danych,
- Naciśnięcie przycisku „Zarejestruj Managera”,
- Obserwacja, czy pojawił się komunikat odnośnie poprawnej rejestracji,
- Weryfikacja, czy na adres mailowy podany w rejestracji zostało dostarczone jednorazowe hasło użytkownika.

6.5. Testy jednostkowe

Testy jednostkowe są to testy polegające na testowaniu poszczególnej jednej funkcjonalności, metody lub klasy. Celem tych testów jest zapewnienie, że fragment kodu lub dana metoda, działa prawidłowo niezależnie od innych części oprogramowania. Przykładem takiego testu zaimplementowanego w aplikacji jest test klasy „EmailValidator”:

```
@Test
    public void testValidEmail() {
        assertTrue(validator.isValid("jan.kowalski@example.com"));
    }

    @Test
    public void testInvalidEmail() {
        assertFalse(validator.isValid("niepoprawny_adres_email"));
    }
    @Test
    public void testInvalidEmptyEmail() {
        assertFalse(validator.isValid(""));
    }
```

Listing 6.1 Test walidacji adresu mailowego [Opracowanie własne]

W fragmencie kodu przedstawionym na listingu 6.1. wykonano 3 testy jednostkowe dla klasy „EmailValidator”, w pierwszym podano prawidłowy adres e-mail oraz oczekiwano, że metoda „isValid” zwróci „True”, taki efekt też uzyskano. W kolejnych dwóch podano nieprawidłowy adres e-mail oraz pusty ciąg znaków. Dla tych dwóch testów oczekiwano, że metoda „isValid” zwróci „False”, również uzyskano wskazany efekt.

6.6. Testy integracyjne

Testy integracyjne są to testy sprawdzające większy zakres funkcjonalności niż testy jednostkowe. Testy te skupiają się na poprawności weryfikacji, czy dane komponenty, czy też moduły odpowiednio ze sobą współpracują. Są one większe, bardziej obszerne niż testy jednostkowe. Przykład zastosowania testu integracyjnego został przedstawiony w listingu 6.2.

```
@Test
    public void testAddUserToTeam() {
        // Given
        Long userId = 1L;
        Long teamId = 2L;
        User user = new User();
        Team team = new Team();
        userRepository.save(user);
        teamRepository.save(team);

        // When
        userTeamAssignmentService.addUserToTeam(userId, teamId);

        //THEN
        Optional<User> foundUser = userRepository.findById(userId);
        assertTrue(foundUser.isPresent());
        Optional<Team> foundTeam = teamRepository.findById(teamId);
        assertTrue(foundTeam.isPresent());
        List<UserTeamAssignment> userTeamAssignments =
              userTeamAssignmentRepository.findAll();
        assertEquals(1, userTeamAssignments.size());
    }
```

Listing 6.2 Test integracyjny [Opracowanie własne]

Przeprowadzony test integracyjny w przypadku poprawnej implementacji danych metod powinien potwierdzić, że opcja dodawania użytkownika do zespołu działa poprawnie. Taki test ma za zadanie weryfikację, czy poszczególne komponenty systemu współpracują ze sobą w prawidłowy sposób.

6.7. Implementacja testów w projekcie.

W Projekcie zostały zaimplementowane testy, aby zweryfikować poprawność działania odpowiednich funkcjonalności.

6.7.1. Test dodawania proponowanych godzin pracy

```
@Test
    public void updateProposedWorkTime_Success() {
        Authentication authentication = mock(UsernamePasswordAuthenticationToken.class);
        SecurityContext securityContext = mock(SecurityContext.class);
        when(securityContext.getAuthentication()).thenReturn(authentication);
        when(authentication.getName()).thenReturn("testUser");
        SecurityContextHolder.setContext(securityContext);

        Time startTime = Time.valueOf("08:00:00");
        Time endTime = Time.valueOf("17:00:00");
        User user = new User();
        user.setId(1L);
        user.setUsername("testUser");
        when(mockUserRepository.findByUsername("testUser")).thenReturn(user);
        when(mockUserRepository.findById(1L)).thenReturn(Optional.of(user));
        when(mockUserRepository.save(any(User.class))).thenReturn(user);
        Boolean result = userService.updateProposedWorkTime(startTime, endTime);
        assertTrue(result);
        assertEquals(startTime, user.getProposedWorkTimeStart());
        assertEquals(endTime, user.getProposedWorkTimeEnd());
        verify(mockUserRepository, times(1)).save(user);
        SecurityContextHolder.clearContext();
    }
```

Listing 6.3 Test dodawania proponowanych godzin pracy [Opracowanie własne]

W przedstawionym na Listingu 6.3 teście została przetestowana funkcjonalność dodawania proponowanych godzin pracy. Sprawdzane jest, czy metoda odpowiadająca za aktualizacje proponowanych godzin pracy aktualnie zalogowanego pracownika, poprawnie zwraca wartość „True” oraz czy godziny są prawidłowo przypisywane.

6.7.2. Test aktualizacji danych użytkownika.

W przedstawionym na listingu 6.4 teście została przetestowana funkcjonalność aktualizacji danych kontaktowych użytkownika. Na początku zostały ustawione dane wejściowe oraz symulacja zachowania metody „updateUserDetails”. Następnie została ona wywołana z odpowiednimi argumentami. Za pomocą metody „verify” (z pomocą frameworka Mockito) zweryfikowano, czy metoda na pewno wywołała się z odpowiednimi argumentami.

Jest to kluczowy krok weryfikujący, czy serwis rzeczywiście deleguje wykonanie operacji do repozytorium z odpowiednimi danymi.

```
@Test
   public void testUpdateUserDetails() {
       Long userId = 1L;
       String email = "test@example.com";
       Integer phoneNumber = 123456789;
       int priority = 5;
       doNothing().when(mockUserRepository)
                  .updateUserDetails(userId, email, phoneNumber, priority);

       userService.updateUserDetails(userId, email, phoneNumber, priority);
       verify(mockUserRepository)
             .updateUserDetails(userId, email, phoneNumber, priority);
   }
```

Listing 6.4 test aktualizacji danych użytkownika[Opracowanie własne]

6.7.3. Test dodawania czasu pracy.

```
@Test
    public void addTimeRegistrationTest() {
        Authentication authentication = mock(Authentication.class);
        SecurityContext securityContext = mock(SecurityContext.class);
        when(securityContext.getAuthentication()).thenReturn(authentication);
        when(authentication.getName()).thenReturn("testUser");
        SecurityContextHolder.setContext(securityContext);
        LocalDate date = LocalDate.now().minusDays(1);
        Integer startTime = 9;
        Integer endTime = 17;
        LocalDate date2 = LocalDate.now().minusDays(1);
        Integer startTime2 = 17;
        Integer endTime2 = 9;
        User loggedInUser = new User();
        loggedInUser.setId(1L);
        loggedInUser.setUsername("testUser");
        when(mockUserRepository.findByUsername("testUser"))
                               .thenReturn(loggedInUser);

        Boolean result = timeRegistrationService
               .addTimeRegistration(date, startTime, endTime);
        Boolean result2 = timeRegistrationService
               .addTimeRegistration(date2, startTime2, endTime2);

        assertTrue(result);
        assertFalse(result2);
        verify(mockTimeRegistrationRepository,
               times(1)).save(any(TimeRegistration.class));
    }
```

Listing 6.5 test dodawania godzin pracy [Opracowanie własne]

W przedstawionym na listingu 6.5 teście dodawania godzin pracy zostało przetestowane sprawdzanie poprawności danych zapisywanych przy dodawaniu czasu pracy. W tym teście został zweryfikowany przypadek dodania prawidłowych, jak i błędnych danych. Przy podanych prawidłowych danych funkcja powinna zwrócić wartość „True", w przypadku podanych błędnych danych wartość „False".

6.7.4. Test pobierania wszystkich grup.

Listing 6.6 prezentuje test pobierania wszystkich grup, w celu ich późniejszego wyświetlania. Sprawdzane jest, czy lista grup pobierana z bazy danych jest zgodna z listą grup wcześniej dodaną.

```
@Test
    public void getAllTeam_ReturnsAllTeams() {
        List<Team> expectedTeams = new ArrayList<>();
        expectedTeams.add(new Team());
        expectedTeams.add(new Team());

        when(mockTeamRepository.findAll()).thenReturn(expectedTeams);

        List<Team> teams = teamService.getAllTeam();

        assertEquals(expectedTeams, teams);
        verify(mockTeamRepository, times(1)).findAll();
    }
```

Listing 6.6 test pobierania wszystkich Grup [Opracowanie własne]

6.7.7. Test zgłaszania urlopu.

Test przedstawiony na listingu 6.7 pozwala na weryfikację czy poprawnie są pobierane dane dotyczące zalogowanego użytkownika oraz, czy zapisywane dane dotyczące wniosku o urlop są zapisywane. Sprawdza czy po złożeniu wniosku jego status jest odpowiedni oraz czy użytkownik przypisany do wniosku jest zgodny z użytkownikiem aktualnie zalogowanym.

```
@Test
    public void applyForLeave_Success() {
        when(mockSecurityContext.getAuthentication()).thenReturn(mockAuthentication);
        SecurityContextHolder.setContext(mockSecurityContext);
        LeaveApplication application = new LeaveApplication();
        User loggedInUser = new User();
        loggedInUser.setUsername("testUser");

        when(mockAuthentication.getName()).thenReturn("testUser");
        when(mockUserRepository.findByUsername("testUser")).thenReturn(loggedInUser);
        when(mockLeaveApplicationRepository.save(any(LeaveApplication.class)))
                                           .thenReturn(application);

        LeaveApplication result = leaveApplicationService.applyForLeave(application);

        assertEquals("oczekujący", result.getStatus());
        assertEquals(loggedInUser, result.getUser());
        verify(mockLeaveApplicationRepository, times(1)).save(application);
    }
```

Listing 6.7 test składania wniosku o urlop [Opracowanie własne]

6.7.9. Test dodawania grafiku czasu pracy.

W teście przedstawionym na listingu 6.8 testowana jest funkcjonalność ręcznego tworzenia grafiku czasu pracy dla pracownika. Opisany test powinien zweryfikować zapisanie wprowadzonych przez użytkownika danych dotyczących grafiku czasu pracy na dany dzień.

```
@Test
    public void saveScheduleManuall_UserExists_SchedulesSaved() {
        User mockUser = new User();
        mockUser.setUsername("testUser");
        mockUser.setId(1L);
        LocalDate date = LocalDate.now();
        LocalTime startTime = LocalTime.of(9, 0);
        LocalTime endTime = LocalTime.of(10, 0);
        Date startDate = Date.valueOf(date);
        Date endDate = Date.valueOf(date);
        when(userRepository.findByUsername("testUser")).thenReturn(mockUser);
        Boolean result = workScheduleService
            .saveScheduleManuall(startDate, endDate, date, startTime, endTime, username);
        assertTrue(result);
        verify(mockWorkScheduleRepository, times(1)).save(any(WorkSchedule.class));
    }
```

Listing 6.8 test zapisu ręcznie tworzonego grafiku czasu pracy [Opracowanie własne]

7. Podsumowanie i wnioski

Celem pracy dyplomowej było zaprojektowanieu i zaimplementowanie systemu pozwalającego na usprawnienie pracy w firmie w zakresie bardziej efektywnego i szybszego procesu tworzenia grafiku czasu pracy oraz zarządzania pracownikami. Wszystkie z wymagań funkcjonalnych i niefunkcjonalnych zostały spełnione. Podczas projektowaniu zdecydowano, że aplikacja desktopowa będzie najlepsza w pracy o takiej tematyce. Aplikacja została zaprojektowana z myślą o używaniu jej jako wewnętrzny system danej firmy. Umożliwiłoby to instalację i użytek na stacjach roboczych.

Podczas implementacji sprostano trudnościom takim jak implementacja aplikacji, w której odgrywane są aż cztery role, co wymusiło konieczność implementacji osobnych interfejsów dla każdego z nich oraz odpowiednią obsługę autoryzacji i uwierzytelnienia.

Dzięki realizacji projektu wyciągnięto wnioski odnośnie przydatności aplikacji do zarządzania grafikami czasu pracy w działaniu firmy. Przy użyciu takiego systemu, firmy byłyby w stanie poprawić swoją efektywność w zakresie zarządzania pracownikami. Ze względu na charakter wykonanej aplikacji możliwy jest jej dalszy rozwój. Można to zrealizować poprzez:

- zintegrowanie jej z innym, zewnętrznym systemem płacowym,
- udostępnienie aplikacji na większa ilośc platform,
- rozbudowę o narzędzia funkcjonalności pozwalające na analizę i raportowanie.

Omówiona praca umożliwiła udoskonalenie wiedzy z zakresu projektowania i implementacji aplikacji desktopowych. Pozwoliła na poznanie nowych technologii oraz zgłębienie szerszej wiedzy na temat zarządzania pracownikami.

Bibliografia

[1] Rod Johnson, *Expert One-on-One: J2EE Design and Development.*

[2] Urszula Piechota, Jacek Piechota. *JavaFX. Tworzenie graficznych interfejsów użytkownika.*,

[3] Andrzej Zoła. *Programowanie w języku Java*. 2004

[4] Maven - dokumentacja techniczna. [online]

https://maven.apache.org/

[5] PostgreSQL - dokumentacja techniczna [online]

https://www.postgresql.org/about/

[6] Spring – dokumentacja techniczna. [online]

https://docs.spring.io/

[7] Spring Data JPA [online]

https://spring.io/projects/spring-data-jpa

[8] Spring Security [online]

https://spring.io/projects/spring-security

[9] Bartłomiej Mirosław Klimek, Maria Skublewska-Paszkowska, *Porównanie wydajności relacyjnych baz danych PostgreSQL oraz MySQL dla aplikacji desktopowej.* 2021

[10] Java Persistence API [online]

https://docs.oracle.com/javaee/5/tutorial/doc/bnbpz.html

[11] Brak testów nissan [online]

https://www.nissan.pl/dla-wlasciciela/alert-kampanii-naprawczej/takata.html

[12] IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation

[13] Rodzaje testów [online]

https://udigroup.pl/blog/testowanie-oprogramowania-typy-rodzaje-poziomy/

[13] Testy oprogramowania Spock JUnit [online]

https://blog.onwelo.pl/spock-vs-junit/

[14] Malina Witold, Szwoch Mariusz. *Podstawy projektowania interfejsów użytkownika*

[15] Zasady dobierania kolorów [online]
https://imakeable.com/nasz-blog/jak-dobierac-kolory-do-interfejsu-uzytkownika-kolory-w-it

Spis tabel

Spis rysunków

Spis listingów