System Zarządzania Personelem

Dokumentacja i pełny raport z projektu

Inżynieria Oprogramowania

1. Cele i założenia projektu.

Celem projektu było stworzenie serwisu internetowego umożliwiającego zarządzanie personelem firmy zajmującej się prowadzeniem warsztatu samochodowego. Stworzony serwis umożliwia odczyt i modyfikację danych pracowników, zarządzanie zleceniami realizowanymi przez firmę, zarządzanie urlopami zarówno od strony personelu biurowego jak i pracowników.

2. Skład osobowy projektu.

W skład osób realizujących projekt weszli :

- Jakub Baran – kontakt z klientem, redakcja wymagań projektowych, tworzenie diagramów
- Krzysztof Czerwiński – administracja projektu w serwisie Jira, rozwój backednu, bezpieczeństwo
- Kamil Jaśkiewicz – rozwój backendu, planowanie i zarządzanie architekturą projektu, komunikacja między warstwami, administracja repozytorium kodu
- Bartosz Sękowski – testy jednostkowe i integracyjne kodu, testy interfejsu
- Michał Stasiak – zarządzanie realizacją projektu, rozwój bazy danych, redakcja dokumentacji
- Wiktor Woszczyna – rozwój frontendu, projektowanie interfejsów

3. Przebieg pracy.

Zgodnie z wybraną techniką prowadzenia projektów Scrum i Agile, przebieg pracy podzielony został na trzy sprinty :

- Sprint I – zbieranie wymagań, planowanie architektury projektu, podział ról w zespole, projekt bazy danych, prototyp interfejsu, definicja endpointów
- Sprint II – realizacja pełnej architektury warstwy backend, utworzenie bazy danych, usprawnienie funkcjonalne i wizualne interfejsu, projekt testów
- Sprint III – realizacja dokumentacji, przeprowadzenie testów, usprawnienia wizualne interfejsu

4. Wymagania projektu.

Wymagania funkcjonalne i poza funkcjonalne sporządzone zostały w postaci listy epików z podejściem zwinnym zamierające opis, kryteria akceptacji oraz User Stories. Są one zawarte w pliku **SZP_wymagania.pdf**.

Zostały sporządzone scenariusze do wybranych procesów zachodzących w aplikacji, Są one zawarte w pliku **SZP_scenariusze.pdf**.

Oba pliki zawarte są w katalogu **diagrams** w repozytorium kodu.

5. Architektura i technologia projektu.

5.1. Diagramy.

Diagramy i schematy powstałe w celu stworzenia architektury procesów i funkcji zachodzących w serwisie znajdują się w katalogu **diagrams** w repozytorium kodu.

5.2. Wykorzystane technologie.

W trakcie tworzenia aplikacji wykorzystano następujące technologie i narzędzia :

5.2.1. Języki programowania i technologie :

- Java 21
- Framework Spring
- JavaScript
- Standard JSON Web Token
- Biblioteka Lombok
- Framework Hibernate
- Silnik szablonów Thymeleaf
- Technologia Rest API
- HTML
- CSS
- PostgreSQL

5.2.2. Środowiska programistyczne :

- IntelliJ – część backend
- Visual Studio Code – część frontend
- DBeaver – część bazy danych

5.2.3. Pozostałe narzędzia :

- Discord – komunikacja i przesyłanie mniejszych plików
- Jira – organizacja pracy z podziałem na sprinty, podział obowiązków
- GitHub – repozytorium kodu
- SQLPowerArchitect – generacja diagramów ERD dla bazy danych
- Visual Paradigm – tworzenie diagramów
- Microsoft Word – tworzenie scenariuszy, raportów i dokumentacji
- ElephantSQL – hosting bazy danych
- Postman – weryfikacja zapytań http
- MockMvc – testowanie części aplikacji wykorzystującej framework Spring MVC
- Mockito – testowanie kodu napisanego w języku Java
- Selenium – testowanie interfejsów użytkownika

5.3. Architektura aplikacji.

Wykonaną aplikację można podzielić na trzy warstwy – frontend, backend i bazę danych.

Interfejs użytkownika jest napisany w językach HTML i CSS. Każdy z paneli zbudowany jest na bazie odpowiedniego pliku HTML w folderze odpowiadającemu roli użytkownika, stylów CSS z folderu **styles** oraz skryptu JavaScript z folderu **scripts**. Strony generowane są dynamicznie na podstawie ściśle określonych szablonów stworzonych przez silnik Thymeleaf. Interfejs komunikuje się z serwerem za pomocą zapytań HTTP zrealizowanych w technologii Rest. Autoryzacja użytkowników zachodzi przy pomocy tokenów JSON Web Token.

Serwer za pomocą dedykowanych kontrolerów nasłuchuje zapytań na ściśle zdefiniowanych punktach końcowych drzewa zapytań. Wewnątrz części backend, napisanej w języku Java z wykorzystaniem framework'a Spring, znajduje się logika biznesowa przetwarzająca dane wejściowe i wyjściowe. Komunikacja z bazą danych następuje za pomocą narzędzi z framework'a Hibernate. Powtarzalna część kodu wygenerowana została przez narzędzie Lombok.

Baza danych zdefiniowana i zarządzana w języku PostgreSQL zawiera relacyjną sieć tabeli powiązaną kluczami. Poza danymi znajdują się w niej wyzwalacze dokonujące ewentualnej korekty we wprowadzonych danych.

6. Działanie interfejsu.

W związku z przeznaczeniem systemu dla różnych grup pracowników, panele interfejsu zostały podzielone na cztery grupy.

6.1. Panele wspólne.

- index – Strona otwierająca
- login-page – Strona logowania

6.2. Panele pracownika.

- employee – Strona otwierająca dla pracownika
- success – Strona informująca o pomyślnym złożeniu wniosku
- updateEmploye – Strona edycji danych osobowych
- vacation – Strona z urlopami
- vacationRequest – Strona do składania wniosków o urlop

6.3. Panele HR.

- assignment-form – Strona do dodawania zleceń
- assignments – Strona ze zleceniami
- clients – Strona z klientami
- customer-form – Strona do dodawania nowych klientów
- employees – Strona z pracownikami
- hr – Strona otwierajaca dla pracownika HR.
- leaves – Strona z urlopami
- success-assignment – Strona informująca o pomyślnym otwarciu zlecenia
- workstations – Strona z miejscami pracy

6.4. Panele administratora.

- admin -strona otwierająca dla administratora.

7. Dokumentacja kodu.

Dokładny opis i wyjaśnienie działania poszczególnych klas, metod i pól zawarty jest w kodach źródłowych (gałąź documentation w repozyterium). Klasy składające się na architekturę warstwy backend podzielić można na następujące grupy klas pełniące identyczne role w stosunku do innych obiektów.

7.1. Controller.

Klasy zakończone słowem Controller odpowiedzialne są za kontrolowanie wysyłanych do serwera żądań http, gdzie zawarta jest obsługa określonych endpointów. Odpowiadają również za wyświetlanie konkretnej zawartości użytkownikowi.

7.2. Model.

Klasy zakończone słowem Model reprezentują wiersze z tabel zawartych w bazie danych.

7.3. Repository.

Klasy zakończone słowem Repository odpowiedzialne są za komunikacje z bazą danych, wykorzystując do tego obiekty klasy Model.

7.4. Service.

Klasy zakończone słowem Service odpowiedzialne są za realizację logiki biznesowej dedykowanej odpowiednim rodzajom danych.

7.5. DTO

Klasy zakończone słowem DTO (Data Transfer Object) odpowiedzialne są za uzyskanie pożądanej treści zawartej w formularzach HTTP wysyłanych do serwera.

8. Baza danych.

Baza danych projektu zrealizowana została w języku PostgreSQL. Uruchomiona została na serwerze ElephantSQL. W skład bazy danych wchodzą następujące tabele :

8.1. Workstation.

Tabela zawierająca następujące dane stanowiska pracy :

- workstation_id – unikalny identyfikator stanowiska pracy
- workstation_name – nazwa stanowiska pracy

8.2. Client.

Tabela zawierająca następujące dane klienta :

- client_id – unikalny identyfikator klienta
- first_name – pierwsze imię klienta
- second_name – drugie imię klienta
- last_name – nazwisko klienta
- email – adres e-mail klienta
- phone_number – numer telefonu klienta
- bank_account – numer konta klienta

Ponadto, do tabeli przypisany jest wyzwalacz client_validation odpowiedzialny za walidację danych do niej zapisywanych.

8.3. Employee.

Tabela zawierająca następujące dane pracownika :

- employee_id – unikalny identyfikator pracownika
- first_name – pierwsze imię pracownika
- second_name – drugie imię pracownika
- last_name – nazwisko pracownika
- birth_date – data urodzenia pracownika
- pesel – numer PESEL pracownika
- login – login do serwisu przypisany pracownikowi
- password – hasło do serwisu przypisane pracownikowi
- role – tytuł lub stanowisko pracownika

Ponadto, do tabeli przypisany jest wyzwalacz employee_validation odpowiedzialny za walidację danych do niej zapisywanych.

8.4. Vacation.

Tabela zawierająca następujące dane dotyczące urlopów :

- vacation_id – unikalny identyfikator urlopu
- employee_id – identyfikator pracownika przypisanego do urlopu
- beginning – data rozpoczęcia urlopu
- ending – data zakończenia urlopu
- vacation_type – rodzaj/powód urlopu

Ponadto, do tabeli przypisany jest wyzwalacz vacation_validation odpowiedzialny za walidację danych do niej zapisywanych.

8.5. Assignement.

Tabela zawierająca następujące dane dotyczące zleceń :

- assignement_id – unikalny identyfikator zlecenia
- workstation_id – identyfikator stanowiska pracy przypisanego do zlecenia
- employee_id – identyfikator pracownika przypisanego do zlecenia
- client_id – identyfikator klienta zlecenia
- description – krótki opis zlecenia
- assign_date – data otwarcia zlecenia
- cost – kosz zlecenia
- state – stan zlecenia

9. Testowanie aplikacji.

W celu sprawdzenia poprawności działania aplikacji stworzono zestaw testów jednostkowych i integracyjnych napisanych w języku Java z wykorzystaniem narzędzi MockMvc i Mockito. Pliki źródłowe i wyniki testów w formie zdjęć umieszczone zostały w katalogu **src/test/passed** w rezpozytorium kodu.

9.1. Test HRController.

Plik `HRControllerTest.java` zawiera testy jednostkowe dla kontrolera HR w aplikacji opartej na Spring MVC. Testy używają narzędzi `MockMvc` oraz `Mockito` do symulowania żądań HTTP i mockowania zależności. Przykładowo, test `testGetHomePage` sprawdza, czy żądanie GET na endpoint `/hr/` zwraca poprawny widok `hr/hr` z kodem statusu 200 (OK). Inny test, `testGetEmployeesInfo`, weryfikuje, czy żądanie GET na `/hr/employees` zwraca widok `hr/employees` również z kodem 200, przy użyciu mockowanych danych pracowników. Testy te zapewniają, że kontroler działa poprawnie i zwraca oczekiwane odpowiedzi.

9.2. Test EmployeeController.

Plik `EmployeeControllerTest.java` zawiera testy jednostkowe dla kontrolera Employee w aplikacji opartej na Spring MVC. Testy wykorzystują `MockMvc` oraz `Mockito` do symulowania żądań HTTP i mockowania zależności. Przykładowo, test `testGetHomePage` sprawdza, czy żądanie GET na endpoint `/employee/` zwraca poprawny widok `employee/employee` z kodem statusu 200 (OK). Dodatkowo, testy obejmują mockowanie serwisu JWT w celu symulacji uwierzytelniania. Testy te zapewniają, że kontroler działa poprawnie, zwracając oczekiwane odpowiedzi i obsługując logikę uwierzytelniania.

9.3. Test JWTAuthFilter.

Plik `JwtAuthFilterTest.java` zawiera testy jednostkowe dla filtra autoryzacyjnego JWT w aplikacji Spring Security. Testy używają `Mockito` do symulowania żądań HTTP i mockowania zależności. Przykładowo, test `testDoFilterInternal_TokenIsEmpty` sprawdza, czy metoda `doFilterInternal` poprawnie wywołuje `filterChain.doFilter`, gdy token JWT jest pusty. Testy te zapewniają, że filtr JWT działa prawidłowo w różnych scenariuszach związanych z autoryzacją.

9.4. Test UserDetailsService.

Plik `UserDetailsServiceTest.java` zawiera testy jednostkowe dla serwisu `UserDetailsService` w aplikacji Spring Security. Testy używają `Mockito` do symulowania zależności. Test `testLoadUserByUsername_UserFound` sprawdza, czy metoda `loadUserByUsername` poprawnie zwraca dane użytkownika i jego uprawnienia, gdy użytkownik istnieje. Test `testLoadUserByUsername_UserNotFound` weryfikuje, czy metoda rzuca wyjątek `UsernameNotFoundException`, gdy użytkownik nie zostanie znaleziony. Testy te zapewniają poprawne działanie serwisu w różnych scenariuszach.

9.5. Test EmployeeService.

Plik `EmployeeServiceTest.java` zawiera testy jednostkowe dla serwisu `EmployeeService` w aplikacji Spring. Testy używają `Mockito` do symulowania zależności oraz sprawdzania metod serwisu. Przykładowo, test `testSaveUser` sprawdza, czy metoda `saveUser` poprawnie zapisuje użytkownika, mockując przy tym kodowanie hasła oraz zapis w repozytorium. Test ten weryfikuje, czy użytkownik został poprawnie zapisany i czy jego dane są zgodne z oczekiwaniami. Testy te zapewniają, że serwis `EmployeeService` działa prawidłowo w zakresie zapisywania użytkowników, poprawnie kodując hasła i przechowując dane w repozytorium.

9.6. Test JWTService.

Plik `JwtServiceTest.java` zawiera testy jednostkowe dla serwisu `JwtService` w aplikacji. Testy sprawdzają poprawność działania metod związanych z obsługą tokenów JWT, takich jak `extractUsername` i `extractExpiration`. Przy użyciu `Mockito` testy mockują niektóre funkcje, aby zweryfikować, czy metody zwracają oczekiwane wyniki. Na przykład, test `testExtractUsername` sprawdza, czy metoda `extractUsername` poprawnie wyodrębnia nazwę użytkownika z tokenu JWT. Testy te zapewniają, że serwis `JwtService` działa prawidłowo w zakresie generowania i parsowania tokenów JWT.

9.7. Test RefreshTokenService.

Plik `RefreshTokenServiceTest.java` zawiera testy jednostkowe dla serwisu `RefreshTokenService` w aplikacji. Testy sprawdzają poprawność generowania i zarządzania tokenami odświeżającymi. Na przykład, test `testCreateRefreshToken` weryfikuje, czy metoda `createRefreshToken` poprawnie tworzy token odświeżający dla danego użytkownika, sprawdzając, czy token ma poprawne powiązanie z użytkownikiem i odpowiedni czas wygaśnięcia. Testy te zapewniają, że serwis `RefreshTokenService` działa prawidłowo, poprawnie generując i obsługując tokeny odświeżające.

9.8. Test VacationService.

Plik `VacationServiceTest.java` zawiera testy jednostkowe dla serwisu `VacationService` w aplikacji. Testy sprawdzają poprawność działania metod serwisu związanych z zarządzaniem urlopami. Przykładowo, test `testGetAllVacationsInRangeIncludeAllMatching` weryfikuje, czy metoda `getAllVacationsInRange` poprawnie zwraca wszystkie urlopy w zadanym zakresie dat. Testy te zapewniają, że serwis `VacationService` działa prawidłowo, zwracając odpowiednie dane na podstawie kryteriów wyszukiwania.

9.9. Testy interfejsu.

Wykorzystując narzędzie Selenium wykonano również testy interfejsu użytkownika **TestEmployee** dla interfejsu pracownika i **TestHR** dla interfejsu pracownika HR.

10. Dane do uruchomienia programu.

Repozytorium kody znajduje się pod linkiem https://github.com/JaskiewiczK/SZP_studia.

Aby zalogować się do serwisu jako pracownik należy użyć loginu **em** i hasła **em**.

Aby zalogować się do serwisu jako pracownik należy użyć loginu **hr** i hasła **hr**.

11. Podsumowanie.

Nie udało się zrealizować wszystkich wymagań zebranych na wczesnym etapie projektu, jednak ten został rozszerzony o inne funkcjonalności. Aplikacja została stworzona z podejściem otwartym na rozszerzenia, dlatego też jej dalszy rozwój jest możliwy i nie wymaga znaczących zmian w strukturze.

Dla każdego z członków zespołu była to szansa na zebranie nowych doświadczeń w pracy w zespole, zarówno przy wykonywaniu pojedynczych zadań z naciskiem na kompatybilność jak i planowaniu komunikacji pomiędzy modułami tworzonymi przez innych członków.

Podział obowiązków ze względu na doświadczenie i indywidualne umiejętności okazał się bardzo skuteczny. Podział projektu na trzy sprinty, każdy kładący nacisk na inny element projektu umożliwił zwinne przesuwanie środka ciężkości prac oraz wielokrotną informację zwrotną o stanie projektu.

Kluczowym narzędziem do organizacji pracy okazała się platforma Jira która pozwoliła w bardzo czytelny sposób podzielić obowiązki każdego z członków zespołu, zaplanować je w czasie, pogrupować w większe zadania i nadać im odpowiednie priorytety.

Podsumowując, mimo że stworzona aplikacja nie wprowadza dużo innowacyjności, była bardzo dobrym poligonem do nauki pracy w zespole i planowania projektów, umiejętności esencjonalnej w dalszej karierze zawodowej.

12. Źródła.

- Materiały przygotowane przez prowadzącego.
- https://spring.io/
- https://circle.visual-paradigm.com/docs/
- https://www.selenium.dev
- https://tools.jboss.org/documentation/
- https://www.thymeleaf.org/documentation.html