



POLITECHNIKA ŚLĄSKA
WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Praca dyplomowa inżynierska

Aplikacja do zarządzania oraz planowania zajęć dla nauczyciela
akademickiego/studenta

autor: Rafał Paprota

kierujący pracą: dr inż. Ewa Płuciennik

Gliwice, styczeń 2021

załącznik nr 2 do zarz. nr 97/08/09

Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej*.

Gliwice, dnia 10 stycznia 2021

.....
(podpis)

.....
(poświadczenie wiarygodności
podpisu przez Dziekanat)

* podkreślić właściwe

Spis treści

1	Wstęp	1
2	Analiza tematu	3
2.1	Wprowadzenie do dziedziny	3
2.2	Podobne rozwiązania	5
3	Wymagania i narzędzia	7
3.1	Wymagania funkcjonalne	7
3.2	Wymagania niefunkcjonalne	8
3.2.1	Bezpieczeństwo aplikacji	8
3.2.2	Niezawodność aplikacji	8
3.2.3	Szybkość działania	8
3.2.4	Użyteczność	9
3.2.5	Wydajność	9
3.3	Przypadki użycia	9
3.4	Wykorzystywane narzędzia	11
3.4.1	Spring Boot	11
3.4.2	Gradle	12
3.4.3	React	12
3.4.4	PostgreSQL	13
3.4.5	Dodatkowe narzędzia	13
3.5	Metodyka pracy	14
4	Specyfikacja zewnętrzna	17
4.1	Wymagania programowe	17

4.2	Sposób uruchomienia	18
4.3	Sposób obsługi	19
4.4	Przykłady działania	26
4.4.1	Logowanie	26
4.4.2	Zmiana hasła	27
4.4.3	Dodanie nowego bloku	29
4.4.4	Dodanie wydarzeniami niepowtarzającego	30
4.4.5	Widoki w harmonogramie	32
4.5	Kwestie bezpieczeństwa	33
5	Specyfikacja wewnętrzna	35
5.1	Architektura systemu	35
5.2	Opis bazy danych	38
5.3	Użyte biblioteki	40
5.3.1	Material-UI	40
5.3.2	Bootstrap	40
5.3.3	Axios	40
5.3.4	Redux toolkit	41
5.3.5	DevExtreme React Scheduler	41
5.3.6	Spring-Boot-Starter	41
5.4	Ważniejsze komponenty	41
5.5	Wybrane fragmenty kodu	43
6	Weryfikacja i walidacja	47
6.1	Sposoby testowania aplikacji	47
6.1.1	Testy integracyjne	47
6.1.2	Testy systemowe	48
6.2	Wykryte i usunięte błędy	49
7	Podsumowanie i wnioski	51

Rozdział 1

Wstęp

Celem pracy jest utworzenie aplikacji internetowej, która umożliwiałaby studentom oraz nauczycielom akademickim zarządzanie własnym planem zajęć, dostosowanym do wybranego przez uczelnię modelu. Plany zajęć utworzone przez użytkowników powinny być zapisywane w bazie danych, aby można było cały czas z nich korzystać.

Dla uczelnianych planów zajęć kluczowe jest, że wydarzenia nie muszą się powtarzać co tydzień, mogą to być różne przedziały np. co dwa tygodnie. Zajęcia są zazwyczaj powieleniem 45 minut, czyli trwają z reguły 1,5 godziny, 2 godziny 15 minut itd. W większości przypadków, pomiędzy kolejnymi zajęciami znajduje się 15 minutowa przerwa, aczkolwiek zdarzają się sytuacje, gdzie plan posiada zajęcia jedno po drugim. Poza samymi zajęciami istnieją też aktywności takie jak: kolokwia, egzaminy, kartkówki czy też konsultacje. Są to wydarzenia, które nie są znane od samego początku, w większości przypadków terminy są ustalane w trakcie trwania semestru. Specjalnym rodzajem wydarzeń są godziny dziekańskie, bądź godziny rektorskie. Oznacza to dla studentów, że w obrębie podanych godzin lub dni, zajęcia dydaktyczne nie będą się odbywały.

Niektóre uczelnie wprowadziły do swoich planów system blokowy. Oznacza to, że poszczególne semestry, dzielone są na kilka mniejszych bloków, w których odbywają się zajęcia przeznaczone na konkretny semestr. Każdy blok posiada mniej więcej zrównoważoną ilość zajęć. Oznacza to, że rozbieżność między liczbą zajęć w poszczególnych blokach nie odbiega od siebie diametralnie. Blok jest traktowany

jako mniejszy semestr, gdzie po ukończeniu konkretnej ilości godzin na konkretnych zajęciach, przeprowadzany jest test zrozumienia tematu. W przypadku ukończenia zajęć podczas trwania jednego bloku, w następnych blokach zajęcia te nie powtarzają się. System ten powstał ze względu na chęć rozłożenia nauki całego materiału, na mniejsze części. Podczas trwania semestru w normalnym trybie na jego końcu odbywa się sesja egzaminacyjna, gdzie trzeba całą zdobytą wiedzę podczas semestru udowodnić na egzaminach. System podzielił materiał na tyle części ile występuje bloków, dzięki czemu nauka odbywa się przez cały semestr, a nie pod sam koniec.

Najlepszym sposobem na przechowywanie informacji o planie zajęć jest kalendarz, ze względu na to, że przynajmniej raz dziennie ludzie sprawdzają jaką mamy datę. W momencie zastosowania kalendarza do obsługi zajęć, łatwiej jest zapamiętać jakie konkretne zajęcia lub aktywności występują konkretnego dnia. Dodatkowo w momencie użycia kalendarza, całościowy plan jest dobrze rozłożony w czasie, ponieważ istnieje przedział godzinowy, dzięki, któremu można sobie wyobrazić cały plan dnia jako listę konkretnych aktywności do wykonania. Kalendarze mają to do siebie, że uwzględniają, liczbę dni w każdym miesiącu, dzięki czemu nie trzeba za każdym razem na nowo definiować przedziałów roku, miesięcy, tygodni. W kalendarzu dodatkowo można dopisywać wydarzenia, które nie są ściśle związane z jakimś zagadnieniem. Można dodać notatkę typu dentysta i zaznaczyć swój przedział godzinowy. Usprawnia to w dużej mierze organizację własnego czasu, dzięki czemu możemy zaoszczędzić czas, którego nie da się kupić.

Połączenie powyższych elementów powinno wspomóc utworzenie systemu, który ułatwiał by zarządzanie własnym planem zajęć.

Praca składa się z siedmiu rozdziałów. W rozdziale drugim zostanie przedstawione wprowadzenie do tematu oraz podobne rozwiązania. Następnie przechodząc do rozdziału trzeciego zostaną ukazane informacje na temat użytych narzędzi, zostaną omówione wymagania aplikacji i opis działania użytkownika w systemie. W rozdziale czwartym, autor przedstawi jak zainstalować oprogramowanie, jak je obsługiwać oraz jakie wymagania muszą zostać spełnione, aby oprogramowanie działało. Zostanie również zilustrowane przykładowe działanie aplikacji oraz poruszony temat bezpieczeństwa. Wraz z pojawieniem się rozdziału piątego pokazana będzie architektura oraz struktury danych, natomiast w kolejnych zostaną zdefiniowane testy, podsumowanie i wnioski wraz z dalszymi planami rozwojowymi.

Rozdział 2

Analiza tematu

Rozdział poświęcony jest analizie tematu, w ramach której zostaną omówione podobne rozwiązania oraz przedstawione zostaną aspekty, które powinny znajdować się w aplikacji.

2.1 Wprowadzenie do dziedziny

Głównym zadaniem kreowanego systemu jest umożliwienie użytkownikom zarządzania własnym planem zajęć. Każdy użytkownik, powinien dostać możliwość dodawania, modyfikacji oraz usuwania wydarzeń. Biorąc pod uwagę specyfikę zajęć na uczelni wyższej, w każdym nowo powstałym wydarzeniu, powinna zostać zawarta informacja o jego typie. Przykładowe typy to wykład, laboratorium, ćwiczenia. Poza samymi zajęciami aplikacja powinna umożliwiać zarządzanie aktywnościami np. sprawozdanie, kolokwium, egzamin. Dodatkowo powinny zostać uwzględnione aktywności związane z rozporządzeniami władz uczelni tak jak np. godziny rektorskie, godziny dziekańskie. Do kalendarza powinny być wpisywane wydarzenia zawierające informacje o ich nazwie, dacie rozpoczęcia, dacie zakończenia, oraz ewentualnie dodatkowych notatkach. W notatkach można umieścić informację np. o grupie dziekańskiej, numerze sali, bądź krótki wpis o charakterze informacyjnym. Wydarzenie powinno posiadać swój status, ze względu na możliwość edycji wydarzeń. Informacja o tym czy zostało ono zmienione czy też dalej jest w pierwotnej postaci może okazać się przydatną. Użytkownik, powinien do-

stać możliwość dodawania wydarzeń powtarzających się. Powinien zostać przygotowany kreator, który umożliwi wybór dni, tygodni, czy miesięcy powtarzania się wydarzenia z doprecyzowaniem daty zakończenia powtarzania. Aplikacja powinna uwzględniać możliwość występowania wydarzeń co tydzień, co dwa tygodnie, bądź w konkretne dni w konkretnym tygodniu.

Aplikacja powinna dawać możliwość tworzenia nowych kont użytkowników oraz zalogowania się do konta. Dzięki tej funkcji można zapewnić, że każdy z użytkowników będzie miał dostęp do własnego planu zajęć.

W aplikacji powinna znajdować się historia wydarzeń, która miała by na celu przechowywanie wydarzeń przedawnionych. Użytkownik powinien mieć wgląd do całej jego historii. Najlepiej żeby historia wydarzeń nie różniła się zbytnio od samego harmonogramu, żeby nie utrudniać korzystania z aplikacji.

Powinna znajdować się opcja, gdzie użytkownik mógłby definiować sam, po jakim czasie wydarzenie powinno zostać uznane za zakończone i zostać przeniesione do historii. Dodatkowo powinna zostać dodana opcja umożliwiająca edycję hasła czy też adresu email.

Do ważnych opcji należała by implementacja modułu odpowiedzialnego za zarządzanie blokami. Użytkownik powinien mieć możliwość definiowania własnych bloków. Każdy z nich powinien przechowywać informację jego dotyczącą czyli jego nazwę, ewentualne notatki, datę rozpoczęcia oraz datę zakończenia. Powinien zostać wprowadzony system sprawdzający czy bloki w obrębie jednego konta użytkownika posiadają unikalną nazwę oraz czy nie nachodzą na siebie czasowo. Każdy powinien posiadać własny przedział czasowy, który nie koliduje z żadnym innym. Dodatkowo powinna zostać wprowadzona opcja modyfikacji oraz usuwania bloków.

Użytkownik powinien posiadać możliwość dodawania wydarzeń w zależności od wybranego bloku. Kreator tworzenia wydarzeń powinien uwzględniać możliwość, dodawania wydarzeń w przedziale czasowym zdefiniowanym w konkretnym bloku. Podczas dodania wydarzenia powtarzającego, zakres dat jego powtarzalności powinien być odpowiednio dopasowany w zależności od dat bloku.

Z punktu widzenia potencjalnego konsumenta, aplikacja powinna zawierać przejrzysty interfejs graficzny, który nie wymagałby dużej wiedzy, aby móc z niej korzystać. Powinien w jak najłatwiejszy sposób przedstawiać dane oraz udostępniać opcje, z których może korzystać użytkownik.

2.2 Podobne rozwiązania

Istnieje obecnie sporo rozwiązań do zarządzania własnym planem zajęć, aczkolwiek są one zbyt skomplikowane i niekoniecznie przyjazne dla użytkownika jeśli chodzi o warstwę interfejsu graficznego jak np. bitrix24. Istnieją też takie, które posiadają przyjemny interfejs, lecz nie mają na tyle dostępnych opcji żeby przyspieszyć proces zarządzania planem lub nie są dostosowane do uczelni wyższej jak aplikacja Kalendarz dostarczana przez Google. Znajdą się też takie rozwiązania, które ułatwiają zarządzanie planem dla całej organizacji, które narzucają odgórnie plan i nie dają możliwości jego edycji potencjalnemu użytkownikowi. Niektóre z nich nie są darmowymi rozwiązaniami, tylko trzeba zapłacić odpowiednią ilość pieniędzy za ewentualną subskrypcję, która może wygasnąć. Do poprzednich dwóch można zaliczyć oprogramowanie ProAkademia. Aplikacje typu dziennik elektroniczny jak Synergia librus, edudziennik dają możliwość ułożenia planu zajęć, natomiast student nie posiada możliwości zarządzania nim. Dostaje tylko i wyłącznie możliwość przeglądania już ułożonego planu przez co, żeby uzupełnić własny grafik, musi przenieść wszystkie zajęcia do zewnętrznej aplikacji, aby móc efektywnie zarządzać swoim planem.

Aplikacja tworzona w ramach tej pracy posiada prosty w obsłudze interfejs graficzny, dodatkowe opcje, które wspomagają układanie planu uwzględniającą specyfikację semestru. Opcje, które ułatwiają czytelność przedstawianych danych, takie jak możliwość wyboru typów zajęć, samoaktualizujący się status wydarzenia. Podział wydarzeń aktywnych, aktualnych oraz tych, które się zakończyły. Dodatkowo aplikacja jest udostępniona za darmo przy pomocy serwisu GitHub [4], dzięki czemu można ją modyfikować, ulepszać, bądź ograniczać według własnych potrzeb.

Rozdział 3

Wymagania i narzędzia

W tym rozdziale zostaną przedstawione wymagania, jakie powinna spełniać aplikacja. Zarówno te funkcjonalne, do których dostęp mają użytkownicy oraz te нефункционалне, które są potrzebne z punktu widzenia jakości jej działania. Dodatkowo zostaną zawarte informacje o niezbędnych narzędziach, które zostały wykorzystane w celu osiągnięcia postawionych wymagań.

3.1 Wymagania funkcjonalne

Do uzyskania pełnego zestawu funkcji aplikacji należało przygotować zbiór wymagań funkcjonalnych:

- Umożliwienie użytkownikowi założenie nowego konta
- Umożliwienie użytkownikowi zalogowania się na konto
- Umożliwienie użytkownikowi zapoznania się ze sposobów korzystania z aplikacji
- Pozwolenie użytkownikowi na zmianę hasła do konta
- Pozwolenie użytkownikowi na zmianę adresu email
- Pozwolenie użytkownikowi na zmianę czasu do archiwizacji wydarzeń
- Umożliwienie użytkownikowi zdefiniowania własnego bloku zajęciowego

- Umożliwienie użytkownikowi usunięcia oraz modyfikacji istniejących bloków
- Danie dostępu użytkownikowi do przeglądania wydarzeń, które zostały zarchiwizowane
- Umożliwienie użytkownikowi stworzenia własnego planu zajęć
- Umożliwienie użytkownikowi dowolnej modyfikacji planu zajęć

3.2 Wymagania niefunkcjonalne

Znajdują się tu wymogi, które nie występują w wymaganiach funkcjonalnych. Dotyczą jakości aplikacji. Zawierają ograniczenia, których aplikacja musi przestrzegać, lecz nie mają styczności z jej funkcjonalnością.

3.2.1 Bezpieczeństwo aplikacji

Aplikacja powinna być bezpieczna, czyli dane użytkowników powinny być zabezpieczone przed próbą modyfikacji przez niepożądaną osobę. Przede wszystkim dane nie powinny być dostępne dla innych. Dane klienta do zalogowania w aplikacji powinny być zaszyfrowane, natomiast meta dane wewnątrz aplikacji powinny być zabezpieczone przynajmniej poprzez autoryzację klienta.

3.2.2 Niezawodność aplikacji

Aplikacja powinna być niezawodna. Oznacza to, że powinna być zabezpieczona przed podaniem przez użytkownika nieprawidłowych danych lub takich, których aplikacja nie przewiduje, bądź nie jest w stanie rozszyfrować. W momencie podania błędnych danych, powinno zostać wysłane odpowiednie powiadomienie, które ma na celu poinformowanie o błędzie. Aplikacja nie powinna zostać wstrzymana ani zatrzymana, tylko powinna posiadać system obsługi do rozwiązywania konfliktów.

3.2.3 Szybkość działania

Aplikacja nie powinna się zbyt długo ładować. Dane powinny być ładowane w jak najprostszej postaci oraz jak najmniej objętościowo, dzięki czemu zostanie

uzyskany mniejszy czas oczekiwania na wyświetlenie, bądź załadowanie nowych informacji.

3.2.4 Użyteczność

Przede wszystkim aplikacja z punktu widzenia interfejsu nie powinna być skomplikowana. Powinna być estetyczna oraz łatwa w obsłudze. Użytkownik powinien móc w krótkim czasie nauczyć się działania oraz obsługi aplikacji.

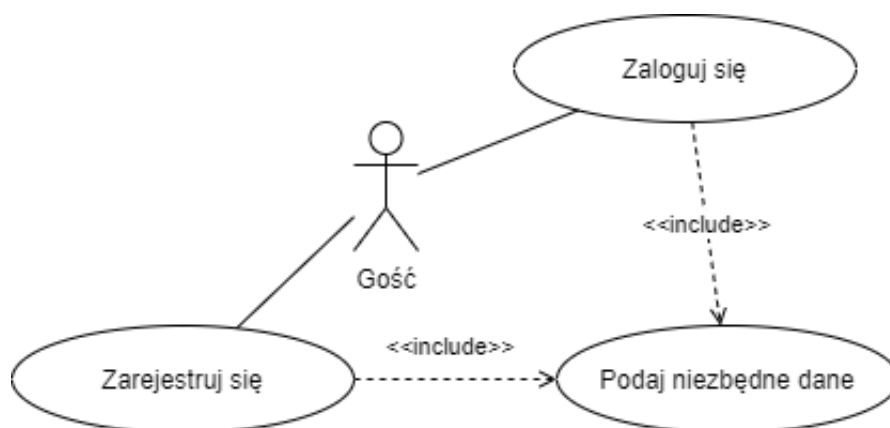
3.2.5 Wydajność

Aplikacja nie powinna wymagać dużej ilości przestrzeni dyskowej i pamięci operacyjnej. Dodatkowo nie powinna w znaczący sposób obciążać procesora. Czas reakcji na wysłane przez użytkownika zapytania powinien być relatywnie krótki, aby nie zniechęciło go długie oczekiwanie na odpowiedź.

3.3 Przypadki użycia

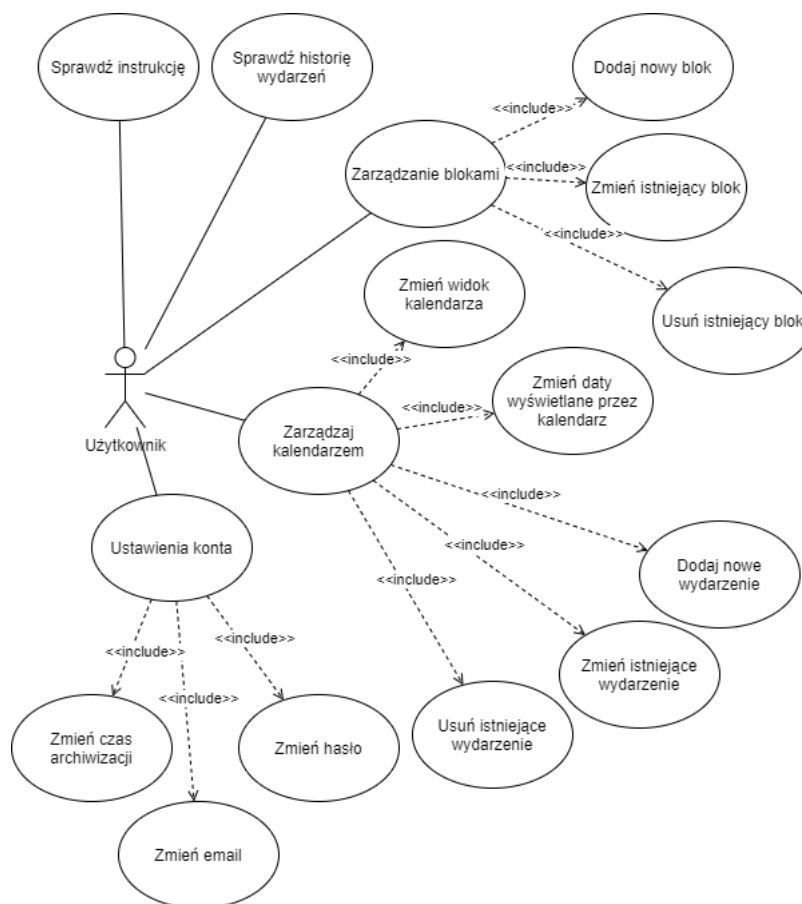
Biorąc pod uwagę funkcje oraz działanie całego systemu, można wyróżnić dwóch aktorów:

- Gość
- Użytkownik



Rysunek 3.1: Diagram przypadków użycia dla gościa

Gość nie ma dostępu do wszystkich modułów systemu. Ma udostępnione dwie opcje, które nie wymagają autoryzacji użytkownika. Na rysunku 3.1 można zaobserwować, że gość nie posiada dużej liczby funkcji oraz przypadków do jakich może przejść. Pierwszą funkcją jest rejestracja, na skutek której zostaje utworzone nowe konto. Drugą jest logowanie, po którym gość staje się użytkownikiem i dostaje dostęp do wszystkich modułów systemu.



Rysunek 3.2: Diagram przypadków użycia dla użytkownika

Na rysunku 3.2 można zaobserwować, że wachlarz funkcji aplikacji dla użytkownika jest znacznie większy. Użytkownik otrzymuje pełen zbiór możliwości aplikacji. Dostępne jest kilka przypadków, które mają różne przeznaczenie. Istnieją przypadki typowo informacyjne, takie jak sprawdź instrukcje, która przedstawia użytkownikowi instrukcję obsługi programu. Sprawdź historię wydarzeń, w któ-

rym użytkownik może sprawdzić historię wszystkich swoich wydarzeń, które uległy przedawnieniu. Istnieją również przypadki, które umożliwiają interakcję takie jak zarządzanie blokami, zarządzanie kalendarzem oraz ustawienia konta. Dla wymienionych przypadków użytkownik może wprowadzać dane, zmieniać je, bądź usuwać. W przypadku ustawień konta, użytkownik dostaje opcje zmiany hasła, adresu email oraz czasu archiwizacji przedawnionych wydarzeń. Dla zarządzania blokami, może definiować nowe bloki, modyfikować je oraz usuwać. Więcej opcji jest dla zarządzania kalendarzem, ponieważ poza dodawaniem, modyfikacją oraz usuwaniem, użytkownik może jeszcze zmienić widoki kalendarza, oraz zmienić daty, które definiują przedział czasowy wyświetlanych wydarzeń.

3.4 Wykorzystywane narzędzia

Znajduje się tu zbiór wszystkich niezbędnych oraz tych opcjonalnych narzędzi, które zostały użyte do stworzenia systemu.

3.4.1 Spring Boot

Convention-over-configuration jest to rozwiązanie pozwalające na tworzenie zaawansowanych aplikacji przy stosunkowo małej ilości kodu oraz konfiguracji [11]. Zanim został opublikowany Spring Boot programiści sami musieli zapewnić aplikacji odpowiednią konfigurację oraz zależności pomiędzy bibliotekami. Teraz urządzenie to dostarcza szablony startowe, które automatyzują oraz w dużym stopniu upraszczają budowanie aplikacji. Podczas startu aplikacji tworzony jest kontekst Springa. Jest to miejsce w którym komponenty *JavaBean* są rejestrowane. Komponenty w kontekście aplikacji są od siebie zależne, natomiast żeby nie powielać ich w kontenerze podczas powoływania nowych komponentów stosuje się wstrzykiwanie zależności. W momencie tworzenia nowego obiektu, który posiada adnotację wstrzykującą zależność, nie jest powoływany nowy komponent do kontekstu. W tym momencie do obiektu jest dostarczany komponent znajdujący się w kontekście aplikacji. Zależności wstrzykiwane są najczęściej w konstruktorach komponentów, poprzez metody ustawiające tzw. *setter*y lub poprzez odpowiednie adnotacje nad polem.

3.4.2 Gradle

Jedno z najpopularniejszych narzędzi, które automatyzuje proces tworzenia aplikacji napisanych w języku Java. Narzędzie to w prosty sposób definiuje jak budowany jest projekt. Proces budowania projektu jest podzielony na trzy etapy. Inicjalizacja, konfiguracja oraz wykonanie. Podczas inicjalizacji, narzędzie określa jakie projekty będą uwzględniane podczas budowania aplikacji. Dla każdego z nich jest utworzony obiekt przechowujący konfigurację. Podczas fazy konfiguracji zostają wykonane pliki konfiguracyjne, dzięki czemu utworzone podczas inicjalizacji obiekty dostają nową konfigurację na podstawie plików budujących (*build.gradle*). W fazie wykonania określany jest zestaw zadań jakie powinny zostać zrealizowane oraz w jakiej kolejności. Zadania do wykonania pobierane są z obiektów utworzonych w poprzednim kroku. Wszystkie zależności jakie są pobierane przez Gradle są przechowywane w schowku. Dzięki temu rozwiązaniu podczas dodawania kolejnych zależności do nowych projektów najpierw Gradle zagląda do schowka, a jeśli nie znajduje odpowiednich plików dopiero pobiera dane z repozytorium [9].

3.4.3 React

Javascriptowa biblioteka, która służy do tworzenia rozległych stron internetowych np. portali społecznościowych. Jest używana również przy tworzeniu interfejsów użytkownika korzystających z Rest API (ang. *Representational State Transfer Application Programming Interface*). Jedną z cech, które wyróżniają bibliotekę React jest wirtualny DOM (ang. *Document Object Model*). Biblioteka ta gromadzi cały model aplikacji w pamięci, a w momencie kiedy status DOM'u ulega zmianie, wyszukuje zmiany pomiędzy obecnym DOM'em, a tym wirtualnym i nadpisuje je. Kolejną cechą, którą posiada React [7] jest specyficzny język JSX, który jest połączeniem języka JavaScript oraz XML. Umożliwia on wstawianie kodu HTML, bądź gotowych rozwiązań samej biblioteki React swobodnie w kodzie. React pozwala na dowolne tworzenie nowych komponentów, obsługujących własny stan, a następnie używanie ich ponownie w kodzie. Komponenty w React umożliwiają przekazywanie danych do swojego wnętrza, a następnie ich edycję. Gotowy element może posiadać własny stan, a po jego zmianie, metoda *render* występująca w każdym komponencie, zaktualizuje go i wyświetli na ekranie jego zmienioną wartość.

3.4.4 PostgreSQL

Jedno z najpopularniejszych rozwiązań jeśli chodzi o relacyjne bazy danych na obecnym rynku. Umożliwia przechowywanie danych w łatwy sposób oraz w najprostszej postaci. Pomimo, że jest to relacyjna baza to umożliwia również składowanie danych w postaci nie relacyjnej, co w niektórych przypadkach może okazać się przydatne. Wybrane zostało również ze względu na to, że jest oprogramowaniem darmowym. Jest również takim, który wspiera język Java, dzięki czemu jego integracja z kodem aplikacji okazała się prosta.

3.4.5 Dodatkowe narzędzia

Znajdują się tu narzędzia, które nie są niezbędne do wytworzenia aplikacji ale znacznie ułatwiają pracę nad tworzonym systemem. Aplikacje te wspomagają testowanie czy też ułatwiają początkującemu programiście, bądź użytkownikowi odczyt danych. Dane te są wizualizowane w sposób łatwy do zrozumienia. Są to też narzędzia, które wspomagają utrzymanie kodu, oraz zabezpieczają przed utratą postępów. Wszystkie rozwiązania są darmowe, dzięki czemu każdy może z nich korzystać.

Postman

Proste w obsłudze oraz przydatne narzędzie do testowania Rest API. Aplikacja umożliwia wysyłanie zapytań HTTP do API oraz ewentualne ich późniejsze zapisanie. Służy ona w dużej mierze do testowania czy zaimplementowane w interfejsie metody poprawnie obsługują nadesłane dane. Aplikacja umożliwia wysyłanie danych w różnych formatach, od zwykłego tekstu, po złożone obiekty takie jak JSON (ang. *JavaScript Object Notation*). Poza testowaniem prostych zapytań do API, umożliwia również wysyłanie zapytań autoryzowanych. Do żądania można dodać token uwierzytelniający, który daje dostęp do zabezpieczonych funkcji API, które są dostępne tylko dla uwierzytelnionych użytkowników. Dzięki graficznej szacie interfejsu użytkownika w prosty sposób można wybierać typ zapytania, adres pod który zapytanie ma się udać, dane, które chcemy umieścić w zapytaniu oraz rodzaj autoryzacji.

Git

Najkrócej ujmując jest to system kontroli wersji. Służy on do przechowywania danych całych projektów oraz do zarządzania wersjami produktu. Umożliwia cofanie zmian do pewnych momentów w kodzie. Za pomocą specjalnych komend konsolowych, bądź przy pomocy programów graficznych użytkownik może bezpiecznie przechowywać swój projekt właśnie za pomocą tej aplikacji. Każdy programista, który dba o swój kod oraz o rezultaty swojej pracy, używa go, aby w razie jakichkolwiek losowych sytuacji, móc zawsze wrócić do ostatnio opublikowanej wersji w repozytorium. Git jest również narzędziem umożliwiającym tworzenie osobnych gałęzi rozwojowych oprogramowania. Programiści po ukończeniu i przetestowaniu części aplikacji znajdującej się na konkretnej gałęzi mogą złączyć ją z gałęzią główną. Gałąź główna jest gotowym produktem, który jest przeznaczony do użytku przez użytkownika. Głównymi zaletami gita jest wydajność, lokalność, kontrola spójności danych, optymalizacja rozgałęzień. Git sprawdza wszystkie pliki w danym folderze, a nie tylko kod źródłowy, folder z takimi plikami nazywany jest repozytorium. W repozytorium znajduje się specjalny folder `.git`, który przechowuje informacje o śledzonych plikach [14].

PgAdmin

Jest to graficzny interfejs do zarządzania oraz pracy użytkownika z bazami danych PostgreSQL potocznie zwanymi postgres. Jest to program polecany dla początkujących, ponieważ w graficzny sposób można analizować zapytania wysyłane do bazy. Umożliwia podgląd konkretnych tablic w banku danych w bardzo prostej postaci. Dane przedstawiane są przy pomocy tabel z opisem kolumn. Daje możliwość łatwego posługiwania się poleceniami w odróżnieniu od programów konsolowych. Umożliwia również szybkie pojęcie wiedzy z zakresu baz danych dzięki oprawie graficznej.

3.5 Metodyka pracy

Na samym początku pracy została podjęta decyzja dotycząca metodyki jaka będzie stosowana podczas tworzenia systemu. Metodyka na jaką postawiono to

jedna z zaliczających się do metodyk zwinnych, które nie kładą nacisku na dokumentację. Nie narzucają sztywnego trzymania się wcześniej spisanych wymagań. Pierwotnie został dogłębnie przeanalizowany temat aplikacji, a następnie zostały spisane wstępne wymagania funkcjonalne oraz нефункционалне. Następnie zostały wybrane technologie, które zostaną użyte podczas tworzenia całego systemu. Po ponownym przeanalizowaniu wymagań, zostały utworzone startowe projekty i rozpoczął się proces implementacji. Funkcje były wdrażane iteracyjnie. Pomimo jednoosobowej pracy szczególny nacisk został położony na stosowanie się do manifestu Agile. W jego zapisie zostało zaznaczone, że praca nad projektem przebiega w krótkich iteracjach. Po każdej z nich powinno zostać dostarczone coś nowego do działającego systemu co oznacza, że po każdej iteracji wytwarzane oprogramowanie jest poszerzane o kolejną funkcjonalność. Każda funkcja po iteracji jest przetestowana oraz gotowa do obsługi przez użytkownika [12]. Po zaimplementowaniu wstępnie ustalonych elementów, system został przedstawiony klientowi. Dalszy etap prac polegał na konsultacji z klientem oraz implementowaniu ulepszeń, bądź zmian, które chciałby znaleźć w programie.

Rozdział 4

Specyfikacja zewnętrzna

W rozdziale tym znajdują się wszystkie niezbędne informacje dla użytkownika, które mają uprościć, instalację, uruchamianie oraz korzystanie z aplikacji. Wymagania zostały sformułowane dla używania aplikacji lokalnie na własnym sprzęcie, natomiast jest możliwość udostępnienia aplikacji na zewnętrznych serwerach. Cały system zarówno aplikacja serwerowa jak i kliencka mogą zostać uruchomione przy pomocy platformy Heroku [5].

4.1 Wymagania programowe

Do uruchomienia aplikacji niezbędne będzie oprogramowanie PostgreSQL obsługujące relacyjne bazy danych. Należy posiadać zainstalowany program, ponieważ bank danych został stworzony przy pomocy tego narzędzia. Aby uruchomić aplikację kliencką niezbędne jest posiadanie środowiska uruchomieniowego do wykonywania kodu JavaScript. Zalecane jest użycie środowiska Node.js. Aby w prosty sposób uruchomić aplikację serwerową zaleca się użycie oprogramowania IntelliJ IDEA, który posiada wbudowany serwer Tomcat i poprzez naciśnięcie jednego przycisku, aplikacja serwerowa zostanie uruchomiona na serwerze lokalnym. Aby móc uruchamiać oraz kompilować aplikację niezbędne będzie zainstalowanie pakietu JDK (ang. *Java Development Kit*). Jest to oprogramowanie udostępniające środowisko niezbędne do programowania w języku Java. Oprogramowanie jest tworzone w wersji Java 11 dlatego odpowiednią wersją będzie JDK 11.

4.2 Sposób uruchomienia

W celu uruchomienia aplikacji lokalnie należy:

- Zainstalować programy opisane w wymaganiach programowych.
- Uruchomić aplikację pgAdmin 4 dołączoną do oprogramowania PostgreSQL, następnie utworzyć nowe konto dla administratora bazy danych, bądź zalogować się na istniejące. Po zalogowaniu rozwinąć zakładkę *Servers*. Wybrać prawym przyciskiem myszy PostgreSQL 12, przejść do *Create*, a następnie wybrać *Database*. W kolejnym kroku podać nazwę bazy danych, oraz wybrać właściciela. Po przejściu wszystkich punktów można wyłączyć PgAdmin 4 i przejść do kolejnego etapu.
- Uruchomić aplikację IntelliJ IDEA. Należy zaimportować projekt z aplikacją serwerową. Odszukać w folderze *src* plik *application.yml*. W miejscu pola *url* należy podać adres utworzonej bazy danych, który domyślnie jest ustawiony na *jdbc:postgresql://localhost:5432/scheduler*. W przypadku lokalnego uruchamiania z opcji podstawowych, wystarczy zmienić nazwę z *scheduler* na nazwę utworzonej bazy. Nadać port na którym będzie pracowała nasza baza, domyślnie ustawiony na lokalny czyli 8080. Następnie w polu *username* zmienić nazwę na konta uprawnionego do wprowadzania zmian do bazy, a w polu *password* podać hasło.
- Aby uruchomić aplikację serwerową, należy wybrać z górnego paska zakładkę uruchom i uruchomić *SchedulerServerApplication*
- W plikach aplikacji klienckiej odszukać plik *config.js*, a następnie podać adres IP serwera na którym będzie uruchomiona aplikacja serwerowa. Domyślnie ustawiony jest adres lokalny.
- Po przeprowadzeniu konfiguracji. Należy wejść w katalog główny z aplikacją kliencką. Otworzyć konsolę w tej lokalizacji, bądź najpierw otworzyć konsolę, a następnie za pomocą komend dostać się do folderu głównego z aplikacją. W następnym kroku wpisać komendę *npm i*, bądź *npm install*, która pobierze

wszystkie wymagane biblioteki oraz moduły, które są niezbędne do działania oprogramowania. W następnym kroku wpisać komendę `npm start`, która uruchomi program kliencki.

- Po wystartowaniu serwera oraz aplikacji klienta, można uruchomić przeglądarkę i wpisując w górnym pasku `http://localhost:3000/` zostanie załadowana strona logowania.

4.3 Sposób obsługi

Aby móc korzystać z całości dostępnych funkcji, należy się zarejestrować poprzez podanie niezbędnych danych. Po pomyślnej rejestracji, powinno się przejść do zakładki logowania, oraz uwierzytelnić swoje konto w systemie po czym zostają udostępnione wszystkie funkcje systemu.

Ustawienia - zakładka ta umożliwia użytkownikowi zmianę hasła, emaila lub czasu archiwizacji. Czas archiwizacji jest to okres po jakim ukończone wydarzenia będą przenoszone do historii wydarzeń. Każdą zmianę należy zatwierdzić poprzez podanie aktualnego hasła do konta użytkownika.

Zarządzanie blokami - moduł ten umożliwia użytkownikowi definiowanie własnych bloków zajęciowych. Bloki posiadają swoją nazwę, która powinna być unikalna w obrębie jednego użytkownika. Oznacza to, że każdy blok powinien mieć własną nazwę. Dodatkowo w blokach udostępnione jest pole na notatki, które może, lecz nie musi być uzupełnione przez użytkownika. Każdy blok powinien mieć z góry określony przedział czasowy, a daty powinny być ułożone w prawidłowy sposób, czyli data zakończenia bloku nie może występować przed datą rozpoczęcia. Nie jest możliwe aby blok, rozpoczynał się oraz kończył tego samego dnia. Ważną informacją jest, że bloki nie mogą łamać swoich przedziałów czasowych. Oznacza to, że w momencie kiedy użytkownik będzie chciał podać blok, który nachodzi na inny to zostanie wyświetlony błąd. Do każdego nowo dodanego bloku, przypisane zostają trzy opcje:

- *Załaduj* - opcja ładuje do harmonogramu wszystkie wydarzenia, które znajdują się w obrębie czasowym bloku
- *Zmień* - umożliwia użytkownikowi edycję bloku m.in. przedział czasowy oraz notatki
- *Usuń* - usuwa blok

Dodatkowo znajduje się opcja *załaduj pełny*, który umożliwia załadowanie wszystkich dostępnych wydarzeń do harmonogramu.

Historia - jest to zakładka, służąca tylko i wyłącznie do przeglądania wydarzeń, które zostały już zarchiwizowane. Istnieje możliwość zmiany widoków, zmiany przedziału dat, które aktualnie są przeglądane oraz rozwinięcie szczegółów danego wydarzenia.

Harmonogram - główny element aplikacji klienckiej. Szczegóły widoku harmonogramu, oznaczone czerwonymi numerami, zostały pokazane na rysunku 4.1.

1) Przycisk, który ustawia w kalendarzu widok i datę tak, aby w przedziale znajdowała się aktualna data. W przypadku widoku ustawionego na *DZIEŃ* przenosi do konkretnego dnia.

2) Lista rozwijalna, z której użytkownik może wybrać widok kalendarza. W liście znajdują się opcje widoku dla dnia, tygodnia oraz miesiąca. Na zdjęciu przedstawiony jest widok tygodnia, a widok dnia można znaleźć na rysunku 4.14.

3) Pole tekstowe, które zawiera informację o tym, jakie wydarzenia są aktualnie ładowane do harmonogramu. Może to być konkretny blok i podany przedział czasowy, lub tak jak na rysunku 4.1, informacja o ładowaniu wszystkich wydarzeń.

4) Pole tekstowe, które informuje w jakim aktualnie przedziale dat znajduje się harmonogram. Dzięki temu rozwiązaniu, w jednym miejscu można szybko dowiedzieć się, w którym punkcie znajduje się kalendarz.

5) Dwa przyciski, które służą do zmiany daty w harmonogramie. Jeden służący do cofnięcia miejsca w którym znajduje się kalendarz, a drugi do przejścia w następny punkt. Dzięki temu, użytkownik może zmieniać przedział wyświetlany aktualnie na harmonogramie, na poprzedni lub następny. W momencie zmiany przedziału zostają pokazane wydarzenia, które się w nim znajdują.

6) Pojedyncza komórka znajdująca się w całej siatce harmonogramu. Podwójne wybranie lewego przycisku myszy w miejscu gdzie się znajduje, otworzy komponent odpowiedzialny za dodawanie nowego wydarzenia, który umiejscowiony jest na rysunku 4.3. W przypadku wykonania akcji na utworzonym wydarzeniu, zostanie otwarty mały panel, na którym pojawią się niektóre szczegóły wydarzenia jak przedstawione jest to na rysunku 4.2.



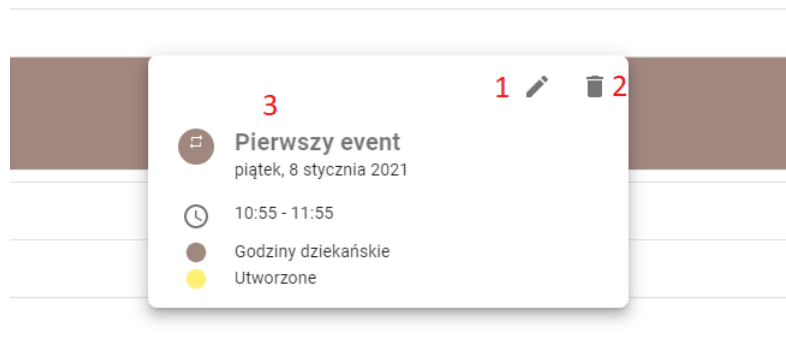
Rysunek 4.1: Harmonogram z zaznaczonymi elementami

Szczegóły widoku panelu informacyjnego dla pojedynczego wydarzenia zostały zaznaczone czerwonymi cyframi na rysunku 4.2.

1) Przycisk umożliwiający przejście do widoku edycji wydarzenia. Po wybraniu otworzy się zakładka, która wygląda identycznie jak ta służąca do dodawania, lecz będzie już wypełniona danymi.

2) Przycisk, który daje możliwość usunięcia wydarzenia. Po jego wybraniu zostanie wyświetlony element z opcjami w jaki sposób usunąć wydarzenie. Dla wydarzeń powtarzających się można usunąć: tylko wybrany element; wybrany element oraz te następujące po nim; całe wydarzenie.

3) Zbiór pól zawierających niektóre informacje o wydarzeniu, takie jak: tytuł, czas, typ oraz status.



Rysunek 4.2: Panel informujący o niektórych szczegółach wydarzenia

Następnym elementem harmonogramu jest zakładka umożliwiająca dodanie nowego wydarzenia, rysunek 4.3. Na rysunku zostały oznaczone ważniejsze elementy przy pomocy czerwonych numerów.

1) Pole tekstowe, służące do wprowadzania tytułu, który będzie widniał na wszystkich elementach wyświetlających informacje o wydarzeniu.

2) Dwa pola tekstowe z konkretnie ustawionym formatem, które służą do wprowadzania daty wydarzenia. Można wpisać ją ręcznie, bądź po naciśnięciu przycisku z ikoną kalendarza, wybrać z listy. Należy pamiętać, że data rozpoczęcia nie może znajdować się po dacie zakończenia.

3) Pole wyboru, które przechowuje informację o tym czy dane wydarzenie trwa przez cały dzień. W momencie wybrania tej opcji, zdarzenie jest wyświetlane w specjalnym miejscu na górze harmonogramu, które widoczne jest przez cały czas.

4) Opcja wyboru czy wydarzenie jest powtarzające czy też nie. Jeśli zostanie wybrane, pokazuje się wtedy zakładka *Powtarzaj*, która udostępnia opcje, definiujące warunki powtarzania wydarzenia.

5) Pole tekstowe w którym użytkownik może dodawać dowolne notatki. Mogą one wyglądać tak jak na załączonym obrazku 4.3, aczkolwiek nie wymagają ściśle określonego formatu. Konsument może dowolnie komentować swoje wydarzenia.

6) Lista rozwijalna, która zawiera w sobie wszystkie możliwe aktywności występujące na uczelni wyższej. Występują tam takie pola jak wykład, laboratorium, sprawozdanie, kolokwium, seminarium, godziny dziekańskie itd. W przypadku nie

zaznaczenia żadnej z opcji, przy tworzeniu zdarzenia zostanie automatycznie zaklasyfikowane jako inne.

7) Lista rozwijalna, która przechowuje informacje o wszystkich zdefiniowanych blokach. W samej liście znajduje się dodatkowy element *Bez bloku*, który mówi, że wydarzenie nie jest związane z żadnym blokiem. Użytkownik może dowolnie wybrać czy zajęcia lub aktywności mają znajdować się w konkretnym przedziale czasowym czy też występować bez bloku jako wolne wydarzenie. W momencie jeśli zostanie wybrana któraś z opcji zdefiniowanych przez użytkownika to w przypadku jeśli daty wydarzenia nie pokrywają się z przedziałem bloku to automatycznie zostaną przeniesione daty aktywności.

Aktualnie ładujesz wszystkie swoje wydarzenia.

×

ZAPISZ

Szczegóły

Teoria Układów Cyfrowych 1

2 05/01/2021 10 - 05/01/2021 11

3 ☐ Cały dzień ☒ Powtarzaj 4

Więcej informacji 5

Wydział: AEI
Grupa 2
Temat: Przerzutniki asynchroniczne

Rodzaj wydarzenia 6

Wykład

Wybierz gdzie ma występować wydarzenie 7

Bez bloku

Status 8

Utworzone

Powtarzaj

TYGODNIE 9

Powtarzaj co 1 10

NIEDZ. PON. WT. ŚR. CZW. PT. SOB.

Koniec powtarzania: 11

☒ Powtarzaj cały czas

☐ Po 13 powtórzeniach

☐ W dniu 05/01/2021 11:00 A

TYDZIEŃ

sob. 9

Pierwszy eve... 10:55 - 11:55

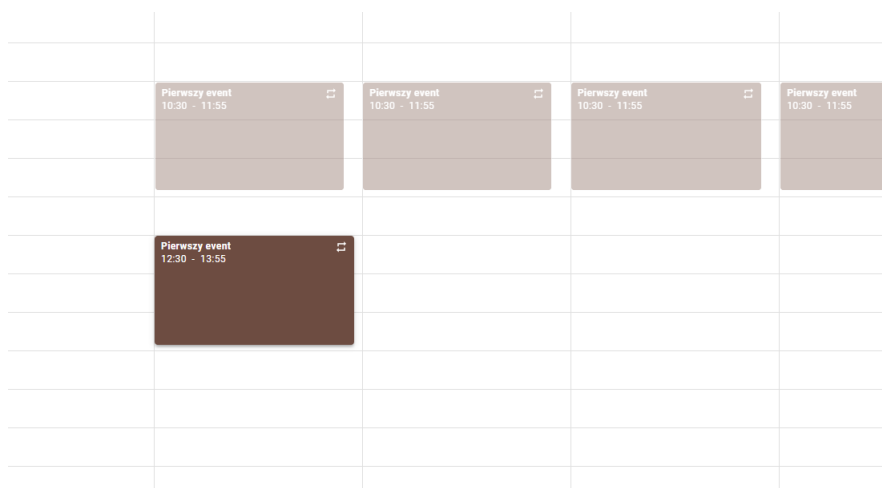
Rysunek 4.3: Zakładka tworzenia/edycji wydarzenia powtarzającego

8) Lista posiadająca możliwe statusy wydarzenia. Wszystkie statusy nadawane są automatycznie ale klient ma możliwość sprawdzenia jakie możliwości istnieją w aplikacji.

9) Lista rozwijalna gdzie znajdują się opcje z których można wybrać w jakich ramach wydarzenie ma się powtarzać. Mogą to być dni, tygodnie, miesiące lub lata. Na obrazku przedstawione zostały opcje dla tygodni.

10) Pola wyboru określające zasady co ile oraz w jaki sposób wydarzenie ma się powtarzać. Biorąc pod uwagę sytuację z rysunku 4.3, zostały ustalone konkretne opcje. Wydarzenie ma się powtarzać co tydzień, a w momencie wpisania np. cyfry 2, wydarzenie powtarzało by się co dwa tygodnie. Zostały też zdefiniowane konkretne dni powtarzania wydarzenia. Wybrany został poniedziałek, środa oraz piątek. Oznacza to, że tworzone jest wydarzenie, które występuje co tydzień w powyższe dni.

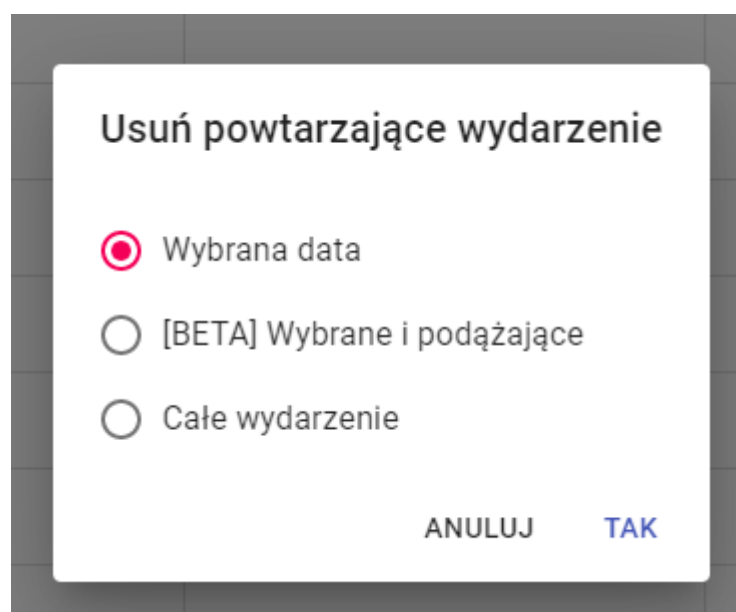
11) Pola wyboru definiujące warunek zakończenia wydarzenia. Zdarzenie może nie mieć końca, skończyć się po konkretnej liczbie powtórzeń lub po konkretnej dacie. Najlepiej opisać to na podstawie przykładu tygodni. W przypadku wybrania ilości powtórzeń wybierana jest liczba tygodni po jakich ma wydarzenie się zakończyć. Oznacza to, że jeżeli skonfigurujemy wydarzenie jak w punkcie 10, to będzie ono się powtarzać przez 13 tygodni tak jak jest to zawarte na zdjęciu. Po wyborze opcji zakończenia po dacie, powtarzalność ustanie w wybranym dniu.



Rysunek 4.4: Działania opcji, przeciągnij i upuść

W aplikacji zaimplementowany został system *przeciągnij i upuść*. Oznacza to, że użytkownik, może modyfikować przedział czasowy przy pomocy myszy komputerowej. Na rysunku 4.4 można zaobserwować, że wydarzenie, które zostało poruszone przy pomocy myszy, stało się bardziej przezroczyste, a to, które zachowało swój kolor, jest aktualnie przenoszone.

W momencie próby edycji lub usunięcia wydarzenia, wyświetlane są dodatkowe opcje zaprezentowane na rysunku 4.5. W momencie wyboru jednego zostaje dodana nowa identyczna aktywność, lecz bez parametrów powtarzania. Jeśli wybrana zostanie opcja całe wydarzenie, to w przypadku edycji początek wydarzenia zostanie usytuowany w miejscu do którego chciano je przenieść, a dla usunięcia zostanie usunięte ono całe. Opcja od wybranego i podążające jest dalej w fazie rozwoju, natomiast jest już funkcjonalna w pewnym stopniu. W momencie kiedy zostanie użyta to wydarzenie jest dzielone na dwa osobne i dla edycji, pierwsze z nich ma ustawione parametry na dzień aktualny, a drugie rozpoczyna się w miejscu wybranym. W przypadku usunięcia, jedno również przechodzi do dnia dzisiejszego, a drugie jest usuwane.

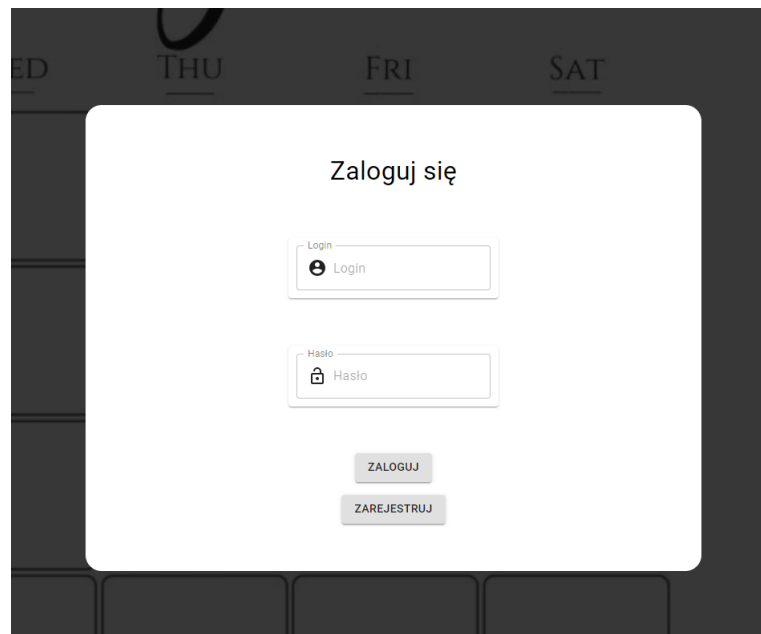


Rysunek 4.5: Opcje wyświetlane przy usuwaniu wydarzenia

4.4 Przykłady działania

W podrozdziale przedstawione są przykładowe zastosowania aplikacji.

4.4.1 Logowanie



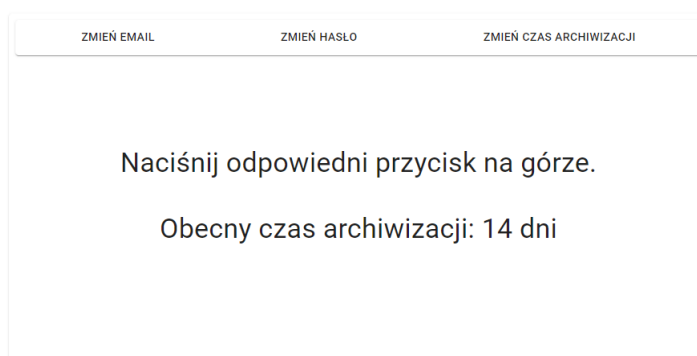
Rysunek 4.6: Widok strony logowania

Na rysunku 4.6 przedstawiona została strona logowania. Aby zalogować się na konto w aplikacji, należy uzupełnić pole 'Login' oraz 'Hasło' odpowiednio nazwą użytkownika oraz hasłem do istniejącego konta. Następnie nacisnąć przycisk 'ZALOGUJ'. W momencie jeśli dane zostaną poprawnie podane, użytkownik zostanie przekierowany do strony z instrukcją obsługi oraz udostępnione zostaną mu wszystkie możliwości aplikacji. W momencie podania błędnych danych zostanie wyświetlony komunikat z informacją o niepoprawności danych wejściowych tak jak jest to pokazane na rysunku 4.7. Po naciśnięciu przycisku 'ZAREJESTRUJ', użytkownik zostanie przekierowany do strony tworzenia konta.



Rysunek 4.7: Widok po błędnie wpisanych danych

4.4.2 Zmiana hasła



Rysunek 4.8: Ekran główny ustawień

Na rysunku 4.8 pokazano widok ustawień w początkowym stanie. Aby przejść do konkretnej opcji ustawień należy kliknąć w odpowiedni przycisk na pasku nad tekstem. W momencie przejścia do którejkolwiek ze stron zostaną wyświetlone dwa pola. Jedno do podania aktualnego hasła, a drugie do podania nowej wartości. Wygląd przedstawiony na rysunku 4.9. Po wprowadzeniu danych należy nacisnąć przycisk zmień. W momencie podania nieprawidłowych danych zostaną wyświetlone odpowiednie komunikaty informujące o błędzie.

ZMIEN EMAIL ZMIEN HASŁO ZMIEN CZAS ARCHIWIZACJI

Zmiana hasła

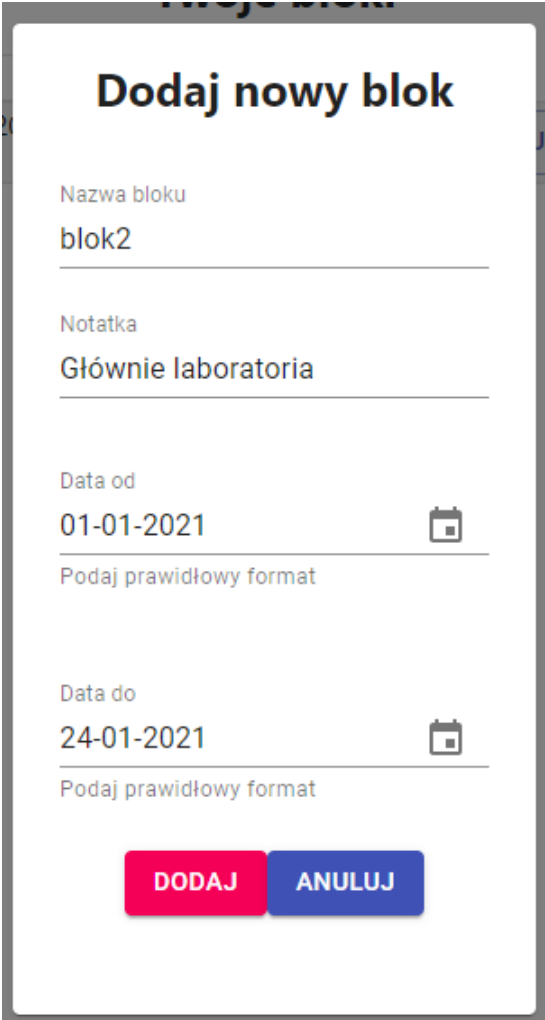
Nowe hasło

Aktualne hasło

ZMIEN

Rysunek 4.9: Komponent zmiany hasła

4.4.3 Dodanie nowego bloku



Dodaj nowy blok

Nazwa bloku
blok2

Notatka
Głównie laboratoria

Data od
01-01-2021

Podaj prawidłowy format

Data do
24-01-2021

Podaj prawidłowy format

DODAJ **ANULUJ**

Rysunek 4.10: Uzupełnienie komponentu danymi o bloku

Rysunek 4.10 ukazuje komponent do ustawiania danych nowego bloku. Należy uzupełnić pola danymi, w szczególności datę od, datę do oraz nazwę bloku. Notatki są opcjonalne. Po naciśnięciu przycisku dodaj, następuje walidacja danych, która w przypadku błędnych wyświetli użytkownikowi błąd, a w momencie jeśli dane będą poprawne, blok zostanie dodany. Zostaje zaktualizowana lista bloków, a finalny rezultat jest przedstawiony na rysunku 4.11.

Twoje bloki

Lp	Nazwa	Notatka	Data od	Data do			
1	blok1	Notka	14-12-2020	31-12-2020	<button>ZAŁADUJ</button>	<button>ZMIEN</button>	<button>USUŃ</button>
2	blok2	Głównie laboratoria	01-01-2021	24-01-2021	<button>ZAŁADUJ</button>	<button>ZMIEN</button>	<button>USUŃ</button>

DODAJ
ZAŁADUJ PEŁNY

Rysunek 4.11: Dodany blok na tle listy bloków

4.4.4 Dodanie wydarzeniami niepowtarzającego

×
ZAPISZ

Szczegóły

Fizyka

05/01/2021 08:30 AM

-

05/01/2021 09:00 AM

☐ Cały dzień
 ☐ Powtarzaj

Więcej informacji

Sala: 512
 Grupa: 2
 Temat: Wahadło harmoniczne

Rodzaj wydarzenia

●
Laboratorium
▼

Wybierz gdzie ma występować wydarzenie

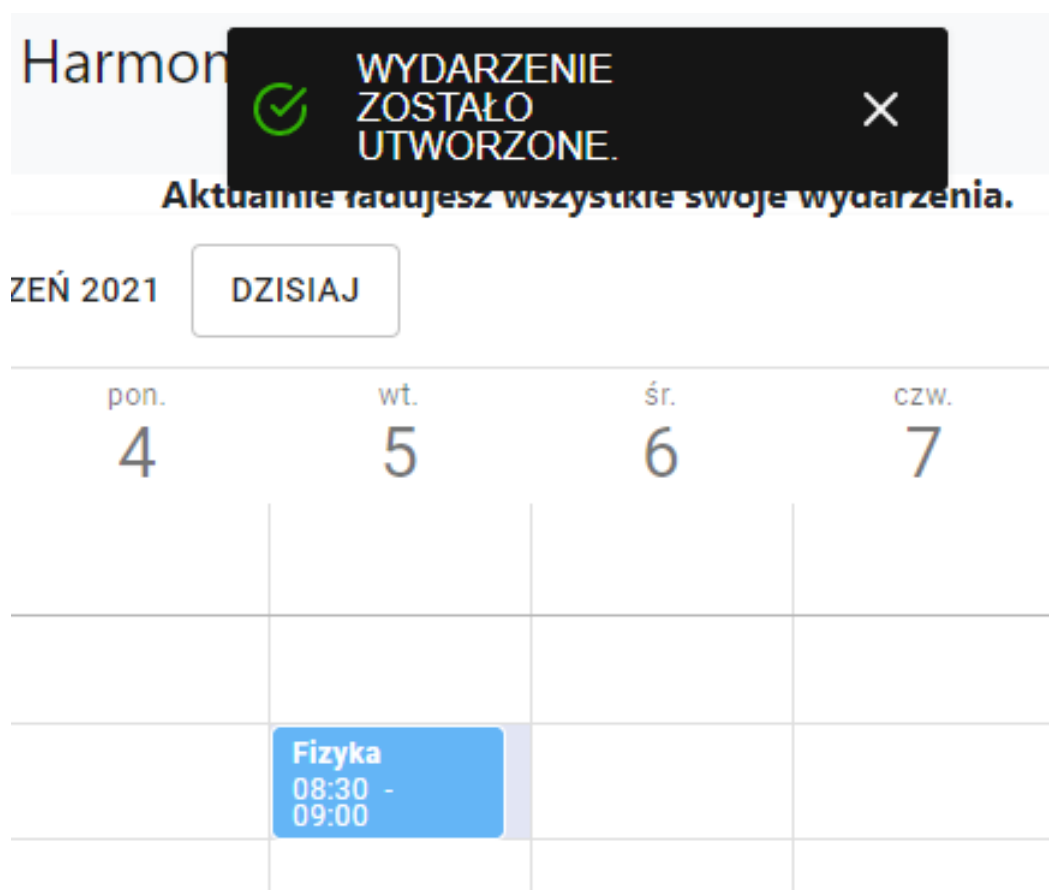
●
blok2
▼

Status

●
Utworzone
▼

Rysunek 4.12: Uzupełnienie komponentu danymi o wydarzeniu

Na rysunku 4.12 został przedstawiony formularz dodania wydarzenia. Aby go uruchomić należy dwukrotnie nacisnąć lewym przyciskiem myszy na pustą komórkę w kalendarzu. W momencie pozostawienia pustych pól, zostaną przypisane wartości domyślne. W rodzaju wydarzenia, można wybrać jego typ np. kolokwium, seminarium. Status uzupełniany jest automatycznie. Występowanie wydarzenia można wybrać czy wydarzenie, ma być w konkretnym bloku czy ma pozostać wydarzeniem nieblokowym. Daty można ustawiać poprzez wpisanie nowej, bądź wybranie ikonki kalendarza, a następnie wybranie odpowiedniej daty i godziny. Po uzupełnieniu danych należy zatwierdzić wybór przyciskiem zapisz. Jeśli dane są poprawne zostaje utworzone wydarzenie oraz wyświetlone na harmonogramie jak na rysunku 4.13.



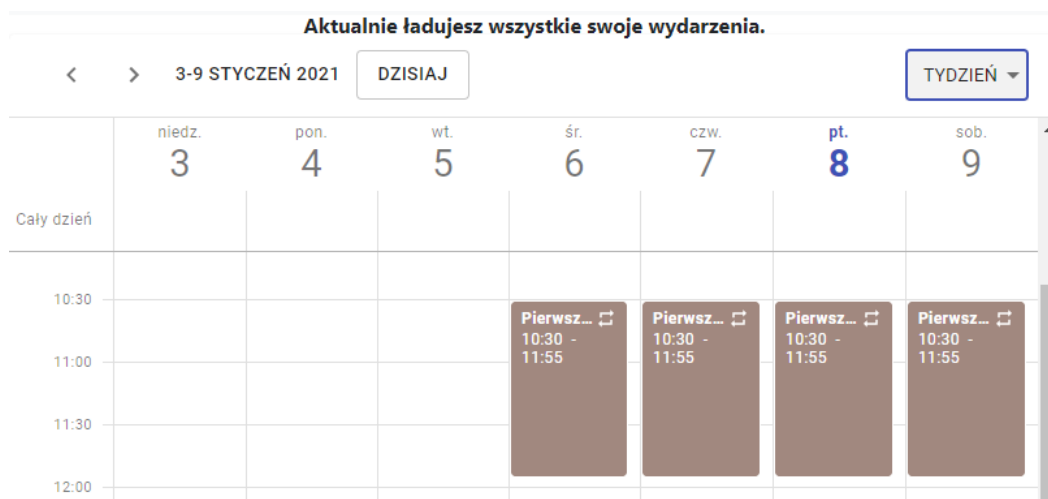
Rysunek 4.13: Dodane wydarzenie w harmonogramie

4.4.5 Widoki w harmonogramie



Rysunek 4.14: Wygląd dla widoku dnia

Użytkownik ma możliwość zmiany widoku harmonogramu, dostępne są opcje widoku dla dnia, tygodnia oraz miesiąca. Na rysunku 4.14 przedstawiony jest widok dla dnia, w którym na całej stronie widnieje plan wyłącznie z dnia aktualnego. W przypadku rysunku 4.15 ukazany jest widok tygodnia, który nie skupia się wyłącznie na aktualnej dacie ale ukazuje cały tydzień kalendarzowy.



Rysunek 4.15: Wygląd dla widoku tygodnia

4.5 Kwestie bezpieczeństwa

Do zabezpieczenia aplikacji zostało użyte Spring Security. Dostarcza ono narzędzia, które pozwalają na zabezpieczenie wytwarzanej aplikacji. W pracy został użyty standard JWT (ang. *JSON Web Token*). Określa on sposób bezpiecznego przesyłania danych pomiędzy aplikacją kliencką, a serwerem przy pomocy obiektów JSON [6]. Narzędzie to może zostać użyte do uwierzytelniania oraz autoryzacji przy czym w aplikacji występuje tylko jedna rola, zatem pełnić, będzie jedynie funkcję uwierzytelniającą. Pomiedzy aplikacją kliencką oraz serwerową jest przesyłany tzw. token, w którym znajdują się zaszyfrowane informacje na temat użytkownika. W aplikacji udostępnione zostały dwie metody, które nie wymagają autentykacji. Pierwszą z nich jest logowanie, które generuje nowy token w przypadku poprawnego zalogowania, a następnie przy każdym wysłanym zapytaniu jest on sprawdzany czy się nie przedawnił, bądź czy dane w nim zawarte są poprawne. Drugą jest rejestracja. Jest to punkt w którym można utworzyć nowe konto, natomiast nie wpływa ono na proces generowania tokenu. Dodaje nowego użytkownika do bazy danych, lecz nie ma powiązania z uwierzytelnieniem. W systemie, działanie uwierzytelniania jest procesem, który wymaga kilku kroków. Podczas logowania wysyłane jest zapytanie do kontrolera odpowiadającego za uwierzytelnianie. Następnie weryfikuje on czy użytkownik, o podanych danych istnieje w systemie. Jeśli tak to zostaje odesłany token uwierzytelniający, który identyfikuje użytkownika w systemie oraz daje mu możliwość korzystania z zabezpieczonych funkcji. Informacja odesłana z aplikacji serwerowej jest zapisywana w lokalnym schowku aplikacji klienckiej. W momencie wysyłania żądania po uwierzytelnieniu, do każdego zapytania dopisywany jest nagłówek z tokenem. W momencie jeśli token się przedawni, bądź będzie niezgodny z informacjami zawartymi na serwerze, serwer zwróci błąd w postaci statusu HTTP o numerze 403. Oznacza to, że serwer zabrania dostępu do strony, którą użytkownik chce otworzyć w aplikacji klienckiej. Zastosowanie JWT umożliwia późniejszą, łatwą rozbudowę aplikacji o kolejne role użytkowników.

Rozdział 5

Specyfikacja wewnętrzna

W tym rozdziale znajdują się informacje, które opisują rodzaj użytej architektury, bazę danych, biblioteki użyte w projekcie oraz opis ważniejszych komponentów, fragmentów kodu.

5.1 Architektura systemu

W pracy należy przygotować internetowe API (ang. *Application Programming Interface*), oparte o protokół HTTP (ang. *Hypertext Transfer Protocol*). Powinno dawać możliwość, uwierzytelniania użytkowników, zarządzania całym planem zajęć, zarządzania blokami. Dodatkowo należy przygotować bazę danych umożliwiającą przechowywanie informacji. Niezbędnym elementem jest również aplikacja przeglądarkowa, dzięki której użytkownik mógłby manipulować danymi. Jednym z celów jest zastosowanie wzorca architektonicznego MVC (ang. *model-view-controller*).

MVC jest to wzorzec projektowy, stosowany przy tworzeniu nowoczesnych systemów informatycznych. Wzorzec ten dzieli całą aplikację na trzy segmenty:

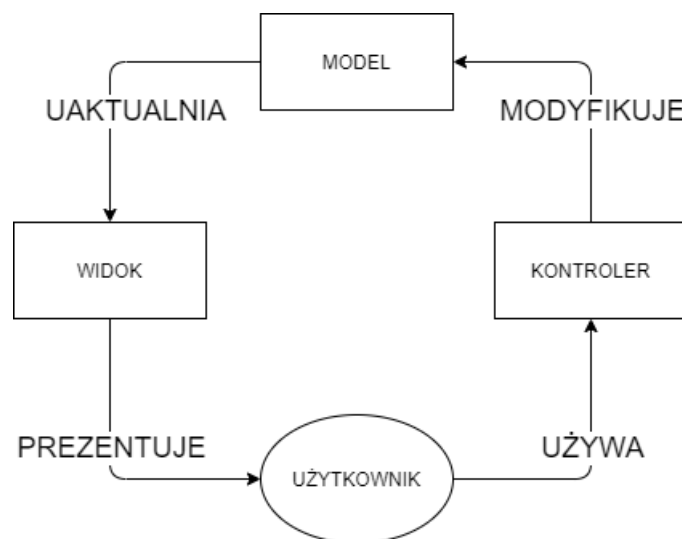
- model - część aplikacji, która implementuje logikę biznesową
- widok - jest to graficzny interfejs użytkownika, lecz łatwiej to zdefiniować jako wszelkiego rodzaju dane interpretacyjne

- kontroler - służy do koordynowania komunikacji pomiędzy warstwą widoku i modelem

Zazwyczaj widoki i kontrolery są konstruowane w stosunku 1:1, ponieważ każdy widok powinien mieć przynajmniej jeden kontroler do zapewnienia komunikacji z modelem [10]. Poszczególne warstwy działają jak adaptery, ponieważ przechodząc z jednej do drugiej, przygotowują dane do tego, aby mogły być w łatwy sposób wpisane do bazy danych. Z danych wejściowych wyciągane są wszystkie informacje niezbędne do zaktualizowania bazy danych [16]. Podobnie działa to w drugą stronę. Dane są opakowywane w konkretne warstwy danych, aby były proste w odczycie przez użytkownika systemu.

Podział na warstwy został użyty ze względu na chęć uporządkowania architektury. Dzięki temu zachowaniu, można oddzielić od siebie każdą z części systemu, a w przypadku chęci poprawy lub rozszerzenia oprogramowania nie jest wymagana edycja każdego z wydzielonych elementów.

Przepływ danych w systemie, stosującym wzorzec MVC, znajduje się na rysunku 5.1. Prosty schemat umożliwiający zrozumienie drogi danych w sposób abstrakcyjny.



Rysunek 5.1: Przepływ danych w architekturze MVC

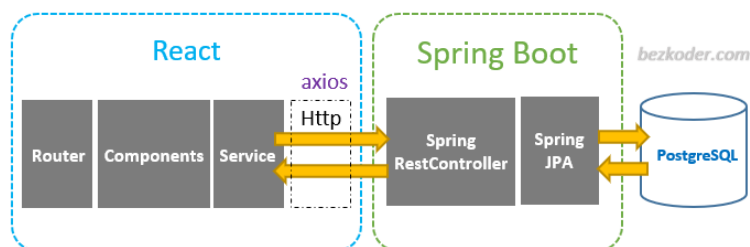
Bardziej szczegółowe działanie oraz tego jak przechodzi to przez konkretne punkty systemu można zaobserwować na rysunku 5.2. Zostały wyszczególnione

konkretne elementy architektury. React, to widok, Spring Boot kontroler oraz logika aplikacji, a PostgreSQL zawiera logikę biznesową.

W widoku znajdują się elementy, które służą do przekazywania informacji użytkownikowi. Element Router służy do wyznaczenia drogi w aplikacjach React. Pozwala on na przenoszenie się pomiędzy komponentami poprzez podanie ich ścieżek w menu wyboru. Components są to komponenty, które są rdzeniem widoku aplikacji [13]. Natomiast Service umożliwia przy pomocy odpowiednich bibliotek np. axios, komunikację z kontrolerami, zawierającymi się w Spring Boot. Do komunikacji używa protokołu HTTP.

W kontrolerze znajduje się Spring Rest Controller, który jest interfejsem wystawianym przez aplikację serwerową. W szczególności służy on do odbierania żądań od widoku. Rest, mówi Springowi, że wszystkie metody obsługi w kontrolerze, powinny zwracać pewną wartość. Powinna ona być przekazywana bezpośrednio w odpowiedzi, a nie zmieniana w modelu, a następnie przekazywana do widoku poprzez renderowanie [18]. Kolejny element to Spring JPA (ang. *Java Persistence API*). Narzędzie to pozwala na zarządzanie zawartością bazy danych za pomocą obiektów utworzonych w języku Java [1].

Model aplikacji jest dosyć prosty, ponieważ jest to baza danych z określonymi tabelami oraz adnotacjami.



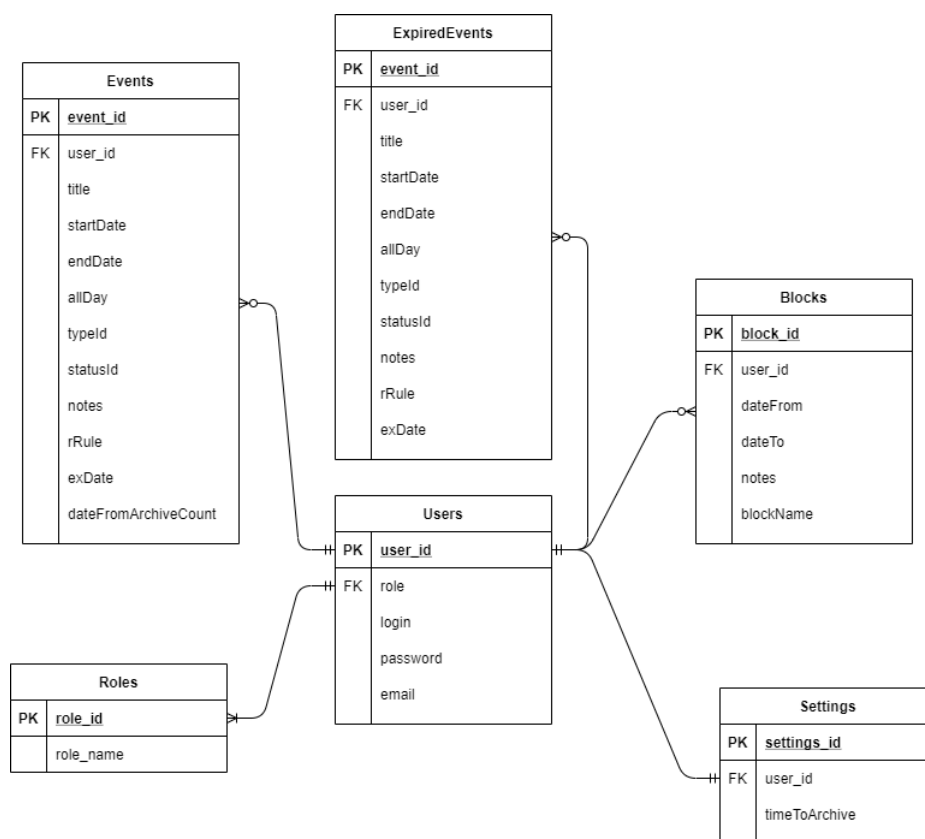
Rysunek 5.2: Szczegółowy przepływ danych w architekturze MVC
Źródło: <https://bezkoder.com/wp-content/uploads/2020/11/react-spring-boot-postgresql-crud-example-rest-api-architecture.png>

Przepływ danych odbywa się z widoku do kontrolera. Kontroler przetwarza dane na przyjazne dla modelu, po czym mu je przekazuje. Następnie dane z modelu wracają do kontrolera, a ten odsyła je do widoku. Następnie widok jest aktualizowany.

5.2 Opis bazy danych

Dla każdego projektu, który zakłada użycie aplikacji serwerowej oraz przeglądarkowej jedną z niezbędnych rzeczy jest miejsce w którym należało by niektóre przetworzone dane, przechowywać, żeby móc je użyć ponownie. W pracy zostało użyte narzędzie, które służy do utrzymywania relacyjnych baz danych, jakim jest PostgreSQL. Dane znajdujące się bazie relacyjnej pozostają w relacjach, a dla użytkowników przedstawiane są w formacie tabel. Każda relacja zbudowana jest z krotek oraz atrybutów. Wszystkie rekordy posiadają unikatowe pole, dzięki którym można je łatwo zidentyfikować [15].

Na rysunku 5.3 została przedstawiona baza danych, która jest obsługiwana przez system. Ze względu na to, że jest relacyjna, toteż każda z tabel powinna mieć jakiegokolwiek połączenie w celu ustalenia do kogo przypisane są konkretne dane.



Rysunek 5.3: Model relacyjnej bazy danych

Każda z tabel posiada ściśle określoną rolę, oraz pola, które są niezbędne do poprawnego funkcjonowania całej aplikacji. Role każdej z tabel w systemie:

- *Roles* - przechowuje dane o rolach jakie znajdują się w systemie, została dodana w celu przechowywania danych o roli użytkownika i ułatwienia późniejszego rozwoju aplikacji
- *Settings* - przechowuje informacje o użytkowniku, który jest posiadaczem konkretnych ustawień. Do tej pory w aplikacji została dodana opcja, która przechowuje ustalony przez użytkownika czas, po którym wydarzenia są archiwizowane
- *Users* - w tabeli przechowywane są informacje o użytkowniku, takie jak login, hasło oraz informacja o numerze ID (ang. *index*) roli jaką posiada w systemie
- *Blocks* - służy do przechowywania informacji o zdefiniowanych blokach, nazwa bloku, przedział czasowy, notatka oraz kto dany blok utworzył
- *ExpiredEvents* - jedna z większych tabel, która przechowuje informacje na temat wydarzeń, które zostały już zarchiwizowane. Dane jakie posiada to właściciel wydarzenia, tytuł, przedział czasowy, czy jest wydarzeniem całonocnym, dane o typie, statusie, notatki, oraz dwa pola, które służą do definicji wydarzenia powtarzającego. *RRule* definiuje powtarzalność wydarzenia, natomiast *exDate* przechowuje wykluczone daty
- *Events* - zawiera prawie te same informacje co *ExpiredEvents*, natomiast posiada dodatkowe pole określające datę końcową wydarzenia, od której liczony jest czas do zarchiwizowania wydarzenia

Ważnym pytaniem jest, czy tabela *Blocks* oraz *Events* nie powinna być połączona relacją? Otóż, wydarzenia, które są tworzone w kreatorze, mogą być definiowane jako blokowe lub jako nie posiadające bloku. W momencie jeśli użytkownik próbowałby załadować dane z konkretnego bloku to pomijane byłyby informacje o danych, nie zawierających się w tym bloku ale znajdujących się w odpowiadającym przedziale czasowym. Dlatego też, aby uniknąć sytuacji, że wydarzenia się nie pokazują, została pominięta relacja, a sama filtracja polega na sprawdzeniu dat zdarzeń, które posiada użytkownik.

W przypadku jeśli użytkownik chciałby utworzyć wydarzenie powtarzające przez dwadzieścia dni w miesiącu, baza danych musiałaby zostać uzupełniona o kolejne dwadzieścia pozycji. W celu odciążenia, od nadmiernego tworzenia wydarzeń, został zastosowany mechanizm *recurrence rule*, który przechowuje informacje na temat powtarzalności wydarzenia w całym harmonogramie. Stosując ten mechanizm, można w łatwy sposób, łącząc, kolejne dane typu *String*, utworzyć reguły, które informują o rodzajach powtarzalności [2]. Informacja przechowywana jest w polu *rRule* w tabelach wydarzeń.

5.3 Użyte biblioteki

W tym podrozdziale zostały omówione biblioteki, które w znacznym stopniu wspomogły wytwarzanie oprogramowania, oraz są jednymi z głównych jeśli chodzi o całość systemu.

5.3.1 Material-UI

Biblioteka zawierająca darmowe oraz gotowe komponenty. Przyspiesza w znacznym stopniu tworzenie oprogramowania, dzięki udostępnionym gotowym rozwiązaniom. Dodatkowo udostępnia dokumentację oraz przykładowe zastosowania konkretnych elementów.

5.3.2 Bootstrap

Podobnie do material-UI jest biblioteką zawierającą gotowe komponenty, które mogą być modyfikowane w trakcie ich używania.

5.3.3 Axios

Biblioteka stworzona do komunikacji z API. Używa się jej łatwiej niż bibliotekę Fetch ze względu na to, że wszystkie metody HTTP są dostępne i nie trzeba dopisywać, żadnych kolejnych parametrów z tym związanych. Umożliwia wysyłanie żądań wymagających uwierzytelnienia oraz w bardzo prosty sposób odbiera dane od serwera przekształcając je na odpowiednie formaty.

5.3.4 Redux toolkit

Biblioteka, która ułatwia korzystanie z Redux. Konkretniej służy do zarządzania stanem w aplikacji. Umożliwia utworzenie globalnego schowka w aplikacji, dzięki czemu można z dowolnego komponentu używać elementów zdefiniowanych w innej części aplikacji.

5.3.5 DevExtreme React Scheduler

Biblioteka firmy devexpress, która udostępnia komponent kalendarza oraz komponenty w nim zawarte, które w prosty sposób wyświetlają informację. Została wybrana ze względu na obsługę *recurrence rule* z dokumentu [2] oraz ze względu na łatwą w zrozumieniu szatę graficzną.

5.3.6 Spring-Boot-Starter

Biblioteka dostarczająca biblioteki, które posiadają związek z obsługą mechanizmów webowych. Dostarcza biblioteki do testowania, serwerowe, czy też bezpieczeństwa.

5.4 Ważniejsze komponenty

Jednym z najważniejszych komponentów całego systemu jest kalendarz, znajdujący się w aplikacji klienckiej. Sam kalendarz daje wiele możliwości, dzięki którym, aplikacja może spełniać wymagania funkcjonalne. Przykładowy wygląd harmonogramu można zaobserwować na rysunku 4.1. Większość funkcji aplikacji zależy od tego modułu są to np. dodawanie, edycja, usuwanie wydarzeń. Najważniejszą częścią tego elementu jest opcja przedstawiona na rysunku 4.3. To dzięki niej cały kalendarz może funkcjonować. Fragment komponentu znajduje się na rysunku 5.4. W tym kodzie przedstawiony jest fragment odpowiadający renderowaniu elementów na ekranie. Można zauważyć, że tworzenie samego kalendarza składa się z dołączania komponentów postronnych do samego komponentu głównego. Łatwo można dodawać kolejne elementy, aczkolwiek powinny to być odpowiednio przemyślane elementy z biblioteki w której zawiera się sam kalendarz.

Kolejnym z ważniejszych komponentów jest sam proces logowania, nie wymaga on dużego nakładu pracy po stronie klienckiej, natomiast po stronie serwerowej potrzebuje dodatkowej konfiguracji przedstawionej na rysunku 5.5. W przypadku występowania jednej roli nie wygląda on na skomplikowany, natomiast dla wysoko rozwiniętego systemu, z dziesiątkami ról, może okazać się on bardziej zawiły. Dodatkowo, przy każdym logowaniu musi być generowany token uwierzytelniający dla użytkownika, aby mógł się poruszać w systemie. Metoda umożliwiająca uwierzytelnienie użytkownika wygląda tak jak na rysunku 5.6. Podczas wysłania żądania logowania, sprawdzane jest czy podane dane są zgodne z przechowywanymi, następnie w przypadku prawidłowych danych generowany jest token, który jest zapisywany w kontekście oraz wysyłany do aplikacji klienckiej.

5.5 Wybrane fragmenty kodu

```

148     return (
149       <Paper style={{ height: "89vh", justifyContent: "center" }}>
150         <Scheduler
151           data={this.props.events}
152           height={"100%"}
153           locale={locale}
154         >
155           <ViewState
156             defaultCurrentDate={currentDate}
157             onCurrentDateChange={this.currentDateChange}
158             defaultCurrentViewName="week"
159           />
160           <EditingState
161             onCommitChanges={this.commitChanges}
162             addedAppointment={addedAppointment}
163             onAddedAppointmentChange={this.changeAddedAppointment}
164             appointmentChanges={appointmentChanges}
165             onAppointmentChangesChange={this.changeAppointmentChanges}
166             editingAppointment={editingAppointment}
167             onEditingAppointmentChange={this.changeEditingAppointment}
168           />
169           <EditRecurrenceMenu messages={getEditRecurrenceMessages(locale)} />
170           <DayView startDayHour={7} endDayHour={21} displayName="Dzień" />
171           <WeekView
172             name="week"
173             startDayHour={7}
174             endDayHour={21}
175             displayName="Tydzień"
176           />
177           <MonthView displayName="Miesiąc" />
178           <Toolbar />
179           <ViewSwitcher />
180           <TodayButton messages={getTodayButtonMessages(locale)} />
181           <DateNavigator />
182           <ConfirmationDialog
183             messages={getConfirmationDialogMessages(locale)}
184           />
185           <Appointments />
186           <AppointmentTooltip showOpenButton showDeleteButton />
187           <AppointmentForm dateEditorComponent={DateEditor} messages={getAppointmentFormMessages(locale)} />
188           <AllDayPanel messages={getAllDayMessages(locale)} />
189           <DragDropProvider />
190           <Resources data={resources} mainResourceName="typeId" />
191         </Scheduler>
192       </Paper>
193     );

```

Rysunek 5.4: Fragment kodu odpowiedzialnego za komponent kalendarza

Na rysunku 5.4 można zaobserwować kod, komponentu renderujący wygląd kalendarza. Widać, że samo budowanie kalendarza składa się z dokładania do jego wnętrza kolejnych modułów. Rozwiązanie to ułatwia edycję samego wyglądu i umożliwia rozdzielenie na pliki konkretnych elementów. Do każdego komponentu zawierającego wyświetlane nazwy, zostają dodane własne tłumaczenia, poprzez przekazanie listy do pola *messages*.

```
25      @Override
26      protected void configure(final HttpSecurity http) throws Exception {
27          http.csrf().disable().authorizeRequests()
28              .antMatchers(...antPatterns: "/login", "/register").permitAll()
29              .antMatchers(...antPatterns: "/api/**").hasAnyRole(...roles: "USER")
30              .anyRequest().authenticated()
31              .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
32
33          http.addFilterBefore(this.authenticationTokenFilterBean(), UsernamePasswordAuthenticationFilter.class);
34      }
```

Rysunek 5.5: Fragment kodu odpowiadający konfiguracji zabezpieczeń

Kod na rysunku 5.5 przedstawia konfigurację, niezbędną do używania uwierzytelniania w systemie. Ustawiane są tutaj warunki mówiące, które punkty systemu zostaną udostępnione dla konkretnej roli użytkownika. W tym przypadku można zaobserwować, że do punktów końcowych aplikacji serwerowej o nazwach */login* i */register*, dostęp mają wszyscy. Niezależnie od tego czy jest to użytkownik zalogowany, czy też gość nieposiadający konta. Natomiast w dalszej części konfiguracji zaznaczone jest, że wszystkie punkty końcowe, których początkową frazą jest */api* mogą zostać wywołane tylko i wyłącznie przez użytkownika z rolą *USER*.

```
45      @RequestMapping(method = RequestMethod.POST, value = "/login")
46      public ResponseEntity<?> login(@RequestBody final LogRegUserDTO userDTO) throws Exception {
47
48          System.out.println(userDTO.getLogin() + " " + userDTO.getPassword());
49          final User user = this.userService.getUserByLogin(userDTO.getLogin());
50
51          final Authentication authentication;
52
53          try {
54              authentication = this.authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(userDTO.getLogin(), userDTO.getPassword()));
55              SecurityContextHolder.getContext().setAuthentication(authentication);
56          } catch (final BadCredentialsException e) {
57              throw new Exception("Bad login creds", e);
58          }
59
60          final String token = this.jwtUtil.generateToken(authentication);
61
62          final String role;
63
64          this.eventService.moveEventsToExpiredEventsWhenReachArchiveTime();
65          role = user.getRole().getName();
66
67          return ResponseEntity.ok(new AuthenticationResponse(token, role));
68      }
```

Rysunek 5.6: Fragment kodu odpowiadający za uwierzytelnianie użytkownika w systemie

Rysunek 5.6 przedstawia punkt końcowy kontrolera aplikacji, odpowiadający za

wygenerowanie tokenu uwierzytelniającego dla użytkownika. W bloku *try* wywołana jest metoda służąca do uwierzytelnienia użytkownika. W momencie podania poprawnych danych do kontekstu zostają dopisane dane o autentykacji klienta, a w przeciwnym wypadku rzucany jest wyjątek mówiący o złych parametrach. Po wyrzuceniu wyjątku odsyłana jest wiadomość ze statusem 403 oznaczającym, że serwer zrozumiał żądanie ale nie dał dostępu. Po pomyślnej autentykacji do użytkownika odsyłany jest token uwierzytelniający JWT. Podczas logowania sprawdzane są również wydarzenia czy nie są przedawnione, a w momencie przedawnienia są przenoszone do historii wydarzeń.

Rozdział 6

Weryfikacja i walidacja

Rozdział jest poświęcony opisaniu przeprowadzonych testów oraz odnalezionych i usuniętych błędów.

6.1 Sposoby testowania aplikacji

W podrozdziale przedstawione zostały wszystkie testy przeprowadzone w projekcie.

6.1.1 Testy integracyjne

Testy integracyjne są to działania podejmowane w celu sprawdzenia poprawności kooperacji oraz wzajemnych oddziaływań kilku modułów. Pod nazwą "moduł" należy rozumieć osobną część całego systemu, odrębną część kodu, który jest niezależny od komponentów ale jest ściśle powiązany z całością projektu [17].

W powyższej pracy do zapewnienia poprawnego działania aplikacji zostały zaimplementowane testy integracyjne. Są to testy, które sprawdzają działanie większości funkcji systemu. Testowaniu poddawane są kontrolery czyli moduły, które podejmują interakcję na rzecz działań wykonywanych na widoku. Otrzymują one dane, jeśli jest to wymagane to je obrabiają na sposób ustalony w kodzie, a następnie odsyłają do modelu.

Do testowania została użyta technologia Spring Boot Test [8], która zawiera w sobie wiele przydatnych narzędzi pozwalających na testowanie oprogramowania.

Do samego testowania została powołana odrębna baza danych oparta na postgresql oraz użyto moduł Spring Web Test, zawierający się w Springu. Powstałe testy uruchamiane są na stałej, lokalnej bazie danych, do której ładowane są początkowe dane przy pomocy Spring Boot *CommandLineRunner* [3]. Użycie tego narzędzia umożliwia ominięcie pisania ręcznie komend SQL osobno w pliku. *CommandLineRunner* daje możliwość używania metod modelu czyli np. metod z repozytoriów czy też serwisów i uruchamiany jest na samym początku po wystartowaniu programu. Dzięki modułowi Spring Web Test dopuszczone zostało testowanie modułów komponentów wymagających autoryzacji. Testy zostały napisane w sposób nieinwazyjny, tu autor ma na myśli użycie adnotacji *@Transactional* co oznacza, że po każdym wykonanym teście, dane, które zostały usunięte zostają przywrócone, a te zmodyfikowane cofnięte do poprzedniego stanu.

6.1.2 Testy systemowe

Testy systemowe obejmują sobą wymagania funkcjonalne jak i нефункционалне. Jest to punkt w którym ocenia się produkt całościowo. Nie kładzie się nacisku na architekturę czy też sam kod aplikacji. Kluczem do napisania dobrych testów jest wprowadzenie ich w środowisku jak najbardziej przypominającym środowisko produkcyjne działającej aplikacji. A weryfikacja w takich warunkach zmniejsza ryzyko pominięcia błędów oraz problemów, które mogą występować ze względu na różnice w środowiskach [17].

Testy systemowe prowadzone były manualnie, ponieważ dużo szybciej i prościej było przejść kilka scenariuszy danej funkcji ręcznie aniżeli pisać osobne testy automatyczne. Testy te były prowadzone na bieżąco wraz z powiększającymi się możliwościami aplikacji. Aplikacja serwerowa była testowana przy pomocy aplikacji Postman, dzięki, której bez posiadania strony klienckiej systemu, można wysłać zapytania i porównywać otrzymane odpowiedzi z oczekiwaniami. Aplikacja kliencka była testowana od razu w momencie implementacji, gdzie każdy widok strony oraz element funkcjonalny taki jak np. przycisk były weryfikowane od razu po dodaniu. Do usprawnienia testowania została użyta wbudowana opcja przeglądarki *zbadaj*. Umożliwia ona w pewnym stopniu podgląd komponentów aktualnie wyświetlanych na ekranie, pokazuje statusy wysyłanych zapytań oraz umożliwia

sprawdzenie komunikatów jakie serwer wysyła do konsoli przeglądarki.

Aplikacja kliencka została przetestowana na większości popularnych przeglądarek (Chrome, Mozilla Firefox, Opera).

6.2 Wykryte i usunięte błędy

Po części serwerowej największe błędy występowały podczas przeliczania daty zakończenia wydarzenia, od której jest liczony czas po jakim dane wydarzenie jest archiwizowane. Ze względu na występowanie pola warunków rekurencyjnych w postaci tekstowej. Przy każdej zmianie, bądź dodaniu wydarzenia występowała potrzeba od nowa przeliczyć czas zakończenia wydarzenia. W szczególności błędy zostały zaobserwowane w momencie testowania aplikacji, gdzie po dogłębnej analizie okazało się, że końcowa data była źle przeliczana. Algorytm dekodujący pole warunków rekurencyjnych posiadał nieliczne błędy, aczkolwiek udało się je usunąć, poprzez refaktoryzację kodu oraz zmianę niektórych instrukcji warunkowych.

Kolejnym problemem było rejestrowanie użytkowników, a następnie próba zalogowania. Proces rejestracji przebiegał pomyślnie, natomiast po wpisaniu identycznych danych podczas logowania, aplikacja zwracała błąd, o błędnych danych. Problemem okazało się szyfrowanie, oraz szyfrator. Podczas logowania odczytywany był inny szyfrator niż obecny w kontekście Springa. Algorytmy szyfrujące nie pokrywały się, co prowadziło do uzyskiwania nieprawidłowych danych.

Po stronie klienckiej problemem okazał się sam komponent odpowiedzialny za wyświetlanie oraz aktualizację kalendarza. Jego stan był aktualizowany za każdym razem po wykonaniu jakiegokolwiek zmiany i miało to wpływ na wywołanie funkcji odpowiedzialnych za komunikację z serwerem. Zapytania w dużej mierze były wysyłane zduplikowane przez co aplikacja serwerowa nie mogła poprawnie modyfikować bazy danych. W celu poprawienia oraz naprawy błędu, została usunięta część aktualizująca stan aplikacji, a mianowicie listy wewnątrz komponentu harmonogramu. Został dodany nowy komponent, który aktualizował stany wydarzeń poza samym modułem kalendarza. Zarządza przekazywanymi danymi przez harmonogram. W momencie wykonywania działań w komponencie klasowym harmonogramu, zostają wywoływane funkcje aktualizujące z komponentu nadrzędnego. Dzięki temu uniknięto podwajania zapytań oraz ich niepotrzebnego wywołania.

Rozdział 7

Podsumowanie i wnioski

Stworzona aplikacja spełnia wcześniej zdefiniowane wymagania. Projekt utworzony w ramach pracy ma za zadanie wspomóc zarządzanie własnym planem zajęć studentów i nauczycieli akademickich. Dzięki zastosowaniu wzorca MVC, udało się stworzyć aplikację, która jest prosta w rozbudowie oraz łatwa w zrozumieniu pod kątem ewentualnych modyfikacji. Kolejną zaletą wzorca jest to, że podczas zmiany kodu w aplikacji klienckiej, nie są wymagane zmiany w kodzie aplikacji serwerowej, dlatego można uniknąć lawinowego zmieniania kodu. Cały system został utworzony przy pomocy nowoczesnych technologii. Wszystkie wymagania funkcjonalne oraz niefunkcjonalne zostały zawarte w wytworzonym oprogramowaniu.

Dzięki protokołowi HTTP zapewnienie komunikacji pomiędzy warstwą serwerową, a kliencką okazało się łatwe i bezproblemowe.

Stosując narzędzie JWT zostało wprowadzone do systemu zabezpieczenie funkcji, które mają być udostępnione tylko dla wybranych użytkowników. Pod nazwą wybrani użytkownicy kryją się wszystkie zalogowane osoby. Ze względu na zastosowanie darmowych narzędzi do produkowania oprogramowania, każda osoba, może aplikację pobrać oraz modyfikować pod kątem własnych upodobań. A ponieważ cały kod źródłowy znajduje się w zdalnym repozytorium GitHub, daje to możliwość dostępu do niego w bardzo prosty sposób.

Dzięki skorzystaniu z testowania automatycznego, a konkretniej testów integracyjnych, można było w szybki sposób sprawdzić działanie aplikacji serwerowej oraz jej komunikacji z zewnętrznymi zapytaniami. Testy automatyczne zapewniają

szybką walidację napisanego kodu. Nie trzeba przechodzić tych samych ścieżek po kolei testując aplikację manualnie ale przy pomocy jednego przycisku zostaje uruchomiana lista testów do wykonania, gdzie w przypadku negatywnego wyniku wyświetlany jest odpowiedni komunikat.

Aplikacja daje dużo możliwości rozwoju. Żeby skutecznie zapewnić działanie wszystkich komponentów podczas awarii jednego z nich można przejść na architekturę mikroservisową. Można dopisać aplikację mobilną obsługującą polecenia serwera oraz umożliwić użytkownikowi nie tylko podgląd ale i edycję planu zajęć. Kolejnym punktem rozwojowym może być umożliwienie dodawania do znajomych, dodanie czatu, czy też przeglądanie planów zajęć innych jeśli je udostępnią. Dodanie możliwości ustawiania planu zajęć jako prywatny lub publiczny. Wytworzenie funkcji do przywracania hasła użytkownika, poprzez dodanie systemu obsługi wiadomości email.

Autor dzięki pracy, mógł szerzej zapoznać się z narzędziami umożliwiającymi pisanie aplikacji serwerowej takimi jak Spring Boot. Od podstaw poznał narzędzie React do pisania aplikacji przeglądarkowych oraz poszerzył wiedzę z zakresu baz danych. Dodatkowym plusem jest poznanie narzędzia JWT odpowiedzialnego za bezpieczeństwo.

Bibliografia

- [1] <https://www.samouczekprogramisty.pl/pogodynka-jpa-i-spring-data/>. [data dostępu: 2020-01-05].
- [2] Dokumentacja icalendar. <https://icalendar.org/iCalendar-RFC-5545/3-8-5-3-recurrence-rule.html>. [data dostępu: 2020-01-05].
- [3] Dokumentacja spring commandlinerunner. <https://docs.spring.io/spring-boot/docs/current/api/org.springframework.boot.CommandLineRunner.html>. [data dostępu: 2020-12-29].
- [4] Strona internetowa github. <https://github.com/>. [data dostępu: 2021-01-04].
- [5] Strona internetowa heroku. <https://www.heroku.com/>. [data dostępu: 2020-01-08].
- [6] Strona internetowa jwt. <https://www.unity.pl/blog/jak-napisac-aplikacje-ktora-pozwoli-na-uwierzytelnienie-uzytkownika-wykorzystujaco> [data dostępu: 2020-12-31].
- [7] Strona internetowa react. <https://pl.reactjs.org/>. [data dostępu: 2020-12-29].
- [8] Strona internetowa spring boot test. <https://spring.io/guides/gs/testing-web/>. [data dostępu: 2020-12-29].
- [9] Strona internetowa z informacjami gradle. <https://www.samouczekprogramisty.pl/pierwszy-projekt-z-gradle/>. [data dostępu: 2020-12-30].

-
- [10] Igor Barbarić. *Design Patterns in Object-Oriented ABAP*. SAP PRESS, [b.m.], 2009.
 - [11] Przemysław Bykowski. Strona internetowa z informacjami spring. <https://bykowski.pl/convention-over-configuration/>. [data dostępu: 2020-12-30].
 - [12] Mike Cohn. *Agile estimating and planning*. Pearson Education Inc., b.m.w, 2005.
 - [13] Cory Gackenhaimer. *Introduction to React*. Springer Science, Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013, 2015.
 - [14] Włodzimierz Gajda. *Git. Rozproszony system kontroli wersji*. Helion, ul. Kościuszki 1c, 44-100 Gliwice, 2013.
 - [15] Michael J. Hernandez. *Projektowanie baz danych dla każdego. Przewodnik krok po kroku*. Helion, ul. Kościuszki 1c, 44-100 GLIWICE, 2014.
 - [16] Robert Cecil Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson Education Inc., [b.m.], 2017.
 - [17] Rafał Pawlak. *TESTOWANIE OPROGRAMOWANIA*. Helion, ul. Kościuszki 1c, 44-100 Gliwice, 2014.
 - [18] Craig Walls. *Spring in action*. Manning Publications Co., 20 Baldwin Road, PO Box 761, Shelter Island, NY 11964, 2019.

Dodatki

Spis skrótów i symboli

MVC model – widok – kontroler (ang. *model-view-controller*)

WebAPI ang. *Web Application Programming Interfaces*

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

JSON ang. *JavaScript Object Notation*

DOM ang. *Document Object Model*

JDK ang. *Java Development Kit*

JWT ang. *JSON Web Token*

REST ang. *Representational State Transfer*

API ang. *Application Programming Interface*

JPA ang. *Java Persistence API*

ID ang. *Index*

Zawartość dołączonej płyty

Do pracy dołączona jest płyta CD z następującą zawartością:

- praca (źródła \LaTeX owe i końcowa wersja w pdf),
- źródła programu,
- dane testowe.

Spis rysunków

3.1	Diagram przypadków użycia dla gościa	9
3.2	Diagram przypadków użycia dla użytkownika	10
4.1	Harmonogram z zaznaczonymi elementami	21
4.2	Panel informujący o niektórych szczegółach wydarzenia	22
4.3	Zakładka tworzenia/edycji wydarzenia powtarzającego	23
4.4	Działania opcji, przeciągnij i upuść	24
4.5	Opcje wyświetlane przy usuwaniu wydarzenia	25
4.6	Widok strony logowania	26
4.7	Widok po błędnie wpisanych danych	27
4.8	Ekran główny ustawień	27
4.9	Komponent zmiany hasła	28
4.10	Uzupełnienie komponentu danymi o bloku	29
4.11	Dodany blok na tle listy bloków	30
4.12	Uzupełnienie komponentu danymi o wydarzeniu	30
4.13	Dodane wydarzenie w harmonogramie	31
4.14	Wygląd dla widoku dnia	32
4.15	Wygląd dla widoku tygodnia	32
5.1	Przepływ danych w architekturze MVC	36
5.2	Szczegółowy przepływ danych w architekturze MVC Źródło: https://bezkoder.com/wp-content/uploads/2020/11/react-spring-boot-postgresql-crud-example-rest-api-architecture.png	37
5.3	Model relacyjnej bazy danych	38
5.4	Fragment kodu odpowiedzialnego za komponent kalendarza	43

5.5	Fragment kodu odpowiadający konfiguracji zabezpieczeń	44
5.6	Fragment kodu odpowiadający za uwierzytelnianie użytkownika w systemie	44