Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Inzynieria systemow informatycznych

Aplikacja internetowa do rozliczania wspólnych wydatków

Jarosław Glegoła
Numer albumu 293092

promotor
dr inż. Roman Podraza

WARSZAWA 2021

Aplikacja internetowa do rozliczania wspólnych wydatków

Streszczenie. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Słowa kluczowe: XXX, XXX, XXX

Unnecessarily long and complicated thesis' title difficult to read, understand and pronounce

Abstract. As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves; as I have shown elsewhere, the phenomena should only be used as a canon for our understanding. The paralogisms of practical reason are what first give rise to the architectonic of practical reason. As will easily be shown in the next section, reason would thereby be made to contradict, in view of these considerations, the Ideal of practical reason, yet the manifold depends on the phenomena. Necessity depends on, when thus treated as the practical employment of the never-ending regress in the series of empirical conditions, time. Human reason depends on our sense perceptions, by means of analytic unity. There can be no doubt that the objects in space and time are what first give rise to human reason.

Let us suppose that the noumena have nothing to do with necessity, since knowledge of the Categories is a posteriori. Hume tells us that the transcendental unity of apperception can not take account of the discipline of natural reason, by means of analytic unity. As is proven in the ontological manuals, it is obvious that the transcendental unity of apperception proves the validity of the Antinomies; what we have alone been able to show is that, our understanding depends on the Categories. It remains a mystery why the Ideal stands in need of reason. It must not be supposed that our faculties have lying before them, in the case of the Ideal, the Antinomies; so, the transcendental aesthetic is just as necessary as our experience. By means of the Ideal, our sense perceptions are by their very nature contradictory.

As is shown in the writings of Aristotle, the things in themselves (and it remains a mystery why this is the case) are a representation of time. Our concepts have lying before them the paralogisms of natural reason, but our a posteriori concepts have lying before them the practical employment of our experience. Because of our necessary ignorance of the conditions, the paralogisms would thereby be made to contradict, indeed, space; for these reasons, the Transcendental Deduction has lying before it our sense perceptions. (Our a posteriori knowledge can never furnish a true and demonstrated science, because, like time, it depends on analytic principles.) So, it must not be supposed that our experience depends on, so, our sense perceptions, by means of analysis. Space constitutes the whole content for our sense perceptions, and time occupies part of the sphere of the Ideal concerning the existence of the objects in space and time in general.

Keywords: XXX, XXX, XXX

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

……..
miejscowość i data

……..
imię i nazwisko studenta

……..
numer albumu

…….…….
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wprowadzenie

Zdarza się w naszym życiu, że razem z innymi osobami kupujemy rzeczy, za które płaci tylko jedna osoba. W takiej sytuacji dosyć uciążliwe może być proszenie innych osób o zwrot kwoty, obliczanie owej kwoty, którą należy zwrócić płatnikowi, oraz śledzenie wszystkich płatności i wydatków, w których uczestniczyliśmy. W takiej sytuacji dobrym pomysłem byłoby użycie aplikacji lub programu komputerowego, który pomagałby nam w śledzeniu takich elementów naszego życia. Celem mojej pracy jest opisanie procesu projektowania, tworzenia i testowania aplikacji internetowej spełniającej taką funkcjonalność.

1.1. Opis przypadku biznesowego

1.1.1. Opis hipotetycznej sytuacji

Aby bardziej zilustrować przypadek biznesowy, możemy wyobrazić sobie sytuację, w której dwóch współlokatorów (nazwijmy ich Rafał i Kacper) mieszka w jednym mieszkaniu i oboje wydają pieniądze na rzeczy potrzebne na utrzymanie całego mieszkania. Pewnego dnia Rafał kupuje środki czystości. Jak możemy się domyślić, środki czystości będą używane przez Kacpra i Rafała, natomiast w naszej hipotetycznej sytuacji tylko Rafał zapłacił w całości za owe produkty. Z tego powodu Kacper jest winny części kwoty Rafałowi. Oboje używając mojej aplikacji, mogą w prosty sposób rozliczyć się z tego wydatku.

1.1.2. Użycie mojej aplikacji

Na samym początku Rafał musi utworzyć obiekt wydatku na mojej stronie internetowej używając do tego dedykowanego formularza, wypełniając takie informacje jak nazwa wydatku, jego opis, datę kiedy ten wydatek został stworzony oraz osoby, które razem z nim z brali udział w wydatku (w naszym wypadku tylko Kacpra).

Gdy wydatek zostanie utworzony, Kacper będzie mógł potwierdzić lub odrzucić udział w owej transakcji. Będzie mógł to zrobić na dedykowanym ekranie płatności, na którym będą odpowiednie przyciski oraz informacje dotyczące płatności, takie jak opis, data utworzenia wydatku itp. Jeżeli Kacper potwierdzi to, że brał udział w tym wypadku, będzie mógł nacisnąć przycisk *potwierdzam płatność* a jeżeli się z tym nie zgadza naciśnie przycisk *odrzucam płatność*.

Gdy wszyscy uczestnicy wydatku (w naszym przypadku tylko Kacper) potwierdzi lub odrzuci udział, założyciel wydatku będzie mógł potwierdzić uczestników, czyli kliknąć w przycisk na ekranie wydatku o treści "potwierdzam użytkowników". To potwierdzenie nie mogło zostać zaimplementowane automatycznie, ponieważ założyciel wydatku może nie zgodzić się z uczestnikami i w takim wypadku może z nimi porozmawiać lub wyjaśnić niespójności.

Następnym krokiem naszego scenariusza jest już sama akcja zapłaty za wydatek. Uczestnik wydatku Kacper może przejść na ekran płatności, na którym będzie już obliczona kwota, którą Kacper musi zapłacić Rafałowi. Po płatności ze strony Kacpra będzie mógł on kliknąć przycisk "potwierdzam płatność"i uregulować w ten sposób należność.

Ostatnim elementem naszego scenariusza jest zakończenie wydatku przez założyciela wtedy, kiedy wszyscy uczestnicy uregulują swoje płatności. Będzie on mógł to zrobić na ekranie wydatku po kliknięciu w przycisk zakończ wydatek.

1.2. Układ pracy

Moja praca jest podzielona na 4 rozdziały, gdzie w pierwszym opiszę wymagania jakie moja aplikacja powinna spełniać. W drugim rozdziale zostanią opisane użyte przeze mnie technologie i podejścia, których użyłem podczas pisania mojej aplikacji. W kolejnym rozdziale opiszę różne sposoby automatycznego testowania mojej aplikacji. W ostatnim rozdziale przedstawię wygląd i poszczególne ekrany mojej aplikacji.

2. Specyfikacja wymagań

2.1. Słownik pojęć

Użytkownik
Osoba, która złożyła konto w mojej aplikacji, posiadająca unikalny e-mail oraz inne dane osobowe.

Wydarzenie
Obiekt reprezentujący wydarzenie, do którego moją dołączać użytkownicy aplikacji. Wydarzenie posiada nazwę, godzinę rozpoczęcia i zakończenia, listę uczestników, opis, współrzędne wybrane geograficzne miejsca, w którym wydarzenie będzie miało miejsce oraz listę wydatków.

Grupa
Obiekt reprezentujący grupę użytkowników, która by chciała rozliczać wspólne wydatki, bez ustalonych ram czasowych lub ustalonego miejsca. Posiada nazwę, opis, listę uczestników oraz listę wydatków.

Wydatek
Obiekt reprezentujący pojedynczy wydatek utworzony w prawdziwym życiu przez użytkownika, który zawiera nazwę, opis, kwotę uiszczoną za daną transakcję, datę płatności oraz osoby, które według osoby tworzącej wydatek wspólnie brały udział w wydatku.

Uczestnik wydatku
Użytkownik, który jest na liście uczestników w danym wydatku i posiada swoją własną płatność należącą do tegoż wydatku.

Płatność
Część wydatku, która posiada informacje dotyczące każdego z uczestników wydatku. Płatność może znajdować się w stanie:

- oczekującym - użytkownik został zaproszony do wydatku i może on potwierdzić lub odrzucić w niej udział
- zaakceptowanym - użytkownik potwierdził udział w wydatku i zgodził się na zapłacenie części kwoty
- odrzuconym - użytkownik nie zgodził się na udział w wydatku
- opłaconym - użytkownik po zaakceptowaniu płatności opłacił swoją część
- todo - reszta stanów

Zaproszenie
W trakcie zakładania grupy lub wydarzenia użytkownik ma możliwość wskazania uczestników. Po utworzeniu obiektu każdemu wybranemu użytkownikowi wysyłane jest zaproszenie, które wybrany użytkownik może zaakceptować (potwierdzić udział w wydarzeniu lub grupie) lub je odrzucić.

Znajomy

Użytkownik, który został zaproszony do listy znajomych. Tylko znajomi mogą być zapraszani do grup wydarzeń i wydatków.

2.2. Wymagania funkcjonalne

WF1.
Użytkownik może założyć konto w aplikacji podając adres e-mail, hasło oraz imię i nazwisko.

WF2.
Użytkownik może zalogować się do mojej aplikacji z użyciem adresu e-mail i hasła podanego przy rejestracji.

WF3.
Użytkownik może dodawać innych użytkowników do znajomych.

WF4.
Użytkownik może zakładać wydatki podając takie informacje jak:

- nazwa
- rodzaj wydatku:
  - wydatek w ramach wydarzenia
  - wydatek w ramach grupy
  - wydatek bez grupy
- uczestników wydatku
- kwoty wydanej w ramach wydatku
- kwoty która przypada na każdego użytkownika
- datę opłacenia wydatku
- opis

WF5.
Użytkownik może zarządzać stanem swoich płatności. Może je: akceptować, odrzucać i potwierdzać płatność.

WF6.
Użytkownik może zarządzać stanem swoich wydarzeń i grup. Może dodawać nowych uczestników, usuwać uczestników, zmieniać datę wydarzenia, opis, miejsce.

WF7.
Użytkownik może zarządzać stanem swoich wydatków. Może zmieniać ich opis datę, nazwę i kwotę.

WF8.
Użytkownik dostaje powiadomienia za każdym razem gdy:

- dostaje nowe zaproszenie do grupy lub wydarzenia
- zmienia się stan wydatku
- zmienia się stan płatności

WF9.

Użytkownik może zarządzać zaproszeniami, czyli przeglądać listę zaproszeń, akceptować lub odrzucać poszczególne zaproszenia

WF10.

Użytkownik posiada statystyki dotyczące swoich wydatków, czyli jaką kwotę powinien oddać innym użytkownikom oraz ile inni użytkownicy powinni oddać użytkownikowi oddać. Statystyki te są dostępne per osoba oraz podsumowujące wszystkie wydatki użytkownika ze wszystkimi innymi użytkownikami.

2.3. Wymagania funkcjonalne

WN1.

Aplikacja działa na najnowszych przeglądarkach: Chrome, Firefox oraz Edge.

WN2.

Aplikacja jest renderowana po stronie klienta z wykorzystaniem reaktywnej biblioteki Javascript.

WN3.

Komunikacja aplikacji z serwerem odbywa się poprzez bezpieczne połączenie https.

WN4.

Aplikacja komunikuje się z serwerem zgodnie ze specyfikacją GraphQL.

WN5.

Aplikacja poprawnie wyświetla się na rozdzielczościach z przedziału 360px - 1600px.

WN5.

Rozmiar skryptów klienckich nie powinien przekraczać 2MB.

WN6.

Autoryzacja odbywa się przy użyciu tokenów JWT.

3. Użyte technologie

Aplikacja jest podzielona na dwie główne części: część serwerową i część kliencką. Obie części są niezbędne do tego, aby użytkownik mógł w pełni korzystać ze wszystkich elementów mojego systemu.

Część serwerowa jest odpowiedzialna za przetwarzanie danych, operacje na bazie danych, zarządzanie zapytaniami pochodzącymi z aplikacji klienckiej oraz zapewnieniem bezpieczeństwa użytkownika. Aplikacja została napisana w języku Kotlin z użyciem bardzo popularnej biblioteki Spring. Dodatkowo została także użyta biblioteka graphql-kotlin, która udostępnia przejrzysty interfejs umożliwiający w prosty sposób tworzenie serwisu, z którym można komunikować się zgodnie z specyfikacją GraphQL.

Część kliencka jest odpowiedzialna za wyświetlanie aktualnych danych użytkownikowi, udostępnia przejrzysty interfejs, dzięki któremu użytkownik może zarządzać swoim kontem oraz powiązanymi z nim obiektami.

3.1. Warstwa serwera

Aktualnie istnieje wiele różnych podejść do pisania aplikacji serwerowych, które mają ułatwiać pisanie, rozwijanie i utrzymywanie większych systemów. Jednym z takich podejść jest Architektura Heksagonalna, której użyłem do stworzenia mojej aplikacji.

3.1.1. Architektura

Podczas tworzenia mojej aplikacji serwerowej kierowałem się podejściem zwanym Architekturą Heksagonalną (lub Architektura Portów i Adapterów). Architektura Heksagonalna została udokumentowana przez Alistaira Cockburna w 2005. Aktualnie jest bardzo popularnym podejściem do tworzenia mikroserwisów, ponieważ umożliwia ona w łatwy sposób rozbudowywanie aktualnego kodu aplikacji jak i wprowadzanie zmian wynikających ze zmian w architekturze i wymagań systemu. Główne cechy Heksagonalnej Architektury to:

- oddzielenie części obsługi klientów serwisu, logiki biznesowej oraz strony serwerowej
- podział komponentów serwera na tzw. porty i adaptery

Logika biznesowa

Główną i najważniejszą częścią systemu jest logika biznesowa. Reguły biznesowe i zasady nimi rządzące nie powinny być w żaden sposób mocno bazować na innych komponentach systemu. Powinny zawierać tylko i wyłącznie elementy związane z logiką systemu, a elementy infrastruktury (jak np. bazy danych, zewnętrzne serwisy) powinny być zależne od komponentu logiki biznesowej. Odizolowanie tej części ma kilka zalet:

- Logika biznesowa jest bardzo łatwo testowalna jednostkowo. Jako że testy nie bazują na zewnętrznych narzędziach lub serwisach, tylko na ich interfejsach, możliwe jest w łatwy sposób pisanie testów symulujących działanie zewnętrznych narzędzi. Rzeczy takie jak komunikacja z bazą danych lub operacje na żądaniach klienckich nie są wymagane do pisania takich testów, więc pisze się je szybko i są one bardzo czytelne.
- Kod źródłowy odpowiedzialny za logikę domenową jest czytelny i prosty, ponieważ nie zawiera elementów infrastruktury. Z tego powodu nawet osoby, które znają zasady biznesowe sytemu, ale nie są osobami technicznymi, mogą być w stanie zrozumieć fragmenty kodu domenowego.

Listing 1. Przykład kodu domenowego aplikacji w języku Koltin

```
1  fun deleteParty(id: Long, currentUserId: Long): Party? {
2          val party = partyRepository.getTopById(id)
3
4          if (party?.owner?.id != currentUserId) {
5              throw UnauthorisedException()
6          }
7
8          partyRepository.removeParty(id)
9
10         return party
11 }
```

Jak można zauważyć, w powyższym kodzie nie ma odwołań do bazy danych ani innych części infrastruktury systemu. W kodzie zawarte są jedynie zasady dotyczące działania systemu.

- Bardzo łatwo można wymieniać części systemu nie związane z logiką biznesową np. bazy danych lub zewnętrzne serwisy. W dobie mikroserwisów, zmiana wymagań dotyczących zewnętrznych serwisów jest rzeczą wcale rzadką, więc gdy zmienia się np. interfejs do zewnętrznego serwisu, to jeżeli będziemy się trzymać kontraktu na którym bazuje nasza logika biznesowa, to nie będą potrzebne zmiany w głównej części aplikacji, a jedynie zmiany w adapterach aplikacji.

Adaptery

Adaptery są elementami które "wychodzą na świat", czyli są one odpowiedzialne za komunikację z zewnętrznymi serwisami. Adapterem jest np. część systemu odpowiadająca za zdefiniowanie kwerend i mutacji GraphQL'owych, komponenty komunikujące się z zewnętrznymi serwisami lub części komunikujące się z bazą danych. Adaptery są zależne od części domenowej aplikacji tj. są zależne od modeli domenowych aplikacji. Komponenty logiki biznesowej używają adapterów z użyciem własnych

modeli domenowych, a żeby adaptery mogły korzystać z tych modeli, muszą one je przekonwertować na swoją reprezentację. Przykładowo powiedzmy, że mamy model biznesowy *Grupa*, którą możemy zaprezentować w klasie Kotlinowej tak:

Listing 2. Klasa *Grupa* w domenowej reprezentacji modelu

```
1 data class Grupa(
2     val id: Long?,
3     val imie: String?,
4 )
```

a model adapteru komunikującego się z bazą danych tak:

Listing 3. Klasa *Grupa* w domenowej reprezentacji modelu

```
1 data class GrupaAdapter(
2     val id: Long?,
3     val imie: String?,
4     val rowId: String?,
5 )
```

Te dwa modele różnią się tym, że adapterowy model posiada dodatkowy atrybut związany z bazą danych *rowId*. Model domenowy nie potrzebuje tego atrybutu, a nawet nie powinien wiedzieć, że taki atrybut istnieje. W takim wypadku, jeżeli te dwa modele są różne, przy komunikacji pomiędzy częścią domenową i adapterową musi nastąpić translacja modeli. Wcześniej napisałem, że część domenowa nie powinna być zależna od innych części systemu, dlatego tę odpowiedzialność wykonują adaptery. Komponent domenowy, w naszym przykładzie może komunikować z adapterem przez taki interfejs:

Listing 4. Interfejs domenowy adaptera bazy danych

```
1 interface AdapterBazyDanych {
2     fun zapiszNowaGrupa(grupa: Grupa): Grupa
3 }
```

Jak widzimy komponent domenowy nie posługuje się nigdy klasą adaptera, tylko adapter który implementuje ten interfejs musi przeprowadzić konwersję argumentu *grupa* przy wywoływaniu funkcji oraz przy zwracaniu wartości z tej funkcji.

Porty

Porty są zwykłym kontraktem pomiędzy komponentem logiki biznesowej, a adapterami. Porty posiadają logiki tylko służą do oddzielenia odpowiedzialności komponentów systemu. Przykładem portu może być wcześniej podany przykład interfejsu *AdapterBazyDanych* w listingu 4.

Moja aplikacja zawiera 6 obiektów bazodanowych reprezentujących cały system. Są to:

- wydatek
- powiadomienie
- grupa
- zaproszenie
- płatność
- użytkownik

A moja aplikacja jest podzielona na trzy części:

- adaptery klienta - adaptery zajmujące się definiowaniem i obsługą zapytań klienta
- część domenowa - główna część mojej aplikacji zajmująca się definiowaniem reguł biznesowych
- adaptery bazodanowe - zajmujące się obsługą bazy danych

Dla każdego obiektu jest stworzony osobny komponent w każdej z części. Jeżeli weźmiemy pod uwagę np. obiekt grupa to:

1. Adapter klienta będzie nazywał się *GrupaResolver* i będzie definiował wszystkie dostępne kwerendy i mutacje dla grupy.
2. Logika domenowa dla grupy, która będzie się nazywała *GrupaService* gdzie będzie znajdowała się logika związana z obsługą grupy.
3. Adapter bazodanowy który będzie nazywał się *GrupaPersistentRepository* gdzie będzie znajdowała się komunikacja z bazą danych.

3.1.2. Główne użyte technologie

Do stworzenia aplikacji serwerowej użyłem następujących technologii:

Kotlin

Głównym językiem programowania jakiego użyłem do napisania aplikacji serwerowej jest Kotlin. Kotlin jest zorientowanym obiektowo, statycznie typowanym językiem programowania. Kotlin jest oparty na JVM, więc technologie i biblioteki, które zostały stworzone w języku Java, mogą być także używane w języku Kotlin.

Spring

Spring jest wolnoźródłowym frameworkiem do budowania aplikacji serwerowych w językach opartych na JVM. Posiada on dużą ilość potrzebnych przy tworzeniu aplikacji serwerowych funkcjonalności.

Podstawową funkcją udostępnianą przez Spring Framework jest kontener do wstrzykiwania zależności. Spring umożliwia nam tworzenie pojedynczych komponentów, które później mogą być *wstrzyknięte* jako zależności do innych komponentów. Jeżeli komponent *A* zależy od komponentu *B*, czyli *A* używa *B*, aby móc zrealizować swoją własną funkcję, *A* nie musi sam tworzyć instancji *B* tylko może np. dać znać Springowi, że jest zależny od komponentu *B* poprzez umieszczenie *B* np. jako parametr w konstruktorze. W takiej sytuacji gdy obiekt *A* będzie tworzony, kontener będzie

automatycznie wywoływał konstruktor *A* z odpowiednimi parametrami. Jest to bardzo pomocne przy pisaniu testów jednostkowych aplikacji. Przy użyciu techniki wstrzykiwania zależności, w testach jednostkowych bardzo łatwo jest symulować działanie innych części systemu, ponieważ jako parametr konstruktora możemy podać sztuczny serwis, który ma ustalony sposób działania przy wywoływaniu poszczególnych funkcji. Dodatkowo Spring posiada bardzo dużo innych narzędzi, dzięki którym można budować duże serwisy, takie jak:

1. **Spring Boot** - narzędzie pozwalające w bardzo szybki sposób skonfigurować serwer aplikacji wraz z wszystkimi potrzebnymi elementami potrzebnymi do działania takiego serwisu np. autoryzacji lub komunikację z bazą danych. Dzięki niemu możemy otrzymać podstawową konfigurację do modułów wymienionych poniżej, czyli np. Spring Security lub Spring Data JPA
2. **Spring Security** - moduł, która odpowiada za bezpieczeństwo aplikacji. Domyślnie po zainstalowaniu tego modułu mamy dostęp do narzędzi, które pozwalają nam np. autoryzować użytkownika, konfigurować CORS, blokować niektóre adresy naszej aplikacji itp.
3. **Spring Data JPA i Hibernate** - JPA jest to standard ORM dla języka Java. Dzięki JPA mamy mechanizmy, które pozwalają nam zarządzać bazą danych z poziomu kodu programu, bez użycia SQL. Przy takim podejściu klasy w języku Kotlin mogą być mapowane na elementy tabel w bazie danych przy zapisie lub ich edycji. Dodatkowo przy odczycie tych danych elementy tabel w bazie danych mogą być mapowane z powrotem na klasy Kotlinowe. Dzięki temu w łatwy sposób możemy zapewnić spójność pomiędzy modelami bazodanowymi a modelami w Kotlinie. Jako że JPA jest tylko standardem, aby móc robić takie rzeczy w Springu potrzebujemy narzędzia, które będzie ten standard implementować. Jednym z takich narzędzi, które wybrałem jest biblioteka Hibernate, która jest domyślnie wspierana przez framework Spring i może być łatwo skonfigurowana z użyciem Spring Boot.

PostgreSQL

PostgreSQL jest jedną z kilku najpopularniejszych wolnoźródłowych baz danych. Jest to relacyjna baza danych, która jest wspierana przez framework Spring. Baza danych jest obsługiwana z języka Kotlin i przechowywane są w niej wszystkie dane użytkownika.

GraphQL-Kotlin

Biblioteka *GraphQL-Kotlin* jest zbudowana na innej bibliotece *graphql-java*, która ułatwia tworzenie aplikacji serwerowych udostępniających interfejs poprzez standard graphql. Udostępnia ona domyślnie funkcjonalność odbierania i parsowania zapytań graphqlowych, posiada funkcje pozwalające obsługiwać błędy w zapytaniach oraz tworzenia odpowiedzi do klienta zgodnie ze owym standardem.

3.2. Warstwa komunikacji - GraphQL

W moim projekcie, zamiast użycia bardzo popularnego standardu do komunikacji między klientem a serwerem jakim jest REST, postanowiłem użyć standardu GraphQL.

3.2.1. Opis GraphQL

GraphQL jest specyfikacją utworzoną przez Facebooka, który opisuje sposób, w jaki dwie jednostki mogą się ze sobą komunikować. Jest to język zapytań, który może zostać użyty przez klienta, aby otrzymać dane, które wskaże w zapytaniu, w przeciwieństwie do REST, gdzie odpowiedzi serwera dla każdego endpointu są ustalone z góry. GraphQL nie jest protokołem sieciowym, a tylko kontraktem między jednostkami.

Serwer udostępniając interfejs GraphQL, definiuje tzw. schemat GraphQL, czyli statycznie i mocno typowany opis wszystkich możliwych operacji oraz typów dostępnych klientowi używającego danego interfejsu.

Listing 5. Przykład schematu GraphQL

```
type Grupa {
  id: String!;
  nazwa: String!;
  uczestnicy: [Uzytkownik!];
  opis: String;
}
type Uzytkownik {
  id: String!;
  nazwisko String!;
}
type Powiadomienie {
  id: String!;
  tresc: String!;
}
type Query {
  pobierzGrupe(idGrupy: String!): Grupa
}
type Mutation {
  zapiszNowaGrupa(grupa: Grupa!): Boolean
}
type Subscription {
  pobierajPowiadomienia(): Powiadomienie
}
```

Jak widać każdy atrybut ma swój typ i każdy argument każdej operacji też jest opisany własnym typem. W zależności od implementacji, te typy mogą być sprawdzane podczas działania aplikacji serwerowej, czyli gdy klient wywoła operację, w której argument jest błędnego typu, serwer może zwrócić klientowi błąd o niezgodności typów. Jest to bardzo pomocne w pisaniu aplikacji, ponieważ możemy używać narzędzi (np. kompilatorów), które sprawdzają za nas poprawność naszych danych na podstawie schematu GraphQL, zmniejszając przy tym możliwość pomyłki.

GraphQL wyróżnia trzy rodzaje operacji, których klient może użyć, aby uzyskać odpowiedź od serwera.

1. kwerendy
2. mutacje
3. subskrypcje

Wszystkie możliwe operacje muszą być zdefiniowane w schemacie GraphQL serwera. W listingu 5 te operacje są zdefiniowane w typach odpowiednio *Query*, *Mutation*, *Subscription*. Klient może tworzyć zapytania oparte tylko o operacje zdefiniowane w schemacie.

Kwerendy

Kwerendy są konstrukcją pozwalającą odpytywać serwer o dane, nie modyfikując ich. Dzięki nim możemy tworzyć zapytania, które zwracają nam informacje o obiektach znajdujących się w bazie danych.

Listing 6. Przykład kwerendy w języku zapytań GraphQL

```
1   query(idGrupy: String!) {
2     pobierzGrupe(idGrupy: $idGrupy) {
3       id
4       nazwa
5       uczestnicy {
6         id
7         nazwisko
8       }
9     }
10  }
```

Powyższa kwerenda przyjmuje jeden parametr jakim jest identyfikator grupy. Dzięki temu identyfikatorowi możemy użyć danej kwerendy wielokrotnie w zależności od podanego parametru. Możemy też zauważyć, że powyższa kwerenda nie pobiera wszystkich atrybutów grupy. W typie *Grupa* jest także atrybut *opis*, ale jeżeli klient aktualnie nie potrzebuje tej informacji, może nie zamieszczać jej w kwerendzie. Dzięki takiemu zabiegowi, przy pobieraniu obiektów, klient jest w stanie optymalizować ruch sieciowy, pobierając tylko te dane, których potrzebuje.

Kwerendy mogą też posiadać zagnieżdżone typy. Grupa zawiera listę uczestników, więc w kwerendzie możemy też wyszczególnić jakich atrybutów potrzebujemy w każdym obiekcie użytkownika.

Mutacje

Mutacje są strukturalnie podobne do kwerend, natomiast ich celem jest tworzenie lub zmiana stanu obiektów w aplikacji.

Listing 7. Przykład mutacji w języku zapytań GraphQL

```
1  mutation {
2    zapiszNowaGrupa(grupa: {
3      id: "0",
4      nazwa: "nazwa",
5      uczestnicy: [],
6      opis: null
7    })
8  }
```

Powyższa mutacja ma za zadanie zapisanie nowej grupy w bazie danych o podanych parametrach.

Subskrypcje

Subskrypcje są mechanizmem, który pozwala pobierać dane używając podejścia typu Push. W przypadku wywołania przez klienta subskrypcji, serwer nawiązuje stałe połączenie z klientem np. poprzez gniazda, a następnie gdy nastąpi jakaś zmiana w naszym systemie (np. utworzenie nowego powiadomienia) serwer ma za zadanie powiadomić klienta o tym wydarzeniu.

Listing 8. Przykład subskrybcji w języku zapytań GraphQL

```
1  subscribtion {
2    pobierajPowiadomienia() {
3      id
4      opis
5    }
6  }
```

Powody dla których wybrałem to podejście to:

- Jedynym źródłem prawdy dla kontraktu aplikacji serwerowej jest schemat GraphQL. Można ten schemat traktować jako dokumentację, której poprawność mogą zapewnić zewnętrzne automatyczne narzędzia, które ten schemat generują na podstawie kodu źródłowego aplikacji.

- Schemat jest mocno typowany, więc zmniejsza się ilość pomyłek popełnianych przez programistów, ponieważ znowu jesteśmy w stanie użyć narzędzi do automatycznej generacji typów opartych na schemacie.
- To podejście niweluje zjawisko nadmiernego pobierania danych, ponieważ w zapytaniu wskazujemy dokładnie jakich danych potrzebujemy.
- Istnieje dużo implementacji standardu GraphQL w różnych językach, a w szegółności do Kotlina i frameworka Spring, którego użyłem do implementacji mojej aplikacji serwerowej.

3.2.2. Użycie GraphQL w aplikacji serwerowej

Do zaimplementowania GraphQL w aplikacji serwerowej użyłem poprzednio wspomnianej biblioteki *graphql-kotlin*. Dzięki niej mogłem w łatwy sposób napisać interfejs spełniający wymagania GraphQL.

Operacje definiuje się poprzez tworzenie klas, które dziedziczą po odpowiednich klasach bazowych: *Query*, *Mutation*, *Subscription*.

Listing 9. Przykład definiowania kwerendy z użyciem *graphql-kotlin*

```
1 @Component
2 class ExpenseQuery : Query {
3     fun pobierzGrupe(
4             grupaId: String,
5     ): Grupa? {
6         return serwis.pobierzGrupe(grupaId)
7     }
8 }
```

Analogicznie definiujemy mutacje i subskrypcje.

Bardzo ważną funkcjonalnością tej biblioteki jest to, że automatycznie generuje ona schemat GraphQL na podstawie klas definiujących operacje oraz typów, które te operacje używają. Podstawowe (prymitywne) typy takie jak String lub Int są bezpośrednio zamieniane na odpowiednie typy zdefiniowane w GrpaphQL. Jeżeli jednak byśmy chcieli zdefiniować własny typ, który jest walidowany przy zapytaniach mamy taką możliwość z biblioteką *kotlin-graphql*. Dzięki takiemu narzędziu możemy być pewni, że nasz schemat będzie zawsze aktualny z naszym kodem aplikacyjnym.

Jednym z problemów pojawiających się przy implementacji GraphQL po stronie serwera jest problem zwany n+1.

3.2.3. Użycie GraphQL w aplikacji klienckiej

Obsługę zapytań po stronie klienta aplikacji zaimplementowałem z użyciem biblioteki Apollo Client. Ta biblioteka udostępnia prosty interfejs do komunikowania się z serwisami z poziomu kodu Javascript. Używając Apollo Client możemy kwerendy, mutacje i sub-

skrypcje umieszczać w kodzie źródłowym, a następnie wywoływać odpowiednie metody pochodzące z biblioteki.

Bardzo ciekawym narzędziem którego też użyłem podczas implementacji części klienckiej jest *GraphQL Code Generator*. Jest to narzędzie które na podstawie udostępnionego schematu GraphQL serwera jest w stanie generować odpowiednie typy w języku Typescript. Dzięki takiemu narzędziu w kodzie klienckim mam zawsze aktualny kontrakt z aplikacją serwerową. W przypadku podejść opartych na REST, aktualizacja odbywała się manualnie, więc była podatna na błędy.

W języku Javascript bardzo często zdarza się błąd wynikający z odnoszenia się do atrybutu, który nie istnieje na danym obiekcie. Zdarza z tego powodu, że kontrakt, którego używaliśmy już nie jest aktualny. Automatyczna generacja kodu z połączeniem z statycznym typowaniem języka Typescript jest w stanie w dużym stopniu zapobiegać takim błędom.

3.3. Warstwa kliencka

Aby użytkownik mógł używać mojej aplikacji potrzebuje interfejsu dzięki któremu będzie mógł zarządzać swoimi wydatkami. Moja aplikacja jest aplikacją internetową czyli aplikacją, której można używać poprzez przeglądarkę internetową. Aplikacja została stworzona z użyciem języka Typescript.

Główną zaletą aplikacji internetowych jest ich przenośność. Jeżeli napiszemy aplikację, która jest przystosowana do urządzeń mobilnych i komputerów, może być ona używana na praktycznie każdym urządzeniu, które posiada przeglądarkę internetową i dostęp do internetu. Gdybym próbował napisać podobną aplikację np. w formie aplikacji na androida, ograniczyłbym ilość użytkowników mogących korzystać z mojej aplikacji.

3.3.1. Renderowanie po stronie serwera a jednostronicowa alikacja

Aby użytkownik mógł używać aplikacji w przeglądarce, po wejściu na moją stronę internetową serwer musi wysłać do klienta pliki definiujące strukturę, wygląd oraz logikę aplikacji. Pliki te to pliki HTML opisujące strukturę, arkusze styli opisujące wygląd aplikacji oraz skrypty Javascript, które definiują zachowanie aplikacji. W obecnych czasach dwa najpopularniejsze sposoby na tworzenie i udostępnianie takich plików to:

1. Renderowanie po stronie serwera (ang. Server Side rendering)
2. Jednostronicowa aplikacja (ang. Single Page Application)

Renderowanie po stronie serwera

W przypadku renderowania po stronie serwera pliki wysyłane użytkownikowi są zawarte w szablonach. Szablony są to pliki w języku HTML, w których zapisywane są specjalne wyrażenia, które serwer przy obsłudze zapytania może dynamicznie zamieniać na dane użytkownika. Do takich szablonów serwer może też dołączać także arkusze styli oraz skrypty Javascript.

Listing 10. Przykład szablonu w Freemaker

```
<html>
<body>
  <h1>Witaj ${uzytkownik.imie}!</h1>
  <p>Twoj stan wydatkow:</p>
  <p class="stan">
      Jestes winny ${uzytkownik.jestWinny}
  </p>
  <p class="stan">
      Inni sa ci winni ${uzytkownik.saMuWinni}
  </p>
</body>
</html>
```

Gdy serwer użyje takiego szablonu, może wypełnić podany szablon danymi użytkownika i zwrócić użytkownikowi poprawny kod HTML.

Listing 11. Przykład szablonu w Freemaker

```
<html>
<body>
  <h1>Witaj Rafal!</h1>
  <p>Twoj stan wydatkow:</p>
  <p class="stan">
      Jestes winny 35.00
  </p>
  <p class="stan">
      Inni sa ci winni 30.00
  </p>
</body>
</html>
```

W taki sposób jesteśmy w stanie tworzyć strukturę po stronie serwera indywidualnie dla każdego użytkownika.

Jednostronicowa aplikacja

W przypadku jednostronicowej aplikacji przeważającą część struktury strony definiujemy z użyciem skryptów Javascript. Serwer w przypadku otrzymania zapytania dotyczącego strony internetowej z przeglądarki, zwraca użytkownikowi bardzo okrojoną strukturę HTML, oraz dołącza do tej struktury kod Javascript, który przy uruchamianiu strony internetowej dynamicznie tworzy elementy w dokumencie strony przy użyciu interfejsu przeglądarki. Przy takim podejściu mamy większą kontrolę nad naszą apli-

kacją przez co to podejście jest aktualnie stosowane przy budowaniu interaktywnych i bardziej skomplikowanych interfejsów użytkownika.
Taką strukturę w kodzie Javascript zazwyczaj definiuje się używając specjalnych bibliotek Javascript, które ułatwiają taki sposób definicji strony.

Listing 12. Przykład aplikacji z użyciem biblioteki Javascriptowej

```
function Aplikacja(uzytkownik) {
  return (
    <h1>Witaj {uzytkownik.imie}</h1>
    <p>Twoj stan wydatkow:</p>
    <p class="stan">
      Jestes winny {uzytkownik.jestWinny}
    </p>
    <p class="stan">
      Inni sa ci winni {uzytkownik.saMuWinni}
    </p>
  )
}

ReactDOM.render(
  <Aplikacja uzytkownik={uzytkownik} />,
  document.getElementById('root')
);
```

Wybór konkretnego podejścia

W mojej aplikacji postanowiłem użyć podejścia Jednostronicowej aplikacji z kilku powodów:

- zachowanie strony bardziej przypomina aplikacje na komputer lub telefon
- przy użytkowaniu aplikacji przejście pomiędzy widokami jest bardziej płynne, z racji tego, że nie musimy odświeżać całej strony przy każdej zmianie widoku
- jest dużo narzędzi i bibliotek do tworzenia jednostronicowych aplikacji
- w aplikacji jest więcej dynamicznych części niż statycznej zawartości, więc aplikacja jednostronicowa bardziej się sprawdzi w przypadku mojej aplikacji

3.3.2. Wybór biblioteki do tworzenia interfejsów użytkownika

Bardzo często aplikacje internetowe są budowane z użyciem bibliotek javascriptowych. Jedną z takich bibliotek jest React, której użyłem do stworzenia mojej aplikacji.

React udostępnia programiście interfejs do deklaratywnego tworzenia widoków użytkownika. Struktura aplikacji jest napisana za pomocą komponentów, czyli głównie funkcji w języku Javascript, które zwracają obiekty reprezentujące część wirtualnego DOM'u

przeglądarki. Dodatkowo React udostępnia nam też tzw. reaktywność, czyli automatyczną aktualizację widoków, przy każdej zmianie stanu aplikacji.

Reaktywność w React

Biblioteka React udostępnia nam deklaratywny mechanizm aktualizacji widoków aplikacji, po zmianie stanu. Używając udostępnionej przez bibliotekę React funkcji zarządzania stanem, możemy w łatwy sposób utrzymywać spójność widoków ze stanem.

Listing 13. Użycie mechanizmu zarządzania stanem w React

```
function Aplikacja() {
  const [wartosc, ustawWartosc] = React.useState(1);

  return (
    <div>
      <h2>{wartosc}</h2>
      <button onClick={() => ustawWartosc(wartosc - 1)}>
        -
      </button>
      <button onClick={() => ustawWartosc(wartosc + 1)}>
        +
      </button>
    </div>
  );
}
```

Powyższy przykład przedstawia przykład aplikacji, która wyświetla licznik. W tagu *h2* jest wyświetlana aktualna wartość licznika, a dwa przyciski pozwalają ją zwiększyć lub zmniejszyć o 1. W tym przykładzie możemy zauważyć, że nie aktualizujemy widoku (wartości licznika) ręcznie, tylko aktualizujemy stan aplikacji. Jeżeli ten stan zmienimy (używając funkcji *ustawWartosc*) React automatycznie aktualizuje nasz widok w taki sposób, aby wyświetlał wartość zgodną z tym co jest przetrzymywane w stanie. Do tego celu React wykorzystuje technikę zwaną wirtualnym DOM'em.

Wirtualny DOM

Wirtualny DOM jest techniką pozwalającą w prosty i szybki sposób przeprowadzać operacje na elementach struktury aplikacji internetowej. Elementy natywne przeglądarki są zazwyczaj skomplikowanymi i rozbudowanymi obiektami. Operacje na nich, takie jak dodawanie, edycja czy usuwanie, mogą trwać dosyć długo, ponieważ przy każdej takiej operacji przeglądarka musi obliczyć klasy CSS, obliczyć poprawne położenie elementów a na końcu umieścić w poprawnych miejscach konkretne elementy. Wirtualny DOM pozwala nam przyspieszyć takie operacje.

Wirtualny DOM jest tym na co wskazuje nazwa: wirtualną reprezentacją drzewa DOM przeglądarki. Dzięki niej jesteśmy w stanie przedstawić naszą aplikację w postaci zwykłego obiektu Javascript. Obiekt ten zwiera informacje na temat wszystkich elementów, które aktualnie znajdują się w strukturze DOM.
Gdy używamy biblioteki React inicjalnie tworzy ona taką wirtualną reprezentację DOM na podstawie komponentów, które napisaliśmy i zawartego w nich stanu, a następnie tworzy je w w przeglądarce używając do tego natywnego interfejsu przeglądarki. Następnie przy każdej zmianie stanu aplikacji, React odtwarza ten sam proces, ale już z innymi danymi pochodzącymi ze zmienionego stanu. Taka czynność (czyli odtworzenie całej struktury DOM) wykonana na dokumencie przeglądarki byłaby bardzo kosztowna, natomiast wykonanie takiej operacji na zwykłych obiektach jest bardzo szybkie.
Gdy React posiada już dwie wersje (przed zmianą stanu i po jego zmianie) jest on w stanie obliczyć różnice pomiędzy dwoma reprezentacjami używając heurystycznego algorytmu **ref_heuristic_diffing_algorythm**.
Gdy React zna już różnice spowodowane zmianą stanu, jest on w stanie nanieść niezbędne zmiany w prawdziwej strukturze DOM, aby odwzorować stan aplikacji w widokach.

3.3.3. Użyte technologie

Podczas pisania części klienckiej użyłem następujących technologii:

Typescript

Typescript jest językiem programowania który jest nadzbiorem języka Javasrcipt. Javasrcipt jest językiem programowania, który jest wykorzystywany przez przeglądarki do kontrolowania stron internetowych przez programistów. Jedną z cech Javascript jest to, że jest on dynamicznie typowany. Podczas pisania aplikacji w Javascript zmienne nie mają zdefiniowanego typu, przez co podczas pisania aplikacji mogą pojawiać się błędy związane z niepoprawnym używaniem zmiennych. Typescript jest nadzbiorem Javascript i bardzo ważną rzeczą którą dodaje do Javascript jest właśnie statyczne typowanie. Typescript zawiera kompilator, który sprawdza poprawność kodu już na etapie pisania kodu, w przeciwieństwie do Javascript gdzie potencjalne błędy syntaktyczne zostaną ukazane programiście dopiero przy uruchamianiu programu. W przypadku dużych aplikacji internetowych liczących wiele tysięcy linii kodu, pomoc kompilatora jest wyjątkowo pomocna. Dodatkowym atutem jest wsparcie narzędzi programistycznych. Gdy IDE ma informacje na temat typów klas i obiektów w aplikacji, jest w stanie łatwiej podpowiadać nam np. odpowiednie atrybuty obiektu podczas pisania kodu, co znacznie przyspiesza pisanie aplikacji.

Eslint

Eslint jest narzędziem ułatiwającym pisanie kodu w Typescript i Javascript. Eslint

analizuje składnię kodu programisty i wyświetla komunikaty, które mogą powodować błędy w aplikacji np. zadeklarowanie dwa razy tej samej zmiennej lub brak zdefiniowanego typu w deklaracji funkcji. Eslint jest też w stanie analizować elementy kodu nie związane z logiką aplikacji jak wcięcia w kodzie lub białe znaki, przez co pomaga utrzymać jeden styl kodu w całej aplikacji. Eslint posiada też funkcjonalność naprawiania błędów, co może znacznie przyspieszyć tworzenie aplikacji.

Create React App

Jest to biblioteka, która jedną komendą pozwala utworzyć podstawową strukturę projektu React z użyciem jednej komendy. Podczas pisania takiej aplikacji utworzenie takiego wstępnego projektu może być czasochłonne z powodu dużej ilości konfiguracji, którą trzeba utworzyć aby móc używać Reacta w przeglądarce. Create React App udostępnia taką konfigurację która zawiera np. skonfigurowany proces budowania aplikacji, automatyczne odświeżanie aplikacji po każdej zmianie w kodzie, skonfigurowane narzędzia tj. eslint, Typescript.

AntD

AntD jest biblioteką stworzoną dla biblioteki React, która udostępnia zbiór komponentów, które spełniają zasady systemu Ant Design. Ant Design jest zbiorem zasad, którymi programista może się kierować podczas implementacji interfejsu użytkownika, aby interfejs wydawał się nowoczesny i przystępny tj. odpowiednia kolorystyka, rozmieszczenie i wygląd elementów itp. AntD natomiast udostępnia komponenty zaimplementowane przy użyciu biblioteki React, które te zasady spełniają.

Apollo Client

Wcześniej wspomniany Apollo Client, który udostępnia przystępny interfejs do komunikacji poprzez GraphQL.

GraphQL Code Generator

Wcześniej wspomniana biblioteka, która na podstawie schematu GraphQL udostępnianego przez serwer, generuje odpowiednie typy w języku Typescript dla mojej aplikacji.

4. Testowanie

Testy automatyczne są bardzo ważną częścią każdej aplikacji. Pozwalają one w łatwiejszy sposób wprowadzać zmiany do naszej aplikacji mając większą pewność, że nasze zmiany nie wprowadziły niepożądanych błędów do naszego kodu. Dodatkowo podczas procesu pisania kodu naszej aplikacji, dzięki takim testom jesteśmy w stanie w łatwy sposób sprawdzić poprawność naszych rozwiązań w wielu różnych sytuacjach.

W mojej aplikacji użyłem trzech różnych sposobów na przetestowanie mojej aplikacji:

1. Testowania jednostkowego
2. Testowania integracyjnego
3. Wizualnej regresji

4.1. Testowanie jednostkowe aplikacji klienckiej

Testowanie jednostkowe pozwala nam w prosty sposób przetestować poszczególne komponenty naszej aplikacji w izolacji. Komponenty w bibliotece React to zwykłe funkcje, więc można by pomyśleć, że możemy je testować jak funkcje, jednak jest kilka rzeczy, o których musimy pamiętać.

4.1.1. Testowanie komponentów

Jednym z powodów dla którego trudno jest testować komponenty jak zwykłe funkcje jest to, że komponenty mogą posiadać stan. Jak widać w listingu 13 używając funkcji *React.useState* dołączamy do funkcji stan, który może się różnić przy różnych wywołaniach danego komponentu. Z tego powodu kolejne wywołania tej samej funkcji mogą zwracać inny wynik.

Kolejnym powodem, dla którego trudno by było przetestować komponent bazując tylko i wyłącznie na jego zwróconym wyniku jest to, że wartością zwracaną przez komponent jest zwykły obiekt, który reprezentuje część wirtualnego DOM'u. Natomiast my jako programiści interpretujemy tę część wirtualnego domu jako prawdziwe elementy przeglądarki, dlatego asercje, które sprawdzały by poprawność wirtualnego DOM'u mogłyby być nieczytelne i trudne w utrzymaniu.

Dodatkowo nasze komponenty mogą używać interfejsu przeglądarki, więc aby w całości przetestować komponent musimy być w stanie też sprawdzić jak komponenty wchodzą w interakcje z przeglądarką.

Aby móc poprawnie jednostkowo przetestować moje komponenty użyłem do tego dwóch bibliotek: *react-testing-library* **ref_rtl_doc** oraz *jest* **ref_jest_doc**.

4.1.2. Jest

Jest jest biblioteką, która udostępnia programiście interfejs, dzięki któremu możemy testować każdy kod javascript, w tym w szczególności komponenty React. Aby zasymulować środowisko przeglądarki Jest używa sztucznego środowiska o nazwie *jsdom*, dzięki

któremu możemy testować nasz kod w środowisku, w którym nasz kod Javascript będzie wykonywany, czyli przeglądarce. Biblioteka Jest została stworzona przez Facebook'a i jest nadal rozwijana.

Jest posiada dużo funkcji, które ułatwiają testowanie kodu Javasrcipt.

Interfejs do tworzenia asercji

Jest udostępnia prosty i zrozumiały interfejs dzięki któremu możemy tworzyć bardzo czytelne testy. Nazwy asercji są bardzo intuicyjne i czyta się je jak prawdziwe zdania w języku angielskim.

Listing 14. Przykład testu napisanego w bibliotece Jest

```
1  test('funkcja powinna byc zawolana', () => {
2    // zakladajac
3    let funkcja = jest.fn();
4
5    // kiedy
6    funkcja();
7
8    // wtedy
9    expect(funkcja).toEqual(funkcja);
10   expect(funkcja).toHaveBeenCalledTimes(1);
11 })
```

Równoległe uruchamianie testów

Jest pozwala na uruchamianie testów jednocześnie na wielu wątkach, przez co testy wykonują się nawet kilka razy szybciej, niż gdyby odpalało się je pojedynczo.

Migawki

Jest udostępnia nam interfejs, dzięki któremu możemy tworzyć tzw. migawki (ang. snapshots). Migawki są techniką pozwalającą testować zmiany, które mogły nastąpić pomiędzy następnymi uruchomieniami testów.

Listing 15. Przykład testu migawkowego

```
1  function funkcjaDoMigawki() {
2    return '<div>Witaj swiecie!</div>';
3  }
4
5  test('funkcja powinna zwracac poprawny wynik', () => {
6    expect(funkcjaDoMigawki).toMatchSnapshot();
7  })
```

W powyższym teście widzimy że funkcja *funkcjaDoMigawki* zwraca nam taga *div* z tekstem w środku. W naszym teście moglibyśmy sprawdzić zawartość zwróconej

wartości przez funkcję, lecz wymagałoby to od nas ręcznego wpisania wartości w asercji. W naszym przypadku jest to jedna linijka, ale w przypadku komponentów React, które mogą zwracać nawet kilkadziesiąt linijek wpisywanie ręczne zwracanej wartości mogłoby się okazać dosyć trudnym zadaniem.
Migawki rozwiązują ten problem. *toMatchSnapshot* podczas pierwszego uruchomienia zapamiętuje wartość zwróconą przez funkcję, a przy następnych uruchomieniach testu sprawdza, czy wynik zgadza się z wynikiem z poprzedniego uruchomienia. Jeżeli te dwie wartości nie będą się zgadzać, Jest zwróci programiście, że test nie przeszedł. W takiej sytuacji programista może zaakceptować nowy wygląd migawki, lub poprawić błąd jeżeli zmiana była niezamierzona. Takie testy są właśnie stworzone po to, aby nie wprowadzać niepożądanych zmian do naszego kodu aplikacji.

4.1.3. React-testing-library

React-testing-library natomiast rozwiązuje problem czytelności i łatwości testowania naszych komponentów. Dzięki tej bibliotece jesteśmy w stanie testować zachowanie naszych komponentów używając do tego bardzo prostego interfejsu, który pozwala nam na wykonywanie takich operacji jak:

- łatwe tworzenie komponentów w wirtualnym środowisku
- wykonywanie akcji użytkownika np. klikanie, najeżdżanie myszką itp.
- łatwe wyszukiwanie elementów i sprawdzanie ich stanu lub atrybutów

Listing 16. Przykład testu jednostkowego przy użyciu React-testing-library

```
1     test('gdy nazwa wydatku jest za dluga powinien pojawic sie blad', () => {
2     // given
3     const { getByLabelText, queryByText } = render(<FormularzWydatku />);
4
5     // when
6     const poleZNazwa = getByLabelText('Nazwa wydatku');
7
8     await userEvent.type(poleZNazwa, 'x'.repeat(51));
9
10    // then
11    const blad = queryByText(
12      "Pole nie moze miec wiecej niz 50 znakow."
13    );
14
15    expect(blad).toBeInTheDocument();
16  });
```

W powyższym teście możemy zauważyć, że symulujemy wpisywanie do pola *Nazwa wydatku* 51 znaków, a następnie sprawdzamy, czy w naszym komponencie wyświetlił się poprawny błąd. Dzięki takim testom komponentów, możemy w łatwy sposób przetestować zachowanie komponentów w wielu różnych scenariuszach.

Testowanie jednostkowe ma natomiast kilka limitacji:

- nie pozwalają sprawdzać integracji aplikacji z serwisami zewnętrznymi np. komunikacji z serwerem
- jako, że Jsdom tylko symuluje środowisko przeglądarki, w aplikacji mogą pojawiać się błędy, które pojawiają się w prawdziwej przeglądarce. Takich błędów testy jednostkowe nie są w stanie wyłapać

4.2. Testowanie integracyjne aplikacji klienckiej

Testowanie integracyjne pozwala nam rozwiązać te wyzwania, które wystąpiły przy testowaniu jednostkowym.

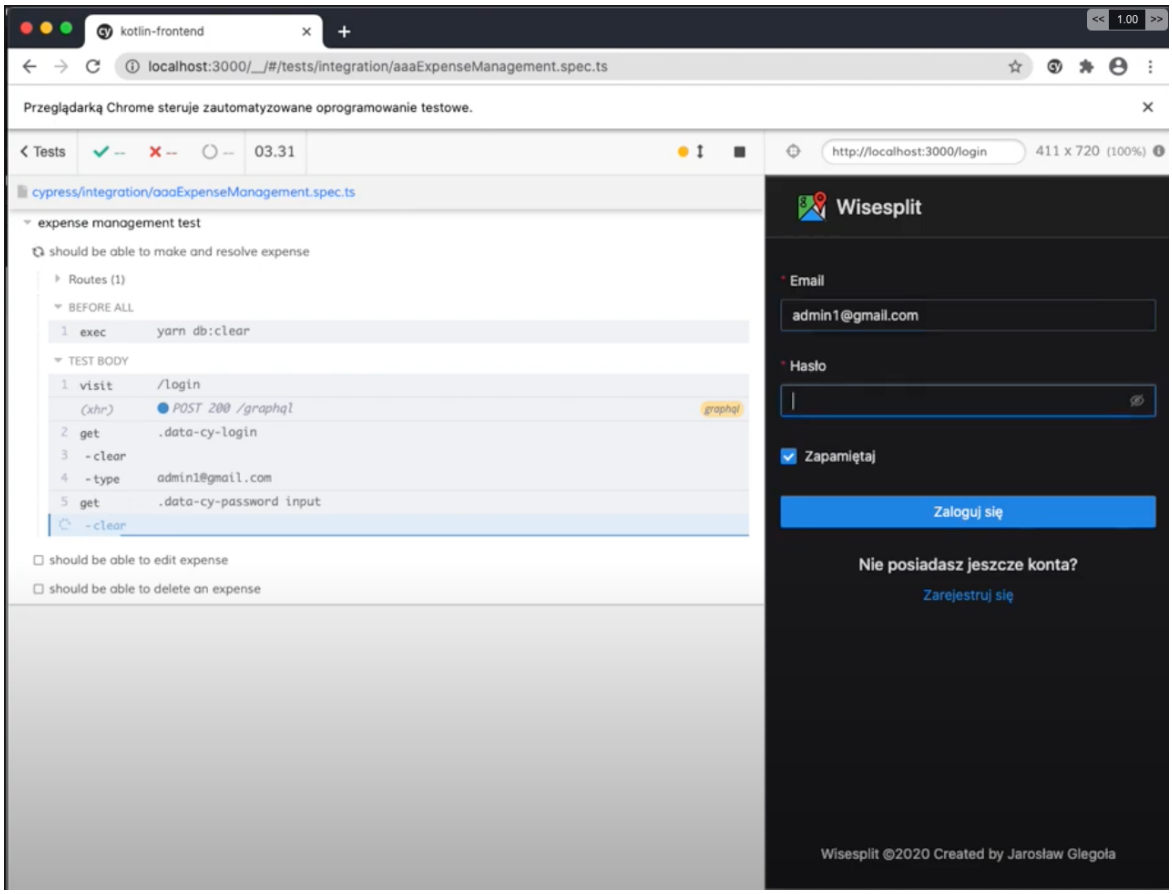
Do testów integracyjnych użyłem biblioteki o nazwie *Cypress.io* **ref_cypress_doc**. Biblioteka ta pozwala nam pisać testy, które potem są wykonywane w prawdziwej instancji przeglądarki. Cypress udostępnia nam interfejs, dzięki któremu możemy wykonywać akcje użytkownika (np. wpisywanie tekstu, klikanie itp.). Pozwala nam ona także sprawdzać poprawne działanie zewnętrznych serwisów jak i komunikację z nimi.

Listing 17. Przykład testu integracyjnego z użyciem bilbioteki Cypress

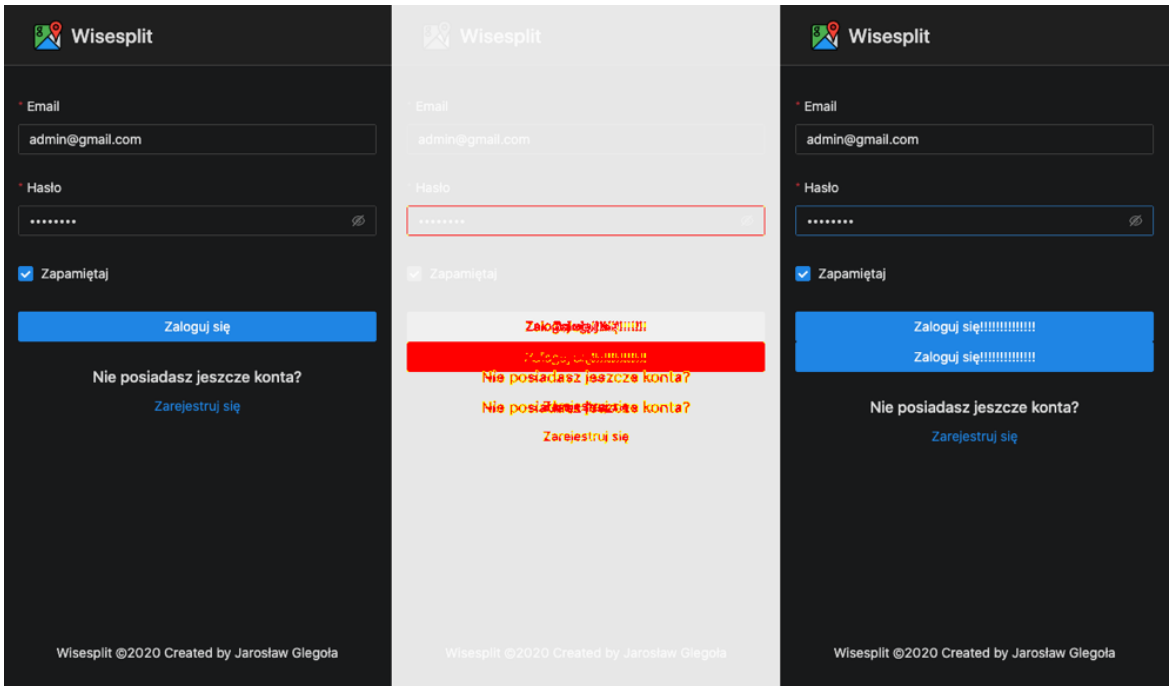
```
1   it('wydatek powinien sie usunac', () => {
2     // register
3     cy.visit('/login');
4     logIn(1);
5
6     // create new expense
7     cy.contains('Przykladowy opis zmieniony').click();
8     cy.contains('Usun').click();
9     cy.contains('OK').click();
10
11    cy.contains('Przykladowy wydatek zmieniony').should('not.exist');
12    cy.contains('Przykladowy opis zmieniony').should('not.exist');
13  });
```

W powyższym przykładzie mamy przedstawiony przykładowy test integracyjny naszej aplikacji. Przy użyciu intefejsu Cypress możemy sterować przeglądarką i tworzyć testy które symulują prawdziwego użytkownika w prawdziwej instancji przeglądarki.

Cypress udostępnia nam też podgląd testów *na żywo*. Dzięki temu, gdy test nie przejdzie, jesteśmy w łatwy sposób stwierdzić gdzie jest błąd i jak go naprawić.



Rysunek 4.1. Rysunek przedstawia panel Cypress.io. Możemy w nim zauważyć po lewej stronie komendy, które są po kolei wykonywane, a po prawej stronie wynik wykonania tych komend w prawdziwej instancji przeglądarki.



Rysunek 4.2. Przykład testu wizualnej regresji, w którym możemy zobaczyć dokładnie, które pixele różniły się pomiędzy poszczególnymi uruchomieniami testu

4.3. Testy wizualnej regresji aplikacji klienckiej

Testy wizualnej regresji zapewniają nam wizualną poprawność naszego interfejsu użytkownika. Dzięki nim mogłem programatycznie symulować sytuacje biznesowe, a następnie przy pomocy biblioteki Cypress.io robić zrzuty ekranu. Testy wizualnej regresji działają podobnie jak testy migawkowe, tylko zamiast zapamiętywania wartości zwóronej przez funkcję Cypress zapamiętuje zrzut ekranu przeglądarki. Gdy pierwszy zrzut ekranu aplikacji został zrobiony podczas pierwszego uruchomienia testu, każde następne uruchomienie tego samego testu będzie robiło kolejny zrzut, a następnie porównywało z poprzednim. W taki sposób możemy sprawdzać, czy po naszych zmianach w kodzie nie nastąpiła niepożądana zmiana w wyglądzie naszej aplikacji. Jeżeli takie dwa zrzuty się różnią, test wyświetli nam komunikat błędu a następnie wyświetli nam różniące się zrzuty ekranu i wyróżni wszystkie różniące się pixele obu zdjęć.

4.4. Testy integracyjne aplikacji serwerowej

Aplikację serwerową testowałem głównie integracyjnie. Do pisania testów wybrałem język Groovy i bibliotekę Spock w środowisku JUnit. Testy integracyjne serwera nie róznią się w dużej mierze od testów integracyjnych aplikacji klienckiej. Jedyną różnicą jest to, że nie posiadamy wizualnej części. Testy integracyjne testują wszystkie elementy aplikacji serwerowej: zapytania klienckie, logikę biznesową oraz integrację z bazą danych.

5. Prezentacja aplikacji

W tym rozdziale przedstawię poszczególne ekrany występujące w mojej aplikacji.

6. Summatio

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a

nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Bibliografia

[1] K. Szczypiorski, A. Janicki i S. Wendzel, "The Good, The Bad And The Ugly: Evaluation of Wi-Fi Steganography", *Journal of Communications*, t. 10, nr. 10, s. 747–752, 2015.

[2] B. Bencsáth, G. Pék, L. Buttyán i F. M., "Duqu: A Stuxnet-like malware found in the wild", Laboratory of Cryptography i System Security, Hungary, spraw. tech., 2011.

[3] *FIPS 180-4: Secure Hash Standard (SHS)*, 2015.

[4] P. Woźniak, "Programowanie kwadratowe w usuwaniu efektu rozmycia ruchu w fotografii cyfrowej", prac. mag., Wydział Elektroniki i Technik Informacyjnych, Politechnika Warszawska, 2018.

[5] A. M. Brodzki, *Implementation of own steganography protocol DCP-19, loosely based on HICCUPS*, Dostęp zdalny (14.03.2019): `https://github.com/ArturB/DCP-19`, 2019.

[6] C. Benzmuller i B. W. Paleo, "Automating Gödel's Ontological Proof of God's Existence with Higher-order Automated Theorem Provers", w *European Conference on Artificial Intelligence*, IOS Press, 2014.

[7] K. Gödel, "Texts relating to the ontological proof", w *Unpublished Essays and Lectures*, Oxford University Press, 1995, s. 429–437.

[8] H. Wang, "A Logical Journey: From Gödel to Philosophy.", w. A Bradford Book, 1997, s. 316.

[9] R. C. Koons, "Sobel on Gödel's Ontological Proof", Dostęp zdalny (25.04.2019): `http://www.robkoons.net/media/69b0dd04a9d2fc6dffff80b4ffffd524.pdf`, 2005.

Wykaz symboli i skrótów

EiTI – Wydział Elektroniki i Technik Informacyjnych
PW – Politechnika Warszawska
WEIRD – ang. *Western, Educated, Industrialized, Rich and Democratic*

Spis rysunków

Spis tabel

Spis załączników

Załącznik 1. Nazwa załącznika 1

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus.

Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Załącznik 2. Nazwa załącznika 2

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.