Obraz zawierający tekst

Opis wygenerowany automatycznie

**Wydział Informatyki**

**Katedra Metod Programowania**

Programowanie aplikacji biznesowych

**Arkadiusz Dariusz Pańczyk**

Nr albumu s18706

**Aplikacja internetowa wspomagająca monitorowanie i planowanie budżetu domowego**

Praca inżynierska

pod kierunkiem:

dr Krzysztof Barteczko

Warszawa, styczeń 2022

**Streszczenie**

Jako podmiot niniejszej pracy wybrano problem codziennego monitorowania oraz planowania budżetu domowego. Celem jest utworzenie rozwiązania przedstawionej trudności w oparciu o nowoczesną aplikację internetową. Zapoczątkowana analiza produktów dostępnych na rynku od pewnego czasu pozwoliła na przedstawienie architektury i wymagań systemowych omawianego oprogramowania. W prowadzonym projekcie zadbano o rozeznanie w sprawdzonych i popularnych technologiach programistycznych. Elementy zapisane przy użyciu stosunkowo nowych języków programowania Kotlin lub TypeScript zostały odpowiednio opisane oraz przedstawione poprzez proste fragmenty implementacyjne na przykładach bibliotek szkieletu Spring oraz React. Produktem przedstawianym dla czytelnika jest gotowa aplikacja internetowa pozwalająca na wprowadzanie wybranych operacji finansowych wprost do systemu informatycznego opartego na relacyjnej bazie danych.

**Słowa kluczowe:** Aplikacja internetowa, zarządzanie budżetem domowym, Kotlin, Spring Framework, TypeScript, React.

**Spis treści**

[1. Zarządzanie budżetem domowym oraz cele projektu 4](#_Toc91780487)

[2. Koncepcje, funkcjonalności i architektura aplikacji 7](#_Toc91780488)

[2.1. Analiza podobnych rozwiązań dostępnych na rynku 7](#_Toc91780489)

[2.2. Wymagania systemowe i koncepcje 10](#_Toc91780490)

[2.3. Szablony diagramów modelujących oraz architektura aplikacji 12](#_Toc91780491)

[2.4. Architektura aplikacji 16](#_Toc91780492)

[3. Rozwiązania technologiczne 19](#_Toc91780493)

[3.1. Backend 19](#_Toc91780494)

[3.1.1. Kotlin 19](#_Toc91780495)

[3.1.2. Spring Boot 21](#_Toc91780496)

[3.1.3. Hibernate / Spring Data JPA 23](#_Toc91780497)

[3.1.5. JUnit / Spring Test 25](#_Toc91780498)

[3.2. Frontend 26](#_Toc91780499)

[3.2.1. TypeScript 27](#_Toc91780500)

[3.2.2. React 28](#_Toc91780501)

[3.2.3. Material-UI 30](#_Toc91780502)

[3.3. Baza danych 30](#_Toc91780503)

[3.3.1. PostgreSQL 32](#_Toc91780504)

[3.3.2. H2 33](#_Toc91780505)

[3.4. Narzędzia programistyczne 34](#_Toc91780506)

[3.4.1. IntelliJ IDEA 34](#_Toc91780507)

[Pozostałe rozwiązania technologiczne 35](#_Toc91780508)

[3.4.4. Git 36](#_Toc91780509)

[4. Implementacja 39](#_Toc91780510)

[4.1. Moduł rejestracji i logowania 39](#_Toc91780511)

[4.2. Tabela operacji 42](#_Toc91780512)

[4.3. Dodawanie oraz modyfikacja operacji 44](#_Toc91780513)

[4.4. Dostępne wykresy 46](#_Toc91780514)

[4.5. Testy jednostkowe dostępne w aplikacji 47](#_Toc91780515)

[4.6. Ciągła integracja w Github Actions 49](#_Toc91780516)

[Podsumowanie i wnioski 50](#_Toc91780517)

[Bibliografia 52](#_Toc91780518)

1. Zarządzanie budżetem domowym oraz cele projektu

Zarządzanie pieniędzmi w zakresie budżetu domowego rządzi się wieloma zasadami, a najbardziej podstawową z nich jest praktyka wydawania mniejszej ilości pieniędzy niż się zarabia. Najprostszą drogą do trzymania się planu jest rozpisanie swoich podstawowych wydatków i następna analiza czy aby na pewno wydatki, które ponosi się przez pewien okres są kosztem bezwzględnie potrzebnym czy może niezrozumiałym.

W przypadku wydajnego oszczędzania pieniędzy przydatna może okazać się zasada 50/30/20[[1]](#footnote-1), opracowana przez profesor Elizabeth Ann Warren. Praktyka okazuję się pomocna w przypadku osób mających problemy z oszczędzaniem pieniędzy. Takie osoby często wydają całe swoje miesięczne wydatki anulując tym samym możliwość zakupu droższych przedmiotów, które w praktyce mogą mieć większą użyteczność niż inne wydatki poniesione w danym miesiącu. Według zasady 50/30/20, cały miesięczny przychód należy odpowiednio podzielić na trzy części. Pięćdziesiąt procent przychodów powinno zostać w całości użyte na niezbędne do życia potrzeby, takie jak jedzenie, rachunki czy transport. Pozostałe dwie części należy podzielić odpowiednio na trzydzieści procent na przyjemności a pozostałe dwadzieścia procent ma służyć jako oszczędności w celu zabezpieczenia przyszłości.

Jest to jedynie jeden z wielu sposobów na jaki można podzielić budżet, w celu oszczędzania pieniędzy. Z pewnością sprawdza się on dobrze w przypadku przeciętnych obywateli jednak pomimo swoich wielu zalet nie jest on idealnym rozwiązaniem dla każdego[[2]](#footnote-2). Pomimo tego podstawowa zasada monitorowania swoich wydatków zawiera się w zasadzie 50/30/20 i z pewnością jest wydajnym sposobem na podstawową analizę przychodów i rozchodów. Pozwala ona w wyjątkowej większości przypadków na miarodajną ocenę sytuacji budżetu domowego.

W takich przypadkach wystawione zostają zalety najprostszych aplikacji pozwalających na proste monitorowanie wydatków. Bez wykresów i wydajnego wyselekcjonowania na kategorie wydatkowe, z pewnością cięższa jest ocena podziału na płatności bardziej potrzebne i na takie, na których można oszczędzić duże ilości pieniędzy a także w dłuższej perspektywie czasu, również wzbogacenie się na zgromadzonym kapitale.

* 1. **Cel projektu**

Celem świecącym tworzenia projektu jest utworzenie przykładowego systemu aplikacji internetowej obsługującej: zarządzanie, monitorowanie oraz planowanie budżetu domowego. W dużej mierze zakres pracy opiera się na analizie danych dostarczonych do oprogramowania, a następnie przetworzeniu i zamodelowaniu materiałów w sposób bardziej przejrzysty, opierający się w większym stopniu na funkcjach poznawczych standardowego użytkownika końcowego.

Jako główny cel stawia się wytworzenie archetypu aplikacji przy pomocy technologii uważanych przez środowisko programistyczne, jako metodyki na czasie, rozwijające się przez ostatnie lata w dynamicznym tempie, jak również takie zajmujące wysokie pozycje w popularności względem pozostałych.

* 1. **Środki użyte przy pracy**

Do utworzenia aplikacji internetowej w architekturze *Model-View-Controller[[3]](#footnote-3)*, planowo do celów implementacyjnych zostaną użyte nowoczesne szkielety aplikacyjne w warstwie backendu i frontendu.

Podzielono na główne części:

* Backend – aplikacja napisana w języku Kotlin w wersji 1.5.21 kompilowanych na Javie w wersji 11, przy użyci Spring Boot Framework w wersji 2.5.4 i towarzyszącym jej mniejszych części modułów. Ukierunkowana na udostępnienie dla warstwy frontendu internetowej API, potrzebnej do komunikacji z bazą danych i przetwarzania danych po protokole http.
* Frontend – aplikacja w warstwie wizualnej, napisana przy pomocy języka TypeScript w wersji 4.4, oraz na bibliotece React.js w wersji 17.0. Komunikacja zaimplementowana przy pomocy protokołu http.
* Baza danych – dostępna na licencji otwartego oprogramowania system bazy danych PostgreSQL w wersji 3.7.

Charakterystyka narzędzi użytych w aplikacji oraz powyższych technologii wraz z potrzebnymi mniejszych modułów użytymi do implementacji, ich opis i historia została dokładniej opisana w rozdziale nr 3 – Rozwiązania technologiczne.

* 1. **Kompozycja całości**

Efektem całości jest otrzymanie aplikacji pozwalającej na podstawowe monitorowanie wydatków w budżecie domowym, przy pomocy oprogramowania udostępnianego bezpośrednio na przeglądarce użytkownika końcowego.

W następnych częściach rozdziałów przygotowanej pracy omówione zostały koncepcje i funkcjonalności systemowe wyciągnięte w oparciu o zapotrzebowania i funkcjonalności w tego typu aplikacjach. Całość aplikacji nie zostałaby poprawnie zaimplementowana bez wiedzy o nowoczesnych systemach służących do implementacji logiki w oprogramowaniu, o której więcej mówi rozdział trzeci. Czwarty rozdział w całości poświęcony jest na omówienie zagadnień implementacyjnych w aplikacji, podzielonym z zachowaniem najważniejszych bibliotek i języków użytych podczas opisywania rozdziału trzeciego. Zakończeniem pracy jest ostatni rozdział poświęcony podsumowaniu i omówieniu działania podstawowych funkcjonalności znajdujących się w zaimplementowanej aplikacji.

1. Koncepcje, funkcjonalności i architektura aplikacji

Kryterium podstawowym jest utworzenie aplikacji internetowej przy pomocy dostępnych narzędzi, które pozwolą na rozdzielenie jej na dwie aplikacje: jedną serwerową pozwalającą na wymianę danych z bazą oraz przetwarzającą zapytania i drugą reprezentująca szatę graficzną dla użytkownika. W niniejszym rozdziale opisywane są wymagania systemowe, na podstawie których należy zaimplementować system. W dalszej części rozdziału przedstawione będzie podsumowanie planowania w aplikacji. Natomiast w ostatniej części rozdziału dostępne są diagramy implementowanych w następnych rozdziałach funkcjonalności.

* 1. Analiza podobnych rozwiązań dostępnych na rynku

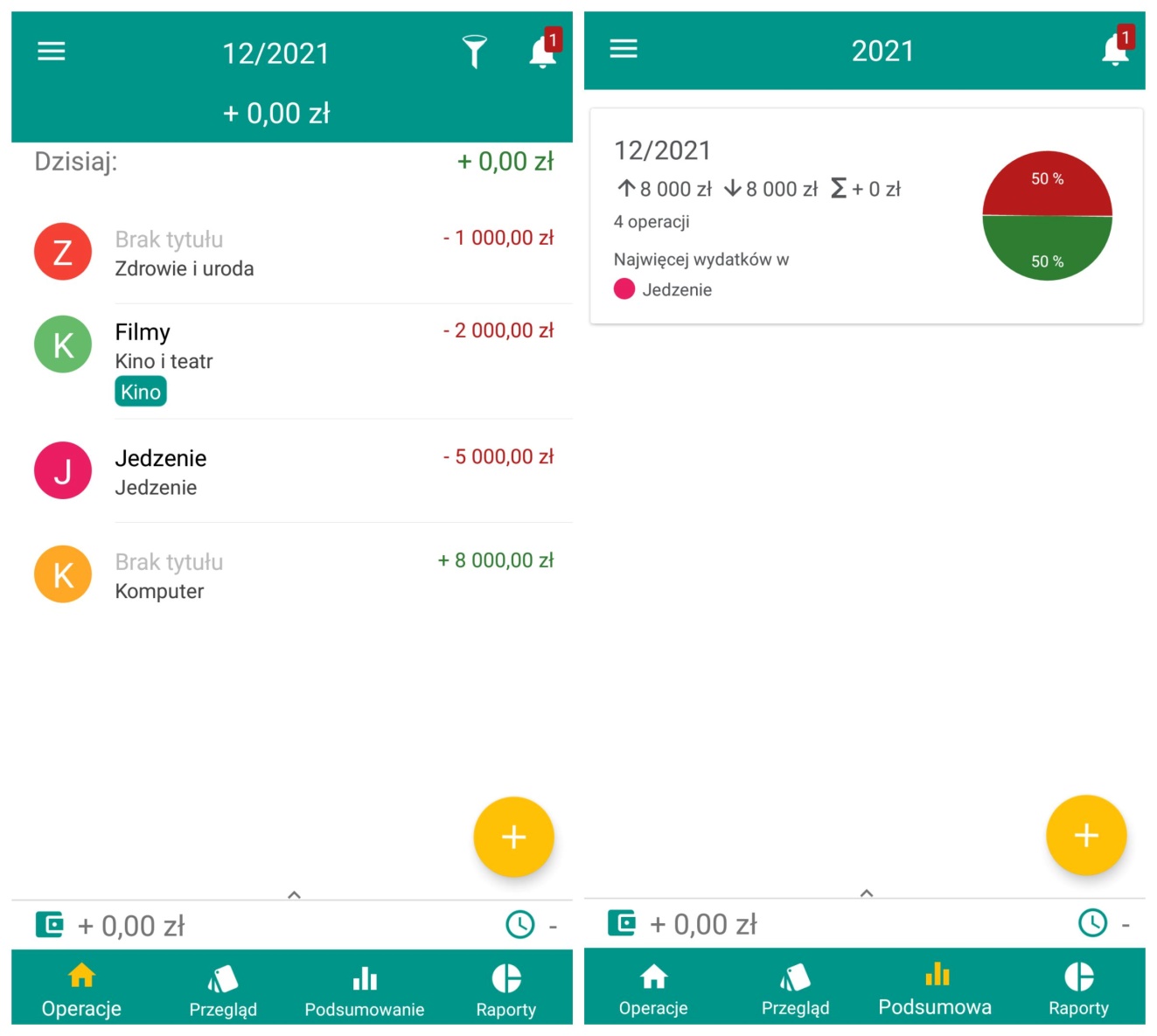
Przed rozpoczęciem pracy nad aplikacją ważnym punktem w procesie planowania aplikacji jest zaobserwowanie aktualnej sytuacji na rynku aplikacji zaprojektowanych w przeszłości. W oparciu na opinie innych użytkowników na temat oprogramowania wydanego pod szyldem innych twórców oraz na własnych obserwacjach można w szybki sposób wymyślić i w późniejszym czasie również podjąć się naprawy błędów napotkanych uprzednio już w własnej wersji aplikacji.

Analiza przeprowadzona została na podstawie dwóch aplikacji, znajdujących swoje zastosowanie w organizacji finansów domowych oraz osobistych wydatków. Obie aplikacje występują na rynku już od dłuższego czasu przez co zostały już odpowiednio przetestowane przez użytkowników.

**Moje Finanse – 7cssolutions[[4]](#footnote-4)**

Znajdująca swoich odbiorców na sklepie Google Play, w ilości ponad jednego miliona pobrań oraz posiadająca pokaźną liczbą pięćdziesięciu tysięcy ocen z niebywale wysoką średnią. Niezwykłym atutem aplikacji wydaję się być jej dostępność oraz fakt tego, że jest bezpłatna - pomijając fakt obecności reklam zajmujących pewną część interfejsu użytkownika. Możliwe jest również używanie programu w wersji płatnej – premium.

Po otwarciu aplikacji odbiorca jest w stanie zauważyć przyjemny dla oka interfejs użytkownika. Warstwa wizualna wydaję się być poprawna i czysta jednak nie jest ona wysoce rozwinięta. Odbiorca to samo zobaczy również po przejrzeniu podstawowych wykresów dostępnych w aplikacji, nie są one szczegółowe a jedynie pokazują podstawowe informacje potrzebne do zobrazowania aktualnej sytuacji finansowej.



Rysunek 1 Aplikacja Moje Finanse; Poglądowe operacje dostępne w aplikacji; źródło: opracowanie własne w aplikacji; https://play.google.com/store/apps/details?id=com.sevencsolutions.myfinances; dostęp 21.12.2021 r.

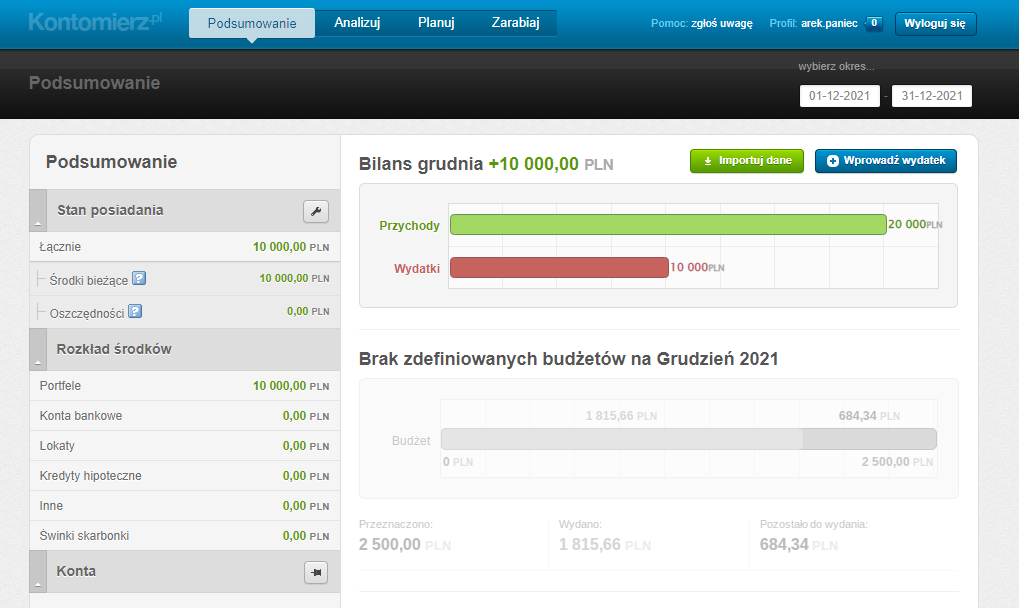
Jako szczególną zaletę w „Moje Finanse” można uznać duża ilość kategorii jakie użytkownik może wybrać podczas dodawania operacji do swojego portfela w aplikacji. Całokształt można ocenić jako intuicyjna z ciekawymi elementami statystyk wydatków niestety nieco ograniczoną, przy zaawansowaniu wykresów.

Największą wadą aplikacji jest jej ograniczenie wyłącznie do platformy Android. Po utworzeniu wszystkich operacji w swoim wirtualnym portfelu, użytkownik jest skazany na dalsze tworzenie ich wyłącznie na telefonie z systemem od firmy Google. Dostępność aplikacji nie jest możliwa poprzez system iOS. Sytuacja wydaję się również niesprawiedliwa przy dostępie poprzez aplikację internetową, z którą komunikacją możliwa jest wyłącznie przy pomocy konta premium.

**Kontomierz - Kontomatik[[5]](#footnote-5)**

Kontomierz dostępny jest dla użytkowników jako aplikacja internetowa oraz dodatkowo w wersji na urządzenia mobilne na systemy Android i iOS. Produkt kontentowany głównie pod użytkowników z polski jednak całkiem popularny o czym może świadczyć ponad dziesięć tysięcy pobrań w sklepie Play.

Interfejs użytkownika aplikacji jest wyjątkowo prosty, próżno w nim szukać zaawansowanych diagramów. Kontomierz nie wydaję się być wyjątkowo intuicyjny, szata graficzna wydaję się być nieodświeżana od długiego czasu, przez co wszystko wygląda na niedopracowane. W przypadku wersji na platformy mobilne, warstwa wizualna jest tworzona przy użyciu najprostszych komponentów dostępnych w latach tworzenia jej. Używalność aplikacji możliwa jest praktycznie wyłącznie przy użyciu przeglądarki internetowej, co łatwo zauważyć po niskich ocenach wystawionych w mobilnych sklepach z oprogramowaniem. Pomimo większego zaangażowania w produkcje przeglądarkowej wersji aplikacji internetowej brak reaktywności komponentów powoduje, że używanie produktu na telefonach jest niezwykle utrudnione, a biorąc pod uwagę okrojoną wersję aplikacji mobilnej czasami wręcz niemożliwe.



Rysunek 2 Strona główna Kontomierz; Poglądowe operacje dostępne w aplikacji; źródło: opracowanie własne w aplikacji; secure.kontomierz.pl; dostęp 21.12.2021 r.

Pomimo wad dostrzeganych przy warstwie wizualnej Kontomierz jest dobrze rozwinięty dzięki możliwości podłączenia go do praktycznie każdego możliwego konta bankowego w celu pobrania aktualnych wydatków. Kategorie dostępne w aplikacji również wydają się wystarczające przez co aplikacja zachowuje swoją użyteczność i jest spójna.

**Podsumowanie rozwiązań dostępnych na rynku**

Po zapoznaniu się z niektórymi produktami dostępnymi na rynku od lat można dojść do wniosku, że niektóre funkcjonalności są czasami wyjątkowo niedopracowane przez twórców pomimo upływu lat. Ciągłe utrzymywanie różnych modułów na platformy mobilne wydaję się być niezwykle ciężkie podczas nieregularnego tworzenia produktu przez małe zespoły programistów.

Reaktywna aplikacja internetowa wydaje się odpowiednim rozwiązaniem pomiędzy ciągłym utrzymywaniem dodatkowych modułów w celu zachowania dostępności na najpopularniejszych systemach operacyjnych dla platform mobilnych. Produkt wykonany przy użyciu nowoczesnej biblioteki frontendowej może pozwolić na zachowanie spójności w projektowanej aplikacji.

* 1. Wymagania systemowe i koncepcje

Przeprowadzenie analizy rynkowej w oparciu o inne dostępne produkty, odkryciu wad jak i najpotrzebniejszych wymagań owych przedmiotów pozwoliło w wykonaniu propozycji szablonu oraz zaprojektowaniu nowej aplikacji na rynek wraz z potrzebnymi dla niej funkcjonalnościami. Jako punkt docelowy objęto prototyp aplikacji pozwalający na budżetowanie dowolnych kwot na podstawie dostępnych kategorii oraz przedstawianie ich na prostych oraz logicznie zaprojektowanych wykresach pozwalających na szybką analizę wydatków.

**Wymagania funkcjonalne, biznesowe oraz właściwości aplikacji**

Po wcześniejszej diagnozie właściwości jakie może potrzebować aplikacja do budżetowania codziennych finansów wyklarowany został model poglądowy projektu. Wymagania funkcjonalne jakie oczekuje użytkownik aplikacji kształtują się następująco:

1. Możliwość samodzielnej rejestracji użytkownika w systemie oraz zapisanie go na bazie danych, przy pomocy prostego i bezpiecznego szyfrowania hashującego hasło bcrypt[[6]](#footnote-6).
2. Umożliwienie użytkownikowi szybkiego logowania do serwisu.
3. Logowanie powinno być walidowane poprzez token dostępu.
4. Użytkownik powinien mieć możliwość dodania odpowiedniej dla siebie liczby wirtualnych portfeli w liczbie nie większej niż 5.
5. Użytkownik powinien mieć możliwość dodać do swojego portfela operacje przychodów i rozchodów.
6. Przychody i rozchody powinny być opatrzone wymaganą kwotą, datą i kategorią oraz dodatkowo nieobowiązkową nazwą i miejscem przypisania operacji finansowej.
7. Podczas dodawania nowej operacji użytkownika obowiązuje walidacja na odpowiednich polach.
8. Użytkownik powinien mieć umożliwioną modyfikację wszystkie operacje znajdujące się w swoim wirtualnym portfelu.
9. Podczas modyfikacji utworzonej wcześniej operacji użytkownika obowiązuje walidacja na odpowiednich polach.
10. Użytkownik powinien mieć możliwość filtrowania wszystkich operacji ze względu na wysokość kwoty, nazwy, daty operacji.
11. Użytkownik powinien mieć dostęp do wykresu przychodów i wydatków na podstawie wszystkich dodanych operacji z podziałem na miesiące przypisane do każdej operacji.
12. Każda operacja dodana do systemu powinna być zapisywana na bazie danych i automatycznie dodawana jako składowa w wykresie przychodów i wydatków w aplikacji.
13. Po wybraniu odpowiedniego miesiąca użytkownik powinien mieć możliwość sprawdzenia raportu wybranego terminu z podziałem na kategorie.

Wymagania natury niefunkcjonalnej jakie oczekuję się przy projektowaniu aplikacji:

1. Aplikacja jest zobowiązana do zabezpieczeń przy pomocy JWT[[7]](#footnote-7) z dziesięciominutowym czasem wygaszenia.
2. Aplikacja jest zobowiązana umożliwiać dostęp poprzez jak największą liczbę platform usługowych, przy pomocy łącza internetowego.
3. Od aplikacji oczekuję się responsywnego dopasowania komponentów w celu użycia na różnych platformach.
4. Aplikacja jest zobowiązana działać na najpopularniejszych przeglądarkach internetowych. Środowisko uruchomieniowe aplikacji i to w jakich porusza się użytkownik powinno operować na projekcie Chromium.
5. Wszystkie narzędzia użyte do zaprojektowania aplikacji powinny być udostępnione na zasadzie licencji otwartego oprogramowania bądź udostępnionej licencji dla sektora edukacyjnego.

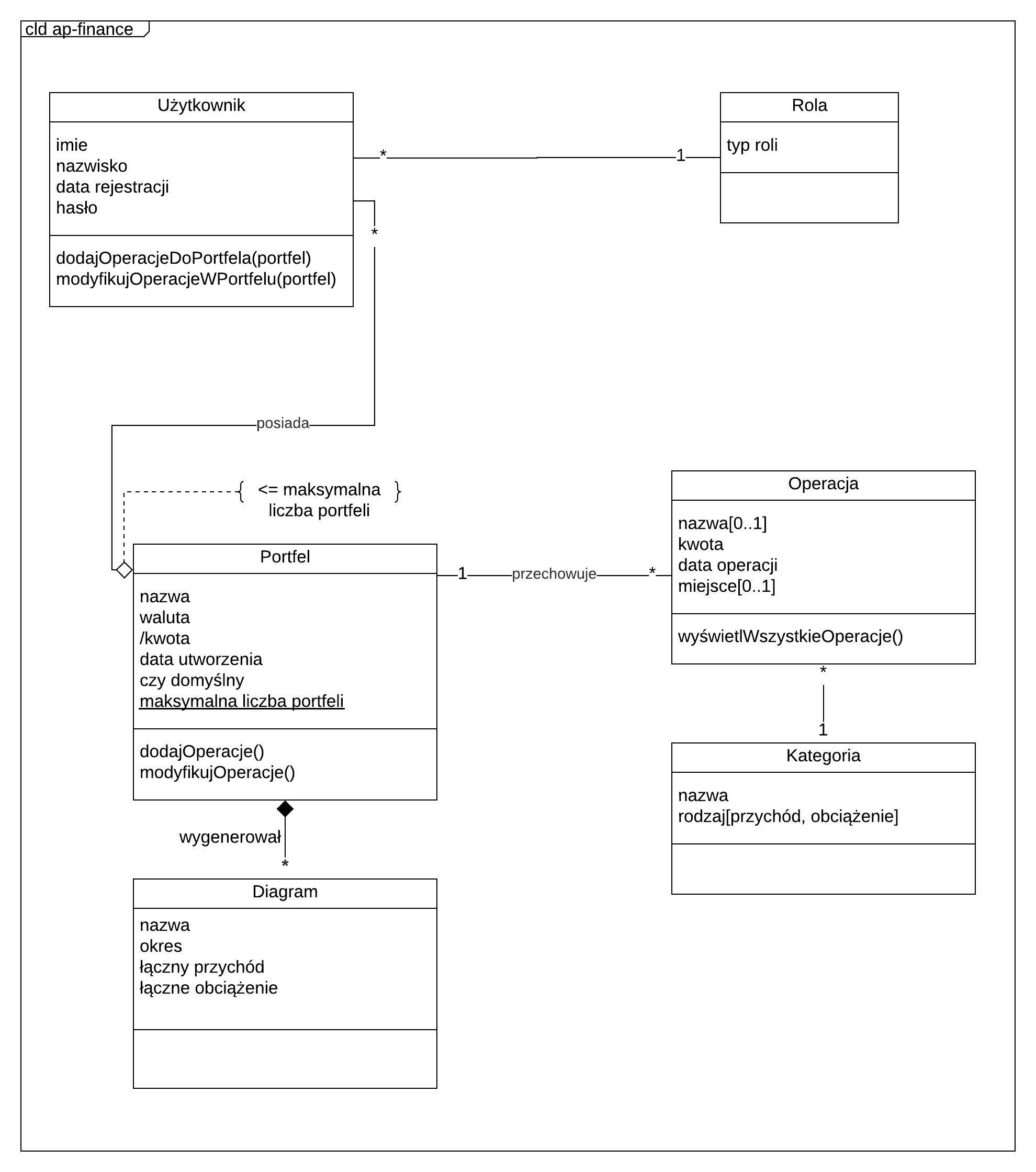
Wymagania biznesowe:

1. Niezalogowany do systemu użytkownik jest automatycznie przenoszony na stronę logowania, z której może przejść do strony rejestracji nowego użytkownika.
2. Niezalogowany użytkownik nie ma dostępu do podstawowych funkcjonalności oferowanych przez system.
3. Zalogowany użytkownik jest w stanie raportować swoje operacje finansowe operując na następujących kategoriach wydatkowych: rachunki, jedzenie, transport, wynajem, usterki, hobby, zdrowie, przyjemności, raty, kredyt oraz inne.
4. Zalogowany użytkownik jest w stanie raportować swoje przychody operując na następujących kategoriach: pensja, przychód nieokresowy, dodatkowa aktywność finansowa.
5. Po pojawieniu się co najmniej jednej dostępnej w systemie operacji powinien wyświetlić się pierwszy diagram obrazujący wydatki i przychody wyselekcjonowany na miesiące, w których wykonywane były operacje, jak i poprzednie miesiące.
   1. Szablony diagramów modelujących oraz architektura aplikacji

Podstawą organizacji pracy w kierunku implementacji systemów informatycznych jest jego wcześniejsze zaprojektowanie. Narzędzia takie jak UML[[8]](#footnote-8) od dziesiątek lat pomagają deweloperom i analitykom w codziennej pracy przy projektowaniu systemów informatycznych. Bez uprzedniego zaprojektowania systemu, proces następującego projektowania aplikacji może się wydłużać.

Dodatkowo początkowe modelowanie systemu pozwala w jakimś stopniu na ponownym przemyśleniu działania owego projektu i możliwym oszczędzeniu czasu przy odkryciu błędów na fazie początkowego projektowania.

**Diagram klas**



Rysunek 3 Diagram klas przedstawianej aplikacji; opracowanie własne

Statyczny diagram klas pozwala na przedstawienie w oparciu o modele obiektowe poprzez zilustrowane struktury połączeń pomiędzy klasami. W ukazanej wcześniej strukturze systemu jest możliwość wyeksponowania następujących modeli na podstawie danych:

* Użytkownik – tabela zawierająca informacje o kliencie aplikacji. Znajdujące się w niej informacje przechowywane są w celu zebrania podstawowych danych w celu autoryzacji w systemie.
* Portfel – podstawowa struktura zawierające połączenie pomiędzy użytkownikiem aplikacji oraz listą operacji jakie znajdują się w systemie.
* Operacja – tabela zapisująca informacje o przychodach i wydatkach używanych do sumowania ich do tabeli portfela.
* Kategoria – tabela zawierająca podstawowe instancje pozwalające na skategoryzowanie wydatków w tabeli operacji.
* Diagram – tabela zawierająca ustrukturyzowany ze względu na miesiące wydatkowe podsumowania operacji finansowych.
* Rola – podstawowa instancja informująca o roli użytkownika. Tworzona z zamiarem dalszego rozwijania modułów z częścią biblioteki do zabezpieczeń użytkowników i dodawania personalizowanych uprawnień.

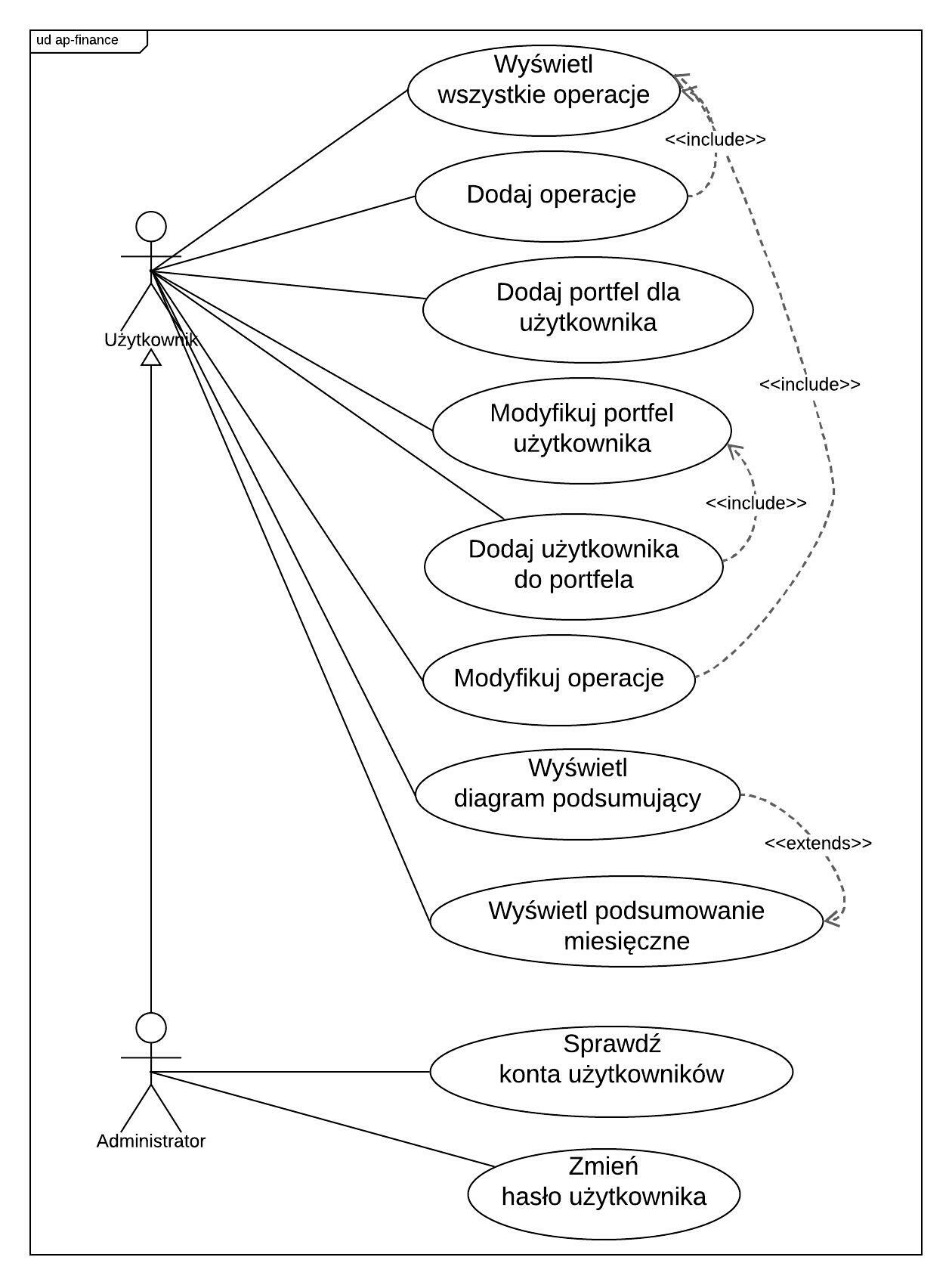
Diagram klas względem aplikacji może się w niektórych momentach różnić z rzeczywiście zaimplementowanym względem składników dostępnych w projekcie. Bazowo planowane jest użycie ORM, który w łatwy sposób pozwoli na mapowanie bazy danych na obiekty.

Przez użycie narzędzia ORM pominięto w diagramie podstawowe zależności pomiędzy komunikującymi się pomiędzy sobą tabelami. Takimi elementami mogą być między innymi numery identyfikujące – id bardzo popularne w bazach danych jako klucze główne, czy chociażby obiekty używane do mapowania zależności typu jeden do wielu, jeden do jednego czy wiele do wielu znane też w bazach danych jako klucze obce.

**Diagram przypadków użycia**

Podstawowym założeniem istnienia diagramów przypadków użycia jest możliwość uchwycenia przez niego dynamicznych czynności, które występują w tworzonym systemie informatycznym. Diagram używa się w przypadku czynności występujących po analizie, odpowiednio weryfikuję się informacje otrzymane w wymaganiach.

Po procesie wystawia się czynności/umiejętności niektórych aktorów występujących w systemie, które pozwalają na działanie na najważniejszych funkcjach odpowiadających za działanie projektowanego systemu. Pozwala on pokazać interakcje jakie występują pomiędzy użytkownikami aplikacji oraz do mapowania ich na przyjemne dla oka diagramy, rzucające w oczy wszystkie dostępne funkcje.

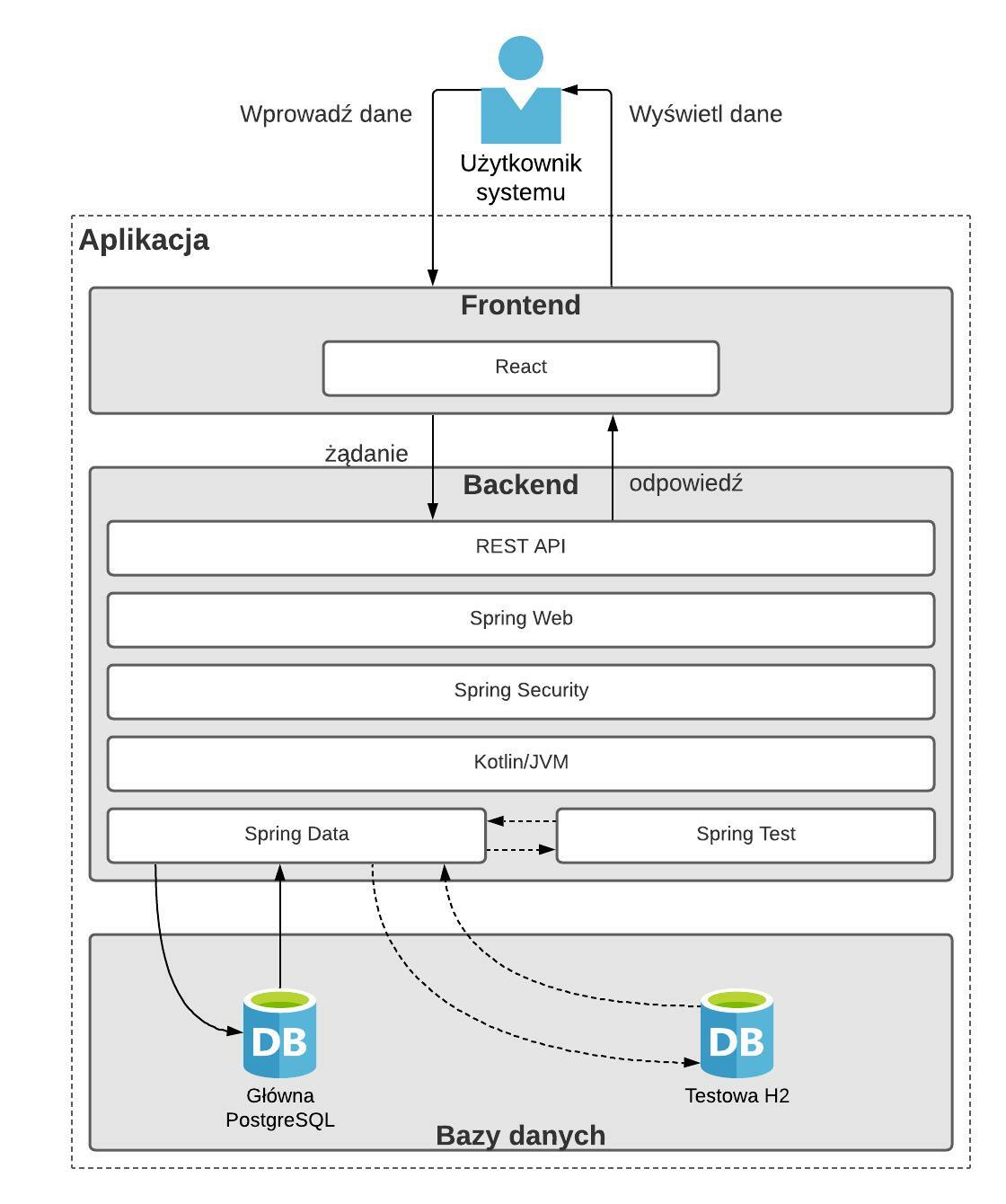


Rysunek 4 Diagram przypadków użycia przedstawianej aplikacji; opracowanie własne

Z diagramu łatwo wyciągnąć informacje o dziedziczeniu wszystkich funkcjonalności przez aktora Administrator oraz dołączeniu dla niego przydatnych w monitorowaniu aplikacji funkcjonalności takich jak sprawdzanie innych kont oraz zmiana ich haseł, czy modyfikacja innych rekordów manualnie.

* 1. Architektura aplikacji

Architekturę aplikacji najprościej można zobrazować sobie jako szkielet wyświetlający najbardziej widoczne interakcje oraz hierarchie dostępnych w tym miejscu komponentów oprogramowania, które jest projektowane, takich jak interfejsami użytymi w programie, bazami danych oraz warstwami oprogramowania.



Rysunek 5 Diagram architektury systemowej; opracowanie własne

Wartkie spojrzenie pozwala na pozyskanie wystarczająco dużej ilości informacji dotyczącej tworzonego systemu informatycznego. Szablonem tworzonego oprogramowania jakim jest aplikacja do zarządzania finansami jest stos technologiczny złożony z rozwiązań frontendu połączonego z backendem, który następująco łączy się z wewnętrzną bazą danych. Więcej informacji na temat charakterystyki oraz historii rozwiązań użytych w architekturze tworzonego oprogramowania można znaleźć w następnym rozdziale.

Charakterystyka wydawanego światu produktu spełnia ważne fundamenty tworzenia aplikacji. Wydajność systemową potrafi dostarczyć użycie odpowiedniej biblioteki w procesie komunikacji, która używana i chwalona jest w środowiskach korporacyjnych już od wielu lat. Dzięki rozdzieleniu fundamentów systemu na dwa środowiska zyskano charakterystykę elastyczności, pozwala ona na modyfikację obu komponentów niezależnie od siebie co przy aplikacjach monolitycznych jest niezwykle utrudnione, a błędy potrafią być ciężkie do namierzenia. Dzięki dołączeniu plikowej bazy danych oprogramowanie zyskało możliwość testowania integracyjnych komunikatów dokładnie tak jak wygląda to w procesie produkcyjnym.

Pomimo zalet w wydajności i dalszym rozwijaniu jakie potrafi przedstawić powyższa architektura należy pamiętać, że odpowiednio dobrane komponenty aplikacyjne nie zawsze potrafią zastąpić umiejętności i doświadczenie w wytwarzaniu oprogramowania informatycznego przy użyciu innych – mniej popularnych lub równocześnie branych za zupełnie niewydajne. Dużą częścią sukcesu i dalszego rozwijania systemu jest wybór preferowanych narzędzi przy produkcji ponad cenę werdyktów napotkanych na temat wydajności.

**Architektura rozwiązania interfejsów REST’owych[[9]](#footnote-9)**

W tworzeniu komunikacji poprzez interfejs REST ważne jest zastosowanie standardów dotyczących nazewnictwa końcówek potrzebnych do porozumiewania się zewnętrznych serwisów z tworzonym przez programistę interfejsem sieciowym.

Zachowanie jednego standardu w obrębie aplikacji nie tylko pozwala na szybsze tworzenie przestrzeni nazewniczej bez napotykania błędów w wypadku użycia tego samego nazewnictwa końcówki REST’owej co inny interfejs, ale także przypadkowych błędów w komunikacji podczas tworzenia warstwy łączącej się z uprzednio utworzonym mapowaniem kontrolerów.

Jako standard w komunikacji z internetowym api w całej aplikacji starano się używać konwencji zaproponowanej w artykule Thomasa Busha[[10]](#footnote-10) i później odpowiednio przyjęto następującą strukturę:

* /api/{nazwa kontrolera}
* /api/{nazwa kontrolera}/{numer id w bazie}
* /api/{nazwa kontrolera}/{numer id w bazie}/{nazwa zmiennej}

Wyjątek mogą stanowić nazwy używane do testowania dedykowanych końcówek utworzonych w celu sprawdzenia zabezpieczeń używanych przy pomocy Spring Security oraz wydawanych przez niego ról dla użytkowników. Jako drugi wyjątek zainicjowano ścieżki przy komunikacji z autentykacją użytkowników przy pomocy Spring Security, wskazujących na funkcje logowania i rejestrowania nowych klientów aplikacji. Wszystkie końcówki wyrażane są w następującej formie:

* /api/test/all
* /api/test/user
* /api/test/admin
* /api/auth/register
* /api/auth/login

Wszystkie końcówki odpowiednio powinny być opakowane w encje pozwalające na zwracanie dodatkowo kodów błędów przy komunikacji poprzez kontroler w aplikacji. Idąc następującą formułą przykładowe końcówki wraz z wyjaśnieniem będą wyglądały następująco:

* GET /api/operations – zwrócenie listy wszystkich operacji.
* GET /api/operations/1 – zwrócenie dedykowanej operacji.
* POST /api/operations – utworzenie nowej operacji na bazie danych oraz zwrócenie jego ciała.
* POST /api/operations/1 – modyfikacja ciała operacji na bazie danych oraz zwrócenie jej nowego ciała.
* DELETE /api/operations/1 – usunięcie z bazy danych operacji
* GET /api/user/1/wallets – zwrócenie z bazy danych wszystkich portfeli użytkownika

1. Rozwiązania technologiczne

Tematem aktualnego rozdziału zostały postawione główne technologie użyte podczas procesu tworzenia rozwiązań opisywanej aplikacji internetowej. W poniższych podrozdziałach zaproponowany został podział na mniejsze części, biorąc pod uwagę to w jakim stopniu można je podzielić ze względu na używane w nowoczesnych aplikacjach, warstwy technologiczne.

Wybór języków programowania oraz używanych w procesie budowania aplikacji bibliotek, został wyselekcjonowany ze względu na nowoczesność, ogólnodostępność oraz trendy występujące aktualnie na rynku programistów, oraz polecanych przez nich technik.

* 1. Backend

Warstwa backend, bardzo często nazywana także częścią serwerową aplikacji internetowej, która w praktyce jest całkowicie niezauważalna przez użytkownika końcowego. To właśnie po stronie serwera stoi obsługa całej logiki biznesowej oraz procesów wymiany informacji wewnętrznych oprogramowania tworzonego dla użytkowników jak i również odpowiednie sprawdzenie czy ze strony klienta aplikacji. Aby nie pojawiały się krytyczne błędy, mogące zagrozić niepoprawnemu funkcjonowaniu aplikacji.

W najbardziej podstawowych aplikacjach część serwerowa odpowiada również za dostarczenie potrzebnych informacji do bazy danych, która często jest jednocześnie brana jako cząstka całej warstwy backend.

* + 1. Kotlin

Kotlin, czyli powstały w lipcu 2011, statycznie typowany język programowania. Jego pierwsza oficjalnie stabilna wersje wyszła dopiero w lutym 2016 roku. Twórcy języka, czyli firma JetBrains, podczas procesu tworzenia, projektowali go z myślą o pełnym współdziałaniu z innym, bardzo popularnym językiem - Java oraz JVM - jego wirtualną maszyną.

Od późniejszych wersji dostępny jest również Kotlin/JS, czyli łatwe tłumaczenie kodu napisanego, w języku Kotlin na JavaScript, pozwalające na pisanie również frontendu aplikacji bezpośrednio w języku Kotlin. Głównym zamysłem powstania Kotlin/JS było wytworzenie własnej logiki pisania części frontendu aplikacji, jak również części backendowej w przypadku używania Node.js jako głównej technologii użytej do pisania logiki serwerowej[[11]](#footnote-11).

Projektanci języka, trzymali się tego, że Kotlin ma być obiektowym językiem, ukierunkowanym na działanie w środowisku produkcyjnym oraz poprawiającym znane błędy z Javy. Współdziałanie z kodem języka Javy, zostało zachowane, aby w prosty sposób umożliwić firmom, operującym od wielu lat na Javie na stopniową migrację na produkt firmy JetBrains.

Niektóre charakterystyczne cechy Kotlina[[12]](#footnote-12):

Bezpieczeństwo względem błędów z *null’ami* – w przeciwieństwie do często spotykanej wady języka Javy, bazowo, wszystkie zmienne ustawiane są przez kompilator jako *non-null,* co już częściowo pozwala wyeliminować niechciane wyjątki. Przy pomocy operatora „?.”, Kotlin pozwala na bezpieczne wywołanie metody lub zmiennej. W przypadku, kiedy któraś z wartości okaże się *null’em*, właściwość nie zostanie wywołana. Pomocny okazuję się równie operator Elvis „?:”, który pozwala zastąpić wartością domyślną, jeżeli przypisywana przez kompilator wartość okaże się *null’em*.

* Brak możliwości wywołania sprawdzonych wyjątków (*Checked Exception*) – taka funkcjonalność została całkowicie pominięta, przez twórców Kotlina.   
  W języku Java, używanie podstawowych metod, często wiąże się z przymusowym obudowaniem wywołania instrukcją *try-catch*. Omawiane rozwiązania mogą prowadzić do obniżenia jakości kodu, jak i zaciemnienia rzeczywistego obrazu działania metody.
* Korutyny w programowaniu współbieżnym – Początkowo w języku Kotlin dostępne były wyłącznie, dobrze znane z innych języków wątki. Korutyny, podobnie do wątków pozwalają na obsłużenie, większej ilości procesów jednocześnie. Na korzyść, w przeciwieństwie do wątków, korutyny pozwalają na dynamiczne zarządzanie procesami. Po zawieszeniu działania, uprzednio używany wątek procesora, jest zwalniany i może zostać użyty przez inny proces, co pozwala na bardziej przemyślane zarządzanie wykorzystywanymi wątkami procesora.

Prostota oraz mądrze przemyślane decyzje rozwojowe języka, pozwalają Kotlinowi od wielu lat znajdować się w listach najbardziej dynamicznie rozwijających się języków na świecie. W rozwoju swoją cegiełkę dołożył także Google, który na swojej konferencji I/O w 2017 roku, ogłosił Kotlina oficjalnym językiem programowania Androida – platformy posiadającej 73%, udziału rynku mobilnego na całym świecie[[13]](#footnote-13).

Widząc dotychczasowy rozwój, języka firmy JetBrains, można oczekiwać od Kotlina, że z pewnością będzie dalej piąć się w górę w rankingach popularności wszystkich języków programowania. Jednocześnie wprowadzając dodatkowe możliwości dla deweloperów pracujących na platformie wirtualnej maszyny Javy.

* + 1. Spring Boot

Początkowo całkowity produkt jako Spring Framework rozwijany był już od października 2002 roku przez Rod Johnson. Stworzony na platformie Java Enterprise Edition jako bardziej przyjazne programiście rozwiązanie, w pisaniu backendu w zaawansowanych aplikacjach internetowych. Głównymi powodami, dla których deweloperzy oprogramowania stawiali na szkielet Spring Framework, to ukierunkowanie pracy programisty na funkcjonalność aplikacji i zmniejszenie nakładów pracy na pisaniu powtarzającego się kodu programu.

Rewolucyjny sposób na pisanie servletów, czyli mniejszych części aplikacji wystawionych na działanie serwera oraz następnie zwracanych zawartość opakowaną w protokół http. W bardzo krótkim czasie pozwolił Spring Framework’owi objąć szczyty popularności i przedstawić go jako niezbędne narzędzie, codziennie używane podczas pisania biznesowych aplikacji działających w obszarze internetowym.

Kolejnym wielkim krokiem w rozwoju szkieletu aplikacji Spring’a był moduł Spring Boot. Powodem, dla którego powstał Spring Boot, był lepszy, ale nadal nie do końca zrozumiały sposób zarządzania projektem. W odświeżonej wersji twórcy próbowali pozbyć się skomplikowanego sposobu zarządzania ziarnami w aplikacji przy pomocy plików XML. W tworzeniu aplikacji webowych, Spring Boot wniósł także funkcjonalności automatycznej konfiguracji zależności oraz mechanicznego wystawiania projektu do web serwera Jetty lub Tomcat, co pozwoliło na możliwość szybszego stawiania aplikacji, a wcześniej wymagało skomplikowanego procesu wdrażania projektu do zewnętrznych plików War.

Diagram

Description automatically generated

Rysunek 6 Pivotal Software; Schemat modułów Spring Runtime; źródło: https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch01s02.html

Moduły Spring Framework:

* *Core* – Najbardziej kojarzone ze Spring’iem funkcjonalności. Wprowadza do aplikacji popularne właściwości takie jak wstrzykiwanie zależności do ziaren aplikacji jak i języka wyrażeń Spel, pozwalające rozwijać dynamiczne odwołania do klas za pomocą ciągów znaków.
* *Data* – Zawierający potrzebne, do łączenia się z bazą danych funkcjonalności przy pomocy JDBC lub ORM.
* *Web* – Moduł wyciągający użyteczność łączenia się użytkownika zewnętrznego przy użyciu protokołu http.
* *AOP* i *Instrumentation* – Moduł wprowadzający programowanie oparte na aspektach. Dodatkowo pozwala na wprowadzenie przydatnych serwisów, poprzez instrumentowanie kodu bajtowego do metod, ukierunkowane na gromadzenie danych.
* *Test* – Moduł zawierający klasy potrzebne, do przetestowania projektu przy użyciu funkcjonalności takich jak testy jednostkowe i integracyjne.

Podstawowym cyklem rozwoju aplikacji, jest używanie ziaren dostępnych w tworzonym programie przy użyciu różnego rodzaju adnotacji, dostępnych bezpośrednio ze Spring Framework lub wyciągania ich prosto z kontekstu aplikacji. Podstawowa funkcjonalność wstrzykiwania zależności pomiędzy klasami i zmiennymi, realizowana jest właśnie przez ziarna Java.

Dobrodziejstwo udostępnione przez tworzenie nowych obiektów, może być realizowane poprzez wzorzec projektowy, wstrzyknięcia zależności, udostępniana właściwość pozwala na oczyszczenie kodu, z powtarzających się wywołań nowych obiektów o tej samej budowie. Zakres tworzenia nowych bean’ów można podzielić na pięć typów[[14]](#footnote-14):

* *Prototype* – debiutująca instancja, przy następnym wstrzyknięciu
* *Singleton* – taka sama na wszystkie aplikacje
* *Session* – przy łączeniu kolejnej sesji
* *Request* – przy nowym żądaniu http
* *Websocket* – taka sama dla nowego łączenia z WebSocket
* *Application* – taka sama na jedną aplikacje

W Spring’u dostępne są adnotację, oznaczane poprzez „@” oraz słowo opisujące zależność, które pozwalają na dodanie przydatnych funkcjonalności dla klas, jak i zmiennych. Niektóre z przykładowych zależności (każdy poprzedzony „@”):

* *Configuration* – zależność bazowa, definiująca konfigurację ustawioną poprzez kod.
* *Bean* – zależność ustawiająca przedstawianą metodę jako ziarno.
* *Component* – zależność stereotypu, ustawia klasę jako ziarno w kontekście.
* *Controller* – opisuje klasę jako kontroler potrzebny do WebApi.
* *Service* – identyfikująca klasę jako implementacja dla logiki biznesowej.
  + 1. Hibernate / Spring Data JPA

Nienaturalna wydaje się dyskusja o opisywanych technologiach, bez znajomości pojęcia ORM. ORM jest z definicji mapowaniem obiektów na relacje, technika używana w językach ukierunkowanych obiektowo, polega na utworzeniu z wcześniej zdefiniowanych obiektów, klas zaimplementowanych w wybranych języku np. Kotlin, na informację utworzoną w relacyjnej bazie danych. W praktyce okazuję się, że proces mapowanie polega na przetłumaczeniu podstawowych zależności na prosty kod w języku SQL. Translacje znane są z tego, że często pomijają dostępne w SQL składnie językowe, pozwalające na optymalizacje zapytań i tworzą relacje w bazie danych, w prymitywny sposób, zwiększając tym samym czas wywoływania, podstawowych interakcji z bazą danych.

Hibernate jest wysoce zaawansowanym projektem powstałym w maju 2001 roku, w celu gruntownego rozwiązania problemu powstałego w zarządzaniu trwałością danych z jakimi spotykali się twórcy oprogramowania w języku Java. Uczestniczy w procesie współpracy aplikacji z relacyjną bazą danych, pozwalając programiście na skupienie się na implementacji wymagań biznesowych w jak najbardziej przypominającym charakterystyką podstawę w tworzeniu klas domenowych. Praktycznie całkowicie pozwalając na przestawieniu się z implementacji kodu w języku SQL na dodawanie odpowiednich adnotacji bezpośrednio w klasach Javy. Hibernate jest implementacją uznawaną za nieinwazyjną, rozumiane przez to, że nie twórca oprogramowania nie jest zobligowany w użyciu wszystkich reguł i wzorców programistycznych dla Hibernate. Pozwala to na prostsze implementacje trwałości klas co pomaga na łatwiejsze wdrożenie działania wcześniej używanych klas domenowych na działanie w architekturze Hibernate.

Spring Data JPA, jak wcześniej zostało to wyjaśnione, jest częścią Spring Framework. Głównym celem powstania repozytorium Spring Data, jest częściowa redukcja często powtarzającego się kodu, potrzebnego do utworzenia podstawowych form działania przykładowo, używanych przez siebie ORM.

Różnicą pomiędzy dwoma rozwiązaniami jest to, że Spring Data JPA nie jest zaimplementowanym rozwiązaniem używanym jako ORM. Jest wyłącznie abstrakcyjnym dodatkiem i szablonem pozwalającym na wydajniejsze używanie, wybranego przez twórcę oprogramowania ORM.

Przykładowe adnotacje dostępne w Hibernate, przy użyciu Spring Data JPA -uzupełniane w kodzie przy pomocy dodania znaku „@”[[15]](#footnote-15):

* *Entity* – Znacznik pozwalający na użycie klasy jako ziarna w Spring Framework.
* *Table* – Podstawowy znacznik używany do ustawienia klasy jako przyszłej tabeli.
* *Id* – Znacznik używany do oznaczeniu zmiennej jako klucz główny.
* *OneToMany* – Znacznik relacji jeden-wiele, pozwala na oznaczeniu zmiennej jako dostępu do wielu obiektów przypisanych do głównego obiektu, realizowanego poprzez relacje w bazie danych.
* *ManyToOne* – Znacznik relacji wiele-jeden, pozwala na oznaczeniu zmiennej jako dostępu do obiektu przypisanego do głównego obiektu, realizowanego poprzez relacje w bazie danych.
* *ManyToMany* – Znacznik relacji wiele-wiele, pozwala na oznaczeniu zmiennej jako dostępu do wielu obiektu przypisanego do głównego obiektu, realizowanego poprzez relacje w bazie danych, pozwala na stworzeniu tabeli asocjacyjnej bezpośrednio pomiędzy dwoma klasami.
* *OneToOne* – Znacznik relacji jeden-jeden, pozwala na oznaczeniu zmiennej jako dostępu do obiektu przypisanego do głównego obiektu, realizowanego poprzez relacje w bazie danych.

Podsumowując Hibernate jest świetnym narzędziem pomocnym przy mapowaniu obiektów na relacje, w bardzo prosty sposób, przystępny dla twórców oprogramowania biznesowego. Pomimo znajomości minusów takich jak gorsza optymalizacja zapytań, podczas tworzenia bardziej zaawansowanych operacji, jest bardzo często używany w wielkich projektach.

* + 1. **Spring Security**

Spring Security jest częścią Spring Framework, pozwalająca na wprowadzenie do serwerowej części aplikacji funkcjonalności takich jak: logowanie, rejestracja, autoryzacja czy autentykacja, bazowo pozwala także na ochronę przeciwko niektórym, najpopularniejszym podatnościom aplikacji takim jak ataki CSRF.

Dzięki odrębnej części szkieletu Spring, twórcy oprogramowania mają ułatwione zarządzanie zabezpieczeniami w aplikacji. Framework bazowo dostarcza konfigurację pozwalająca na szyfrowanie haseł w najpopularniejszych możliwych sposobach kodowania. Szkielet pozwala na użycie bardzo popularnych *JSON Web Token (JWT)*, używanych w celu nadzorowania możliwości ciągłości korzystania z aplikacji internetowej przez użytkownika końcowego.

* + 1. JUnit / Spring Test

Testy jednostkowe, to rodzaj testowania aplikacji, w procesie którego odpowiednio testowane są poszczególne jednostki lub funkcje zaimplementowane podczas procesu tworzenia oprogramowania. Głównym celem jest szczegółowe przetestowanie opisanych wcześniej jednostek, w wyniku którego odnalezione zostaną błędy, a następnie wyeliminowane. Testowane funkcje najczęściej mają pojedyncze możliwe wyjścia. Testowanie oprogramowania jest procesem ciągłym, wyjątkowo ważne jest, aby testy zawierały jak najmniej nieszczelnych części. Testy pisane są w celu sprawdzenia błędów w dopiero co zaimplementowanych funkcjonalnościach, ale także na przyszłość, aby sprawdzać czy dodatkowe zmiany, nie wprowadzają niechcianych błędów w odpowiednio wcześniej utworzonych fragmentach kodu.

JUnit jest to specjalny framework przydatny do przeprowadzania testów jednostkowych w aplikacjach z wachlarza języków operujących na maszynie wirtualnej Java. Użycie funkcjonalności biblioteki JUnit w projekcie, pozwala na uzyskanie prostego sposobu przydatnego, do włączania testów jednostkowych. Dostęp do większości z nich można uzyskać poprzez użycie odpowiednich adnotacji podczas tworzenia funkcji testowych oraz klas. Logika użycia oraz pracy przy pomocy JUnit, jest wyjątkowo prosta, należy stworzyć odpowiednią funkcję, w środku której zademonstruje się działanie przykładowego kawałku gotowego kodu używanego w aplikacji. W celu walidacji danych używa się asercji, czyli metod sprawdzających różne właściwości podanych przez twórcę oprogramowania danych użytych w teście.

Lista przykładowych adnotacji używanych w celu implementacji testów jednostkowych przy użyciu JUnit – uzupełniane poprzez podanie znaku „@” na początku wyrażenia[[16]](#footnote-16):

* *Test* – Adnotacja używane w celu oznaczenia funkcji jako przykładowego testu.
* *ParametrizedTest* – Adnotacja używana w celu oznaczenia funkcji jako przykładowego testu, z możliwością dodania zmiennych jako parametry funkcji w konstruktorze.
* *BeforeAll* – Adnotacja używana do wywołania wybranego działania bezpośrednio przed wszystkimi testami.
* *BeforeEach* – Adnotacja używana do wywołania wybranego działania bezpośrednio przed każdym testem pojedynczo.

Wywoływane testy, najczęściej kończą się asercją, sprawdzającą czy wybrane przez twórcę zestawy danych, odpowiadają oczekiwanemu wynikowi. W celu takiego sprawdzenia, deweloper wywołuje metody z pakietu klasowego Assertions typu assertTrue, która zawiera w środku wyrażenie lub zmienną, wykorzystywaną wcześniej w pojedynczym teście. Analogicznie do szukanego przez programistę wyniku, jest możliwość użycia innych asercji, typu assertFalse, assertEquals, assertThrows czy assertTimeout.

* 1. Frontend

Warstwa frontend to część aplikacji webowej, która pozwala użytkownikowi na swobodną interakcję z oprogramowaniem, najczęściej przy pomocy przeglądarki internetowej. Jest to z reguły, pierwszy możliwy kontakt odbiorcy treści z zaimplementowanym systemem.

Część graficzna reprezentowana jest przez uprzednio utworzony wygląd oraz uzupełniona dostępem do najważniejszych funkcjonalności obsługiwanych przez stronę serwerową, oprogramowania wydaną do obsługi dla użytkowników końcowych aplikacji internetowej.

* + 1. TypeScript

TypeScript pojawił się w roku 2012 jako znacząco rozwinięty JavaScript. Jako główne rozszerzenie względem języka JavaScript można uznać wzbogacenie o dodatkową możliwość dopełnienia informacji o typie zmiennej. Pierwsza, stabilna wersja języka została wydana dopiero po dwóch latach, 12 kwietnia 2014 roku, przez twórców – firmę Microsoft. Swoją rosnącą popularność zawdzięcza w dużej mierze produktowi Angular, przy produkcji którego jego twórcy (przedsiębiorstwo Google) jako główny język w procesie tworzenia używali właśnie TypeScript. Użyteczność języka wykorzystywana jest również w konkurencyjnym produkcie dla Angulara – ReactJS, gdzie społeczność co raz częściej korzysta z TypeScript jako głównego języka podczas pisania aplikacji internetowych.

Język dobrze został przyjęty przez społeczność, między innymi ze względu na rewolucyjne zmiany rozwiązujące problemy występujące w JavaScript. W odróżnieniu od innych, stworzonych w przeszłości języków transkompilowanych do JavaScript ’u nie był wyłącznie wizualną nakładką na składnie kodu, ale także dodawał wysoce znaczące zmiany przydatne w codziennej pracy dla deweloperów oprogramowania.

Główne zmiany w stosunku do języka JavaScript[[17]](#footnote-17):

* Typowanie zmiennych, klas, interfejsów
* Interfejsy
* Typy wyliczeniowe
* Typy generyczne
* Kompilowane wywoływanie

Różnice na korzyść, którą przemawia TypeScript zostały szczególnie dostrzeżone w środowisku korporacyjnym, gdzie Angular wraz z językiem Microsoftu jest wykorzystywany w większości aplikacji, z zamiarem tworzenia części frontendu oprogramowania. Dzięki swojej wewnętrznej właściwości kompilowania się do języka JavaScript, możliwe jest pisanie oprogramowania zarówno po stronie frontendu – Angular czy React, jak również z perspektywy backendu w technologiach takich jak NodeJS.

Pomimo swoich niezwykle przydatnych funkcjonalności oraz zalet nad językiem JavaScript, należy pamiętać również o wadach powstających z statycznego typowania danych. Popularne biblioteki używane do pisania kodu na frontend aplikacji, często pisane są bazowo w JavaScript i to pod szyldem tego języka projektowane są rozwiązania. Biorąc pod uwagę te właściwości, implementacja niektórych funkcjonalności często potrafi być nieznacznie trudniejsza, niż bazowo miało to wyglądać.

Podsumowując, TypeScript jako zbiór nadrzędny języku JavaScript, pozwala osiągać te same funkcjonalności a nawet pozwalać na dodanie, jeszcze większej ilości przydanego dla programisty kodu. Szeroka gama oferowanych przez język, wzorców takich jak: imperatywność, obiektowość jak i funkcyjność oraz pozostałe pozwoliły twórcom oprogramowania użytkowego na bezpieczniejszą kontrolę typów, w przeciwieństwie do JavaScript, którego odwieczną wadą była błędogenność w implementowaniu wszelakiego rodzaju rozwiązań.

* + 1. React

Biblioteka frontendowa zaprojektowana i napisana w języku JavaScript, 29 maja 2013 roku przez Jordana Walke, w Facebook’u (aktualnie Meta), wraz z aktualnie pracującymi tam deweloperami. Miała ona ułatwiać oraz rozszerzać możliwości jakie dotychczasowo były osiągalne przy tworzeniu interfejsu użytkownika.

Z założenia pozwala na tworzenie aplikacji typu SPA – *Single Page Application*, czyli realizującej założenia aplikacji, ładując dynamicznie zawartość na stronie, jednocześnie bez potrzeby przeładowywania całego serwisu.

Na swoją popularność i renomę React mocno pracuję już od paru lat, początkowo goniąc framework Angular w uwadze użytkowników, aż po wieloletnie królowanie w większości cenionych przez ogólnoświatowych deweloperów oprogramowania rankingów. Różnice w cechach w odróżnieniu do innych frameworków dla frontendu, które zostały dostrzeżone przez użytkowników pozwalają, jeszcze na wiele lat bycia numerem jeden dla React.

Graphical user interface, application

Description automatically generated

Rysunek 7 Porównanie popularności największych technologii framework, kolory odpowiednio:  
 niebieski – React, czerwony – Angular, żółty – vue.js; źródło: https://trends.google.com

Na powyższej ilustracji łatwo zauważyć dominację React’a w ostatnich latach względem innych najpopularniejszych bibliotek w projektowaniu warstwy wizualnej użytkownika końcowego. Po samej liczbie wyszukań w najpopularniejszej wyszukiwarce – Google, łatwo zauważyć, praktycznie 50% większą popularność względem drugiej najpopularniejszej technologii – Angular.

Proces pisania interfejsu użytkownika w React opiera się na dzieleniu całości aplikacji na mniejsze, niezależne od siebie komponenty wielokrotnego użytku. Celem komponentów jest dostarczenie do kodu w aplikacji dodatkowej funkcjonalności w sposób izolowany, przez swoją logikę, bardzo często zwracający opakowany fragment kodu w HTML.

Komponenty można podzielić na dwa rodzaje:

* Komponenty funkcyjne – Reprezentowane przez prostą funkcję, w teorii rozwijające większą wydajność. Dzięki swojej prostszej budowie i posiadaniu mniejszej ilości zapisanego kodu, również bardziej czytelny, jak i zapewniający łatwiejszy proces debugowania i testowania. Pierwotnie wyróżniał go brak stanu, jednak po wprowadzeniu React Hooks, również ta funkcjonalność została wprowadzona. Do utworzenia prostego komponentu wystarczy podanie mu zmiennej z właściwościami oraz zwrócenie poprawnego fragmentu JSX.
* Komponenty klasowe – Rozwijające funkcjonalności komponentów poprzez dostarczenie zaimplementowanej klasy. Bardziej wymagająca do zaimplementowania niż komponent funkcyjny. React wymaga, aby klasa zawierała wyspecjalizowany konstruktor, dziedziczenie z klasy *Component* jak izwrócenie metody render z ciałem HTML. Główną korzyścią jaką można czerpać z komponentów klasowych są cykle życia aplikacji, takie jak *componentDidMount*, których implementacja w ciele funkcji, byłaby zdecydowanie dużo cięższa oraz wymagałaby większych starań.

Rozwiązania technologiczne na podstawie których budowy był React oraz język TypeScript, całkowicie pozwalają na wspólne działanie ze sobą. Powody, na korzyść których warto wybrać TypeScript ponad JavaScript, zaczynają się już od opisu właściwości obu języków. Głównym benefitem, TypeScript jest jego pełne statyczne typowanie zależności.

Najszybszą drogą na wdrożenie nowej aplikacji React/TypeScript jest utworzenie jej przy pomocy menadżera pakietów npm oraz odpowiedniego doklejeniu zależności w komendzie utworzenia nowego projektu React „--template typescript”. Podstawową różnicą jaką będzie można zaobserwować przy użyciu zależności o języku TypeScript będą, zupełnie inaczej wygenerowane pliku projektowe, między innymi[[18]](#footnote-18):

* .tsx – wszystkie pliki z rozszerzeniem .js i .jsx należące do JavaScript, zostaną odpowiednio zastąpione plikami TypeScript i TypeScript JSX.
* tsconfig.json – w miejscu pliku jsconfig.json, wyspecjalizowany plik konfiguracyjny zawierający szablon, z domyślnie dostarczonymi ustawieniami projektu.
* react-app-env.d.ts – pozwala w osadzaniu w aplikacji podstawowych zależności z szablonu aplikacji React oraz wprowadzaniu z niego plików.
  + 1. Material-UI

Material UI jest najpopularniejszą biblioteką służącą do tworzenia interfejsu użytkownika na platformie React. MUI reprezentuje porządną, gotową do modyfikacji oraz dostępną bibliotekę podstawowych i zaawansowanych komponentów, umożliwiającą budowę własnego systemu zaprojektowanego i szybciej stworzonego na platformie biblioteki React. Według niektórych źródeł ma być idealnym wyjściem pomiędzy deweloperami zajmującymi się tworzeniem oprogramowania w React a osobami znającymi Google Material Design w stopniu pozwalającym na tworzenie zaawansowanych interfejsów.

Integracja kodu napisanego w React oraz rozszerzenie go komponentami Material-UI charakteryzuje się wyjątkową prostotą. Większość komponentów zawiera podobieństwa do podstawowych znaczników HTML oraz ich dostępnych właściwościach. Sama obecność komponentów jest wysoce powiązana z kodem HTML, który można naprzemiennie stosować z dostępnymi w Material-UI częściami.

Komponenty dostępne w MUI można podzielić na następujące kategorie:

* Przyjmujące dane – Wszelkiego rodzaju guziki, pola na tekst, pola wyboru, pozwalające na zmianę treści w aplikacji.
* Wyświetlające dane – Ikony, listy, tabele pozwalające na wyświetlenie danych wyciągniętych z innego miejsca w aplikacji.
* Wyświetlające informacje zwrotne – Alerty, paski progresu informujące o aktualnym stanie aplikacji.
* Płaszczyzny – Puste powierzchnie, pozwalające na umieszczenie komponentów wewnątrz nich.
* Nawigacyjny – Paski nawigacyjne, menu, pozwalające na ustawienie wewnątrz nich ścieżek do innych adresów dostępnych w aplikacji.
  1. Baza danych

Baza danych to odpowiednio skonfigurowany system przechowujący dane, którego celem jest łatwiejszy dostęp do nich, zarządzanie nimi oraz możliwość aktualizacji na potrzebę działania systemu. Komputerowe bazy danych zazwyczaj przechowują agregacje rekordów danych lub pliki zawierających informacje, takie jak transakcje pomiędzy różnymi podmiotami, dane gości, dane o finansach przedsiębiorstwa i informacje o dostępnych w systemie produktach.

Te informacje następnie są gromadzone na odpowiednio długi okres, po którym często dane przechodzą do archiwalnych baz, żeby potem zostać usunięte. Logika baz często nie zakłada archiwizacji danych i przechowuje je przez cały okres działania systemu komputerowego połączonego z bazą informacji. Wszystkie informacje trzymane wewnątrz systemu, magazynowane są w celu późniejszej obserwacji i analizy.

Typy baz danych:

* Relacyjne – oparta na relacjach między tabelami
* Rozproszone – używana w dwóch lub większej ilości miejsc
* Chmurowe – działająca pod szyldem chmury operacyjnej
* NoSQL – odrzucająca mechanizm relacyjności
* Obiektowe – zachowanie zależne pomiędzy obiektami
* Grafowa – wykorzystująca struktury grafów

Wydaje się, że najpopularniejszym[[19]](#footnote-19) i najczęściej używanym rodzajem baz danych są modele o charakterze relacyjnym, dlatego właśnie ten typ zostanie bardziej przybliżony.

Model relacyjny to oferujący tabelowe podejście używane do przedstawiania szeregu danych. Twórcą podejścia jest Edgar Frank Codd, który w 1970 roku, zaproponował taki rodzaj modelu, zrewolucjonizował sposób w jaki trzymane są informacje po dziś. Dzięki tabelom można uzyskiwać dostęp do potrzebnych przez użytkownika danych na wiele różnych sposobów.

W tabelach dane są umieszczane w predefiniowanych kategoriach. Każda tabela zawiera kolumny z co najmniej jednym typem danych oraz przynajmniej jednym kluczem głównym w tabeli. Informacje w relacyjnej bazie danych o konkretnym elemencie zapakowane są odpowiednio w tabelach, które zawierają kolumny i wiersze. Głównym oraz tytułowym sposobem łączenia różnych tabel w modelu relacyjnym są relacje, zaimplementowane pomiędzy innymi tabelami. Kolumny są odpowiednio indeksowane, w celu prostszego wyszukiwania ich za pomocą specjalnie stworzonego języka zapytań SQL.

SQL, jak wynika bezpośrednio z nazwy to strukturalny język zapytań. Używany bezpośrednio do operowania na bazach danych. Został opracowany w tym samym momencie co model relacyjny baz informacji, w celu uniwersalizacji w jaki sposób operowane i administrowane będą bazy relacyjne. W składni SQL można wymienić takie podzbiory składni języka służące do operacji na danych jak DQL, DML i DDL.

DML – używany do manipulacji na danych dostępnych w bazie danych:

* UPDATE – służy do modyfikacji danych.
* INSERT – służy do osadzenia danych w bazie.
* DELETE – służy do kasowania danych z bazy.

DDL – używany do manipulacji struktur używanych w bazie danych (tabela, baza itp.):

* CREATE – służy do utworzenia jednostki.
* DROP – służy do usunięcia jednostki.
* ALTER – służy do zmiana istniejącej jednostki.

Dodatkowo DQL, najczęściej używany do wyszukiwania danych poprzez odpowiednie filtrowanie ich poprzez dopisywanie kolejnych, modularnych części. Podstawową instrukcją, zaczynającą każde wyrażenie jest SELECT, następnie dopisywane poprzez nazwę wiersza lub gwiazdkę. Do dalszego filtrowania rekordów używa się między innymi takich nazw języka składni jak:

* FROM – z dopisaniem nazwy oczekiwanej przez użytkownika tabeli.
* WHERE – z dopisanym wyrażeniem filtrującym i ograniczającym wynik.
* ORDER BY – Sortującym wynik po oczekiwanej zmiennej z tabeli.
  + 1. PostgreSQL

Rozpoznawalny, również, w nieco skróconej formie jako Postgres. Według wielu rankingów jeden z topowych, pod względem popularności, otwartych systemów służących do zarządzania relacyjną bazą danych. Rozwijana, przez bardzo długi czas, jednak oficjalnie wydana, pod pierwszą stabilną wersją dopiero w lipcu 1996 roku przez absolwentów Uniwersytetu Berkley. Od samego swojego początku baza PostgreSQL oferowała, darmowy dostęp do swoich zasobów na podstawie licencji wolnego oprogramowania. Przez samych twórców, opisywana jako najbardziej zaawansowana relacyjna baza danych, na licencji wolnego oprogramowania na świecie[[20]](#footnote-20). Dzięki swojej dostępności oraz zakresu zaawansowanych rozwiązań zaimplementowanych w swoim systemie, wybierana jest nie tylko przez, pojedynczych użytkowników, ale przez całe korporacyjne środowiska, które od parudziesięciu lat opierają swoje rozwiązania teleinformatyczne na bazach danych PostgreSQL.

Dużą zmianą względem zwykłego SQL i operacji dostępnych przy pomocy samego języka jest wprowadzenie wewnętrznego języka proceduralnego dostępnego bezpośrednio w systemie PostgreSQL. PL/pgSQL, czyli wewnętrzny język proceduralny bazy relacyjnej Postgres, w pewnym stopniu przypominający konkurencyjne rozwiązanie bazy Oracle – PL/SQL, jednak dostępnego jedynie na licencji komercyjnej. Za swój początek uznaje się październik 1998 roku, kiedy to Jan Wieck zaimplementował wewnętrzny język, a następnie, wraz z wersją 6.4 udostępniony dla społeczności PostgreSQL. Niektóre z funkcjonalności dodanych do systemu bazy danych w języku proceduralnym:

* Struktury kontroli – bardzo ważna zmiana wprowadzająca podstawowe funkcjonalności operacji na danych znajdujących się w bazie PostgreSQL. Dodaje, znane z popularnych języków programowania pętle, wyrażenia warunkowe, funkcje i inne.
* Kursory – zwracający, zgodnie z ustaleniem przez użytkownika, przefiltrowane dane występujące w tabeli, a następnie pozwalający na korzystanie z posegregowanych, przez nie informacji. Pozwala na operowanie na każdym pojedynczym zwróconym materiale, aż nie zostaną użyte wszystkie, czyli działanie można porównać do pętli *for-each*, znanej z popularnych języków programowania dostępnych na rynku.
* Wyzwalacze – stworzone w celu dodawania przydatnych modyfikacji w przypadku wywoływania operacjach usuwania, modyfikowania oraz tworzenia danych na wyznaczonej przez wyzwalacz tabeli.
* Procedury składowane – pozwala na tworzenie i modyfikowanie nowatorskich funkcji w celu dalszego używania i wywoływania ich przy pomocy innych tworzonych przez użytkownika funkcjonalności w tworzonej aplikacji w systemie Postgres.
  + 1. H2

System zarządzania relacyjną bazą danych, napisany przy pomocy języku Java. Dostępny przy pomocy licencji otwartego oprogramowania, od grudnia 2005. System H2 pozwala na stworzenie osadzonej plikami bazy danych, bezpośrednio w plikach systemowych, bez konieczności trudzenia się z połączeniem z zewnętrznym systemem wyposażonym w potrzebne dla użytkowników informacje. Przez swoje właściwości, a mianowicie proces w jakim informacje usuwane są z bazy danych bezpośrednio po zerwaniu połączenia, nie jest praktycznie wcale używana jako baza produkcyjna. System H2 często spotykany jest jednak jako miejsce, w którym trzyma się informacje podczas procesu testowania wcześniej napisanego oprogramowania informatycznego. Pomimo właściwości, które często uznawane są za negatywne jest nieustannie używana, z pewnością wpływ na to może mieć niezwykle wysoka wydajność, oraz łatwość konfiguracji, co perfekcyjnie wpasowuje się w projekty, w których ilość funkcji dostępnych w bazie danych oraz jej zaawansowanie nie gra pierwszej roli. Dobrym adresatem są również omówione wcześniej fragmenty testów jednostkowych, w których ważna jest szybkość połączenia z miejscem wymiany informacji.

Do połączenia się z treścią wewnętrzną bazy danych, dostępne są dwie możliwości,  
a mianowicie użycie przeglądarki internetowej oraz udanie się na identyfikujący   
bazę danych adres IP. Drugim sposobem dostania się do treści jest użycie konsoli dostępnej jako plik jar. Dzięki zamysłowi w kierunku jakiego, tworzona była baza H2, wydaję się, że jest to najłatwiejsze miejsce do dostania się przy pomocy JDBC, czyli interfejsu do wymiany informacji pomiędzy programem napisanym w Javie a bazą danych.

Przez niektórych doświadczonych deweloperów oprogramowania, baza H2 może być uznawana jako niewarta używania, nawet podczas procesu testowania[[21]](#footnote-21). Jako   
alternatywę podają odpowiednio ustawione kontenery Docker’owe oraz poprawnie postawioną na nich bazy danych „z prawdziwego zdarzenia”. Jako argumenty przeciwko korzystaniu z plikowych baz danych podają to, że użytkownik nie ma całkowitego kontaktu z prawdziwym produktem, a jedynie jego imitacją, przez co nie jest możliwe całkowite oddanie realnego zachowania. Pomimo przedstawianych faktów wydaję się, że wygoda jaką oferuję baza ustawiona w plikach systemowych, oraz brak przymusu w dodatkowej konfiguracji, jest wystarczający, aby w dalszym ciągu korzystać z baz typu H2, nawet w przypadku, gdy na produkcyjnych środowiskach nie są one wykorzystywane.

* 1. Narzędzia programistyczne

W codziennej pracy dewelopera oprogramowania bardzo ważny jest dobór narzędzi, które umożliwiają wygodniejsze i pozwalające na oszczędzenie dużej ilości czasu w wykonywanym zawodzie. W wyborze akcesoriów potrzebnych do pracy, podobnie jak przy technologiach ważne były wcześniejsze opinie programistów o większym doświadczeniu oraz popularność, jak i udział w rynku wybranego narzędzia.

* + 1. IntelliJ IDEA

IntelliJ, wydany w styczniu 2001 roku przez firmę JetBrains jako zintegrowane środowisko programistyczne napisane przy użyciu języka Java. Jest w dalszym ciągu na bieżąco rozwijane jako narzędzie do tworzenia wszelakiego oprogramowania komputerowego, ukierunkowanego na platformę Java i języki na jej platformie wirtualnej maszyny. Wyróżnia się dwa wydania, jedno darmowe - społecznościowe oraz drugie płatne – komercyjne, jednak z obu wersji można korzystać jako główne środowisko programistyczne wykorzystywane do programowania komercyjnego.

Mówiąc o IntelliJ, warto także wspomnieć o Android Studio, czyli głównym środowiskiem uruchomieniowym do tworzenia aplikacji na platformie Android, które stworzone zostało na podstawie IntelliJ i w dalszym ciągu jest na niej rozwijane.

Niektóre z funkcji na rzecz których IntelliJ zostaje wybierany przez większość programistów tworzących w językach wirtualnej maszyny Javy to asystent kodu podpowiadający najlepsze implementacje, wsparcie większości najpopularniejszych technologii dostępnych na rynku oraz moduł *plugin’ów* pozwalających przykładowo na szybkie wdrożenie do środowiska obsługi nowych języków albo ciekawych dodatków umilających prace.

* + 1. **VisualStudio Code**

Visual Studio Code jest edytorem kodu źródłowego stworzony w kwietniu 2015 roku przez Microsoft wspierającym najpopularniejsze systemy operacyjne. Pomimo bycia zwykłym edytorem tekstu pozwala na operowanie debugowania kodu, wykrycie błędów oraz podświetlenie ich składni, uzupełnianie kodu oraz jego refaktoryzację oraz interfejs połączenia z Git. Bardzo popularne wydają się być instalowane rozszerzenia, wspierane i wydawane przez społeczność, które pozwalają na dodanie ciekawych funkcjonalności.

Przez swoją lekkość i modularność pozwala na dodawanie, praktycznie większości istniejących języków programowania. Potrafi być także dostępny jako natywny edytor tekstu bezpośrednio na niektórych popularnych hostingach pozwalających na trzymanie kodu źródłowego, takich jaki GitHub czy GitLab.

* + 1. **GitHub**

GitHub jest platformą internetową, powstałą w celu umożliwienia magazynowania kodu źródłowego na zewnętrznym repozytorium w Internecie, współbieżnie pozwala na działanie z systemem kontroli git. Utworzona w lutym 2008 roku przez strona gromadzi na swoich serwerach rekordową dla hostingów kodu źródłowego, liczbę około 200 milionów różnych repozytoriów od ponad 73 milionów kont deweloperów. Od 2018 roku rozwijana pod szyldem firmy Microsoft. Dzięki swojej łatwości w obsłudze oraz możliwości trzymania repozytoriów w większości bez opłat, pozwolił doprowadzić do statusu najpopularniejszego serwisu służącego do trzymania kodów źródłowych.

Pozostałe rozwiązania technologiczne

W większości rozwiązań technologicznych istnieją podobieństwa, na które można podzielić je na różne kategorie. W wypadku niektórych systemów, nie ma takiej możliwości, pomimo tego nie należy bagatelizować poziomu w jakim stopniu wpływają one, na wygląd końcowy projektu.

* + 1. Git

Kontrola wersji, czasem również znana pod nazwą kontrola źródła, to praktyka zarządzania, poprzez śledzenie pojedynczych w kodzie źródłowym oprogramowania. Systemy wersji kontroli to narzędzia zawarte w systemie operacyjnym, które pomagają twórcom oprogramowania, chociaż najczęściej przez swoje właściwości całym zespołom programistycznym zarządzać zmianami w kodzie źródłowym z podziałem na czas dodania zmian.

Ze względów tego, że zmiany w środowiskach programistycznych zyskują na szybkości, systemy kontroli wersji pomagają coraz to większym zespołom programistycznym pracować szybciej i mądrzej. Systemy kontroli wersji pozwalają na proste śledzenie zmian, względem starszych wersji projektu, daje to możliwość szybkiego cofnięcia się do poprzednich, w pełni działających wariantów projektu.

Git jest najpopularniejszym systemem kontroli wersji, stworzonym w kwietniu 2005 roku przez głównego twórcę jądra Linux’a, Linusa Torvalds jako system zarządzania wersjami używany do koordynowania właśnie tego systemu. Najlepiej zoptymalizowany pod Linux, jednak dostępny dla wszystkich najpopularniejszych systemów operacyjnych na podstawie darmowego oprogramowania na licencji otwartego źródła.

Chart, diagram

Description automatically generated

Rysunek 8 Schemat rozwoju drzewa przy pomocy gałęzi git; źródło: www.danielebachicchi.com/metodologia/organizzare-repository-git

Proces rozwoju oprogramowania najczęściej dzieli się na rozwijanie oddzielnych gałęzi, podzielonych na przeróżne etapy w celu zachowywania punktów kontrolnych. Na powyższym wykresie można wyróżnić niektóre konwencje nazewnicze co do wersjonowania zmian.

Gałąź *master* lub czasem *main* zawiera wersje z oprogramowaniem produkcyjnym, posiadającym pełne wersje wypuszczanych produktów. Gałąź *develop* służy do przechowywania wszystkich zmian tworzonych przez twórców oprogramowania, aż do wypuszczenia ich na środowisko produkcyjne lub przedprodukcyjne, czyli gałąź *release*. Wszystkie małe zmiany tworzone są bezpośrednio na nowo utworzonych gałęziach *feature*, łączone następnie z gałęziami develop poprzez „*merge requesty*” i zatwierdzanie zmian przy pomocy recenzji kodu źródłowego przez innych programistów z zespołu.

Podstawowe komendy Git, służące do obsługi systemu kontroli wersji[[22]](#footnote-22):

* Git init – Utworzenie nowego repozytorium.
* Git add – Dodanie zmian do statusu *staging.*
* Git commit – Utworzenie z plików znajdujących się w statusie *staging*, nowego punktu kontrolnego.
* Git branch – Wylistowanie wszystkich dostępnych gałęzi, z dodatkową nazwą pozwala na stworzenie nowej gałęzi z wersji, na której aktualnie znajduje się użytkownik.
* Git checkout – Zmiana gałęzi, na której aktualnie znajduję się użytkownik.
* Git push – Wypchnięcie zmian do zdalnego repozytorium.
  + 1. **Docker**

Docker to powstała w marcu 2013 roku technologia wirtualizacji kontenerów. Można ująć, że jest to wyjątkowo lekka wirtualna maszyna[[23]](#footnote-23) pozwalająca na trzymanie w swoich zasobach, mniejsze kontenery, utrzymujące w specyficznym stanie zewnętrzną aplikację wizualizującą, swoje istnienie na maszynie użytkownika.

Głównym problemem jaki rozwiązuje konteneryzacja przy pomocy systemu Docker jest częste ustawianie tych samych aplikacji, z tym stanem początkowym. Dzięki istnieniu narzędzia jakim jest Docker, można zautomatyzować ten proces, w którym coraz częściej pomagają sami twórcy rozwiązań technologicznych.

Kontenery można uznać za abstrakcyjne byty w warstwie aplikacji połączonej z Docker, w której dzieją się procesy stawiania aplikacji razem poprzez opakowywanie kodu i zależności pod jednym szyldem. Logika kontenerów jest taka, że wiele bytów może działać na tej samej maszynie i współdzielić jądro systemu operacyjnego z innymi bytami. Jednak każdy kontener musi działać w sposób izolowanych procesów o innych portach docelowych.

* + 1. **Gradle**

Gradle jest narzędziem wspomagającym proces automatyzacji budowania aplikacji, stworzonym i rozwijanym głównie dla języków wirtualnej maszyny Javy. Powstały w kwietniu 2008 na licencji otwartego oprogramowania Apache License 2.0 technologia powiela wcześniej popularne zamysły używane w takich systemach jak Maven.

Narzędzie pozwala na dołączenie do plików aplikacji gotowych zewnętrznych bibliotek, poprzez dodanie ich w plikach konfiguracyjnych. Budowanie aplikacji odbywa się poprzez rozczytanie plików konfiguracyjnych napisanych w rozszerzeniu gradle, przypominającym plik typu Json, napisany przy pomocy takich języków jak Groovy lub Kotlin.

* + 1. **GitHub Actions**

GitHub Actions jest wewnętrznym, bezpośrednio w GitHub’ie systemem integracji[[24]](#footnote-24) i dostarczania, pozwalającym na stawianie prostych procesów testowania aplikacji. Przy pomocy prostych plików w formacie yml pozwala na utworzenie ciągu zdarzeń, mających miejsce po konkretnie wywołanych przez wdrożenie aplikacji procesach. Dzięki GitHub Actions twórca oprogramowania może otrzymać opakowaną wizualnie informację na temat występujących problemach, podczas wdrażania aplikacji na różnych etapach zarządzania projektem.

Niezwykle ważna w rozumowaniu narzędzia jakim jest GitHub Actions jest ciągła integracja. Jest to jedna z najczęstszych praktyk jakie występują w projektach komercyjnych. Polega na ciągłym dodawaniu zmian każdego programisty dostępnego w zespole do kodu znajdującego się na głównej gałęzi zmian w projekcie. Dzięki serwerowi lub zwykłemu narzędziu, który pozwala na włączenie wirtualnej wersji systemu w dedykowanym przez siebie miejscu w głównym systemie. Za każdym razem, gdy narzędzie odkryje, że jakieś zmiany zostały wprowadzone do dedykowanej gałęzi automatycznie włączy dedykowane w pliku konfiguracyjnym procesy. Najczęściej w takich procesach znajduje się informacja o zbudowaniu aplikacji i późniejszym przeprowadzeniu testów.

Pomimo popularności takiego rodzaju sprawdzania tworzonego oprogramowania, nie ogranicza się ona wyłącznie do takich działań. Bardzo popularne jest również ustawianie prac w pliku konfiguracyjnym, typu włączanie benchmarków oprogramowania i ustawianiu tego na dedykowaną godzinę przy pomocy polecenia *cron* znanego z konsoli jądra systemów unix.

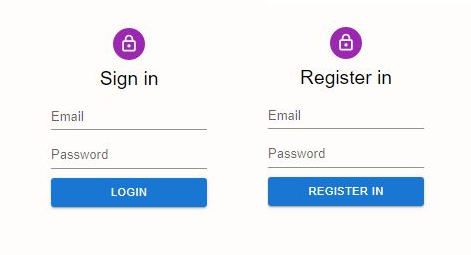
1. Implementacja

Następujący rozdział opiera się w dużej mierze na implementacji najważniejszych funkcjonalności w systemie oraz przedstawieniu niektórych z narzędzi używanych podczas zarządzania projektami według trendów istniejących na rynku oraz dobrych praktyk programistycznych.

* 1. Moduł rejestracji i logowania

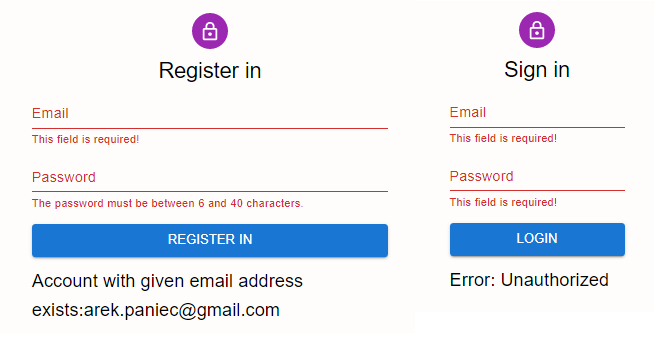
Niezwykle ważnym modułem dostępnym w projekcie, pomimo małego pokrycia z tematem biznesowym jest moduł rejestracji oraz logowania użytkowników aplikacji. To na jego oknach użytkownicy końcowi aplikacji zaczynają drogę w procesie korzystania oraz komunikowania się z przedstawianym oprogramowaniem. W przypadku niezalogowanych klientów to właśnie na poniższym oknie zaczyna się cały proces. Jak widoczne jest to bezpośrednio podczas korzystania z oprogramowania, w oknie przeglądarki rozwijane są widok logowania się z możliwością przejścia do przypadku rejestracji nowego użytkownika.

Na poniższym zdjęciu nr 9 reprezentowany jest przykładowy fragment implementacji tego rozwiązania przy pomocy abstrakcji wizualnej. Widoki implementowane są w sposób prosty ale treściwy, bez zbędnych dynamicznych dodatków.



Rysunek 9 Moduł logowania i rejestracji użytkownika dostępny w aplikacji; opracowanie własne

Po odpowiednim użyciu przedstawionych uprzednio modułów użytkownik może przejść przez procesy pomyślnie lub zakończyć interakcje z modułem z odpowiedzią błędną zwróconą bezpośrednio z serwera i następnie wyświetloną w oknie przeglądarki. Opisywana reprezentacja wizualizacji uznawanej jako błędna odpowiedź przedstawiona jest na poniższym zdjęciu nr 10.



Rysunek 10 Błędy walidacyjne dostępne w aplikacji w kierunku logowania użytkowników; opracowanie własne

Obie wizualizacje nie byłyby możliwe bez odpowiedniego wykorzystania metod dedykowanych po stronie warstwy wizualnej oraz serwera. Na poniższych fragmentach kodów źródłowych zaprezentowane jest wykorzystanie dostępnych bibliotek jak i podstawowych składni w używanych technologiach w celu implementacji działań logowania oraz rejestracji nowych użytkowników oraz dalszego ich wykorzystywania.

class UserDto(

val firstName: String?,

val lastName: String?,

@NotBlank

@Email

val email: String,

@NotBlank

@Size(min = 4, max = 20)

var password: String,

var role: String = Role.USER.name

)

Fragment kodu źródłowego 1 Klasa transferu obiektu nowego użytkownika; opracowanie własne

Powyższa podstawowa klasa Kotlin’owa z automatycznie inicjalizującymi się metodami generycznymi odpowiada za przekazanie obiektu i zawartych w nim potrzebnych informacji do stworzenia konta dla nowego użytkownika w opisywanym oprogramowaniu. Klasa odpowiada także za fragment uchodzący za walidacje danych wykorzystanych w warstwie serwerowej. Do jej implementacji użyto podstawowej biblioteki walidacyjnej[[25]](#footnote-25) wykorzystanej w Spring Boot, w celu aktywacji działania metod walidacyjnych zastosowano oznaczenia typu @NotBlank, @Email oraz @Size.

validateRegistration() {

return Yup.object().shape({

email: Yup.string()

.email("Email not valid")

.required("Field required"),

password: Yup.string()

.test(

"len",

"Password should be between 4 and 20 characters",

(passwordInput: any) =>

passwordInput &&

passwordInput.toString().length >= 4 &&

passwordInput.toString().length <= 20

)

.required("Field required"),

});

}

Fragment kodu źródłowego 2 Metoda służąca do walidacji danych przy rejestracji nowego użytkownika; opracowanie własne przy pomocy https://github.com/jquense/yup

Podobną ścieżką należy podążać przy projektowaniu walidacji po stronie wizualnej do której dostęp mają wszyscy użytkownicy. Błędy po stronie serwerowej będą wyłapywane dopiero po przejściu przez niektóre z dalszych etapów już na wewnętrznym oprogramowaniu. Warto zadbać, aby błędy niepotrzebujące przejścia do serwera były walidowane już po stronie klienta. Jako takie rozwiązanie pomocna może okazać się biblioteka Yup użyta w powyższym rozwiązaniu metody walidacyjnej po stronie klienta.

fun registerUser(newUser: UserDto): User {

if (userRepository.findUserByEmail(newUser.email) != null) {

throw EmailExistsException("Account with given email exists:" + newUser.email)

}

val encodedPassword = passwordEncoder.encode(newUser.password)

val user = User(

firstName = newUser.firstName,

lastName = newUser.lastName,

email = newUser.email,

password = encodedPassword,

role = Role.roleOf(newUser.role)

)

return userRepository.save(user)

}

Fragment kodu źródłowego 3 Metoda służąca do rejestracji użytkowników; opracowanie własne

W bazowej metodzie służącej do rejestracji nowych użytkowników użyto mechanizmów walidujących, czy aby na pewno do stworzenia nowego konta nie wykorzystano wcześniej występującego w bazie danych adresu email. Innym wyróżniającym mechanizmem wykorzystującym jest metoda pozwalająca na zakodowanie podanego podczas rejestracji hasła. Kodowanie użyte podczas szyfrowania hasła jest prosty mechanizm bcrypt. Jako ciało używane do zwrócenia po całej operacji zastosowano obiekt powstały po całym transferze danych.

fun logUser(authRequest: AuthRequest): JwtResponse {

val authentication: Authentication = authenticationManager.authenticate(

UsernamePasswordAuthenticationToken(authRequest.email, authRequest.password))

SecurityContextHolder.getContext().authentication = authentication

val jwt: String = jwtUtils.generateJwtToken(authentication)

val userDetails = authentication.principal as UserDetailsImpl

val role = userDetails.authorities.stream()

.map { item: GrantedAuthority -> item.authority }

.collect(Collectors.toList())

return JwtResponse(

accessToken = jwt,

id = userDetails.id,

email = userDetails.email,

role = role[0]

)

}

Fragment kodu źródłowego 4 Metoda służąca do logowania, zwracająca token; opracowanie własne

Metoda pozwalająca na logowanie użytkowników do aplikacji dostępna na powyższym fragmencie kodu źródłowego oparta jest na popularnych na rynku JSON Web Token’ach uprzednio wspomnianych w rozdziale nr 3. Powyższe operacje nawiązują do procesów potrzebnych podczas walidacji danych użytkownika i utworzeniu ich Tokenu w szablonie biblioteki jjwt[[26]](#footnote-26) użytej podczas implementacji kodu źródłowego.

* 1. Tabela operacji

Tabela operacji jest bezpośrednim miejscem w opisywanym oprogramowaniu, w którym wyświetlane są wszystkie operacje finansowe. Dzięki nowoczesnym rozwiązaniom jest możliwość szybkiego segregowania dostępnych wyników bezpośrednio po stronie klienta. Taka opcja pozwala na ograniczenie części serwerowej w wymianie segregowania już uprzednio pobranej listy operacji finansowych.

![Obraz zawierający stół

Opis wygenerowany automatycznie](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RDoRXhpZgAATU0AKgAAAAgABAE7AAIAAAAKAAAISodpAAQAAAABAAAIVJydAAEAAAAUAAAQzOocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEFya2FkaXVzegAABZADAAIAAAAUAAAQopAEAAIAAAAUAAAQtpKRAAIAAAADNzMAAJKSAAIAAAADNzMAAOocAAcAAAgMAAAIlgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADIwMjE6MTI6MjkgMjA6MzE6MDUAMjAyMToxMjoyOSAyMDozMTowNQAAAEEAcgBrAGEAZABpAHUAcwB6AAAA/+ELHGh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iLz48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMjEtMTItMjlUMjA6MzE6MDUuNzMyPC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyI+PGRjOmNyZWF0b3I+PHJkZjpTZXEgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpPkFya2FkaXVzejwvcmRmOmxpPjwvcmRmOlNlcT4NCgkJCTwvZGM6Y3JlYXRvcj48L3JkZjpEZXNjcmlwdGlvbj48L3JkZjpSREY+PC94OnhtcG1ldGE+DQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgIDw/eHBhY2tldCBlbmQ9J3cnPz7/2wBDAAcFBQYFBAcGBQYIBwcIChELCgkJChUPEAwRGBUaGRgVGBcbHichGx0lHRcYIi4iJSgpKywrGiAvMy8qMicqKyr/2wBDAQcICAoJChQLCxQqHBgcKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKir/wAARCAFlA5UDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwD6PooopDCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiqNvbQXKyS3EKSuZXGZFDYAYgAZ6cCgB2qXUtlp0lzCEJiKs+8HGzcN5477c496zJteuUnvEjSEjISyJBO9w4jbdg9A7DpjjNbckEctq1uygRMhQqOPlIxiqUei28aWKh5SbMkqxIzIT1LcdzhuMcgfSjqD2KFz4pjW5ubKKBo7u3KErK0bBlMirnCOWXhgQGCnnp1qCTxtY2EcY1RlWV3l3BHRAiLKyBsO4Lfd6LuPB46Vai8J2kR4ubpkG7YhKBUDSLIQMKP4l6nJ5OSeMSHw1Arhre9vLYkOsphkVTKrOX2k7cjBZsFcMMnmhdLh1ZNpOpy6jdanHLD5aWl15MZ4+dfLRs8E/3s9uCO+a06q2mnx2VxeTRO5N3KJnViCA2xV44z0Udc0v9n2X/PnB/wB+l/woAs0VW/s+y/584P8Av0v+FH9n2X/PnB/36X/CgCzRVb+z7L/nzg/79L/hR/Z9l/z5wf8Afpf8KALNFVv7Psv+fOD/AL9L/hR/Z9l/z5wf9+l/woAs0VW/s+y/584P+/S/4Uf2fZf8+cH/AH6X/CgCzRVb+z7L/nzg/wC/S/4Uf2fZf8+cH/fpf8KALNFVv7Psv+fOD/v0v+FH9n2X/PnB/wB+l/woAs0VW/s+y/584P8Av0v+FH9n2X/PnB/36X/CgCzRVb+z7L/nzg/79L/hR/Z9l/z5wf8Afpf8KALNFVv7Psv+fOD/AL9L/hR/Z9l/z5wf9+l/woAs0VW/s+y/584P+/S/4Uf2fZf8+cH/AH6X/CgCzRVb+z7L/nzg/wC/S/4Uf2fZf8+cH/fpf8KALNFVv7Psv+fOD/v0v+FH9n2X/PnB/wB+l/woAs0VW/s+y/584P8Av0v+FH9n2X/PnB/36X/CgCzRVb+z7L/nzg/79L/hR/Z9l/z5wf8Afpf8KALNFVv7Psv+fOD/AL9L/hR/Z9l/z5wf9+l/woAs0VW/s+y/584P+/S/4Uf2fZf8+cH/AH6X/CgCzRVb+z7L/nzg/wC/S/4Uf2fZf8+cH/fpf8KALNFVv7Psv+fOD/v0v+FH9n2X/PnB/wB+l/woAs0VW/s+y/584P8Av0v+FH9n2X/PnB/36X/CgCzRVb+z7L/nzg/79L/hR/Z9l/z5wf8Afpf8KALNFVv7Psv+fOD/AL9L/hR/Z9l/z5wf9+l/woAs0VVt0WG9mhiG2MIjhB0BJYHHp0FWqACiiigAooooAKKKKACiiigAooooAKK5PxtrF/b3GiaFotyLO91y7aAXe0M1vEkbSSOqngthcDOQCc84rJ8S6NqHgrw9deI9B17WLqbTYzc3NpqN41zFeRLy67Wz5bbQSCm0A9sUvNjs72R6FRXFT+OtSuPEtxovh7w4dSlisbe+E0t4II9ku/gkqcN8owADnJztxkub4jWr+H7G8tLCaTUb66kso9OlkSNo5493mK7k7VVdhywznjAOQKbVhXv/AF8/yOzorhZPiQ1lpGvPqmlRwapo2nnUGtIb1ZoriLDYKShQeqkHKAjjg5rQTxRrh0Rb5/CsxnupY0srWK6DsyuM75m27YgB15b2yaAOqorio/GOtS/21pd5oEdhrljYi8gjW/EsE0bFlDCXYCCCpyClZHhbxBql9Z/D19dFxHdahbSMWgv90dwBbKwkmTyxuJznbkbT3aje/wAvxv8A5A3bf+tv8z0yiuDl+I15/Z1xrtn4bmufDVq7iTUBdKszIjFXlSDHzICDyWBIGQDVq78b6jL4nu9E8OaAuqS29nBeLcPfCCJklL8E7CQfkGMA5yc7ccnQDsqK4j/hY63Hh/R7vTtLaTUNXuZLSKyublYVimj3iRXkwRwUYDAJPGBXQeH9WvtUt7gato82lXdtN5Txu/mRycAh45MDepz1wCCCCOKLbga9FFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFVrH/AI92/wCu0v8A6MarNVrH/j3b/rtL/wCjGoAs0Vl+INSn0nTUvIER0juIhPvH3YmcK7DkYwDnPt0qmniCf/hMpdNkSJbAL5KS4Ic3CoJGXOcFfLYEe6tSA6CiuWsfFl7LoVnetol7qEl3G9yq6fGirHAWPlktLIqlim3Kgk5ycAVW1XxvJHYNfaQkUls+gzapAZ42DFl27QRkcYbkdfen1t/XX/Jjtrb+ui/U7KiufuvFsNoZCbC8ube1VTe3luqeVaEqGO7c4Y4UhjsVsA80W/iK7l8Q6zZHSrh7fT7eOaGSIxkzlgx2r+8zk44yF6HJ6UE3Wh0FFYXhfWr7Wo9SbUrJbGS1vWgWAOHZVCIw3MCQW+bnHA6AnGTu0DvcKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKZLNHBC0s8ixxoMs7sAFHuTQA+iqC67pDuETVbJmY4AFwhJP51foArJ/yE5/8ArjH/AOhPVmqyf8hOf/rjH/6E9WaACiiigAoJwMngVBe3cdhp9xeXGfKt4mlfA52qMn9BXgF3q+qeP/FcNrc3bQxXUwjhiyTHCCeOB1Pv1Nd2EwcsTeV7RW7OLFYuOHsrXb2R9DUV8/8AhHxdfeFvEK28tzJNYGbyp4mORjONyg9CP/rV9AUsZg5YWSTd09mVhcVHExbSs1uFFFFcR1hRRRQBzXjHw/e6sum6jokkKavo919ptRcEiOXKlHjYgEgMrEZAODg4NY+tx+K/GujzeH5/D/8AYNneDyr+8uL2KY+SfvrCsZJJYZGX24znB6V3tFAzzIXuraN8Xddj0HQ/7Vt10iwR4Y7lIZIyGmCEb8KV655yOMA1WvfhvftommX13pmm61qUGq3Op3ulXG1oJxcbt8as6kbkBTDEAEr2zXpMOkWMGtXWrRQbb67ijhml3sd6IWKjGcDG9ug71doevr/wbi8un/At/meW3PhC8vfCHiqLSfA2j+HJL7S5LSztreOBbqeRlOfMkjPlqudoC5PIJJHSt/xjpevXPh3SIdHjuJkt5ozqNlaX32Se4iCEFElBGMNtJ+ZcgYzXZ0Uf8D8P+HBafj+P/DHmHhrwdqVp431PU/7CXSLG+0Y2sYkvvtMvm+YTmVixO4g9iwAHXPFS+HfD+vqngJNR0h7P+wIZrW7LXETggW4jWRdrHKsRwOo7gV6VRR0t/W7f6sVtb/1sl+iPMF0TxdYeB7rwJZ6NDPBJFNZ2+stdosKW8hYBnj/1m9VbGACCRncM1v8Ah3w3d6N471K58kjTm0mxs7eYup3tF5oYYzkYDL1Heuwopptf1/Xcb1/r0f6I8xtvDGr2vgaPTdR8LWOuQNql5PdabdPGZGjkmkeN4mJ2BvmBwxBweoNbfw80DUNDi1M3FrNpmn3EyNY6VNem6a0ULhvnywG5udqsQMdea7OiktAeoUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVWsf+Pdv+u0v/oxqs1Wsf8Aj3b/AK7S/wDoxqAE1Swj1TSbuwm/1dzC8TewYEf1rmbnwtq9x4JFn9st110zNctdEExrI5YNg4zgRuyg47DIrsKKVh3Zx2veErq+vLWO3tbC/wBMitFt0s9RnkENu6kgS+SqlJzgjhtuNgwwySKQ8Dap/wAIzBpxmsxNF4dl0okO20yttww+XO35fr7Gu+op3t/Xr/mwWm3l+Fv8jgrvwEf7YvbhNB8N6ot8ySG61SLfLavsVGAXyz5i/KGA3pySMjrXTWmnXdr4qvbsLbmxuraJQwkIkR49wxs24KkNnO7IxjHeteinfSxNkZWi6XNp1zq0k7RsL2+a4j2EnCmNFwcjrlT61q0UUuiQwooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigApGUMpVgCCMEEdaWigDnNK8DaPpGuzapawnzHOYo25WD12/X9O1dHRRQBWT/AJCc/wD1xj/9CerNVk/5Cc//AFxj/wDQnqzQAUUUUAUtYsf7T0O+sQdpubd4gfQspGf1rwrwx4i1Lwdr6WF5M9pZx3WbyEwqx9D1GegHSvoKsfWfCuieIGD6tYRzyKMCQEqwHpuUg16ODxcKMZU6sbxkcGKw06so1KbtJHiDXWsePfElpp8s5uV81hF8ioI4yQWJwPQDrnpX0MqhFCqMADAHpWfpGgaVoMTR6RZRWwc/MVGWb6seTWjSxuLjiHGNONox2Hg8NKgm5u8nuFFFFeedwUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFVrH/j3f8A67S/+htVmoHs4XkL/OrNyfLkZM/kRQBPRVb7DF/fn/8AAiT/AOKo+wxf35//AAIk/wDiqALNFVvsMX9+f/wIk/8AiqPsMX9+f/wIk/8AiqALNFVvsMX9+f8A8CJP/iqPsMX9+f8A8CJP/iqALNFVvsMX9+f/AMCJP/iqPsMX9+f/AMCJP/iqALNFVvsMX9+f/wACJP8A4qj7DF/fn/8AAiT/AOKoAs0VW+wxf35//AiT/wCKo+wxf35//AiT/wCKoAs0VW+wxf35/wDwIk/+Ko+wxf35/wDwIk/+KoAs0VW+wxf35/8AwIk/+Ko+wxf35/8AwIk/+KoAs0VW+wxf35//AAIk/wDiqPsMX9+f/wACJP8A4qgCzRVb7DF/fn/8CJP/AIqj7DF/fn/8CJP/AIqgCzRVb7DF/fn/APAiT/4qj7DF/fn/APAiT/4qgCzRVb7DF/fn/wDAiT/4qj7DF/fn/wDAiT/4qgCzRVb7DF/fn/8AAiT/AOKo+wxf35//AAIk/wDiqALNFVvsMX9+f/wIk/8AiqPsMX9+f/wIk/8AiqALNFVvsMX9+f8A8CJP/iqPsMX9+f8A8CJP/iqALNFVvsMX9+f/AMCJP/iqPsMX9+f/AMCJP/iqALNFVvsMX9+f/wACJP8A4qj7DF/fn/8AAiT/AOKoAs0VW+wxf35//AiT/wCKo+wxf35//AiT/wCKoAs0VW+wxf35/wDwIk/+Ko+wxf35/wDwIk/+KoAs0VW+wxf35/8AwIk/+Ko+wxf35/8AwIk/+KoAs0VW+wxf35//AAIk/wDiqPsMX9+f/wACJP8A4qgCzRVb7DF/fn/8CJP/AIqj7DF/fn/8CJP/AIqgCzRVb7DF/fn/APAiT/4qj7DF/fn/APAiT/4qgCzRVb7DF/fn/wDAiT/4qj7DF/fn/wDAiT/4qgAT/kJz/wDXGP8Am9WajigjgUiMEbjkkkkn6k8mpKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACoZ7u3tnhW5uIoWnfy4hI4UyPgnauepwCcD0qauO+IGmR6xN4csZZHh83UyUljOGicW8zK6+6sAR9KBpXv8zrRcQm5a3EqGdUDtEGG4KSQCR1wSDz7GpK8zXxLOmra5LeNLaapaWFnYziCNHcTNPKqlA5C/OHVlLHA3DPQilsdf1+7uU0s6nd2z/259hea4itXuFjNmZSreWpi3BuhAOMDOeRTSu9P61/4KBq39eVz0uivPNN1vXYrzSJbnVZb1LjUr3TpLZoIlEiwrOUkyqgiQ+UM4IXk4UVL4M8Qa7qOoafJqX2o2+p2TXJW6azVEYbSPIETmQp8xB3hiPlyQcglhPTc76iiikAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVDNd21vNDFPcRRSXDFIUdwDIwBJCg9TgE4HYVNXnfiqbUdT1q/k0rR9RvjpKRpaTW0kKIlyrLM4bfKrEECJThTwXHtR11A9EorznU/Euq3kOr6tousNBaWdvZ3NrB9njZZPNGWEhI3FSMcKVI9asPqet22sTaC2s3Eqy6nDbpqLwwia3RrcykABNhJZNoJQ/f7nBpdbdRJqSujvqK4rRfEGozapplndXy3EZur61efy0X7T5JARjgYDcNnbgEg8DoMyw1rXddjtvJ1+W0Q6ZcXZltoIGMjpOUTlkYbdvXA545HeXJK39dL/kPrb+t7fmekUVw3gqW61DxRqGqXV3I7Xel2ErQBUEasyOTjC7uDnGWP3j14x3NaSTi7MmMlJXQUUUUigooooAKKKKACiiigAooooAKKKKACsm98V+HtOvjZahr2mWl2uM2895GkgyMj5Sc85Fa1cHENbm8T+L7XSdP025gmuIkke9vHj2k2sY+4sTBhj/AGhnp70rvoUkne53CXEEk8kMc0byxAGSNWBZM9MjtntUleaxGXwxNqWm213ePcqmmWMUtvHE08zCNh8vmsIwxCnliQPQnAqfSde1jVLTTtOk1O4tZLjU7u1e9KW7XAWIMyx/Kph8zjBIUjCNjn5qejbS6f1+qM1LRN/1v/keh0V51rPiDXYrx7TTby5u10/Txcte2S2SxTvvkU+d5zjCL5YDeWQcluV4Fd/ZzNc2ME8ihHljV2VWDBSRnAI4P1FHQrbT+v61JqKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiqmpXU9pabrO1a6uHYJHGOFye7Hso7n+tAFuiuO0K7121v71r6VtUtvtxt5RGmGt22ph1HdMtgjqAM+udbxl9q/4QvVfsPmed9mbHlZ34x823HOcZxjvSeiGXLLXdI1K9ns9O1Syu7q3JE0EFwjvFg4O5Qcjnjmr9chr/wDZ/wDZ/hj+w/svmfbrf+zPJxjyf+WmzH8Pk7+nGMVjaR4l1pNP0fU5tU/tZtSN1G9isMaqnlJIwZCqhsgxhW3EjL8AcU3ZJt9L/hqJaux6RRXli+LfElt4cv72SW53SaHLqEEt4bLCyKFIMKQuzNEd/wDGCRhcsc1d8X67q2h2dzFY6vqc91p+nm9lkSGzWMFmbZ5xcAlMoVCxLu45JJFNxatf+v6sC12/rb/M9BnuYLWNXupo4UZ1jVpHCgsxAVee5JAA7k1LXmWpPcQeJNWu/tTSia+0ZfImhieNN8ygkZTOQM4OcjOeoBF6HxBq/wBrsr9tRZkvtUu7BtMMUeyBYhNtdTt37x5KlssV+Y4A4wvs8z7tfl/mEfe2/rf/ACO/qK4uYLO2kuLuaOCCJS8ksrBVRR1JJ4AriPCGra1Nc+HW1TVpL9dZ0ZryZJIIkEUiiHGzYoOD5hyGLdsY6VU+IF1e32m+LbX+1TY22naWrC1EcZFz5ivlnLKWxxtXaV5BznpVOLUrP+tbBHU9HBz0orzvWvFGo23iNRp95e/ZrfUrOwlj8q2W1zIY9yEsTOz7ZNwK7VGB1wc0bG91bTLL7Dp9zqVw2oa9qCtJbLaecoR3O1PN2x5YgschjgNgDjE+f9dP8w+zf+uv+R6lRWN4U1C81Pw5BcakFFyHkiciSNy2x2UFjGzIGIA3AHAOR2rZoejAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKj89PST/v23+FHnp6Sf9+2/woAkoqPz09JP+/bf4UeenpJ/37b/AAoAkoqPz09JP+/bf4UeenpJ/wB+2/woAkoqPz09JP8Av23+FHnp6Sf9+2/woAkoqPz09JP+/bf4UeenpJ/37b/CgCSio/PT0k/79t/hR56ekn/ftv8ACgCSio/PT0k/79t/hR56ekn/AH7b/CgCSio/PT0k/wC/bf4UeenpJ/37b/CgCSopbaCeSJ54Y5HhffEzoCY2wRlc9DgkZHYml89PST/v23+FHnp6Sf8Aftv8KAK9xpOnXf2j7Xp9rP8AaoxFcebCrecgzhWyPmAycA+ppttomlWYjFpplnAImDxiK3RdjBNgIwOCF+XPpx0q156ekn/ftv8ACjz09JP+/bf4UARLptijRlLO3UxStNGREo2SNnc444Y7myep3H1qKz0XS9PvLi7sNNs7W5uTmeaCBUeU5zlmAy3JzzVrz09JP+/bf4UeenpJ/wB+2/woAkoqPz09JP8Av23+FHnp6Sf9+2/woAkoqPz09JP+/bf4UeenpJ/37b/CgCSio/PT0k/79t/hR56ekn/ftv8ACgCSio/PT0k/79t/hR56ekn/AH7b/CgCSio/PT0k/wC/bf4UeenpJ/37b/CgCSio/PT0k/79t/hR56ekn/ftv8KAJKKj89PST/v23+FHnp6Sf9+2/wAKAJKigtoLZWW2hjhV3aRhGgUMzHLMcdyTknvS+enpJ/37b/Cjz09JP+/bf4UAVo9F0uK3lgi02zSGYASRrAoVwDkZGMHBJpbrSdOvobiG90+1uIrrb9oSWFXWbb93cCPmxgYz0xVjz09JP+/bf4UeenpJ/wB+2/wosC02KVx4f0a802HT7vSLGeyt8eTbS2yNHHgYG1SMDAOOKnj02xhOYbK3jOxo8rEo+VjuK9OhPJHrU3np6Sf9+2/wo89PST/v23+FAEEOlafb3SXVvYWsVxHALdJkhVXWIHIjBAyFz/D0q3Ufnp6Sf9+2/wAKPPT0k/79t/hQBJRUfnp6Sf8Aftv8KPPT0k/79t/hQBJRUfnp6Sf9+2/wo89PST/v23+FAElFR+enpJ/37b/Cjz09JP8Av23+FAElFR+enpJ/37b/AAo89PST/v23+FAElFR+enpJ/wB+2/wo89PST/v23+FAElFR+enpJ/37b/Cjz09JP+/bf4UASVFHbQQzSywwxxyTsGldUAMhAABY9zgAc9hS+enpJ/37b/Cjz09JP+/bf4UAVrzRtM1GKePUNNtLqO42+cs8CuJdv3dwI5x2z0pkugaPPp72E+k2MlnIwd7Z7ZDGzDGCVIwTwOfYVc89PST/AL9t/hR56ekn/ftv8KAKNz4d0S9htIbzR7C4ishi1SW1RlgHHCAj5eg6egrSqPz09JP+/bf4UeenpJ/37b/CgCSio/PT0k/79t/hR56ekn/ftv8ACgCSio/PT0k/79t/hR56ekn/AH7b/CgCSio/PT0k/wC/bf4UeenpJ/37b/CgCSio/PT0k/79t/hR56ekn/ftv8KAJKKj89PST/v23+FHnp6Sf9+2/wAKAJKKj89PST/v23+FHnp6Sf8Aftv8KAJKKj89PST/AL9t/hR56ekn/ftv8KAJKKj89PST/v23+FHnp6Sf9+2/woAr2GnpYPdsjs/2q4adsj7pIAwP++auVH56ekn/AH7b/Cjz09JP+/bf4UAVLPQtI068nvNO0uytLq4JM08FuiPLk5O5gMnnnmqfh3wrp3h21j8i2tWvvL8uW+S3VJZhnIDEZJHTgk9K1/PT0k/79t/hR56ekn/ftv8ACgCjD4c0S3W8W30bT4hfAi7CWqL9oznO/A+bqeuepp97oek6lcRz6jpdldzRI0cck9ujsisMMoJHAIJBHerfnp6Sf9+2/wAKPPT0k/79t/hQBVk0XS5roXU2m2clwqoglaBS4VG3IM4zhWAI9DyKVdG0tNUl1JNNtFv5k2S3QgUSyLxwXxkjgcE9hVnz09JP+/bf4UeenpJ/37b/AAoAih0+yt/s/kWcEX2aLyYNkSr5UfHyLgfKvyrwOOB6VDf6HpOqzRy6ppdleyRKyxvc26SFA3BALA4B71b89PST/v23+FHnp6Sf9+2/wo8wKcug6PPqBvptKsZLxgqm4e2QyEKQVG7GeCoI9CB6Uk/h/RrqG5iutIsZo7uQS3CSWyMJnHRnBHzEYHJq756ekn/ftv8ACjz09JP+/bf4UAJb20FnbR29pDHBBEoSOKJAqoo6AAcAe1S1H56ekn/ftv8ACjz09JP+/bf4UASUVH56ekn/AH7b/Cjz09JP+/bf4UASUVH56ekn/ftv8KPPT0k/79t/hQBJRUfnp6Sf9+2/wo89PST/AL9t/hQBJRUfnp6Sf9+2/wAKPPT0k/79t/hQBJRUfnp6Sf8Aftv8KPPT0k/79t/hQBJRUfnp6Sf9+2/wo89PST/v23+FAElFR+enpJ/37b/Cjz09JP8Av23+FAElFR+enpJ/37b/AAo89PST/v23+FAElFR+enpJ/wB+2/wo89PST/v23+FAElFR+enpJ/37b/CgTKSAA/PrG3+FAElFFFABRRRQAUUUUAFFFFABRRVPUtX03RrdbjWNQtbCFm2LJdTrEpbrjLEDPB4oAuUVn6X4g0bWzINF1ax1AxY8wWlykuzPTO0nGcH8q0KACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAoqN7iGOeOGSWNZZQTHGzAM+OuB3xmqVp4h0XUNQlsLDV7C6vId3m20NyjyJtODlQcjB4OaANGiqmnatp2sQPPpN/a30SOY2ktplkVWHJUlScHkce9W6ACiiigAooooAKKKKACiiigAooooAKKKjM8IuFtzKgmZS6xlhuKggEgdcAkc+4oAkoqqup2D2r3K3tu1vHIY3lEq7FcNtKk5wCG4x68VaoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKo3OuaTZalDp95qdnb3txjybaW4RZJMnA2qTk8gjigC9RVT+1dPGqjTDf2328x+YLTzl83Z/e2Zzj3xTINb0q51SXTLbU7Oa/hBMtpHcI0sYGMkoDkdR1HegC9RUazwvcPAsqNNGoZ4ww3KDnBI7A4OPoab9stfLjk+0w7JX2Rt5gw7f3Qe54PHtQBNRVF9b0qPV00qTU7NNRkGUs2uEEzDGchM5PAJ6dqstcwKZQ00YMK7pcuP3YxnJ9Bgd6OlwJaKajrJGrxsHRgCrKcgg9wadQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUVUbVLFNVTTXu4lvZEMiQFvmZR3x+f5H0q3Taa3EmnsFFFGecd6Qwoozjr36UUAFFFFABRRRQAUUUUAFFFFABRRRQAUyb/USf7p/lT6QjIweRSkrqw1ozzO01e6tPAvhWx0+7v4JpNJ+0uLGO23bI0QFme4OxUBcZAUseMYAOZG8V609j4ckTUY0OvWNu11IYhjTWbb++XCEHezbAHON20jIDCu6m0HSLiK0iuNKspY7Ig2qPbowtyOhQEfLjA6Y6UqaJpUdtNbx6ZZrDPEsMsS26BZEAwFYYwVA4APGKtyTbfn/n/mvu+4f9f1/X+d0DCgEk8dT3paQAKoCgAAYAHalqRBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHm41eG6+J2mXl1b6jDct9qto1n064RIolAC/MU2kMQXLA4AKgngVBNc63pPgt9JmvdM1EX+jXc8EmloyuhCbt4YufNVi/3gF5I4549Le3hknjmkhjaWIERyMoLJnrg9s4qrZaLpWm3Vxc6dplnaT3R3TywW6o0xyTliBluSTz61NtLFJ2lcw9AEA8ZX40/b9k/smx/1eNuczbenfZt/DHtXVVU07SdO0eB4NJsLWxidzI0dtCsasx4LEKBk8Dn2q3Vt3ZCVgooopDCiiigAooooAKKKKACiiigArmfFs/8AZFzp2uiPcLTzopcDJ8t4ycD3Lxxj8a6aobm0t72Dyby3iuItwby5UDLkHIOD3BAI9xSauNaHBQaY2m3tp4euBvimlttRmkx95o0JkOO/7yGMn/rpWWvjPWxb3k8N/evHceH7rU7Z7yG1UKylPLeJIizKh3n5ZSzcD0OfUmtoGukuWhjNwiGNZSg3qpIJUHqASoJHsPSqMPhrQrZJUt9F06JZg4kWO0RQ4fG8HA53bRn1wM9KpPXXz/K3+X3ArJ6+X+f46nJ63qGvaS6Srq17dWtrZfa7l7VLNpIyWJJmicKTDtBC+WQ/ysCScGotb8VahF4kD6ZeXxtYtTs7GRPJtltsymPcp3Ezs+2TcCu1RgdcHPa32haTqctvLqWl2V5JanMD3Fukhi6fdJHy9B09BRLoOkT6g1/PpVlJeMFDXD26GQhSCoLEZ4Kgj0IHpQmk1fb/AIP9Inp8jjD4u1e0G6Zjcf2ektrdpsT99cZlER46E+SnAAH74cend2MdxDp9vHez/aLlIlWWbaF8xwOWwOBk88VGdL09mlZrG2LTSpPITCvzyLja545YbVweowPSrdLp/X9d/lYb1f8AX9f0wooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAK5TxNFFrD3Ph3SreJry9Ebahc7QBbRA8Mx7uQpCL+JwBXV1kX3hPw5ql415qegaXeXL43T3FlHI7YGBliCeAKOqDocsRGNQkx/x/8A/CVJ1K78eUvTvjyc/hntWnHFDrniSw/si2ih0vQ7qWR7lFCrLcFXRo4wOoBdi7dN3Ayd2Oi/srTv7UGpfYLX7eI/KF15K+aE/u78Zx7Zqnb+E/DtpqAv7TQNLgvFYuLmKzjWQMep3AZycnn3ojokv62S/S4mtW/63b/WxzlrHB4Z13xdPZJI7rZ2037yRpJJpSJcZZiSSTtUDoOAMAAU/V/D9rp8nhOQ+ZJcWd5BbIzysVA2PuYLnG5j1bGT0zXYPY2kkkkj2sLPLs8xjGCX2nK5PfB5HpT5beGcxmeGOQxOJI96g7GHRhnoeTzS6r1X4B0a73/E5bWIodc1VdF0i2iAt76C91O7VQBE6Msirn+KVtq/7q8k8qDV1PTbbW/BfiC9vVlaO4e4nRBKyq6xoY48gH5kIQPtPBzkiugm8J+HbjUTqE+gaXLelxIbl7OMyFh0bcRnPHWtIWtuLT7KIIhb7Nnk7Bs24xt29MY7UpK8HH+un+RadpX/AK/rUr6L/wAgGw/69o//AEEVdpqIscapGoRFACqowAB2Ap1XJ3k2ZxVopBRRRSKCiiigAooooAKKKKACiiigArkvHPjm38KWXlQbZtTmX91CeiD++3t7d/zI62uRtvh7YL4wuPEF/O97LJKZYoZF+WJux6nOBgD0/LHThvYqXNW2XTv5HPiPbOPLS3fXsZ/gjw1f3kcGt+Lx598shltPNBEsQOc7vbJyFI+X2zgd9RRUVq0q0+Z/d0RdGlGlDlX39wrnfGa29vpC6pJetYXdixa1mUbiznjy9n8YbGMfj2roqp3WlWV7fWl5dQCWazLNAWJIQnGTjpnjr2qKclGSbOinJRkmzlfCss+v67LqPiPdb6pZDbBpZBUWqMP9Zg/eLevbpXbVVm020n1G3v5YFN1bBlilBIIDDBHHUex+tWqqrNTd1p5diqk1N3QUUUVkZBRRRQAUUUUAFFR+U/8Az3k/Jf8ACjyn/wCe8n5L/hQBJRUflP8A895PyX/Cjyn/AOe8n5L/AIUASUVH5T/895PyX/Cjyn/57yfkv+FAElFR+U//AD3k/Jf8KPKf/nvJ+S/4UASUVH5T/wDPeT8l/wAKPKf/AJ7yfkv+FAElFR+U/wDz3k/Jf8KPKf8A57yfkv8AhQBJRUflP/z3k/Jf8KPKf/nvJ+S/4UASUVH5T/8APeT8l/wo8p/+e8n5L/hQBJRUflP/AM95PyX/AAo8p/8AnvJ+S/4UASUVH5T/APPeT8l/wo8p/wDnvJ+S/wCFAElFR+U//PeT8l/wo8p/+e8n5L/hQBJRUflP/wA95PyX/Cjyn/57yfkv+FAElFR+U/8Az3k/Jf8ACjyn/wCe8n5L/hQBJRUflP8A895PyX/Cjyn/AOe8n5L/AIUASUVH5T/895PyX/Cjyn/57yfkv+FAElFR+U//AD3k/Jf8KPKf/nvJ+S/4UASUVH5T/wDPeT8l/wAKPKf/AJ7yfkv+FAElFR+U/wDz3k/Jf8KPKf8A57yfkv8AhQBJRUflP/z3k/Jf8KPKf/nvJ+S/4UASUVH5T/8APeT8l/wo8p/+e8n5L/hQBJRUflP/AM95PyX/AAo8p/8AnvJ+S/4UASUVH5T/APPeT8l/wo8p/wDnvJ+S/wCFAElFR+U//PeT8l/wo8p/+e8n5L/hQBJRUflP/wA95PyX/Cjyn/57yfkv+FAElFR+U/8Az3k/Jf8ACjyn/wCe8n5L/hQBJRUflP8A895PyX/Cjyn/AOe8n5L/AIUASUVH5T/895PyX/Cjyn/57yfkv+FAElFR+U//AD3k/Jf8KPKf/nvJ+S/4UASUVH5T/wDPeT8l/wAKPKf/AJ7yfkv+FAElFR+U/wDz3k/Jf8KPKf8A57yfkv8AhQBJRUflP/z3k/Jf8KPKf/nvJ+S/4UASUVH5T/8APeT8l/wo8p/+e8n5L/hQBJRUflP/AM95PyX/AAo8p/8AnvJ+S/4UASUVH5T/APPeT8l/wo8p/wDnvJ+S/wCFAElFR+U//PeT8l/wo8p/+e8n5L/hQBJRUflP/wA95PyX/Cjyn/57yfkv+FAElFR+U/8Az3k/Jf8ACjyn/wCe8n5L/hQBJRUflP8A895PyX/Cjyn/AOe8n5L/AIUASUVH5T/895PyX/Cjyn/57yfkv+FAElFR+U//AD3k/Jf8KPKf/nvJ+S/4UASUVH5T/wDPeT8l/wAKPKf/AJ7yfkv+FAElFR+U/wDz3k/Jf8KPKf8A57yfkv8AhQBJRUflP/z3k/Jf8KPKf/nvJ+S/4UASUVH5T/8APeT8l/wo8p/+e8n5L/hQBJRUflP/AM95PyX/AAo8p/8AnvJ+S/4UASUVH5T/APPeT8l/wo8p/wDnvJ+S/wCFAElFR+U//PeT8l/wo8p/+e8n5L/hQBJRUflP/wA95PyX/Cjyn/57yfkv+FAElFR+U/8Az3k/Jf8ACjyn/wCe8n5L/hQBJRUflP8A895PyX/Cjyn/AOe8n5L/AIUASUVH5T/895PyX/Cjyn/57yfkv+FAElFR+U//AD3k/Jf8KPKf/nvJ+S/4UASUVH5T/wDPeT8l/wAKPKf/AJ7yfkv+FAElFR+U/wDz3k/Jf8KPKf8A57yfkv8AhQBJRUflP/z3k/Jf8KPKf/nvJ+S/4UASUVH5T/8APeT8l/woEbAgmZz7ELz+lAElFFFABRRRQAUUUUAFFFFABRRWX4g1aTR9LEttAtxdTTR29vE77VaR2CjJwcAZyeDwDQBqUVyOraz4k8N6Pf3epjTryOG3E0d5DC0EcbbwCjxtKzHg5DBscEEDjOl/wmWhi1uLiS6liS2ljhlWa1ljcNIQEwjKGIYnAIBB7GgLm5RXNweLrVWvpLyUmJLpIbeGCxuGuPmhWTa8WzeW5J+UcLjOCDTLjx3pdvqkcDib7G+myai14IJSiIhGQcJwcZzzkEAEZIo8v62v+Qf1+h09Fc4/j7w7GsrSXk6iLaZCbGcbEbOJT8nERwf3n3P9qrlz4p0m01VdOnuJROzpEWW2laJHfGxGlClFY5GFZgTuX1GXZhdbmvRWXZeI9M1DUnsbWaVpl3YL28iRybThtkjKFfB4O0nFN1jxPpOgyBNTuJIz5TTOY7eSURxjq7lFIRfdsDg+lTdAa1FcTqvjWe3LxWE1u7TajFa28qafc3AhRoUkJkRcZY7uAGAww7hhVvTfiBpt3phurqG8gf7VPbrCljcSSOImKlwix7tuMZOMKW2k5p/1+X+Y/wCvz/yZ1dFRwTxXNvHPbyLLFKodHU5DKRkEH0qSgW4UUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRXJaR4outV16REv9HW28yUDTixF6sSEp533jkFl+7sHDD5sjBp2njfUILH+09Xgt2s7vS59Us4reNkkjSPafLdixDMVkQ5AUAg8dDR/X9fcxpNuy/r+v8Ag7Hc0Vz/AIe1TVZtRutM1/7K13Fbw3SyWkTxpsk3DYQzMcq0bc55BHAroKbVhBRRRSAKKKKACiiigAooooAKKKKACiisu91Oaz8QafauIhaXcUwMjZDCVArKB2wV8w/8BpAalFcdY+LdRvND87yLYX0t9BFAm1gvkTFWV2XOciNmzzjKHoOK04/G2gypcuLuVI7a3e6aSS0mRXhT70kbMoEqjI5TcOR6in/X6h/X6G9RWFJ4z0SLyjJNdKJF35NhPiNN20O52fu0JBw7YUgEgkAmprrxRpNnqq6dcTyicyJEWW2laJHfGxGlClFY5GFZgTuX1GQDXorMh8RaVcNarFdjddwPcQhkZdyLjcTkfKRnocHg8cHF2zu4b+ygu7Vi8E8ayRsVKkqwyDg4I4PegCaiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAorB8S3WuWNpcXumXGl29pZ2zzyG9ieRpWUE7chlEYwPvfN1+6Mc5V54t1MO93aW8ENhY29tPexTxu0r+dyyowIClF55Vsk4+XrSTv/X9dgem/wDX9XOzormtT1PXNJ1W1luJNOfT7u+S0jtEifz8Pxv8wtgkYLFdnAB+bjNTQa9fTeOG0eTTvs1kLSSaOeRwXmZXRcqATtT5u/JPYAAs1rt/Wl/yBu2/9XdvzN+iuS1TxXqVn4fTULPTVuRLcsqylgscUPnBFZudxZgcgKPrirHifU9c0WGfVLeTTRp1qEzbSxO01wSwBAcMAhOQFG18n64CTuB0tFY9zq93HfX8dnYtei0hixFEVV3lcn5csQoAXaSfQ9+lc/c+NbqPwnoN/c3elaRcapOY5Z73LW8ACSMeroTygUfMOv4UXA7iiqumSyT6XbzS3dtetJGH+0WibIpQeQyjc3BGP4jVqm9GAUUUUAFFFFABRRRQAUUUUAFFFFABRRUdxcRWtvJPcyLFDGpZ3c4CgdSTRuGxJRXE6V4xvfEfiaE+H0iuNHQtFdrINksR52ydehxxx6g4OCO2rWrRnRaU9/63MqdWNVXjsFFFZeta1/Ygt7i5t3axZitxcIc/Z+m1mH93PU9qzjFydkbxi5OyNSisWLxHHe+IhpmlQ/bEiXdeXSOPLgyPlXP8TH0HQVtU5RcdwlFx3CiiipJCiiigAooooAKKKKACiiigArL8QaTJrGliK2nW3uoZo7i3ldNyrIjBhkZGQcYPI4JrUooA5PVdJ8S+INJu7e/bT7ASQ+XHa287TpI29W3tI0SsMBSAoXvkk8YTWfCt7qOr3d1BLbqk1xp0qh2YEC3mMj546kdPfriutooWjTXT/hxWOM1LwnfT3GpzxwWN39p1FLuFJLua1kjAgSPKzxgtG2VPQHKkjjJqpeeDdeuNLS3a6t7maXSbuwmkubmRmiMpDR4fYTKFwELNtYgbuTxXfUULT+vK35FX1v8A1vc4/XPCV9qcWvLBLbqdS0WPT4d7MNsi+bktgcL+8HIyevFU9X8H67qWvvcNdRSWq31rdQGS/nAiSJoy0QtwPLySjsJDk/NjA6jvKKrmd0+3/Di6W/raxyOkeG9UsPFH2sC3srINM8iWl9O0dyXOR/orjy4Tn5iyMSTn+8ayvHV6mkapqbw3tglxqmkfZjBeSukjFTJs8hQp85iZCDGCD931Feh0VDV1yjTs2zitP8JahE8c0kkAH9pW16V3NkJHbLEw6fe3KeOmO/aoJvCGrC38oWmn3Riv7q4glGpXFnMqTOXyJokLJjcVKYYNwdwxiu8opt3b/rsv0EtEl/XX/MpaPa3Vlotna6hdG8uoYUSW4brIwGC1XaKKbd3cAooopAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHHWfhO9gvrS0aOwXS7K/mv4rhHb7RI0nmHYU24XHmkF953BfujdxXtPBN/PZf2bq8tulnaaXNpdnJbys8kiSbR5jqVAVgsaDALAknkcCu5ooWisv66fkO7Tujn/D+l6pFqV1qevC0S7lt4bVY7SZ5E2R7juJZVOWaRuMcADk10FFFNu4gooopAFFFFABRRRQAUUUUAFFFFABWJ4q0a51rSVi0+SOK7jk3RSyMV2blaNyCAedjtjjrjtW3RQNOxz8/hx28U2moW7Rx20Fts287lkQOsZC4xjbLJnnsv4cnF4A8QTW9x9vureS5l0a605ppdQuLgzSyeXiX94uI1JQ5RRhe2c8emUU03e/9df8AP+rAnbb+rHHeJfDGp6lcQy6WltFOtp9nW+S/ntJ7c5zn92CJ0BwRG+0Ag8nccVNW8H69qWvNcSXkU9st9aXULy3067EiaMtH9nUeVklHYScn5sYHUd5RQm00/wCu4uljhrnwPfTLfolxCiSXWLbErgpbOZPNBGMBv38uAMj5U5HbuEVURUQBVUYAHYUtFLpYHq7hRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBy3ifStc1XVbVbe006+0eFRI9pdX0lv5swbKl9sThkXAIXgEnJBwKj1Xw7q9/eXIi+xpaarHbrfhp3LwGM/N5eFw+5flydmMZ56V1tFAPU5OPTfER8Xyalf2Gl3cCSmOzkOoSK1rAcAlYvIIMhGSTu5+6CB10NS0q/l186nYPArppk1tEJWI/esyspOAfl+Xk9fatyilZWS/rawdf673MLU/D7TeDk0TTzGnlJAkZkJACxsp7A9lqjq+m+IbvxQl2tjpd/p1rtayguNQkg2S4+aR1ELhmHRecDrjJyOrop7y5mHSxn6VYS2n22W58vz7u6eZjGSRtwETqBzsRM++ax7LRdV0jw/pUNpa6XfXtgHBFy7Rgbs8pKEYrwcEbOc9R36iik0BleG9Jk0XQ47Sd42mMkk0nlAiNWkdnKpn+EFsD2FatFFMAooooAKKKKACiiigAooooAKKKKACvKfFE+u+O/FkvhvTo3tdNtJcTysDtYg8sx78/dXv1+nq1FdOHr+wk5qN308vM569H20VFuy6+fkZmgaBY+G9KSw02Pai8u5+9I3dmPrWnRRWEpSnJyk7tm0YqK5YrQKxfEcWrXsEWm6QFhS73Lc3rkHyI8c7V7sc4HYVtUURlyu5pGXK7nM6BoFz4UvV0/TF8/RJsv8AOwEltJjk5/jVsfUH2rpqKKc5ubu9xzm5u73CiiioICiiigAooooAj89PST/v23+FHnp6Sf8Aftv8KkooAj89PST/AL9t/hR56ekn/ftv8KkooAj89PST/v23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/AAqSigCPz09JP+/bf4UeenpJ/wB+2/wqSigCPz09JP8Av23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/CpKKAI/PT0k/79t/hR56ekn/ftv8ACpKKAI/PT0k/79t/hR56ekn/AH7b/CpKKAI/PT0k/wC/bf4UeenpJ/37b/CpKKAI/PT0k/79t/hR56ekn/ftv8KkooAj89PST/v23+FHnp6Sf9+2/wAKkooAj89PST/v23+FHnp6Sf8Aftv8KkooAj89PST/AL9t/hR56ekn/ftv8KkooAj89PST/v23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/AAqSigCPz09JP+/bf4UeenpJ/wB+2/wqSigCPz09JP8Av23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/CpKKAI/PT0k/79t/hR56ekn/ftv8ACpKKAI/PT0k/79t/hR56ekn/AH7b/CpKKAI/PT0k/wC/bf4UeenpJ/37b/CpKKAI/PT0k/79t/hR56ekn/ftv8KkooAj89PST/v23+FHnp6Sf9+2/wAKkooAj89PST/v23+FHnp6Sf8Aftv8KkooAj89PST/AL9t/hR56ekn/ftv8KkooAj89PST/v23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/AAqSigCPz09JP+/bf4UeenpJ/wB+2/wqSigCPz09JP8Av23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/CpKKAI/PT0k/79t/hR56ekn/ftv8ACpKKAI/PT0k/79t/hR56ekn/AH7b/CpKKAI/PT0k/wC/bf4UeenpJ/37b/CpKKAI/PT0k/79t/hR56ekn/ftv8KkooAj89PST/v23+FHnp6Sf9+2/wAKkooAj89PST/v23+FHnp6Sf8Aftv8KkooAj89PST/AL9t/hR56ekn/ftv8KkooAj89PST/v23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/AAqSigCPz09JP+/bf4UeenpJ/wB+2/wqSigCPz09JP8Av23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/CpKKAI/PT0k/79t/hR56ekn/ftv8ACpKKAI/PT0k/79t/hR56ekn/AH7b/CpKKAI/PT0k/wC/bf4UeenpJ/37b/CpKKAI/PT0k/79t/hR56ekn/ftv8KkooAj89PST/v23+FHnp6Sf9+2/wAKkooAj89PST/v23+FHnp6Sf8Aftv8KkooAj89PST/AL9t/hR56ekn/ftv8KkooAj89PST/v23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/AAqSigCPz09JP+/bf4UeenpJ/wB+2/wqSigCPz09JP8Av23+FHnp6Sf9+2/wqSigCPz09JP+/bf4UeenpJ/37b/CpKKAI/PT0k/79t/hQJlJAAfn1jb/AAqSigAooooAKKKKACiiigAooooAKiubmCztZLm8mjggiUvJLK4VUUdSSeAPepawfF9vbXejwwXV8lg7XkDW08sXmRiZZA0YZcjILKBjIzkAEHFAF+117SL6x+22Wq2VxaeYIvPhuEePeSAF3A4ySQMdeRV+vMPEE9/uvrHVYbFNYE2nP9vsWcxSxfbcR74WPyOOeNzZ5+bjA0f7W1qHUp9AbV7h86vHZrqbww+dFG1qJugQRklhtBKdG9eae60/rb/Mdn9x31Feew63rF9faRpQ102/nXt9aSXsUERkuBB90qGUoH4IOFxw3A4xXsPEGtazeLat4hXT1h064nM8UERNwYbh4hL86kBCqhmAx1G0gVN1a/8AW1xf1+Nj0ZLiGSeSCOaNpYsGSNWBZM9MjtmpK8yh8Q6vqG2e3lj0y51B9KWR4rZCyCaNi4+ZcnHbdnH5g34dU1ma/i0F9anjP9qz2p1LyYfPdEgEqpgp5e4ljzs+6h4zzT1u12v+Fv8ANEKasn3/AOD/AJHfUVz3gjUrvVvC63WoXa3k32u6i89UVQ6pPIikBeMbVH/166Gm1Z2LCiiikAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBGZ4RcLbmVBMyl1jLDcVBAJA64BI59xVa01nTNQubm3sNRtLqe0bbcRQzq7QnkYcA5U8Hr6GueGl29r8UfOh8wT3mlTmWV5GduJY8AZPyqMnCjAGSepJrlX0u8fTdL8PTXukwWiaJeQDUrSYyb7YxqonfIUICdpI3MCc/NxR9m/9bv/AC/qw1rK3mv0/wAz0zTdX03WYHm0jULW/iR9jSWs6yqrdcEqTg8jirlcb4el1J/G88d/b6fAYdKhSZLCVpUB3t5eXKr23kLjgdzmuyo/r8bEp3/ryuFFFFAwooooAKKKKACiiigAooooAKjM8IuFtzKgmZS6xlhuKggEgdcAkc+4qSuZ8Wz/ANkXOna6I9wtPOilwMny3jJwPcvHGPxpXsNK5uLqdg9q9yt7btbxyGN5RKuxXDbSpOcAhuMevFWq88g0xtNvbTw9cDfFNLbajNJj7zRoTIcd/wB5DGT/ANdKy18Z62Le8nhv7147jw/danbPeQ2qhWUp5bxJEWZUO8/LKWbgehzSV3b+tFf9H9wLV/d+L/4a/qer0VwGt6hr2kukq6te3Vra2X2u5e1SzaSMliSZonCkw7QQvlkP8rAknBqLW/FWoReJA+mXl8bWLU7OxkTybZbbMpj3KdxM7Ptk3ArtUYHXByJNtL+t7C6X/rueiUV56fF2r2g3TMbj+z0ltbtNifvrjMoiPHQnyU4AA/fDj07uxjuIdPt472f7RcpEqyzbQvmOBy2BwMnnil0v/X9f5oHo7f1/X+TJ6KKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAKOo65pOkSQpq2p2di9wSIVubhIzIeOF3EZ6jp60+41XT7S+t7K7v7aC6us+RBJMqyTY67VJy34Vi+KZI5Vm0rTbKC61nVLVoPnQYjg5BeVuvlqXOF6sTgDqRzmrWkVjF4gtWdpLqC002K1eQrubaxEZGe/mZ/HpST/r+v6+4Unbb+v6/rc7r+3NJOsHSRqdn/aQGTZ/aE87GM52Z3dOenSrRnhFwtuZUEzKXWMsNxUEAkDrgEjn3FcxfRRa3rcGm6RbxJb6ffreX92qgBZV+YRrj70hJG49lyDyQKie0s9J+Jj6gS4abSZ5biWWRnO1ZY8AZPyqBnCjAGSepJImtL+f4K/9f8EH5eX52/r7uh1LXlqkLzPcwrFG2x3MgCq2cYJ7HPGKgudc0my1KHT7zU7O3vbjHk20twiySZOBtUnJ5BHFcdr+jRzfDu1u9Rjk+1xyR3TR+awRZpZkZiVzhiCSASOO2K1PE0UWsPc+HdKt4mvL0RtqFztAFtEDwzHu5CkIv4nAFGqdmO+l0dDe6pp+mwyS6jfW1pHEoeR55lQIpOASSeATxn1qCTxBo0Wkpqsur2Kac5wl41yghbnHD5weRjrWPJeWMF5qmsanEsyQ3MdrZgJ5ju6DgRj++ZJHXj0646ZkeharaXFjqoGmrqcl/cXh026uDGg8yPbhZFVv3iqoJYKQcv0BzQnp/X9bfd8wf9f166f8MdzFLHNCksLrJHIoZHQ5DA8gg9xT65vwFLPP4RjlnMZElzctCYhhDEZ3KFf9nbjHtiukqnoAUUUUgCiiigAooooAKKKKACiiigAoorkvHPjm38KWXlQbZtTmX91CeiD++3t7d/zI0pUp1pqEFdszqVIUoOc3ZI6JtUsU1VNNe7iW9kQyJAW+ZlHfH5/kfSrdcD4I8NX95HBrfi8effLIZbTzQRLEDnO72ychSPl9s4HfVdenCnPki7237X8iaNSdSPNJWvt6eYUZ5x3ornfGa29vpC6pJetYXdixa1mUbiznjy9n8YbGMfj2rOEeaSj3OmEeaSidFnHXv0orifCss+v67LqPiPdb6pZDbBpZBUWqMP8AWYP3i3r26V21OpD2cuUdSHI+UKKKKzMwooooAKKKKACiiigAooooAKiubaC8tZLa8hjnglUpJFKgZXU9QQeCPapaKAM+08P6NYWRs7HSbG2tTKJjBDbIiGQEEPtAxuBVeevA9KkudI028huYbzT7W4iuyGuElgVlmIAALgj5sbRjPoPSrlFG4bbGFqHhHS9Qm01JrO0On2EckYsHtVaJgwXA29AF25xj8qS58H6Ve6xHd31lZ3NtDapbw2c1qrpEVYkMueBgHAwOK3qKXW4dLFZ9OspJjNJZ27Sl0cu0SliyfdOcdRk4PasvxB4cGrWXkWiaZGHn86eO+01bqG4O3GWTcp3D5SG3du9btFAGdoOkJoejQ2CSeaVLvJJsCb3di7ttHABZjx2rRoopt3BKysFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAEZt4TdC5MMZnVDGJdo3BSQSueuMgHHtVaz0XS9OmuZdP02ztZLtt1w8ECoZjycuQPmPJ6+pq7RQBT03SNN0a3a30fT7WwhZt7R2sCxKWwBkhQBnAHPtVyiigAooooAKKKKACiiigAooooAKKKKACobm0t72Dyby3iuItwby5UDLkHIOD3BAI9xU1FAETW0DXSXLQxm4RDGspQb1UkEqD1AJUEj2HpVGHw1oVskqW+i6dEswcSLHaIocPjeDgc7toz64GeladFAFC+0LSdTlt5dS0uyvJLU5ge4t0kMXT7pI+XoOnoKJdB0ifUGv59KspLxgoa4e3QyEKQVBYjPBUEehA9Kv0UAVDpens0rNY2xaaVJ5CYV+eRcbXPHLDauD1GB6VboooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAMvUvDOg6zcLcaxomnX8yrsWS6tI5WC9cZYE45PFTNomlPPaTvplm0tiu21kNupa3GMYQ4+UY9MVeooDcyD4S8OHUv7RPh/Sze+b532k2UfmeZnO/djO7POetX57G0umZrm1hmZ4mhYyRhi0bfeQ5/hOBkdDViigOtyOe3guYTDcwxzRHGUkUMpwcjg+4BrMvvCfhzVLxrzU9A0u8uXxunuLKOR2wMDLEE8AVr0UdbgULzQtI1Cy+x3+l2V1a+YZvInt0dN5JJbaRjcSxOevJ9agXwp4dXTG05dB0wWLSea1qLOPyi+MbimMZwOuM1rUUANRFjjVI1CIoAVVGAAOwFOoooAKKKKACiiigAooooAKKKKACiiigArkbb4e2C+MLjxBfzveyySmWKGRflibsepzgYA9Pyx11Fa06s6d+R2voZzpQqW51ewUUUVkaBVO60qyvb60vLqASzWZZoCxJCE4ycdM8de1XKKabWw02tirNptpPqNvfywKbq2DLFKCQQGGCOOo9j9atUUUXb3C7YUUUUhBRRRQAUUUUAR+U/8Az3k/Jf8ACjyn/wCe8n5L/hUlFAEflP8A895PyX/Cjyn/AOe8n5L/AIVJRQBH5T/895PyX/Cjyn/57yfkv+FSUUAR+U//AD3k/Jf8KPKf/nvJ+S/4VJRQBH5T/wDPeT8l/wAKPKf/AJ7yfkv+FSUUAR+U/wDz3k/Jf8KPKf8A57yfkv8AhUlFAEflP/z3k/Jf8KPKf/nvJ+S/4VJRQBH5T/8APeT8l/wo8p/+e8n5L/hUlFAEflP/AM95PyX/AAo8p/8AnvJ+S/4VJRQBH5T/APPeT8l/wo8p/wDnvJ+S/wCFSUUAR+U//PeT8l/wo8p/+e8n5L/hUlFAEflP/wA95PyX/Cjyn/57yfkv+FSUUAR+U/8Az3k/Jf8ACjyn/wCe8n5L/hUlFAEflP8A895PyX/Cjyn/AOe8n5L/AIVJRQBH5T/895PyX/Cjyn/57yfkv+FSUUAR+U//AD3k/Jf8KPKf/nvJ+S/4VJRQBH5T/wDPeT8l/wAKPKf/AJ7yfkv+FSUUAR+U/wDz3k/Jf8KPKf8A57yfkv8AhUlFAEflP/z3k/Jf8KPKf/nvJ+S/4VJRQBH5T/8APeT8l/wo8p/+e8n5L/hUlFAEflP/AM95PyX/AAo8p/8AnvJ+S/4VJRQBH5T/APPeT8l/wo8p/wDnvJ+S/wCFSUUAR+U//PeT8l/wo8p/+e8n5L/hUlFAEflP/wA95PyX/Cjyn/57yfkv+FSUUAR+U/8Az3k/Jf8ACjyn/wCe8n5L/hUlFAEflP8A895PyX/Cjyn/AOe8n5L/AIVJRQBH5T/895PyX/Cjyn/57yfkv+FSUUAR+U//AD3k/Jf8KPKf/nvJ+S/4VJRQBH5T/wDPeT8l/wAKPKf/AJ7yfkv+FSUUAR+U/wDz3k/Jf8KPKf8A57yfkv8AhUlFAEflP/z3k/Jf8KPKf/nvJ+S/4VJRQBH5T/8APeT8l/wo8p/+e8n5L/hUlFAEflP/AM95PyX/AAo8p/8AnvJ+S/4VJRQBH5T/APPeT8l/wo8p/wDnvJ+S/wCFSUUAR+U//PeT8l/wo8p/+e8n5L/hUlFAEflP/wA95PyX/Cjyn/57yfkv+FSUUAR+U/8Az3k/Jf8ACjyn/wCe8n5L/hUlFAEflP8A895PyX/Cjyn/AOe8n5L/AIVJRQBH5T/895PyX/Cjyn/57yfkv+FSUUAR+U//AD3k/Jf8KPKf/nvJ+S/4VJRQBH5T/wDPeT8l/wAKPKf/AJ7yfkv+FSUUAR+U/wDz3k/Jf8KPKf8A57yfkv8AhUlFAEflP/z3k/Jf8KPKf/nvJ+S/4VJRQBH5T/8APeT8l/wo8p/+e8n5L/hUlFAEflP/AM95PyX/AAo8p/8AnvJ+S/4VJRQBH5T/APPeT8l/wo8p/wDnvJ+S/wCFSUUAR+U//PeT8l/wo8p/+e8n5L/hUlFAEflP/wA95PyX/Cjyn/57yfkv+FSUUAR+U/8Az3k/Jf8ACjyn/wCe8n5L/hUlFAEflP8A895PyX/Cjyn/AOe8n5L/AIVJRQBH5T/895PyX/Cjyn/57yfkv+FSUUAR+U//AD3k/Jf8KPKf/nvJ+S/4VJRQBH5T/wDPeT8l/wAKPKf/AJ7yfkv+FSUUAR+U/wDz3k/Jf8KPKf8A57yfkv8AhUlFAEflP/z3k/Jf8KPKf/nvJ+S/4VJRQBH5T/8APeT8l/woEbAgmZz7ELz+lSUUAFFFFABRRRQAUUUUAFFFFABWX4g1aTR9LEttAtxdTTR29vE77VaR2CjJwcAZyeDwDWpWX4g0mTWNLEVtOtvdQzR3FvK6blWRGDDIyMg4weRwTS/r/MPQrW15rOlyTt4lksZrGO3M5v7WEwLEVPKNG0jk5HIYHsQQOMi+MtENjPdyXE8EdvLHDKlxZzRSK0hAQeW6BzuJABxg1UvrLxZq2m3qPcWWkytavFBDaXDyhpDjDtKY0ZMAFQFHG4nJIAGRb+CNVC3TSPbRm4vdPuQjX090yCCUM4Mso3OSBwTjrjAAyaWrSfl+f+X/AAdRPy/rT/P/AIB3VvOl1bRzxCRUkUMoljaNgD6qwBU+xANS0UUhhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHPT+I5h44tNFtoEa1aKU3Fwx5WRVVgij2VgST/eXHfGPo/jd7zR9S1qTWdGv7eztprk6bYRn7REqklA7+c3JA5+Qcn2q0ngq9tvEmn31vr97La273EkkFwsB5lOSBtiBIJznLZxjBqGXwjqGsItpq0Gl6faW1hPZQtphfMnmqFJ2FVESgDPl5fnHzfLkzrYpW5tdtP67mr4e1TVZtRutM1/7K13Fbw3SyWkTxpsk3DYQzMcq0bc55BHAroK5/wAP6XqkWpXWp68LRLuW3htVjtJnkTZHuO4llU5ZpG4xwAOTXQVcrX0IV+v9d/xCiiikMKKKKACiiigAooooAKKKKACsu91Oaz8QafauIhaXcUwMjZDCVArKB2wV8w/8BrUrE8VaNc61pKxafJHFdxybopZGK7NytG5BAPOx2xx1x2pa9Bq3UybHxbqN5ofneRbC+lvoIoE2sF8iYqyuy5zkRs2ecZQ9BxWnH420GVLlxdypHbW73TSSWkyK8KfekjZlAlUZHKbhyPUU2fw47eKbTULdo47aC22bedyyIHWMhcYxtlkzz2X8OTi8AeIJre4+33VvJcy6Ndac00uoXFwZpZPLxL+8XEakocoowvbOeKVr/wBdtPxX4/cLfXy/4P4fkdfJ4z0SLyjJNdKJF35NhPiNN20O52fu0JBw7YUgEgkAmprrxRpNnqq6dcTyicyJEWW2laJHfGxGlClFY5GFZgTuX1GcXxL4Y1PUriGXS0top1tPs63yX89pPbnOc/uwROgOCI32gEHk7jipq3g/XtS15riS8intlvrS6heW+nXYkTRlo/s6jysko7CTk/NjA6gSTaT/AK1/pk62+X4nUw+ItKuGtViuxuu4HuIQyMu5FxuJyPlIz0ODweODi7Z3cN/ZQXdqxeCeNZI2KlSVYZBwcEcHvXGXPge+mW/RLiFEkusW2JXBS2cyeaCMYDfv5cAZHypyO3cIqoiogCqowAOwpdL/ANf1t87je+n9f1/kLRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBg+JbrXLG0uL3TLjS7e0s7Z55DexPI0rKCduQyiMYH3vm6/dGOcq88W6mHe7tLeCGwsbe2nvYp43aV/O5ZUYEBSi88q2ScfL1q14n0rXNV1W1W3tNOvtHhUSPaXV9Jb+bMGypfbE4ZFwCF4BJyQcCo9V8O6vf3lyIvsaWmqx2634ady8BjPzeXhcPuX5cnZjGeelJX/AK+f9f8ADilfp/X9fr5FjU9T1zSdVtZbiTTn0+7vktI7RIn8/D8b/MLYJGCxXZwAfm4zU0GvX03jhtHk077NZC0kmjnkcF5mV0XKgE7U+bvyT2AALUo9N8RHxfJqV/YaXdwJKY7OQ6hIrWsBwCVi8ggyEZJO7n7oIHXQ1LSr+XXzqdg8CummTW0QlYj96zKyk4B+X5eT19qI30+f5afj+fkNptu3l+ev4fkZ2qeK9Ss/D6ahZ6atyJbllWUsFjih84IrNzuLMDkBR9cVY8T6nrmiwz6pbyaaNOtQmbaWJ2muCWAIDhgEJyAo2vk/XAsan4fabwcmiaeY08pIEjMhIAWNlPYHstUdX03xDd+KEu1sdLv9OtdrWUFxqEkGyXHzSOohcMw6LzgdcZOQbSsG6bNG7vdWudRurLRDZxGBIt092juEZixb5VYbsKF4yv3s54xWHF4p1i5nt9Jil02PUZL+4tDfPC7QOIUDkpF5gJY527d/G1zk4xV/UdK19dIvhoz2aX9/fGWd2uGixDwoCOI2Ify0QZ2nBJI6CoZNDvpfC8Gmv4a8OyRQS8adc3LzwMmOH8xoch9xJOUbPPOTR0/ry/LYHv8A1/Xn/wAObXh7Vxreix3eY2cSSQyGE5QvG7IxX2JUkexrTrK8N6OdC0KGyd0kl3PLM6LtUySOXbaOy5Y4HpitWqe4IKKKKQBRRRQAUUUUAFFFFABRRRQAVHcXEVrbyT3MixQxqWd3OAoHUk1JXlPiifXfHfiyXw3p0b2um2kuJ5WB2sQeWY9+fur36/Tpw1D20rN2S1b8jnxFb2MdFdvZHQaV4xvfEfiaE+H0iuNHQtFdrINksR52ydehxxx6g4OCO2rM0DQLHw3pSWGmx7UXl3P3pG7sx9a06mvKnKf7pWS+9+pVGNSMf3ju3/WgVl61rX9iC3uLm3drFmK3Fwhz9n6bWYf3c9T2rUrF8RxatewRabpAWFLvctzeuQfIjxztXuxzgdhWdNJySlsdNNJySewReI473xENM0qH7YkS7ry6Rx5cGR8q5/iY+g6CtquZ0DQLnwperp+mL5+iTZf52AktpMcnP8atj6g+1dNVVVBP3Nv6/r+rlVFFP3NgooorIyCiiigAooooAKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/PaP/voUfaIf+e0f/fQoAkoqP7RD/wA9o/8AvoUfaIf+e0f/AH0KAJKKj+0Q/wDPaP8A76FH2iH/AJ7R/wDfQoAkoqP7RD/z2j/76FH2iH/ntH/30KAJKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/AD2j/wC+hR9oh/57R/8AfQoAkoqP7RD/AM9o/wDvoUfaIf8AntH/AN9CgCSio/tEP/PaP/voUfaIf+e0f/fQoAkoqP7RD/z2j/76FH2iH/ntH/30KAJKKj+0Q/8APaP/AL6FH2iH/ntH/wB9CgCSio/tEP8Az2j/AO+hR9oh/wCe0f8A30KAJKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/PaP/voUfaIf+e0f/fQoAkoqP7RD/wA9o/8AvoUfaIf+e0f/AH0KAJKKj+0Q/wDPaP8A76FH2iH/AJ7R/wDfQoAkoqP7RD/z2j/76FH2iH/ntH/30KAJKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/AD2j/wC+hR9oh/57R/8AfQoAkoqP7RD/AM9o/wDvoUfaIf8AntH/AN9CgCSio/tEP/PaP/voUfaIf+e0f/fQoAkoqP7RD/z2j/76FH2iH/ntH/30KAJKKj+0Q/8APaP/AL6FH2iH/ntH/wB9CgCSio/tEP8Az2j/AO+hR9oh/wCe0f8A30KAJKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/PaP/voUfaIf+e0f/fQoAkoqP7RD/wA9o/8AvoUfaIf+e0f/AH0KAJKKj+0Q/wDPaP8A76FH2iH/AJ7R/wDfQoAkoqP7RD/z2j/76FH2iH/ntH/30KAJKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/AD2j/wC+hR9oh/57R/8AfQoAkoqP7RD/AM9o/wDvoUfaIf8AntH/AN9CgCSio/tEP/PaP/voUfaIf+e0f/fQoAkoqP7RD/z2j/76FH2iH/ntH/30KAJKKj+0Q/8APaP/AL6FH2iH/ntH/wB9CgCSio/tEP8Az2j/AO+hR9oh/wCe0f8A30KAJKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/PaP/voUfaIf+e0f/fQoAkoqP7RD/wA9o/8AvoUfaIf+e0f/AH0KAJKKj+0Q/wDPaP8A76FH2iH/AJ7R/wDfQoAkoqP7RD/z2j/76FH2iH/ntH/30KAJKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/AD2j/wC+hR9oh/57R/8AfQoAkoqP7RD/AM9o/wDvoUfaIf8AntH/AN9CgCSio/tEP/PaP/voUfaIf+e0f/fQoAkoqP7RD/z2j/76FH2iH/ntH/30KAJKKj+0Q/8APaP/AL6FH2iH/ntH/wB9CgCSio/tEP8Az2j/AO+hR9oh/wCe0f8A30KAJKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/PaP/voUfaIf+e0f/fQoAkoqP7RD/wA9o/8AvoUfaIf+e0f/AH0KAJKKj+0Q/wDPaP8A76FH2iH/AJ7R/wDfQoAkoqP7RD/z2j/76FH2iH/ntH/30KAJKKj+0Q/89o/++hR9oh/57R/99CgCSio/tEP/AD2j/wC+hR9oh/57R/8AfQoAkoqP7RD/AM9o/wDvoUCeIkASoSegDCgCSiiigAooooAKKKKACiiigAooqtf6lY6VZtd6peW9lbKQGmuZVjQE8DLMQKALNFUNM13SdbWRtG1Sy1BYiBIbS4SUIT0ztJx0q/QAUUhIVSWIAAySe1Mt7iG7to7i1ljmhlUPHLGwZXUjIII4II70ASUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUVGZ4RcLbmVBMyl1jLDcVBAJA64BI59xVK08Q6LqGoS2Fhq9hdXkO7zbaG5R5E2nByoORg8HNAGjRVTTtW07WIHn0m/tb6JHMbSW0yyKrDkqSpODyOPerdABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRTZJEijaSVlREBZmY4CgdSTQA6iq7X1oklsj3UKvdZ+zqZADNgbjtH8XHPHbmnxXME0kscM0cjwtslVGBMbYBww7HBBwexFAEtFRC5ga6e2WaMzogd4gw3KpJAJHUAkHn2NS0AFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRVHUdc0nSJIU1bU7Oxe4JEK3NwkZkPHC7iM9R09afcarp9pfW9ld39tBdXWfIgkmVZJsddqk5b8KALdFUf7c0k6wdJGp2f9pAZNn9oTzsYznZnd056dKtGeEXC25lQTMpdYyw3FQQCQOuASOfcUASUVA15apC8z3MKxRtsdzIAqtnGCexzxioLnXNJstSh0+81Ozt724x5NtLcIskmTgbVJyeQRxQBeoqre6pp+mwyS6jfW1pHEoeR55lQIpOASSeATxn1qCTxBo0Wkpqsur2Kac5wl41yghbnHD5weRjrSA0aKZFLHNCksLrJHIoZHQ5DA8gg9xT6YBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRVRtUsU1VNNe7iW9kQyJAW+ZlHfH5/kfSrdNprcSaewUUUZ5x3pDCijOOvfpRQAUUUUAFFFFABRRRQAUUUUAFFFFABXMeO1nfR7BbOSOKc6paeW8sZkVT5q4JUMpI9sj6109RzW8NyqrcQxyhHEih1DbWByGGe4PINHVPzX5g9U15M5zVbnXNP0y0t7vUbb7TfX8dsb2zs/KFvGwPIWR5AWJG0E5GWHy8c5Vrq+r3t7DoZ1eWPF/dW51aKKHzZREisEwUMe/LMCQuP3TcKenbXdpbX9pJa31vFc28q7ZIZkDo49CDwRVWXQNHm0lNLm0mxk06PGyze2Qwrg5GExgc+1LXX+u3+XzuD6f13/z/AAOMtNd1XXoLG3k1gaeP7OuLl7mGKMi9McvlhsOGAj2/MwXB+dcMBXSeA+Ph14d5z/xLLfn/ALZrWjd6LpV/b28F9plncw2rBoI5rdXWIgYBUEYUgelWoIIrW3jgtokhhiUJHHGoVUUcAADgD2qk7Jrv/wAH/P8AAX9fl/kSUUUUhhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAciunW1h8UvtUW7zrrSpnlkllZs4ljwBknaoycAYAyTjJOeemudb0nwW+kzXumaiL/RrueCTS0ZXQhN28MXPmqxf7wC8kcc8elm3hN0LkwxmdUMYl2jcFJBK564yAce1VbLRdK026uLnTtMs7Se6O6eWC3VGmOScsQMtySefWlb3UvX83/mOLtLm81+n+Rh6AIB4yvxp+37J/ZNj/AKvG3OZtvTvs2/hj2rqqqadpOnaPA8Gk2FrYxO5kaO2hWNWY8FiFAyeBz7Vbqm7slKwUUUUhhRRRQAUUUUAFFFFABRRRQAUUUUAFNdFkjZHAZWBBB7inUUtwPNbaKY6ZG+GL+FIFjVmUncY5iH/ExQj/AL+ehp0uoarDJZwaUt+p1hrvUnl00WplYB1EYzcME2iNlzgFuBjAzXoCWFpH9o8u1hX7Uxa42xgecSAuW/vHAA57Cob3Q9J1OxistS0yzu7WEgxwT26SRpgYGFIwMDijX+vPf8bfcHX+vl+F/vuedw6jqd9q13qh1mHSbqPw5bXcxhEMqSMrznk5ceVnrsOeRhx39I025kvdJtLqaPypJ4EkeP8AuEqCR+GaguvD+jXskcl7pNjcPFs8tpbZHKbM7MEjjbuOPTJx1rRqr/19/wDn+APV3/rZL9PxCiiikAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBzXimSOVZtK02ygutZ1S1aD50GI4OQXlbr5alzherE4A6kc5q1pFYxeILVnaS6gtNNitXkK7m2sRGRnv5mfx6V2mpeGdB1m4W41jRNOv5lXYsl1aRysF64ywJxyeKmbRNKee0nfTLNpbFdtrIbdS1uMYwhx8ox6YpJf19/+Ypa/16f18kYV9FFretwabpFvElvp9+t5f3aqAFlX5hGuPvSEkbj2XIPJAqJ7Sz0n4mPqBLhptJnluJZZGc7VljwBk/KoGcKMAZJ6kk7J8JeHDqX9onw/pZvfN877SbKPzPMznfuxndnnPWr89jaXTM1zawzM8TQsZIwxaNvvIc/wnAyOhoWlref4q3+X3DaTevl+DucRr+jRzfDu1u9Rjk+1xyR3TR+awRZpZkZiVzhiCSASOO2K1PE0UWsPc+HdKt4mvL0RtqFztAFtEDwzHu5CkIv4nAFdPPbwXMJhuYY5ojjKSKGU4ORwfcA1mX3hPw5ql415qegaXeXL43T3FlHI7YGBliCeAKLK/kHTXczZLyxgvNU1jU4lmSG5jtbMBPMd3QcCMf3zJI68enXHTMj0LVbS4sdVA01dTkv7i8Om3VwY0HmR7cLIqt+8VVBLBSDl+gOa6280LSNQsvsd/pdldWvmGbyJ7dHTeSSW2kY3EsTnryfWoF8KeHV0xtOXQdMFi0nmtaizj8ovjG4pjGcDrjNG39elwev9ev6FHwFLPP4RjlnMZElzctCYhhDEZ3KFf9nbjHtiukpqIscapGoRFACqowAB2Ap1N26AFFFFABRRRQAUUUUAFFFFABRRRQAVyXjnxzb+FLLyoNs2pzL+6hPRB/fb29u/5kdbXI23w9sF8YXHiC/ne9lklMsUMi/LE3Y9TnAwB6fljpw3sVLmrbLp38jnxHtnHlpbvr2M/wAEeGr+8jg1vxePPvlkMtp5oIliBznd7ZOQpHy+2cDvqKKitWlWnzP7uiLo0o0ocq+/uFc74zW3t9IXVJL1rC7sWLWsyjcWc8eXs/jDYxj8e1dFVO60qyvb60vLqASzWZZoCxJCE4ycdM8de1RTkoyTZ0U5KMk2cr4Vln1/XZdR8R7rfVLIbYNLIKi1Rh/rMH7xb17dK7aqs2m2k+o29/LApurYMsUoJBAYYI46j2P1q1VVZqbutPLsVUmpu6CiiisjIKKKKACiiigAoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40bZv+ekf/fB/xoAkoqPbN/z0j/74P+NG2b/npH/3wf8AGgCSio9s3/PSP/vg/wCNG2b/AJ6R/wDfB/xoAkoqPbN/z0j/AO+D/jRtm/56R/8AfB/xoAkoqPbN/wA9I/8Avg/40AS5GXQjvhD/AI0ASUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFc9c+LJI9Uv7Kx8ParqR09lSeW2NuFDMgcACSZWPDDtSuOzZ0NFVdN1C31bS7bULFy9vdRLLGxGCVIyMg9D7VaptWdhBRRRQAUUUUAFFFFABRRRQAUUUUAFFFYf9sapceIbix03TrOW0s3jjubia9aN1LKGIWMRMGwrA8suc9utHWwG5RRRQAUUUUAFFFFABRWTrviXTPDqW/wDaU+JbqVYoIUG6SQswXIHoCwyegz7itC7uY7KzmupziKCNpHI9AMn+VJtJXHZt2JqKzNG1afVoTLNo99pqbVaM3jQnzAeeBHI+McdcdfrWnTasK9worN0rWU1ee8W3tLiOG1naAXEuwJM6sVcIAxb5WBGWAz2yKq23jHRry91aC3ut6aREst3cAZiUHfkBv4ivltnHAPHUEA6X+Y7O9jcorI0PxDFrZmT7Fd2E8SpIYLxUDmNwSjjYzDBww65BUggVr0NWEFFFFABRRVe7vIbNUMxOXYKiKMsx9hQBYooooAKKKKACisvxB4i07wxpq3+sSPDatMkJkWMsELHALY+6o7k8fpWlHIksayROro4DKynIYHoQaAHUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABXB3Gga9d6z4sn0fWL7SZZZojbKsMPkzkW8YzueNm6grlTwR0JFd5RSauUnY8yvbGyn0yxN74b1CXTE0g29jYNYyzS2l0pIORglXIChZTgcE7vmzUj6RrMMptbmC6meKGHWWmRSyyXccSoYgy9SWQNjvk16TRT11t/XX8/yRFrv+vL9F+LPK9d0SRLe1uptPmudQmgluFiudJlvIxNI5k2pJCQ9tKpKr5hwMBcZK8dH8RcyeC7b7VBIS2oWHmQxMS3/AB8R5VSOSe3FdjVa/wBOtdTgSG+i82OOaOdRuIw8bh0PB7MoOOlEdGuyaf4jet/Q8zvtJmksZW0fS2tPDh1NJPsNzpEssZTyCrN9iBR9nm7Ttx94b9pHNXdA8PvPqugf2lYSXNnbwXzxC409oI4czRNEBE7uUAwSisQQAOFxgekUULR/12sBwfjWPTZfG+hjW9Ol1Gy+w3nmQR2r3IJ3Q4LRKGLDOOxwcHjGRy/iDS9XuPD0FjdaI81zFpDmzkn06W+mVyzlY0kDbIJFUR/OwJY4AyQBXrcmn2supwahJFm6gjeKOTcflVypYYzg5KL+VWaXReX/AAf8yr6/15HlnjayiaHW5tY0u5u7uSzhOmXS2rv5ChfnXzcYjO7cWBK7gwGG6VL/AGXqY8eebOmL06mJI7qLQ5Xk+zddpvPNEax7MqUIznOFJIJ7e+8L6TqWpC+vIJZJcoWQXMqxSFDlS8QYI5B6FlPQegrXprRp/wBf1/wSGtLFezFuLfFnD5Me9/k8kxfNuO47SB1OTnvnPOc1YoooGFcLrHhoXPiKSax8LrHqM95DP/b6zxnykXbu5J8xTtQr5aqUOeTy2O6ooWjuHSxFcwC5tpIWeSMOuN8TlGX3BHQ1y/h7w1rNjrE8+sa5d3dvE+LaMztiQddzjPbpj1B7V1tFABRRRQAUUUUAc/42tZbvw2UtYHmm+12h2xoWbaLmNieOwAJPsKdr2n6jf2Wp29xcrJpVzaPEbWztcXfzLg7ZWk2E9cAp6fjvUUrFczRyXhFGXVb0acdYGjrBEqDV/P3mfLbin2j94F27M/w5+7zurGvNMmbxRIw0y5bXW1mKaDUvszFFsRs3L5+NqrsEimMnJYk7fmzXo1FUnaSl/W9/68tCGrq39bW/rzOX8CaNDo2m6kkOmpYGXVLpwqwCLenmtsOMDI24x2xjFUdW0W81DVvF1vbW7Kt7ocFtbuylY3f/AEgbQ3TjcufTIrtqKSulbysXze85ed/xuct4bkk1PxFfauLK6tLdrG2tVW7tXgcyIZHf5XAJA8xRnoTnBOK6miim3ckKKKKQBWTcxvZawL6RGuIXATIGWg+g9D+f9daigAooooAKKKKAMDxrrVlofheebUbCXU1uCLaKxiiLtcyPwseMHGT3P6nAOf8ADPw3qvhjwilprdzulkkaWOzRt8dijciFWOS2PXJHYdMnr6KACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigD//2Q==)

Rysunek 11 Lista wszystkich operacji dostępnych dla użytkownika; opracowanie własne w oparciu o https://mui.com/components/tables/

Powyższa implementacja wizualna przedstawia listę występującą bezpośrednio w oknie przeglądarki. Oparta na module wspomnianej wcześniej w rozdziale nr 3 biblioteki MUI pozwala na dynamiczne filtrowanie po większości z właściwości każdej operacji.

W liście operacji zawarta jest możliwość dodania nowej operacji dostępna pod znakiem plusa zawartym w prawym górnym rogu. Reszta aktywności dostępnych na liście operacji znajduje się bezpośrednio obok każdej z pojedynczych operacji. Znak ołówka pozwala na modyfikacje wybranej operacji finansowej oraz wyświetlenie modułu z wypełnionymi danymi wypełnionymi uprzednio w modyfikowanym rekordzie. Zaraz obok ikony ołówka znajduje się kolejna przypominająca znak „x”. Pozwala ona na usunięcie wybranego rekordu z listy operacji i bezpośrednio z bazy danych.

<TableCell align = "right" >

<IconButton

color = "primary"

onClick = { ()=> openInPopup(operation)}>

<EditIcon fontSize = "small" / >

</IconButton>

< IconButton

color = "secondary"

onClick = { () => deleteOperation(operation) }>

<CloseIcon fontSize = "small" / >

</IconButton>

</TableCell>

Fragment kodu źródłowego 5 Fragment wyświetlający do każdej operacji metodę edycji oraz usunięcia; opracowanie własne

Wyświetlony powyżej fragment pozwala na dodanie do listy operacji modułów odpowiedzialnych za modyfikacje oraz usuwanie rekordów z bazy danych. Dalsze funkcje oparte są na wyświetleniu okna modalnego lub prostego strzału do części serwerowej z końcówką służącą do usuwania operacji.

* 1. Dodawanie oraz modyfikacja operacji

Jako podstawową modyfikację danych potrzebną do aktualizacji całkowitego budżetu finansowego użytkownika używane są odpowiednie funkcje dodawania modyfikacji oraz dodawania operacji finansowej w wytworzonym oprogramowaniu.

Obraz zawierający stół

Opis wygenerowany automatycznie

Rysunek 12 Okno modalne wyświetlane podczas dodawaniu lub modyfikacji operacji; opracowanie własne przy pomocy https://mui.com/components/grid

Okno modalne wyświetlające się po odpowiednim wybraniu niektórych z dostępnych czynności. Akcja całego procesu ukazywana jest bezpośrednio na poprzednim widoku. Tło z poprzedniej fasady zostaje lekko przyciemnione i na pierwszym planie ukazuje się powyższe okno modalne.

Z dostępnych akcji jakie dostępne są w wyświetlonym oknie jest odpowiednie wpisanie wszystkich przydatnych informacji o operacji, wybór daty oraz kategorii czy jej typu.

interface OperationFormProps {

openPopup: boolean

setOpenPopup: Dispatch<SetStateAction<boolean>>

recordForEdit: Operation

editOrAddOperation: (operation: Operation) => void

}

Fragment kodu źródłowego 6 Właściwości dostępne w całej klasie pozwalającej na modyfikację dodawanej operacji; opracowanie własne

Z dostępnych w powyższym fragmencie kodu źródłowego ukazują się dwie właściwości pozwalające na wyświetlenie okna modalnego z przodu widoku oraz dwie pozwalające na wybranie ciała obiektu na którym przeprowadzana będzie edycja. Właściwości dostępne w React w powyższym interfejsie używane są również w celu przekazania ich pomiędzy klasami zawierające informacje z poprzedniego widoku.

export default function OperationForm(props: OperationFormProps) {

const [values, setValues] = React.useState<Operation>(emptyOperation);

const { openPopup, setOpenPopup, recordForEdit, editOrAddOperation } = props

const handleSubmit = (formValue: {

name: string; amount: number,

place: string, date: string,

category: string, walletId: number }) => {

const { name, amount, place, date, category, walletId } = formValue;

const data: Operation = {

id: values.id,

name: name,

amount: amount,

place: place,

date: date,

category: category,

walletId: walletId

};

if (data.id === "0")

api.post(`/api/operations`, data, { headers: authHeader() })

else

api.post(`/api/operations/${data.id}`, data, {headers: authHeader() })

setRecordForEdit(emptyOperation)

setOpenPopup(false)

refreshData(null)

. . .

}

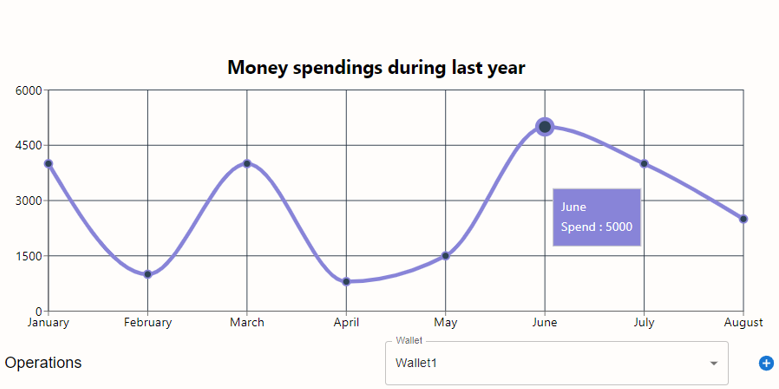
}

Fragment kodu źródłowego 7 Metoda służąca do obrobienia danych i wysłania ich bezpośrednio do części serwerowej; opracowanie własne; logika z paru funkcji zawarta w jednej

Powyższy przykład zawierający w sobie fragmenty z wielu implementacji ukazaną w jednej metodzie na celu ma przetworzenie danych oraz wysłanie ich za pomocą biblioteki axios[[27]](#footnote-27) służącej do komunikacji z zewnętrznym serwisem wystawionym za pomocą aplikacji typu REST. Na samym początku ukazany jest pierwszy fragment metody użytej jako komponent funkcyjny React wraz z dostępnymi React Hook’ami opisanymi w małym stopniu w poprzednim rozdziale. Dane odpowiednio są obrabiane przez cały proces zawarty w metodzie a następne dodawane do zapytania typu REST. Zależenie od ścieżki dodania nowej operacji lub jej modyfikacji odpytywany jest odpowiednia końcówka która powinna poprawnie obsłużyć dany zakres informacji. Metoda kończona jest zamknięciem okna modalnego przy pomocy zmiany stanu React Hook’a używanego w danej chwili do odpowiedniej akcji.

* 1. Dostępne wykresy

Niezwykle ważnym elementem każdej aplikacji służącej do monitorowania wydatków finansowych jest moduł pozwalający na sprawdzenie wydatków w sposób przyjazny użytkownikowi, czyli wizualnie.



Rysunek 13 Główny wykres udostępniany nad listą wszystkich operacji; opracowanie własne na podstawie biblioteki służącej do tworzenia wykresów https://blog.logrocket.com/using-recharts-react-add-charts/

Z powyższej prezentacji wynikają główne właściwości jakie rozwija użytkownikom użyta w rozwiązaniu biblioteka wykresów Recharts[[28]](#footnote-28). Punkty na wykresie łączone są przy pomocy linii przez co wyświetlane dla użytkownika dane są dobrze widoczne. Wydatki wydają się wyświetlane w sposób ciągły bez przerw co pozwala na rozeznanie, które miesiące były pod tym aspektem najdroższe. Dodatkowo klient aplikacji w łatwy sposób może zobaczyć tendencje wzrostowe lub spadkowe w kierunku którym idą łączone z punktami się linie.

Wyjątkowo pomocną wartościa jest rozwinięcie danych zaraz po najechaniu na wybraną przez użytkownika kropke. Całość przedstawionego rozwiązania jest zachowana w standardach reaktywności aplikacji na zmiane rozmiaru okna, linie zaraz po przeskalowaniu są dynamicznie zmieniane przy pomocy przyjemnej dla oka animacji rysowania wykresu.

export default function OperationChart() {

const[operations,setOperations]=React.useState<Array<GroupedOperation>>();

const { wallet } = props

React.useEffect(() => {

DataService.getGroupedOperations(wallet)

.then(response => {

setOperations(response.data)

})

.catch(e => {

console.log(e);

});

}, [wallet]);

return (

<>

<h2>Money spendings during last year</h2>

<ResponsiveContainer aspect={3}>

// Potrzebny w przypadku użycia reaktywności

<LineChart

width={150}

height={150}

data={operations}

margin={{

left: 15,

right: 30,

top: 15,

bottom: 10,

}}

>

// Dotyczy prostokątów służących do lepszego zrozumienia

<CartesianGrid horizontal={true} stroke="#243240" />

// Informacje dotyczące podpisów na wykresie

<XAxis dataKey="month" tick={{fill:"#000"}}/>

<YAxis tick={{fill:"#000"}}/>

// Narzędzie służące do wyświetlania po najechaniu na kropki

<Tooltip contentStyle={{. . .}}/>

// Właściwości linii używanej na wykresie

<Line type="monotone" dataKey="Cost" . . . />

</LineChart>

</ResponsiveContainer>

</>

);

}

Fragment kodu źródłowego 8 Przykład wykresu linearnego użytego w projekcie

Powyższa implementacja wykresu jest uzupełniana poprzez zaczytane i obrobione wcześniej dane wyszukane bezpośrednio po stronie serwera. Właściwość „data” służąca jako kontener na dane potrzebne do wyświetlenia na wykresie pobierana jest przy użyciu znanej z wcześniejszych implementacji biblioteki axios bezpośrednio z backendu oprogramowania. Całość rozwiązania dostępna jest jeszcze w innych zawartych w aplikacji wykresach pozwalających na przedstawianie mniej ogólnych informacji dotyczących wydatków.

fun groupOperations(wallet: Wallet): List<GroupedOperation> {

val operations = wallet.operations

val groupedOperation = mutableListOf<GroupedOperation>()

val monthsFromLastYear = createMonthList(MONTHS\_IN\_YEAR)

val previousYear = LocalDate.now().minusMonths(MONTHS\_IN\_YEAR)

for(month in 1..MONTHS\_IN\_YEAR) {

val currentRotationDate = previousYear.plusMonths(month)

operations.filter {

it.date.isAfter(

LocalDate.of(currentRotationDate.year,

currentRotationDate.monthValue, 1)

) && it.date.isBefore(

LocalDate.of(currentRotationDate.year,

currentRotationDate.monthValue % 12 + 1, 1)

) && it.category.type == CategoryType.COST

}.sumOf { it.amount }.let {groupedOperation.add(

GroupedOperation(capitalize(

monthsFromLastYear[(month - 1).toInt()]), it.toInt())

)

}

}

return groupedOperation

}

Fragment kodu źródłowego 9 Fragment kodu źródłowego 8 Przykład implementacji grupowania danych wysyłanych z serwera jako dane w wykresie użytego w projekcie

Powyższa implementacja wykonywana po stronie serwera pozwala pogrupować wszystkie operacje dostępne przez ostatni rok, odpowiednio dla każdego obiektu portfela użytkownika. Całość wysyłana jest poprzez listę zawierającą obiekty klasy *GroupedOperation* zawierającej pola informujące o miesiącu w postaci nazwy któremu odpowiada kolejność począwszy od nazwy ostatniego miesiąca sprzed roku idąc kolejno w górę. Dodatkowo obiekt zawiera w sobie przeliczoną wartość wydanej sumy każdego miesiąca. Całość rozwiązania oparta jest na kolekcjach języka Kotlin oraz jej filtrom oraz metodom sumującym. Podczas liczenia wszystkich operacji każdy obiekt utrzymywany na bazie danych jest filtrowany ze względu na przedział czasowy odpowiadający każdemu miesiącowi oraz ze względu na wydatki, czyli pod względem wartości enuma CategoryType.COST odpowiadającemu przeciwieństwu wpływów w portfelu. Odpowiednio pogrupowane wartości są następnie wyświetlane na wykresie w formacie takim jak przedstawiono na ilustracji znajdującej się na samym początku opisywanego rozdziału.

* 1. Testy jednostkowe dostępne w aplikacji

Implementacja testów jednostkowych w każdym prezentowanym oprogramowaniu ma ważną wagę. Bez odpowiedniego ciągłego testowania aplikacji pod kątem nieoczekiwanych zmian w najbardziej trywialnych przypadkach użycia aplikacji może zakończyć się niespodziewanym oraz niepoprawnym zakończeniem działania podstawowej funkcji biznesowej.

@SpringBootTest

class UserIT {

@Autowired

private lateinit var userRepository: UserRepository

@Autowired

private lateinit var userService: UserService

@BeforeEach

fun remove() {

userRepository.deleteAll()

}

. . .

}

Fragment kodu źródłowego 10 Klasa bazowa testu integracyjnego klasy użytkowników; opracowanie własne

Powyższa implementacja klasy testów integracyjnych zawiera w sobie wszystkie pisane metody potrzebne do odpowiedniego przetestowania pod kątem poprawnego obrobienia danych przetrzymywanych w aplikacji. Podstawowe adnotacje pozwalające na podpięcie odpowiednich komponentów bezpośrednio w klasie aprobują możliwość wymiany danych w środowisku podobnym do występującego w etapach aplikacji występujących także w normalnym środowisku działania metod biznesowych zawartych w oprogramowaniu.

@Test

fun `should throw email exception while creating user`() {

//given

val request = USER

userService.registerUser(request)

//when

val result=assertThrows<EmailExistsException>{

userService.registerUser(request)

}

//then

assertThat(result.message).isEqualTo("Account with given email address exists:" + request.email)

}

Fragment kodu źródłowego 11 Przykładowy test integracyjny sprawdzający obsługę wyjątku aplikacji; opracowanie własne

Powyższa metoda jest wyłącznie przykładowym testem funkcjonalności udostępnianej przez omawiane oprogramowanie. Jako wynik środowisko programistyczne sprawdza informacje przepływające przez całą metodę i na końcu decydującą czy wynik pokrywa się z zamierzoną kalkulacją. Przedstawione metody często opisywane były przy pomocy techniki GWT[[29]](#footnote-29) pozwalającej na podzielenie przykładowego testu na wygodne i dobrze znane sekcje.

4.6. Ciągła integracja w Github Actions

Do utworzenia wyjątkowo przydatnego testowania aplikacji za każdym razem przy najmniejszej zmianie często używa się dodatkowych narzędzi kontrolujących przepływ informacji i odpowiednio je obrabiając. Przykładem jest utworzenie prostego pliku i następne dodanie go do projektu. Po odpowiedniej konfiguracji wybranego przez twórcę narzędzia, będzie ono automatycznie wykrywało pliki potrzebne do przeprowadzenia działania. Poniższy fragment pozwala na zbudowanie aplikacji i przetestowanie jej na podstawie zaimplementowanych testów. Pod koniec przeprowadzanej operacji w wybranym narzędziu zostaje informacja czy wszystkie etapy utworzone w pliku przeszły pomyślnie czy negatywnie.

jobs:

build:

steps:

- name: Set up Java Environment

uses: actions/setup-java@v2

with:

java-version: '11'

distribution: 'adopt'

- name: Validate Gradle wrapper

uses: gradle/wrapper

- name: Build application

run: |

cd ./server

./gradlew build

Fragment kodu źródłowego 12 Plik ciągłej integracji; opracowanie własne na podstawie artykułu Dmitry Shibanov https://github.com/actions/setup-java#caching-packages-dependencies

Podsumowanie i wnioski

W poniższym rozdziale przeprowadzono analizę pozwalającą na wykazanie niektórych wad i zalet powiązanych z zaimplementowanym już produktem. Jako rozwinięcie przedstawione zostały niektóre z wielu dostępnych kierunków rozwojowych dostępnych dla początkowej wersji aplikacji.

**Zalety i wady aplikacji**

Pomimo staranności w zaprojektowaniu prototypu aplikacji oraz późniejszej implementacji wymagań ciężko ominąć wszystkie narastające problemy podczas procesu tworzenia. Po zaimplementowaniu wszystkich opisów winno podsumować się przedstawione procesy. Z pewnością jako atrybuty występujące w aplikacji można wymienić między innymi:

* Utworzenia spójnego i prostego w obsłudze systemu monitorowania przychodów i wydatków finansowych.
* Atrakcyjny i czysty interfejs graficzny dedykowany użytkownikom końcowym aplikacji.
* Zaimplementowanie responsywnej aplikacji co umożliwia operowanie przez przeglądarkę internetową na większości urządzeń.
* Dobrze dobrane wykresy obrazujące stan finansowy.

Niektóre z niedoskonałości jakie udało się zobrazować dopiero po zakończeniu procesu tworzenia rozwiązania:

* Brak implementacji przydatnego logowania OAuth, przykładowo przy pomocy Google lub Facebook.
* Logowanie oraz rejestracja odbywają się bez udziału poczty mailowej.
* Niezwykle mało czasu zostało poświęconego na testowanie gotowego produktu przez przykładowego użytkownika systemu.

**Kierunki rozwojowe**

Wyjątkowo popularnym w ostatnich latach stał się handel jak i posiadanie kryptowalut. Charakteryzują się one przyzwoitym jak na swoje skoki cenowe przedziałem cenowym związanym z wymianą. Ludzie oraz światowe banki coraz częściej wymieniają swoje oszczędności i inwestują w rynek kryptowalut, dopóki duża ilość osób postrzega go jako ciekawe źródło alokowania pieniędzy należałoby się zastanowić czy nie jest to, aby na pewno dobry kierunek na rozwój aplikacji.

Wyjątkowo ciekawą właściwością jaką dało się zauważyć już przy analizie produktów rynkowych była możliwość podpięcia praktycznie wszystkich kont bankowych pod portfel finansowy w przypadku aplikacji „Kontomierz”. Owe rozwiązanie charakteryzuje się ciekawym kierunkiem rozwojowym ze względu na swoją wygodę z praktycznie automatycznym uzupełnianiem danych. Pomimo pozytywnego postrzegania tej opcji jako kierunek rozwojowy należy przemyśleć, czy aby na pewno takie rozwiązanie nie byłoby wyjątkowo irytujące szczególnie przy prowadzeniu paru portfeli. Integracja jaką udostępnia wytworzona aplikacja, na którymś etapie może weryfikować czy aby na pewno portfel powinien pobierać dane z konta bankowego i jeżeli to jakiej kategorii.

**Trudności podczas tworzenia**

ALKSJDLKASJDLKJASDJLKASJDLAKSJADSLKL

**Wnioski i zestawienia**

Podsumowując, produktem jaki udało się wytworzyć jest działający prototyp aplikacji do zarządzania finansami domowymi. Pozwala na operowanie aktywności zarobkowych poprzez bardzo intuicyjny interfejs graficzny. Wszystkie operacje pokazywane są w przyjazny dla użytkownika sposób na wbudowanych diagramach.

Zaraz po przeprowadzeniu implementacji wszystkich bazowych funkcjonalności zaczęto postrzegać wysoki poziom ekspansywności kolejnych rozwiązań możliwych do rozplanowania w przyszłych wersjach aplikacji.

Użyte technologie pozwalają użytkownikowi na praktycznie całkowitą dowolność w wyborze urządzenia używanego do operowania na modułach aplikacji. Przy dzisiejszych możliwościach łącza internetowego oraz jego powszechnej dostępności rozwiązanie wykazuje niemal całkowitą dowolność również w miejscu, z którego przyszli użytkownicy będą mieli możliwość na zarządzania operacjami na swoim mobilnym koncie klienckim. Dobrany moduł logowania oraz ciągłe połączenie z bazą danych przez warstwę serwerową aplikacji zapewnia użytkownikowi dostęp do finansów bez konieczności pamiętania o zbędnym transferze danych o operacjach pomiędzy urządzenia, z których korzysta odbiorca.

Bibliografia

1. Banks, A. i Eve, P. (2020). *Learning React.* O'Reilly Media.
2. Iwuć, M. (2008). *Jak zadbać o własne finanse?* Jachimczuk.
3. JetBrains. (2021). *Dokumentacja Kotlin*. https://kotlinlang.org/docs/home.html. dostęp 10.12.2021 r.
4. Loeliger, J. (2012). *Version Control with Git.* O'Reilly Media.
5. Masse, M. (2011). *REST API Design Rulebook.* O'Reilly Media.
6. Microsoft. (2021). *Dokumentacja React*. https://pl.reactjs.org/docs/getting-started.html. dostęp 10.12.2021 r.
7. Microsoft. (2021). *Dokumentacja TypeScript*. https://www.typescriptlang.org/docs/. dostęp 10.12.2021 r.
8. MUI. (2021). *Dokumentacja Material-Ui*. https://mui.com/getting-started/usage/. dostęp 10.12.2021 r.
9. Obe, R. i Hsu, L. (2017). *PostgreSQL: Up and Running, 3rd Edition.* O'Reilly Media.
10. Software, P. (2021). *Dokumentacja Spring Framework*. https://spring.io/learn. dostęp 10.12.2021 r.
11. Walls, C. (2020). *Spring in Action.* ISBN 9781617297571.

1. Profesor Elizabeth Ann Warren; Zasada oszczędzania 50/30/20; http://fiftythirtytwenty.com; dostęp 08.12.2021 r. [↑](#footnote-ref-1)
2. Marta Kozioł; Wady zasady oszczędzania 50/30/20; https://kantoronline.pl/blog/jak-oszczedzac-pieniadze-efektywnie-poznaj-ceniona-przez-ekspertow-zasade-50-30-20/; dostęp 08.12.2021 r. [↑](#footnote-ref-2)
3. Rafael D. Hernandez; Wzorzec MVC pozwalający na wymianę danych pomiędzy widokiem, logiką biznesową przy pomocy kontrolerów przepływu; https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/; dostęp 08.12.2021 r. [↑](#footnote-ref-3)
4. Strona główna 7csolutions; https://7csolutions.com/; dostęp 21.12.2021 r. [↑](#footnote-ref-4)
5. Strona główna Kontomierz; https://kontomierz.pl/; dostęp 21.12.2021 r. [↑](#footnote-ref-5)
6. Dan Arias; Wyjaśnienie hashowania przy pomocy szyfrowania bcrypt; https://auth0.com/blog/hashing-in-action-understanding-bcrypt; dostęp 22.12.2021 r. [↑](#footnote-ref-6)
7. Bolesław Nius; Sposób działania oraz czym jest JSON Web Tokens;https://global4net.com/ecommerce/co-to-jest-i-jak-dziala-json-web-tokens-jwt/; dostęp 22.12.2021 r. [↑](#footnote-ref-7)
8. Anna Buszydlik; Zagadnienia o języku adnotacji UML;https://www.jcommerce.pl/jpro/artykuly/podstawy-uml-czyli-modelowanie-dla-kazdego; dostęp 22.12.2021 r. [↑](#footnote-ref-8)
9. Zell Liew - https://www.zellwk.com; Potrzebne informacje na temat interfejsów REST’owych; https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/; dostęp 23.12.2021 r. [↑](#footnote-ref-9)
10. Thomas Bush; Najlepsze praktyki nazywania końcówek w REST Api; https://nordicapis.com/10-best-practices-for-naming-api-endpoints/ [↑](#footnote-ref-10)
11. JetBrains; Przyczyna powstania Kotlin/JS https://kotlinlang.org/docs/js-overview.html#use-cases-for-kotlin-js; dostęp 15.11.2021 [↑](#footnote-ref-11)
12. JetBrains; Charakterystyka Kotlina; www.kotlinlang.org/; dostęp 15.11.2021 [↑](#footnote-ref-12)
13. S. O'Dea, 29.06.2021; Statistica.com; Udział różnych platform w rynku mobilnym 2012-2021 www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/; dostęp 15.11.2021 [↑](#footnote-ref-13)
14. Chris Schaefer, Clarence Ho i Rob Harrop; Typy ziaren w Spring; Pro Spring - Apress [↑](#footnote-ref-14)
15. Jeff Linwood. Dave Minter; Adnotacje w Hibernate; Beginning Hibernate [↑](#footnote-ref-15)
16. Stefan Bechtold, Sam Brannen, Johannes Link, Matthias Merdes, Marc Philipp, Juliette de Rancourt, Christian Stein; Junit - dokumentacja; junit.org/junit5/docs/current/user-guide [↑](#footnote-ref-16)
17. Microsoft; Różnice w TypeScript https://www.typescriptlang.org/; dostęp 21.11.2021 [↑](#footnote-ref-17)
18. React; Dodanie TypeScript do aplikacji; https://create-react-app.dev/docs/adding-typescript/; dostęp 21.11.2021 [↑](#footnote-ref-18)
19. Solid IT; Ranking popularności baz danych; https://db-engines.com/en/ranking; dostęp 24.11.2021 [↑](#footnote-ref-19)
20. PostgreSQL; Strona główna PostgreSQL; https://www.postgresql.org/ dostęp 24.11.2021 [↑](#footnote-ref-20)
21. Philipp Hauer; Dlaczego nie warto używać baz plikowych do testów; https://phauer.com/2017/dont-use-in-memory-databases-tests-h2/ [↑](#footnote-ref-21)
22. [Sahiti Kappagantula](https://dzone.com/users/3288749/sahiti-k.html); Komendy używane w Git z przykładami; https://dzone.com/articles/top-20-git-commands-with-examples dostęp 29.11.2021 r. [↑](#footnote-ref-22)
23. Microsoft; Czym jest wirtualna maszyna; https://azure.microsoft.com/pl-pl/overview/what-is-a-virtual-machine/; dostęp 21.12.2021 r. [↑](#footnote-ref-23)
24. RedHat; Czym dokładnie jest ciągła integracja; https://www.redhat.com/en/topics/devops/what-is-ci-cd#continuous-integration [↑](#footnote-ref-24)
25. Spring Framework Dokumentacja; Obsługa walidacji danych w aplikacjach przy pomocy Spring Boot;hdocs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/validation.html [↑](#footnote-ref-25)
26. Les Hazlewood; Biblioteka JWT dla aplikacji Javy oraz Androida; https://github.com/jwtk/jjwt; dostęp 29.12.2021 r. [↑](#footnote-ref-26)
27. Axios; Biblioteka do komunikacji z klientami http przy pomocy node.js używanego w React; https://github.com/axios/axios [↑](#footnote-ref-27)
28. Recharts Group; Biblioteka służąca do tworzenia wykresów w aplikacjach React; https://recharts.org/ [↑](#footnote-ref-28)
29. Mateusz Starzyk; Wzorzec Given-When-Then w testach jednostkowych; https://blog.j-labs.pl/2017/02/Given-When-Then-pattern-in-unit-tests dostęp 21.12.2021 r. [↑](#footnote-ref-29)