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

**Spis treści**

[Wstęp 4](#_Toc89810127)

[1. Zarządzanie budżetem domowym 5](#_Toc89810128)

[1.1. Cel projektu 5](#_Toc89810129)

[1.2. Środki użyte przy pracy 5](#_Toc89810130)

[2. Koncepcje oraz funkcjonalności systemowe 6](#_Toc89810131)

[3. Rozwiązania technologiczne 7](#_Toc89810132)

[3.1. Backend 7](#_Toc89810133)

[3.1.1. Kotlin 7](#_Toc89810134)

[3.1.2. Spring Boot 9](#_Toc89810135)

[3.1.3. Hibernate / Spring Data JPA 11](#_Toc89810136)

[3.1.4. Spring Security 13](#_Toc89810137)

[3.1.5. JUnit / Spring Test 13](#_Toc89810138)

[3.2. Frontend 15](#_Toc89810139)

[3.2.1. TypeScript 15](#_Toc89810140)

[3.2.2. React 16](#_Toc89810141)

[3.2.3. Material-UI 18](#_Toc89810142)

[3.3. Baza danych 19](#_Toc89810143)

[3.3.1. PostgreSQL 21](#_Toc89810144)

[3.3.2. H2 22](#_Toc89810145)

[3.4. Narzędzia programistyczne 23](#_Toc89810146)

[3.4.1. IntelliJ IDEA 23](#_Toc89810147)

[3.4.2. VisualStudio Code 24](#_Toc89810148)

[3.4.3. GitHub 24](#_Toc89810149)

[3.5. Pozostałe rozwiązania technologiczne 24](#_Toc89810150)

[3.5.1. Git 24](#_Toc89810151)

[3.5.2. Docker 26](#_Toc89810152)

[3.5.3. Gradle 26](#_Toc89810153)

[3.5.4. GitHub Actions 27](#_Toc89810154)

[4. Zamysł architektury systemu 28](#_Toc89810155)

[5. Implementacja 29](#_Toc89810156)

[5.1. Moduł rejestracji i logowania 29](#_Toc89810157)

[6. Podsumowanie 30](#_Toc89810158)

[6.1. Zalety i wady aplikacji 30](#_Toc89810159)

[6.2. Kierunki rozwojowe 30](#_Toc89810160)

[6.3. Wnioski i zestawienia 30](#_Toc89810161)

[Bibliografia 31](#_Toc89810162)

Wstęp

Przedmiotem

1. Zarządzanie budżetem domowym

Zarządzanie pieniędzmi w zakresie budżetu domowego, rządzi się wieloma zasadami, z których jako 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 czasu, jest 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 całe miesięczne wydatki. 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 przykładowy sposób 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 utworzeniu 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 oraz funkcjonalności systemowe

Kryterium podstawowym jest utworzenie aplikacji internetowej przy pomocy dostępnych narzędzi w sposób rozdzielenia jej na dwie aplikacje, jedną serwerową pozwalającą na wymianę danych z bazą oraz przetwarzającą zapytania i drugą reprezentująca szatę graficzną dla użytkownika. W aktualnym rozdziale opisywane są wymagania systemowe, na podstawie których należy zaimplementować system. W dalszej części rozdziału zauważyć będzie można podsumowanie planowania w aplikacji. W końcówce rozdziału dostępne będą diagramy implementowanych w następnych rozdziałach funkcjonalności.

* 1. Podobne rozwiązania dostępne na rynku
  2. Wymagania systemowe
  3. Szablony diagramów modelujących

1. Rozwiązania technologiczne

Jako temat 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[[4]](#footnote-4).

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[[5]](#footnote-5):

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[[6]](#footnote-6).

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 1 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[[7]](#footnote-7):

* *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 wydaję 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 „@”[[8]](#footnote-8):

* *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[[9]](#footnote-9):

* *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 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, czyli firma 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[[10]](#footnote-10):

* 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 pracującymi tam deweloperami. Miała 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 2 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. 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[[11]](#footnote-11):

* .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[[12]](#footnote-12) 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[[13]](#footnote-13). 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 nierzadko 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[[14]](#footnote-14). 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 oszczędzające duże ilości czasu w wykonywaniu zawodu. 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.

* 1. 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 3 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[[15]](#footnote-15):

* 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, 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 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.

1. Implementacja
   1. Moduł rejestracji i logowania
2. Podsumowanie
   1. Zalety i wady aplikacji
   2. Kierunki rozwojowe
   3. Wnioski i zestawienia

Bibliografia

1. Profesor Elizabeth Ann Warren; Zasada oszczędzania 50/30/20; http://fiftythirtytwenty.com; dostęp 08.12.2021r. [↑](#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. JetBrains; Przyczyna powstania Kotlin/JS https://kotlinlang.org/docs/js-overview.html#use-cases-for-kotlin-js; dostęp 15.11.2021 [↑](#footnote-ref-4)
5. JetBrains; Charakterystyka Kotlina; www.kotlinlang.org/; dostęp 15.11.2021 [↑](#footnote-ref-5)
6. 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-6)
7. Chris Schaefer, Clarence Ho i Rob Harrop; Typy ziaren w Spring; Pro Spring - Apress [↑](#footnote-ref-7)
8. Jeff Linwood. Dave Minter; Adnotacje w Hibernate; Beginning Hibernate [↑](#footnote-ref-8)
9. 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-9)
10. Microsoft; Różnice w TypeScript https://www.typescriptlang.org/; dostęp 21.11.2021 [↑](#footnote-ref-10)
11. React; Dodanie TypeScript do aplikacji; https://create-react-app.dev/docs/adding-typescript/; dostęp 21.11.2021 [↑](#footnote-ref-11)
12. Solid IT; Ranking popularności baz danych; https://db-engines.com/en/ranking; dostęp 24.11.2021 [↑](#footnote-ref-12)
13. PostgreSQL; Strona główna PostgreSQL; https://www.postgresql.org/ dostęp 24.11.2021 [↑](#footnote-ref-13)
14. Philipp Hauer; Dlaczego nie warto używać baz plikowych do testów; https://phauer.com/2017/dont-use-in-memory-databases-tests-h2/ [↑](#footnote-ref-14)
15. [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-15)