



Politechnika Łódzka

Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

Aplikacja webowa do organizacji wydarzeń kulturalnych i dystrybucji biletów

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

Promotor: dr inż. Marcin Kacprowicz

Dyplomant: Maciej Pracucik

Nr albumu: 216869

Kierunek: Informatyka Stosowana

Specjalność: Technologie gier i symulacji komputerowych

Łódź, 2020/2021



Instytut Informatyki

90-924 Łódź, ul. Wólczańska 215, **budynek B9**
tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

Spis treści

Spis treści	2
1 Wprowadzenie	5
1.1 Problematyka	5
1.2 Cel i założenia pracy	6
1.3 Struktura pracy inżynierskiej	6
2 Zakupy internetowe, analiza konkurencji i techniczne aspekty aplikacji	6
2.1 Zakupy internetowe	6
2.2 Analiza najpopularniejszych internetowych metod płatności	7
2.3 Analiza istniejących portali do promowania i dystrybucji biletów na wydarzenia kulturowe	8
2.4 Bazy danych jako środek przechowywania danych	9
2.5 Nierelacyjne bazy danych	10
2.6 Techniki tworzenia aplikacji webowych	12
2.7 REST API	13
2.8 Projektowanie i tworzenie systemu informatycznego	14
3 Narzędzia i technologie wybrane do realizacji projektu	15
3.1 Node.js	15
3.2 React.js	19
3.3 TypeScript	20
3.4 Najistotniejsze wykorzystane biblioteki	21
3.5 MongoDB	23
3.6 JWT - JSON Web Token	24
4 Proces tworzenia aplikacji	25
5 Aplikacja od strony użytkownika	31
6 Podsumowanie	36
7 Możliwości dalszego rozwoju aplikacji	36
Bibliografia	38

Spis rysunków	40
Spis tabel	41
Listingi	42

Abstract: The aim of this thesis is to design and implement web application for promoting cultural events and selling tickets for them. There are plenty of similar applications, but this one was created with aim to simplify and shorten to maximum, user's experience and time spent with the app. User is presented with simple, yet very clear, interface and is provided with secure and effortless payments.

This work compares apps from the competition, pointing out the best aspects of each, in order to show what features and functionalities were taken into consideration while designing the app. It provides information with current online shopping trends, as well as with the most popular online payment methods.

The result of the analysis and designing is web application called ConcertsApp.

Streszczenie: Celem niniejszej pracy dyplomowej jest zaprojektowanie oraz zaimplementowanie aplikacji webowej do promocji wydarzeń kulturowych oraz sprzedaży biletów na nie. Na rynku jest wiele podobnych aplikacji, jednakże ta została stworzona, z myślą by uprościć i zminimalizować do maksimum, czas i doświadczenie z obcowaniem z aplikacją. Użytkownik ma do czynienia, z prostym, jednakże bardzo czytelnym interfejsem aplikacji oraz z bezpiecznymi i niewymagającymi dużego zaangażowania płatnościami.

Niniejsza praca porównuje aplikacje konkurencji i przedstawia ich najlepsze aspekty, w celu pokazania, które ich cechy i funkcjonalności były brane pod uwagę podczas tworzenia aplikacji. Również zawiera informację o trendach jeśli chodzi o zakupy przez internet, jak również najpopularniejsze metody płatności w sieci.

Rezultatem wyżej wymienionej analizy oraz projektowania jest aplikacja webowa o nazwie ConcertsApp.

1 Wprowadzenie

W dobie trwającego rozwoju technologicznego, każdy aspekt życia codziennego jest usprawniany i przenoszony do internetu. Nieinaczej jest z promowaniem i sprzedażą biletów na wydarzenie kulturowe pokroju: spektakli w teatrze, koncertów, festiwali. Aplikacja ConcertsApp ma służyć dokładnie temu, ma zminimalizować czas, jaki należałoby kiedyś przeznaczyć na zdobycie biletów. Czy też na uzyskanie informacji o wydarzeniach z interesującej konsumenta dziedziny, lub odbywających się w pobliżu.

1.1 Problematyka

Czas pędzi nieustannie, w obecnych czasach wydawać by się mogło, że nawet pędzi niczym TGV. Ani człowiek się obejrzy i z poniedziałku staje się piątek. Ludzie są wiecznie zabiegani, nie mają czasu, dlatego firmy, twórcy programów i aplikacji, cały czas wymyślają coraz to nowe sposoby, jak uprościć i skrócić czas, poświęcany na codzienne czynności. Dlatego też wiele usług, sklepów, serwisów przenoszonych jest w całości do sieci, lub mają sieciowy odpowiednik sklepu/oddziału stacjonarnego. W obecnych czasach zdecydowanie ciężiej jest prosperować firmom nie posiadającym strony lub sklepu internetowego. Na rynku jest zbyt duża konkurencja, aby móc sobie pozwolić na brak obecności w sieci i/lub mediach społecznościowych. Poszukując, przykładowo, krawca w okolicy, wpisuje się odpowiednią frazę w wyszukiwarkę internetową. Jeśli strona nie pojawi się w wynikach wyszukiwania, lub przynajmniej informacja o firmie i jej oceny na Google Maps, to nie ma możliwości, żeby dana osoba skorzystała z usług krawca, który mimo posiadanego fachu, nie widnieje w sieci. Każda osoba/firma chcąc podbić rynek i osiągnąć sukces musi posiadać minimum stronę na jednym z portali społecznościowych, np. FaceBook, oraz widnieć w Google Maps.

Minimalizacja czasu poświęcanego na codzienne czynności dotyczy również, strefy takiej jak kultura. Chcąc wybrać się np. na spektakl w teatrze, są dwie możliwości: kupno biletu bezpośrednio w teatrze oraz zakup biletu przez stronę teatru lub stronę udostępniającą bilety. Pierwszy z nich wymaga przebicia się przez zakorkowane miasto, możliwe, że nawet stania w kolejce po bilet i dopiero powrót do domu. Potrafi to pochłonąć ogromną ilość czasu, na co nie każdy może sobie pozwolić. Dlatego powstała alternatywa, czyli zakup biletu w internecie, zakup jest bez kolejek, bez wychodzenia z domu i cały proces może potrwać góra 10 minut. W usprawnienie tej części rynku celuje powstała na potrzeby niniejszej pracy inżynierskiej aplikacja, aby przyspieszyć i ułatwić dostęp do jakże ważnej części ludzkiego życia, jaką jest kultura

i sztuka.

1.2 Cel i założenia pracy

Celem niniejszej pracy dyplomowej jest stworzenie aplikacji webowej, która umożliwi w łatwy i szybki sposób na promowanie wydarzeń kulturowych i zakup biletów na nie. Zakup odbywać się będzie za pomocą płatności online, jedynie podając dane karty kredytowej lub debetowej.

Zakres pracy obejmuje zaprojektowanie i implementację aplikacji klienckiej i serwerowej, przy wykorzystaniu, najnowszych i bardzo popularnych na rynku pracy technologii. Do strony klienckiej został wykorzystany React, z kolei do stworzenia serwera użyto Node.JS.

1.3 Struktura pracy inżynierskiej

Pierwszy rozdział ma posłużyć jako wprowadzenie do problemu podjętego w niniejszej pracy dyplomowej. Następny przedstawia aspekty takie jak: konkurencyjne rozwiązania do tworzonej aplikacji, analiza najpopularniejszych metod płatności online, opowiada o bazach danych oraz przybliża metody tworzenia aplikacji webowych. Trzeci rozdział przedstawia wykorzystane technologie do implementacji aplikacji oraz uzasadnia dlaczego akurat ona została wybrana, a nie inna. Czwarty rozdział pokazuje jak przebiegał proces twórczy, czyli projektowanie i implementacja. Piąty rozdział mówi o tym, jak wygląda proces obcowania z aplikacją, od strony klienta.

2 Zakupy internetowe, analiza konkurencji i techniczne aspekty aplikacji

2.1 Zakupy internetowe

Sklepy internetowe z roku na rok rosną w siłę, przybywa ich liczba w zatrważającym tempie. Aż 73% ankietowanych w raporcie przygotowanym przez Gemius Polska[15] deklaruje kupowanie online i ta forma zakupów cieszy się niezmiennie dobrym wizerunkiem wśród kupujących. Brak sklepu internetowego stanowi ogromną niekorzyść dla obecnych przedsiębiorców. Zakupy online charakteryzują się dużo większym wyborem produktów, łatwością w porównywaniu ofert, czy też łatwością w znalezieniu interesujących produktów. Lecz, to nie z wyżej wymienionych

powodów, sklepy w sieci cieszą się taką popularnością. Wśród najczęściej wymienianych powodów są: dostępność przez całą dobę (aż 82% wybrało ten powód), brak konieczności wyjazdu do sklepu - 78%, nieograniczony czas wyboru - 72% oraz atrakcyjniejsze ceny niż w sklepach tradycyjnych - 71%.

Niestety taka forma zakupów ma też swoje słabe strony, najpopularniejszymi wymienianymi problemami są: wysokie koszty dostawy, długi czas oczekiwania na dostawę, irytujące reklamy produktów, wcześniej poszukiwanych. Również bardzo często wymienianą przeciwnością jest uszkodzona przesyłka w transporcie, wynikać to może ze źle zapakowanej i zabezpieczonej paczki lub z winy firmy kurierskiej i jej pracowników.

Co jednak sprawia, że klienci decydują się na wybór danego portalu na zakupy?

Najczęściej wybór sklepu pada dzięki kodom rabatowym, dopiero w dalszej kolejności, klienci patrzą na aspekty takie jak: dokładne informacje o warunkach zamówienia, ostępne na stronie dane firmy czy przejrzysta i funkcjonalna strona internetowa. Wydawać by się mogło, że to na te kolejne cechy portali, powinno się w pierwszej kolejności zwracać uwagę, bo to dzięki nim w od razu można stwierdzić czy strona może być oszutwem, czy też nie.

2.2 Analiza najpopularniejszych internetowych metod płatności

Najpopularniejszymi metodami płatności w internecie, według raportu "E-commerce w Polsce 2020" [15] są kolejno:

- szybki przelew przez serwis płatności np. payU, przelewy24
- przelew tradycyjny
- płatność kartą płatniczą przy składaniu zamówienia
- płatności mobilne np. BLIK

Nie zostały wymienione płatności pokroju wysyłki za pobraniem, płatności w sklepie przy odbiorze, ponieważ nie są to płatności realizowane online, a tych dotyczy analiza przedstawiona w niniejszym rozdziale.

Zdecydowanie nie powinna dziwić obecność szybkich przelewów na pierwszym miejscu tego rankingu. Aż 70% ankietowanych odpowiedziało, że choć raz korzystało z tej metody, przy robieniu zakupów przez internet. Cechują się błyskawicznym czasem realizacji, w przeciwieństwie do tradycyjnych przelewów. Zaledwie 46% ankietowanych zdecydowało się choć raz na przelew tradycyjny. Różnica jest znaczna, jednakże nie powinna ona dziwić, ponieważ to oszczędność

czasu sprawia, że klienci decydują się na zakupy online w pierwszej kolejności. Przelewy tradycyjne dodatkowo wydłużają czas realizacji zamówienia, ponieważ ich przetwarzanie odbywa się jedynie w dni robocze, o wyznaczonych godzinach, różnych, w zależności od banku.

Na płatność kartą decyduje się 40% pytanym, jest to dość zaskakujące, zważywszy na fakt, że jest to zdecydowanie jedna z najszybszych metod płatności. Do jej realizacji niezbędne jest jedynie numer karty, data wygaśnięcia oraz kod CVV. Powodem na dość niską popularność tej metody, mogą być dwie rzeczy, strach przed podawaniem danych karty, żeby nie zostały skradzione, lub brak karty podczas składania zamówienia. Druga z nich wydaje się być bardziej prawdopodobna, mało kto kupując produkty przez internet ma akurat przy sobie kartę płatniczą, by odczytać z niej niezbędne liczby. Zdecydowanie łatwiej jest zalogować się do banku i wykonać przelew, czy to tradycyjny, czy szybki.

Dziwić może obecność płatności mobilnych, na ostatnim miejscu, z zaledwie 35% wykorzystania. Jest to sposób bardzo wygodny, zważywszy na fakt iż mało kto nie posiada przy sobie telefonu. Potrzebna jest jeszcze tylko aplikacja banku i można dokonywać dowolnych płatności. W przypadku BLIK-u generowany jest sześciocyfrowy ciąg liczb, który wystarczy wpisać w odpowiednie pole, zatwierdzić płatność w aplikacji i gotowe.

2.3 Analiza istniejących portali do promowania i dystrybucji biletów na wydarzenia kulturowe

Najpopularniejszymi portalami zbliżonymi do tworzonej aplikacji są zdecydowanie:

- Eventim
- GoingApp

Eventim jest znacznie starszym portalem, co można stwierdzić chociażby, po jego szacie graficznej, łatwo to zauważyć porównując do wyżej wymienionego GoingApp. W swojej ofercie ma wydarzenia pokroju: koncertów, przedstawień teatralnych, oper czy baletów.

Same bilety sprzedawane są w formie papierowej, przychodzą na maila, lub istnieje możliwość zakupu biletu mobilnego. Kupno może odbyć się zarówno poprzez stronę internetową, jak również poprzez aplikację mobilną. Płatności można dokonać za pomocą przelewu tradycyjnego, karty kredytowej, szybkiego przelewu Dotpay

GoingApp charakteryzuje się bardziej nowoczesną i przejrzystą szatą graficzną w porównaniu do Eventim. Oferty obu portali są bardzo do siebie zbliżone, lecz tutaj można znaleźć takie

wydarzenia, jak chociażby imprezy związane z filmem czy jedzeniem. Jednakże nie posiada on biletów na balet lub opery.

Wejściówki można, podobnie jak w Eventim nabyć poprzez ich stronę internetową lub aplikację mobilną. Sam bilet jest dostępny w formie dokumentu PDF, lub kodu QR, dostęp do niego mamy zarówno poprzez maila, którego otrzymujemy zaraz po zakupie, oraz aplikację. Użytkownik może dokonać płatności za pomocą systemów płatniczych takich jak: payU, eCard, MasterPass oraz Paymento®

Aplikacja webowa, tworzona na potrzeby niniejszej pracy inżynierskiej, czerpie z obu portali najlepsze cechy. Z Eventim czerpie różnorodność wydarzeń, z kolei z GoingApp przejrzystość interfejsu i bilety w formie, bardzo popularnego kodu QR. Metody płatności zostają ograniczone do kart płatniczych: debetowej i kredytowej. Podawane są numer karty, data ważności oraz kod CVV.

2.4 Bazy danych jako środek przechowywania danych

”Baza danych to zorganizowany zbiór ustrukturyzowanych informacji, czyli danych, zwykle przechowywany w systemie komputerowym w formie elektronicznej. Bazą danych steruje zwykle system zarządzania bazami danych (DBMS). Dane i system DBMS oraz powiązane z nimi aplikacje razem tworzą system bazodanowy, często nazywany w skrócie bazą danych.”. [16] Innymi słowy jest to kontener na dane, o dowolnej postaci, mogą to być liczby, ciągi znaków, a nawet zdjęcia czy filmy. Dane te nie są, najczęściej, przetrzymywane lokalnie na komputerach, tylko na serwerach czy w chmurze.

Dlaczego więc korzysta się z baz danych a nie np. z arkuszy kalkulacyjnych?

Odpowiedź jest bardzo prosta, arkusze kalkulacyjne nie zostały stworzone do pracy z ogromną ilością danych, przy jednoczesnym dostępie, nawet kilkuset lub więcej użytkowników. Są wręcz idealne do pracy z mniejszą ilością danych dla jednego lub małej grupy użytkowników, którzy nie potrzebują wielu skomplikowanych funkcji do manipulacji danymi. Bazy danych z kolei, przeznaczone są do pracy z ogromnymi ilościami informacji, umożliwiając dodatkowo jednoczesny dostęp do nich, wielu użytkownikom na raz. Co więcej praca z bazami danych, charakteryzuje się wysokim bezpieczeństwem i szybkością wykonywanych operacji. Operacje na danych, tworzenie zapytań odbywa się za pomocą logiki i języka o wysokim stopniu złożoności. Jako przykład może posłużyć, zdecydowanie najpopularniejszy z nich, czyli język zapytań SQL. Bazy danych dzielimy, między innymi na:

- relacyjne
- hurtownie danych
- NoSQL
- chmurowe

Relacyjne bazy danych zyskały ogromną popularność w latach 80[16]. Dane zorganizowane są w tabelach, składające się z wierszy i kolumn. Po dziś dzień stanowią jedne z najpopularniejszych baz danych, jak nie najpopularniejsze, dostępne na rynku. Przykładami relacyjnych baz danych są: MySQL i Microsoft SQL Server.

Hurtownie danych, inaczej centralne repozytorium danych, to typ bazy danych wykorzystywany do wykonywania zapytań i analizy. Ma on umożliwić i wspierać działania z zakresu analizy biznesowej, w szczególności analityki. Często operuje na danych historycznych, pochodzących z wielu źródeł. Jej umiejętności analityczne pozwalają przedsiębiorstwom przetrzymywać cenne dane biznesowe, które ułatwiają podejmowanie decyzji.[17]

Baza danych NoSQL, inaczej nierelacyjna, od relacyjnych baz różni je przede wszystkim to, że w relacyjnych mają jasną strukturę organizacji danych. W nierelacyjnych dane, pochodzące z tej samej kolekcji, mogą posiadać kompletnie różne atrybuty. Szerzej na temat baz NoSQL zostanie omówione w następnym podrozdziale.

Ostatnim typem bardzo popularnym na rynku baz danych są bazy chmurowe. Bazy te charakteryzują się przede wszystkim tym, że dane przetrzymywane są na prywatnej, publicznej lub hybrydowej platformie przetwarzania danych w chmurze. Najpopularniejszymi bazami w chmurze są Microsoft Azure SQL Database, Oracle Database, Google Cloud SQL oraz Amazon Relational Database Service.

2.5 Nierelacyjne bazy danych

”Bazy danych NoSQL są zamiennie nazywane „nierelacyjnymi” lub „nie SQL”, aby podkreślić fakt, że potrafią obsłużyć ogromne ilości szybko zmieniających się, nieustrukturyzowanych danych innymi sposobami niż relacyjna (SQL) baza danych z wierszami i tabelami.” [8]

Co jednak wyróżnia bazy typu NoSQL na tle relacyjnych baz danych?

Ich głównym wyróżnikiem zupełnie inny schemat przechowywania danych. W bazach nierelacyjnych dane nie są przechowywane w tabelach, mogą być dowolnie skalowane, każdy wiersz może zawierać różne kolumny (atrybuty opisujące dany obiekt), nie jest wymuszana relacja

między obiektami.

Bazy typu SQL są znacząco lepsze jeśli skalowalność zachodzi wertykalnie, a horyzontalnie atrybuty są jasno określone. Nie jest wskazane aby każdy obiekt, z jednej tabeli, opisany był innymi atrybutami.

W przypadku prostych obiektów, opisanych małą ilością atrybutów, lepszym wyborem będą bazy typu NoSQL. Z kolei dla bardziej skomplikowanych encji, lepiej wykorzystać bazy relacyjne, ze względu na ich schludność i uporządkowanie, zdecydowanie łatwiej uniknąć niechcianego bałaganu i błędów.

Bazy NoSQL dzielimy, ze względu na typ na[3]:

- Klucz - wartość
- Dokumentowe
- Grafowe
- Kolumnowe

Pierwsze z nich opierają się na kolekcji słowników, w których z kluczem powiązane są wartości różnych atrybutów encji. Dodatkowo wykorzystuje się funkcje haszujące do przyspieszenia odczytu. Jako przykłady mogą posłużyć: Windows Azure Table Storage oraz Amazon SimpleDB. Bazy dokumentowe stosuje się do przechowywania dokumentów posiadających różne atrybuty oraz mają możliwość zagnieżdżania jednych dokumentów w drugie. Przykładem wymienionego typu baz jest wykorzystane przy implementacji projektu baza MongoDB.

Bazy grafowe oparte są na grafach i algorytmach grafowych. Każdy obiekt jest innym węzłem w grafie, a relacje między nimi to krawędzie. Przykłady to: Titan, Giraph. Ostatnim typem baz nierelacyjnych są bazy kolumnowe. Oparte są na architekturze hybrydowej, wykorzystują techniki i podejści relacyjnej bazy oraz bazy klucz-wartość do przechowywania schematów danych. Przykładem takiej bazy jest Cassandra.

Podsumowując bazy nierelacyjne lepiej sprawdzają się przy małej ilości atrybutów, łatwiej uniknąć bałaganu, dlatego też to właśnie nierelacyjna baza danych została wybrana do realizacji projektu, a dokładniej MongoDB. O tym dlaczego akurat MongoDB, zostanie przedstawione w następnym rozdziale 3.5.

Factor	Traditional Web App	Single-Page Application
Required Team Familiarity with JavaScript/TypeScript	Minimal	Required
Support Browsers without Scripting	Supported	Not Supported
Minimal Client-Side Application Behavior	Well-Suited	Overkill
Rich, Complex User Interface Requirements	Limited	Well-Suited

Tabela 1: Tabela decyzyjna wyboru pomiędzy SPA, a MPA. [9]

2.6 Techniki tworzenia aplikacji webowych

Przy tworzeniu aplikacji webowych istnieją dwa podejścia, mówiące o tym jak należy tworzyć aplikację. Są to Single-Page Application (SPA) oraz Multiple-Page Application (MPA, czyli tradycyjne strony internetowe).

SPA jest aplikacją webową lub stroną internetową, która nie przeładowuje swoich stron za pomocą serwera oraz interakcja z użytkownikiem zachodzi, za pomocą dynamicznego zmieniania aktualnie wyświetlanej strony. Kod źródłowy strony jest zapisywany tylko przy pierwszym załadowaniu, oraz dodatkowe zasoby są ładowane, tylko wtedy kiedy jest to wymagane, w zależności od zachowania użytkownika. SPA są interaktywne oraz przyjazne użytkownikowi, są bardziej responsywne niż tradycyjne strony, ponieważ ładują się tylko raz i ich komunikacja z serwerem, jest ograniczona do minimum. [1]

MPA jest klasycznym podejściem do tworzenia stron internetowych. Praktycznie każde kliknięcie, w dowolną rzecz na stronie, wysyła zapytanie do serwera o wyrenderowanie nowej strony w przeglądarce. Każda strona jest innym plikiem, nie ma możliwości jak w SPA, że renderowany jest tylko wymagany komponent. Cała strona musi być ponownie rerenderowana.

Kiedy należy używać jakiego podejścia?

Microsoft w poradniku do tworzenia aplikacji w .NET, proponuje poniższą tabelkę decyzyjną[9]:

Do stworzenia aplikacji na potrzeby pracy inżynierskiej zostało wybrane podejście SPA, gwarantuje ono szybsze, przyjemniejsze oraz bardziej responsywne zachowanie aplikacji dla użytkownika.

2.7 REST API

REST oznacza w skrócie Representational State Transfer, jest to architektura zaproponowana przez Roya Fieldinga, jako nowe podejście do projektowania usług internetowych. Architektura ta jest niezależna od wszelkich podstawowych protokołów, w tym HTTP. Jednak w najbardziej typowych implementacjach REST protokół HTTP pełni funkcję protokołu aplikacji.[10] Architektura REST oparta jest na kilku głównych zasadach, oto 6 najważniejszych[10]:

- Interfejsy API są oparte na zasobach - dowolnym obiekcie, danych lub usłudze, które są dostępne dla klienta
- Zasób ma identyfikator URI służący do unikatowej identyfikacji tego zasobu
- Interakcja z usługą odbywa się poprzez wymianę reprezentacji zasobów. Najpopularniejszym formatem wymiany danych jest JSON.
- Do wykonywania operacji na zasobach używa się standardowych zapytań HTTP, najczęściej używane operacje to GET, POST, PUT, PATCH i DELETE.
- Interfejsy API REST korzystają z bezstanowego modelu żądań. Każde zapytanie do serwera musi posiadać niezbędne informacje do zrozumienia zapytania. Stan sesji jest przechowywany tylko i wyłącznie po stronie klienta.
- Interfejsy API REST są sterowane za pomocą hipermedialnych linków, zawartych w reprezentacji.

Najpopularniejszymi metodami do operacji na zasobach, jak zostało wyżej wymienione są:

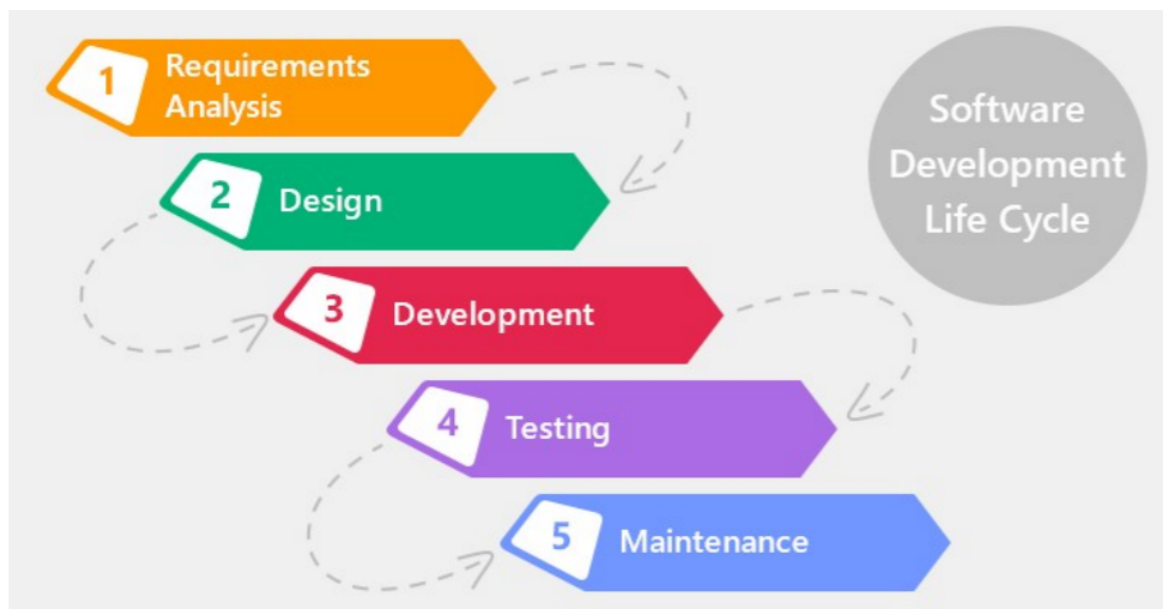
- GET - pobiera reprezentację danego zasobu np. strona potrzebuje listę wszystkich produktów dostępnych do zakupu na stronie
- POST - tworzy nowy zasób, np. użytkownik rejestruje się na danej stronie internetowej i przy kliknięciu "ZAREJESTRUJ" wysyłane jest zapytanie POST, o dodaniu nowego użytkownika do bazy danych.
- PUT - tworzy zasób lub aktualizuje istniejący
- PATCH - wykonuje częściową aktualizację zasobu, np. użytkownik zmienia adres zamieszkania, podany wcześniej
- DELETE - usuwa zasób

W zależności od podanego URI, może zostać zwrócony zasób w formie kolekcji lub pojedynczego elementu. Przykładowo podając URI `https://adventure-works.com/orders`, dla metody GET uzyskamy listę wszystkich zamówień. Z kolei podając URI `https://adventure-works.com/orders/1` uzyskamy pojedyncze zamówienie, np. o `id = 1`. Każda metoda będzie zachowywać się podobnie dla podanych wyżej URI, z tą różnicą, że przykładowo metoda POST zamiast pobrać listę wszystkich zamówień, utworzy nowe zamówienie, metoda PUT zaktualizuje wszystkie zamówienia, w zadany sposób.

Architektura REST API cieszy się ogromną popularnością, między innymi z tego powodu została wybrana do stworzenia aplikacji.

2.8 Projektowanie i tworzenie systemu informatycznego

W inżynierii oprogramowania, czyli inaczej wiedzy technicznej, opisującej wszystkie fazy cyklu życia oprogramowania, istnieje wiele różnych modeli cyklu życia oprogramowania. Do najpopularniejszych należą: kaskadowy, piramidy oraz spiralny. Każdy z nich w podobny sposób przedstawia każdy krok, jaki powinien przejść program, aby mógł zostać wypuszczony na rynek. Na poniższym rysunku przedstawiono jak wygląda model kaskadowy, w dalszej części zostaną omówione, pokrótce, jego poszczególne etapy.



Rysunek 1: Model kaskadowy [7]

Każda z faz cechuje się innym zestawem czynności jakie należy wykonać. Faza określenia wymagań cechuje się, jak sama nazwa wskazuje, określeniem wymagań. Ustalane są cele, a

następnie zamieniane są na faktyczne wymagania jakie musi posiadać oprogramowanie. Wymagania dzielimy odpowiednio na funkcjonalne i нефункционалне. Wymagania funkcjonalne to takie, które opisują funkcję lub czynności wykonywane przez system. Z kolei нефункционалне opisują ograniczenia, przy zachowaniu których system powinien realizować swoje funkcje.

W fazie projektowania odpowiada się na pytania, jak system ma działać i jak system ma być zaimplementowany. W wyniku czego powstaje projekt i projekt implementacji.

Kolejno przechodzi się do fazy implementacji, jest to moment, w którym wszystkie czynności związane z projektowaniem zostają zakończone i wcielone w faktyczny program.

Po fazie implementacji następuje faza testowania. W tej fazie następuje sprawdzenie systemu, czy jest zgodny z postawionymi wymaganiami, oraz czy nie posiada jakichś błędów mogących doprowadzić do niepożądanego działania lub błędów z samym wykonywaniem się oprogramowania.

Na samym końcu cyklu życia oprogramowania, znajduje się część przeznaczona na konserwację programu. W tej fazie poprawiana jest jakość produktu, dostosowanie oprogramowania do zmian zachodzących w środowisku pracy oraz usuwanie wcześniej niewykrytych błędów.

Warto również wspomnieć o fazie dokumentacji, jest ona wykonywana równolegle z praktycznie wszystkimi pozostałymi czynnościami. Tworzona jest dokumentacja, w której znajdują się takie elementy jak: podręcznik użytkownika, opis instalacji czy podręcznik administratora. Dokumentacja jest integralną częścią projektu i nie powinna być pomijana lub traktowana bez należytej uwagi.

3 Narzędzia i technologie wybrane do realizacji projektu

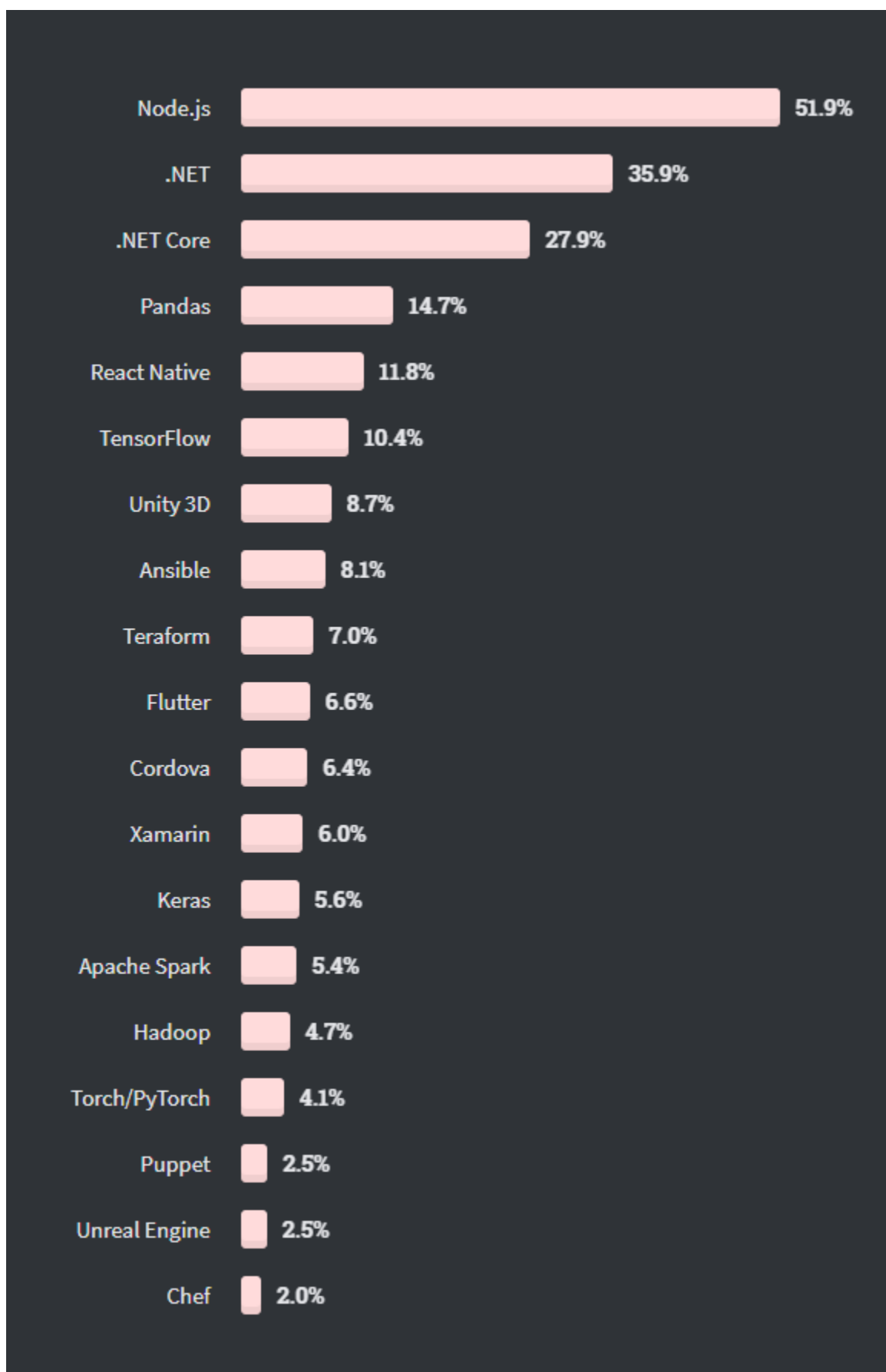
3.1 Node.js

”Node.JS jest wieloplatformowym oprogramowaniem o otwartym kodzie, które pozwala deweloperom na tworzenie wszelkiego rodzaju oprogramowania w języku JavaScript pracującym po stronie serwera. Jest to środowisko uruchomieniowe, które działa poza przeglądarką, współpracując bezpośrednio z systemem operacyjnym. W ten sposób środowisko Node udostępnia swoim aplikacjom API systemu operacyjnego, w tym dostęp do systemu plików, bibliotek systemowych czy uruchomionych procesów, w tym serwerów HTTP.”[13] Warty wspomnienia jest fakt, że jest oparty na silniku JavaScript Chrome V8.

Zalety jakie niesie ze sobą Node:

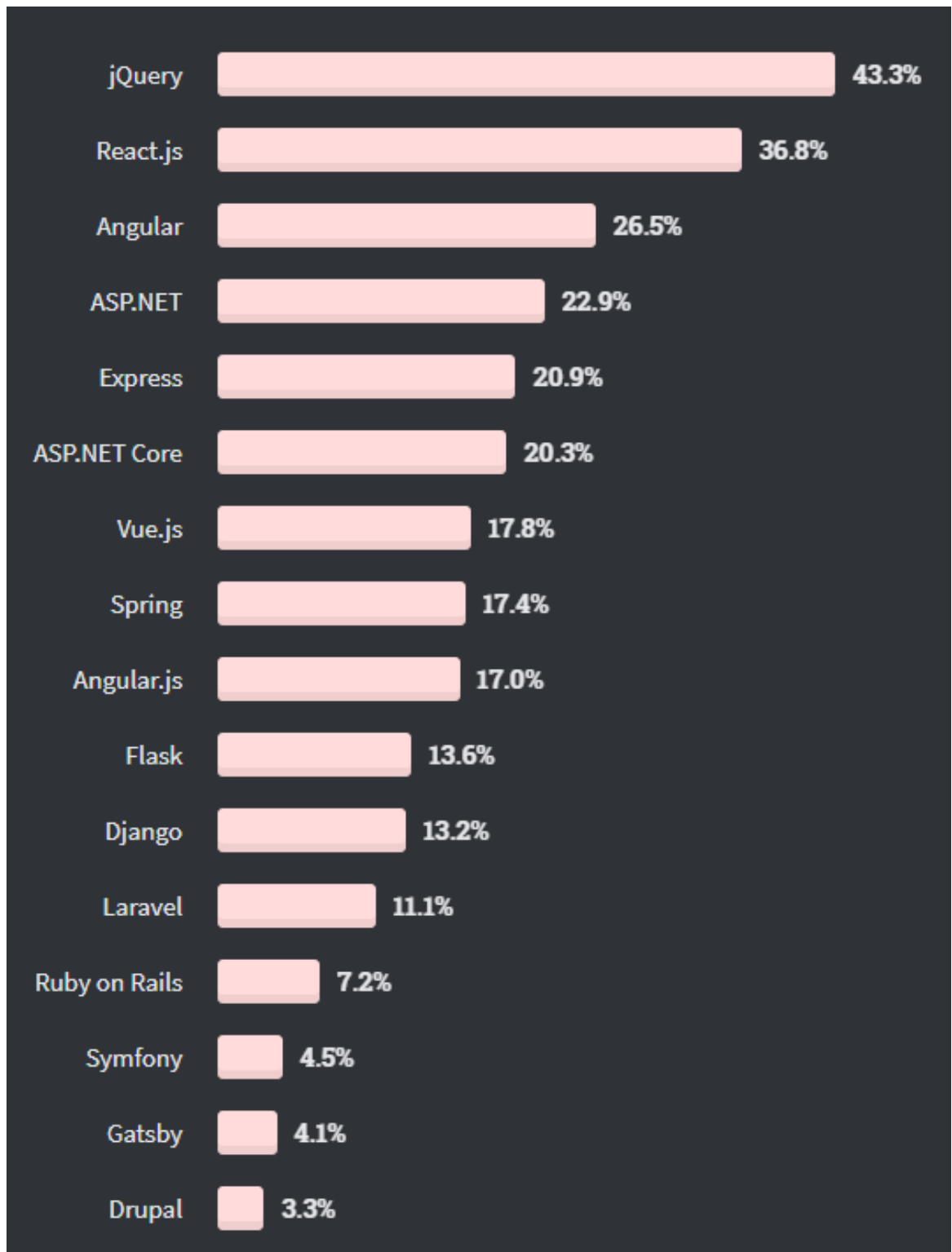
- wysoka wydajność, został zaprojektowany tak aby optymalizować wydajność i skalowalność aplikacji webowych
- pracując nad kodem po stronie klienta jak i po stronie serwera, poruszamy się w tym samym języku programowania
- dostęp do menadżera pakietów np. Yarn czy NPM, dzięki niemu uzyskuje się dostęp do setek tysięcy przeróżnych pakietów.
- jest przenośny. Można korzystać z niego zarówno na systemie macOS, Linux, jak również Windows.
- Dostęp do ogromnego community, gotowego służyć pomocą.

Node.JS jest bardzo uniwersalnym narzędziem, nie służy jedynie do tworzenia serwerów backendowych dla stron internetowych. Można za jego pomocą tworzyć zwykłe aplikacje, aplikacje z dziedziny IoT(Internet of Things), posiada nawet pakiety do uczenia maszynowego. Za pomocą Node jesteśmy w stanie wykonać przeróżne rzeczy ogranicza jedynie znajomość JavaScript oraz własna wyobraźnia.



Rysunek 2: Najpopularniejsze frameworki i biblioteki według ankiety Stack Overflow[14]

Jak zostało zaprezentowane na obrazku 2, Node.JS cieszy się ogromną popularnością. Użył 51.9% na 33 913 ankietowanych, tę grupę stanowili jedynie profesjonalni developerzy.



Rysunek 3: Najpopularniejsze webowe frameworki według ankiety Stack Overflow[14]

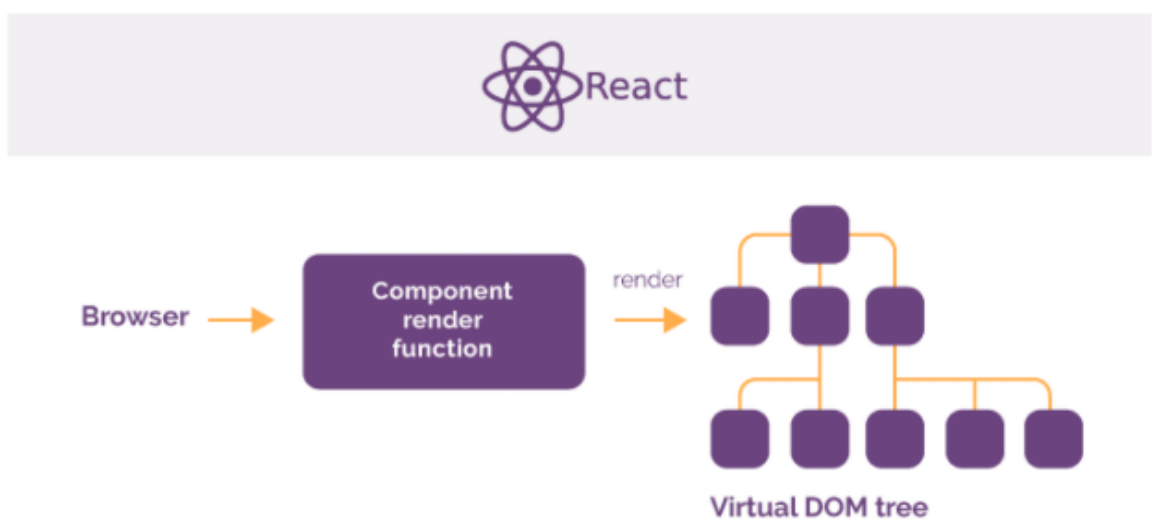
Dodatkowo Express, czyli framework do Node.JS, za pomocą, którego został stworzony cały backend projektu, znajduje się na drugim miejscu w rankingu popularności wśród frameworki.

ków backendowych, osiągnął popularność na poziomie 20.9%. Wyprzedza takie frameworki jak: Java Spring, Flask, Django, Laravel czy Ruby on Rails, jedynie ASP.NET plasuje się wyżej w rankingu.

3.2 React.js

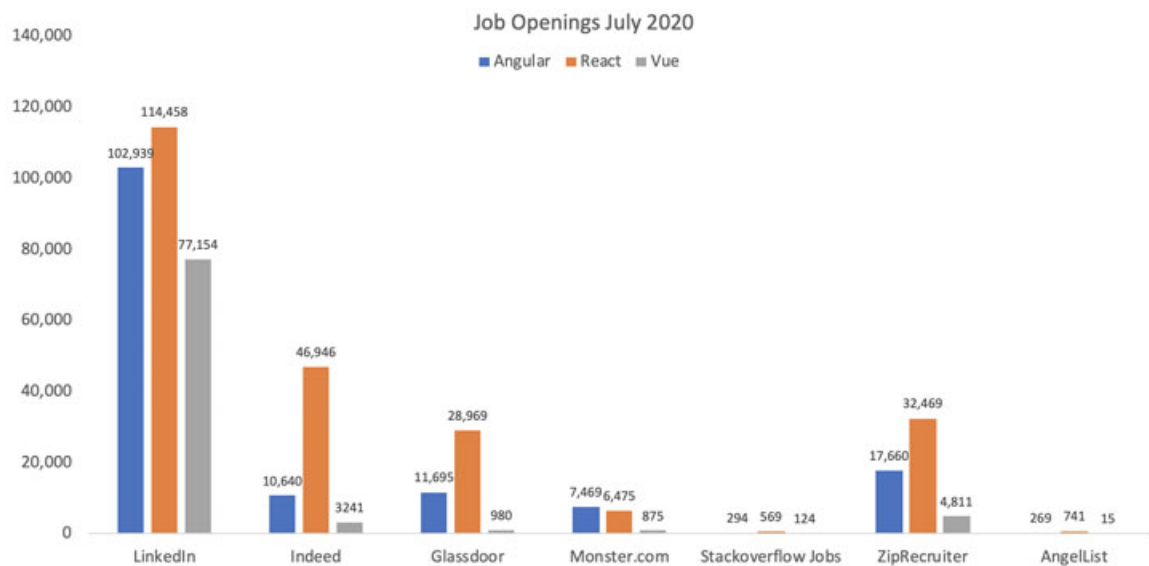
”React A JavaScript library for building user interfaces.”[2] Tłumacząc na polski, React jest biblioteką do tworzenia interfejsów użytkownika. React jest oparty na komponentach, każdy komponent może posiadać własny stan i nim zarządzać, korzystając z komponentów można tworzyć bardzo złożone UI. Sporym plusem Reacta jest fakt, że raz poznany umożliwia również tworzenie aplikacji mobilnych przy wykorzystaniu React Native. Co dodatkowo wyróżnia React jest własna składnia, zbliżona do XML-a, czyli JSX. Jest to mieszanka HTML-a i JavaScriptu, możemy dzięki niemu bez najmniejszych przeszkód umieszczać tagi znane z HTML-a, w kodzie JavaScriptowym, podczas tworzeniu komponentów. Dodatkowo własne komponenty, również podczas tworzenia struktury dokumenty, mają składnię taką jak HTML.

Kolejnym wyjątkowym aspektem Reacta jest Virtual DOM. Jest to koncept programistyczny gdzie wirtualna reprezentacja UI jest przetrzymywana w pamięci i synchronizowana z prawdziwym DOM-em. Umożliwia to podawanie UI w jakim stanie ma się znajdować i jakie komponenty mają być wyrenderowane, również upewnia się, że faktyczny DOM odpowiada temu stanowi. Ściąga to z programisty konieczność stworzenia samemu takich elementów jak: obsługę wydarzeń np. kliknięcie przycisku, manualne aktualizowanie DOM-u oraz manipulację atrybutami.



Rysunek 4: Skrókowa prezentacja działania Reacta

Konkurencje dla Reacta stanowi Angular i Vue, jednakże to React króluje zarówno jeśli chodzi o popularność, pobrania, gwiazdki na GitHubie, liczbę pakietów, które są zależne od Reacta. Niekwestionowanym mistrzem jest również w kategoriach GitHub "Used by", tematach tworzonych na GitHubie.[6] Dodatkowym atutem przemawiającym za Reactem jest ilość pracy na rynku. Jak zostało zaprezentowane na obrazku poniżej, ponownie React króluje i w tej kategorii.



Rysunek 5: Zapotrzebowanie na rynku, na specjalistów z danego frameworku.[4]

3.3 TypeScript

Czym jest TypeScript? W skrócie jest to JavaScript rozszerzający go o statyczne definicje typów, takie jak np. string, number czy boolean. Został stworzony przez firmę Microsoft, przy wsparciu Google. Pisząc w Typescripcie nie trzeba używać typów, interfejsów, typów generycznych czy też innych elementów udostępnianych. Można pisać zwykły JavaScript i mimo, że zostaną wyświetlone błędy informujące o nieokreślonym typie, cały kod wykona się mimo wszystko. Każdy kod napisany w JavaScriptcie będzie w pełni funkcjonalnym kodem TypeScriptowym. Jednakże pisząc w Typescripcie przeglądarka nie rozumie tego języka, zna tylko i wyłącznie JavaScript, dlatego kod musi być skompilowany przy pomocy TypeScriptowego kompilatora lub Babela.[11] Dodatkowo sprawia to, że napisany kod będzie działał na dowolnym urządzeniu i przeglądarce, o ile te wspierają JavaScript.

Dlaczego programiści decydują się na TypeScript?

Wynika to przede wszystkim z obecności typów, pisząc w JavaScriptcie należałoby pisać dodatkowe linijki kodu tylko i wyłącznie sprawdzające np. czy dane otrzymane przez serwer są odpowiednich typów, TypeScript robi wszystko za programistę. Błędy związane z typami należą do najczęściej popełnianych wśród programistów JavaScript, szczególnie w projektach o większej skali bardzo łatwo o takie błędy. Typować można nie tylko zmienne, ale też funkcje, zarówno argumenty jakie przyjmuje oraz typ zwracany.

TypeScript to nie tylko typy, posiada liczne dodatkowe funkcje, których JavaScript nie posiada. Są to chociażby wcześniej wspomniane interfejsy, można deklarować własne typy, typy generyczne oraz typy wyliczeniowe - enumy. Inną niedostępną funkcją jest chociażby przeciążanie funkcji. Wyglądem nie przypomina to kodu znanego z języków programowania typu Java czy C++, lecz spełnia dokładnie takie samo zadanie jak w tamtych językach.

3.4 Najistotniejsze wykorzystane biblioteki

W tej sekcji mogłoby znaleźć się ogromna ilość przeróżnych bibliotek, które są używane w projekcie. Create-React-App(służy do generowania projektów Reactowych), sam w sobie pobiera kilkadziesiąt bibliotek. Tutaj jednak znajdują się tylko biblioteki, który zostały dodane na potrzeby projektu i warto o nich wspomnieć, a są to:

- Express
- Mongoose
- Stripe
- Nodemailer
- QRCode
- Material UI
- React Redux
- jsonwebtoken

Pierwsza z nich czyli Express, jest minimalistycznym, elastycznym frameworkiem do Node.js. Dostarcza takie mechanizmy jak[18]:

- Tworzenie funkcji obsługujących żądania metod HTTP i routing.

- Integrację z różnymi silnikami do generowania widoków, opartych na szablonach stron.
- Konfiguracja podstawowych ustawień aplikacji webowych np. port.
- Dodatkowe przetwarzanie żądań warstwy pośredniej(Middleware).

Sam w sobie Express jest minimalistyczny i nie posiada wielu istotnych funkcji. Dopiero zewnętrzne biblioteki umożliwiają takie aspekty jak: logowanie użytkowników, walidacja danych, zarządzanie ciasteczkami czy też parsowanie.

Drugi z nich czyli Mongoose, służy do porozumiewania się z bazą danych MongoDB. Umożliwia w prosty sposób tworzenie modeli danych, walidację, przeszukiwanie bazy danych oraz zapisywanie zmian w bazie. Lecz sam w sobie jest zdefiniowany jako narzędzie do modelowania danych.

Kolejną spośród wymienionych bibliotek jest Stripe. Stripe jest biblioteką do obsługi płatności internetowych, poprzez podanie danych karty kredytowej lub debetowej, wysyłane jest zapytanie do API, które tworzy płatność, pobiera odpowiednią kwotę z konta i informuje o powodzeniu lub niepowodzeniu transakcji.

Nodemailer jest prostym modułem dla aplikacji opartych na Node.js umożliwiającym i znacznie upraszczającym wysyłanie maili. Jest on o tyle niezbędny, że każdy bilet wysyłany jest na maila, jedynie zalogowani użytkownicy mają wgląd do historii transakcji i mogą podejrzeć kod QR z poziomu aplikacji. Użytkownicy goście nie mają innego dostępu do biletów, jedynie przychodzą one na maila.

QRCode jest biblioteką do generowania kodów QR. Jest to technologia bardzo obecnie popularna i wygodna, praktycznie każdy obecny smartfon posiada czytnik kodów QR.

Material UI jest jedną z najpopularniejszych bibliotek komponentów. Służy do tworzenia interfejsów użytkownika, bazując na gotowych komponentach, dodatkowo znacznie usprawniające tworzenie responsywnych interfejsów.

React Redux jest to biblioteka do Reacta, udostępniająca przetrzymywanie globalnego stanu aplikacji i dostępu do niego z każdego jej poziomu. Jest to o tyle przydatne narzędzie,

że bez jego wykorzystania developer byłby zmuszony do przekazywania danych między komponentami w bardzo nieelegancki i toporny sposób. Redux umożliwia manipulację i dostęp do globalnego stanu z dowolnego komponentu.

Jsonwebtoken jest biblioteką do tworzenia JWT, służący przede wszystkim do autoryzacji użytkownika, szczególnie przydatne jest to przy logowaniu. Szerzej o JWT przedstawione zostanie w jednej z kolejnych podsekcji.

3.5 MongoDB

MongoDB jest typem nierelacyjnej bazy danych, posiada strukturę zbliżoną do JSON-a, czyli klucz-wartość. Oznacza to, że każdy dokument może mieć inną strukturę, zupełnie inaczej niż jest to w relacyjnych bazach. MongoDB charakteryzuje się wysokim bezpieczeństwem, dużą skalowalnością i łatwością obsługi. Świadczyć o tym może fakt, że z usług tej bazy danych korzystają takie firmy jak Google, Adobe czy ebay.

MySQL vs MongoDB (informacje pochodzą z [12])

MySQL jest popularną, darmową i open-source relacyjną bazą danych stworzoną przez firmę Oracle. Jak to w relacyjnych bazach danych, informacje przechowywane są w formie tabel, wykorzystywany jest język zapytań SQL.

MongoDB również jest darmowy oraz open-source, dane przechowywane są w formie dokumentów, jako pary klucz-wartość.

Biorąc pod lupę wydajność obu baz, ciężko jest porównywać dwie zupełnie różne bazy, o dwóch różnych typach. Jednakże można zauważyć, że MySQL jest szybsze przy pobieraniu dużej liczby rekordów, z kolei MongoDB jest zdecydowanie szybsze przy dodawaniu i aktualizacji dużej liczby rekordów.

Innym istotnym aspektem porównawczym jest skalowalność. MongoDB zważywszy na fakt, iż jest bazą nierelacyjną posiada strukturę dokumentu, zbliżoną do JSONa, jest wysoce skalowalna i można to robić w bardzo prosty sposób, bez obaw o utraty czy uszkodzenia danych. MySQL jest dużo bardziej ograniczone, ponieważ jest bazą relacyjną. Jedyne dostępne opcje na skalowanie to skalowanie pionowe lub tworzenie replik do odczytu.

Innym istotnym aspektem jest elastyczność bazy. Tutaj zdecydowanie króluje MongoDB, każdy z rekordów może posiadać inne pola, o innych typach, podczas gdy w MySQL nie jest to dostępne. Tam struktura tabeli jest sztywna i nie jest możliwe by choćby jeden rekord miał inne

pole zamiast obecnie występującego.

Decyzja o wyborze MongoDB była dość prosta, biblioteka Mongoose do Node.JS, jest łatwa i przyjemna w obsłudze, dodatkowo korzyści płynące z korzystania z MongoDB są na tyle interesujące, że to na tę bazę danych padł wybór.

3.6 JWT - JSON Web Token

Jest otwartym standardem (RFC 7519), który definiuje kompaktową i samowystarczalną możliwość, bezpiecznej transmisji danych w formacie JSON[5]. Informacje przekazywane za pomocą JWT są weryfikowalne i można im ufać, ponieważ są podpisywane cyfrowo. JWT można podpisać za pomocą algorytmu HMAC przy wykorzystaniu sekretu lub pary klucza publicznego i prywatnego przy wykorzystaniu algorytmów RSA lub ECDSA. Sekret jest tajnym hasłem przechowywanym przez serwer do podpisywania i autoryzacji tokenów. Najczęstszym powodem wykorzystania JWT jest autoryzacja użytkownika, czy to przy logowaniu, czy też przy sprawdzaniu dostępu do np. jakiegoś url w aplikacji. Innym popularnym zastosowaniem jest wymiana informacji.

JSON Web Token składa się z następujących elementów: nagłówka (Header), payloadu oraz podpisu, oddzielonych od siebie kropkami. Przeważnie struktura JWT wygląda następująco:
xxxxxx.yyyyyy.zzzzzz

Nagłówek przeważnie składa się z dwóch części: typu wygenerowanego tokena oraz algorytmu, jakim został podpisany.

Payload jest odpowiedzialny za przetrzymywanie danych zawartych w tokenie.

Zarówno nagłówek, jak i payload są kodowane za pomocą Base64-URL do stworzenia odpowiednio pierwszej i drugiej części JWT.

Do stworzenia podpisu należy wziąć zakodowany nagłówek i payload, sprecyzować algorytm szyfrujący oraz prywatny klucz lub sekret i podpisać to. Przykładowo przy wykorzystaniu algorytmu HMAC SHA256, podpis zostanie stworzony w następujący sposób:


```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret)
```

Rysunek 6: Sposób tworzenia podpisu JWT przy wykorzystaniu algorytmu HMAC SHA256.

Wynikiem takiej operacji są trzy ciągi znaków Base64-URL, oddzielonych od siebie kropkami.

JWT w tworzonej aplikacji wykorzystywane jest przy rejestracji, logowaniu oraz dostępu do niektórych ścieżek. Rejestracja przebiega następująco:

1. Użytkownik podaje dane niezbędne do rejestracji.
2. Użytkownik zatwierdza podane dane.
3. Wysyłane jest zapytanie do serwera, gdzie tworzony jest token, hashowane jest hasło i tworzony nowy wpis w bazie danych.

Dzięki temu po zalogowaniu użytkownik ma dostęp do inaczej niedostępnej sekcji, czyli historii zamówień, gdzie można znaleźć wszystkie posiadane bilety wraz z ich kodami QR.

4 Proces tworzenia aplikacji

Dobłą praktyką i ogólnie przyjętą zasadą jest rozpoczęcie tworzenia aplikacji webowej od strony backendowej. Zgodnie z tym podejściem prace rozpoczęły się od backendu, został stworzony podstawowy serwer logujący proste hasło "działa". Kolejnym etapem było podpięcie bazy danych MongoDB.

```
import express from 'express';  
import mongoose from 'mongoose';  
import keys from './config/keys';  
require('./models/Ticket');
```

```
mongoose.connect(keys.mongoURI, { useNewUrlParser: true, useUnifiedTopology: true });

const app = express();
```

Listing 1: Kod odpowiadający za integrację z MongoDB.

Do połączenia z bazą danych wykorzystywana jest biblioteka Mongoose, o której szerzej w sekcji 3.4. W następnej kolejności został wygenerowany za pomocą Create-React-App, projekt Reactowy, będący bazą dla frontendu aplikacji. Niezbędnym zabiegiem do połączenia obu stron aplikacji, jest dodanie oprogramowania pośredniczącego proxy, aby połączenie między serwerem stworzonym w Node.JS, a stroną Reactową mogło zostać nawiązane. Do tego została wykorzystana biblioteka http-proxy-middleware. Do funkcjonowania tego połączenia wystarczyło stworzyć nowy plik setupProxy.js, prezentujący się następująco:

```
const {createProxyMiddleware} = require('http-proxy-middleware');

module.exports = function (app) {
  app.use("/api", createProxyMiddleware({target: "http://localhost:3001"}));
}
```

Listing 2: Proxy Middleware do połączenia strony frontendowej z backendową.

Kolejnym celem było stworzenie prostego formularza, służącego do przyjmowania danych osobowych użytkownika oraz dane karty płatniczej. Następnie powstał cały system płatności, a dokładniej dane podane przez użytkownika są przekazywane do serwera, który następnie tworzy "Payment Intent", czyli Stripową(3.4), nową płatność.

```
const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {
  e.preventDefault();
  if(!stripe || !elements) {
    return;
  }

  const { data: clientSecret } = await axios.get("/api/payment_intent");

  const cardElement = elements.getElement(CardElement);

  setIsProcessing(true);
```

```

const fullName = firstName + " " + lastName;
const paymentMethodReq = await stripe.createPaymentMethod({
  type: 'card',
  card: cardElement!,
  billing_details: {email: email, name: fullName, phone: phoneNum
});
if (paymentMethodReq.error) {
  console.log('error', paymentMethodReq.error);
  setIsProcessing(false);
  return;
}
const { error } = await stripe.confirmCardPayment(clientSecret, {
  payment_method: paymentMethodReq.paymentMethod?.id,
});
if (error) {
  setIsProcessing(false);
  return;
}
console.log('payment method', paymentMethodReq.paymentMethod);
}

```

Listing 3: Obsługa przekazywania danych klienta na serwer w celu dokonania nowej płatności.

```

app.get("/api/payment_intent", async (req, res) => {
  try {
    const paymentIntent = await stripe.paymentIntents.create({
      amount: 400,
      currency: "pln",
      payment_method_types: ['card'],
      metadata: {integration_check: "accept a payment"},
    });
    res.status(200).send(paymentIntent.client_secret);
  } catch (error) {

```

```

        res.status(500).json({ statusCode: 500, message: error.message });
    }
});

```

Listing 4: Podstawowy system obsługi płatności przez serwer

Jednakże sam system płatności to mało, niezbędne było aby był tworzony nowy rekord w bazie danych, odpowiadający biletowi dla danego wydarzenia. Warunkiem jaki musi zająć, aby bilet mógł zostać stworzony jest by liczba biletów, na wydarzenie nie przekroczyła określonej, maksymalnej liczby dostępnych biletów.

```

Ticket.find({ eventId: event?.id }, (error, tickets) => {
    if (!error) {
        if (event?.toJSON().maxTicketsAmount - 1 < tickets.length) {
            res.status(403).send("No tickets left");
            return;
        }
    } else res.status(404).send("No tickets found")
});

const ticket = new Ticket({
    email: email.trim(),
    firstName: firstName.trim(),
    lastName: lastName.trim(),
    phoneNumber: phoneNumber.trim(),
    eventId: eventFound.id,
    purchaseDate: new Date(),
});

ticket.save((error) => {
    if (error) {
        res.status(500).send("Ticket cannot be added to database");
        return;
    }
});

```

Listing 5: Tworzenie nowego biletu w bazie danych.

Kolejny etap odbył się po stronie frontendowej, została stworzona mała karta, prezentująca w

skrótce każde z wydarzeń. Dla celów estetycznych, niezbędne było aby widoczne były na nich zdjęcia, co dodatkowo wymagało, aby były one pobierane z serwera.

```
app.use(express.static("public"));
```

Listing 6: Linijka kodu odpowiadająca za podpięcie folderu public, w którym znajdują się zdjęcia wydarzeń.

W następnej kolejności został wykonany wstępny Nav Bar z możliwością wyszukiwania wydarzenia po nazwie, jak również filtrowania po typach wydarzeń. Dla wygody programisty wymagało to podpięcia biblioteki Redux, w celu stworzenia globalnego stanu, który może być współdzielony przez wszystkie komponenty. Ponieważ, filtrowane wartości są, w późniejszym etapie rozwoju aplikacji, użyteczne.

Następnym bardzo istotnym etapem, było stworzenie walidacji danych przesyłanych na serwer w formie do płatności, oraz danych wpisywanych przez użytkownika. Do walidacji wykorzystane zostały biblioteki express-validator oraz React-Hook-Form, odpowiednio dla strony backendowej i frontendowej. Sprawdzane są typy danych podawane w formie oraz za pomocą REGEX, czy podana wartość, np. adres email, jest zgodny ze wzorem.

```
<Input
  name="email"
  type="email"
  id="input_email"
  autoComplete="off"
  inputRef={register({
    required: "Please specify email.",
    pattern: {
      value:
        /^[^<>()[\]\.\,\;\s@"]+(\.[^<>()[\]\.\,\;\s@"]+)*|("[^"]+")@
      message: "Invalid email.",
    },
  })}
  required
/>
```

Listing 7: Przykładowy pattern walidacji emaila, po stronie klienta.

Kolejną funkcjonalnością zaimplementowaną, zostało wysyłanie maili. Do klient, po dokonaniu poprawnej płatności, zostaje wysłany email, z informacją o poprawnie dokonanej płatności i docelowo z kodem QR, będący biletem. Do tego została wykorzystana biblioteka nodemailer3.4.

```
const mailTemplate = `
  <h1>Hello ${firstName} ${lastName}</h1>
  <p>Thanks for buying ticket for ${event.nameOfEvent}, in ${event.
  
  `;

const message = {
  from: env.email,
  to: email,
  subject: 'Ticket for ${event.nameOfEvent}',
  html: mailTemplate,
};

transporter.sendMail(message, (error, info) => {
  if (error) {
    let err = new StatusError("Error while sending mail", 500);
    return next(err);
  } else console.log("Mail sent:", info.response);
});
```

Listing 8: Kod odpowiadający za wysyłanie emaili.

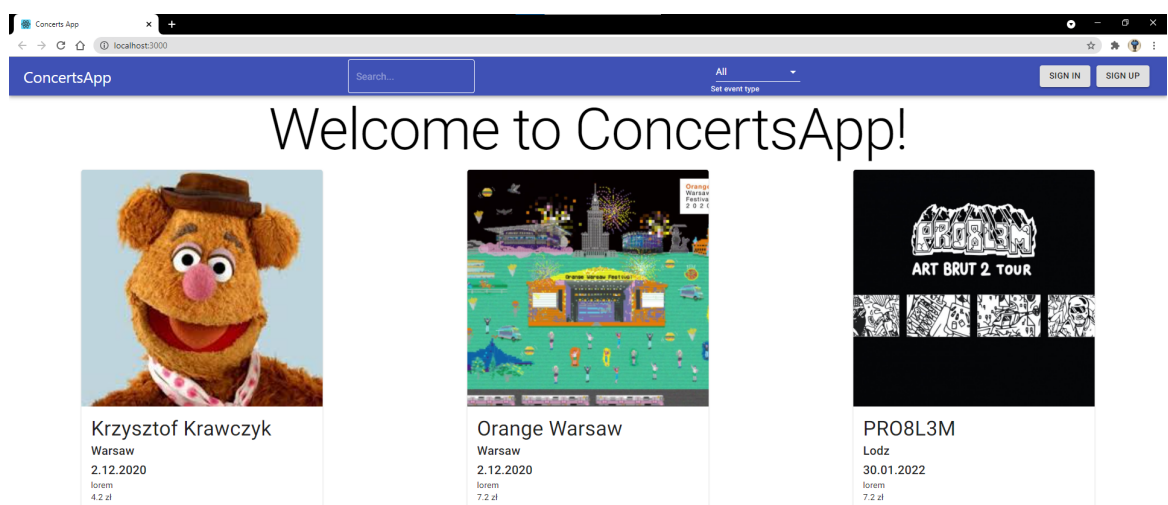
Kolejnym, ale nie ostatnim elementem, było generowanie, wysyłanie w mailu oraz przetrzymywanie kodów QR. Kod jest generowany na podstawie id biletu, znajdującego się w bazie danych. Do jego realizacji wykorzystano bibliotekę QRCode3.4.

```
const qr = await toDataURL(ticket.id);
if (!qr) {
  let err = new StatusError("Error while creating QR Code", 400);
  return next(err);
}
```

Listing 9: Kod odpowiadający za generowanie kodów QR.

Ostatnią funkcjonalnością, ale nie ostatnim krokiem było logowanie się i rejestracja użytkowników. Do jej zrealizowania było potrzebne kilka bibliotek, a dokładniej: jsonwebtoken, bcrypt, o tej pierwszej szerzej w rozdziale 3.4, druga z kolei odpowiada za hashowanie hasła. Logowanie się użytkownika, umożliwia mu dostęp do historii zakupionych biletów, na wydarzenia. Dodatkowo otrzymuje dostęp do kodów QR swoich biletów. Niezalogowany użytkownik ma możliwość zakupu biletu, kod QR otrzymuje na maila, ale w razie np. utraty maila nie ma innej opcji dostępu do niego. Co czyni rejestrację użytkownika bardzo atrakcyjnym.

Finalnym krokiem zostało dodanie stylu CSS i w pełni wykorzystanie biblioteki komponentów Material-UI. Po wszystkich zmianach wizualnych, aplikacja prezentuje się następująco:

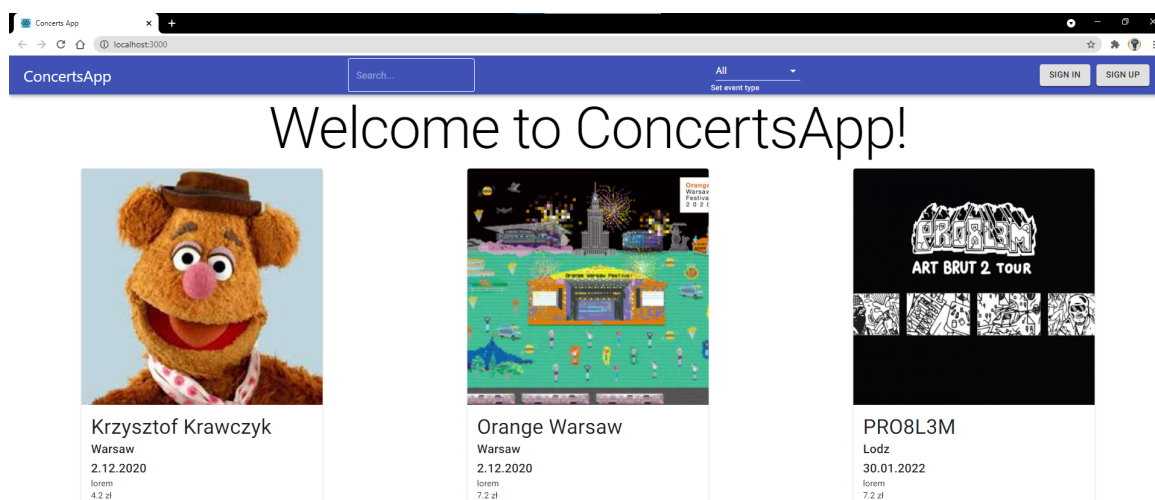


Rysunek 7: Finalny wygląd aplikacji. (opracowanie własne)

5 Aplikacja od strony użytkownika

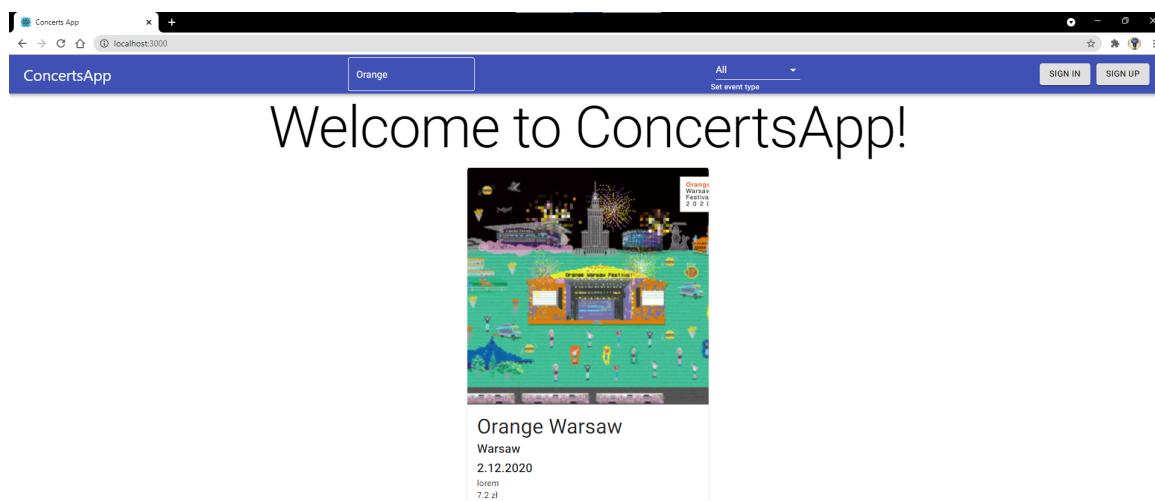
Po otwarciu aplikacji oczom klienta ukaże się strona główna, zawierająca takie elementy jak:

- wyszukiwarkę wydarzeń po nazwie
- filtr po typach wydarzeń
- przyciski do logowania i rejestracji użytkownika
- listę wszystkich dostępnych na stronie wydarzeń

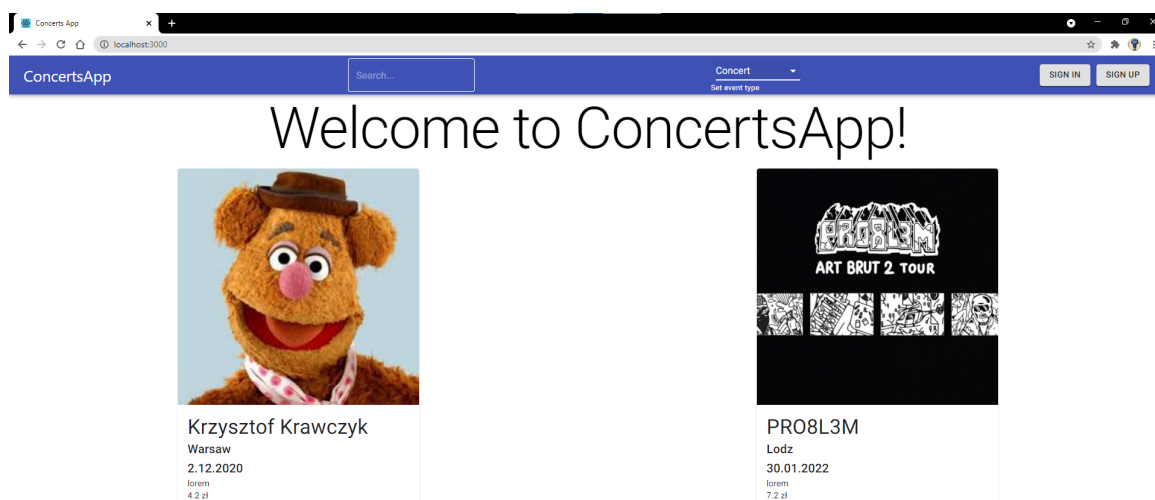


Rysunek 8: Strona główna aplikacji. (opracowanie własne)

Z poziomu strony głównej użytkownik ma kilka opcji, może wybrać bezpośrednio interesujące go wydarzenie i przejść do niego, zarejestrować się, zalogować się oraz filtrować wydarzenia według jego uznania. Filtrowanie odbywa się po typie wydarzeń i jest to zrobione w formie listy rozwijanej, oraz wyszukiwać, w odpowiednim polu, po nazwie wydarzenia.

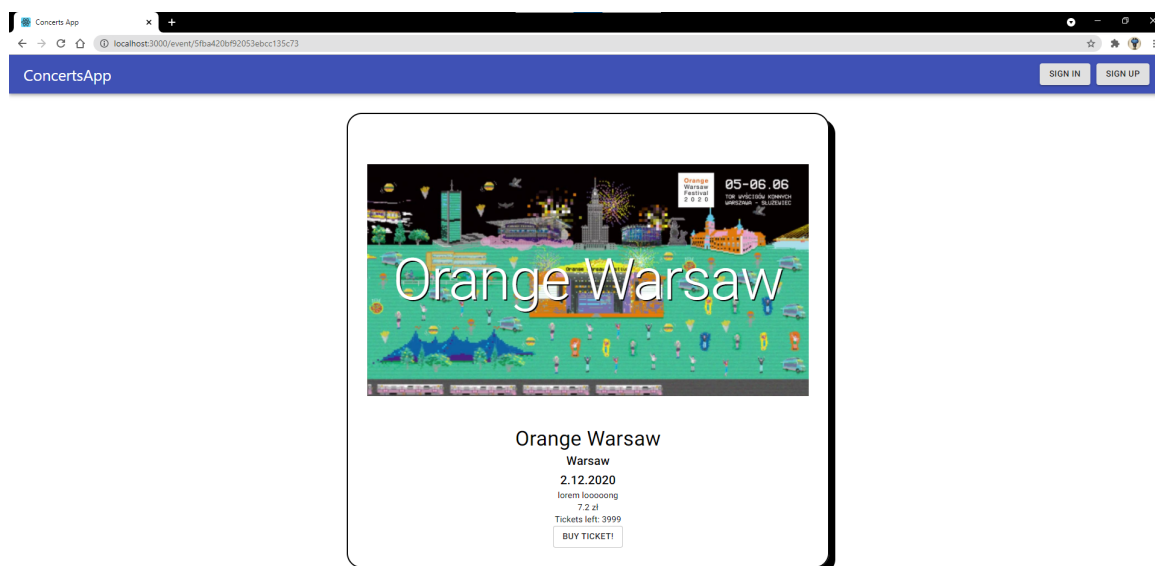


Rysunek 9: Wyszukiwanie po frazie. (opracowanie własne)



Rysunek 10: Filtrowanie po typie wydarzenia. (opracowanie własne)

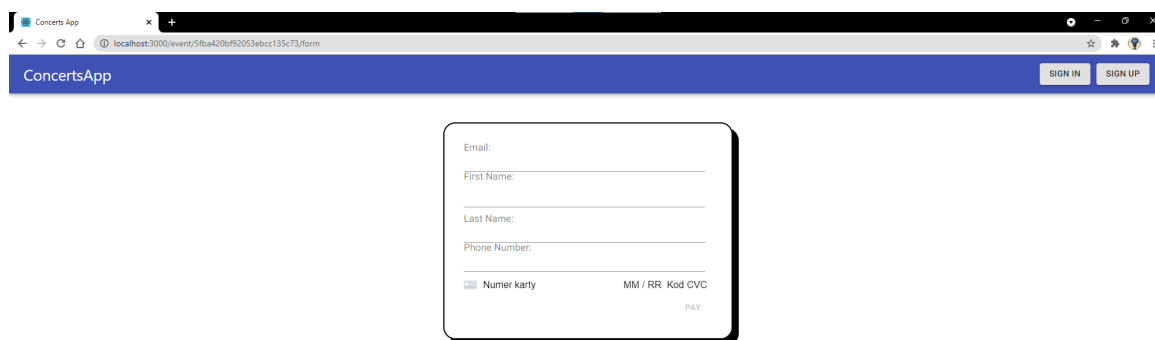
Po naciśnięciu jednej z kart wydarzenia, widocznych na stronie głównej, klient zostaje przeniesiony na stronę konkretnego wydarzenia, gdzie może nieco więcej dowiedzieć się o danym evencie i kupić na nie bilet.



Rysunek 11: Strona konkretnego wydarzenia, w tym przypadku Orange Warsaw. (opracowanie własne)

Kupowanie biletu jest dostępne zarówno dla użytkowników zalogowanych, jak i niezalogowanych. Po naciśnięciu na przycisk "Buy Ticket" klient jest przenoszony do forma, umożliwia-

jącego podanie danych do realizacji zakupu.

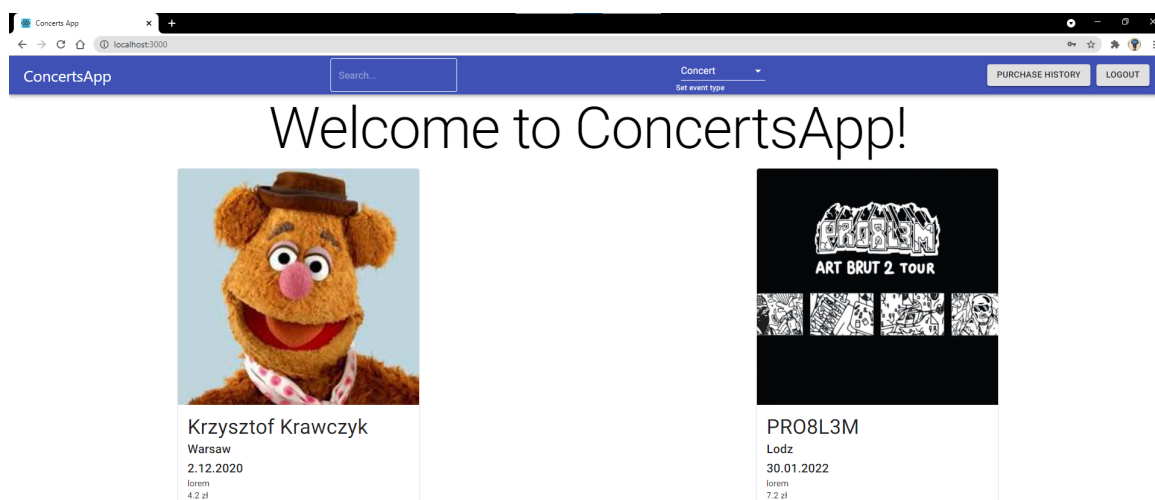


The screenshot shows a web browser window with the title "Concerts App". The address bar displays "localhost:3000/event/5fba420bf92053ebcc135c73/form". The page has a blue header with the "ConcertsApp" logo on the left and "SIGN IN" and "SIGN UP" buttons on the right. The main content area features a white registration form with the following fields: "Email:", "First Name:", "Last Name:", and "Phone Number:". Below these fields are two checkboxes: "Numer karty" (checked) and "MM / RR Kod CVC". At the bottom right of the form is a "PAY" button.

Rysunek 12: Form do zebrania danych, niezbędnych do realizacji zakupu biletu. (opracowanie własne)

Po podanych poprawnych danych, w formacie wymaganym przez aplikację, użytkownik jest informowany o pomyślnie zrealizowanym zakupie, lub w przypadku gdy coś poszło nie tak zostaje przeniesiony na stronę informującą o niepowodzeniu.

Użytkownik ma możliwość założenia konta lub jeśli już je posiada zalogowania się na nie. Minimalnie zmienia to widok aplikacji, zamiast przycisków "Sign In" oraz "Sign Up" pojawiają się dwa inne przyciski "Purchase history" oraz "Logout".



Rysunek 13: Strona główna aplikacji, po zalogowaniu. (opracowanie własne)

Pierwszy z wymienionych nowych przycisków, daje użytkownikowi opcję niedostępną w innym przypadku, a konkretniej podgląd wszystkich biletów, które zakupił, wraz z ich kodem QR.



Rysunek 14: Lista biletów klienta, dostępna za pomocą Purchase History. (opracowanie własne)

Dodatkowym atutem zalogowanych użytkowników jest fakt, iż przy kupnie biletu nie muszą oni ponownie podawać tych samych danych, które podawali przy rejestracji np. imię czy numer telefonu. Jedyne dane jakie są zmuszeni podać to dane karty płatniczej. Co zdecydowanie

przyspiesza i ułatwia obcowanie z aplikacją, przy częstym jej użytkowaniu.

Tak prezentuje się aplikacja od strony klienta, jest wyjątkowo, prosta, czytelna i łatwa w obsłudze. Celem nie było sprawienie, aby użytkownik był przytłoczony tym co widzi, tylko w łatwy sposób, mógł osiągnąć cele takie jak: kupno biletu lub zdobycie informacji o wydarzeniu.

6 Podsumowanie

Głównym celem i założeniem pracy było zaprojektowanie i stworzenie aplikacji webowej do promocji wydarzeń kulturowych i sprzedaży biletów na nie. Udało się spełnić założony cel przy wykorzystaniu technologii takich jak: React.JS, Node.JS czy MongoDB. Aplikacja sama w sobie nie jest rewolucyjna, celem pracy nie było odkrywanie koła na nowo, tylko stworzenie aplikacji umożliwiającej potencjalnemu klientowi, w prosty i szybki sposób zapoznanie się z dostępnymi wydarzeniami, informacjami o nich oraz zakup biletów. Wszystkie te założenia zostały spełnione, interfejs jest bardzo prosty i przejrzysty, wydarzenia są widoczne już na stronie głównej, co dodatkowo sprawia, że użytkownik, nie musi przeszukiwać strony aby je odnaleźć, płatności są realizowane w przystępny sposób oraz same bilety są w najprostszej i jednocześnie najwygodniejszej postaci, w formie kodów QR. Dodatkowo w samej pracy przybliżone zostały takie kwestie jak bazy danych, ich typy i zastosowanie, technika tworzenia aplikacji jaką jest model kaskadowy, typy aplikacji webowych, Rest API, najpopularniejsze metody płatności w internecie i przeanalizowane zostały konkurencyjne portale do sprzedaży biletów na wydarzenia kulturowe. Zaprezentowane zostały również narzędzia, jak również technologie, które zostały wykorzystane na potrzeby aplikacji, pokrótce również zostało omówione, jakie potencjalne zmiany i rozszerzenia samej aplikacji mogą się znaleźć, aby doświadczenie w obcowaniu z aplikacją było jeszcze lepsze. Krok po kroku zostało pokazane jak powstawała sama aplikacja, jakie funkcjonalności były po kolei dodawane i jaki jest efekt finalny. Dodatkowym rozdziałem jest zaprezentowanie aplikacji od strony użytkownika, znajdują się w nim screeny oraz krótkie opisy, przez jaki proces przechodzi każdy użytkownik aplikacji.

7 Możliwości dalszego rozwoju aplikacji

Możliwości dalszego rozwoju aplikacji są ogromne i różne. Pierwszą przykładową jest stworzenie systemu newslettera, na który użytkownik mógłby się zapisać i otrzymywać maile z najnowszymi informacjami i zaproszeniami na wydarzenia.

Inną, istotną funkcjonalnością mogłoby być resetowanie hasła użytkownika. W chwili obecnej nie występuje żadna możliwość resetu hasła, co stanowi bardzo ważny element każdej aplikacji, w której w grę wchodzi logowanie się użytkowników.

Również bardzo przydatne byłaby obsługa innych typów płatności, aktualnie jest to tylko płatność danymi karty płatniczej. Szczególnie wartymi dodania byłaby płatność PayPal, Blikiem czy zwykłym przelewem bankowym.

Bardzo istotnym jest stworzenie systemu sprawdzającego bilety i kody QR. Aktualnie nie ma żadnej weryfikacji biletów, więc generowane kody są w zasadzie pustym zlepkiem pikseli. Jest to funkcja o tyle istotna, że żadna firma nie zdecyduje się na publikowanie swojego wydarzenia i dystrybucji biletów przez stronę, która nie daje możliwości, sprawdzenia czy kod QR prezentowany przez klienta odpowiada temu, który został zakupiony i czy jest on powiązany z konkretnym wydarzeniem.

Kluczową funkcją mogłaby okazać się obsługa płatności i języków innych krajów. Aktualnie płatność odbywa się w złotych, a językiem figurującym w aplikacji jest język angielski. Język angielski, ma to do siebie, że jest znany przez bardzo szerokie grono ludzi, jednakże płatność dla obcokrajowców, w innej walucie, może okazać się niekorzystna. Dochodzą takie opłaty jak chociażby przewalutowanie, a dodatkowe opłaty nigdy nie są mile widziane przez klientów.

Ostatnim przykładową możliwością rozwoju aplikacji jest stworzenie aplikacji mobilnej, w której nawet niezalogowany użytkownik, miałby dostęp do wszystkich posiadanych przez siebie biletów, tak jak jest to chociażby w GoingApp.

Bibliografia

- [1] Smita Deshmukh, Deepak Mane, and Abhijeet Retawade. “Building a Single Page Application Web Front-end for E-Learning site”. In: dostęp na: 15.04.2021. 2019.
- [2] Facebook. *React Docs*. dostęp na: 30.04.2021. URL: <https://reactjs.org/>.
- [3] Adrian Horzyk. *NIERELACYJNE BAZY DANYCH – NoSQL I ASOCJACYJNE STRUKTURY DANYCH*. dostęp na: 5.04.2021. URL: <http://home.agh.edu.pl/~horzyk/lectures/db/BazyDanychAccess-NoSQL.pdf>.
- [4] Daniel Hvidding and Ned Visolyaputra. *What’s the most popular front-end framework?* dostęp na: 30.04.2021. 2020. URL: <https://www.accenture.com/us-en/blogs/software-engineering-blog/hvidding-visolyaputra-front-end-framework>.
- [5] *JWT*. dostęp na: 15.06.2021. URL: <https://jwt.io/introduction>.
- [6] Tanguy Krotoff. *Frontend frameworks popularity*. dostęp na: 30.04.2021. URL: <https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190>.
- [7] Cybersecurity UMCS Lublin. *Software Development Life Cycle*. dostęp na: 3.04.2021. URL: <https://cybersecurity.umcs.lublin.pl/wp-content/uploads/kmazur/I02020/lab6.pdf>.
- [8] Microsoft. *Baza danych NoSQL — co to jest NoSQL?* dostęp na: 5.04.2021. URL: <https://azure.microsoft.com/pl-pl/overview/nosql-database/>.
- [9] Microsoft. *Choose Between Traditional Web Apps and Single Page Apps (SPAs)*. dostęp na: 15.04.2021. URL: <https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps>.
- [10] Microsoft. “Projekt internetowego interfejsu API”. In: (). dostęp na: 27.04.2021. URL: <https://docs.microsoft.com/pl-pl/azure/architecture/best-practices/api-design>.
- [11] Microsoft. *TypeScript Docs*. dostęp na: 30.04.2021. URL: <https://www.typescriptlang.org>.
- [12] *MongoDB vs MySQL Differences*. dostęp na: 15.06.2021. URL: <https://www.mongodb.com/compare/mongodb-mysql>.

- [13] Mozilla. *Wprowadzenie do Express/Node*. dostęp na: 27.04.2021. URL: https://developer.mozilla.org/pl/docs/Learn/Server-side/Express_Nodejs/Introduction.
- [14] Stack Overflow. *2020 Developer Survey*. dostęp na: 27.04.2021. URL: <https://insights.stackoverflow.com/survey/2020#developer-profile-coding-as-a-hobby>.
- [15] Gemius Polska. *E-commerce w Polsce 2020*. dostęp na: 2.04.2021. 2020. URL: <https://www.gemius.pl/wszystkie-artykuly-aktualnosci/e-commerce-w-polsce-2020.html>.
- [16] Oracle Polska. *Co to jest baza danych?* dostęp na: 5.04.2021. URL: <https://www.oracle.com/pl/database/what-is-database/>.
- [17] Oracle Polska. *Co to jest hurtownia danych?* dostęp na: 5.04.2021. URL: <https://www.oracle.com/pl/database/what-is-a-data-warehouse/>.
- [18] *Wprowadzenie do Express/Node*. dostęp na: 27.04.2021. URL: https://developer.mozilla.org/pl/docs/Learn/Server-side/Express_Nodejs/Introduction.

Spis rysunków

1	Model kaskadowy [7]	14
2	Najpopularniejsze frameworki i biblioteki według ankiety Stack Overflow[14]	17
3	Najpopularniejsze webowe frameworki według ankiety Stack Overflow[14]	18
4	Skrótowa prezentacja działania Reacta	19
5	Zapotrzebowanie na rynku, na specjalistów z danego frameworku.[4]	20
6	Sposób tworzenia podpisu JWT przy wykorzystaniu algorytmu HMAC SHA256.	25
7	Finalny wygląd aplikacji. (opracowanie własne)	31
8	Strona główna aplikacji. (opracowanie własne)	32
9	Wyszukiwanie po frazie. (opracowanie własne)	32
10	Filtrowanie po typie wydarzenia. (opracowanie własne)	33
11	Strona konkretnego wydarzenia, w tym przypadku Orange Warsaw. (opracowanie własne)	33
12	Form do zebrania danych, niezbędnych do realizacji zakupu biletu. (opracowanie własne)	34
13	Strona główna aplikacji, po zalogowaniu. (opracowanie własne)	35
14	Lista biletów klienta, dostępna za pomocą Purchase History. (opracowanie własne)	35

Spis tabel

1	Tabela decyzyjna wyboru pomiędzy SPA, a MPA. [9]	12
---	--	----

Listingi

1	Kod odpowiadający za integrację z MongoDB.	25
2	Proxy Middleware do połączenia strony frontendowej z backendową.	26
3	Obsługa przekazywania danych klienta na serwer w celu dokonania nowej płatności.	26
4	Podstawowy system obsługi płatności przez serwer	27
5	Tworzenie nowego biletu w bazie danych.	28
6	Linijka kodu odpowiadająca za podpięcie folderu public, w którym znajdują się zdjęcia wydarzeń.	29
7	Przykładowy pattern walidacji emaila, po stronie klienta.	29
8	Kod odpowiadający za wysyłanie emaili.	30
9	Kod odpowiadający za generowanie kodów QR.	30