**WYŻSZA SZKOŁA**

**INFORMATYKI i ZARZĄDZANIA**

**z siedzibą w Rzeszowie**

**WYDZIAŁ INFORMATYKI STOSOWANEJ**

**Kierunek: INFORMATYKA**

**Specjalność: Programowanie**

Oleksandr Lisivka

Nr albumu 53410

***Serwis oglószeń do podroży***

Promotor: dr Marek Jaszuk

**PRACA DYPLOMOWA INŻYNIERSKA**

**Rzeszów 2018**

Ja niżej podpisany/a oświadczam, że składana przeze mnie praca dyplomowa pt. „...........................” została przygotowana samodzielnie.

Oświadczam również, że praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna ze złożoną wersją elektroniczną.

.........................................................

data czytelny podpis autora

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

................................................................

data czytelny podpis promotora

Оглавление

[**Wstęp** 5](#_Toc510563829)

[**Cel pracy** 5](#_Toc510563830)

[**Uzasadnienie wyboru tematu pracy** 6](#_Toc510563831)

[**Zakres pracy** 6](#_Toc510563832)

[**Omówienie układu pracy, krótka charakterystyka poszczególnych rozdziałów** 7](#_Toc510563833)

[**Część teoretyczna** 8](#_Toc510563834)

[**Front-end** 8](#_Toc510563835)

[**HTML5** 8](#_Toc510563836)

[**CSS3** 8](#_Toc510563837)

[**Bootstrap** 9](#_Toc510563838)

[**JavaScript** 9](#_Toc510563839)

[**JQuery** 9](#_Toc510563840)

[**Back-end** 9](#_Toc510563841)

[**Node.js** 10](#_Toc510563842)

[**Npm** 10](#_Toc510563843)

[**REST API** 11](#_Toc510563844)

[**Baza danych** 11](#_Toc510563845)

[**Relacji pomiędzy danymi** 11](#_Toc510563846)

[**Git oraz Github** 12](#_Toc510563847)

[**Visual Studio Code oraz Sourcetree** 12](#_Toc510563848)

[**Część praktyczna** 13](#_Toc510563849)

[**Specyfikacja wymagań** 13](#_Toc510563850)

[**Wymagania biznesowe** 13](#_Toc510563851)

[**Wymagania funkcjonalne** 13](#_Toc510563852)

[**Wymagania niefunkcjonalne** 13](#_Toc510563853)

[**Diagram przypadków użycia** 14](#_Toc510563854)

[**Mockup** 14](#_Toc510563855)

[**Git** 14](#_Toc510563856)

[**Tworzęnie aplikacji** 14](#_Toc510563857)

[**Front-end** 15](#_Toc510563858)

[**JavaScript** 15](#_Toc510563859)

[**CSS** 15](#_Toc510563860)

[**Back-end** 16](#_Toc510563861)

[**Npm pakiety** 16](#_Toc510563862)

[**body-parser** 16](#_Toc510563863)

[**connect-flash** 17](#_Toc510563864)

[**ejs** 17](#_Toc510563865)

[**express** 17](#_Toc510563866)

[**express-fileupload** 18](#_Toc510563867)

[**express-session** 18](#_Toc510563868)

[**method-override** 18](#_Toc510563869)

[**moment** 18](#_Toc510563870)

[**mongoose** 19](#_Toc510563871)

[**node-geocoder** 19](#_Toc510563872)

[**nodemon** 19](#_Toc510563873)

[**passport** 19](#_Toc510563874)

[**passport-local** 20](#_Toc510563875)

[**passport-local-mongoose** 20](#_Toc510563876)

[**Routing** 20](#_Toc510563877)

[**Baza danych - Mongoose NONSQL database** 20](#_Toc510563878)

[**Relacji pomiędzy tabelami** 21](#_Toc510563879)

[**Autoryzacja** 21](#_Toc510563880)

[**Authentyfikacja oraz Middleware** 21](#_Toc510563881)

[**Kroki wykonywane przy tworzeniu projektu** 21](#_Toc510563882)

[**Sprawdzenie działania aplikacji** 24](#_Toc510563883)

[**Demonstrowanie alikacji i jej działania** 24](#_Toc510563884)

[**Napotykane problemy** 24](#_Toc510563885)

[**Możliwości do przyszłego rozwinięcia aplikacji** 24](#_Toc510563886)

[**Zakończenie** 26](#_Toc510563887)

1. **Wstęp**

W raz z wynalezieniem Internetu świat się zmienia i się rozwija coraz szybciej. Cały świat jest powiązany z każdą jego częścią za pomocą połączenia Internetowego. Jednocześnie z tym w każdym momencie mając dostęp do sieci również masz dostęp do wszystkich jego serwisów ułatwiających życie. Komunikacja od czasów jej wynalezienia w życiu codziennym odgrywa najważniejszą role połączając ludzi. Do wynalezienia Internetu ludzie komunikowały się najpierw ustnie później pisemnie i już mogły przekazywać istotną dla nich informację na odległość. Progres ze swojej kolei w metodach przekazywania informacji wystąpił odnośnie nie tak dawno licząc od początków historii. I teraz mając za sobą podgruncie komunikacja ma możliwość łączyć nie tylko osobnych ludzi, ale i kompanie i nawet urządzenia (Internet of things). Praca i byt codzienny coraz większej ilości osób i przedsiębiorstw jest związany za pomocą tego cuda — globalnego systemu połączeń sieciowych. Jednocześnie z tym ilość uzależnionych od Internetu też co raz rośnie, ale to nie jest niczym dziwnym, kiedy idziesz w nogę z czasem. Komputery, laptopy, telefony, zegarki, stacji gier, konsole i wiele innych rzeczy komunikują się z innymi rzeczami czy nawet serwerami i ilość danych codziennie rośnie.W dniach dzisięjszych żeby dostać to czego chcesz nie potrzebujesz dużo. Również jak i komunikowanie sprawa wymiany czy póżniej sprzedaży rzeczy lub usługi pojawiła się bardzo dawno a teraz ma swoj kolejny pik. Teraz żeby kupić rzecz nie koniecznie musisz iść gdzieś i jej szukać, wybierać czy porównywać. W ostatnich latach się rozwija sprzedaż za pomocą Internetu. Sklepy, serwisy internetowe, salony, fabryki i temy podobne rzeczy można odnaleść na stronach WWW (World Wide Web).Co raz więcej towarów i usług dostępne w postaci elektronicznej. Dość prosty i zrozumiały z ekonomii wyraz: popyt powoduje propozycję. Tak i z serwisami internetowymi. Popularyzacja serwisów internetowych powoduję powiększenie ich ilości. Prosty i zrozumialy wygląd tranzakcji znajomy wszystkim ze swiatu rzeczywistego został przeniesiony do Internetu. Teraz każdy użytkownik może występować jako sprzedawca czy jako konsument. Ze swojej strony to powoduję przyrost serwisów w każdej branży i dziedzinie.

Dla przykładu, żeby teraz dostać się z punktu A do punktu B wystarczy pobrać czy otworzyć mapę i jechać za marszrutom z listy proponowanych. Albo wyszukać i wybierać samochód co jedzie za potrzebnym tobie kierunkiem i umawiać się z kierowcem, żeby oczekiwał, że pojedziesz z nim, kiedy i gdzie jest z min spotkanie. Jeździć że przy pomocy kolei też nie jest tak trudno. Dane o przyjazdach, wyjazdach i cenach są dostępne każdemu i w każdej chwili w Internecie. Loty samolotem też są podobne. Wpisując miasto wylotu oraz docelowe zostaniesz proinformowany o rejsach i kierónkach, za jakimi oni lecą. Kupujesz bilet i jesteś gotowy do podróży – wsiadasz i lecisz. Na podstawie tych danych teraz działają biura podróży. Mając dostęp do danych o wylotach czy odjazdach i przyjazdach busów, czy innych środków transportu wybierają skąd i dokąd i rezerwują hotel dla wybranej osoby na wybrany okres czasu. Czyli podają informacje hotelowi, który może się znajdować na przeciwnej stronie kuli ziemnej. I oni mając taki komunikat już oczekują że w czasie rezerwacji przyjedzie wybrana ilość osób.

Są również serwisy, na których można zarezerwować mieszkanie w określonym terminie albo orędować pokój. Jest to dość prostym i użytecznym kiedy planujesz sobie wyjazd do innego miasta. Takie serwisy są dobrą zamianą rezerwacji hotelów, kiedy nie planujesz albo nie masz możliwości wydawać dużo za nocleg w innym mieście czy państwie. Ale czy są serwisy pozwalające do rezerwacji kulturalnej programy wycieczki. Jeżeli porównywać z hotelami to są serwisy gdzie być mógł to zrobić ale na przykład jako dodatek od biura podroży. Serwis, na którym można by było samemu powiedzieć czego szukasz i zaplanować to wspólnie z kimś albo dołączyć się do kogoś w już zaplanowanej podróży, czy zarezerwować sobie wycieczka po mieście u tego, kto ją proponuje jeszcze niema.

**Cel pracy**

Celem pracy byłoby stworzyć serwis dla pomocy podrożującym, na którym oni by mogli dodawać ogłoszenia o swoich planach na wyjazd, do którego potrzebna jest większa kompania lub dodania komentarzy w odpowiedź do danego ogłoszenia.

Praca w swoją kolej byłaby podzielona na stworzenie stron Internetowych do danego serwisu oraz web serwisu, po którego stronie wykonywałyby się działania na stronach. Serwis powinienen zawierać możliwości logowania, dodawania ogłoszeń, edytowania ogłoszeń i ich usuwania przez osobę co ich dodała. Komentarze również powinni mieć podobną funkcjonalność. Strony Internetowe powinne być responsywne i mieć prostą strukture, żeby użytkownicy się w niej nie gubili oraz żeby strona nie była przepęłniona kontentem, który byłby trudnym w ogarnięciu (nie byłby niezbędnym) dla wykonania konkretnego działania na stronie (User Experience).

Dany serwis jest punktem wyjściowym. Każdy, kto chce znaleźć grupę lub dołączyć się do grupy znajdzie swoją okazję na All X Travel, który im w tym pomoże. Wszystko, co jest potrzebne, dla tego jest w jednym jedynym miejscu. Każdy może odkryć dla siebie wszystkie grupy, podróży lub utworzyć własne.

Osoby co chcieliby rozpocząć podróż, ale szukają kampanii znajdą jej tutaj, ponieważ mogliby napisać o swoim docelowym planie i dodać X na mapie i czekać, aż inni poszukiwacze przygód spełnią ich marzenie.

Znaleźć grupę z kierunkiem i marszrutem podróży, jakim chcesz jest to bardzo łatwo, po prostu potrzebno by było napisać do twórcy podróży i innych użytkowników swój komentarz lub napisać bezpośrednio na maila autora czy skontaktować się telefonicznie. Użytkowniki mogliby grupować się razem, razem podróżować światem do ich pragnienionego i wymarzonego X z All Xem.

**Uzasadnienie wyboru tematu pracy**

Potrzebności na rynku rosną coraz szybciej. Dowodem tego jest każdy popularny produkt. Czasami, żeby dostać produkt jednym z pierwszych, ludzie stoją w kolejce dniami i nocami. Zapotrzebowanie na rynku jest szybko rozwijające się i żeby być pierwszym — potrzebno cały czas myśleć na przód. W tym celu został stworzony serwis dla podróżujących gdzie użytkownicy by moli znaleźć kogoś do wspólnego wyjazdu czy przewodnika po mieście, do którego jadą, który byłby bardzo przydatny podróżującym. W drugą stronę ludzi by mogli sami dodawać ogłoszenia i proponować innym się do nich dołączyć. W danym momencie rynek jeszcze nie ma podobnego produktu w swojej branży, temu on by był wybrany jako temat mojej pracy dyplomowej.

Przed rozpoczęciem danej pracy ja (Oleksand Lisivka) miałem doczynienie ze stronami Internetowymi podczas praktyk zawodowych od uczelni. Pod czas danych praktyk nauczyłem się i pogłębiłem poznania w części Front-end’owej, czyli z HTML5, CSS (SCSS, SASS, Bootstrap), JavaScript (jQuery). Odnośnie do wiedzy niezbędnej do napisania części serwerowej w wybranym przeze mnie języku (JavaScript w środowisku Node.js), to takiej wiedzy nie posiadałem. Na uczelni nauczyłem się języka C# i wzorzeca architektoniczny służącego do organizowania struktury aplikacji MVC i mógłbym wykonać pracę na podstawie danej więdzy, ale wybrałem inny język, żeby ciągle uczyć się czegoś nowego, który i aby się rozwijać, ponieważ w dziedzinie Technologij informacyjnych języki, techniki, standardy, i technologie ciągłe się zmieniają i potrzebno iść razem z nimi. Typ bazy danych (NONSQL) był wybrany z tego samego powodu — żeby powiększyć wiedzę profesjonalną.

**Zakres pracy**

Cały system można podzielić na części, które muszą być zrealizowane w kolejności, w której muszą być dodawane do serwisu:

1. REST Routing
2. Tworzenie stron
3. Tworzenie ogłoszenia
4. Konfigurowanie bazy danych i podpinanie jej do aplikacji, żeby dać możliwość zapisywania danych do niej
5. Dodawanie komentarzy
6. Logowanie
7. Autentyfikacja
8. Edytowanie lub usunięcie ogłoszenia przez go właściciela
9. Edytowanie lub usunięcie komentarzy przez ich właścicieli
10. Wyświetlenie komunikatów dla użytkownika oraz rozszerzenie istniejącego funkcjonału
11. Responsywność stron do różnych rozszerzeń ekranu.
12. Podtrzymywanie różnych wersji przeglądarek internetowych.

Dana lista wskazuje na kolejność realizowanych zadań ze względu na ich ważność dla realizacji danego projektu. Idąc z góry do dołu — tworzymy system od szkieletu do gotowego produktu.

Na przykład, żeby każdy użytkownik mógłby utworzyć ogłoszenie, on musi najpierw przejść na odpowiednią stronę i się zalogować. Z punktu widzenia serwisu bez logowania dla wykonania żądania do serwera z informacją o ogłoszeniu użytkownik musiałby przejść się przez 4 pierwsze części z listy, czyli otworzyć stronę pod potrzebnym adresem (REST Routing). Po tym użytkownikowi musi wyświetlić się strona z treścią, która odpowiada adresowi (Tworzenie stron). Użytkownik tworzy odpowiednie żądanie do serwera z treścią ogłoszenia i po stronie serwera po przetwarzaniu danych musi się utworzyć ogłoszenie (Tworzenie ogłoszenia oraz Konfigurowanie bazy danych i podpinanie jej do aplikacji, żeby dać możliwość zapisywania danych do niej). Dalej, dla rozszerzenia funcjanalności, dodajemy po stronie serwera odpowiednie ścieżki routingu i strony co by się renderowały po przejściu na nich dla dodawania komentarzy i ścieżki do oczekujące na wchodzące od użytkownika dane o komentarze (Dodawanie komentarzy). Więcej szczegółowa informacja zostanie podana w kolejnych rozdziałach pracy.

**Omówienie układu pracy, krótka charakterystyka poszczególnych rozdziałów**

Praca dla łatwości jej przetwarzania zostanie podzielona za treścią na części (np. część wstępu, część główna pracy, zakończenie, literatura, streszczenie pracy, załączniki.). Wstęp oraz część główna zostaną podzielone na działy (np. w przypadku części głównej na część teoretyczną oraz część praktyczną). Żeby działy ze swojej kolei zostały się więcej formatowanymi, żeby odrazu wiedzieć do czego należy część pracy, w wyszczególnionych poddziałach zostałaby podana szczególniejsza informacja (np. część front-endowa część back-endowa).

Charakterystyka nie potrzebuję szczególnego omówienia z tego powodu, że o treści każdej z części byłoby wiadomo odrazu z miejsca, w którym ona się znajduje (np. Część teoretyczna -> Front-end -> Html5 miałaby w sobię teorije o HTML5, co się odnosi do części front-endowej).

**Rozdział 1. Część teoretyczna pracy dyplomowej**

Poniżej są opisane technologie oraz narzędzia, które zostały użyte przy napisaniu danej pracy dyplomowej. Niektóre są wymagane dla jej realizacji (np. HTML5 jest teraz standardem w napisaniu stron internetowych i pisać stronę dla przykładu na tabelach nie byłoby dotrzymaniem się standardów) a niektóre zostały wybrane ze wzgłendów na łatwość w użyciu oraz możliwości, które daję dane narzędzia, ciągle wsparcie i rozwijają nie (np. Bootstrap, jQuery czy npm).

Dwie najważniejsze części projektu to oczywiście front-end i back-end. Front-end odpowiada za komunikowanie się z użytkownikiem, pobieranie danych od użytkownika, czy ich wyświetlenie oraz przekazanie ich do back-endu. Back-end centrem w działaniu aplikacji, ponieważ w nim na podstawie danych wykonują się określone zadania. To nie znaczy że front-end jest mniej ważny. Dużo czego można zrobić za pomocą JavaScripu, ale żeby nikt nie miał na to wpływu i żeby to było chroniono od innych - rzeczy ważniejsze są zaimplementowane po stronie serwera. Front-end w tej części występuję jako graficzna reprezentacja dla aplikacji konsolowej. Taki typ podzielenia obowiązków set dość typowym i popularnym pośród oprogramowania danego typu. Tak podzielona praca daje osobie tworzącej stronę wolne ręce i jednocześnie dana osoba nie koniecznie musi znać cech serwera i co się na nim dzieje, a programista serwera nie potrzebuje znajomości HTML i CSS, bo zajmuję się nie wyglądem a tylko funkcjonalnością.

**Front-end**

Fornt-end jest częścią aplikacji, która odpowiada za komunikację z użytkownikiem przy pomocy interfejsu graficznego. Dość często front-end jest używany jako pośrednik pomiędzy użytkownikiem systemu a aplikacją na serwerze. W systemach takiego typu front-end posługuje dla wyświetlenia informacji oraz pobierania informacji od użytkownika. Dość proste strony czy np. e-maili mogą nie mieć serwera, czy w przypadku e-maili nawet skryptów. Takie strony wykorzystywane do reprezentacji danych. Są również i typy stron pomiędzy dwoma powyższymi. Takie strony nie są zbyt skomplikowane, żeby niezbędny był serwer, ale i nie są statyczne. Takie strony napisane są przy pomocy języka JavaScript dla dodawania interaktywności na strony. Więcej szczegółowo każdy z elementów jest opisany poniżej.

**HTML5**

HTML albo ang. HyperText Markup Language jest językiem znaczników do markowania, strukturyzacji i prezentowania treści w Internecie. Cyfra 5 w nazwie wskazuję na aktualną wersję – piątą, dlatego wykorzystanie języku ze standardami ostatniej wersji zostało nazwane HTML5, co połączyło w sobie normy i standardy HTML oraz XHTML. HTML był stworzony, żeby opisać strukturę informacji zawartych wewnątrz strony internetowej w postaci modelu obiektowego (DOM). HTML z tekstu, który ma buduję hiperłącza, akapity, nagłówki i temu podobne rzeczy. Bez dodawania CSS strona z otwierających i zamykających się tagów ma swoją strukturę, ale nie ma żadnego wyglądu. Jest to „ściana tekstu” dla maszyny, żeby ona wiedziała, kiedy jedna go część się kończy, a kolejna się zaczyna.

Struktura modelu DOM (ang. Document Object Model) przedstawia dokument HTML w postaci struktury drzewa. Model DOM definiuje obiekty i właściwości wszystkich elementów HTML oraz metody dostępu do nich. Korzystając się z DOMu, przy pomocy JavaScriptu, można łatwo dodać, zmienić lub usunąć element.HTML.

**CSS3**

Kaskadowe arkusze stylów (Cascading Style Sheets) jest językiem służącym do opisywania wyglądu, czyli wyświetlenia strony internetowej. CSS jest listą regul, które wskazują przeglądarce, w jaki sposób dana strona musi być wyświetlona i jak wyglądają wybrane elementy. CSS jest używany dla określenia kolorów, czcionek i innych aspektów wizualizacji stron internetowych. Był on stworzony, żeby trzymać osobno strukturę dokumentu a jego stylowanie. Z tego powodu strony, na których stylowanie jest wpisane liniowo do kodu HTML, muszą być dawno napisane i tego się nie zmienia, żeby trzymać kod jednolitym. Czytać takie strony staje trudno, kiedy przykładowo na jedną linię HTML przychodzi się po 3 stylowania. Również użycie klas pomogłoby trzymać kod suchym i niepowtarzającym się. Osobny plik ze stylowaniem może być użyty na różnych stronach, co ułatwia o wiele ich napisanie.

**Bootstrap**

Bootstrap[[1]](#footnote-1) jest frameworkiem dla CSS, stworzony przez programistów Twittera. Główną cechą bootstrapu jest responsywność oraz już gotowe klasy, które w łatwy i szybki sposób pomagają przestylować stronę internetową. Bootstrap napisany jest w taki sposób, żeby strona również dobrze wyglądała w większości najpopularniejszych przeglądarek, takich jak Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Opera i Safari. Bótstrap jest łatwy w użyciu i pomaga zaośczędzić dużo czasu na stulowaniu stron. Dla działania danego frameworku niezbędne jest wykorzystanie biblioteki jQuery. Dodając trzy referencje do swojego kodu (pierwsza do stylowania, druga do jQuery i trzecia do scryptów Bootstrapowych) do pobranych plików lub z CDN serwisów można używać go klas w każdej chwili.

**JavaScript**

JavaScript jest wieloplatformowym obiektoorijentowanym językiem skryptowym i jest trzecim, ale nie za ważnością tytanem front-endu. JavaScript jest bardzo wielofunkcojalnym, poniważ jest wykorzystywany w przeglądarkach, po stronie serwera (NodeJS), mobilnych ta desktopowych aplikacjach. W przeglądarkach JavaScript występuję jak na stronach tak i w web rozszerzeniach. Strona bez wykorzystania JS wygląda statyczną i niezmieniającą się, ale z dodawaniem JS wszystko się zmienia. Dla przykładu JS jest używany do stworzenia scenariuszy, za którymi strona się zachowuje, co daje możliwość na stronie współdziałać z końcowym użytkownikiem na go urządzeniu. Również dobrze i często wykorzystywany do walidowania formularzy na stronie, wymieniać się danymi z serwerem, zmieniać strukturę oraz widoki stron.

Czasami przy stylowaniu strony potrzebno wybrać jakieś elementy ze strony, ale stylowanie przez CSS nie daje takich możliwości do przestylowania ich i JS jest jedynym wyjściem. Z drugiej strony tak jak HTML CSS oraz JavaScript są różne za swoim znaczeniem, warto by trzymać ich osobno, o ile się to daję. Niestety nie wszystkie przeglądarki działają jednakowo i temu, jak i niektóre stylowania tak i niektóre skrypty mogą nie działać wszędzie, temu przy napisaniu kodu warto uważać na przeglądarki i ich wersje. Większość problem można rozwiązać prototypowaniem (JS jest językiem z koncepcją prototypów, temu podtrzymywanie obiektów jest takie samo i się nie różni od tradycyjnych języków OOP) Takie problemy są z czasem rozwiązywane poprzez standaryzację.

Standardem ostatniej wersji JS jest ECMAScript wersji 6 . ECMAScript jest ustandaryzowany przez ECMA obiektowoskryptowy język programowania. Specyfikacja go znana jest jak ECMA-262. ECMA-262 definiuje semantykę języka ECMAScrpt (którego implementacją jest JavaScript oraz inne), oraz np. podstawowe typy danych (String, Boolean, Number, Object itd.) i obiekty (np. Math, Array).

**JQuery**

JQuery[[2]](#footnote-2) jest szybką, niewielką i bardzo popularną biblioteką języka JavaScrip. Powiedzieć popularna przy określeniu może nie wystarczyć, ponieważ jQuery jest najpopularniejsza biblioteką języka JavaScript i wykorzystuję się na ponad połowie najpopularniejszych stron internetowych. Pomaga ona w wybieraniu obiektów DOMu oraz go manipulacją. Animacja, tworzenie eventów i ich obsługa jest też o wiele łatwiejsza z jQuery. Ilość kodu napisanego z jQuery jest mniejszą, temu i łatwiejsza w go czytaniu. Nie koniecznie zawsze potrzebno używać jQuery, ale licencja MIT pozwala każdemu używać danej biblioteki za darmo w każdej chwili.

**Back-end**

Back-end na przeciwieństwo front-endowi nie działa po to, żeby kontaktować się z użytkownikiem tylko po to, żeby generować kod maszynowy dla procesorów z różnymi przeznaczeniami. Front-end występuje pośrednikiem pomiędzy back-endem a użytkownikiem końcowym, temu programista części front-entedowej jest odpowiedzialny za wyróżnienie aplikacji wykonanej przez programistę części back-endowej. Dla przykładu mając do porównania dwa serwisy internetowe dla sprzedaży, back-endowe części mogłyby mieć więcej wspólnego pomiędzy sobą, w kodzie źródłowym, aniż front-endowe. Oczywiście back-end mógłby być napisany w różnych językach i z użyciem różnych platform, ale funkcjonalność napisana po stronie back-endu miałaby większe podobieństwo aniż wygląd po stronie front-endu. Tak jak front-end dostępny każdemu użytkownikowi jakiejś strony to back-end odpowiadający za działalność całej aplikacji i jej funkcjonał jest ukryty od większości. Przyczyn na to jest wiele od tego, że korzystając się ze strony, nie potrzebujesz wiedzieć, co się dzieje „pod spodem” takiej strony, żeby nadał normalnie jej używać, kończąc polityką bezpieczeństwa. Końcowy użytkownik nawet nie zawsze może wiedzieć, że coś się zmieniło po stronie serwera, dla przykładu, kiedy w 2011 Twitter zmienił swoją część back-endową z Ruby on Rails na Java końcowe użytkownicy, mając dotyczenie tylko z front-endem nawet nie zauważyli różnicy.

Programowanie back-endowe to programowanie, którego celem jest implementacja strony serwera integracja bazy danych i połączenie jej ze stroną użytkownika (front-end). Mówiąc najprościej, front-end przesyła informacje i polecenia od użytkownika do serwera, który z kolei je przetwarza.

**Node.js**

Node.js[[3]](#footnote-3) jest to środowisko uruchomieniowe zaprojektowane do tworzenia aplikacij internetowych napisanych w języku Javascript po stronie serwera. Działa ona na Chrome V8 silniku z otwartym kodem od Googla który również jest wykorzystywany w Google Chrome. Jest to szkielet do budowy aplikacji. Jako Framework albo platforma programistyczna definiuje on bazową strukturę aplikacji oraz ogólny mechanizm jej działania. Node.js ma w sobie zestaw komponentów i bibliotek do wykonywania – npm (node pachage manager). Programista, tworząc aplikację, dostosowuję już gotowe pakiety do wymagań realizowanego projektu, tworząc w ten sposób gotową aplikację.

Node.js ma architekturę sterowaną zdarzeniami, zdolną do asynchronicznych operacji wejścia / wyjścia (I / O). Taka architektura ma na celu optymalizację skalowalności i efektywności w aplikacjach internetowych z wieloma operacjami wejścia / wyjścia, również w aplikacjach internetowych działających w czasie rzeczywistym.

Badania[[4]](#footnote-4)[[5]](#footnote-5) robione przez Stack Overflow pokazują, że w 2017 za rezultatami danych badań JavaScript jest najpopularniejszym językiem programistycznym, a w kategorii Platformy, biblioteki oraz inne technologie wygrywa Node.js. Czyli Node.js aktualnie jest najpopularniejszą wśród deweloperów technologią. JavaScript przy tym, że co roku jest coraz więcej popularniejszym, wspiera tym samym i Node.js w go popularności. Netflix. PayPal, Yahoo!, Walmart i wiele mniej wiadomych w strefie technologii firm wykorzystują Node jako korporacyjną platformę internetową.

JavaScript, występując w większości aplikacji internetowych, jest bardzo popularnym. Z tego powodu uniwersalność kodu ze względu na technologie jest ważnym plusem korzystania z JS i na serwerze, o którym należy pamiętać.

Pojawienie się asynchronicznej oraz oczekującej konstrukcji (async / await) całkowicie zmieniło podejście do pisania asynchronicznego kodu, bo taki kod za swoim działaniem przypomina zwykły kod synchroniczny, zarówno pod względem wyglądu, jak i zachowania. Pomaga to w rozwiązaniu problemu zwanego callback hell, kiedy w oczekiwaniu na wykonanie żądania musimy napisać kod, który zarówno przewidywuję porażkę oraz powodzenie żądania, i o ile więcej jest takich żądań asynchronicznych jedna po drugiej, tym więcej rośnie i gromadzi się kod związany z ich obsługa.

**Npm**

Node.js ma domyślny manager pakietów napisany całkowicie w języku JavaScript dla używania przez każdego w każdej chwili. Npm[[6]](#footnote-6) jest największym na tę chwilę zgromadzeniem pakietów w sieci Internet. W tym momencie są dostępne ponad 470 000 bezpłatnych pakietów. Taka ilość pakietów, kiedy chcesz zbudować pierwsze aplikacje, może przeszkadzać, bo wybór każdego komponentu do swoich potrzeb i do dopasowania ich z innymi komponentami też potrzebuję czasu.

Żeby, dla przykładu, zbudować REST API serwis z routingiem potrzebny jest jeden dwa pakiety. Im więcej funkjonalności jest zapotrzebowano tym więcej pakietów z gotowymi funkcjami zostają dodane.

**REST API**

RESTfull Web Api (Applikation Programming Interface)[[7]](#footnote-7) jest aplikacją, która może być manipulowana przy użyciu protokołu HTTP i formatu JSON lub XML. Pomaga to, żeby powiązać między sobą różne części oraz różne aplikacje. Web api pomaga w kontaktowaniu się web interfejsom i aplikacjom pomiędzy sobą. Protokół HTTP zawiera metody dla manipulacji typu: „GET”, „POST”, „PUT”, „DELETE”, które są najważniejszymi. Architektura REST odnośnie do której była zrobiona dana praca, bazując się na danych metodach, pozwala na manipulowanie poprzez routing.

Roouting wygląda następująco:

|  |  |
| --- | --- |
| Akcja | HTTP Metod |
| INDEX | GET |
| SHOW | GET |
| CREATE | POST |
| NEW | GET |
| EDIT | GET |
| UPDATE | PUT |
| DESTROY | DELETE |

Każda akcja ma swoje znaczenie, które jak można się domyślić z nazwy, odpowiada za:

Wyświetlenie wszystkiego, konkretnego egzemplarza, tworzenie nowego obiektu, pokazywanie formy dla do tworzenia, pokazywanie formy dla zmiany obiektu, go zmiana i ostatnie – go usunięcie.

**Baza danych**

Bazą danych jest zbiór danych zapisany za pełnymi regułami. Jest to gromadzenie danych i istnieje to od dawna: przepisy osób do obremu mieszkaniowego czy liczenie towaru, co został dostarczony brygadą morską do portu, ma dużo wspólnego, a mianowicie regulowanie tych danych. Mając np. liste produktów dostarczonych moglibyśmy zobaczyć nazwe produktu, ilość może jeszcze cenę, datę dostawy i skąd pochodzi, i tak dalej. Zapis danych z pojawieniem komputerów się zmienił i nie są to już książki czy osobne kartki tylko dane w postaci cyfrowej. Również dobrze można by było zapisywać dane i na żółtych karteczkach tylko czy jest to efektywne?

Pojawienie się baz danych, tabel i połączeń pomiędzy nimi ułatwia późniejszy dostęp do tych danych. Np. każdy rekord, czyli pojedynczy element ma 4 pola przechowujące dane o użytkowniku — imie, nazwisko, adres i miasto. Tworzenie osobnej tabeli z miastami standaryzowałoby wybór miasta i późniejsze odczytanie danych np. wszystkich mieszkańców miasta za wyborem za danym kryteriom.

**Relacji pomiędzy danymi**

Relację pomiędzy tabelami mogą być 2 typów, jeden do jednego albo jeden do wielu. Różnią się połączeniami pomiędzy danymi. Np. do jednej osoby jest przypisany jeden numer indefikacyjny płatnika i również pod jednym numerem płatnika istnieje tylko jedna osoba (1 - 1) albo inny przykład jedna osoba jest przypisana do jednego miasta, ale w danym miastu jest dużo mieszkańców (1 - \*).

**Git oraz Github**

Git jest systemem kontroli wersji, który ułatwią życie każdemu, kto się z niego korzysta. Git pomaga zapisywać wszystkie wersje plików z odniesieniem do każdej zmiany w pliku. W każdym momencie maż dostęp do każdej swojej, potwierdzonej zapisywaniem do gitu, zmiany wprowadzonej do plików. Programista, mając problemy z kompilowaniem kodu, poprzez wprowadzony błąd, może zobaczyć jakie zmiany zostały wprowadzone do danego kodu. Pracując w zespole git, mając odgałęzienie, pomaga tworzyć niezależne części projektu wspólnie, a przy zapisywaniu i wysyłaniu zmian każdy by mógł je odebrać i mieć do użycia. Później gałęzięnia można połączyć razem albo cofnąć zmiany w kodzie do poprzedniej wersji. Użycie gitu pozwala łatwo śledzić za zmianami, co się zdarzyło w plikach i ich nazwach.

Github jest usługą internetową, co działa na podstawie systemu kontroli wersji, za pomocą której można przechowywać i łatwo się dzielić repozytoriami z innymi. Można na nim tworzyć prywatne oraz publiczne projekty, do których mieliby dostęp albo wybrane osoby, albo wszyscy chętni.

**Visual Studio Code oraz Sourcetree**

Visual Studio Code jest dystrybuowany bezpłatnie i jest dostępny w wersjach dla platform Windows, Linux i OS X. Jest wykorzystywany do napisania kodu i był wybrany ze względu na wygodność oraz terminal wbudowany do programu. Wykorzystywanie wbudowanego terminalu pozwalało na efektywniejsze wykorzystanie czasu przy każdym odpalaniu serwera, po zmianach do niego wprowadzonych. Mięć osobno terminal a IDE było nie zbyt wygodnym, bo potrzebowało ciągłego przełączania się pomiędzy nimi.

Edytor również zawiera wbudowany debugger, narzędzia do pracy z Git, który nawet pokazuję zmiany od ostatniego commitu, narzędzia do refaktoryzacji, nawigację po kodzie, autodopęłnienia typowych konstrukcji i wskazówki kontekstowe. Produkt wspiera rozwój platform ASP.NET i Node.js i jest pozycjonowany jako lekkie rozwiązanie, które pozwala na pracę bez w pełni zintegrowanego środowiska programistycznego. Wśród obsługiwanych języków i technologii: JavaScript, C ++, C #, TypeScript, jade, PHP, Python, XML, Batch, F #, DockerFile, Coffee Script, Java, HandleBars, R, Objective-C, PowerShell, Luna, Visual Basic, Markdown JSON, HTML, CSS, LESS i SASS, Haxe.

Sourcetree jest darmowy klient Git dla systemów Windows i Mac. Upraszcza on sposób interakcji z repozytoriami Git. Poprzez prosty Git GUI pomaga w wizualizacji i zarządzaniu swoimi repozytoriami. Sourcetree jest narzędziem do wprowadzenia zmian do repozytorium gitu na Githubie czy innych aplikacjach pomagających w tym.

**Część praktyczna**

Poniżej jest opisany przebieg realizacji praktycznej części pracy dyplomowej, który by zawierał opisanie specyfikacji wstępnych do programu, tworzenie repuzutoriumu, tworzenie aplikacji z podziałem na części front-endowe oraz back-endowe, demonstrowanie aplikacji i jej działania oraz omówienia napotykanych problemów i możliwych dodatków dla rozwinięcia aplikacji.

**Specyfikacja wymagań**

W tym rozdziale opisane są wymogi do serwisu, który zostanie zaprogramowany oraz przypadki użycia danego serwisu.

**Wymagania biznesowe**

FILL ME.

**Wymagania funkcjonalne**

Wymagania funkcyjne określają zakres możliwości oraz operacje, które dany serwis musi posiadać dla wykonania. Wymagania funkcjonalne muszą określić, w jaki sposób system realizuję postawione cele oraz warunki, przy których dana funkcjonalność musi istnieć. Zdefiniowanie i opisanie wymagań funkcjonalnych pomaga programistom zrozumieć zadanie do zaprojektowania.

Dany web serwis musi posiadać:

1. Zalogowani użytkownicy muszą mięć możliwość tworzenia ogłoszeń;
2. Każdy użytkownik musi mieć możliwość zalogowania się;
3. Dane podawane użytkownikami przy logowaniu tworzeniu i edytowaniu czegokolwiek muszą być zapisane w bazie danych;
4. Każdy użytkownik może zostawiać komentarze do ogłoszeń;
5. Ogłoszenia od niezalogowanych użytkowników muszą być wyróżnione w jakiś sposób;
6. Każde ogłoszenie musi mieć możliwość edytowania i usunięcia;
7. Każdy komentarz musi mieć możliwość edytowania i usunięcia;
8. Tylko zalogowani użytkownicy muszą mieć możliwość edytowania i usunięcia ogłoszenia i komentarza;
9. Tylko twórca ogłoszenia musi mieć możliwość edytowania i usunięcia ogłoszenia;
10. Tylko twórca komentarza musi mieć możliwość edytowania go;
11. Tylko twórca ogłoszenia i twórca komentarza muszą mieć możliwość usunięcia komentarza;
12. Niezalogowani użytkowniki nie mogą edytować czy usuwać komentarzy przez nich stworzone i nie dostają możliwości edytowania, czy usunięcia anonimowych komentarzy nawet po zalogowaniu.

**Wymagania niefunkcjonalne**

Wymagania niefunkcjonalne – są to warunki, które nie stosują się funkjanalności, ale również muszą być spełnione i opisują ograniczenia dla systemu, przy zachowaniu których system powinien nadał prawidłowo realizować swoje założenia ta funkcje.

1. System ma być uwzględniający i przewidywający możliwe błedy (error handling) i mieć ich obsługę, żeby nie było to krytycznym w działaniu serwera i nie przerywało go działanie;
2. System musi starać się zapewniać dostępność 24 godziny 7 dni w tygodniu przez cały rok oprócz czasu niezbędnego dla aktualizacji, możliwej naprawy oraz testowania systemu;
3. Baza danych do wykorzystania w systemie jest MongoDB typu NonSQL;
4. Pakiety wykorzystywane z npm mogą być dowolne, aby tylko kompatybilne ze sobą i łatwe w użyciu i obsłudze;
5. System musi korzystać z następujących technologii oraz standardów:
   1. WebAPI;
   2. RESTfull Web serwis;
   3. Npm
   4. MongoDB;
   5. Bootstrap;
   6. JQuery
6. System musi być user friendly i być zrozumiałym oraz łatwy w użyciu każdemu użytkownikowi.
7. System powinien wydawać komunikaty użytkownikowi, żeby użytkownik wiedział co się dzieje z systemem;
8. Kod musi być DRY(Don't Repeat Yourself) o ile się da, żeby był czytelny oraz łatwy w modyfikacji i być zoptymalizowanym ta minimalistycznym;
9. Username użytkownika musi być unikalnym;
10. Do każdego ogłoszenia niezbędne są następne dane:
    1. Stworzono przez;
    2. Kiedy stworzono;
    3. Nazwa podróży jest wymagana;
    4. Teks opisujący podróż jest wymagany;
    5. Cena podróży jest wymagana;
       1. Cena podróży musi być do .01;
       2. Cena podróży musi być w zakresie od 0 do nieskończoności;
    6. Obrazek jest wymagany;
    7. Użytkownik nie może wprowadzić do system żadnych plików niebędących obrazkami;
    8. Dozwolone rozszerzenia obrazku to .png .jpeg .jpg;

**Diagram przypadków użycia**

FILL ME

**Mockup**

FILL ME

**Git**

Przy rozbudowaniu kodu dla wygodności można korzystać się z gita. Daje to możliwość dość prosto panować nad wersjami kodu i mieć z głowy dublowanie plików po to, żeby zmiany wprowadzone ni zepsuły już działającego kodu bez możliwości cofania się. Z tego powodu przy tworzeniu danego projektu był używany git. Po stworzeniu plików i żeby dodać ję do repozytoriumu używynę są następujące polecenia[[8]](#footnote-8)

* git init – inicjalizuje, że w tym folderu będzie wykorzystywany git;
* git status – zobaczyć status, co jest zrobiono;
* git add – dodać wszystkie pliki do systemu kontroli wersji;
* git status;
* git commit –m „first commit into git, adding files to git” – zapisywanie plików do gitu gdzie –m oznacza, że zostanie dodany tekst commit'u;
* git status;
* git log – pokazuje commit'y danego gitu.

Żeby wysłać dane do GitHabu dodajemy jeszcze następne polecenia

* git remote add origin http://ścieżka do gitu – wskazuję na repozytorium gitu;
* git status;
* git push –u origin master – wysyła zmiany na serwer.

**Tworzęnie aplikacji**

Po planowani, zdefiniowaniu wymagań które potrzebno spełnić, założeniu gitu i instalacji środowisk do napisania kodu można zaczynać tworzyć aplikację. Poniżej opisanie został produkt końcowy oraz czasami elementy przejściowe, ponieważ kod nie był napisany z góry do dołu od razu takim, jaki został na końcu, tylko był rozwijany ewolucyjno-przyrostowo. Również opisanie zostało podzielone na części (front-end oraz back-end ewentualne podsumowania).

**Front-end**

Część front-endowa aplikacji nie jest dużą, ponieważ funkcjonalność jest zaimplementowana po stronie serwera. Rolą front-endu jest przesyłania danych do serwera oraz odebranie ich i wyświetlenie na stronie użytkownikowi.

Żeby nie tworzyć specjalnych skryptów do odbierania danych po stronie klienta (czyli front-endowej) i tym samym zmniejszyć obciążenie na stronę przy jej ładowaniu oraz ukryć od użytkownika końcowego wszystko, co nie jest niezbędnym do pokazywania, został wybrany inny metod – przekazywanie danych po stronie serwera przy renderowaniu strony. Wygląda to następującą po stronie serwera, kiedy idzie żądanie do serwera, serwer go odbiera i wykonuję swoje czynności z przygotowania danych, czyli szuka danych w bazię i np. w przypadku ogłoszenia odrazu populuję komentarzy oraz obrazek do ogłoszenia za ID komentarzów ta obrazka przypisanych danemu ogłoszeniowi i poszukiwaniu i odczytaniu komentarzy a obrazku w ich dokumentach (bazie danych odnoszącej się do nich). Po wyszukiwaniu wszystkich danych serwer w odpowiedź na żądanie renderuje użytkownikowi stronę i przekazuję do niej dane, co pobrał z bazy.

Dla łatwości napisania stron, kiedy oni zostały tworzone, był używany pakiet ejs, który pomagał w przygotowaniu plików końcowych użytkownikowi. Czyli podczas tworzenia strony populowane dane przekazane z serwera w odpowiednie im miejsca. Kod, który potrzebno było dla tego dodać do strony wyglądał w postaci:

<%= dane z serwera, które potrzebno jest dostać i pokazać w danym miejscu %>

<% kod JavaSriptu, który był wykonany przy przygotowaniu strony użytkownikowi przed jej wyświetleniem %>

Również zostały wykorzystane <% include ścieżka do pliku, który potrzebno dodać%>; dla podpinania plików. Było to wykorzystane, żeby kod był jednakowy na każdej stronie bez konieczności podmiany w każdym pliku zmian, które potzrebno by było wprowadzić oraz żeby nie dublować kodu (DRY) i trzymać ko łatwym do modyfikacji.

Jak widać na obrazkach poniżej, tak wygląda strony napisane w połączeniu z ejs, chociaż przy otwieraniu kodu już z przeglądarki (view sorce) kod już wygląda nako normalny HTML bez żadnych <% %> czy <%= %> jak w przypadku plików źródłowych a kody z plików dodawanych poprzez <% include %> wyglądają tak, jak by znajdowały się w tym pliku od początku. Przy przygotowaniu strony pliki ejs konwertują się do normalnych plików typu HTML.

FILL ME WITH IMAGES OF EJS IN CODE AND IN PAGE VIEW SORCE

**JavaScript**

Po stronie klienta nie jest dużo kodu JavaScripu, ponieważ, jak mówiłem powyżej, wszystko, co można było zrobić po stronie serwera - zostało zrobiono po stronie serwera. Temu po stronie klienta była dodana tylko walidacja.

Walidacja wygląda następującą:

FILL ME WITH IMAGES

Jest to wykorzystywane dla walidacji obrazku dodanego przez użytkownika do ogłoszenia. Typ obrazku musi być z obszaru dozwolonych, czyli .png, .jpg, jpng. Walidacja działa następującą, że przy dodawaniu obrazku trigeruję się funkcja walidacyjna, która sprawdza rozszerzenie pliku i nie pozwala dodawać plików rozszeżenie których jest inne od opisanych powyżej.

Kod walidacji można by było zamienić na JQuery validate[[9]](#footnote-9) i stworzyć walidacje dla każdego pola, a nie tylko dla obrazka, żeby użytkownik ni wpisywał danych w niekorektnej postaci, ale tak jak danych nie jest dużo i od użytkowników zależy, jeżeli oni chcą, żeby ktoś odpisał na ich ogłoszenie i miał prawidłowe dane do kontaktu z nimi jakie dane oni wprowadzają. Na danym etapie tego wystarczy, ale można by było dodać to do listy rzeczy możliwych dla przyszłego rozwoju aplikacji.

**CSS**

Każda strona musi mieć stylowanie, żeby mieć wygląd nie tylko tekstowy, ale więcej wizualny co ułatwiałoby życie. Wygląd strony zależy od kaskadowych arkuszy stylów co jej stylują w odpowiedniości do reguł napisanych w nich. Dla stylowania stron w danym serwisie był używany bootstrap, ponieważ stworzone już gotowe klasy są łatwe w użyciu oraz stylują stronę w miarę dobrze. Używa się bootstrapu poprzez podpinanie jego do kaskadowych arkuszy stylów oraz jeżeli potrzebna jest interakcja związana z bootstrapem dodawane również JQuery, oraz JavaScript bootstrapu. Dla ich używania w odpowiednich miejscach wpisywałem klasy albo inne rzeczy co już są napisane przez Twiter bootstrap. Oprócz bootstrapu dla dodawania własnych stylów podpięty był własny niewielki plik ze stylowaniem.

**Back-end**

Część back-endowa jest główną, ponieważ cały funkcjonał jest wykonany po jej stronie. O ile aplikacja była rozwijana, były dodawane nowe pakiety dla nowej funkcjonalności i nowych możliwości. W aplikacji używane są middleware dla ustaleń w aplikacji i jej działania.

Middleware jest częścią aplikacji, co pomaga w komunikowaniu pomiędzy jej częściami. Np. przed tym, jak dać dostęp użytkowniku do edytowania czy usunięcia ogłoszenia czy komentarza przy pomocy pośredniego sprawdzenia sprawdzamy, czy użytkownik ma do tego uprawienia, czy dane ogłoszenie zostało stworzone przez niego a w przypadku komentarza czy może on jest twórcą ogłoszenia.

Tak jak dana część jest dużą, została ona podzielona na mniejsze części dla opisania wykorzystanych części i czym oni posługują w danej aplikacji. Daje to, że każda część zostanie, chociaż by częściowo wyróżniona ta opisana. Później jako podsumowaniem do tego wystąpi opis toku oraz podsumowanie projektu.

**Npm pakiety**

Poniważ część serwerowa aplikacji byłą pisana w Node.js to najłatwiej było pisać używająć już gotowych narzędzi gotowych do użycia i latwych w ustaleniach. Każdy pakiet daję jakąś przygotowaną część funkjanalności. Wszystkie pakiety użyte w danej aplikacji są opisanę w package.json do danej aplikacji. Jest to plik za pomocą którego można łatwo przenieść daną aplikację w inne miejsce i tam zainstalować.

Kiedy tylko zaczyna się napisanie kodu w Node, zaczyna się ono od instalacji package.json. Komenda „npm init” zaczyna instalacje i pyta się o startowe wartości takie jak name, version, description, main, author, licence. Służą oni do tego, żeby było wiadomo czym jest dany pakiet i jakie ma dependencies. Żeby przenieść serwer po kopiowaniu plików aplikacji i package.json również potrzebno napisać „npm init” i pakiety zostaną zainstalowane.

Żeby dodać inny pakiet do aplikacji potrzebno w folderze gdzie jest package.json napisać następne polecenie „npm i –s nazwa\_pakietu”, gdzie i jest skrótem od install a flaga s odpowiada –save co dodaję dany pakiet do pliku package.json do obiektu dependencies dany pakiet.

Niektóre pakiety np. nodemon zowtały zainstalowane globalnie (-g przy instalacji) i nie zostały dodane do pliku package.

Wybrane pakiety definiują od części funkcjonalność programu. Poniżej zostaną krótko opisane pakiety używane w mojej pracy dyplomowej oraz w jakim celu oni służą.

**body-parser**

Dany pakiet[[10]](#footnote-10) pomaga w parsowaniu danych otrzymanych ze strony internetowej. Jest to middleware, który parsuje dane z żądania wysłanego do serwera i robi ich dostępnymi za pomocą właściwości body (req.body). Po podpinaniu danego pakietu dane z formularzów teraz łatwe do osiągania (req.body.name\_atribute\_from\_form).

Instaluje się poprzez:

* npm install body-parser –save

W aplikacji jest podpinany poprzez następujący kod:

* const bodyParser = require("body-parser");

app.use(bodyParser.urlencoded({

extended: true

}));

Daje to parsowanie URLów i udostępnienie ich w req.body.

**connect-flash**

Pakiet connect-flash[[11]](#footnote-11) jest specjalnym obszarem sesji służącym do przechowywania wiadomościktóre są zapisywane w pamięci flash i kasowane po wyświetleniu użytkownikowi. Czyli np. przy logowaniu użytkownikowi po wprowadzeniu nieprawidłowego hasła zastanie pokazany komunikat z wybranym tekstem (w danym przypadku, że hasło jest nie prawidłowe), ale po odświeżeniu strony dany komunikat już nie jest aktualnym i wygasa. Flash jest zwykle używany w połączeniu z przekierowaniami, dzięki czemu wiadomość jest dostępna dla następnej strony, która ma być renderowana. Deklaruje się tekst i typ komunikatu bezpośrednio przed przekierowaniem.

Instaluje się poprzez:

* npm install connect-flash –save

W aplikacji jest podpinany poprzez następujący kod:

* const flash = require('connect-flash');

app.use(flash());

req.flash('info', 'Flash mesaage text')

res.redirect('/');

Daje to, że po przekierowaniu tekst byłby dostępny poprzez req.flash (‘info’) i mógłby być przesłany do strony docelowej użytkownikowi dla wyświetlenia.

**ejs**

Ejs pakiet[[12]](#footnote-12) pomaga przy przekazywaniu danych do strony pod czas jej renderowania dla końcowego użytkownika. Działa to poprzez specjalne symbole co dają możliwość wykorzystywać JavaScript przy napisaniu stron. Dla napisania JavaScriptu oraz wyświetlenia obiektów przekazanych z serwera używane są konstrukcję typu <% JS code %> oraz <%= JS object to be render as text value %>. Również ejs daję możliwość użycia innych plików ejs jako części siębie. Czyli po dołączeniu kodu <% include ./partials/header %> dodaję plik header.ejs z folderu partials na miejsce linii z include. Pozwala to nie dublować kodu i przy edycji i dodawaniu zmian dodać je tylko w jednym miejscu.

Instaluje się poprzez:

* npm install ejs –save

W aplikacji jest podpinany poprzez następujący kod:

* app.set("view engine", "ejs");

Daje to, że serwer podczas renderowania oczekuję, że strony domyślnie mają rozszerzenia .ejs.

**express**

Express[[13]](#footnote-13) jest szybkim, minimalistycznym frameworkiem sieciowym dla Node.js. Express pomaga z napisaniem routingu oraz przy przekierowaniach w HTTP. Jest to jeden z podstawowych pakietów w npm i jest w 10 najpopularniejszych pakietów. Cały REST routing danej aplikacji bazuję się na danym pakiecie.

Instaluje się poprzez:

* npm install express –save

W aplikacji jest podpinany poprzez następujący kod:

* const express = require('express');

const app = express();

app.get('/', function (req, res) {

res.send('Hello World');

})

app.listen(3000);

Daje to, że przykładowo aplikacja oczekuję na lokalnym porcie 3000 i po przejściu do „/” na danym porcie zostanie wyświetlony tekst „Hello World”.

**express-fileupload**

Przy pomocy express-fileupload[[14]](#footnote-14) po przesłaniu pliku do aplikacji plik będzie dostępny z req.files. W danej aplikacji, kiedy użytkownik dodaje plik z obrazkiem do ogłoszenia. Dany plik jest odczytany po stronie serwera.

Instaluje się poprzez:

* npm install express-fileupload –save

W aplikacji jest podpinany poprzez następujący kod:

* const fileUpload = require('express-fileupload');

**express-session**

Express-session[[15]](#footnote-15) pomaga w zidentyfikowaniu użytkownika poprzez tworzenie sesji. Dane sesji są przechowywane po stronie serwera.

Instaluje się poprzez:

* npm install express-session –save

W aplikacji jest podpinany poprzez następujący kod:

* const session = require ( 'express-session' );

app.use(session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true,

}));

Daje to, że użytkownik po zalogowaniu i do wylogowaniu byłby znanym użytkownikiem i miałby za sobą dane danej sesji.

**method-override**

Tak jak HTML formy w danym momencie nie podtrzymują „PUT” i „DELETE” metod[[16]](#footnote-16) temu method-override[[17]](#footnote-17) pomaga w tworzeniu RESTfull routingu z obejściem danego. Jeżeli ustawić method=”POST”, ale action trochę modyfikować to po stronie serwera method-override zmieni typ requesty na PUT albo DELETE odpowiednio.

Instaluje się poprzez:

* npm install method-override –save

W aplikacji jest podpinany poprzez następujący kod:

* const methodOverride = require('method-override');

app.use(methodOverride('\_method'));

Daje to, że każde PUT żądanie z dopiskiem „\_method=PUT” na końcu atrybutu action formy zostanie traktowane jako PUT request, i tak samo z żądaniami typu DELETE.

**moment**

Moment[[18]](#footnote-18) jest lekką biblioteką JavaScript do analizowania, sprawdzania poprawności, manipulowania i formatowania dat. W przypadku danej aplikacji dla wyświetlenia jak dawno temu było stworzone ogłoszenie albo komentarz biblioteka moment pomada w formatowaniu danych pokazywanych użytkownikowi.

Instaluje się poprzez:

* npm install moment –save

W aplikacji jest podpinany poprzez następujący kod:

* const moment = require('moment');

res.locals.moment = moment;

Daje to, że przy renderowaniu strony użytkownikowi po stworzeniu ogłoszenia czy komentarza byłoby widać, że jest stworzony kilka sekund temu, a później ile minut, godzin itd. temu.

**mongoose**

Mongoose[[19]](#footnote-19) to narzędzie do modelowania obiektów MongoDB przeznaczone do pracy w środowisku asynchronicznym. Czyli mongoos posługuję do działania z bazą danych MogoDB. Opisanie bazy danych da jej schematów jest w jednym z następnyh działów.

Instaluje się poprzez:

* npm install mongoose –save

W aplikacji jest podpinany poprzez następujący kod:

* const mongoose = require("mongoose");

mongoose.connect("DB\_path");

Daje to możliwość działania na bazie danych dla odczytywania, zapisywania oraz modyfikowania i usunięcia danych z bazy.

**node-geocoder**

Node-geocoder[[20]](#footnote-20) to biblioteka do geokodowania i geodekodowania. Czyli kiedy użytkownik podaje adres on jest wyszukiwany w danej bibliotece, która zwraca później dane o lokalizacji, co zostaną użyte w systemie. I z powrotem przy odczytywaniu z bazy geocoder przewraca dane o lokalizacji do postaci czytelnej o ile szczegółowo zostały wskazane przy podaniu (kraj, miasto, ulica, numer).

Instaluje się poprzez:

* npm install node-geocoder –save

W aplikacji jest podpinany poprzez następujący kod:

* const options = {

provider: 'google',

httpAdapter: 'https',

apiKey: 'YOUR\_API\_KEY',

formatter: null

};

const geocoder = NodeGeocoder(options);

**nodemon**

Nodemon[[21]](#footnote-21) jest pakietem co pomaga przy tworzeniu oprogramowania. Serwer podczas go napisania potrzebno by było uruchamiać ponownie co raz po wprowadzaniu zmian. Zamiast robić to ręcznie można dodać nodemona, który będzie oglądał pliki w katalogu, w którym nodemon został uruchomiony, a jeśli jakiekolwiek pliki ulegną zmianie, nodemon automatycznie zrestartuje aplikację noda. Dany pakiet nie jest w pliku package.json ponieważ jest instalowany globalnie.

Instaluje się poprzez:

* npm install -g nodemon

Daje to możliwość do automatycznego ponownego uruchamiania serwera po zmianach poprzez polecenie „nodemon nazwa\_serwera”.

**passport**

Passport[[22]](#footnote-22) jest pakietem, który pomaga w autentyfikowaniu użytkowników. Passport wykorzystuje koncepcję strategii do autentyfikowanie żądań. Strategie mogą obejmować weryfikację nazwy użytkownika i poświadczeń hasła. Passport jest dobry tym, że nie narzuca żadnych schematów bazy danych, co pozwala podejmować decyzję programistom systemu. Passport można by było połączyć z logowaniem poprzez Google czy Facebook ta innych, ale dla danego projektu została wybrana lokalna strategia logowania.

Instaluje się poprzez:

* npm install passport –save

W aplikacji jest podpinany poprzez następujący kod:

* const passport = require('passport');

app.use(passport.initialize());

app.use(passport.session());

passport.use(new LocalStrategy(User.authenticate()));

passport.serializeUser(User.serializeUser());

passport.deserializeUser(User.deserializeUser());

Daje to, że passport wykorzystuje lokalną strategię dla autentyfikowaniu użytkowników.

**passport-local**

Passport-local[[23]](#footnote-23) przedstawia lokalną strategię logowania przy użyciu nazwy użytkownika i hasła.

Instaluje się poprzez:

* npm install passport-local –save

W aplikacji jest podpinany poprzez następujący kod:

* const LocalStrategy = require('passport-local');

passport.use(new LocalStrategy(User.authenticate()));

Daje to możliwość logowania się lokalnie, bez wykorzystania innych profilów użytkowników z innych intancij.

**passport-local-mongoose**

Passport-local-mongoose[[24]](#footnote-24) jest plaginem dla konfiguracji mongoose dla używania passportu.

Instaluje się poprzez:

* npm install passport-local-mongoose –save

W aplikacji jest podpinany poprzez następujący kod:

* const passportLocalMongoose = require('passport-local-mongoose');

User.plugin(passportLocalMongoose);

**Routing**

Routing w aplikacji jest bazowym funkcjonałem, bez którego manipulowanie serwisem byłoby dość skomplikowane. Routing był robiony bazując schematu architektury REST, gdzie dla każdej operacji (INDEX, SHOW, CREATE, NEW, EDIT, UPDATE, DESTROY) jest własny routing z właściwym dla niego funkcjonałem. Przykładowy routing wygląda następująco:

* app.get("/", (req, res) => {

res.render("landing", {pagename: "All X Travel"});

});

Gdzie “/” to adres na naszym serwerze (w danym przypadku http://localhost:3005/), req to parametry wejściowe do routingu (parametry z formularza również przychodzą tutaj req.body.parametr) a res parametry wyjściowe. Res.render odpowiada za renderowanie strony i przyjmóje dwa parametry, pierwszy to nazwa pliku co zostanie wyświetlony i jest to landing.ejs, gdzie rozszerzenie .ejs jest domyślnie oczekiwane w aplikacji, a drugi jest to objekt który przyjmuje parametry, co zostaną przekazane do strony, czyli przy renderowaniu strony ejs podstawi zamiast <%= pagename %> text All X Travel.

Routing został podzielony oraz zapisany do różnych plików ze względy na obsługiwaną funkcjonalność, czyli np. w pliku comments ścieżki odpowiadające za tworzenie i edytowanie komentarza co jest logicznym. Dzielenie na osobne pliki pomaga podtrzymywać kod czytelnym i nie przegromadzonym oraz łatwiejszym w edycji, bo za nazwą pliku można zrozumieć jaki routing on podtrzymuje.

**Baza danych - Mongoose NONSQL database**

Dla projektu została wybrana baza danych typu NONSQL, oznacza to, że baza charakteryzuje się dużą skalowalnością, wydajnością oraz brakiem ściśle zdefiniowanej struktury obsługiwanych baz danych (np. dokument ma 100 zadeklarowanych pól, ale każdy wiersz może mieć od 0 do 100 pół, o ile nie są wymagane) na przeciwieństwo baz danych typu SQL, gdzie struktura odrazu jest stałą i każdy wiersz (pojedynczy obiekt) ma te pola zawsze (np. tabela ma 100 zadeklarowanych kolumn to i każdy wiersz ma 100 kolumn odpowiednio). Wykorzystanie bazy typu NONSQL daje możliwość zaoszczędzać przestrzeń, co jest bardzo istotne w przypadku wielkiej ilości danych.

**Relacji pomiędzy tabelami**

Relacje pomiędzy dokumentami mogą być albo 1 – 1 albo 1 - \*. W danej aplikacji są cztery różne dokumenty opisujące połączenia pomiędzy nimi i ich danymi.

* Każdy użytkownik może utworzyć wiele ogłoszeń, ale jedno ogłoszenie może mieć tylko jednego autora (1 - \*);
* Każdy użytkownik może utworzyć wiele komentarzy do ogłoszenia, ale jeden komentarz może mieć tylko jednego autora (1 - \*);
* Każde ogłoszenie ma jeden obrazek do go przedstawienia oraz ten obrazek jest przypisany tylko do jednego zdefiniowanego ogłoszenia (1 - 1);
* Każde ogłoszeni może mieć dużo komentarzy, ale jeden komentarz może być tylko w jednym ogłoszeniu (1 - \*).

Za zdefiniowanie połączeń pomiędzy dokumentami oraz zdefiniowanie pól, które te dokumenty mają, odpowiadają schematy. Ich wygląd można zobaczyć poniżej.

FILL ME WITH IMAGES

**Autoryzacja**

Za autoryzację użytkownika w systemie odpowiadają 3 dołączone pakiety I są to:

* passport;
* passport-local;
* passport-local-mongoose.

Pasport zapełnia aplikacje funkcjonalnością do logowania użytkownika, passport-local mówi passportowi, że dla logowania używamy lokalną strategię, a passport-local-mongoose odpowiada za dodawanie username, pół hash i salt, aby zapisać nazwę użytkownika, hashowane hasło i wartość salt.

Całą autoryzację można wykonać poprzez konfiguracje danych pakietów oraz używanie gotowych funkcji, bez napisania własnego kodu co odpowiadałby za logowanie i rejestrowanie użytkowników w systemie.

**Authentyfikacja oraz Middleware**

Część oprogramowań pośredniczących była wykorzystywana z dołączonych pakietów, ale niestety nie wszystko można się zrobić w taki sposób, ponieważ każda aplikacja na własną funkcjonalność i przewidywane miejsca ich wywołania za kolejnością, temu do danej aplikacji musiałem napisać cztery własne middleware odnoszących się do uprawień użytkownika, ale żeby wiedzieć kim jest dany użytkownik on musi by zaauteryzowany oraz zaautentyfikowany. Poniżej są opisane znaczenia działania czterech middleware:

* Dlatego, żeby dodawać, edytować albo usuwać ogłoszenie użytkownik musi być zalogowany do systemu;
* Dlatego, żeby edytować albo usuwać ogłoszenie użytkownik musi być autorem danego ogłoszenia;
* Dlatego, żeby edytować komentarz użytkownik musi być autorem danego komentarza;
* Dlatego, żeby usuwać komentarz użytkownik musi być autorem danego komentarza albo autorem ogłoszenia, do którego przypisany jest komentarz.

FILL ME WITH IMAGES

Po wykonaniu danych sptardzań program był wywoływany dalej w swoim rządaniu, a jeżeli nie to żądanie zostawało odrzucone. Przystąpienie do kolejnej części było poprzez polecenie next().

**Kroki wykonywane przy tworzeniu projektu**

System był rozwijany przyrostowo, co w wyniku dawało, że w każdym momencie jakaś część już była gotowa i nie że potrzebno było czekać do samego końca, żeby wszystko zadziałało. Dane rozbudowanie systemu było wykonywane zgodnie z zakresem pracy, ponieważ tam zostały opisane i zdefiniowane poszczególne zadania do realizacji oraz ich kolejność.

1. REST Routing

Na samym początku było pobieranie pakietów przez npm i ich podpinanie w aplikacji. Dane działanie występowało na stałe przy rozszerzeniu funkcjonalności. W zależności od tego jaka akcja musiała być dodana zależał jej funkcjonał, ale najpierw dane zdarzenie musiałoby mieć własną ścieżkę marszrutyzacji. Np. po instalowaniu, użyciu pakietów i wybraniu, na jakim porcie musi działać aplikacja (również dobrze mogłaby działać nie lokalnie a na hostowanym serwerze pod wykupioną domeną), pierwszym co było napisane to była ścieżka „/” odpowiadająca za stronę główną aplikacji. Przy testowaniu jakiejkolwiek ścieżki zaczynałem od wysyłania res.send („text”), żeby zobaczyć, że dana ścieżka działa i sprawdzić jej działanie. Czyli kod wyglądał jako:

var express = require('express');

var app = express();

app.get('/', function(req, res){

res.send('hello world');

});

app.listen(3005, function (){

console.log("Server Started");

console.log("Port 3005");

});

Przy otwarciu w przeglądarce danego portu i przejściu na jego ścieżke „/” wiedziałem tekst 'hello world' co oznaczało, że ścieżka działa i można iść dalej.

1. Tworzęnie stron

Kolejnym krokiem było tworzenie strony co była wyświetłona użytkownikowi przy przejściu do danej strony. Najpierw to była strona HTML w tekstowej postaci z tekstem co mówił nie wiele więcej, że jest to strona główna i tego było wystarczająco, bez żadnych stylowań, później w kolejnych krokach była ona stylowana i wyglądała coraz lepiej. Dla wyświetlenia strony linia z res.send była podmieniona na res.render ze ścieżką do pliku, ale tak jak plik był w tym samym miejscu to było to proste.

1. Tworzenie ogłoszenia

Dalej rozwijając aplikacje dodana była ścieżka i strona dla ogłoszenia. Zaczynać pasowało od ścieżek „NEW” i „CREATE”, ponieważ odpowiadają oni za pokazanie strony dla stworzenia ogłoszenia oraz dodawania danego ogłoszenia. Na początku wystarczyło formularza, co wysyłał dane, a po stronie serwera ich wyświetlenia w konsoli. Przy tworzeniu stron było zauważono, że części początku z dodaną nawigacją i końca pliku z podpinanym bootstrapem również takie samo, jak i w przypadku strony głównej, temu dodany i używany został pakiet ejs, co pomógł wynieść części wspólne do jednego wspólnego pliku.

1. Konfigurowanie bazy danych i podpinanie jej do aplikacji, żeby dać możliwość zapisywania danych do niej

Po odebraniu danych przesłanych ze stromy i wyświetleniu ich wystąpiła konieczność gdzieś ich przechowywać. Mongoose – pakiet npm odpowiadający za połączenie z bazą mongoDB, którą również pobrałem i dopasowałem dla zapisywania danych z serwera. Po wykonaniu danych działań został stwożony schemat dla ogłoszenia, co miał w sobie pola dla podania autora, nazwy oraz tekstu ogłoszenia. Po stronie serwera dodane zostało zapisywanie do bazy oraz nowe ścieżki dla odczytywania z bazy. „INDEX” odpowiadał za wyświetlenie wszystkich ogłoszeń a „SHOW” za wyświetlenie ściególnego wybranego ogłoszenia. Tak jak w danym kroku było najwięcej kodu do napisania, od początku projektu do danego momentu, to zauważono było, że przy każdej zmianie nawet drobnej, czy literówce potrzebno było odnowa odpalać serwer, żeby zmiany zostały dodane do aplikacji. Wyjściem było dodanie pakietu nodemon, co pomagał w tym i automatycznie restartował serwer odnowa przy zmianach w plikach .js. Ułatwiało to pisanie kodu bez ciągłego odpalania serwera odnowa. Zmieniło się polecenie dla odpalania z „node app.js” na „nodemon app.js”.

1. Dodawanie komentarzy

Dalej miało sens dodawanie komentarzy do ogłoszeń, temu zdefiniowany został schemat dla bazy danych oraz napisane zostały ścieżki, oraz strona dla obsługi dodawania komentarzy. Tak jak komentarzy nie mają własnej strony i nie ma strony do pojedynczego wyświetlenie komentarza to i ścieżki „INDEX” oraz „SHOW” nie są potrzebne. Natomiast komentarzy są pokazywane na stronie ogłoszenia, ponieważ do niej i mają bezpośrednie odnoszenie. Wygląda to w następujący sposób. Tak jak w ogłoszeniu oprócz parametrów zwykłych, pole z komentarzami jest powiązane z komentarzami w połączeniu 1 - \* to wyszukując za parametrem id zapisanym w ogłoszeniu, można znaleźć komentarzy. To wszystko jest robiono przy wyszukiwaniu ogłoszenia w bazie danych. Jako dodatkowy parametr przy odczytywaniu obiektu ogłoszenia zadany jest .populate("comments") co wyszukuję za parametrem id komentarza. Na samej stronie był napisany JavaScript, co przechodził po każdym objekcie i wyświetlał każdy osobny komentarz. Tak jak ilość stron się gromadziła, to potrzebno było podzielić ich na logiczne części i tak został stworzony folder view oraz pod foldery za ich znaczeniem.

1. Logowanie

Przed dodawaniem możliwości usunięcia oraz edycji ogłoszenia czy komentarza potrzebno by było wiedzieć kto dane ogłoszenie, czy komentarz napisał. Czyli potrzebno było zaimplementować schemat bazy dla użytkownika oraz pobrać pakiety z npm ta użyć ich w aplikacji. Przy pomocy pakietów passport oraz dodatków do niego to było bardzo prosto, ponieważ opisywanie tego, w jaki sposób połączyć to wszystko razem było na stronie pakietów. Po zaimplementowaniu ścieżek logowania oraz rejestracji użytkownika ilość kodu była podzielona, jak i w przypadku stron i folderu view, był stworzony folder routes i jego podkatalogi.

1. Autentyfikacja

Dany problem też był rozwiązany przy pomocy pakietów passport ta innych. Żeby wiedzieć informacje o użytkowniku, co stworzył ogłoszenie czy komentarz, ich schematy oraz ścieżki ulegli zmienię dla przechowywania danej informacji.

1. Edytowanie lub usunięcie ogłoszenia przez go wsaściciela

Po dodawaniu ogłoszenia pasowałoby dodać funkcjonalność dla obsługi go edytowania. Po dodawaniu odpowiednich ścieżek dla edycji obiektu w bazie, na podstawie id ogłoszenia, dodane zostały middleware, co sprawdzały, czy użytkownik co teraz jest zalogowany, ma lub nie ma uprawnienia dla edycji, lub usunięcia. Najpierw był napisany middleware, co sprawdza, czy użytkownik jest zalogowany a później i funkcji sprawdzające za parametrem id autora ogłoszenia. Po przejściu middleware użytkownik miał dostęp do ścieżek „EDIT”, „UPDATE” ta „DELETE”.

1. Edytowanie lub usunięcie komentarzy przez ich wsaścicieli

Edytowanie ta usunięcia komentarza różnią się uprawnieniami. Edytować ogłoszenie może tylko zalogowany użytkownik co był już zalogowany przy tworzeniu ogłoszenia i jest go autorem. Również dana osoba może usuwać ogłoszenie. Jeszcze uprawnienia dla usunięcia komentarzy ma autor ogłoszeń.

1. Wyświetlenie komunikatów dla użytkownika oraz rozszerzenie istniejącego funkcjonału

Po tworzeniu funkcjonalnej części pasowałoby ją rozszerzyć, temu dodawane zostały nowe pola do ogłoszeń, formularzy z nimi związane oraz schematy w bazie danych. Teraz do ogłoszenia dodawany jest obrazek, co jest zapisywany po stronie serwera, mapa Google z jej API, kalendarz dla wyboru dat wyjazdu ta przyjazdu oraz jak i z komentarz ma teraz datę tworzenia. Obrazek został zapisany na dysk twardy, a nie do bazy danych z tego powodu, że dla zapisu obrazku do bazy pasowałoby go przekonwertować do postaci base64, ale wysoka ilość zapytań do serwera powodowałaby to, że każdy raz przy reagowaniu na żądanie obrazek musiałby być konwertowany do base64 albo z powrotem i obciążenie po stronie serwera byłoby większe niż w przypadku zapisywania na dysk. Wszystkie te dane pomogą w zdefiniowaniu punktu oraz daty wyjazdu. Obrazek byłby pokazany na stronie „INDEX” dla wyszczególnienia ogłoszenia nie tylko tekstem, a i grafiką również.

Dlatego, żeby system w jakiś sposób kowanikował się z użytkownikiem i użytkownik się nie gubił w nim, nie wiedząc co się dzieje, do użytkownika muszą być wysyłane komunikaty z serwera z informacją informującą o toku reakcji na zdarzenie po stronie użytkownika końcowego. W celu tego podpinany został pakiet connect-flash co i odpowiada za dane komunikaty. Komunikaty zostałyby wyświetlone użytkownikowi w razie przewidywanych błędów albo w odpowiedź na zakończone działania, ale już po przeładowaniu strony nie byliby widoczne. Odbyła się w następujący sposób, że przed przekierowaniem do innej strony definiuję się komunikat, co zostaję wyświetlony w konkretny, miejscu na stronie.

1. Responsywność stron do różnych rozszeżeń ekranu.

Po zakończeniu dodawania funkcjonału stronę go reprezentujące zostały ostyłowane. Styłowanie musiało być odpowiednie do różnych rozmiarów ekranów, ponieważ im mniejszy jest ekran, tym mniej elementów mogłoby być pokazywane na jednym ekranie użytkownika.

Dla styłowania był używany bootstrap, ponieważ ma on już gotowe klasy ze zdefiniowanym stylowaniem oraz system grid co pomógł przy rozmieszczeniu elementów przy różnej rozdzielczości. Np. Google map, co została dodana, znajduję się po prawej stronie, kiedy rozdzielciość ekranu na to pozwala, a jak nie — przechodzi wdół pod główna część ogłoszenia, ale nad komentarze i zostaje ukryta. Po kliknięciu na przycisk zostaję ona pokazywana albo ukrywana z powrotem. To, co nie było w bootstrapie albo specyficzne stylowanie do stron było napisano ręcznie w pliku ze stylowaniem. @media odpowiada za stylowanie przy zdefiniowanym rozszerzeniu.

1. Podtzymywanie różnych wersii przeglądarek internetowych.

Żeby podtrzymywać różne przeglądarki (również IE) została wybrana 3 wersja bootstrapu, chociaż na moment napisania pracy wersja 4 była dostępna do użycia. Było to wybrano z tego powodu, że w wersji 4 bootstrap był robiony na display flex, którego IE nie podtrzymuje do wersji 11. Wersja 3 jest optymalną w danym przypadku, żeby podtrzymywać kompatybilność z większą ilością przeglądarek (nawet do IE 9).

Po wykonaniu danych kroków projekt był gotowy i spełniający wymagania.

**Sprawdzenie działania aplikacji**

FILL ME

**Demonstrowanie alikacji i jej działania**

FILL ME WITH FINAL IMAGES

**Napotykane problemy**

Kiedy człowiek się uczy czegoś nowego, to ciągle ma jakieś problemy przy programowaniu czegoś, bo nie wie, w jaki właściwie sposób byłoby najlepiej dany problem rozwiązać. Np. przy napisaniu programu miałem problemy z następnymi rzeczami:

Po instalacji, mając probmemy z uruchamieniam serwera przy użyczi nodemona, skożystałem się z rozwiązania znalezianego na Stack Overflow[[25]](#footnote-25), które pomogło.

Mając problem z podpinaniem Google API do własnej aplikacji, przy ograniczeniach klucza do localhostu temu zostało utworzono 2 kluczę, gdzie jeden był do przesyłania geodanych i ich opracowania a drugi przy wyświetleniu przy pomocy JavaScriptu wyników na mapie.

Inne drobniejsze problemy konfiguracyjne najczęściej zostały rozwiązane po przeczytaniu dokumentacji do pakietu.

**Możliwości do przyszłego rozwinięcia aplikacji**

Tak jak funkcjonał danej pracy opierał się głównie na założeniach wstępnych, ale również od części rozwijał je, mógłby on również mieć następujące czynności, ale na danym etapie nie było potrzeby w zaimplementowaniu danych rzeczy:

* JQuery Validate

Validowanie formularza przed wysyłką mogłoby być zrealizowane przez JQuery Validate. Przy pomocy JQery Validate można by było zdefiniować wymagane reguły oraz tekst walidacji do wybranych pól.

* User profile

Również, jak i ogłoszenia użytkownik mógłby mieć własny kabinet z ustawieniami, możliwością zmiany hasła i temu podobnych rzeczy, ale zwiększyłoby to kod minimum o jedną trzecią. Dany funkcjonał byłby preferowany w dodawaniu kolejnym przy rozwinięciu aplikacji.

* Admin profile

Dobrzą praktyką by było mieć użytkownika z prawami administratora dla zarządzania systemem oraz ogłoszeniami i kontrolowaniem innych użytkowników.

* Captcha

Działałoby to tak jak i weryfikacja dla niwelacji spamu w systemie. Wprowadzenie captch jest dość typowym metodem dla tego.

* Double opt-in

Dla potwierdzenia rejestracji można by było używać weryfikowania przez wysyłanie potwierdzenia rejestracji na konto e-mailowe. Zmniejszyłoby to ilość niepoprawnych danych przy podaniu e-maili oraz ilość danych nieprawdziwych czy też ze spam systemów. Również zamiast e-maili albo w dołączeniu do tego można by było wysyłać SMS-y do użytkowników, systemu z proszeniem o podanie kodu wysłanego do nich jako potwierdzenia rejestracji.

* Podział na strony

Podział na strony kontentu ogłoszeń oraz komentarzów zmieniłoby wygląd do mniejszej postaci, ponieważ jeżeli każde ogłoszenie / komentarz wyświetlać po kolei jedno za drugim ich ilość zrobiłaby strony bardzo długimi.

* Podział na kategorie

Kategorie pomogliby w sortowaniu ogłoszeń za działami, do których są przywiązane beż niezbędności przeglądania wszystkich.

* Wyszukiwanie

Wyszukiwanie pomogłoby znałeś ogłoszenia z konkretną treścią albo w przedzielę cenowym czy w odcinku dat wybranych. Podanie wielkiej ilości parametrów wyszczególniłoby najbardziej pasujące ogłoszenia.

* Powiadomienia

Użytkownik mógłby dostawać komunikat, że na jego ogłoszenie albo komentarz ktoś odpowiedział poprzez notyfikacje w systemie oraz wysyłanie e-maili to danego użytkownika.

**Zakończenie**

Po zakończeniu tworzenia projektu w wyniku był stworzony serwis, który pomaga użytkownikom w znalezieniu odpowiedniej grupy dla wspólnego wyjazdu, znalezieniu przewodnika w nowym mieście albo tworzeniu własnych ogłoszeń, żeby znałeś grupę, albo zaproponować pomoc w pokazywaniu miasta. Poprzez tworzenie komentarzy do ogłoszenia można zawiadomić o swojej chęci brania udziału w danej podróże.

Każde ogłoszenie zawiera informacje o datach wyjazdu i powrotu oraz Google map z geodanymy dla wskazania miejsca podróży. Żeby ogłoszenie było wyszczególnione, do niego dołączony jest obrazek, co jest zapisywany na serwerze.

Za działanie serwisu odpowiadają części front-endowa oraz back-endowa. Po zakończeniu danej pracy pogłębiłem poznania w części front-endowej, czyli z HTML5, CSS w tym Bootstrap, JavaScript w tym jQuery. Odnośnie do części serwerowej zapoznałem się z działaniem i nauczyłem się napisania JavaScriptów w środowisku Node.js oraz działania z npm. Również opanowałem podłączenie się do bazy danych typu NONSQL (była to baza MongoBD) i działanie z nią.

Założenia opisane w niniejszej pracy zostały spełnione, rozszerzone oraz przetestowane. Możliwe przyszłe rozwinięcia projektu zostały wyszczególnione oraz opisane w działę powyżej.

Podsumowując, można powiedzieć, że praca powiodła się skutkiem i była skończona tak, jak i było zaplanowano.

Literatura

1. <https://getbootstrap.com/>
2. <https://jquery.com/>
3. <https://nodejs.org/en/>
4. <https://insights.stackoverflow.com/survey/2017#technology>
5. <https://insights.stackoverflow.com/survey/2017#technology-frameworks-libraries-and-other-technologies>
6. <https://www.npmjs.com/>
7. <http://sd.blackball.lv/library/RESTful_Web_APIs_(2013).pdf>
8. <https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>
9. <https://jqueryvalidation.org/>
10. <https://www.npmjs.com/package/body-parser>
11. <https://www.npmjs.com/package/connect-flash>
12. <https://www.npmjs.com/package/ejs>
13. <https://www.npmjs.com/package/express>
14. <https://www.npmjs.com/package/express-fileupload>
15. <https://www.npmjs.com/package/express-session>
16. <https://softwareengineering.stackexchange.com/questions/114156/>
17. <https://www.npmjs.com/package/method-override>
18. <https://www.npmjs.com/package/moment>
19. <https://www.npmjs.com/package/mongoose>
20. <https://www.npmjs.com/package/node-geocoder>
21. <https://www.npmjs.com/package/nodemon>
22. <https://www.npmjs.com/package/passport>
23. <https://www.npmjs.com/package/passport-local>
24. <https://www.npmjs.com/package/passport-local-mongoose>
25. <https://stackoverflow.com/a/17976504>

**Zalecana zawartość treści pracy**

Wstęp

Kilkustronicowa analiza ogólna wprowadzająca w realizowane zagadnienie, zarysowująca problematykę pracy i użyteczność wynikającą z jej realizacji (uzasadnienie realizowanej problematyki). W ramach wstępu należy wyszczególnić następujące elementy:

* uzasadnienie wyboru problematyki/tematu pracy,
* cel pracy
* aktualny stan wiedzy dotyczący problemu (zagadnienia) będącego przedmiotem pracy,
* zakres pracy,
* omówienie układu pracy, to jest krótka charakterystyka poszczególnych rozdziałów.

Pod pojęciem celu pracy dyplomowej na studiach pierwszego stopnia należy rozumieć nie tyle samo rozwiązanie problemu, ile korzyści jakie ono przyniesie (cel to przyszły efekt podjętego działania - rozwiązywania problemu - który działanie to uzasadnia). Cel pracy powinien być tak sformułowany, aby możliwa była ocena efektywności działań podjętych w ramach jego rozwiązywania oraz stopnia osiągnięcia założonego celu.

Część główna pracy (podzielona na numerowane rozdziały, ewentualnie podrozdziały)[[26]](#footnote-26)

Część główna pracy powinna być dostosowana do specyfiki i przedmiotu rozwiązywanego problemu (zagadnienia) i składać się z części teoretycznej i praktycznej.

W części teoretycznej autor powinien dokonać charakterystyki problematyki pracy, analizy aktualnego stanu rzeczy z określeniem pojawiającego się problemu oraz wskazać potencjalne kierunki jego rozwiązania z uzasadnieniem wyboru jednego z nich.

W części praktycznej autor powinien wykazać się znajomością adekwatnych do kierunku studiów procedur (metodologii) rozwiązywania problemów, umiejętnością określania stopnia osiągniecia założonego celu pracy oraz wyciągania na tej podstawie wniosków na temat uzyskanych wyników i efektywności podjętych działań.

Zakończenie

Zakończenie powinno zawierać ocenę skuteczności podjętych działań i uzyskanych efektów pod kątem stopnia osiągniecia założonego celu pracy.

Literatura

Należy podać wszystkie źródła, które wykorzystano w pracy, w tym również - oprócz aktów prawnych, wydawnictw zwartych i ciągłych - adresy stron www.

**I. ZASADY FORMATOWANIA TEKSTU PRACY**

* Plik z pracą dyplomową należy zapisać jako dokument programu Word 97-2003 (\*.doc).
* Nie należy wstawiać nagłówków i stopek z oddzielną treścią, np. tytułem rozdziału.
* Każdą część pracy należy rozpoczynać od nowej strony.
* Nie należy stawiać kropki po tytule danej części pracy (w tym rozdziału/podrozdziału) oraz po tytule pracy.
* Numeracja stron - obowiązkowo automatyczna (w stopce, wyrównana do środka strony, Czcionka Times New Roman 12 pkt.). Pierwszą stroną pracy jest strona tytułowa i od niej należy zacząć numerowanie, ale nie umieszczać na niej numeru strony.
* Automatyczny spis treści – na trzeciej stronie pracy dyplomowej.
* Czcionka - Times New Roman, rozmiar 12 pkt., grubość normalna.
* Odstępy między znakami – standardowe (0 pkt.).
* Pomiędzy wyrazami nie należy wstawiać więcej niż jednego znaku spacji.
* Odstępy między wierszami - pojedyncze.
* Tekst justowany obustronnie.
* Włączona opcja automatycznego dzielenia wyrazów.
* Jeżeli dany akapit ma znaleźć się na nowej stronie należy zastosować funkcję podziału strony (kombinacja klawiszy: *Ctrl + Enter*).
* Na końcu wiersza nie mogą występować pojedyncze litery lub spójniki. W celu przeniesienia spójnika do kolejnego wiersza należy użyć tzw. twardej spacji (kombinacja klawiszy: *Ctrl* + *Shift* + *spacja*). „Twardą spację” należy wstawić bezpośrednio po spójniku, a przed następującym po nim wyrazem (powoduje to „przyklejenie” spójnika do wyrazu).
* Zabrania się wstawiania na końcu wiersza (w środku zdania) zwykłych oraz „miękkich” enterów.
* Pomiędzy wyrazem a następującym po nim znaku przestankowym (przecinek, kropka, średnik, dwukropek) nie należy wstawiać znaku spacji.
* Wyrazy (zdania) pisane w nawiasie nie mogą być oddzielone od tych nawiasów spacjami.
* Wyliczenia w tekście - za pomocą automatycznego wypunktowania lub numerowania.
* Praca musi być wydrukowana dwustronnie - w zakładce *Plik/Ustawienia strony/Marginesy* należy zaznaczyć „marginesy lustrzane” (margines wewnętrzny - 2 cm, margines zewnętrzny, górny i dolny - 1,5 cm).

**II. STRONA TYTUŁOWA PRACY DYPLOMOWEJ**

Strona tytułowa pracy dyplomowej powinna zawierać:

1. logo w kolorze i nazwę szkoły wg podanego wzoru (czcionka Times New Roman, bold, 16 pkt.),
2. nazwę wydziału, kierunek oraz specjalność (czcionka Times New Roman, bold, 16 pkt.),
3. imię i nazwisko autora (autorów) pracy, nr albumu (czcionka: Times New Roman, 15 pkt.) oraz tytuł pracy dyplomowej (czcionka: Times New Roman, *kursywa*, bold, 20 pkt.),
4. tytuł/stopień naukowy, imię i nazwisko promotora (czcionka: Times New Roman, 12 pkt.),

*UWAGA! Oprawa pracy dyplomowej zawiera okienko o wymiarach 15,5 x 6,5 cm, (umieszczone centralnie w stosunku do marginesów bocznych oraz w odległości 12,5 cm od dolnej krawędzi strony), w którym powinny się zmieścić informacje zawarte w pkt. c) i d).*

1. napis PRACA DYPLOMOWA MAGISTERSKA (lub LICENCJACKA/INŻYNIERSKA), (czcionka: Times New Roman, bold, 19 pkt),
2. napis Rzeszów oraz rok złożenia pracy dyplomowej, np. Rzeszów 2018 (czcionka: Times New Roman, bold, 14 pkt.).

Na kolejnych stronach znajduje się wzór strony tytułowej pracy dyplomowej - wersja dla prac przygotowanych w języku polskim (na przykładzie kierunku Informatyka) oraz wersja dla prac przygotowanych w języku angielskim (na przykładzie kierunku Filologia).

1. <https://getbootstrap.com/> [↑](#footnote-ref-1)
2. <https://jquery.com/> [↑](#footnote-ref-2)
3. <https://nodejs.org/en/> [↑](#footnote-ref-3)
4. <https://insights.stackoverflow.com/survey/2017#technology> [↑](#footnote-ref-4)
5. <https://insights.stackoverflow.com/survey/2017#technology-frameworks-libraries-and-other-technologies> [↑](#footnote-ref-5)
6. <https://www.npmjs.com/> [↑](#footnote-ref-6)
7. <http://sd.blackball.lv/library/RESTful_Web_APIs_(2013).pdf> [↑](#footnote-ref-7)
8. <https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf> [↑](#footnote-ref-8)
9. <https://jqueryvalidation.org/> [↑](#footnote-ref-9)
10. <https://www.npmjs.com/package/body-parser> [↑](#footnote-ref-10)
11. <https://www.npmjs.com/package/connect-flash> [↑](#footnote-ref-11)
12. <https://www.npmjs.com/package/ejs> [↑](#footnote-ref-12)
13. <https://www.npmjs.com/package/express> [↑](#footnote-ref-13)
14. <https://www.npmjs.com/package/express-fileupload> [↑](#footnote-ref-14)
15. <https://www.npmjs.com/package/express-session> [↑](#footnote-ref-15)
16. <https://softwareengineering.stackexchange.com/questions/114156/> [↑](#footnote-ref-16)
17. <https://www.npmjs.com/package/method-override> [↑](#footnote-ref-17)
18. <https://www.npmjs.com/package/moment> [↑](#footnote-ref-18)
19. <https://www.npmjs.com/package/mongoose> [↑](#footnote-ref-19)
20. <https://www.npmjs.com/package/node-geocoder> [↑](#footnote-ref-20)
21. <https://www.npmjs.com/package/nodemon> [↑](#footnote-ref-21)
22. <https://www.npmjs.com/package/passport> [↑](#footnote-ref-22)
23. <https://www.npmjs.com/package/passport-local> [↑](#footnote-ref-23)
24. <https://www.npmjs.com/package/passport-local-mongoose> [↑](#footnote-ref-24)
25. <https://stackoverflow.com/a/17976504> [↑](#footnote-ref-25)
26. dziekan wydziału może w drodze decyzji określić bardziej szczegółowe wytyczne dotyczące zawartości pracy obowiązujące dla danego kierunku studiów [↑](#footnote-ref-26)