Wyższa Szkoła Ekonomi i Innowacji
w Lublinie

Wydział Transportu i Informatyki

Kierunek: Informatyka

Specjalność: Inżynieria oprogramowania

Daniel Mariusz Dereziński

22443

Projekt oraz wdrożenie reaktywnego intranetu dla firmy – wykorzystanie technologi Meteor.js, Bootstrap i MongoDB

Praca inżynierska napisana na Wydziale Transportu i Informatyki pod kierunkiem Prof. Grzegorza Marcina Wójcika

Lublin 2015

Spis treści

Listings

Rozdział 1

Wstęp

1.1 O aplikacjach sieciowych

opisać z [4]

1.2 Intranet i jego historia

Intranet jest to sieć komputerowa ograniczająca się do komputerów np. w danym przedsiębiorstwie lub innej organizacji, dostępna wyłącznie dla pracowników danej organizacji. Intranet dostarcza szeroki zakres informacji oraz usług z wewnętrznych systemów IT organizacji, które nie są dostępne z publicznego — zewnętrznego — Internetu. Firmowy Intranet dostarcza między innymi centralny punkt wewnętrznej komunikacji, współpracy. Intranet stanowi także pojedynczy punkt dostępu do wewnętrznych jakich zewnętrznych zasobów organizacji. W najprostszej formie intranet budowany jest z wykorzystaniem sieci typu *LAN* (sieć lokalna) oraz *WAN* (rozległa sieć komputerowa) [7].

Coś o historii intranetu....

1.3 Intranet obecnie oraz jego zastosowania

1.4 Cel pracy

W dzisiejszych czasach wiele organizacji / firm wykorzystuje w swojej działalności z jakieś formy intranetu — komunikacja, praca zespołowa. Są to rozwiązania oparte o darmowe systemy CMS, komunikatory, kalendarze, systemy do zarządzania zadaniami. Celem niniejszej pracy było opracowanie oraz wdrożenie systemu intranetowego dla firmy zajmującej się produkcją oprogramowania. Firmy z tej branży pracują w oparciu o

projekty. Jednym z podstawowych celów intranetu jest wymiana wiedzy oraz komunikacja. Zaprojektowana oraz zaprogramowana aplikacja umożliwia dodawanie artykułów, dodawania kategorii, dodawanie projektów, komunikację w obrębie projektów wraz z możliwością dodawania artykułów. Aplikacja pozwala utworzyć profil dla organizacji, zaprosić użytkowników do tak utworzonego profilu w celu podjęcia wspólnej pracy. Możliwe jest także tworzenie kont dla użytkowników nie powiązanych z żadną organizacją.

W rodziale 2 zostanie przedstawione ....

Rozdział 2

Technologia

Rozdział ten przedstawia wykorzystane technologie oraz języki programowania użyte podczas projektowania oraz programowania aplikacji Intranet. Aplikacja powstała z wykorzystaniem *JavaScript*, frameworka aplikacji sieciowych *Meteor.js*, nierelacyjnej bazy danych *MongoDB*, *HTML5*, *CSS3*, *less*[1], frameworka CSS *Bootstrap* oraz gotowego szablon dla panelu administracyjnego *AdminLTE* wykorzystujący Bootstrap.

2.1 JavaScript

Język programowania JavaScript został użyty do zaprogramowania zarówno części serwerowej (*back-end*) jaki i części odpowiedzialnej za interakcje z użytkownikiem (*front-end*) — interfejs użytkownika. Obecne strony WWW a w szczególności aplikacje dostępne przez przeglądarkę (Gmail, Google Docs, Google Maps, Facebook) szeroko korzystają z JavaScript w celu dostarczenie wielofunkcyjnego oraz interaktywnego interfejsu użytkownika. Jednym z powodów wykorzystanie JavaScript była możliwości wykorzystania go po stornie serwera oraz klienta. Najpopularniejszy obecnie sposób tworzenia stron/aplikacji internetowych wyróżnia trzy warstwy — warstwę struktury (HTML), warstwę prezentacji (CSS) oraz warstwę zachowania (JavaScript) [3].

Internet powstał jako zbiór statycznych dokumentów HTML, które były powiązane hiperłączami. Po wzroście popularności oraz rozmiaru sieci, autorom stron przestały wystarczać dostępne narzędzia. Widoczna stała się potrzeba poprawienia interakcji z użytkownikiem. U jej podstaw leżała chęć zmniejszenia ilości połączeń z serwerem w celu realizowania prostych zadań takich jak np. walidacja formularzy. W tym czasie pojawiły się dwie możliwości rozwiązania tego problemu — aplety Javy oraz język *LiveScript*, który został zaproponowany przez firmę Netscape w roku 1995. Został on dołączony do przeglądarki Netscape 2.0 pod nazwą JavaScript [3].

[1] Dynamiczny język arkuszy stylów

Możliwość modyfikacji statycznych elementów stron internetowych bardzo szybko została przyjęta przez rynek. Producenci przeglądarek internetowych szybko dostosowali swoje produktu do obsługi JavaScript'u. Microsoft wyposażył w taką obsługę swoją przeglądarkę Internet Explorer (IE) od wersji 3.0. Była to jednak kopia języka JavaScript — *JScript*, wzbogacona o kilka funkcjonalności przeznaczonych tylko dla IE. W wyniku coraz większych różnić pomiędzy przeglądarkami podjęto próbę standaryzacji różnych implementacji języka. Zadanie to przypadło Europejskiemu Stowarzyszeniu na rzecz Systemów Informatycznych i Komunikacyjnych (ECMA). Tak powstała specyfikacja ECMAScript. Obecnie obowiązuje standard ECMA-262 [5] — jego najpopularniejszą implementacją jest JavaScript.

Wzrost popularności JavaScriptu nastąpił w czasie Pierwszej Wojny Przeglądarek (1996-2001) [3]. Okres ten nazywany jest także okresem bańki internetowej. W tym czasie o udział w rynku walczyli dwaj główni producenci przeglądarek Netscape oraz Microsoft. Firmy kusiły klientów za pomocą coraz to nowych dodatków i ozdóbek wprowadzanych do przeglądarek oraz do stosowanych w nich wersji JavaScriptu. W tym czasie wiele osób wyrobiło sobie negatywną opinie na temat tego języka, który w wyniku wspomnianych działań oraz braku standaryzacji bez przerwy ulegał zmianie. Pisanie programów było koszmarem. Skrypty napisane w oparciu o jedną przeglądarkę nie chciały działać w drugiej. Producenci przeglądarek, skupieni na rozszerzaniu o nowe funkcjonalności, nie dostarczali odpowiednich narzędzi do rozwijania aplikacji [3].

Niespójności pomiędzy przeglądarkami była tylko częścią problemu. Drugą częścią byli sami autorzy stron, którzy upychali w witrynach zbyt wiele zbędnych funkcjonalności. Bardzo często korzystali z wszystkich nowych możliwości dostarczanych przez przeglądarkę, przez co strony były „upiększane" o kwiatki takie jak animacje na pasku stanu, jaskrawe kolory, migające napisy, trzęsące się okna przeglądarek, płatki śniegu, obiekty podążające za kursorem itp., co bardzo często utrudniało korzystanie ze stron. Tego typu nadużycia są także powodem złej reputacji JavaScriptu. Problemy te doprowadziły do traktowania języka JavaScript za niewiele więcej niż zabawkę przeznaczoną dla projektantów interfejsów.

Po zakończeniu Pierwszej Wojny Przeglądarek sposób wytwarzania aplikacji sieciowych uległ zmianie. Zmiany — na lepsze — zostały zapoczątkowane przez kilka procesów [3]:

- Microsoft wygrał wojnę i na okres około pięciu lata wstrzymał się od dodawania nowych funkcjonalności do przeglądarki Internet Explorer oraz do samego JavaScriptu. Dzięki temu producenci innych przeglądarek zyskali czas na dogonienie a czasem nawet przewyższenie możliwości IE.

- Ruch na rzecz standardów sieciowych zyskał przychylność programistów jaki i pro-

ducentów przeglądarek. Standardy chronią programistów od konieczności programowania funkcjonalności dwa (lub więcej) razy na wypadek, gdyby coś nie działało, w którejś z przeglądarek. Co prawda nadal nie istnieje środowisko, które spełniało by wszystkie możliwe standardy.

- technologie i sposoby programowania osiągnęły bardzo dojrzały poziom, na którym można już zajmować się zagadnieniami takim jak użyteczność, dostępność czy też progresywne ulepszanie.

Dzięki nowym, zdrowszym metodologiom programiści zaczęli uczyć się lepszych sposób korzystania z już dostępnych narzędzi. Po wydaniu aplikacji takich jak *Gmail* czy *Google Maps*, które w bardzo szerokim stopniu korzystają z programowania po stronie klienta, stało się oczywiste, że JavaScript to dojrzały, jedyny w swoim rodzaju oraz potężny prototypowy język obiektowy [3]. Dobrym przykładem jego ponownego odkrycia jest szeroka akceptacja funkcjonalności dostarczanej przez obiekt `XMLHttpRequest`, który dawniej był obsługiwany tylko przez przeglądarkę Internet Explorer. Obiekt ten jednak został zaimplementowany przez wiele przeglądarek. `XMLHttpRequest` umożliwia wykonywanie żądań HTTP i pobieranie zawartości serwera w celu aktualizacji pewnych części strony bez konieczności przeładowanie jej całej. Dzięki temu narodził się nowy gatunek aplikacji sieciowych, które przypominają samodzielne aplikacje desktopowe. Takie aplikacje określamy mianem aplikacji *AJAX*.

JavaScript po rewolucji spowodowanej rozwojem technologii AJAX zaczął być używany przez programistów do tworzenia rzeczywistych i ważnych aplikacji. Obecnie mamy wiele aplikacji sieciowych JavaScript począwszy od Twittera, przez Facebook, aż po GitHub [1]. Jedną z ciekawszych cech JavaScriptu jest to, że musi on działać wewnątrz *środowiska*. Najpopularniejszym środowiskiem jest przeglądarka internetowa. Istnieją także inne możliwości — JavaScript może działać po stornie serwera, na pulpicie lub wewnątrz tzw. *rich media*[2].

Od chwili wydania przeglądarki Google Chrome w roku 2008 ciągle i w bardzo szybkim tempie poprawia się wydajność działania JavaScript, co jest wynikiem silnej konkurencji pomiędzy producentami poszczególnych przeglądarek internetowych. Wydajność nowoczesnych maszyn wirtualnych JavaScript zmienia rodzaje aplikacji tworzonych dla sieci. Fantastycznym przykładem jest jslinux[3] — utworzony w JavaScript emulator pozwalający na wczytanie jądra systemu Linux, pracę w powłoce oraz kompilację programów w C.

[2] Aplikacje *rich media* — Flash, Flex — tworzy się przy użyciu ActionScriptu, który jest oparty o ECMAScript

[3] `http://bellard.org/jslinux/`

2.1.1 Przegląd cech JavaScript

JavaScript jest imperatywnym oraz strukturalnym językiem programowania. Wspiera większość składni programowania strukturalnego z języka C, np. pętle `while`, instrukcję wyboru `switch`, pętle `do while` oraz wiele innych. Wyjątkiem jest zasięg zmiennych. W JavaScript zasięg zmiennych z wykorzystaniem słowa kluczowego `var` to zasięg do całego ciała funkcji. Nowy standard ECMAScript 2015 (ES6) wprowadza zakres zmiennych co do bloku instrukcji z wykorzystaniem słowa kluczowego `let` — co oznacza że język ten ma obecnie zakres zmiennych co do ciała funkcji jak i do bloku instrukcji. Podobnie jak C JavaScript rozróżnia wyrażenia oraz instrukcji [8].

Jak większość języków skryptowych także JavaScript jest językiem dynamicznie typowanym. Typy powiązane są z wartościami a nie ze zmiennymi. Na przykład do zmiennej x może być przypisana wartość typu liczbowego a następnie możemy do takiej zmiennej przypisać na przykład łańcuch znaków [8].

JavaScript jest prawie w całości obiektowy. Obiekty w JavaScript są to tablice asocjacyjne rozszerzone o prototypy. Nazwy właściwości obiektu to ciągi znaków. Obiekty wspierają dwie różnoznaczne składnie — z kropką `obj.x = 10` oraz z nawiasami kwadratowymi `obj["x"] = 10`. Właściwości oraz ich wartości mogą być dodawane, zmienianie oraz usuwane w każdej chwili działania programu. Większość właściwości obiektu (oraz te dziedziczone w łańcuch prototypów) mogą być wymienione z wykorzystaniem pętli `for ... in`. JavaScript ma dość skromny wachlarz obiektów wbudowanych, między innymi `Function` (funkcje) oraz obiekty daty — `Date`. JavaScript oferuje również wbudowaną funkcje `eval`, która może wykonywać instrukcje dostarczone w postaci ciągów znaków podczas działania programu [8].

JavaScript jest także językiem funkcyjnym. Funkcję są typu pierwszej klasy. Oznacza to, że JavaScript wspiera przekazywanie funkcji jako argumentów do innych funkcji, zwracania ich jako wartości innych funkcji, przypisywania ich do zmiennych oraz zapisywanie ich w strukturach danych. JavaScript wspiera również funkcje anonimowe [6]. Funkcje w JavaScript jako takie posiadają właściwości oraz metody takie jak `.call()` i `bind`. Funkcje zagnieżdżone to funkcje zdefiniowane wewnątrz innych funkcji. Funkcja taka jest każdorazowo tworzona podczas wywołania funkcji zewnętrznej, w której została ona zdefiniowana. Dodatkowo każda tworzona funkcja tworzy *domknięcie*: zasięg zmiennych funkcji zewnętrznej, włączając w to zmienne lokalne oraz wartości argumentów, stają się częścią wewnętrznego stanu każdego wewnętrznego obiektu funkcji, nawet po zakończeniu wykonywania zewnętrznej funkcji [8].

JavaScript w procesie dziedziczenia wykorzystuje prototypy, w odróżnieniu od innych języków zorientowanych obiektowo wykorzystujących klasy. W JavaScript wykorzystując prototypy można za symulować wiele cech dziedziczenia opartego na klasach [8]. Nowe

wydanie ES6 wprowadza do JavaScript składnie umożliwiającą definiowanie klas.

Funkcje obok swojej typowej roli w JavaScript mogą także tworzyć obiekty. Poprzedzając wywołanie funkcji instrukcją wbudowaną `new` zostanie utworzona instancja prototypu dziedzicząca właściwości oraz metody z konstruktora (włączając w to właściwości z prototypu obiektu wbudowanego `Object`). ECMAScript 5 oferuje metodę `Object.create` pozwalającą na jawne tworzenie instancji bez automatycznego dziedziczenia z prototypu `Object`. Właściwość `prototype` konstruktora określa jaki obiekt zostanie użyty jako wewnętrzny prototyp nowo utworzonego obiektu. Nowe metody mogą zostać dodane poprzez modyfikowanie prototypu funkcji użytej jako konstruktor. Wbudowane konstruktory takie jak `Array` lub `Object`, także posiadają prototyp, który może być modyfikowany. Modyfikowanie prototypy `Object` jest uznawane za złą praktykę ponieważ prawie wszystkie obiekty w JavaScript dziedziczą metody oraz właściwości z prototypu obiektu `Object` [8].

W odróżnieniu od wielu języków zorientowanych obiektowo w JavaScript nie ma różnicy pomiędzy definicją funkcji oraz definicją metody. Różnica pojawia się podczas wywołania funkcji oraz metody. Kiedy funkcja wywoływana jest jako metoda obiektu, słowo kluczowe `this` wewnątrz ciała funkcji jest powiązane z obiektem na rzecz, którego dana metoda została wywołana [8].

JavaScript posiada wbudowane implementacje, oparte na funkcjach, wzorców takich jak *cecha* oraz *domieszka*. Jawna, oparta na funkcjach, delegacja nie wspiera kompozycji, natomiast nie jawna delegacja ujawnia się za każdym razem gdy przechodzony jest łańcuch prototypów np. w celu znalezienia metody, która nie należy bezpośrednio do obiektu. Gdy metoda zostanie odnaleziona wywoływana jest w kontekście danego obiektu. Dlatego dziedziczenie w JavaScript jest realizowane przez delegacje, która polega na przypisaniu właściwości do prototypu konstruktora funkcji [8].

Typowo JavaScript uruchamiany jest w jakimś środowisku np. w środowisku przeglądarki internetowej, które dostarcza obiekty oraz metody, przez które uruchamiany skrypt może oddziaływać na środowisko np. obiekt DOM strony internetowej.

JavaScript przetwarza wiadomości z kolejki — po jednej na raz. Podczas ładowania nowej wiadomości, JavaScript wywołuje funkcję powiązaną z daną wiadomości tworząc ramkę stosu wywołania (są to argumenty funkcji oraz zmienne lokalne). Stos wywołania zmniejsza się oraz powiększa zgodnie z potrzebami wywołanej funkcji. Nazwane jest to pętlą zdarzeń, opisywaną także jako „działaj do ukończenia”, ponieważ każda wiadomość jest całkowicie przetwarzana zanim przejdziemy do kolejnej wiadomości. Jednakże wbudowany model konkurencji nadaje pętli zdarzeń charakter nie blokujący. Operacje wejścia/wyjścia realizowane są z użyciem zdarzeń oraz wywołań zwrotnych (funkcji zwrotnych). Oznacza to, że JavaScript może przetworzyć kliknięcie myszy podczas czekania na dane z zapytania

do bazy danych [8]

Do funkcji może zostać przekazana nie określona ilość argumentów. Funkcja ma do nich dostęp poprzez parametry oraz także przez lokalny obiekt `arguments` [8].

Jak wiele języków skryptowych, tablice jak i obiekty mogą zostać utworzone przez zwięzłą składnie. Te literały stanowią także podstawę formatu *JSON*[4] [8].

JavaScript wspiera również wyrażenia regularne w sposób podobny do *Perl*, z zwięzła oraz bogatą składnią do manipulowania tekstem, znacznie bardziej zaawansowaną niż wbudowane funkcje do obróbki łańcuchów znaków.

2.2 Meteor.js

Meteor.js (Meteor, MeteorJS) jest to otwarto źródłowy framework aplikacji sieciowych napisany w JavaScript z wykorzystaniem *Node.js*. Meteor pozwala na szybkie prototypowanie oraz tworzenie między platformowego kodu (aplikacje sieciowe, Android, iOS). Framework wykorzystuje *MongoDB*, używa protokołu *DDP*[5] oraz wzorca *publikacji i subskrypcji* do automatycznej propagacji zmian w danych do klientów bez potrzeby pisanie kodu synchronizującego taką propagację. Po stronie klienta, Meteor wykorzystuje *jQuery*. Meteor jest rozwijany prze *Meteor Development Group*. Meteor po raz pierwszy został publicznie pokazany w grudniu 2011 pod nazwą *Skybreak* [9].

Meteor jest niepokojącą (w dobrym znaczeniu) technologią. Umożliwia budowanie aplikacji nowego typu, które są szybsze oraz prostsze w tworzeniu. Wykorzystuje nowoczesne techniki takie jak reaktywność po stronie klienta oraz serwera (*Full Stack Reactivity*), kompensacja opóźnień (*Latency Compensation*) oraz *data on the wire* - Meteor nie wysyła danych w postaci HTML, serwer wysyła dane i pozwala klientowi je renderować [4].

Meteor pozostaje wierny następującym zasadom [2]:

- *Data on the Wire* — Meteor nie wysyła HTML przez sieć. Serwer wysyła dane i pozwala klientowi je renderować.
- *Jeden język* — Meteor pozwala pisać stronę klienta jak i serwera w języku JavaScript.
- *Wszechobecna baza danych* — Meteor pozwala na użycie tych samych metod dostępu do danych po stronie klienta oraz serwera.
- *Kompensacja opóźnień* — Po stronie klienta Meteor wstępnie pobiera dane oraz symuluje modele aby wykonywane metody serwerowe zwracały wynik natychmiast.

[4] JavaScript Object Notation - lekki format wymiany danych komputerowych, jest to format tekstowy. Opisany w RFC 4627

[5] Distributed Data Protocol - protokół klient–serwer dla zapytań oraz aktualizacji bazy danych po stronie serwera oraz synchronizacji tych aktualizacji wśród innych klientów

- *Korzysta z ekosystemu* — Meteor jest projektem otwarto źródłowym (open source) oraz integruje się z istniejącymi otwarto źródłowymi narzędziami oraz frameworkami.

- *Prostota to produktywność* — Najlepszym sposobem aby coś wyglądało na proste to zrobić to tak aby było proste. Główne funkcjonalności Meteora mają czyste, klasycznie piękne API.

Koncepcja „danych w kablu“ *Data On the Wire* jest bardzo prosta i powstała z zagnieżdżonego wzorca model – widok – kontroler — *MVC*. Zamiast przetwarzania przez serwer każdego żądania, renderowania treści i wysyłania HTML do klienta, serwer wysyła same dane i pozowana klientowi zdecydować co z nimi zrobić [4].

To rozwiązanie w Meteorze zostało zaimplementowane z użyciem *Distributed Data Protocol* (DDP) — protokołu klient – serwer dla zapytań oraz aktualizacji danych po stronie serwera w bazie danych oraz synchronizacji tych zmian pośród innych klientów. Protokół ten oparty jest o składnie JSON oraz komunikuje się z serwerem w sposób podobny do protokołu *REST*[6]. Dodawanie, usuwanie oraz aktualizację są rozsyłane przez sieć oraz obsługiwane przez docelowe urządzenia, usługi oraz klientów. DDP używa *WebSockets*[7] zamiast HTTP, dane pomogą być wysyłane kiedy tylko dane ulegną modyfikacji [4].

Najważniejszą zaletą protokołu DDP to sposób komunikacji. Nie ma znaczenie jaki system wysyła lub odbiera dane może to być serwer, klient, usługa sieciowa - wszystkie one używają tego samego protokołu. Oznacza to, że żaden z systemów nie wiem czy inne systemy to serwery czy klienci. Z wyjątkiem przeglądarki, każdy z systemów może być serwerem lub pełnić role klienta. Cały ruch generowany przez protokół wygląda tak samo i jest traktowany w ten sam sposób. Tradycyjna koncepcja jednego serwera dla jednego klienta staje się przestarzała. Możliwe jest połączenie wielu serwerów, które pełnią określone role. Klient może być połączony z wieloma serwerami, z którymi może pracować w różny sposób [4].

W skład frameworka wchodzi także ciekawa technologia — *mini baza danych*. Jest to „leka” wersja normalnej bazy danych rezydującej w pamięci po stronie klienta. Klient zamiast wysyłać żądania do serwera może zmieniać dane bezpośrednio w mini bazie danych, które znajduje się po stronie klienta. Po aktualizacji danych w mini bazie danych ta automatycznie synchronizuje się z serwerem, który posiada właściwą bazę danych, wy-

[6]Representational State Transfer - (zmiana stanu poprzez reprezentacje) — styl architektury oprogramowania powstały z doświadczeń podczas opracowywania specyfikacji protokołu HTTP dla systemów rozproszonych. REST wykorzystuje między innymi jednorodny interfejs, bezstanową komunikację, zasoby, reprezentacje, hipermedia [11]

[7]Technologia zapewniająca dwukierunkowy kanał komunikacji za pośrednictwem jednego gniazda TCP. Stworzona głównie jako kanał komunikacji pomiędzy przeglądarką internetową a serwerem internetowym. Może być także stosowana w innych aplikacjach typu klient lub serwer. [12]

korzystując protokół DDP. Meteor w roli mini bazy danych wykorzystuje *Minimongo*, właściwa baza po stronie serwera to *MongoDB* [4].

Aktualizacja danych po stronie klienta, najpierw jest realizowana w instancji Minimongo. Klient pozostawia synchronizację zmian Minimongo (z wykorzystaniem protokołu DDP). Jeżeli serwer zaakceptuje zmiany, rozsyła je do wszystkich połączonych klientów, włączając w to także tego, który dokonał zmian. Natomiast jeżeli serwer odrzuci zmiany, bądź pojawi się nowszy zestaw danych od innego klienta, Minimongo na kliencie zostanie skorygowane, co spowoduje aktualizację wszystkich elementów interfejsu użytkownika powiązanych z tymi danymi. Rozwiązanie to w połączeniu z asynchronicznością (realizowaną z wykorzystaniem DDP) jest przełomowe. Oznacza to, że klient nie musi czekać na odpowiedź z serwera. Klient aktualizuje interfejs użytkownika w oparciu o dane zawarte w instancji Minimongo. W przypadku gdy aktualizacja danych została odrzucona przez serwer lub inne zmiany dotarły z serwera, klient zostanie zaktualizowany zaraz po otrzymaniu takiej informacji z serwera.

W przypadku powolnego połączenia z internetem lub jago całkowitego jego braku Meteor kompensuje to poprzez wysyłanie danych do Minimongo oraz natychmiastowymi zmianami w interfejsie użytkownika. W normalnym środowisku klient – serwer żadna aktualizacja nie została by wykonana, interfejs użytkownika pokazywał by stan ładowania podczas kiedy klient czekał by na odpowiedź z serwera. Wszystkie dokonane zmiany miałby swoje odzwierciedlenie w interfejsie użytkownika w oparciu o Minimongo. Kiedy połączenie z Internetem zostało by przywrócone, wszystkie zakolejkowane zmiany zostaną wysłane na serwer, serwer natomiast prześle autoryzowane zmiany do klienta. Meteor pozwala klientom „brać informacje na wiarę". Jeżeli napotkamy problem to dane otrzymane z serwera naprawią nieścisłości. Jednak przez większość czasu zmiany dokonywane na kliencie są natychmiast wysłane na serwer oraz przez niego rozgłaszane do innych klientów. Aby osiągnąć takie zachowanie należy zdefiniować nową kolekcję w poniższy sposób:

Listing 2.1: Definicja kolekcji

```
1  Articles = new Mongo.Collection("Articles")
```

Ta jedna linijka deklaruje kolekcję `Articles`, której swoją wersję będą posiadały zarówno klient jak i serwer — lecz traktują ją w odmienny sposób. Klient subskrybuje zmiany ogłaszane przez serwer i aktualizuje odpowiednio swoją kolekcję. Serwer natomiast publikuje zmiany oraz nasłuchuje zmian z klientów aktualizując swoją kolekcję.

Jedną z najważniejszych cech Meteora jest reaktywność. Po stornie klienta, Meteor oferuje bibliotekę *Blaze*, która używa szablonów HTML oraz *helper'ów*[8] JavaScript do wykrywania zmian oraz renderowania danych. Kiedy dane zostaną zmienione, helpery same ponowienie się uruchamiają zmieniając, usuwając lub dodając właściwe elementy

[8]Pomocniczy kod JavaScript głównie zwracający wartości do szablonu HTML

interfejsu użytkownika na podstawie struktury zapisanej w szablonach. Funkcje, które same ponownie się uruchamiają zaliczane są do *reaktywnych obliczeń* [4].

Meteor oferuje reaktywne obliczenia zarówno po stronie serwera jak i klienta, bez potrzeby odwoływania się do interfejsu użytkownika. Wchodząca w skład framework'a biblioteka *Tracker* oraz jej helpery także wykrywają zmiany w danych oraz potrafią się same ponownie uruchomić. Ponieważ JavaScript używane jest po stronie serwera oraz klienta, bibliotekę Tracker można używać po obu stronach. Ponieważ używamy tego samego języka po stronie klienta (oraz możemy używać tego samego kod) jak i serwera nazywamy to *full stack reactivity* - reaktywność po stronie klienta jak i serwera.

Ponowne uruchamianie funkcji gdy dane ulegną zmianie ma bardzo dużą zaletę dla programisty. Kod pisany jest deklaratywnie, Meteor sam zajmuję się reaktywnością. Programista określa jak dane mają byś wyświetlane a Meteor sam zajmuję się zmianami w danych. Ten deklaratywny sposób pisania, zazwyczaj osiągany jest z wykorzystaniem szablonów. Szablony działają w oparciu o wiązanie danych z widokiem[9] — są to współdzielone dane, które w zależności od ich zmian będą przedstawiane w różny sposób [4].

2.3 MongoDB

Meteor po stronie serwera używa MongoDB, po stronie klienta używana jest Minimongo — wersja MongoDB. Meteor może używać innych baz NoSQL[10] lub baz zorientowanych na dokumenty. Dzięki użyciu MongoDB programy stają się prostsze, łatwiejsze w tworzeniu. MongoDB doskonale sprawdza się jako szybki oraz lekki magazyn danych.

Tradycyjnie większość danych przechowywana jest z użyciem modelu relacyjnego z wykorzystanie relacyjnych baz danych. Relacyjny model wraz ze wszystkimi powiązanymi z nim zasadami, relacjami, logiką oraz składania jest integralną oraz nieocenioną częścią współczesnej informatyki. Sztywna struktura baz relacyjnych, z dokładnymi wymaganiami co do każdego rekordu, relacje oraz asocjacje umożliwiają szybkie wyszukiwanie, skalowalność oraz dostarczają potencjał do głębszej analizy danych.

Taka dokładność nie jest zawsze potrzebna. Dla prostych aplikacji, pełno wymiarowa relacyjna baza była by nad wymiarowa. W niektórych sytuacjach bardzie efektywne jest wykorzystanie elastycznego mechanizmu przechowywania danych, który umożliwia łatwe rozszerzanie schematu danych bez potrzeby znaczących zmian w tworzonym programie.

Jeżeli byśmy chcieli dodać nową właściwość do obiektu `Article` o wiele łatwiejsze było by dodanie takiej właściwości do obiektu i zrzucenie tego na bazę danych niż przepisanie kodu dotyczącego baz danych, dodanie kolumny, aktualizacja wszystkich zapytań

[9] View data bindings
[10] Not Only SQL

SQL oraz upewnienie się, że wszystkie poprzednie wpisy mają nową właściwość.

W takich sytuacjach bardzo dobrze sprawdzają się bazy danych zorientowane na dokument. W tym typie baz danych dane zapisywane są w postaci dokumentów złożonych z par klucz – wartość. Tak przechowywane dokumenty nie muszą mieć z góry ustalonej struktury. Dokumenty mogą przyjmować różne struktury, w kluczach mogą być przechowywane różne wartości — dla bazy danych nie jest to żadne problem. Jednak każdy dokument musi posiadać unikalny klucz za pomocą, którego może on zostać pobrany z bazy danych [4]. Dla przykładu, jeden dokument może mieć bardzo prostą postać:

```
{name: phone_number}
```

Następny dokument może przybrać formę bardziej skomplikowaną (wewnątrz tej samej bazy danych oraz kolekcji) — składający się z zagnieżdżonych list, obiektów oraz innych elementów [4]:

```
{ people: [
  {
    firstname:"STEVE",
    lastname:"Scuba",
    phones :[
      {type:cell, number:8888675309},
      {type:home, number:8005322002}
    ]
  },
    {
      firstname:...
...
  }]
}
```

Dokumenty mogą przybierać dowolną strukturę, o ile każdy z nich posiada unikalny klucz, umożliwiający ich pobranie. Brak struktury wpływa negatywnie na wydajność zapytań, sortowania oraz aktualizowania dokumentów. Podczas tworzenia aplikacji możemy zidentyfikować najczęściej pojawiające się zapytania i zmodyfikować tak strukturę dokumentów by baza była w niektórych zastosowaniach znacznie wydajniejsza od rozwiązań relacyjnych. Dodatkową zaletą wykorzystania takich baz jest szybkość oraz łatwość pisania aplikacji [4]. Duża elastyczność baz zorientowanych na dokumenty ułatwia szybkie i łatwe zmiany a dostarczane wraz z Meteorem biblioteki pozwalają nie martwić się o połączenie oraz strukturę bazy danych. Jedynie to co jest potrzebne to wysoko poziomowe zrozumienie jak wyszukać, dodać oraz zmodyfikować dokumenty, resztą zajmie się sam Meteor.

MongoDB to otwarto źródłowa baz danych typu NoSQL. Jest to baz danych zorientowana na dokumenty z zaawansowanymi funkcjonalnościami takim jak indeksy, replikacje, zapytania ad-hoc, agregacja danych, zapytania do zagnieżdżonych dokumentów, równoważenie obciążenia, możliwość zapisywania plików. Cechuję ją duża skalowalność, wydajność, brak określonej struktury. Została napisana w języku C++. Dane zapisywane są jako dokumenty w stylu JSON. Taki sposób umożliwia aplikacją bardziej naturalne ich przetwarzanie, przy zachowaniu możliwości tworzenia hierarchii oraz indeksowania. Wewnętrzny język do definiowania zapytań oraz funkcji agregujących to JavaScript wykonywany bezpośrednio przez serwer MongoDB. Dokumenty zapisywane są w MongoDB w logicznych grupach nazywanych kolekcjami.

MongoDB posiada ograniczone wsparcie dla transakcji — zasięg jest ograniczony do jednego dokumentu. Zmiany w tym dokumencie mogą być bardzo skomplikowane. Z tego powodu część wdrożeń ogranicza zastosowanie MongoDB do niekrytycznych danych informacyjnych, powierzając obsługę krytycznych operacji relacyjnym odpowiednikom. MongoDB wspiera w niewielkim stopniu kodowanie UTF-8, co jest problemem w przechowywaniu tekstu w języku innym niż angielski. Do sortowania łańcuchów znaków używana jest funkcja *memcmp*, która nie obsługuje poprawnie danych w UTF-8 w różnych ustawieniach regionalnych. Wprowadzenie zmian jest planowane, niestety wiąże się to – według twórców – z wprowadzeniem szeregu poważnych zmian. Z tego powodu termin wydania wersji z poprawną obsługą UTF-8 jest nieokreślony [10].

2.4 HTML5, CSS3, less, Bootstrap, AdminLTE

2.5 Git oraz GitHub

Bibliografia

[1] Mike Cantelon i Marc Harter i TJ Holowaychuk i Nathan Rajlich. *Node.js w akcji*. Wydawnictwo Helion, Gliwice, 2014.

[2] Meteor. Meteor Docs. `http://docs.meteor.com/#/full/quickstart`, Grudzień 2015.

[3] Stoyan Stefanov. *JavaScript programowanie obiektowe*. Wydawnictwo Helion, Gliwice, 2010.

[4] Isaac Strack. *Getting Started with Meteor.js JavaScript Framework*. Packt Publishing Ltd., Birmingham, United Kingdom, 2015.

[5] Wikipedia. ECMAScript. `https://en.wikipedia.org/wiki/ECMAScript`, Listopad 2015.

[6] Wikipedia. First-class function. `https://en.wikipedia.org/wiki/First-class_function`, Listopad 2015.

[7] Wikipedia. Intranet. `https://en.wikipedia.org/wiki/Intranet`, Listopad 2015.

[8] Wikipedia. JavaScript. `https://en.wikipedia.org/wiki/JavaScript`, Listopad 2015.

[9] Wikipedia. Meteor (web framework). `https://en.wikipedia.org/wiki/Meteor_(web_framework)`, Grudzień 2015.

[10] Wikipedia. MongoDB. `https://pl.wikipedia.org/wiki/MongoDB`, Grudzień 2015.

[11] Wikipedia. Representational State Transfer. `https://pl.wikipedia.org/wiki/Representational_State_Transfer`, Grudzień 2015.

[12] Wikipedia. WebSocket. `https://pl.wikipedia.org/wiki/WebSocket`, Grudzień 2015.