Spis treści

Rozdział 1

Wstęp

1.1 Cel pracy

Celem pracy jest przedstawienie różnic, podobieństw oraz wspólnych konceptów między wybranymi frameworkami technologii Node.js. Od momentu jej ukazania się powstało wiele narzędzi ułatwiających oraz poprawiających jakość procesu deweloperskiego. Niektóre z nich skupiają się na usprawnianiu podobnych, pokrewnych lub identycznych przypadków użycia. Aby unkinąc nie trafnego wdrożenia wybranego narzędzia w opracowywany projekt, należy rozumieć kiedy wybrana metoda sprawdzi się najlepiej, a kiedy warto zastosować inne rozwiązanie. Nawet w przypadku użycia tego samego języka programowania przeniesienie aplikacji pomiędzy frameworkami może być kosztownym oraz czasochłonnym zadaniem. Z tego powodu powstała potrzeba zakreślenia, które wyjście najlepiej sprawdza się dla określonego projektu lub jego części.
W pracy zostały zawarte kryteria wyboru technologii, specyfikacje ocenianych parametrów, analiza frameworków oraz podsumowanie przeprowadzonych badań.

1.2 Przeznaczenie technologii Node.js

Node.js jest cross-platformowym, działającym niezależnie od środowiska językiem programowania, napisanym w językach C/C++ oraz JavaScript, wydanym 27 marca 2009 roku, zaprojektowanym przez Ryana Dahla. Początkowym przeznaczeniem technologii było tworzenie serwerów i narzędzi sieciowych, działających po stronie serwera, lecz wraz z jej rozpowszechnieniem się możliwości te znacznie się poszerzyły oferując wytwarzanie między innymi aplikacji desktopowych (Electron), mobilnych (React Native, Cordova) lub rozwiązań embedded (Onoff). Przed jej

powstaniem powstaniem kod w języku JavaScript był wykonywany głównie przez przeglądarkę internetową po stronie klienta. Pozwalało na bezproblemową manipulację kodem źródłowym strony przez użytkownika, dając możliwość wykonywania złośliwych skryptów, naruszenie bezpieczeństwa baz danych lub uzyskania dostępu do chronionych zasobów serwera. Środowisko Node.js może działać niezależnie od środowiska uruchomieniowego. Jest ono zgodne z wieloma systemami operacyjnymi jak Linux, macOS, Microsoft Windows, NonStop, czy serwerami Unix. Język ten cieszy się dużą popularnością oraz pozytywnym odbiorem wśród użytkowników, dzięki czemu, mimo względnie krótkiego okresu życia środowiska, zaowocowało ogromną ilością projektów open-source, tysiącami członków należących do społeczności okołojęzykowej oraz powstaniem wydarzeń poruszających tematy okołośrodowiskowe, takimi jak NodeConf, Node Interactive lub Node Summit. Obecnie wiele największych firm korzysta z serwerów napisanych w języku Node.js. Ich przykładami są między innymi Groupon, IBM, Linkedln, Microsoft, Netflix, PayPal, Yahoo. Najpopularniejszymi API wspierającymi edycję oraz debugowanie kodu Node.js są Atom, Visual Studio Code czy WebStorm.
Node.js zalecany jest do tworzenia aplikacji:

- z dużą liczbą operacji wejścia/wyjścia,
- strumieniowania danych np. video,
- Single Page Applications (SPA),
- udostępniających API w formacie JSON,
- z intensywną wymianą danych w czasie rzeczywistym na wielu urządzeniach, np. portalach społecznościowych.

Ponieważ jest on szybki i lekki, może być stosowany do pisania między innymi bramki API. Jest to skrót od Application Programming Interface; opisuje, jak poszczególne elementy lub warstwy oprogramowania powinny się komunikować. W praktyce to najczęściej biblioteka oferująca metody, które umożliwiają realizację określonych zadań. Node.js pozwala na zoptymalizowanie pracy oraz uzyskanie skalowalności dzięki asynchronicznemu przetwarzaniu danych dostarczanych do aplikacji, w związku z czym idealnie nadaje się do obsługi komunikacji wymagającej pracy w czasie rzeczywistym. W przeciwieństwie do języków gdzie program jest wykonywany linia po linii, funkcje napisane w Node.js nie wykonują się synchronicznie, korzystając z tak zwanych wywołań zwrotnych (ang. callback). Dzięki temu nie powstaje problem blokowania określonych funkcjonalności programu w czasie pracy innych, niezależnych jego części. Przy pomocy wywołań zwrotnych możemy

zapewnić zasygnalizowanie uzyskanych wyników lub zwrócenie, bądź obsługę błędu powstałego w czasie działania bloku kodu.

1.3 Potrzeba rozwoju narzędzi deweloperskich

Nieustany rozwój lub powstawanie nowych narzędzi wywiera ciągłą potrzebę nauki na inżynierach. Bez przerwy pojawiają się nowe rozwiązania, które sprawiają że dotychczasowe odchodzą w niepamięć lub ich użycie kojarzy się z zacofaniem oraz pozostaniem w tyle. Absolutnie nie powinno uważać się tego zjawisko za negatywne. Powstanie nowego narzędzia jest inicjowane przez kogoś kto zauważył pewną regularność danego przypadku użycia. Nie tylko sprawia to że mniej czasu należy poświęcić na uzyskanie pewnej określonej funkcjonalności, ale przede wszystkim daje możliwość skorzystania z wiedzy i praktyk wielu osób mających doświadczenie w danej dziedzinie, które napotykały dokładnie te same problemy. Korzystamy wówczas nie tylko z najlepszych praktyk programistycznych, ale także zapewniamy sobie dostęp do przetestowanych sprawdzonych metod. Bardzo rzadkim zjawiskiem jest natrafienie na jakąś nieścisłość we frameworku, wywołującą niepoprawne działanie własnego kodu. Nawet w takich przypadkach błędy te są bardzo szybko poprawiane przez specjalistów, którzy zdejmują z nas odpowiedzialność w utrzymaniu części kodu. Rozwiązania używane globalnie zapewniają róznież szybsze wdrażanie się w nowe projekty. Nowe osoby zostają odciążone od poznawania części aplikacji, ponieważ korzystały już z tych samych rozwiązań. Należy natomiast upewnić się czy wykorzystując dane gotowe narzędzie nie dołączamy do istniejącego projektu mnóstwa innych, zbędnych zasobów. Wiele funkcji dostarczanych nie jest wykorzystywanych i może zostać nie potrzebnie zawarta w końcowym produkcie. W celu uniknięcia tego możemy skorzystać z wybranych narzędzii przetwarzających kod (bundler tj. webpack), które odrzucą nieużywane części kodu minimalizując rozmiar aplikacji.

Rozdział 2

Kryteria wyboru freameworków

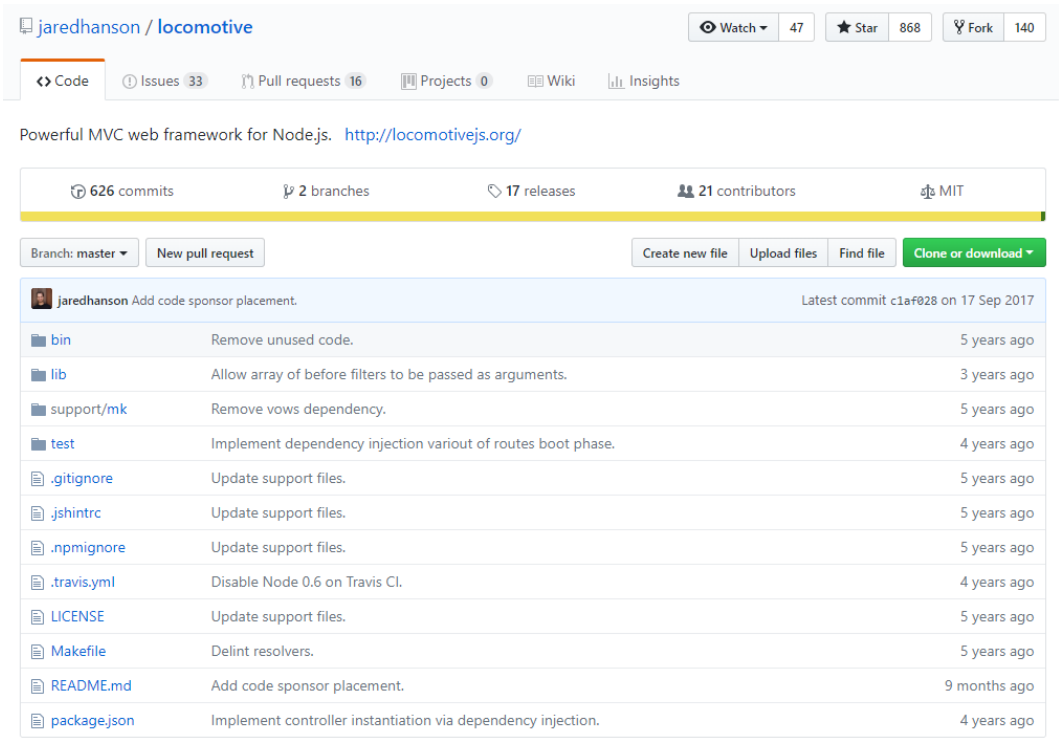
2.1 Co to jest framework

Framework jest to pewna platforma programistyczna oferująca nam zbiór najczęściej generycznej funkcjonalności. Określa sposób tworzenia produktu w określonym środowisku. Jest uniwersalny, reużywalny czasem stanowi część danej platformy. Może być zarówno tylko zbiorem bibliotek lub całego szeregu narzędzi jak kompilatory, debuggery w celu dostarczenia najprostrzego oraz jednocześnie gwarantującego jak najszersze możliwości tworzenia całości lub części systemów czy produktów. Pozwalają one na skupienie się na wysokopoziomowym określaniu wymagań, zamiast na implementacji bazowej funkcjonalności lub określonych standardów. Na przykład w przypadku korzystania z frameworka webowego mozemy stworzyć nasz produkt bez potrzeby tworzenia całości architektury komunikacji między warstwami systemu, czy bez potrzeby martwienia się o zarządzanie częściami widoku strony internetowej. Koncepcji frameworków toważyszy zasada dopasowywania się pod konkretne potrzeby. Znaczy to że kiedy nauczymy wykorzystywać dane narzędzie dla danego projektu z nie wielkimi zmianami możemy wykorzystać to samo podejście przy kolejnych. Znacznie zmniejszamy tym sposobem koszta, tym samym czas wytwarzania gotowych produktów. Największym zarzutem jakie stawia się osobą korzystającym z pomocy frameworków jest utrata kontroli lub w przypadku braku znajomości działania narzędzia całkowity jej brak. Należy więc za tem przed skorzystaniem z gotowego rozwiązania upewnić się że spełnia ono nasze oczekiwania oraz nie spowoduje znacznego spadku wydajności. Architekture frameworków możemy podzielic na częsci "frozen" i "hot" spots. Pierwsze dotyczą ogólnej architektury, niezmiennych konceptów i założeń nią kierujących. Określa relacje między komponentami. Pozostają one niezmienne podczas całkowitego życia aplikacji. Druga część odnosi się do części w której programista może dodać, integrowac się z danym

narzędziem określając własną funkcjonalność.

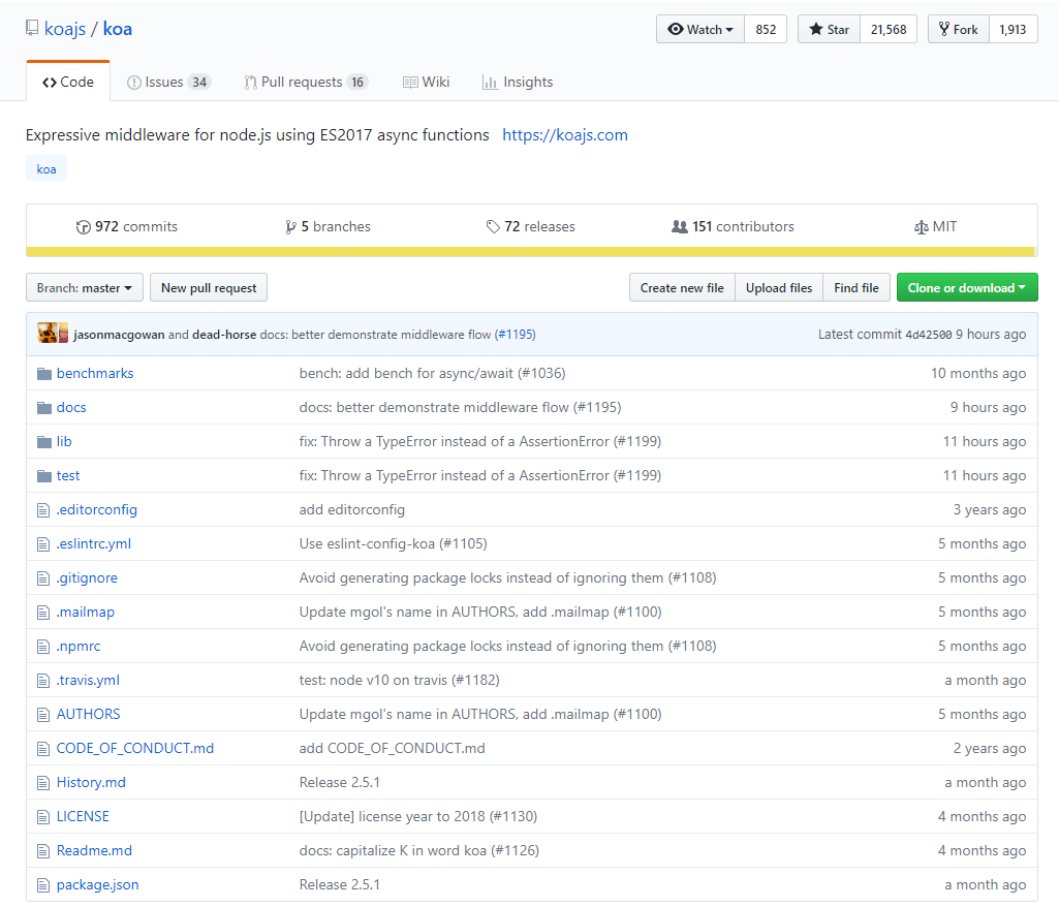
2.2 Poszukiwania

Wszystkie aktualnie liczące się frameworki odnośnie technologi Node.js możemy znaleźć na stronie "http://nodeframework.com/". Strona jest prowadzona przez społeczeństwo deweloperów. Może zostać zaktualizowana poprzez zaakceptowanie przez twórce projektu Azat Mardan'a merge requesta w serwisie Github. Oznacza to, że strona posiada aktualne infromacje na temat używanych na świecie rozwiązań. W celu wybrania konkretnych frameworków do badań warto zwrócić uwagę na pare konkretnych kwesti.
Analizowane narzędzie z pewnością powinno być używane przez deweloperów na liniach produkcyjnych. Gwarantuje nam to że jest to narzędzie sprawdzone w rzeczywistym życiu. Powinno cieszyć się ono również względnie dużą popularnością, aby upewnić się że radzi sobie przy różnych wielkościach projektów, w różnym stadium zaawansowania, oraz różnym poziomie skomplikowania. Najlepiej byłoby, aby framework był wciaż rozwijany, ponieważ znaczyło by to że jego funkcjonalność jest wciąż poszerzana oraz że nie jest przestażały. Parametry te możemy sprawdzić wchodząc na repozytoria frameworków.



Rysunek 2.1: Strona projektu Locomotive w serwisie Github
Źródło: https://github.com/jaredhanson/locomotive/

Przykładowo możemy sprawdzić, że Locomotive posiada tylko 868 gwiazdek oraz że ostatnia aktualizacja była 17 października 2017 roku (dane na dzień 06.06.2018), więc możemy wywnioskować że nie cieszy się on dużą popularnością oraz że nie jest już odświeżany.



Rysunek 2.2: Strona projektu Koa w serwisie Github
Źródło: https://github.com/koajs/koa/

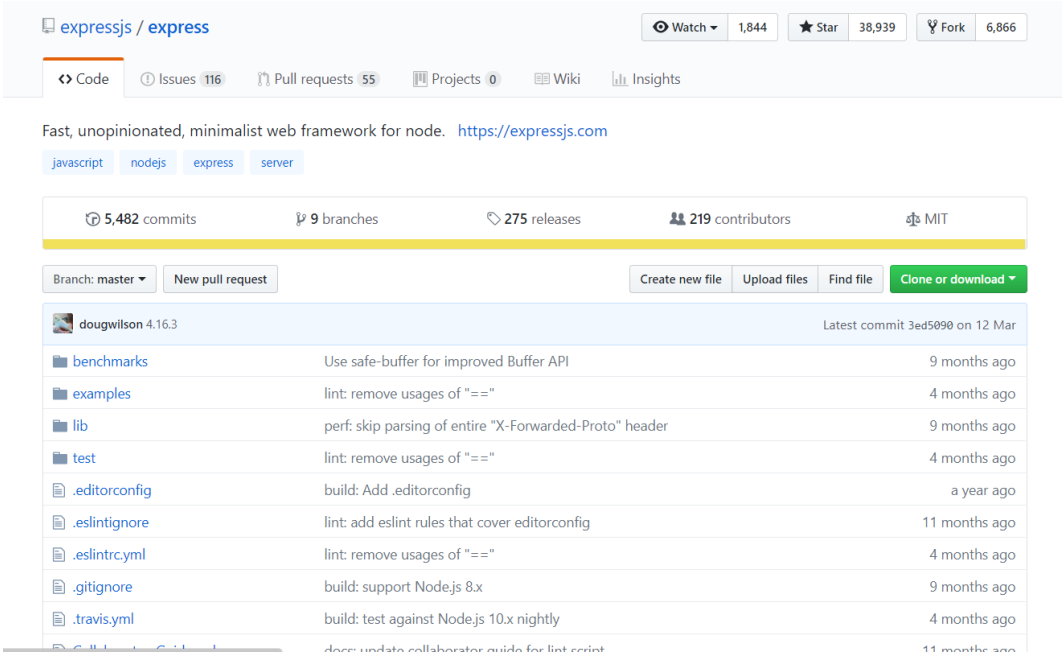
Dla porównania framework Koa posiada 21568 gwiazdek oraz był aktualizowany w dniu pisania pracu (dane na dzień 06.06.2018). Możemy wieć założyć że jest wciąż popularny oraz utrzymywany.

2.3 Wybrane frameworki

Ilość dostępnych narzędzi nie pozwala niestety na porównanie wszystkich rozwiązań. W związku z czym zdecydowałem się przeanalizować w pracy 3 wybrane - Express, Sails oraz Meteor.

2.3.1 Express

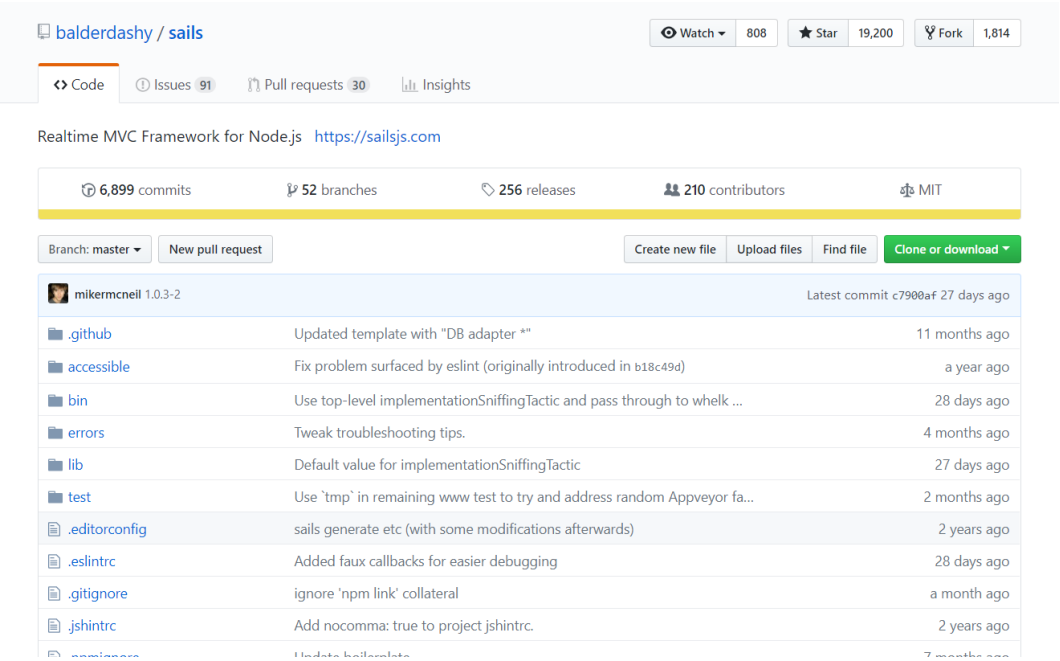
Express wybrałem ze względu na jego ogromną popularność. Repozytorium projektu posiada prawie 40000 gwiazdek w serwisie Github oraz jest najpopularniejszy ze wszystkich frameworków w serwisie Stackoverflow (dane na dzień 06.06.2018). Ze wszystkich konkurencyjnych rozwiązań jest najpopularniej wymienianą technologią w ofertach pracy. Został wybrany aby sprawdzić czy wiodące rozwiązanie jest słusznie liderem na rynku.



Rysunek 2.3: Strona projektu Express w serwisie Github
Źródło: https://github.com/expressjs/express

2.3.2 Sails

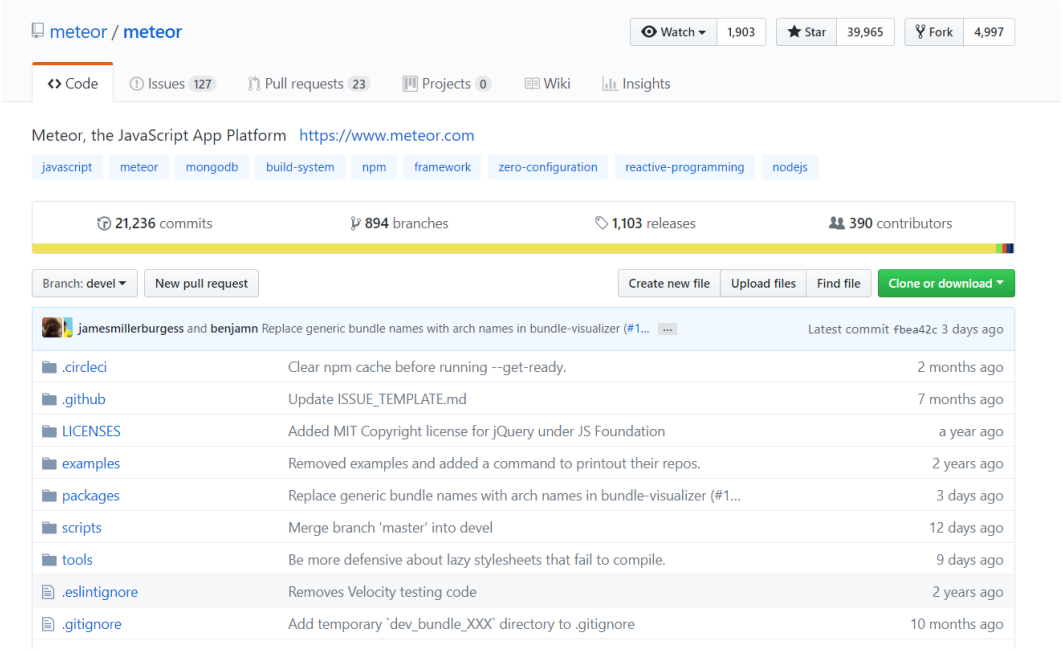
Sails został wybrany ze względu na wyjątkowo niski próg wejścia. Framework dostarcza nam gotowy szkielet aplikacji, który musimy dostosować pod nasze wymagania. Nie zyskał on jeszcze tak dużej popularności jak pozostałe rozwiązania, jednak dzięki prostocie zbiera on coraz więcej fanów. Jego użytkownicy podkreślają skalowalność, szybkość oraz uzyskaną produktywność jako największe jego zalety. Repozytorium projektu posiada prawie 20000 gwiazdek w serwisie Github (dane na dzień 06.06.2018).



Rysunek 2.4: Strona projektu Sails w serwisie Github
Źródło: https://github.com/balderdashy/sails

2.3.3 Meteor

Meteor wybrałem ponieważ mimo odcięcia się od tradycyjnego środowiska technologii Node.js, cieszy się on dużą popularnością oraz dostarcza nam możliwość wytwarzania jednocześnie częsci backendowej, frontendowej a nawet aplikacji mobilnych przy użyciu jednej bazy kodu. Framewok wprowadza wiele własnych narzędzi (np. własnego menadżera zależności, któremu w czystym Node.js odpowiada narzędzie Npm) oraz gotowych modułów zarówno widoku jak i logiki pozwalając na efektywniejsze wytwarzanie gotowego produktu. Repozytorium projektu posiada blisko 40000 gwiadek w serwisie Github (dane na dzień 06.06.2018). Popularność Meteor'a jest więc równa popularności frameworku Express, mimo że jest on od niego 2 lata młodszy.



Rysunek 2.5: Strona projektu Meteor w serwisie Github
Źródło: https://github.com/meteor/meteor

2.4 Alternatywne rozwiązania

Rozdział opisuje alternatywne rozwiązania, które zostały wzięte pod uwagę podczas etapu poszukiwania.

2.4.1 Hapi

Zaprojektowany do wytwarzania nowoczenych serwisów stron internetowych. Posiada wiele zdefiniowanych reużywalnych modułów (walidacja, cache, autentykacja) wymagających tylko minimalnej konfiguracji. Pozwala on na zminimalizowanie budowy infrastruktury projektu, na korzyść skupienia się na tworzeniu logiki oraz funkcjonalności.

2.4.2 Koa

Stworzony przez twórców frameworka Express. Dostarcza możliwość wytwarzania częsci api serwisów. Po pierwszych prezentacjach Koa została przyjęta bardzo pozytywnie pozwalając na zminimalizowanie ilości pracy potrzebną na wytowrzenie produktu przy użyciu Express'a. Narzędzie jest jednak ciągle w fazie silnego rozwoju i nie dostarcza jeszcze tak szerokich jak Express możliwości. Z pewnością jest dla niego dobrą alternatywą w przypadku nieskomplikowanych serwisów.

2.4.3 Loopback

Framework fullstack'owy, zbudowany do tworzenia aplikacji mobilnych (systemy iOS oreaz Android) oraz aplikacji webowych wymagając bardzo niewielkich nakładów kodu. Do działania wykorzystuje Express. Dzięki narzędzią takim jak IBM API Connect możemy wytwarząć kod przy użyciu graficznego interfejsu.

Rozdział 3

Kryteria oceny

3.1 Skala oraz wagi ocen

Rozdział opisuje jakie parametry frameworków zostały przeanalizowane i skonfrontowane ze sobą. Frameworki zostały ocenione w skali od 1 do 3 pod kątem każdego z nich. Każdy z parametrów otrzymał wagę w skali od 1 do 3. Wagi oznaczają wpływ danego parametru na ocene podsumowującą. Końcowe oceny zostały wyliczone ze wzoru:

$$O = \frac{\sum_{i=1}^{n} p_i * w_i}{sw}$$

gdzie:

O - ocena końcowa

n - ilość parametrów

p - ocena parametru

w - waga parametru

sw - suma wag parametrów

Maksymalna ocena do uzyskania to 3.

3.2 Parametry

3.2.1 Popularność

Waga: 3

Popularność jest wskaźnikiem, który niesie wiele informacji na temat badanego frameworka. W najprostrzy sposób możemy ją sprawdzić za pomocą Google trends, w przypadku narzedzi open source przy użyciu statystyk repozytorium, a w przypadku pozostałych poprzez serwisy takie jak Stackoverflow. Dzięki popularności możemy wywnioskować wielokość społeczności jaka jest zainteresowana danym narzedziem. Wiąże się to z możliwym wsparciem, wartością rynkową oraz szerokością możliwych zastosowań jakie wspiera. Oczywistym jest że jeśli narzędzie posiada ogromną popularność zostało sprawdzone w dużej ilośći przypadków, a jeśli potrafi im sprostać umiejętności sprawnego posługiwania się nim stają się cenniejsze.

3.2.2 Rozmiar

Waga: 1

Rozmiar frameworku ma bezpośredni wpływ na rozmiar naszego końcowego produktu. Najbardziej pożądane jest aby narzędzie zajmowało jak najmniej miejsca. W celu zminimalizowania rozmiaru naszego produktu możemy użyć bundlerów takich jak np. webpack, które sprawią że do końcowego produktu zostanie załączona wyłącznie taka funkcjonalność biblioteki którą rzeczywiście wykorzystujemy. Warto o tym pamiętać w przypadku użycia dużych narzędzi dla prostych, niewymagających rozwiązań. Mimo takowej możliwości nie gwarantuje nam to zawsze małego rozmiaru produktu, dlatego rozmiar frameworku został wzięty pod uwagę.

3.2.3 Dokumentacja

Waga: 3

Nawet najbardziej użyteczne narzędzie nie mogłoby zaistnieć bez łatwo dostępnej, zrozumiale przekazanej dokumentacji. Jest ona niezbędna do przyswojenia proponowanych konceptów oraz obycia się z wybranym rozwiązaniem. Ocena dokumentacji dotyczy jej dostępności, szczegółowości oraz zrozumiałości.

3.2.4 Próg wejścia

Waga: 1

Próg wejścia oznacza ilość zagadnień wymaganych w celu opanowania danego frameworka oraz ilość czasu jaki należy poświęcić na jego opanowanie w stopniu conajmniej podstawowym. Pożądane jest aby był on jak najniższy; aby narzędzie nadawało się do użycia przez osoby jak najsłabiej znające koncepty programowania czy budowy systemów, niemal że natychmiast po chwilowym zapoznaniu się z dokumentacja. Ocena progu wejścia została wystawiona na podstawie trudności oraz czasu poświęconego na zaznajomienie się z analizowanym narzędziem w stopniu umożliwiającym stworzenie funkcjonalności przedstawionej w opisanej pracy.

3.2.5 Warstwa widoku

Waga: 2

Parametr opisuje w jakim stopniu badany framework zapewnia obsługę warstwy widoku wytwarzanej aplikacji. Możemy oczekiwać wielu możliwości obsługi warstwy widoku. Wydawać by się mogło, że najlepszą możliwością byłoby gdyby framework kompletnie integrował warstwe widoku z warstwą danych dostarczając określone, proste metody wzajemnego mapowania. Kompletnie nie porządane z koleji jest, aby do obsługi oraz integracji z warstwą widoku wytwarzanego produktu było wymagane użycie zewnętrznego, dodatkowego narzędzia. Ocena została wystawiona na podstawie możliwości jakie oferują analizowane frameworki bez użycia dodatkowych narzędzi.

3.2.6 Warstwa bazy danych

Waga: 2

Podobnie jak parametr warstwy widoku, parametr ten opisuje możliwość integracji frameworka z bazą danych. Najmniej pożądanym zachowaniem jest kiedy moduł obsługi bazy danych należy w całości dołączyć do rozwiązania, natomiast najbardziej pożądanym aby framework w pełni posiadał zaimplementowaną integracje przy użyciu np. komponentu DAO. Ocena została wystawiona na podstawie możliwości jakie oferują analizowane narzędzia bez użycia dodatkowych narzędzi.

3.2.7 Ilość kodu

Waga: 2

Parametr opisuje ilość kodu potrzebnego do napisania aby otrzymać ideantyczną funkcjonalność w analizowanych środowiskach. Ilość kodu pisanego przy użyciu tego samego języka oraz jego standardu z zachowaniem odpowiednich praktych może zostać przełożony na ilość pracy, a co za tym idzie czasu jaki należy poświęcić do osiągnięcia obranego celu. Napisany kod musi być zgodny ze stosowanymi, ogólnie uznanymi praktykami programistycznymi dla danych środowisk. Parametr opisuje również najważniejsze funkcjonalności frameworków, które umożliwiają zmiejszenie ilości wymaganego kodu. Oceny zostały wystawione na podstawie porównania uzyskanych wyników dla analizowanych narzędzi.

3.2.8 Rozszerzalność

Waga: 3

Parametry rozszerzalności dotyczy możliwości rozszerzania istniejącego kodu o nowe oraz łatwości zmian istniejących funkcjonalności. Rozszerzalny kod charakteryzuje czytelny, jasny sposób zapisu, reużywalność istniejących już modułów, klas lub metod oraz odporność na zmiany częsci kodu bez wprowadzania regresji w pozostałych. Taki sposób pisania gwarantuje łatwiejsze zarządzanie oraz utrzymanie kodu. Dobry framework powinien wymuszać na programiście pisanie modularnego, rozszerzalnego kodu lub przynajmniej dostarczać przejrzyste sposoby na jego osiągnięcie. Jako że zbadanie rozszerzalności kodu jest trudne i jego możliwość lub jej brak może zostać zauważona po odpowiednim rozwoju aplikacji ocena tego zagadnienia została wystawiona na podstawie tworzenia przykładowej funkcjonalności do analizy pozostałych parametrów.

3.2.9 Testowanie

Waga: 2

Parametr opisuje stopień trudności wytworzenia testów automatycznych kodu dla danego środowiska. Stworzenie automatycznych testów pozwala nam w najszybszy dostępny sposób sprawdzić czy testowane rozwiązanie zachowuje się zgodnie z założonymi oczekiwaniami. Znacznie skraca to czas wytwarzania oprogramowania dzięki natychmiastowej diagnostyce oraz możliwości wyeliminowania powstałych błedów. Analizowany framework powinien dostarczać metody tworzenia testów

automatycznych lub bezproblemową integracje w przypadku użycia zewnętrznego narzędzia. Powinien w możliwie jak najwięszkym stopniu nakłaniać do prowadzenia projektu zawierającego skrypty sprawdzające działanie kodu oraz promować popularne techniki prowadzenia testów. Ocena została wystawiona na podstawie stworzenia testów automatycznych dla analizownych środowiska przy użyciu dostarczonych przez framework możliwości.

3.2.10 Testy sprawnościowe

Waga: 3

Parametr opisuje ilość przetworzonych zapytań w ciągu określonego czasu. Kolejne zapytania zostają wysłane dopiero po otrzymaniu odpowiedzi na poprzednie. Możemy w ten sposób sprawdzić jak sprawnie aplikacja wykonana w konkretnym środowisku jest w stanie odpowiadać na nieustające zapytania końcowego użytkownika. Zapytania zostały dobranego dla różnych przypadków użycia:

Autoryzacja - uzyskanie autoryzacji na zapytanie zawierające login oraz hasło użytkownika

Zapis danych użytkownika - zapisanie danych tekstowych użytkownika w bazie danych

Proste zapytanie o zasób - uzyskanie dostępu do zasobu tekstowego z bazy danych przez użytkownika

Zapytanie o zasób - uzyskanie dostępu do zasobu multimedialnego (pliku w formacie png) z serwera

Streaming multimediów - uzyskanie dostępu do streamu multimedialnego (w formacie avi) z serwera

Okresy czasu dla których zostały wykonane testy to:

krótki - 2 sekundy

średni - 10 sekund

długi - 30 sekund

Oceny zostały wystawione na podstawie porównania uzyskanych wyników dla analizowanych narzędzi.

3.2.11 Testy obciążeniowe

Waga: 3

Parametr opisuje czas potrzebny na przetworzenie różnych ilości równolegle przychodzących zapytań, symulując prace serwera przy obciążeniu od wielu użytkowników w tym samym czasie. Zapytania zostały wysłane w tym samym momencie i został zmierzony czas po którym uzyskano odpowiedź na wszystkie z nich. Zapytania zostały dobrane dla następujących przypadków użycia:

Autoryzacja - uzyskanie autoryzacji na zapytanie zawierające login oraz hasło użytkownika

Zapis danych użytkownika - zapisanie danych tekstowych użytkownika w bazie danych

Proste zapytanie o zasób - uzyskanie dostępu do zasobu tekstowego z bazy danych przez użytkownika

Zapytanie o zasób - uzyskanie dostępu do zasobu multimedialnego (pliku w formacie png) z serwera

Streaming multimediów - uzyskanie dostępu do streamu multimedialnego (w formacie avi) z serwera

Ilości wysłanych równolegle zapytań to:

mała - 5 zapytań

średnia - 30 zapytań

duża - 150 zapytań

Oceny zostały wystawione na podstawie porównania uzyskanych wyników dla analizowanych narzędzi.

Rozdział 4

Opis wybranych rozwiązań

4.1 Express

Rysunek 4.1: Logo projektu Express
Źródło: https://expressjs.com/

Express jest najbardziej popularnym frameworkiem technologii Node.js. Często jest używany również jako podstawa na której opierają się inne frameworki tej technologii. Został stworzony do tworzenia serwerów oferujących między innymi możliwości:

- obsługi routingu zapytań protokołu http
- integracji z silnikami prezentacji danych poprzez użycie szablonów
- tworzenia narzędzi sieciowych do zarządzania innymi serwerami
- tworzenie pośredniczących mikroserwisów

Sama podstawa frameworka jest utrzymywana w jak najbardziej minimalistycznej formie. Wspiera on polityke tworzenia modułów obsługujących daną funkcjonalność przez społeczeństwo deweloperów. Dzięki repozytorium Npm w łatwy sposób możemy znaleźć rozwiązanie gotowe na praktycznie każdy problem. Powstaje jednak kwestia wyboru właściwego, stworzonego w najbardziej optymalnie. Wiele z modółów zostaje zatwierdzonych przez grupe deweloperów tworzących framework jako sprawdzone, co zwiększa zaufanie oraz pomaga promować najlepsze rozwiązania. Express pozwala pisać aplikacje w bardzo elastyczny sposób, nie narzucając jednej słusznej drogi. Pozwala na tworzenie dowolnej architektury własnego serwera. Został stworzony w 2010 roku, wydany na licencji MIT przez firme StrongLoop. Do analizy została użyta wersja 4.16.3.

4.2 Sails

Rysunek 4.2: Logo projektu Sails
Źródło: https://sailsjs.com/

Sails jest to framework aplikacji webowych działający w architekturze MVC (Model-View-Controller). Inspiracją do jego powstanie był framework Ruby on Rails dla języka Ruby, a do jego napisania został wykorzystany Express. Pozwala on na budowe:

- serwisów rest API
- aplikacji single-page
- aplikacji czasu rzeczywistego

Sails prezentuje ściśle określoną architekturę projektu zdejmując z programistów obowiązek jej tworzenia. Jednym z podstawowych założeń tego narzędzia jest konfigurowalność dostarczonych, gotowych komponentów ponad pisanie własnych. Narzędzie wykorzystuje skonfigurowane pliki w języku JavaScript i generuje na ich podstawie gotowy produkt. Został stworzony w 2012 roku, wydany na licencji MIT przez firme The Sails Company. Do analizy została użyta wersja 1.0.2.

4.3 Meteor

Rysunek 4.3: Logo projektu Meteor
Źródło: https://www.meteor.com/

Meteor wyróżnia przede wszystkim, fakt że jest frameworkiem izomorficznym. Jeden kod aplikacji działa zarówno po stronie klienta jak i serwera. Służy do szybkiego tworzenia wielo-platformowych aplikacji (Android, iOS, web). Platforma odeszła od tradycyjnych narzędzi środowiska Node.js i stworzyła własny ekosystem budowania projektów czy menadzera modułów. Architektura komunikacji między warstwa frontendu, a backendu nie korzysta z zapytań http, a z subskrypcji modelu danych poprzez web sockety. Został stworzony z myślą o aplikacjach:

- single-page
- czasu rzeczywistego
- wymagających wysokiej synchronizacji danych

Framework wymusza na programiście korzystanie z paradygmatu reaktywnego. Skupia się on na reagowaniu na zmiany obserwowanego modelu danych. Został stworzony w 2012 roku, wydany na licencji MIT przez firme Meteor Development Group. Do analizy została użyta wersja 1.9.2.

Rozdział 5

Analiza porównawcza

5.1 Express

5.1.1 Popularność

Na portalu Github możemy zobaczyć, że framework posiada prawie 40000 gwiazdek. Na portalu Stackoverflow znajduję się prawie 45000 wątków dotyczących tej technologii. Obsługa narzędzia jest również najczęsciej poszukiwaną umiejętnością na rynku pracy w technologii Node.js. (dane na dzień 28.06.2018).

5.1.2 Rozmiar

Framework możemy zainstalować używając podstawowego menadzera zależności środowiska Node.js - Npm, używając komendy:

```
npm install express
```

Po instalacji zostanie utworzony folder node_modules zawierający tylko podstawowe komponenty frameworka, ważący 2.19 mb. Aby jednak w jak najbardziej wydajny sposób osiągnąć zadaną funkcjonalność należy zainstalować pare brakujących modułów. Po zainstalowaniu:

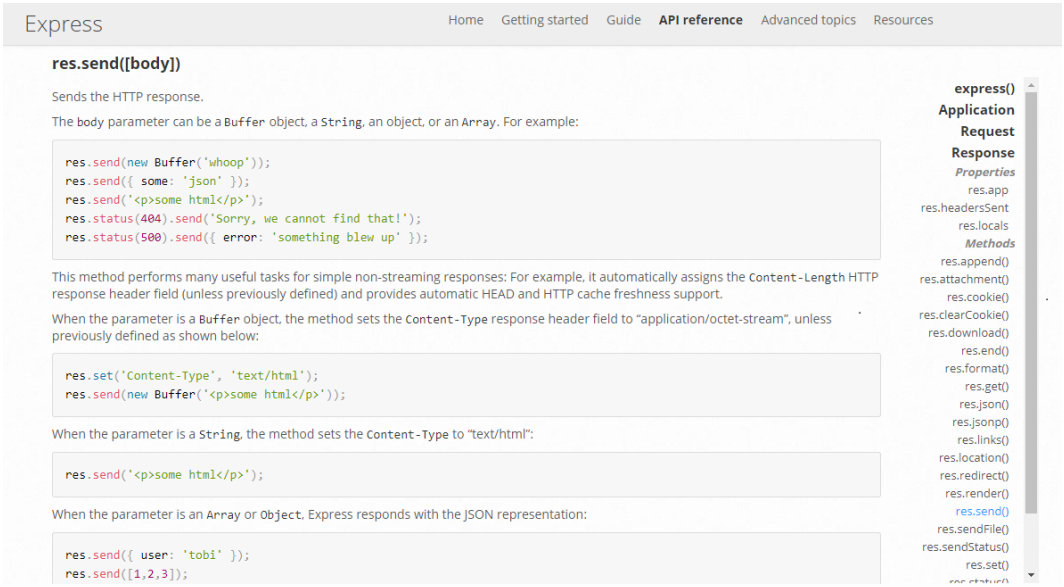
- Mongoose - obsługa bazy danych mongoDB
- Passport - autentykacjia użytkownika
- Bcrypt - przechowywanie haseł
- Helmet - bezpieczeństwo zapytań http

- Body-parser - obsługa ciał zapytań http
- Joi - walidacja zapytań http
- Express-session, Cookie-parser, Connect-flash - obsługa sesji oraz ciasteczek
- Mocha, Supertest, Chai oraz Cheerio - testy

Waga zwiększa się do 18.0 mb. Po stworzeniu plików źródełowych kompletna aplikacjia waży 18.1 mb. Można więc uznać to za wagę wyjściową, która powinna być wzięta przy analizie.

5.1.3 Dokumentacja

Strona dokumentacji jest dostępna na stronie https://expressjs.com/en/api.html. Jest utrzymana w bardzo prostym, minimalistycznym designie, co sprawia że łatwo się po niej poruszać oraz wyszukiwać informacje. Posiada opisy pól i metod z podstawowego modułu Express, oraz nieskomplikowane przykłady użycia. Dla niektórych zagadniej (na przykład template rednering) znajduje się tylko szczątkowa ilość informacji. Możemy znaleźć również dokumentacje pozostałych modułów utrzymywanych lub także polecanych przez zespół stojący za frameworkiem dla poszczególnych funkcjonalności. Na stronie brakuje wskazówek odnośnie polecanych modeli architektury projektu przydatnych dla początkujących użytkowników.



Rysunek 5.1: Dokumentacja projektu Express
Źródło: https://expressjs.com/en/api.html

5.1.4 Próg wejścia

Do stworzenia prostej aplikacji framework wydaje się wymagać tylko znajomości języka JavaScript. Wymagania rosną natomiast wraz z wielkością projektu. W celu uzyskania wielu z podstawowych funkcjonalności (poza routingiem oraz middlewares, które zapewnia Express) należy użyć zewnętrznych bibliotek, co wymaga wcześniejszego ich opanowania. Aby utrzymać kod w należytym ładzie należy również posiadać odpowiednie doświadczenie programistyczne. Wymagania te nie są jednak wygórowane i nie powinny być przeszkodą po krótkim zapozaniu się ze środowiskiem.

5.1.5 Warstwa widoku

W celu opracowania widoku aplikacji możemy wybrać między skorzystaniem z zewnętrznego rozwiazania (np. bibliotek frontendowych Angular, Vue, React) lub użyciem wspieranego przez Express renderingu po stronie serwera przy użyciu szablonów (ang. server side template rendering). Technika ta polega na stworzeniu odpowiednich reużywalnych wzorców widoków stron przy użyciu wybranego silnika (np. Pug lub Ejs) oraz zmiany renderowanych wartości wzorca zależnie od stanu aplikacji. Każdy z silników posiada specyficzną składnie, dzięki której zamienia otrzymane wartości na stronę html. Do opracowania rozwiązania zdecydowałem się na skorzystanie z metod renderowania po stronie serwera z użyciem silnika Ejs.

5.1.6 Warstwa bazy danych

Framework Express nie dostarcza żadnych integracji z bazą danych. W celu jej zapewnienia należy skorzystać z zewnętrznego narzędzia. Dla baz danych typu nosql (np. MongoDb) najpopularniejszym rozwiązaniem jest biblioteka mongoose. Daje ona wysokopoziomowy interfejs na odpowiednie metody wybranych baz danych oraz przekształca rekordy przy użyciu ODM (object document mapping). Dla baz danych typu sql (np. Postgres) najpopularniejszym roziązaniem jest biblioteka Sequelize. Pozwala ona na zaprzestanie używania języka sql na rzecz wysokopoziomowyh metod oraz dostęp do rekordów przy użyciu ORM (object relational mapping). Do oprawcowania aplikacji zdecydowałem się skorzystać z MongoDB oraz modułu Mongoose.

5.1.7 Ilość kodu

Ilość wytworzonych lini kodu to około 500. Najważniejszymi cechami frameworka, które przyśpieszają proces wytwórczy są użycie middlewares oraz routingu. Pierwszy z nich pozwala na ułożenie pewnego przepływu funkcji przy obsłudzę przychodzacych zapytań. Możemy zapewnić ciąg usług jakie zostaną wywołane w odpowiedniej kolejności (np. logowanie, autentykacjia). Drugi pozwala na wysokopoziomowe zarządzanie funkcjami obsługującymi przychodzące zapytania http. Dopasowywane są one do żadanego adresu. Te funkcjonalności pozwalają na skupieniu się na tworzeniu głównie warstwy biznesowej aplikacji bez potrzeby bezpośredniego zarządzania warstwą komunikacji, skutecznie zmiejszając wymagany nakład pracy.

5.1.8 Rozszerzalność

Jako że środowisko Express nie narzuca absolutnie żadnej sztywnej architektury wytwarzanego projektu rozszerzalność jaką uda nam się uzyskać zależy tylko od naszego własnego doświadczenia i umiejętności. Nie wątpliwie jest to problem dla nowych programistów oraz przy napotkaniu nowego projektu o nieznanej strukturze. Aby utrzymać kod czytelnym i klarownym wiele źródeł proponuje skorzystać przy budowie projektu ze wzorca MVC. Projekt podzielić wówczas na:

- model - reprezentacja stanu aplikacji
- widoki - szablony widoków
- kontrolery - obsługa routerów oraz ich logiki
- middlewares - obsługa middlewares (komponentów wspierających dane funkcjonalności)
- public - statyczne pliki jak multimedia czy style
- test - skrypty testujące
- helpers - funkcjonalność współdzielona między różnymi częściami projektu

Zapewni to odpowiednie rozgraniczenie między odpowiedzialnościami poszczególnych części dzięki czemu osiągniemy szczątkowe zapewnienie o łatwości rozszerzalności aplikacji.

5.1.9 Testowanie

Express nie dostarcza nam wraz z frameworkiem metod testujących wytwarzaną aplikacje. Najbardziej popularną biblioteką do testów tego narzędzia jest Mocha. Jest to biblioteka służąca do pisania każdego rodzaju przypadków testowych. Należy określić porządane efekty oraz sposób jak mają zostać uzyskane i zapakować je w test suits oraz test cases. Po użyciu narzędzia otrzymano wyniki:

```
Express app tests - functionality
  static resources
    get
      √ image (32ms)
      √ video (44ms)
  sign
    up
      post
        √ unauthenticated user can sign up (188ms)
        √ authenticated user can not sign up (96ms)
        body validation
          √ username is required (7ms)
          √ username must be not taken yet (76ms)
          √ password is required (4ms)
          √ password and password2 must match (4ms)
      get
        √ authenticated user can not get an sign up page (73ms)
        √ unauthenticated user can get an sign up page (37ms)
    in
      post
        √ authenticated user can not sign in (68ms)
        √ unauthenticated user can sign in (70ms)
        body validation
          √ username has to be valid (9ms)
          √ password has to be valid (58ms)
      get
        √ authenticated user can not get sign in page (76ms)
        √ unauthenticated user can get sign in page (14ms)
    out
      get
        √ authenticated user can sign out (65ms)
        √ unauthenticated user can not sign out (3ms)
  app
    √ authenticated user can get an app (69ms)
    √ unauthenticated user can not get an app (6ms)
    √ user is redirected from root to app (3ms)
  data
    post
      √ authenticated user can post data (1084ms)
      √ unauthenticated user can not post data (6ms)
      body validation
        √ text is required (74ms)
        √ text must be at least 3 chars long (66ms)
        √ text must not be longer then 100 chars (70ms)
    get
      √ authenticated user can get data (65ms)
      √ unauthenticated user can not get data (3ms)


28 passing (4s)
```

Rysunek 5.2: Testy frameworka Express przy użyciu biblioteki Mocha
Źródło: opracowanie własne

5.1.10 Testy sprawnościowe

Testy zostały przeprowadzone przy użyciu biblioteki Mocha, Supertest oraz Chai. Podane wyniki reprezentują ilość przetworzonych zapytań w określonym czasie dla wybranych funkcjonalności.

Okres czasu	Autoryzacja	Zapis danych tekstowych użytkownika	Zapytanie o zasób tekstowy użytkownika	Zapytanie o zasób obraz	Streaming multimediów video
2 sekundy	396	144	420	463	147
10 sekund	2326	1501	1692	2933	696
30 sekund	6126	3103	4019	9143	1857
Średnia ilość dla 1 sekundy	210	113	146	299	64

5.1.11 Testy obciążeniowe

Testy zostały przeprowadzone przy użyciu biblioteki Mocha, Supertest oraz Chai. Podane wyniki reprezentują czas otrzymania odpowiedzi dla odpowiednich ilości zapytań w sekundach dla wybranych funkcjonalności.

Ilości wysłanych równolegle zapytań	Autoryzacja	Zapis danych tekstowych użytkownika	Zapytanie o zasób tekstowy użytkownika	Zapytanie o zasób obraz	Streaming multimediów video
5 jednocześnie wysłanych zapytań	1.174	0.02	0.015	0.002	0.003
30 jednocześnie wysłanych zapytań	1.552	0.107	0.063	0.013	0.014
150 jednocześnie wysłanych zapytań	7.799	0.407	0.282	0.066	0.069
Średni czas dla 1 zapytania	0.056	0.02	0.02	0.0004	0.0004

5.2 Sails

5.2.1 Popularność

Na portalu Github możemy zobaczyć, że framework posiada prawie 20000 gwiazdek. Na portalu Stackoverflow znajduję się około 6000 wątków dotyczących tej technologii. (dane na dzień 28.06.2018).

5.2.2 Rozmiar

Do pobrania frameworka jako aplikacji konsolowej używamy polecenia:

```
npm install sails -g
```

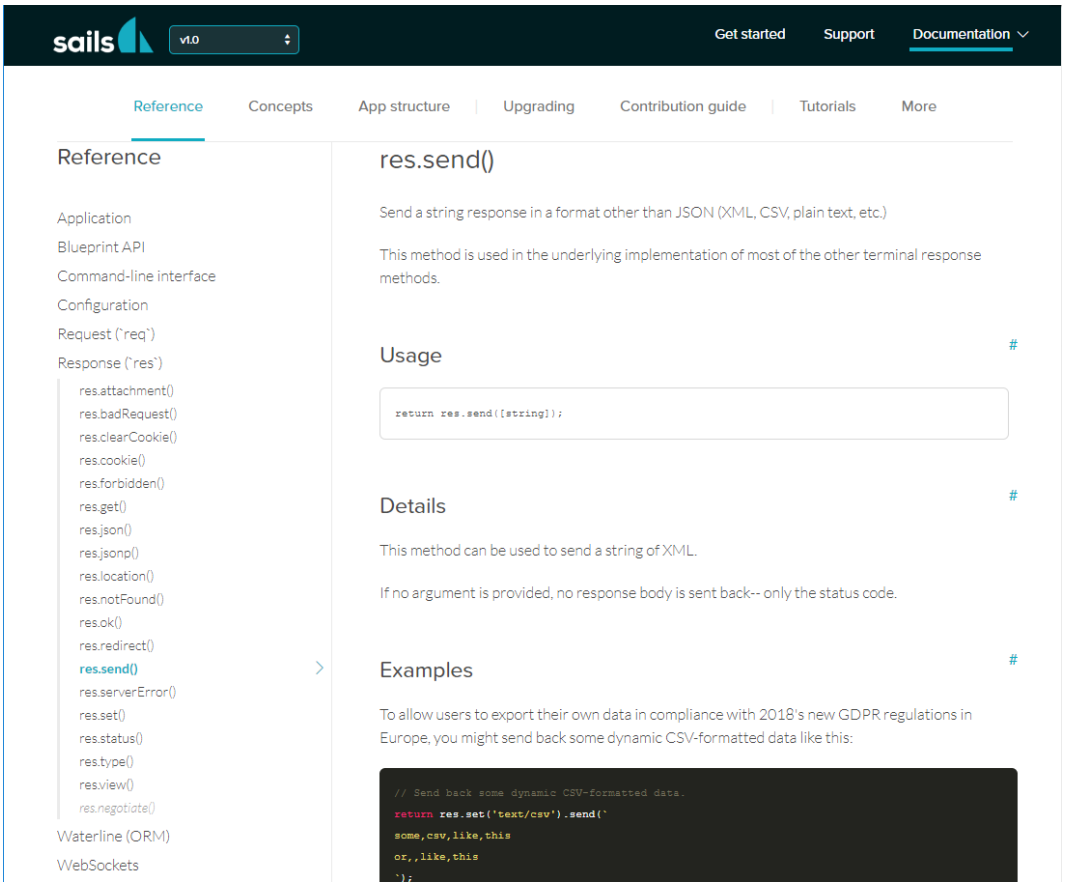
Po zainstalowaniu do stworzenia szkieletu projektu używamy:

```
sails new [nazwa aplikacji]
```

Stworzony projekt (wraz z pobranymi zależnościami) waży 176 mb. Po napisaniu kodów źródłowych waga jest równa 177 mb. Można więc uznać to za wagę wyjściową, która powinna być wzięta przy analizie.

5.2.3 Dokumentacja

Strona dokumentacji jest dostępna na stronie https://sailsjs.com/documentation. Możemy tam znaleźć zarówno ogólne, użyteczne przy gruntownym rozeznaniu oraz szczegółowe opisy poszczególnych części frameworka. W czytelny i łatwy do odszukania sposób mamy dostęp do wydawać by się mogło każdego szczegółu na temat tego środowiska. Dokumentacja jest napisana zrozumiałym ale nie upraszczającym językiem oraz posiada załączone zrzuty i odnośniki do przykładowych kodów źródłowych. Dostępne są opisy struktury architektury, wyjaśnienia konceptów oraz szczegółowe opisy użytecznych funkcjonalności.



Rysunek 5.3: Dokumentacja projektu Sails
Źródło: https://sailsjs.com/documentation

5.2.4 Próg wejścia

Sails jest wyjątkowo prosty w obsłudze. Do stworzenia podstawy projektu używamy jednego polecenia, a dalsza praca polega na pisaniu logiki na wzór już istniejącej. Ponadto dzięki przystępnie napisanej dokumentacji nie istnieją problemy nawet z rozwiązaniem bardziej skomplikowanych zagadnień. Wiele wymagań możemy pokryć już w procesie konfiguracji bez potrzeby pisania kodu. Dzięki sztywnej, z góry określonej strukturze aplikacji, mimo wielu możliwości łatwo jest zrozumieć sposób pracy zarówno przy tworzniu nowego jak i wdrażania w istniejący projekt.

5.2.5 Warstwa widoku

Jako że Sails został stworzony przy użyciu Express warstwa widoku rządzi się dokładnie tymi samymi zasadami. Możemy wybrać między skorzystaniem z zewnętrznego rozwiazania (np. bibliotek frontendowych Angular, Vue, React) lub użyciem renderingu po stronie serwera przy użyciu szablonów (ang. server side template rendering). Domyślnie używanym przez Sails silnikiem jest Ejs, który zdecydowałem się wykorzystać przy tworzeniu przykładowej aplikacji.

5.2.6 Warstwa bazy danych

Do obsługi warstwy bazy danych Sails wykorzystuje biblioteke Waterline. Jest to biblioteka ORM dostarczająca zunifikowane API do obsługi między innymi MySQL, PostgreSQL czy MongoDB. Posiada ona wiele użytecznych funkcjonalności takich jak automatyczne replikacje, czy nawet populowanie między bazami danych w różnych technologiach. W celu wybrania źródła danych nie musimy ustawiać dialektu, wystarczy tylko wskazać adres bazy. Domyślnie, na potrzeby dewelopmentu framework dostarcza baze danych Sails-disk, która symuluje działania realnej bazy danych. Nie powinna być ona używana jednak w środowisku produkcyjnym.

5.2.7 Ilość kodu

Ilość wytworzonych lini kodu to około 400. Są to jednak linie kodu bardziej skupione na konfiguracji wartości lub nieskomplikowanej logice w porównaniu z kodem aplikacji napisanej przy użyciu Express. Ponieważ framework został napisany przy użyciu narzędzia Express możemy korzystać ze wszystkich jego funkcjonalności min. middlewares. Sails zapewnia dodatkowo bardzo wiele możliwości. Najbardziej rozpoznawalną są blueprints, które pozwalają na automatyczne generowanie pełnej funkcjonalności rest api (wraz z interfejsem komunikacji oraz połączeniem z bazą danych) dla stworzonych modeli danych. Kolejną dostępną funkcjonalnością jest routing (pochodzący z Express) wzbogacony o automatyczne tworzenie końcówek http dla wytworzonych kontrolerów wedug ich ścieżki w projekcie. Framework oferuje także nieskomplikowane w obsłudzę zarządzanie dostępem do zasobów aplikacji przez policies, działających na podstawie określonych kontrolerów dostępu. Poza wymienionymi mamy również dostęp do mniej skomplikowanych funkcjonalności jak walidacja zapytań, automatyczne tworzenie dokumentacji czy internacjonalizacja aplikacji. Każda funkcjonalność jest konfigurowana na podstawie osobnych plików konfiguracyjnych w postaci obiektów języka JavaScript. Wszystko to sprawia, że możemy skupić się wyłącznie na pisaniu kodu biznesowego.

5.2.8 Rozszerzalność

Sails działa w sztywno określonej architekturze opartej na wzorcu MVC. Po stworzeniu szkieletu aplikacji jest ona podzielona na foldery:

- api, składające się z:
  - controllers - obsluga przychodzących zapytań, interakcja z modelem
  - helpers - współdzielona między komponentami funkcjonalność
  - policies - zarządzanie dostępem do zasobów aplikacji
  - models - struktura danych aplikacji
  - responses - reużywalne szablony dla odpowiedzi aplikacji
- assets - statyczne pliki serwisu jak multimedia czy style
- views - szablony widoków aplikacji
- tasks - funkcje oraz skrypty wywoływane dla różnych stanów aplikacji (np. Grunt lub Webpack)
- config - pliki konfigracyjne śronajprostrzych dowiska jak tłumaczenia, przełączniki, adresy baz danych

Wymuszenie sztywnego podziału gwarantuje szybkie odnalezienie się w nowych projektach oraz wskazuję droge do osiągnięcia zrozumiałego, czystego kodu. Dzięki konfigurowalności aplikacji jest ona również odporna na możliwe do szybkiego wprowadzenia zmiany. Projekty wytworzone narzędziem są dzięki temu łatwo utrzymywalne i rozszerzalne nawet przy braku dużego doświadczenia programistycznego.

5.2.9 Testowanie

Sails nie dostarcza nam wraz z frameworkiem metod testujących wytwarzaną aplikacje. Najbardziej popularną biblioteką do testów tego narzędzia jest Mocha. Jest to framework służący do pisania każdego rodzaju przypadków testowych. Należy określić porządane efekty oraz sposób jak mają zostać uzyskane i zapakować je w test suits oraz test cases. Po użyciu narzędzia otrzymano wyniki testów:

```
  Sails app tests - functionality
bringing sails up...
sails's up
    static resources
      get
        √ image (35ms)
        √ video (23ms)
    sign
      up
        post
          √ unauthenticated user can sign up (261ms)
          √ authenticated user can not sign up (254ms)
          body validation
            √ username is required (12ms)
            √ username must be not taken yet (221ms)
            √ password is required (9ms)
            √ password and password2 must match (10ms)
        get
          √ authenticated user can not get an sign up page (241ms)
          √ unauthenticated user can get an sign up page (40ms)
      in
        post
          √ authenticated user can not sign in (225ms)
          √ unauthenticated user can sign in (214ms)
          body validation
            √ username has to be valid (10ms)
            √ password has to be valid (218ms)
        get
          √ authenticated user can not get sign in page (234ms)
          √ unauthenticated user can get sign in page (19ms)
      out
        get
          √ authenticated user can sign out (227ms)
          √ unauthenticated user can not sign out (13ms)
    app
      √ authenticated user can get an app (231ms)
      √ unauthenticated user can not get an app (10ms)
    data
      post
        √ authenticated user can post data (224ms)
        √ unauthenticated user can not post data (11ms)
        body validation
          √ text is required (229ms)
          √ text must be at least 3 chars long (227ms)
          √ text must not be longer then 100 chars (217ms)
      get
        √ authenticated user can get data (223ms)
        √ unauthenticated user can not get data (12ms)
  27 passing (7s)
```

Rysunek 5.4: Testy frameworka Sails przy użyciu biblioteki Mocha
Źródło: opracowanie własne

5.2.10 Testy sprawnościowe

Testy zostały przeprowadzone przy użyciu biblioteki Mocha, Supertest oraz Chai. Podane wyniki reprezentują ilość przetworzonych zapytań w określonym czasie dla wybranych funkcjonalności.

Okres czasu	Autoryzacja	Zapis danych tekstowych użytkownika	Zapytanie o zasób tekstowy użytkownika	Zapytanie o zasób obraz	Streaming multimediów video
2 sekundy	210	210	243	280	127
10 sekund	1187	1036	1020	1499	627
30 sekund	2926	2336	2712	3992	1613
Średnia ilość dla 1 sekundy	103	85	95	137	56

5.2.11 Testy obciążeniowe

Testy zostały przeprowadzone przy użyciu biblioteki Mocha, Supertest oraz Chai. Podane wyniki reprezentują czas otrzymania odpowiedzi dla odpowiednich ilości zapytań w sekundach dla wybranych funkcjonalności.

Ilości wysłanych równolegle zapytań	Autoryzacja	Zapis danych tekstowych użytkownika	Zapytanie o zasób tekstowy użytkownika	Zapytanie o zasób obraz	Streaming multimediów video
5 jednocześnie wysłanych zapytań	1.015	0.037	0.025	0.009	0.011
30 jednocześnie wysłanych zapytań	6.271	0.17	0.215	0.07	0.079
150 jednocześnie wysłanych zapytań	31.261	0.796	1.021	0.258	0.254
Średni czas dla 1 zapytania	0.208	0.005	0.007	0.002	0.002

5.3 Meteor

5.3.1 Popularność

Na portalu Github możemy zobaczyć, że framework posiada prawie 40000 gwiazdek. Na portalu Stackoverflow znajduję się około 28000 wątków dotyczących tej technologii. (dane na dzień 28.06.2018).

5.3.2 Rozmiar

Do pobrania frameworka jako aplikacji konsolowej dla systemu windows używamy menadzera Chocolatey, używając polecenia

```
choco install meteor
```

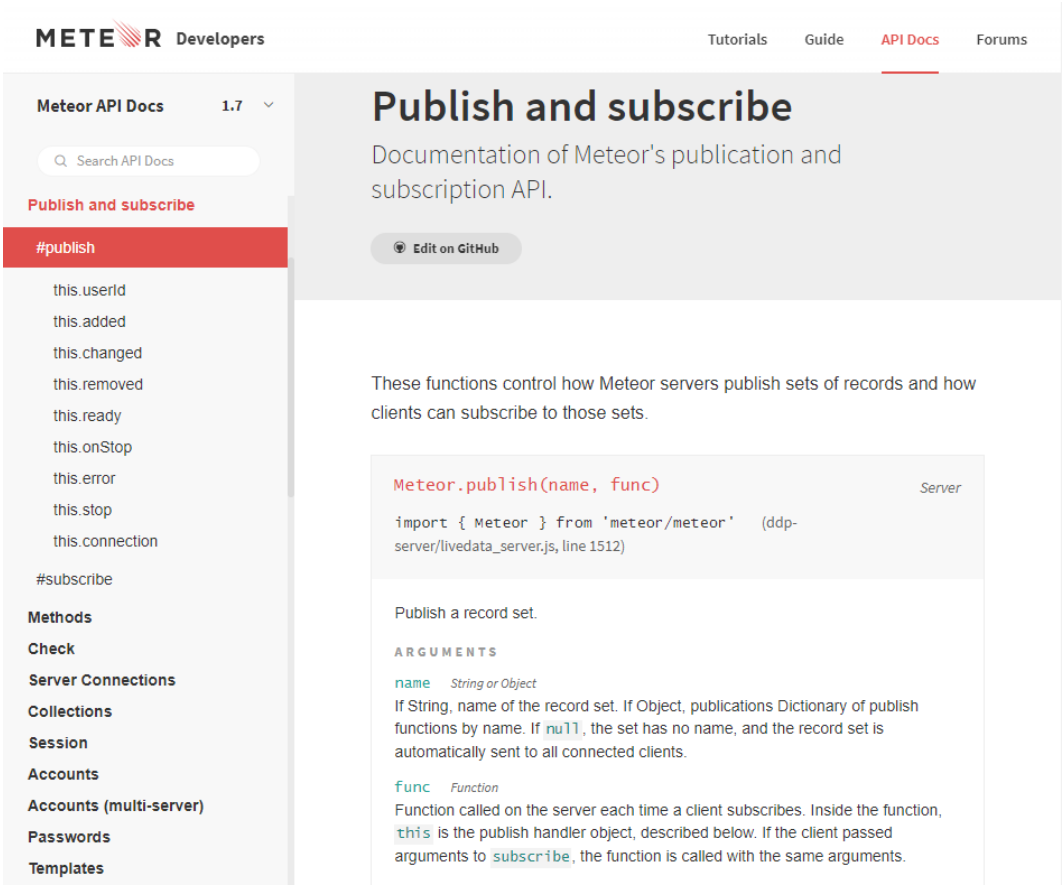
Narzędzie waży 389 mb. Po zainstalowaniu do stworzenia szkieletu projektu używamy:

```
meteor create [nazwa aplikacji]
```

Stworzony kod szkieletu aplikacji waży 7.39 mb. Do folderu projektu zostaje również dołączone środowisko uruchomieniowe oraz wspomagające biblioteki, a całość waży 454 mb. Można więc uznać to za wagę wyjściową, która powinna być wzięta przy analizie.

5.3.3 Dokumentacja

Strona dokumentacji jest dostępna na stronie https://docs.Meteor.com/#/full/. Zawiera ona obszerne opisy wszystkich funkcjonalności oraz wspieranych przez framework rozwiązań. Możemy znaleźć na niej materiały do nauki w postaci poradników jak krok po kroku zbudować kolejne części projektu, przykładowych repozytoriów z kodami źródłowymi oraz filmami wideo. Strona jest zarządzana w czytelny, podzielony na poszczególne częsci sposób. Znajdziemy tutaj również odnośniki do forum, gdzie poruszane są bardziej indywidualne/szczegółowe zagadnienia.



Rysunek 5.5: Dokumentacja projektu Meteor
Źródło: https://docs.meteor.com/#/full/

5.3.4 Próg wejścia

Meteor posiada wyższy próg wejścia niż pozostałe frameworki, ponieważ jest on całym ekosystemem wykorzystującym własne narzędzia. Posiada własny menadżer modułow Atmosphere, który w prawdzie posiada ogromną ilość komponentów wspierających wybrane funkcjonalności, ale nie są one tak popularne jak te z Npm. Jednak w przeciwieństwie do Npm aby umieścić moduł w Atmosphere musi on zostać zaakceptowany przez deweloperów odpowiedzialnych za biblioteke wprowadzając pewien stopień weryfikacji. Poprzez dostarczenie wielu gotowych rozwiązań Meteor posiada bardzo sztywny, specyficzny sposób wytwarzania projektu. Przy pierwszym kontakcie wiele konceptów jest niejasnych i odnaleźenie się w nich może sprawiać trudność. Do wytwarzania najprostszych projektów musimy

posiadać wiedzę o wielu elementach środowika, które nie są wykorzystywana w żadnym innym rozwiązaniu. Ponieważ Meteor jest znacząco inny od pozostałych frameworków nie potrzebujemy mięć dużej wiedzy o technologii Node.js - wystarczy nam JavaScript.

5.3.5 Warstwa widoku

Meteor po stronie widoku wspiera aplikacje napisane w React, Angular oraz silnik Blaze. Do działania na platformach mobilnych wykorzystuje silnik Cordova lub React Native. W repozytorium modułów atmosphere możemy znaleźć w pełni zaimplementowane koponenty służące do autoryzacji, wyświetlania filmów czy bibliotek zdjeć. Meteor kompiluje stworzone pliki html używając narzędzia Spacebars. Silnik Blaze zapewnia pełna synchronizacje (reaktywność) warstwy widoku oraz bazy danych (Minimongo). Obiekty reaktywne to obiekt sesji oraz kolekcje MongoDB. Do stworzenia analizowanego projektu wykrozystałem silnik Blaze.

5.3.6 Warstwa bazy danych

Framework Meteor aktualnie może działać tylko z bazami danych MongoDB. Po stronie serwera technologia używa sterownika MongoDB, a po stronie klienta używa biblioteki Minimongo. Minimongo jest to uproszczony klient bazy MongoDB, dzięki któremu nie musimy obawiać się o synchronizacje danych między warstwą frontendu oraz backendu. Framework automatycznie propaguje zmiany bazy danych w obydwu kierunkach poprzez wykorzystanie websocketów, oraz aktualizuje widok naszej aplikacji u klienta. Dzięki temu unikamy potrzeby tworzenia architektury komunikacji oraz uzykujemy aplikacje, która w każdym momencie może odzwierciedlać swój faktyczny, globalny stan.

5.3.7 Ilość kodu

Ilość wytworzonych lini kodu to około 250. Większość z nich dotyczy bezpośrednio widoku, nie logiki aplikacji do której obsługi zostały wykorzystane gotowe moduly z repozytorium Atmosphere. Najbardziej przydatnimi funkcjonalnościami oferowanymi przez biblioteke są:

- Publish/subscribe - pozwalające na reaktywne programowanie zdarzeń oraz reakcji na ich zajście.
- Blaze oraz Tracker - silnik pozwalający na zarządzanie reaktywne danymi

zarówno po stronie widoku oraz bazy danych bez potrzeby tworzenia reaktywnej logiki.

- Automatyczna obsługa oraz synchronizacjia bazy danych nawet w przypadku wielu użytkowników bez potrzeby tworzenia dodatkowej logiki.
- Użycie globalnej zmiennej Meteor w każdym miejscu w kodzie, która posiad dostęp do wszystkich części aplikacji.
- Wbudowana obsługa wielu strategi autoryzacji użytkownika.
- Framework wymaga wytworzenia wyjątkowo niewielkiej ilości kodu pozwalając na tworzenie projektów w mgnieniu oka.

5.3.8 Rozszerzalność

We frameworku Meteor wytwarzamy aplikacjie działające jednocześnie po stronie serwera oraz klienta. Programując możemy rozgraniczyć pewien kod poprzez trzymanie plików odpowiednio w folderach server oraz client. Pozostałe pliki będą dostępne po obu stronach, dzięki czemu możemy skorzystać z większej ilości reużywalnych komponentów, ale też musimy uważać aby nie udostępnić wrażliwych części aplikacji do klienta. Szkielet projektu przedstawia się następująco:

- lib - ustawienia, konfiguracje globalne
- public - udostępnione zasoby serwera jak zdjęcia, filmy, pliki stylów
- server - kod źródłowy obsługiwany po stronie wyłącznie serwera
- client - kod źródłowy obsługiwany po stronie wyłącznie klienta
- ui - widoki aplikacji
- startup - skrypty startowe

Metor swoją sztywną architekturą wymusza prowadzenie kodu podzielonego na jak najmniejsze, a przy odpowiednim utrzymaniu reużywalne części. Dzięki automatycznej synchronizacji oraz wykorzystaniu reaktywnych zmiennych możemy uniknąć problemów z synchronizacją danych w poszczególnych elementach aplikacji, nawet przy rozbudowanych projektach. Framework wspomaga wytwarzanie rozszerzalnych produktów, jednak przy nieumiejętnym zarządzaniu nie gwarantuje tego.

5.3.9 Testowanie

Metor nie dostarcza wraz z frameworkiem żadnego własnego rozwiązania testującego. Na stronie dokumentacji rekomendowaną biblioteką jest Mocha. Jest to framework służący do pisania każdego rodzaju przypadków testowych. Należy określić porządane efekty oraz sposób jak mają zostać uzyskane i zapakować je w test suits oraz test cases. Po użyciu narzędzia otrzymano wyniki testów:

```
[[[[[ Tests ]]]]]

=> Started proxy.
=> Started MongoDB.
I20180731-08:40:47.070(2)?
I20180731-08:40:47.164(2)? --------------------------------
I20180731-08:40:47.166(2)? --- RUNNING APP SERVER TESTS ---
I20180731-08:40:47.166(2)? --------------------------------
I20180731-08:40:47.166(2)?
I20180731-08:40:47.167(2)?
I20180731-08:40:47.167(2)?
I20180731-08:40:47.167(2)?   Express app tests - functionality
I20180731-08:40:47.167(2)?     static resources
I20180731-08:40:47.168(2)?       get
=> Started your app.

=> App running at: http://localhost:3012/
   Type Control-C twice to stop.

        √ image (109ms)2)?
        √ video:47.242(2)?
I20180731-08:40:47.242(2)?     sign
I20180731-08:40:47.243(2)?       up
I20180731-08:40:47.244(2)?         post
          √ user can sign up (100ms)
I20180731-08:40:47.344(2)?           body validation
            √ username is required
            √ username must be not taken yet (77ms)
            √ password is required
            √ password and password2 must match
I20180731-08:40:47.425(2)?         get
          √ get an sign up page
I20180731-08:40:47.446(2)?       in
I20180731-08:40:47.446(2)?         post
          √ user can sign in
I20180731-08:40:47.448(2)?           body validation
            √ username has to be valid
            √ password has to be valid
I20180731-08:40:47.453(2)?         get
          √ user can get sign in page
I20180731-08:40:47.470(2)?       out
I20180731-08:40:47.470(2)?         get
          √ user can sign out
I20180731-08:40:47.473(2)?     app
      √ user can get an app
I20180731-08:40:47.488(2)?     data
I20180731-08:40:47.488(2)?       post
        √ authenticated user can post data
I20180731-08:40:47.493(2)?         body validation
          √ text is required
          √ text must be at least 3 chars long
          √ text must not be longer then 100 chars
I20180731-08:40:47.495(2)?       get
        √ authenticated user can get data
I20180731-08:40:47.500(2)?
I20180731-08:40:47.500(2)?
I20180731-08:40:47.501(2)?   19 passing (430ms)
I20180731-08:40:47.501(2)?
I20180731-08:40:47.501(2)? Load the app in a browser to run client tests, or set the TEST_BROWSER_DRIVER environment variable.
```

Rysunek 5.6: Testy frameworka Meteor przy użyciu biblioteki Mocha
Źródło: opracowanie własne

5.3.10 Testy sprawnościowe

Testy zostały przeprowadzone przy użyciu biblioteki Mocha. Podane wyniki reprezentują ilość przetworzonych zapytań w określonym czasie dla wybranych funkcjonalności.

Okres czasu	Autoryzacja	Zapis danych tekstowych użytkownika	Zapytanie o zasób tekstowy użytkownika	Zapytanie o zasób obraz	Streaming multimediów video
2 sekundy	172	193	187	178	37
10 sekund	859	957	935	1027	243
30 sekund	2437	2901	2655	2765	800
Średnia ilość dla 1 sekundy	83	96	90	95	26

5.3.11 Testy obciążeniowe

Testy zostały przeprowadzone przy użyciu biblioteki Mocha. Podane wyniki reprezentują czas otrzymania odpowiedzi dla odpowiednich ilości zapytań w sekundach dla wybranych funkcjonalności.

Ilości wysłanych równolegle zapytań	Autoryzacja	Zapis danych tekstowych użytkownika	Zapytanie o zasób tekstowy użytkownika	Zapytanie o zasób obraz	Streaming multimediów video
5 jednocześnie wysłanych zapytań	0.885	0.098	0.071	0.064	0.249
30 jednocześnie wysłanych zapytań	4.372	0.621	0.412	0.299	1.42
150 jednocześnie wysłanych zapytań	19.006	3.598	2.073	0.739	4.923
Średni czas dla 1 zapytania	0.131	0.023	0.014	0.006	0.036

Rozdział 6

Podsumowanie

6.1 Podsumowanie ocen

6.1.1 Popularność

- Express jest najbardziej rozpoznawanym frameworkiem środowiska Node.js. Nie ma żadnych problemów ze znalezieniem materiałów do nauki lub pomocy wśród społeczności.
- Sails jest wciąż traktowany jako nowość aniżeli sprawdzone rozwiązanie (mimo komercyjnych zastosowań). Społeczność jest wciąż nie wielka.
- Meteor posiada nie tak dużą jak Express lecz oddaną rzesze fanów. Możemy obserwować duża aktywność na forach poświęconych temu rozwiązaniu. Społeczność stale rozszerza framework, mimo że często jest traktowany jako ciekawostka.

Oceny:

- Express - 3
- Sails - 1
- Meteor - 2

6.1.2 Rozmiar

- Express (18.1 mb) utrzymany w konwencji minimalnego frameworku, zawierającego tylko niezbędne moduły wypada najlepiej pod względem rozmiaru wytworzonej aplikacji.
- Sails (177 mb) znajduję się na drugim miejscu z nieco większym rozmiarem spowodowanym większą automatyzacją projektu.
- Meteor (454 mb) z powodu potrzeby posiadania środowiska frameworka znacznie odbiega od pozostałych rozwiązań pod kątem rozmiaru.

Oceny:

- Express - 3
- Sails - 2
- Meteor - 1

6.1.3 Dokumentacja

- Express w porównaniu do pozostałych rozwiązań posiada stronę dokumentacji, która wydaje się być uboga, opisująca tylko podstawowe zagadnienia.
- Sails posiada bardzo czytelna i bogata dokumentacje. Łatwo znaleźć w niej informacje dla początkujących oraz zaawansowanych użytkowników.
- Meteor również posiada bardzo przyzwoita dokumentacje, zawierającą dokładne wytłumaczenia konceptów, a nawet filmy instruktarzowe.

Oceny:

- Express - 2
- Sails - 3
- Meteor - 3

6.1.4 Próg wejścia

- Express cechuje się zależnym progiem wejścia od skomplikowania projektu. Dla prostych przypadków jest on niski, a dla skomplikowanych znacząco rośnie.
- Sails nawet dla dużych, skomplikowanych projektów nie wymaga znacznie większej wiedzy poza podstawowymi konceptami.
- Meteor posiada znacznie wyższy próg wejścia. Wprowadza bardzo wiele nowych ideii, zupełnie innych od tych znanych ze środowiska Node.js.

Oceny:

- Express - 2
- Sails - 3
- Meteor - 1

6.1.5 Warstwa widoku

- Express wspiera po stronie widoku aplikacje napisane w dowolnych technologiach frontendowych. Po stronie frameworku oferuję template rendering przy pomocy ogromnej ilości silników.
- Sails jako że został zbudowany na Express wspiera dokładnie te same rozwiązania.
- Meteor posiada dedykowane, bardzo proste w obsłudzę rozwiązanie Blaze, które w pełni implementuje reaktywne zmiany widoku. Wspiera również aplikacje frameworków React oraz Angular.

Oceny:

- Express - 2
- Sails - 2
- Meteor - 3

6.1.6 Warstwa bazy danych

- Express po stronie frameworka nie oferuje żadnego wsparcia bazy danych. Należy użyć zewnętrznych rozwiązań.
- Sails używa biblioteki Waterline dostarczając bardzo prostego, zunifikowanego interfejsu do obsługi bazy danych.
- Meteor bardzo silnie został zintegrowany z bazą danych MongoDB, dzięki czemu zapewnia automatyczną synchronizację dla globalnego stanu aplikacji.

Oceny:

- Express - 1
- Sails - 3
- Meteor - 3

6.1.7 Ilość kodu

- Express (około 500) posiada najmniejszą ilość funkcjonalności oraz wymaga największej ilości kodu.
- Sails (około 400) cechuje się bardziej konfiguracją dostępnych rozwiązań niż tworzeniem logiki.
- Meteor (około 250) znacznie odstaje od pozostałych rozwiązań. Możliwość użycia gotowych, skonfigurowanych modułów pozwala na stworznie aplikacji przy minimalnej ilości pracy.

Oceny:

- Express - 1
- Sails - 2
- Meteor - 3

6.1.8 Rozszerzalność

- Express pozwala na uzyskanie rozszerzalności zależnie tylko i wyłacznie od zastosowanych technik i umiejętności programistycznych użytkownika.
- Sails oferując sztywną architekture oraz możliwość łatwego wprowadzania zmian poprzez konfiguracje nie tworzenie logiki pozwala na łatwe uzyskanie skalowalnego projektu.
- Meteor wymusza prowadzenie kodu w określonej strukturze jednak nieumiejęte zarządzanie projektem może stworzyć problemy z rozszerzalnością.

Oceny:

- Express - 1
- Sails - 3
- Meteor - 2

6.1.9 Testowanie

- Express nie oferuje modułów przeznaczonych do testowania. Należy skorzystać z zewnętrznych rozwiązań.
- Sails nie wprowadza żadnych usprawnień w odniesieniu do Expressa odnośnie modułów testujących. Należy skorzystać z zewnętrznego rozwiązania.
- Meteor nie posiada zintegrowanych metod testujących kod. Dokumentacja wskazuję biblioteke mocha oraz prezętuje przypadki jej użycia.

Oceny:

- Express - 1
- Sails - 1
- Meteor - 1

6.1.10 Testy sprawnościowe

- Express (średnio 166 zapytań na sekunde) uzyskał znacznie lepsze wyniki od pozostałych frameworków w testach sprawnościowych. Ilość obsłużonyh zapytań była około 2 krotnie większa.
- Sails (średnio 95 zapytań na sekunde) osiągnął wyniki znacznie słabsze od Expressa, ale wciąż lepsze od Meteora.
- Meteor (średnio 78 zapytań na sekunde) uzyskał najmniej korzystne wyniki ze wszystkich testowanych rozwiązań.

Oceny:

- Express - 3
- Sails - 2
- Meteor - 1

6.1.11 Testy obciążeniowe

- Express (0.02 sek średni czas odpowiedzi na 1 zapytanie) wykazuję się również największą sprawnością jeśli chodzi o przetwarzanie wielu zapytań przychodzących równocześnie.
- Sails (0.04 sek średni czas odpowiedzi na 1 zapytanie) uzyskał średni czas odpowiedzi dwukrotnie dłuższy niż w przypadku Expressa.
- Meteor (0.04 sek średni czas odpowiedzi na 1 zapytanie) osiągnął takie same rezultaty jak Sails.

Oceny:

- Express - 3
- Sails - 2
- Meteor - 2

6.1.12 Uzyskane oceny

Parametr	Waga	Express	Sails	Meteor
Popularność	3	3	1	2
Rozmiar	1	3	2	1
Dokumentacja	3	2	3	3
Próg wejścia	1	2	3	1
Warstwa widoku	2	2	2	3
Warstwa bazy danych	2	1	3	3
Ilość kodu	2	1	2	3
Rozszerzalność	3	1	3	2
Testowanie	2	1	1	1
Testy sprawnościowe	3	3	2	1
Testy obciążeniowe	3	3	2	2

6.1.13 Końcowe oceny

Po przeprowadzonej analizie frameworków, wyniki zostały zebrane w poniższej tabeli, a końcowe oceny wyliczone ze wzoru: Podstawiając pod wzór:

$$O = \frac{\sum_{i=1}^{n} p_i * w_i}{sw}$$

gdzie:

O - ocena końcowa

n - ilość parametrów

p - ocena parametru

w - waga parametru

sw - suma wag parametrów

Oceny końcowe:

Framework	Express	Sails	Meteor
Ocena końcowa	2.04	2.16	2.08

6.2 Wskazywane zastosowania

Ponieważ końcowe oceny niezanczenie się od siebie różnią warto wskazać zastosowania, w których dany framework bedzie najbardziej odpowiedni.

6.2.1 Express

Express nadaje się idealnie to rozwiązań opracowywanych przez niewielkie zespoły deweloperskie, które nie rozrosną się do ogromnych projektów lub zostaną zaprojektowane jako wiele mikroserwisów z powodu braku ściśle określonych zasad architektury. Brak wyraźnie określonych standardów wpływa na możliwe problemy z rozwojem projektu. Oferuje bardzo szybkie przetwarzanie nawet przy dużym obciążeniu serwera, w związku z czym jest w stanie obsłużyć ogromną ilość jednoczesnych lub ciągle przychodzących połączeń.

6.2.2 Sails

Analiza wykazała że Sails uzyskał najlepszą ocene, otrzymując głównie pośrednie oceny poszczególnych parametrów - wydaje się być najbardziej uniwersalnym wyborem. Sprawdza się on dla wielu rozwiązań, ale jest naprawde ponadprzeciętny jeśli chodzi o projekty podatne na ciągłe zmiany i rozszerzenia, z powodu nacisku na konfigurowalność, a nie wytwarzanie kodu biznesowego. Doskonale sprawdza się przy aplikacjiach czasu rzeczywistego, streamingu oraz nawet serwerach gier wideo.

6.2.3 Meteor

Dostarczając wysoko zautomatyzowany proces deweloperski Meteor gwarantuje szybkie wytwarzanie prototypów czy MVP (ang. minimum viable product), które mogą być wysokoużyteczne w projektach startupowych, gdzie ponad niezawodne działanie ważne jest rozeznanie rynku i prezentacja produktu. Dodatkowe wsparcie platform mobilnych na podstawie tego samego kodu, chociaż nie zapewnia wysoce wydajnego działania pozwala sprawdzić zainteresowanie aplikacją bez potrzeby ponoszenia dodatkowych kosztów. Możemy oczywiście wytwarzać końcowe, docelowe produkty przy użyciu tej technologii, ale musimy być świadomi możliwego spadku jakości użytkowania oraz że zapewniona przez framework funkcjonalność (np. automatyczna synchronizacja baz danych) może okazać się wąskim gardłem kiedy aplikacja rozrośnie się do dużo większej skali.

Bibliografia

[1] E. Brown
"Web Development with Node and Express", 2014

[2] Azat Mardan
"Express.js Guide: The Comprehensive Book on Express.js", 2016

[3] StrongLoop
dokumentacja Express, 2017
Źródło: https://expressjs.com/en/4x/api.html

[4] Mike McNeil, Irl Natha
"Sails.js in Action", 2017

[5] Shahid Shaikh
"Sails.js Essentials", 2016

[6] Mike McNeil
dokumentacja Sails, 2017
Źródło: https://sailsjs.com/documentation/reference

[7] Isaac Strack
"Getting Started with Meteor JavaScript Framework", 2012

[8] Fabian Vogelsteller, Isaac Strack, Marcelo Reyna
"Meteor: Full-Stack Web Application Development", 2016

[9] Meteor Development Group
dokumentacja Meteor API, 2017
Źródło: https://docs.meteor.com/#/full/

[10] Node.js Foundation
dokumentacja języka programowania Node.js, 2017
Źródło: https://nodejs.org/en/docs/

[11] Eberhard Wolff
"Microservices: Flexible Software Architecture", 2016

[12] Bob Zeidman
"The Software IP Detective's Handbook: Measurement, Comparison, and Infringement Detection", 2011

[13] Guillermo Rauch
"7 Principles of Rich Web Applications", 2014

Spis rysunków