

Zastosowanie technologii React i Ruby on Rails w tworzeniu aplikacji webowych na przykładzie serwisu blogowego

(Application React and Ruby on Rails to create web applications by the example of blog site)

Oskar Sobczyk

Praca inżynierska

Promotor: dr Leszek Grocholski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

7 grudnia 2020

Oskar Sobczyk

.....

.....

(adres zameldowania)

.....

.....

(adres korespondencyjny)

PESEL:

e-mail:

Wydział Matematyki i Informatyki

stacjonarne studia I stopnia

kierunek: informatyka

nr albumu: 281822

Oświadczenie o autorskim wykonaniu pracy dyplomowej

Niniejszym oświadczam, że złożoną do oceny pracę zatytułowaną *Zastosowanie technologii React i Ruby on Rails w tworzeniu aplikacji webowych na przykładzie serwisu blogowego* wykonałem/am samodzielnie pod kierunkiem promotora, dr. Leszka Grocholskiego. Oświadczam, że powyższe dane są zgodne ze stanem faktycznym i znane mi są przepisy ustawy z dn. 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity: Dz. U. z 2006 r. nr 90, poz. 637, z późniejszymi zmianami) oraz że treść pracy dyplomowej przedstawionej do obrony, zawarta na przekazanym nośniku elektronicznym, jest identyczna z jej wersją drukowaną.

Wrocław, 7 grudnia 2020

(czytelny podpis)

Streszczenie

Celem pracy było opracowanie koncepcji oraz implementacja platformy blogowej łączącej funkcje popularnych serwisów. Do realizacji projektu użyto nowoczesnych rozwiązań. System składa się z dwóch części: aplikacji serwerowej napisanej w języku Ruby z wykorzystaniem Ruby on Rails oraz strony internetowej będącej interfejsem użytkownika zaimplementowanej za pomocą popularnego frameworka React.

The purpose of the study was to develop the blog platform that combines the functionality of popular services. Modern solutions were used to implement this project. The system comprises of two parts. Server application was written in Ruby using Ruby on Rails and website that is a user interface was implemented using popular framework React.

Spis treści

1. Wprowadzenie	7
2. Wymagania	9
3. Część dla programisty	11
3.1. Architektura aplikacji	11
3.2. Uruchomienie aplikacji	11
3.2.1. Backend	11
3.2.2. Frontend	12
3.3. Backend	12
3.3.1. Ruby on Rails	12
3.3.2. Struktura backendu	13
3.3.3. Baza danych	14
3.3.4. Redis	16
3.3.5. Uwierzytelnianie	16
3.3.6. Wyszukiwanie wpisów	17
3.3.7. Ciągła integracja	17
3.4. Frontend	18
3.4.1. React	18
3.4.2. Ant Design	18
3.4.3. Struktura projektu	19
4. Część dla użytkownika	23
4.1. Dostęp do systemu	23

4.2. Demonstracja aplikacji	23
4.2.1. Niezalogowany użytkownik	23
4.2.2. Zalogowany użytkownik	27
4.2.3. Obszar powiadomień	28
5. Podsumowanie	31
5.1. Dalszy rozwój	31
Bibliografia	33

Rozdział 1.

Wprowadzenie

Tematem pracy jest aplikacja internetowa implementująca wybrane funkcje platformy blogowej. Dzisiaj internet jest przepełniony treściami tworzonymi przez jego użytkowników np. poprzez wpisy na portalach społecznościowych lub artykuły na blogach. Jest to atrakcyjna przestrzeń do tworzenia nowych aplikacji pomagających dzielić się treściami. Mimo dużej konkurencji na rynku nadal istnieje zapotrzebowanie na nowe serwisy. Przykładem jest serwis Reddit [1] założony w 2005 roku. Dziś jest piątą (według Alexa) najczęściej odwiedzaną stroną w Stanach Zjednoczonych, miesięcznie przyciągającą 330 milionów użytkowników. Na początku jej główną funkcjonalnością była możliwość dzielenia się linkami do stron przez użytkowników oraz głosowania na te strony i ich komentowanie. Z czasem, wraz z rozwojem serwisu, użytkownikom dano możliwość dzielenia się nie tylko linkami do innych stron, ale też postami czy obrazkami. Dziś Reddit jest domem dla tysięcy społeczności. Innym przykładem jest powstałe w 2012 roku Medium [2]. Platforma ta skupia zarówno profesjonalnych dziennikarzy, jak i amatorów. Jej głównym celem jest dostarczenie użytkownikom wartościowych treści w przyjemnej formie. W odróżnieniu od Reddita wspomniana strona skupia się na udostępnianej treści, a nie na interakcjach pomiędzy użytkownikami.

Celem pracy jest stworzenie aplikacji webowej, która łączy możliwość tworzenia rozbudowanych postów zaczerpniętą z Medium z udostępnianiem użytkownikom miejsca do dyskusji na ich temat podobnie jak w Reddicie.

W wyborze zestawu narzędzi do realizacji implementacji pracy kierowałem się tym, aby zawierał on jak najwięcej nowych, wykorzystywanych na rynku komercyjnym rozwiązań, co pozwoliłoby mi poszerzyć moje umiejętności. W wyniku realizacji celu pracy powstała aplikacja składająca się z dwóch podstawowych części: **blogger-frontend**, **blogger-backend**.

Rozdział 2.

Wymagania

Pierwsze blogi powstały w późnych latach 90. Były prostymi, statycznymi stronami WWW aktualizowanymi ręcznie. Wpisy były wyświetlane użytkownikowi w odwrotnej chronologicznej kolejności. Wraz z upływem czasu i wzrostem popularności internetowych dzienników powstały specjalne serwisy oraz oprogramowanie służące do ich obsługi. Pozwoliło to na prowadzenie własnego bloga przez przeciętnego użytkownika internetu. Dzisiaj najpopularniejszą odmianą są mikroblogi – strony umożliwiające użytkownikom dzielenie się krótkimi, kilku zdaniowymi przemyśleniami lub obrazkami, np. Twitter [3], Facebook [4], Wykop.pl [5].

Poniżej opisano wymagania funkcjonalne i нефункционалне wytwarzanego systemu.

Wymagania funkcjonalne

- Niezalogowany użytkownik może:
 - zalogować się na istniejące konto;
 - utworzyć nowe konto;
 - przeglądać listę postów;
 - przeglądać post wraz z jego komentarzami;
 - przeglądać listę kategorii – posty w danej kategorii.
- Zalogowany użytkownik może:
 - wylogować się;
 - pisać nowe posty;
 - pisać komentarze;
 - przeglądać listę powiadomień.

Wymagania niefunkcjonalne

- prosty interfejs użytkownika,
- bezpieczny system kont dla użytkowników,
- rozszerzalność,
- wykorzystanie narzędzi do statycznej analizy kodu,
- zastosowanie narzędzi do ciągłej integracji podczas wytwarzania kodu,
- działanie projektu w systemie Linux.

Rozdział 3.

Część dla programisty

3.1. Architektura aplikacji

Aplikacja jest napisana w architekturze klient-serwer, podczas jej tworzenia wykorzystano wzorzec separacji zagadnień (ang. *separation of concerns*, SoC) [6]. Pozwoliło to na podzielenie serwisu na dwie części. Pierwszą jest warstwa dostępu do danych napisana w języku Ruby [7] z wykorzystaniem Ruby on Rails [8]. Odpowiada ona za przechowywanie danych w bazie danych PostgreSQL [9] oraz za ich udostępnianie za pomocą API. Drugą częścią jest warstwa prezentacji, która wykorzystuje JavaScript oraz framework React [10] w celu prezentacji danych. Taki podział pozwala na niezależny rozwój każdej części aplikacji.

3.2. Uruchomienie aplikacji

3.2.1. Backend

Wymagane oprogramowanie do uruchomienia części serwerowej:

- Ruby w wersji 2.6.2,
- Bundler [11] w wersji 1.17,
- Foreman [12] w wersji 0.85.0,
- Pozostałe zależności z pliku *Gemfile*,
- PostgreSQL w wersji 11.3,
- Node.js 10
- Redis [13] w wersji 5.0.5 działający w tle.

Oto czynności, które trzeba wykonać, aby zainstalować część serwerową:

1. W katalogu *blogger-backend* poleceniem *bundle install* zainstalować dodatkowe biblioteki.
2. Poleceniem *rake db:create* utworzyć bazę danych.
3. Wykonać migracje do bazy danych poleceniem *rake db:migrate*.
4. Wypełnić bazę danych przykładowymi danymi poleceniem *rake db:seed*.
5. Poleceniem *foreman start -f Procfile.dev* uruchomić serwer.

3.2.2. Frontend

Wymagane oprogramowanie do uruchomienia frontendu:

- Yarn [14] w wersji 1.13.0,
- Zależności znajdujące się w pliku *package.json*.

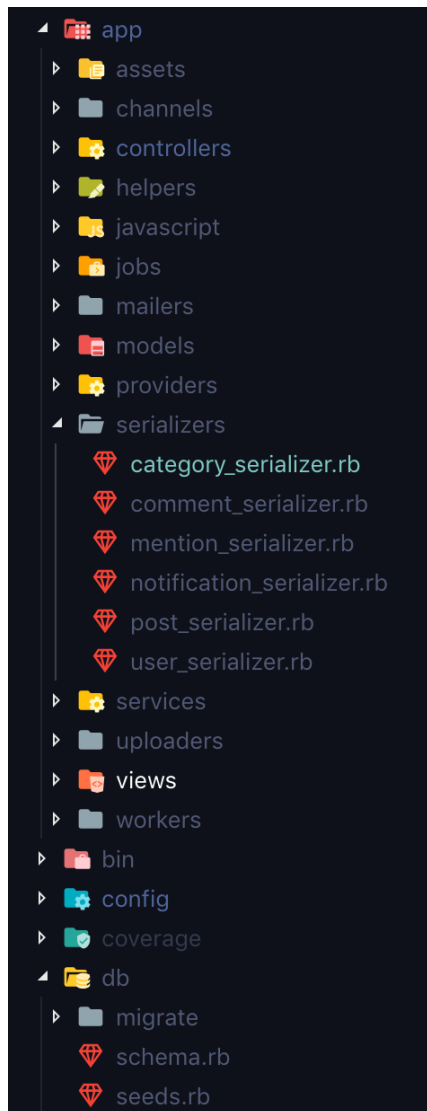
Oto czynności, które trzeba wykonać, aby uruchomić frontend:

1. W folderze *blogger-frontend* poleceniem *yarn* zainstalować brakujące pakiety.
2. Poleceniem *yarn start* uruchomić serwer deweloperski, jego domyślny adres *localhost:3001*.

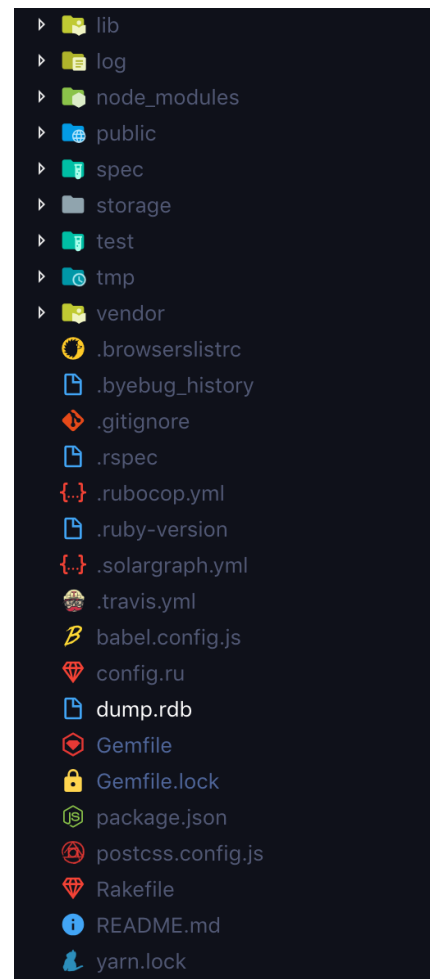
3.3. Backend

3.3.1. Ruby on Rails

Część serwerowa została napisana z wykorzystaniem frameworka Ruby on Rails. Powstał on w 2005 roku [15] jako narzędzie do szybkiego tworzenia aplikacji webowych. Głównymi strategiami projektowymi są „Convention Over Configuration” [16] oraz DRY („Don’t repeat yourself”) [17]. Dzięki temu programista już po zainicjowaniu projektu uzyskuje domyślną strukturę oraz konfigurację pozwalającą ominąć żmudny proces ręcznego definiowania ustawień. Już w momencie powstania Ruby on Rails oferował wiele innowacyjnych funkcji, takich jak generowanie tabel w bazie danych na podstawie modeli, scaffolding oraz system migracji. Dzisiaj napędza takie serwisy jak GitHub, Airbnb, Kickstarter, a inspirację nim możemy znaleźć w wielu innych frameworkach, np. Django [18], Sails.js [19].



Rysunek 3.1: Struktura części serwerowej, cz. 1



Rysunek 3.2: Struktura części serwerowej, cz. 2

3.3.2. Struktura backendu

Struktura części serwerowej przedstawiona na rysunkach 3.1 i 3.2 powstała w wyniku działania wbudowanego w Ruby on Rails generatora. Utworzył on minimalną konfigurację potrzebną do rozpoczęcia pracy. Framework implementuje architekturę **MVC (Model-Widok-Kontroler)** [20]. Zakłada ona podział na trzy główne części. Model jest odpowiedzialny za zarządzanie danymi. Widok ma za zadanie prezentację modelu za pomocą ustalonego formatu. Natomiast kontroler przyjmuje żądania od użytkownika i wykonuje operacje na modelach.

```
module Api
  class CommentsController < ApplicationController
    before_action :authenticate_user!, only: [:create]

    def create
      comment = Comment.new(comment_params)

      if comment.save
        render json: { "message": 'Comment successfully created', "comment": comment }
      else
        render status: 422, json: { "errors": comment.errors.full_messages.join('. ') }
      end
    end

    def show
      comments = Comment.find(params[:id])
      render json: comments
    end

    private

    def comment_params
      params.permit(:content, :post_id).merge(user_id: current_user.id)
    end
  end
end
```

Rysunek 3.3: Kod przykładowego kontrolera

Najważniejszą częścią projektu jest katalog *app* zawierający wszystkie komponenty aplikacji. W podkatalogu *app/controllers* umieszczono kontrolery odpowiadające za nadanie sensu żądaniom przychodzącym do API oraz za wygenerowanie poprawnej odpowiedzi (rys. 3.3). W nazwie każdego kontrolera zawarta jest informacja, na jakim modelu on operuje. W *app/services* znajdują się klasy serwisów, które implementują interakcje użytkownika z aplikacją. Pozwala to na wyodrębnienie logiki biznesowej z kontrolerów. Konfiguracje do serializera odpowiedzialnego za przekształcanie instancji modeli na format JSON znajdują się w folderze *app/serializers*. Definicje modeli (rys. 3.4) zlokalizowane są w folderze *app/models*. Katalog *db* zawiera aktualny schemat bazy danych, co umożliwia przenoszenie bazy pomiędzy komputerami oraz migracje, czyli zestaw plików napisany w Ruby DSL pozwala na zmianę schematu bazy danych bez potrzeby pisania ręcznie poleceń SQL.

3.3.3. Baza danych

Jako system przechowywania danych wybrano PostgreSQL. Jest to otwartoźródłowy silnik do zarządzania relacyjnymi bazami danych. Charakteryzuje się wysoką kompatybilnością ze standardem SQL:2011 [21], zgodnością z ACID [22] (ang. *Atomicity, Consistency, Isolation, Durability*) oraz niezawodnością. Ruby on Rails w domyślnej konfiguracji zapewnia wsparcie dla Postgresa. Pozwala to na zastosowa-

```
class Comment < ApplicationRecord
  after_create :create_mentions

  belongs_to :post
  belongs_to :user

  has_many :mentions, dependent: :destroy
  validates_presence_of :content

  def create_mentions(comment_id: id)
    MentionsWorker.perform_async(comment_id)
  end
end
```

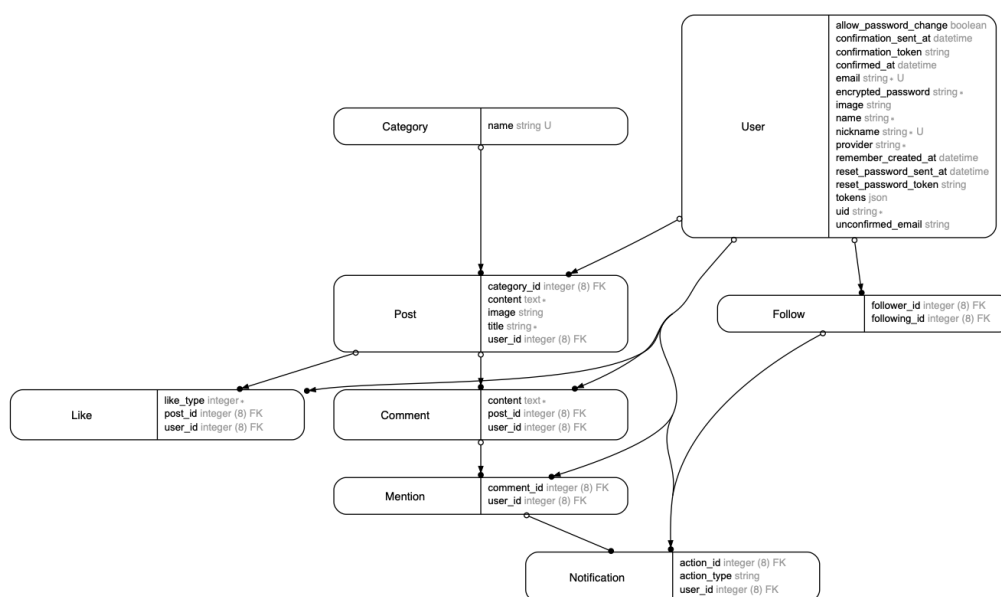
Rysunek 3.4: Kod przykładowego modelu

nie systemu migracji, który wraz z rozbudową aplikacji odpowiednio modyfikuje istniejący schemat bazy danych.

Schemat bazy danych

Baza danych (rys. 3.5) składa się z następujących tabel:

- **user** - zawiera między innymi takie informacje jak imię, e-mail, hasło. Jest wykorzystywana podczas uwierzytelniania.
- **post** - podstawowy typ danych w platformie blogowej. Składa się z tytułu, zawartości oraz łączy do miniaturki.
- **comment** - przechowuje zawartość komentarzy.
- **like** - odpowiada za przechowywanie głosów użytkowników.
- **follow** - przechowuje dane o obserwowanych osobach w postaci par obserwujący, obserwowany.
- **mention** - gromadzi dane z systemu wspomnień. Dla każdej wspomnianej osoby w komentarzu przechowuje wpis z identyfikatorem użytkownika oraz komentarza, co pozwala na szybkie znalezienie zarówno osób skojarzonych z wpisem, jak i postów przypisanych danej osobie.
- **notification** - odpowiada za powiadomienia. Dzięki polimorficznym asocjacjom możliwa jest rozbudowa tego systemu wraz z rozwojem aplikacji. Dla każdego wpisu w tabeli pamiętany jest zarówno identyfikator powiązanego obiektu, jak i jego typ. Pozwala to, aby jedna tabela zawierała odniesienia do wielu tabel.



Rysunek 3.5: Schemat ERD bazy danych

3.3.4. Redis

W projekcie wykorzystano bibliotekę Sidekiq [23] do przetwarzania zadań w tle. Do jej poprawnego działania wymagany jest Redis. Jest to nierelacyjna baza danych klucz-wartość przechowywana w pamięci głównej komputera. Dzięki temu charakteryzuje się wysoką wydajnością.

```

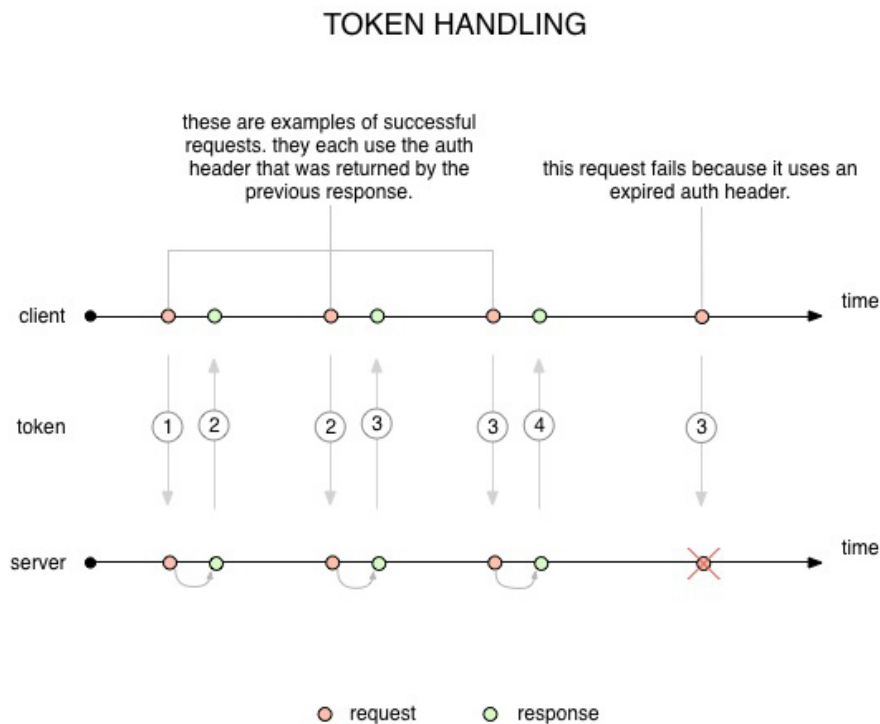
1  "access-token": "wwwwww",
2  "token-type":   "Bearer",
3  "client":       "xxxxxx",
4  "expiry":       "yyyyyy",
5  "uid":          "zzzzzz"
  
```

Rysunek 3.6: Struktura tokenu

3.3.5. Uwierzytelnianie

Aby ograniczyć dostęp do wybranych funkcjonalności API, wykorzystano system uwierzytelniania bazujący na tokenie. Użytkownik po pomyślnym zalogowaniu się otrzymuje w nagłówku odpowiedzi od serwera token (rys 3.6). Jego dołączenie

do nagłówków późniejszych zapytań kierowanych do chronionej części API będzie obligatoryjne. Pozwoli to na identyfikację użytkownika. W przypadku jego braku serwer zwróci w odpowiedzi błąd. W celu zwiększenia bezpieczeństwa serwisu, podczas każdego zapytania zostaje wygenerowany nowy token, natomiast stary zostaje unieważniony (rys. 3.7).



Rysunek 3.7: Proces obsługi tokenu [33]

3.3.6. Wyszukiwanie wpisów

Użytkownikom udostępniono wyszukiwarke pozwalającą przeszukiwać posty ze względu na ich tytuł. Zastosowano wbudowaną w silnik bazy danych metodę przeszukiwania tekstu *Full-text search* [24]. W odróżnieniu od operatorów *ILIKE* oraz *LIKE* które wykorzystują wyrażenia regularne, wyszukiwanie pełnotekstowe uwzględnia dodatkowe informacje np. językowe. Dzięki temu znalezione zostaną nie tylko wystąpienia wzorca, ale również wyrazy pochodne do szukanych.

3.3.7. Ciągła integracja

W procesie wytwarzania oprogramowania zastosowano technikę ciągłej integracji (z ang. *continuous integration*) [25] z wykorzystaniem serwisu Travis CI [26].

Każdy zestaw zmian wysyłany do repozytorium GIT wyzwał proces przeprowadzenia testów oraz statycznej analizy kodu. Taka praktyka przyspiesza wykrywanie błędów oraz wspomaga egzekwowanie wytycznych dotyczących stylu kodu.

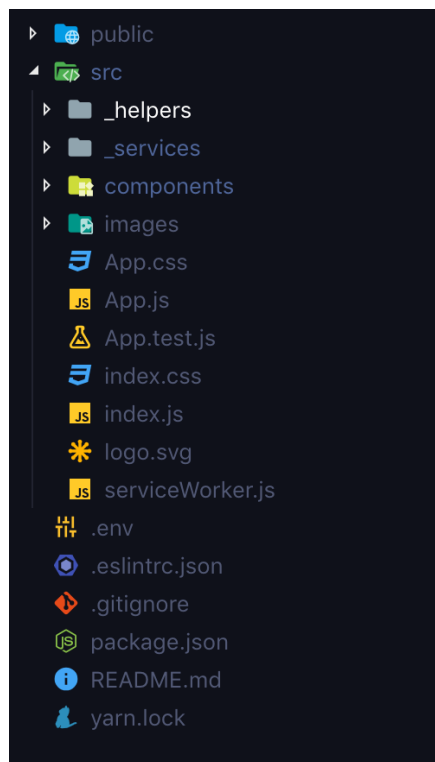
3.4. Frontend

3.4.1. React

Interfejs udostępniany użytkownikowi został napisany z wykorzystaniem Reacta. Jest to biblioteka JavaScript utworzona w 2013 roku [27] i utrzymywana przez Facebook. Dziś jest jednym z najpopularniejszych frameworków do tworzenia interfejsów użytkownika. Jego główną cechą charakterystyczną jest budowanie złożonych projektów za pomocą wielu mniejszych, wyizolowanych części nazywanych komponentami. Są to klasy lub funkcje, które mogą przyjmować jako wejście właściwości (*props*), a zwracają opis, jak dany komponent powinien zostać wyrenderowany przez przeglądarkę. Dodatkowo React pozwala na to, aby komponenty posiadały stan, czyli zestaw danych, które można zmieniać. Biblioteka ta umożliwia również dynamiczne modyfikowanie HTML-a bez potrzeby odświeżania stron. Istnieje możliwość wykorzystania Reacta na urządzeniach mobilnych za pomocą React Native [28].

3.4.2. Ant Design

Bardzo ważnym elementem aplikacji udostępnianej klientowi jest jej wygląd, dlatego aby całość aplikacji prezentowała się spójnie, zdecydowano się zastosować wytyczne Ant Design [29] oraz ich implementację za pomocą biblioteki *antd* zawierającej zestaw komponentów Reacta. Znacznie zmniejszyło to czas potrzebny na dodawanie stylów do nowych komponentów i pozwoliło skupić się na implementacji nowych funkcjonalności.



Rysunek 3.8: Struktura projektu

3.4.3. Struktura projektu

Projekt zainicjowano za pomocą narzędzia *create-react-app* [30] dostarczanego przez Facebook. Tworzy ono gotowe, skonfigurowane środowisko umożliwiające natychmiastowe rozpoczęcie pracy nad aplikacją. Kod aplikacji podzielono w następujących folderach (rys. 3.8):

- **_helpers** - zawiera pomocnicze funkcje nieprzypisane do konkretnego komponentu.
- **_services** - znajdują się tu funkcje odpowiedzialne za interakcję z API.
- **components** - jest to najważniejszy folder. To w nim umieszczone są wszystkie komponenty odpowiedzialne za wygląd i działanie aplikacji. Dodatkowo komponenty zostały podzielone ze względu na swoje zastosowanie.

Punktem wejścia do aplikacji jest *App.js* (rys. 3.9), gdzie został zdefiniowany komponent odpowiedzialny za routing. Wykorzystano do tego celu bibliotekę *react-router*[31] i komponenty *Route*, które zwracają wartość, tylko jeśli adres URL jest zgodny ze zdefiniowanym w *path*. W następnym kroku zostanie wyrenderowany konkretny komponent. React pozwala programiście na budowanie ich za pomocą funkcji lub klas. Od niedawna oba sposoby posiadają zbliżone funkcjonalności. Przykładowy komponent przedstawiony na rysunku 3.10 odpowiedzialny jest za wyświetlanie listy

postów. W pierwszych liniach deklarowany jest stan za pomocą funkcji *useState*. Następnie znajduje się funkcja *useEffect* [32], która jest odpowiednikiem metod cyklu życia z komponentów klasowych. W tym wypadku odpowiada ona za pobranie listy postów z serwera oraz zapisanie ich do stanu. Ostatnią czynnością komponentu jest zwrócenie obiektu, który ma zostać wyrenderowany.

```
class App extends React.Component {
  render() {
    return (
      <Router>
        <AuthProvider>
          <Layout className="layout" style={{ minHeight: "100vh" }}>
            <Header />

            <Content style={{ padding: "50px 50px" }}>
              <div style={{ background: "#fff", padding: 24, minHeight: 280 }}>
                { /* Routing definition */ }
                { /* Main page route */ }
                <Route path="/" exact component={PostListComponent} />
                { /* Sign in route */ }
                <Route path="/signin" component={SignIn} />
                { /* Login route */ }
                <Route path="/login" component={LogIn} />
                { /* Logout route */ }
                <PrivateRoute path="/logout" component={LogOut} />
                <Switch>
                  { /* Route to new post form */ }
                  <PrivateRoute path="/posts/new" component={PostForm} />
                  { /* Route to post edit form */ }
                  <PrivateRoute
                    path="/posts/:id/edit"
                    component={PostEditForm}
                  />
                  { /* Route to post show component */ }
                  <Route path="/posts/:id" component={PostShow} />
                </Switch>
                { /* Route to user edit form */ }
                <PrivateRoute path="/users/edit" component={UserEditForm} />
                { /* Route to posts from selected category */ }
                <Route path="/categories/:id" component={CategoryShow} />
                <Route path="/users/:id" component={UserShow} />
              </div>
            </Content>

            <Footer style={{ textAlign: "center" }}>
              Blogger Created by Oskar Sobczyk
            </Footer>
          </Layout>
        </AuthProvider>
      </Router>
    );
  }
}
```

Rysunek 3.9: Zawartość pliku App.js

```
function PostList(props) {
  const [posts, setPosts] = useState([]);
  const [isLoading, setIsLoading] = useState(false);
  const [errors, setErrors] = useState("");

  useEffect(() => {
    setIsLoading(true);
    postService
      .getAll()
      .then(response => {
        setIsLoading(false);
        setPosts(response.data);
      })
      .catch(error => {
        setIsLoading(false);

        if (error.response) {
          const error_messages = error.response.data.errors.full_messages;
          setErrors(error_messages);
        } else if (error.request) {
          setErrors("Something went wrong. Try again later.");
        }
      });
  }, []);

  return (
    errors ? (
      <div>
        <Alert message="Error" description={errors} type="error" showIcon />
      </div>
    ) : (
      <List itemLayout="vertical" size="large" loading={isLoading}>
        {posts.map(post => (
          <PostContainer post={post} key={post.id} />
        ))}
      </List>
    )
  );
}
```

Rysunek 3.10: Kod przykładowego komponentu

Rozdział 4.

Część dla użytkownika

4.1. Dostęp do systemu

Blogger jest platformą blogową pozwalającą użytkownikowi dodawać wpisy w jednej z predefiniowanych kategorii. Mogą oni również wchodzić w interakcję między sobą poprzez dyskusje pod wpisami lub system ich oceniania. Aby móc korzystać z tych funkcji, wystarczy utworzyć i zweryfikować konto za pomocą e-maila.

Działający system jest dostępny pod adresem `https://young-sea-15783.herokuapp.com/`. Aby móc korzystać ze wszystkich możliwości serwisu, należy założyć konto i potwierdzić je, klikając w link otrzymany drogą e-mailową. Dostępne jest również konto testowe – e-mail: `test@test.pl`, hasło: `test123`.

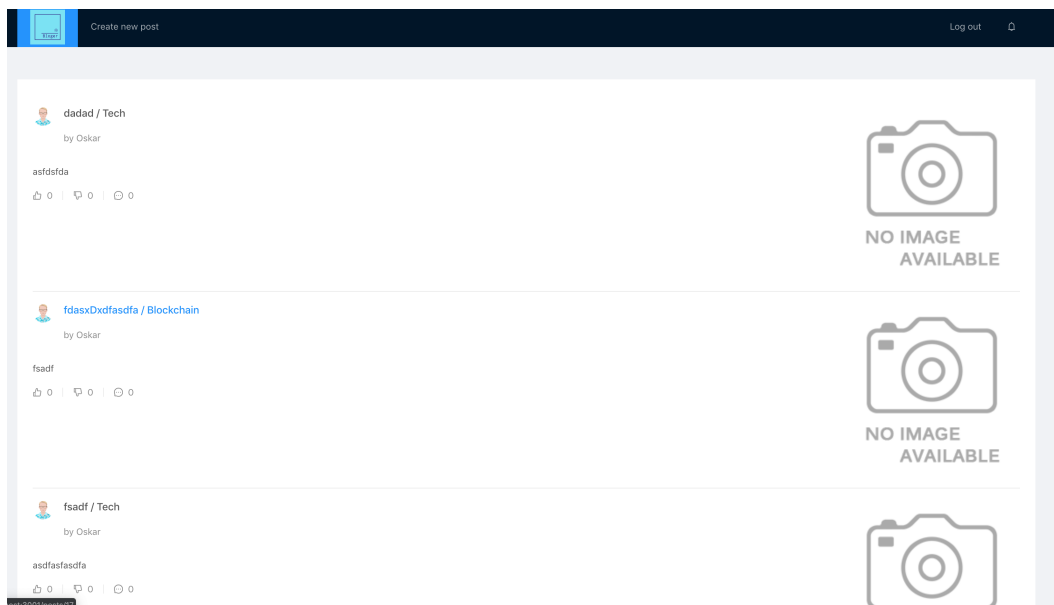
4.2. Demonstracja aplikacji

Serwis można podzielić na dwa moduły: jeden jest dostępny dla każdego użytkownika, drugi – tylko dla zalogowanego.

4.2.1. Niezalogowany użytkownik

Strona główna

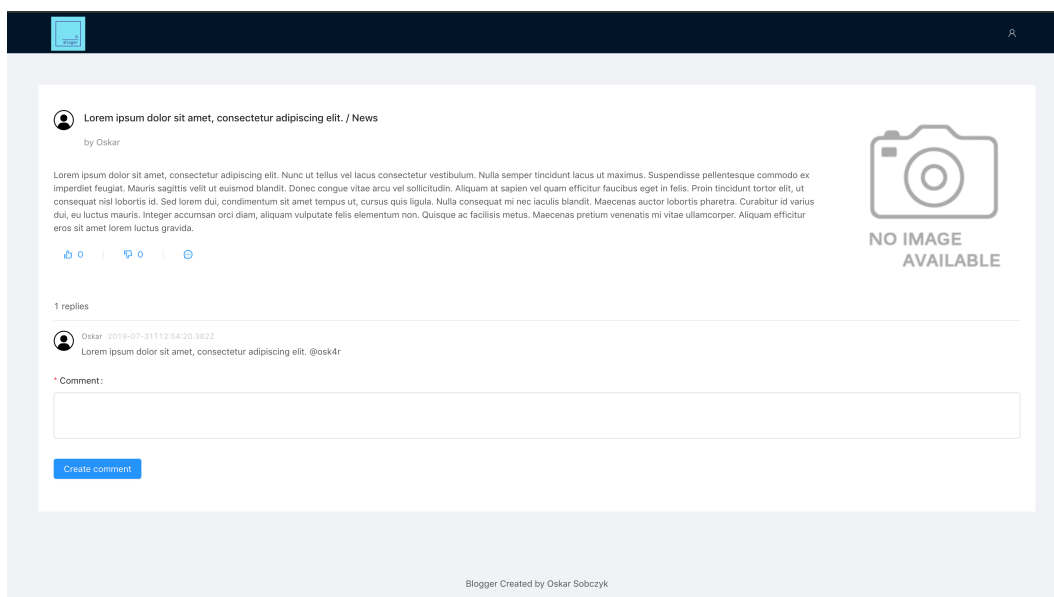
Użytkownik po wpisaniu adresu aplikacji w przeglądarce zostanie przekierowany na stronę główną (rys. 4.1), na której będzie mógł przeglądać listę wszystkich postów, jakie zostały utworzone. Każdy z nich jest reprezentowany przez przeglądowy opis, na który składają się: tytuł, skrócona treść wpisu oraz graficzna miniaturka. Użytkownik może zostać przeniesiony do szczegółowego widoku wpisu po kliknięciu w jeden z nich.



Rysunek 4.1: Ekran strony głównej

Szczegółowy widok wpisu

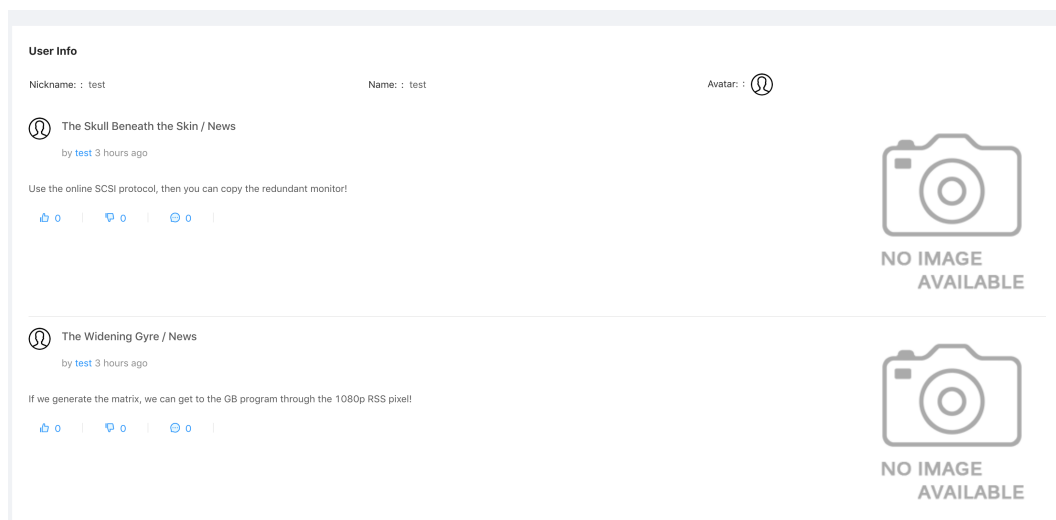
W tym widoku (rys. 4.2) użytkownik ma dostęp do pełnej treści wpisu oraz komentarzy.



Rysunek 4.2: Ekran widoku szczegółowego

Strona użytkownika

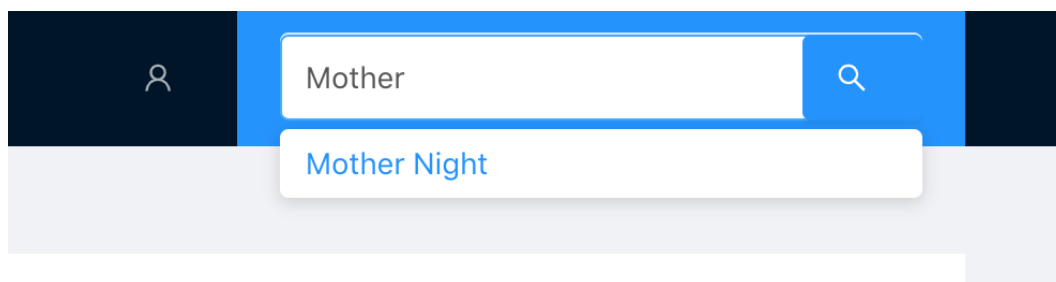
Użytkownik, klikając w pseudonim jednego z użytkowników, zostanie przekierowany na stronę użytkownika (rys. 4.3) zawierającą wszystkie jego wpisy.



Rysunek 4.3: Ekran strony użytkownika

Wyszukiwarka

Użytkownicy mają możliwość wyszukiwania wpisów ze względu na tytuł za pomocą wyszukiwarki (rys. 4.4).



Rysunek 4.4: Ekran wyszukiwania

Rejestracja nowego użytkownika

Aby zarejestrować się w serwisie (rys. 4.5), należy podać imię, nick, e-mail, hasło. Opcjonalnie użytkownik może dodać swój awatar. Po wysłaniu formularza na podany wcześniej adres poczty zostanie wysłany link aktywujący konto.

Sign in

* E-mail:


* Name:

* Nick:

* Password:

* Confirm Password:

Avatar:

 Select File

Register

Rysunek 4.5: Formularz rejestracji nowego użytkownika

Logowanie

Aby móc korzystać ze wszystkich możliwości serwisu, wymagane jest logowanie (rys. 4.6). Możliwe jest ono tylko po wcześniejszej aktywacji konta. W tym celu należy podać adres e-mail oraz hasło.

Log in

* E-mail:

* Password:

Login

Rysunek 4.6: Formularz logowania

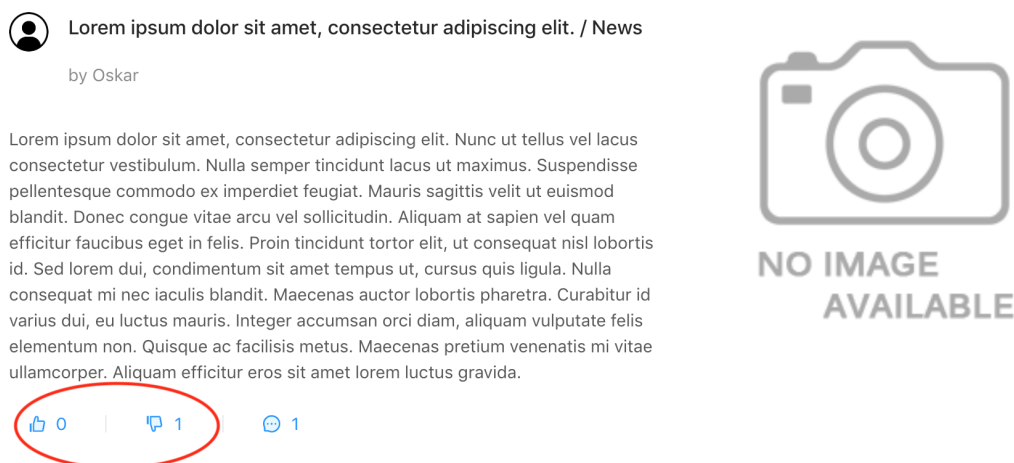
4.2.2. Zalogowany użytkownik

Dodawanie wpisów

Zalogowany użytkownik może dodawać wpisy do serwisu. Aby to zrobić, musi kliknąć przycisk znajdujący się w nagłówku strony *Create new post*, następnie zostanie przekierowany do odpowiedniego formularza. Tam do wypełnienia będzie miał takie pola jak: tytuł, treść, kategoria oraz miniaturka. Po wysłaniu i walidacji formularza wpis zostanie utworzony, a użytkownik przekierowany na jego stronę.

Ocenianie wpisów

Każdy zalogowany użytkownik ma możliwość oceniania innych użytkowników. Wystarczy, że kliknie w łapkę w dół lub w górę przy danym poście (rys. 4.7).



Rysunek 4.7: Ekran modułu oceniania

Komentarze

Dla zalogowanych użytkowników udostępniona została funkcja dyskusji na temat wpisów za pomocą komentarzy (rys. 4.8). Dodawanie nowych komentarzy jest możliwe z wykorzystaniem formularza dostępnego w szczegółowym widoku wpisu. Dodatkowo w system komentarzy wbudowany jest system „wołania”. Aby „zawołać” innego użytkownika w treści komentarza, należy jego nick poprzedzić symbolem @. W ten sposób wspomniany użytkownik otrzyma powiadomienie o naszej aktywności (rys. 4.10).

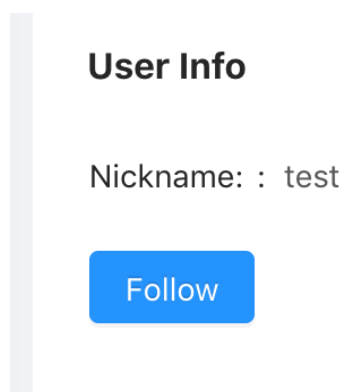
* Comment:

Create comment

Rysunek 4.8: Formularz komentarzy

Obserwowanie użytkowników

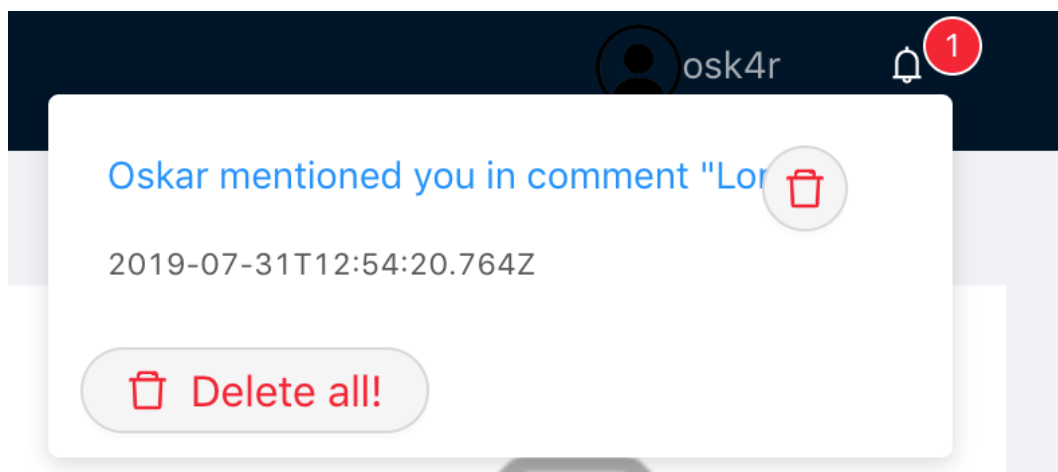
Każdy zarejestrowany użytkownik może otrzymywać powiadomienia o nowych wpisach wybranych użytkowników. Wystarczy, że doda ich do listy obserwowanych za pomocą przycisku *Follow* (rys. 4.9) dostępnego na stronie użytkownika.



Rysunek 4.9: Przycisk dodający do listy obserwowanych

4.2.3. Obszar powiadomień

W obszarze powiadomień (rys. 4.10) użytkownik ma dostęp do listy swoich powiadomień pochodzących z komentarzy lub nowych wpisów obserwowanych osób.



Rysunek 4.10: Ekran obszaru powiadomień

Rozdział 5.

Podsumowanie

Celem pracy było utworzenie platformy blogowej łączącej funkcje serwisów Reddit i Medium. Do implementacji użyto nowoczesnych narzędzi takich jak frameworki React i Ruby on Rails. Pozwoliło to stworzyć aplikację spełniającą nowoczesne standardy programistyczne oraz standardy z zakresu interfejsu użytkownika. System dzięki zastosowanym w nim technologiom i architekturze umożliwia dalszy rozwój.

5.1. Dalszy rozwój

Biorąc pod uwagę skalę popularności serwisów o podobnej funkcjonalności, można stwierdzić, że w wyniku dalszego rozwoju aplikacji będzie ona mogła zostać wprowadzona na rynek komercyjny. Aby zwiększyć częstotliwość korzystania z serwisu, przydatny może być system powiadomień. Jego rozbudowa o nowe typy notyfikacji sprawi, że odbiorcy częściej będą odwiedzać portal. W celu wizualnego uatrakcyjnienia serwisu przydatną funkcją będzie rozbudowa edytora wpisów o możliwość pogrubiania czcionki, stosowania kursywy czy zmiany wielkości czcionki. Należy wziąć pod uwagę, że nie każdy użytkownik jest zainteresowany wszystkimi treściami. Trzeba rozważyć wprowadzenie możliwości personalizacji strony głównej przez wybór kategorii, z jakich mają być wyświetlane wiadomości. Dzięki zastosowanej architekturze oddzielającej część serwerową od interfejsu użytkownika istnieje możliwość, aby w przyszłości powstała aplikacja mobilna wykorzystująca React Native oraz już istniejące API.

Bibliografia

- [1] Reddit <https://www.redditinc.com/>
- [2] Medium <https://medium.com/>
- [3] Twitter <https://twitter.com/>
- [4] Facebook <https://www.facebook.com>
- [5] Wykop.pl <https://www.wykop.pl>
- [6] Separation of concerns https://en.wikipedia.org/wiki/Separation_of_concerns
- [7] Ruby <https://www.ruby-lang.org/>
- [8] Ruby on Rails <https://rubyonrails.org/>
- [9] PostgreSQL <https://www.postgresql.org/>
- [10] React <https://reactjs.org/>
- [11] Bundler <https://bundler.io/>
- [12] Foreman <https://github.com/ddollar/foreman>
- [13] Redis <https://redis.io/>
- [14] Yarn <https://yarnpkg.com/lang/en/>
- [15] Ruby on Rails https://en.wikipedia.org/wiki/Ruby_on_Rails
- [16] Convention Over Configuration https://en.wikibooks.org/wiki/Ruby_on_Rails/Getting_Started/Convention_Over_Configuration
- [17] Don't repeat yourself https://en.wikipedia.org/wiki/Dont_repeat_yourself
- [18] Django <https://www.djangoproject.com/>
- [19] Sails.js <https://sailsjs.com>

- [20] Ruby on Rails: Getting Started with Rails https://guides.rubyonrails.org/getting_started.html
- [21] SQL:2011 <https://www.iso.org/standard/53681.html>
- [22] ACID <https://en.wikipedia.org/wiki/ACID>
- [23] Sidekiq <https://github.com/mperham/sidekiq>
- [24] PostgreSQL: Full Text Search <https://www.postgresql.org/docs/10/textsearch-intro.html>
- [25] Continuous integration https://en.wikipedia.org/wiki/Continuous_integration
- [26] Travis CI <https://travis-ci.org>
- [27] [https://en.wikipedia.org/wiki/React_\(web_framework\)](https://en.wikipedia.org/wiki/React_(web_framework))
- [28] React Native <https://facebook.github.io/react-native/>
- [29] Ant Design <http://ant.design>
- [30] Create React App <https://github.com/facebook/create-react-app>
- [31] React Router <https://reacttraining.com/react-router/web/guides/quick-start>
- [32] React: Introducing Hooks <https://reacttraining.com/react-router/web/guides/quick-start>
- [33] Devise token auth: Conceptual diagrams <https://devise-token-auth.gitbook.io/devise-token-auth/conceptual>