

WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ



ZAAWANSOWANE TECHNOLOGIE INTERNETOWE
Dokumentacja projektu

Aleksandra Poręba

2 lipca 2020

Spis treści

1	Wstęp	3
1.1	Wykorzystane technologie	3
2	Aplikacja serwera	4
2.1	Funkcjonalności	4
2.2	Budowa	6
3	Aplikacja klienta	7
3.1	Funkcjonalności	7
3.2	Budowa	7
4	Testowanie	10
4.1	Testowanie narzędziem <i>curl</i>	10
4.2	Testowanie narzędziem Advanced REST client	10
5	Wdrożenie	13
5.1	Serwer	13
5.2	Klient	14
5.3	Generowanie dokumentacji	15
6	Podręcznik użytkownika	16
6.1	Logowanie oraz tworzenie konta	16
6.2	Wyszukiwanie wydarzenia i zapis	19
6.3	Tworzenie nowych wydarzeń	20

1 Wstęp

Niniejszy dokument stanowi dokumentację projektu zrealizowanego w ramach przedmiotu *Zaawansowanie Technologie Internetowe*.

Projekt stanowi aplikacja *meetUP*, która umożliwia jej klientom zapisywanie się na prywatne wydarzenia. W internecie często brakuje prywatności, szczególnie na serwisach takich jak *Facebook*, gdzie wiadomość o wzięciu udziału w wydarzeniu jest publikowana obserwującym danej osoby. Dodatkowo, aby zaprosić kogoś na wydarzenie, należy najpierw dodać daną osobę do grona znajomych.

Jednocześnie internet jest miejscem, w którym łatwo i szybko można komunikować się z różnymi osobami i konsultować wspólne plany.

W aplikacji *meetUP* na wydarzenie można zaprosić wybrane osoby poprzez udostępnienie unikalnego kodu, który w razie upublicznienia osobom niechcianym można zmienić. Uczestnicy mają również możliwość dodawania komentarzy do danego wydarzenia.

Każdy zarejestrowany użytkownik ma możliwość nie tylko uczestnictwa, ale również zakładania własnych wydarzeń z wybranym kodem.

1.1 Wykorzystane technologie

Aplikacja serwera została przygotowana z wykorzystaniem technologii *Java API for RESTful Web Services* oraz interfejsu *JPA*.

Do stworzenia aplikacji klienta został wykorzystany framework *ReactJS* z wykorzystaniem dodatkowych bibliotek i pakietów, takich jak *redux*, *axios*, *moment.js* czy *Materialize*.

Została również przygotowana baza danych *PostgreSQL* umieszczona w środowisku chmurowym *ElephantSQL*. Zarówno serwer jak i aplikacja klienta zostali umieszczeni w ramach chmury *IBM Cloud*. Są dostępne pod poniższymi adresami:

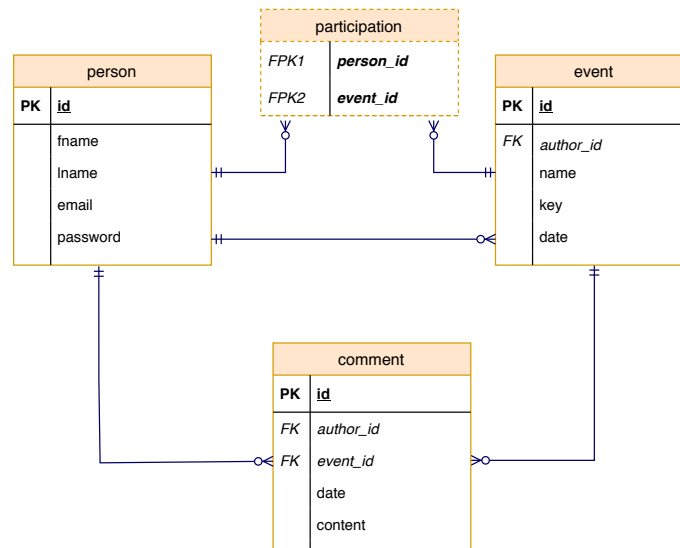
- serwer
`https://alporeba-zti-projekt.eu-gb.mybluemix.net/db/rest/`
- klient
`https://alporeba-zti-projekt-client.eu-gb.mybluemix.net/`

2 Aplikacja serwera

2.1 Funkcjonalności

W ramach projektu został stworzony serwer, którego zadaniem jest odpytanie bazy danych o odpowiednie informacje. Został on przygotowany z użyciem technologii *JPA* oraz *AspectJ* do logowania działania serwera. Do poprawnego działania serwisu wykorzystującego *AJAX* została również wdrożona technologia *CORS* (*Cross-Origin Resource Sharing*).

Aby przybliżyć funkcjonalności serwera konieczna jest znajomość organizacji bazy danych, przedstawiona poniżej.



Rysunek 1: Diagram ERD bazy danych.

Zgodnie z czterema tabelami, funkcjonalności serwisu *RESTful* zostały podzielone na cztery części:

- *person* - obsługujące działania związane z użytkownikami,
- *event* - obsługujące działania związane z wydarzeniami,

- *comment* - obsługujące komentarze do wydarzeń,
- *participation* - obsługujące wzięcie udziału przez użytkownika w wydarzeniu.

Do każdej z 4 części możliwe są operacje *CRUD* - *Create*, *Read*, *Update*, *Delete* dostępne poprzez odpowiednie metody *HTTP*. Odczytać można zarówno wszystkie rekordy, jak i pojedynczy, identyfikowany przez id. Poniżej znajduje się przykładowe zapytanie i odpowiedź:

```
https://alporeba-zti-projekt.eu-gb.mybluemix.net/db/rest/person
```

```
[[{"id":1,"name":"Captain","country":"America","email":"ca@marvel.com","password":"dummyspass"}, {"id":2,"name":"Iron","country":"Man","email":"im@marvel.com","password":"dummyspass"}]]
```

Zostały przygotowane również zapytania bardziej szczegółowe, na potrzeby aplikacji klienta, na przykład:

```
https://alporeba-zti-projekt.eu-gb.mybluemix.net/db/rest/comment/event/12
```

```
[[{"id":1,"text":"Ryby sa fajne","timestamp":"2020-06-29 05:53:27","event_id":5,"author_id":12,"country":"America","author_name":"Captain"}]]
```

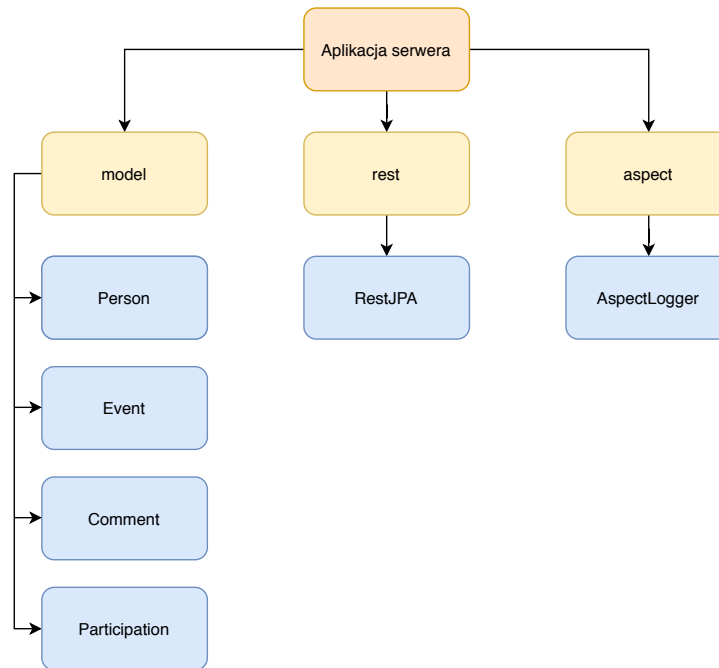
Wszystkie szczegółowe zapytania to:

- */person/email/{email}* - odczytanie danych osoby, identyfikując ją przez e-mail,
- */comment/event/{id}* - odczytanie wszystkich komentarzy dla danego wydarzenia,
- */event/key/{key}* - odczytanie wydarzenia identyfikowanego za pomocą klucza,
- */event/details/{id}* - odczytanie detali wydarzenia, np również danych organizatora,
- */event/author/{id}* - odczytanie wszystkich wydarzeń, których dana osoba jest organizatorem,
- */event/person/{id}* - odczytanie wszystkich wydarzeń, w których dana osoba bierze udział.

Wszystkie zapytania można testować za pomocą narzędzia *curl* dostępnego na maszynach wirtualnych.

2.2 Budowa

Na diagramie poniżej został przedstawiony diagram (organizacja) klas po części serwera.



Rysunek 2: Budowa aplikacji serwera. Kolorem żółtym zaznaczono trzy główne pakiety aplikacji. Dodatkowo pakiet model zawiera w sobie 4 pakiety, dla konkretnych tabel, zawierające klasy *POJO* oraz klucz złożony, w przypadku tabeli Participation.

Przygotowanie zostały 3 pakiety:

- *model* zawierający klasy *POJO* odpowiadające tabelom w bazie,
- *api* obsługujący wysyłanie zapytań,
- *aspect* zapewniające aspektowe logowanie działania serwera.

Poszczególne klasy oraz ich metody zostały szczegółowo w dokumentacji *JavaDoc*.

3 Aplikacja klienta

Do stworzenia aplikacji klienta został wykorzystany framework *React.js*. Udostępnione funkcjonalności, oraz budowa aplikacji zostały przedstawione poniżej.

3.1 Funkcjonalności

Aplikacja klienta udostępnia możliwość zapisywania się na prywatne wydarzenia, znajdujące według klucza. Osoby zapisane mogą dodawać i usuwać komentarze do danego spotkania, a także je opuścić.

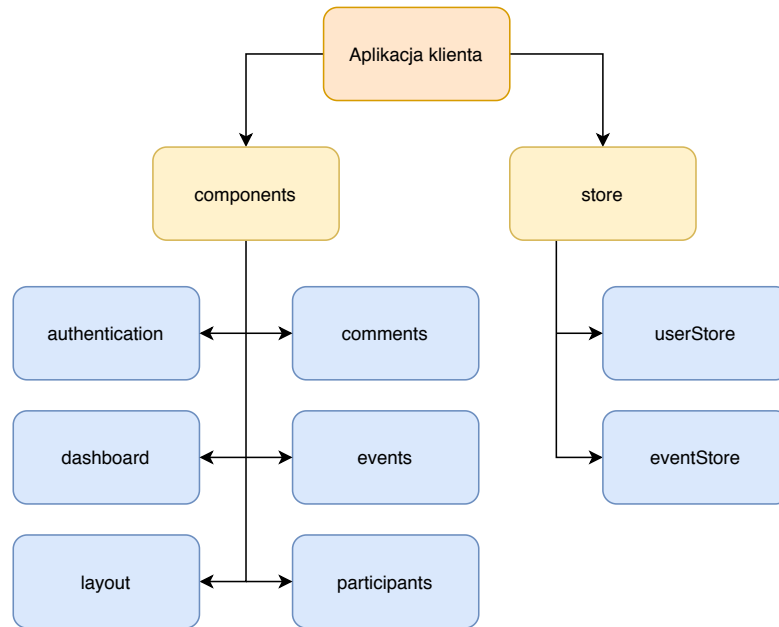
Użytkownik ma również możliwość tworzenia nowych wydarzeń, edytowania istniejących, których jest właścicielem, a także usuwanie ich.

Został przygotowany również serwis autoryzacji użytkownika - logowanie się, tworzenie konta, edycja danych.

Instrukcje poszczególnych operacji oraz ich interfejs zostały przedstawione w rozdziale 6.

3.2 Budowa

Struktura plików aplikacji klienta została podzielona na dwie części: **store** zawierający pliki potrzebne do zarządzania magazynem danych oraz **components** zawierający wszystkie komponenty. Schemat organizacji został przedstawiony poniżej.



Rysunek 3: Budowa aplikacji klienta.

Komponenty zostały podzielone na sześć kategorii:

- *authentication* - obsługa uwierzytelnienia - logowanie, zakładanie konta, edycja danych,
- *comments* - dodawanie, usuwanie oraz wyświetlanie komentarzy,
- *dashboard* - podstawowe komponenty strony głównej użytkownika oraz managera spotkań,
- *events* - działanie związane z wydarzeniami, podzielone na dwie części - część do strony głównej - wyświetlanie listy, szczegółów, oraz część do managera spotkań, z możliwością dodawania wydarzeń, edycji, usuwania,
- *layout* - komponenty związane z wyglądem aplikacji,
- *participants* - wyświetlanie listy uczestników dla danego wydarzenia.

Do obsługi danych aplikacji została wykorzystana biblioteka *Redux*. Zostały stworzone dwa magazyny, zgodnie z podziałem funkcjonalności aplikacji klienta:

- jeden do obsługi zdarzeń: dodawania ich, usuwania, komentarzy, zapisu na wydarzenia,
- oraz do obsługi użytkownika - logowania, tworzenia i edycji konta, pobierania danych użytkownika.

W akcjach wysyłane są zapytania *HTTP* do serwera za pomocą pakietu *axios*. Otrzymane odpowiedzi są parsowane i zapisywane do odpowiednich magazynów. Stamtąd odczytywane są przez komponenty, które uaktualniają widoki.

Kod źródłowy aplikacji klienta został dokumentowany za pomocą adnotacji *JSDoc*.

4 Testowanie

Działanie serwera zostało szczegółowo przetestowane za pomocą testów jednostkowych. Do ich wykonania zostały użyte takie narzędzia jak *curl* oraz *Advanced REST client*. Przykładowe testy zostały przedstawione poniżej.

Dodatkowo przeprowadzane były ręczne testy wydajnościowe i funkcjonalne dla aplikacji klienta, jak i serwera. Finalna aplikacja spełniała wszystkie założone wymagania.

4.1 Testowanie narzędziem *curl*

Na listingu poniżej znajduje się przykładowy zestaw testów do zapytań dotyczących komentarzy, wraz z uzyskanymi wynikami.

```
curl -H "Content-type: application/json" -d '{"password": "dummpass", "fname": "Test", ↵
      "lname": "Test", "email": "test@test.com"}' -X POST ↵
http://localhost:8088/meetUP-server/db/rest/person
> 11

curl -X GET http://localhost:8088/meetUP-server/db/rest/person/11
> {"id":11,"fname":"Test","lname":"Test","email":"test@test.com","password":"dummpass"}

curl -H "Content-type: application/json" -d '{"id": 11, "password": "dummpass", "fname": ↵
      "Test1", "lname": "Zmiana", "email": "test@test.com"}' -X PUT ↵
http://localhost:8088/meetUP-server/db/rest/person
> 11

curl -X GET http://localhost:8088/meetUP-server/db/rest/person/11
> {"id":11,"fname":"Test1","lname":"Zmiana","email":"test@test.com","password":"dummpass"}

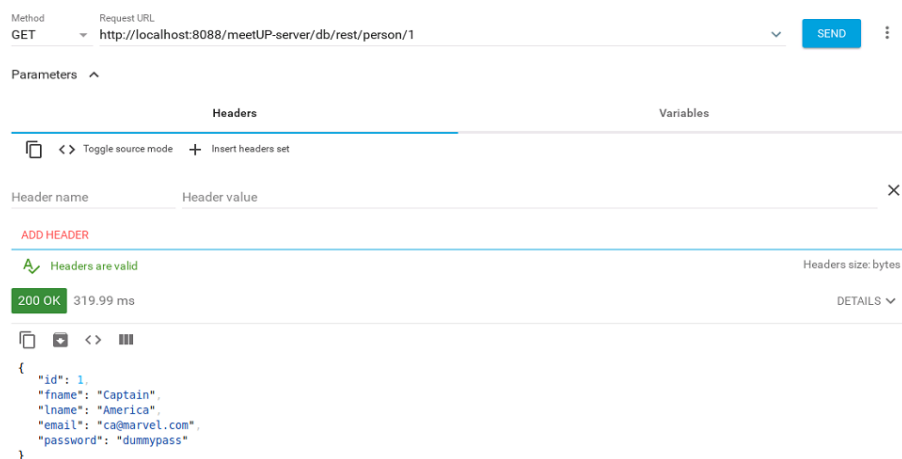
curl -X DELETE http://localhost:8088/meetUP-server/db/rest/person/11
> 1

curl -X GET http://localhost:8088/meetUP-server/db/rest/person/11
>
```

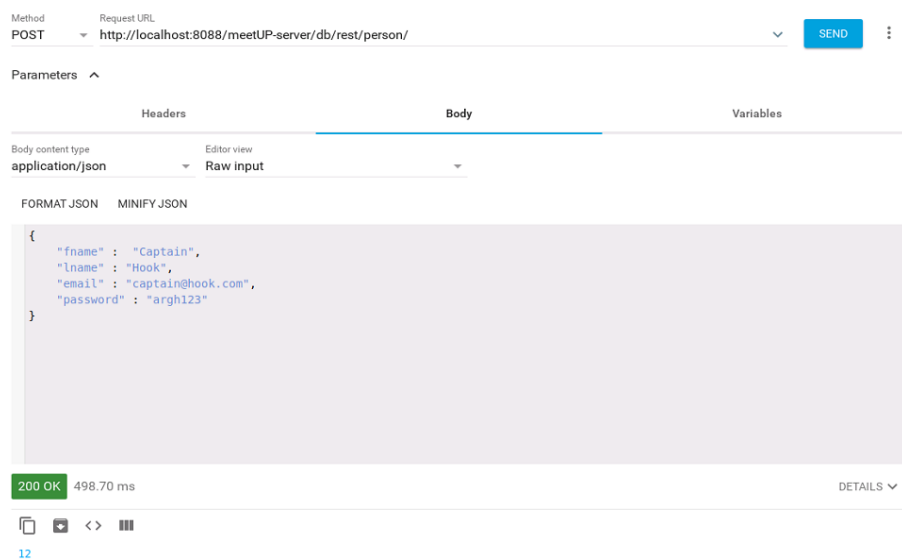
Wyniki testów porównywane były z referencją. Dzięki nim można było sprawdzić poprawność działania serwisu *REST*, czy wszystkie operacje poprawnie działają i są wspierane. Testy dotyczące konkretnych niedozwolonych wartości były wykonywane za pomocą *Advanced REST client*.

4.2 Testowanie narzędziem *Advanced REST client*

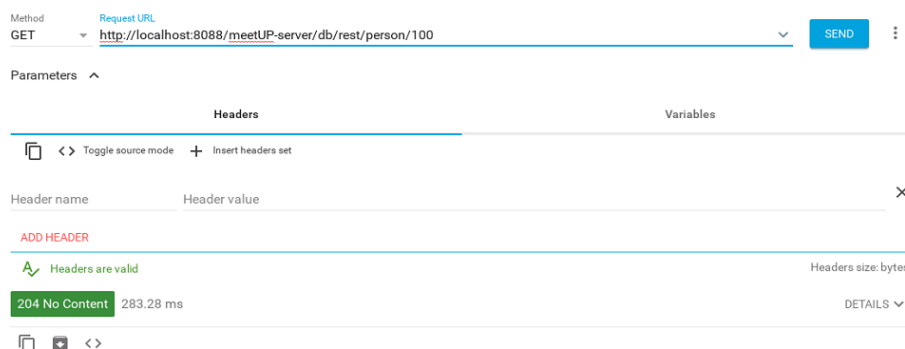
Poniżej przedstawione zostało kilka wykonanych testów za pomocą narzędzia *Advanced REST client* na przeglądarce *Chromium*.



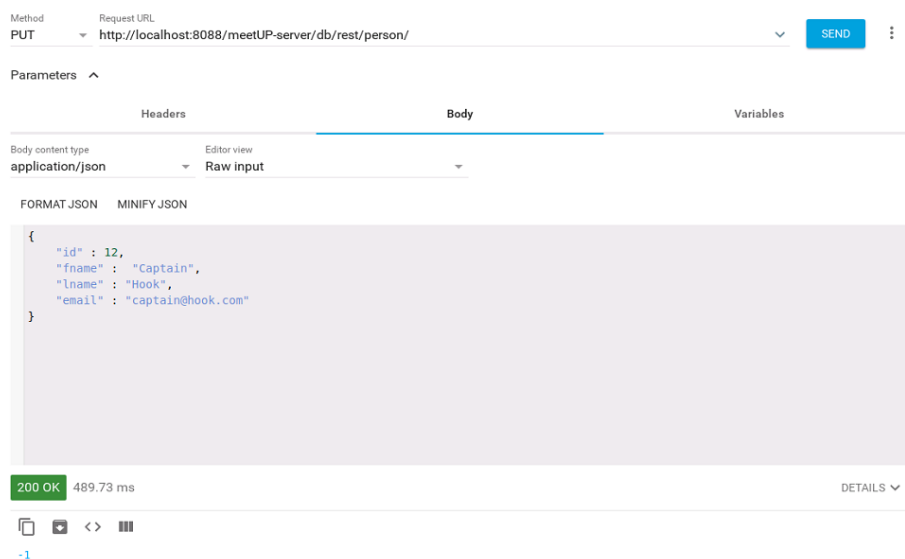
Rysunek 4: Odpowiedź zapytania GET do tabeli Person. Zwrócone zostały zgodnie z oczekiwaniami dane osoby z id = 1.



Rysunek 5: Odpowiedź zapytania POST do tabeli Person. Dodano nowego użytkownika z wykorzystaniem wbudowanego edytora treści zapytania. Zwrócone zostało id nowej osoby.



Rysunek 6: Odpowiedź zapytania GET do tabeli Person. Zwrócony został kod 204 - nie ma użytkownika z $id = 100$, wobec tego odpowiedź jest pusta.



Rysunek 7: Odpowiedź zapytania PUT do tabeli Person. Przesłane zapytanie było niekompletne, został więc zwrócony kod -1, świadczący o niepowodzeniu transakcji.

5 Wdrożenie

5.1 Serwer

Przed przystąpieniem do wdrażania należy uzupełnić plik `WebContent/META-INF/context.xml` o dane potrzebne do połączenia z bazą danych. Konieczne jest również pobranie bibliotek `postgresql-*.jar` oraz `jstl-*.jar` i umieszczenie ich w katalogu `WebContent\lib`.

Aplikację serwera można uruchomić na lokalnym serwerze w środowisku *Eclipse*. Należy wykonać następujące operacje:

1. W środowisku *Eclipse* należy kliknąć PPM na nazwę projektu.
2. Następnie należy wybrać *Run As* oraz *Run On Server*.
3. W nowym oknie możemy wybrać serwer - należy wybrać *Tomcat v9.0* i przejść dalej.
4. W kolejnym kroku wybieramy aplikację na serwerze - należy upewnić się, że *meetUP-serwer* znajduje się na liście aplikacji skonfigurowanych - po prawej stronie.
5. Po kliknięciu *Finish* zostanie uruchomiona strona w wbudowanej przeglądarce.

Tak uruchomiona aplikacja na serwerze jest dostępna pod lokalnym adresem, na przykład: `http://localhost:8088/meetUP-server/db/rest/`.

Aby wdrożyć aplikację do chmury *IBM Cloud* należy najpierw przygotować plik *manifest.yml*. Przykładowy plik przedstawiono poniżej.

```
applications:
- buildpacks:
  - java_buildpack
  path: <nazwa archiwum war>
  memory: 700M
  instances: 1
  name: <nazwa aplikacji>
```

Konieczne jest posiadanie archiwum *war* projektu serwera, które możemy stworzyć np za pomocą środowiska *Eclipse*.

Następnie, należy wykonać następujące polecenia:

```
ibmcloud login
ibmcloud target --cf
ibmcloud cf push
ibmcloud logout
```

5.2 Klient

Aplikację klienta można uruchomić zarówno na maszynach wirtualnych używanych w ramach przedmiotu, jak i na własnym, lokalnym komputerze. Warunkiem jest posiadanie narzędzia `npm`.

Przed rozpoczęciem wdrażania należy uzupełnić plik `\meetup-client\src\config.js` w którym powinien znaleźć się link do serwera.

Aby uruchomić lokalnie aplikację należy wykonać następujące polecenia:

```
cd meetup-client
npm install
npm start
```

Aplikacja zostanie uruchomiona lokalnie, najczęściej na porcie 3000.

Aby wdrożyć aplikację do środowiska *IBM Cloud* należy najpierw przygotować plik `manifest.yml`. Przykładowy plik przedstawiono poniżej.

```
applications:
- buildpacks:
  - https://github.com/cloudfoundry/staticfile-buildpack.git
  path: build/
  memory: 500M
  instances: 1
  name: <nazwa aplikacji>
```

Powinien on znaleźć się w katalogu `meetup-client`. Następnie, aby zbudować aplikację klienta i wdrożyć ją do chmury należy wykonać polecenia:

```
npm install
npm run build
npm install cross-env
npm run build-prd

ibmcloud login
ibmcloud target --cf
ibmcloud cf push
ibmcloud logout
```

5.3 Generowanie dokumentacji

W projekcie została dodana dokumentacja kodu *Javadoc* (serwer) oraz *JSdoc* (aplikacja klienta). Poniżej przedstawione zostały instrukcje, jak wygenerować dokumentację *HTML* na podstawie przygotowanych adnotacji.

Aby wygenerować dokumentację *Javadoc* można skorzystać z narzędzia *Eclipse*. Należy wykonać kolejno kroki:

1. Wybrać z menu *Project*, a następnie *Generate Javadoc*.
2. W oknie należy wybrać ścieżkę do narzędzia (niestety, niedostępnego na maszynach wirtualnych, udostępnionych w ramach przedmiotu).
3. W kolejnych punktach należy wybrać projekt - *meetUP-server*, wszystkie klasy, widoczność *public*, oraz lokalizację dokumentacji.
4. W następnych krokach wybieramy tytuł, tagi - najczęściej odpowiednie będą opcje domyślne.
5. Przechodzimy do ostatniego okna i wybieramy *Finish*.

Aby wygenerować dokumentację *JSdoc* można użyć zainstalowanego modułu:

```
cd meetup-client/src/  
../node_modules/.bin/jsdoc -r .
```

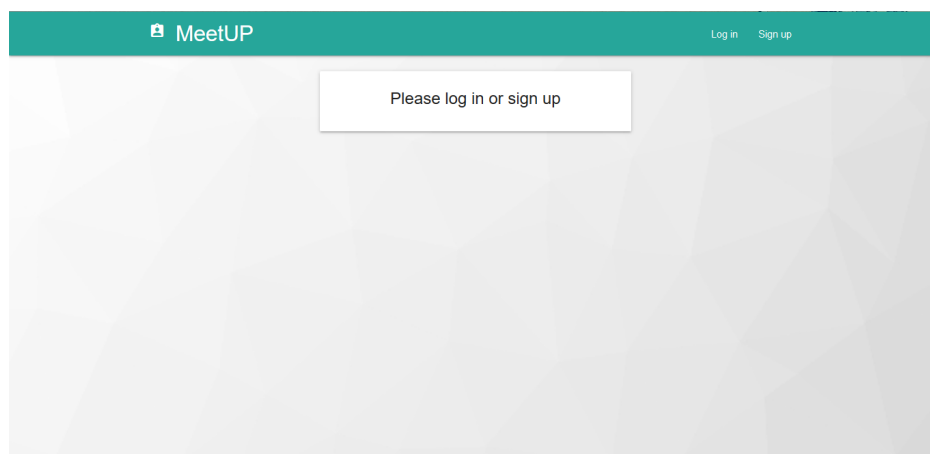
Tak wygenerowana dokumentacja *HTML* zostanie zapisana w folderze *out/*.

6 Podręcznik użytkownika

Poniższy rozdział zawiera instrukcję użytkownika do obsługi aplikacji klienta.

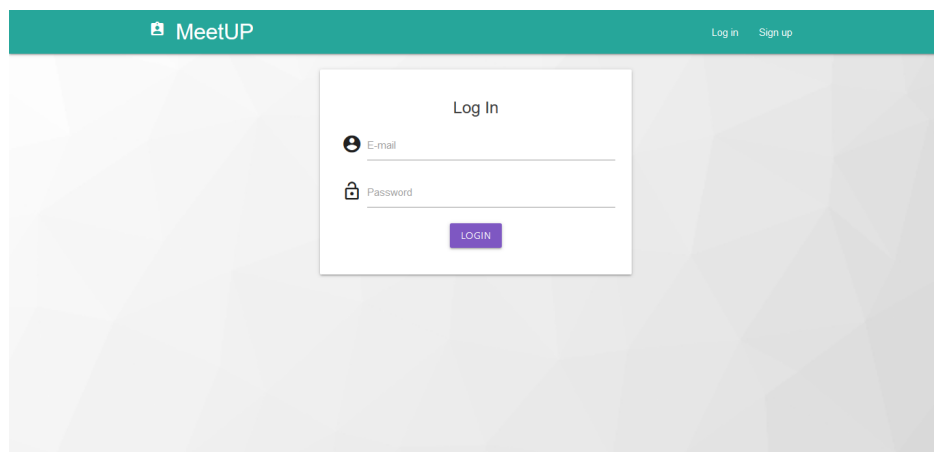
6.1 Logowanie oraz tworzenie konta

Gdy klient uruchomi aplikację, wyświetlony zostanie domyślny ekran.



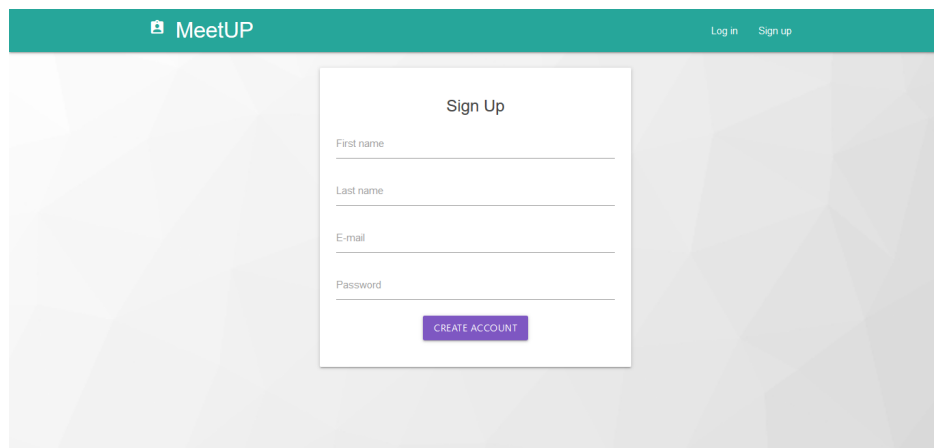
Rysunek 8: Strona główna aplikacji dla użytkownika niezalogowanego.

Na górze strony znajduje się pasek nawigacji, za pomocą którego użytkownik może się zalogować lub zarejestrować. Ekran z poszczególnymi formularzami zostały przedstawione poniżej.



The screenshot shows the MeetUP login interface. At the top, there is a teal header bar with the MeetUP logo on the left and 'Log in' and 'Sign up' links on the right. The main content area has a light gray background with a subtle geometric pattern. In the center, there is a white rectangular box titled 'Log In'. Inside this box, there are two input fields: the first is labeled 'E-mail' with an envelope icon, and the second is labeled 'Password' with a lock icon. Below these fields is a purple button labeled 'LOGIN'.

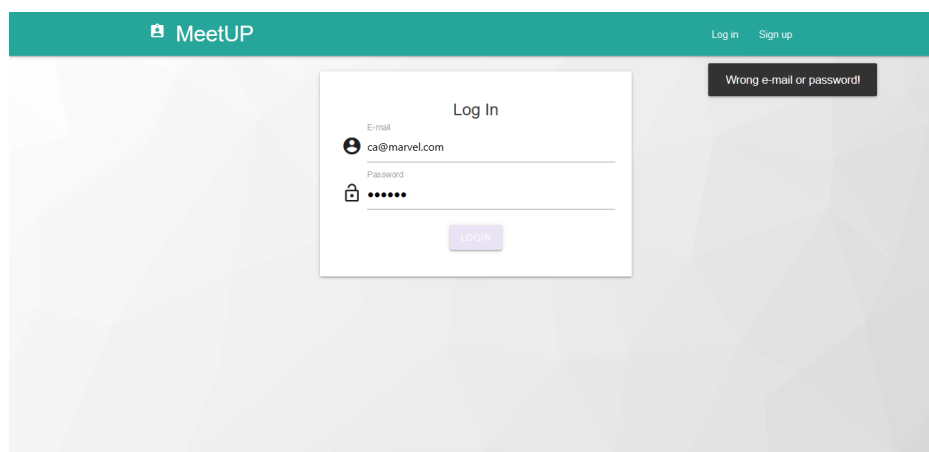
Rysunek 9: Formularz logowania.



The screenshot shows the MeetUP sign-up interface. It has the same teal header bar as the login page, with the MeetUP logo and 'Log in' and 'Sign up' links. The main content area features a light gray background with a geometric pattern. In the center, there is a white rectangular box titled 'Sign Up'. This box contains four input fields: 'First name', 'Last name', 'E-mail', and 'Password'. Below these fields is a purple button labeled 'CREATE ACCOUNT'.

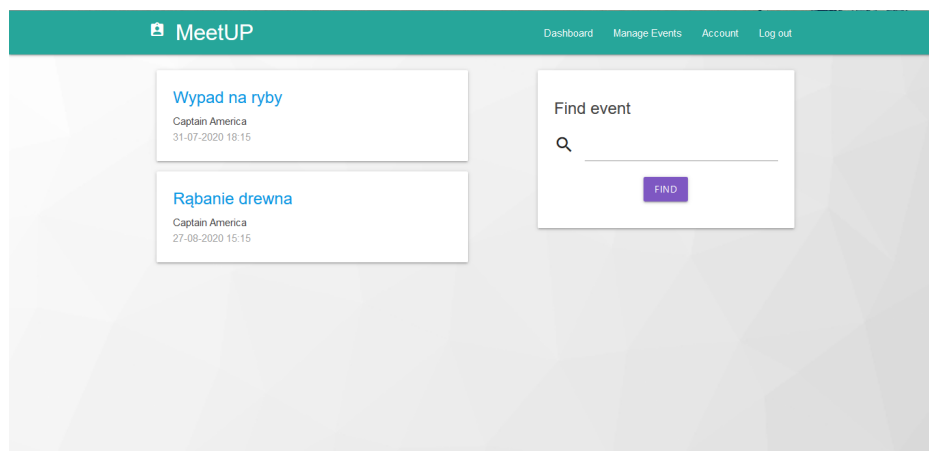
Rysunek 10: Formularz rejestracji.

Gdy użytkownik poda nieprawidłowe dane, na przykład hasło przy rejestracji zostanie wyświetlony odpowiedni komunikat.



Rysunek 11: Informacja o nieprawidłowych danych.

Gdy logowanie lub rejestracja powiedzie się, użytkownikowi ukaze się strona główna aplikacji.

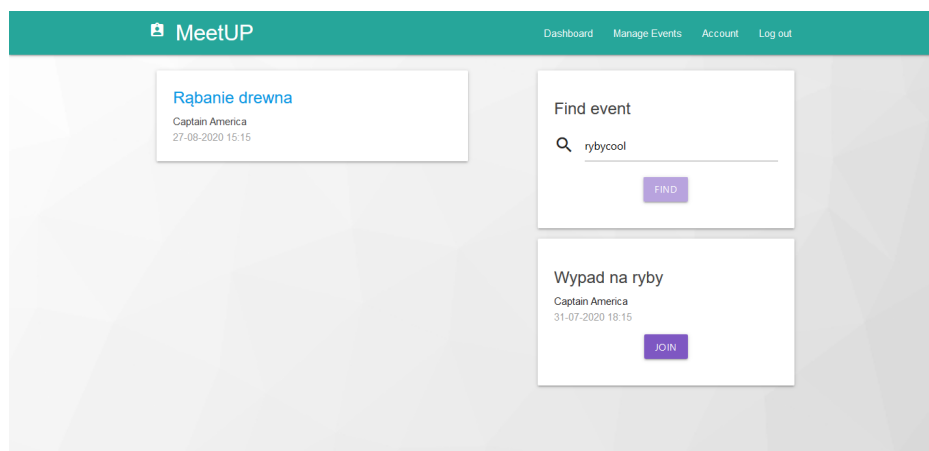


Rysunek 12: Strona główna aplikacji dla użytkownika zalogowanego.

Opcje paska nawigacji są już inne, zgodne z uprawnieniami użytkownika zalogowanego. Może on przeglądać wydarzenia, w których bierze udział, zarządzać wydarzeniami, których jest autorem, edytować swoje dane, czy też wylogować się. Operacje te zostaną opisane w dalszej części.

6.2 Wyszukiwanie wydarzenia i zapis

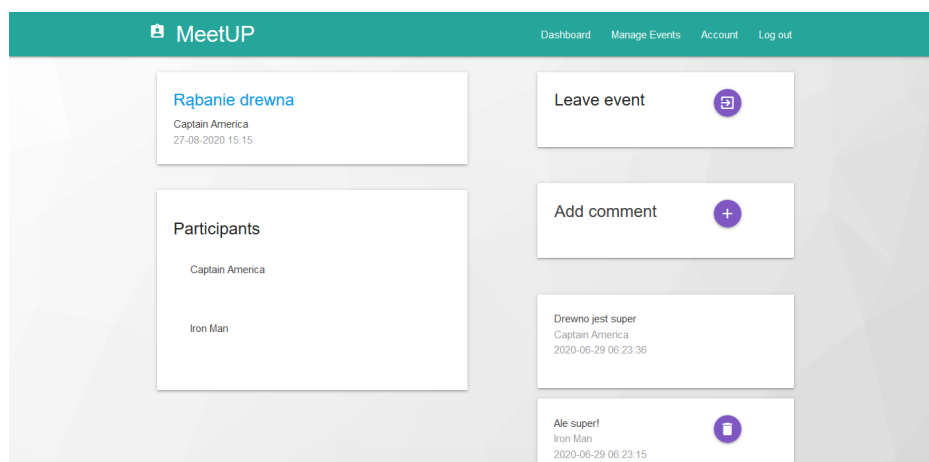
Wydarzenia, na które zapisał się użytkownik wyświetlone są po lewej stronie w kolejności chronologicznej. Nie są wyświetlane spotkania, których data już minęła. Po prawej stronie użytkownik może wyszukać nowe wydarzenia, na które może się zapisać. Identyfikacja następuje za pomocą unikalnego klucza, który ustala organizator.



Rysunek 13: Wyszukiwanie wydarzenia.

Gdy wydarzenie zostanie odnalezione (w razie niepowodzenia użytkownik zostanie poinformowany), można do niego dołączyć za pomocą przycisku *JOIN*. Po tej czynności wydarzenie zostanie dodane na listę wydarzeń użytkownika.

Poprzez kliknięcie na nazwę wydarzenia na stronie głównej można przeglądać jego szczegóły. Przykładowy widok znajduje się poniżej.



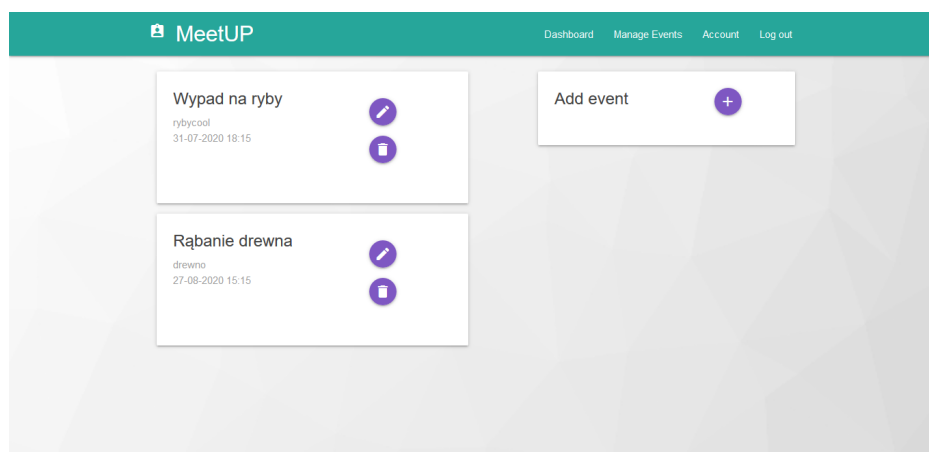
Rysunek 14: Szczegóły wydarzenia.

Na karcie po lewej stronie można znaleźć nazwę, organizatora oraz datę wydarzenia. Poniżej znajduje się lista wszystkich uczestników.

Po prawej stronie znajduje się panel do opuszczenia wydarzenia, do dodania komentarza oraz chronologiczna lista komentarzy. Organizator nie może opuścić wydarzenia - jedynie je usunąć. Użytkownik ma możliwość usunięcia komentarzy, których jest autorem.

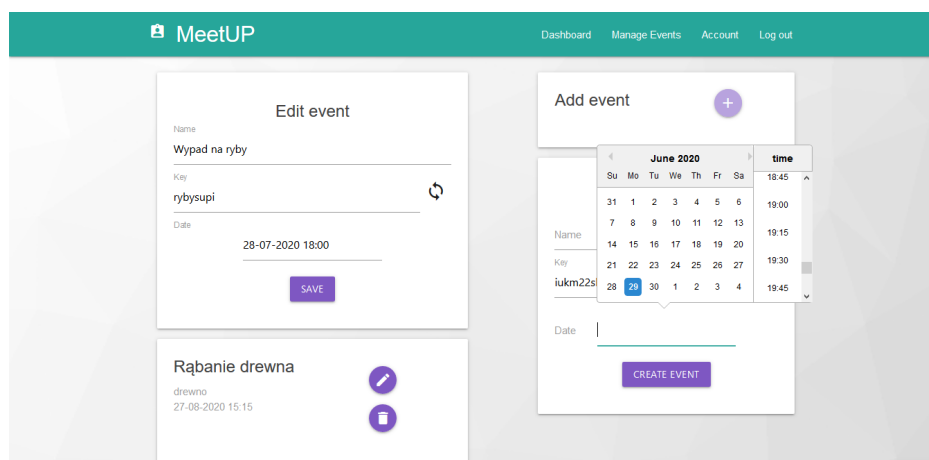
6.3 Tworzenie nowych wydarzeń

W zakładce *Manage events* użytkownik ma możliwość zarządzania wydarzeniami, których jest autorem.



Rysunek 15: Zarządzanie wydarzeniami.

Strona ta wygląda podobnie jak *Dashboard*, ale w tym przypadku użytkownik może edytować lub usunąć wydarzenia oraz dodać nowe. Przykładowe widoki zostały przedstawione poniżej.



Rysunek 16: Zarządzanie wydarzeniami - formularze.

Dodając bądź edytując wydarzenie użytkownik może wybrać datę z wygodnego widgetu i skorzystać z losowego generatora kluczy, bądź ustawić własny. Po zatwierdzeniu operacji przyciskiem dane zostają zapisane.

Literatura

- [1] Dokumentacja *Java Persistence API* - <https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html> (dostęp: 02.07.2020)
- [2] Dokumentacja narzędzia *npm* - <https://docs.npmjs.com/> (dostęp: 02.07.2020)
- [3] Dokumentacja frameworka *React* - <https://reactjs.org/docs/getting-started.html> (dostęp: 02.07.2020)
- [4] Dokumentacja biblioteki *Redux* - <https://redux.js.org/introduction/getting-started> (dostęp: 02.07.2020)
- [5] *Materialize.js* - <https://materializecss.com/> (dostęp: 02.07.2020)