

Programowanie Aplikacji Webowych

laboratorium 10

Cel zajęć:

Celem laboratorium jest przećwiczenie zagadnień związanych z obsługą rzeczywistych danych produkcyjnych pochodzących z serwera z danymi. Nasze dane będą w sposób trwały przechowywane po stronie Backendu.

Tworzenie backendu:

Potrzebujemy więc jakiegoś mechanizmu zapisu danych w sposób trwały. W tym celu musimy skorzystać z funkcjonalności serwera aplikacyjnego (backend) z którym będziemy komunikować się za pomocą wywołań REST. Interesuje nas oczywiście tylko rozwiązanie bazujące na stosie JS.

Są dwa sposoby na realizację tego zadania:

- Skorzystanie z gotowej platformy zbudowanej w oparciu o NodeJS – Google Firebase (ścieżka łatwiejsza)
- Samodzielne zbudowanie serwera aplikacyjnego w oparciu o NodeJS i moduł Express. (ścieżka trudniejsza)

Ścieżka łatwiejsza.

Do tego celu użyjemy gotowe środowisko backendowe – Firebase. W zasadzie cała nasza aktywność sprowadzi się do stworzenia konta oraz przygotowania danych w dostępnej bazie danych. Do wyboru mamy bazę typu RealTime lub Firestore gwarantujące aktualizacje naszego Frontendu w przypadku modyfikacji danych w bazie.

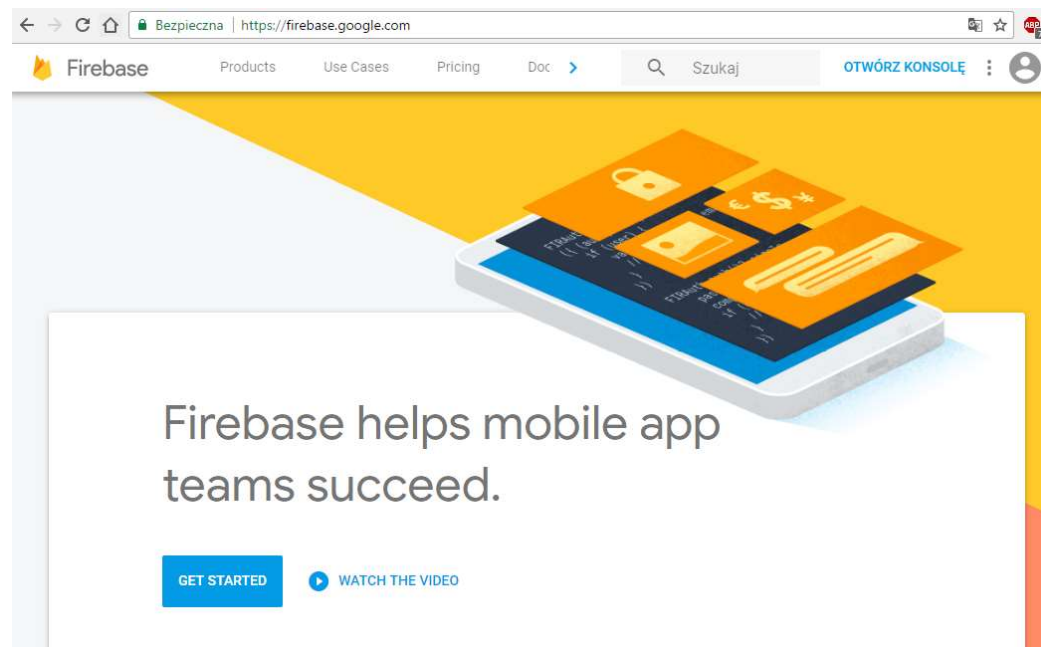
Ze względu na chęć przećwiczenia współpracy z obiektem `httpClient` jeden z projektów zrealizujemy w oparciu o współpracę z fejkowym serwerem `jsonplaceholder` znajdującym się pod adresem <https://jsonplaceholder.typicode.com/>, a który dostarcza kilka przykładowych grup danych.

Informacje wstępne na temat Firebase.

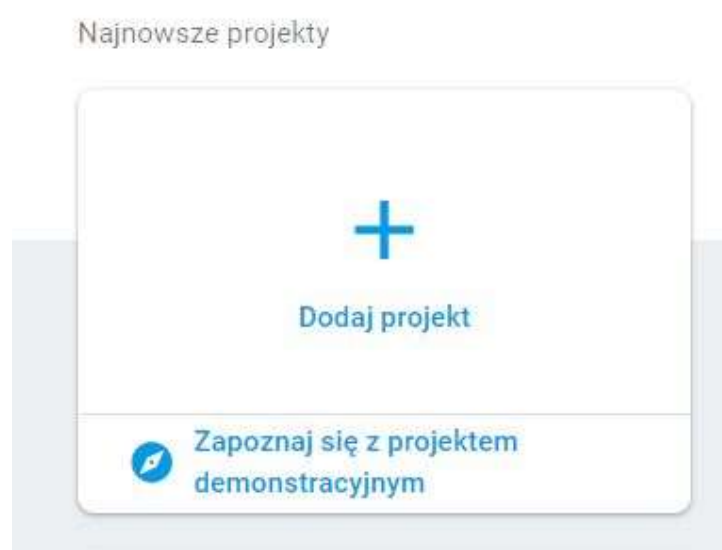
Firebase to tzw. **BaaS** (Backend as a Service), który umożliwia min. przechowywanie danych w formacie JSON oraz plików binarnych (np. jpg, mp4). Firebase dostarcza nam dwie bazy danych typu real-time database, gdzie komunikacja z serwerem jest oparta o Websockets,

dzięki czemu po aktualizacji danych, klient automatycznie dostaje najświeższe dane. Obsługa jest banalnie prosta, w podstawowym zakresie można niemalże wszystko wygenerować z panelu użytkownika! Dobrze rozwiązanie na początek, gdy chce się postawić w szybkim czasie serwer z danymi a nie ma się czasu lub umiejętności aby zrobić to samodzielnie.

Zanim rozpoczniemy komunikację z serwerem, musimy oczywiście sobie go wcześniej przygotować. Rozpoczynamy od odwiedzenia strony [www.firebase.com](https://firebase.google.com), założenia konta, a następnie po zalogowaniu klikamy przycisk „**OTWÓRZ KONSOLĘ**” w prawym górnym rogu:



Następnie w kolejnym widoku, klikamy „**UTWÓRZ PROJEKT**”:



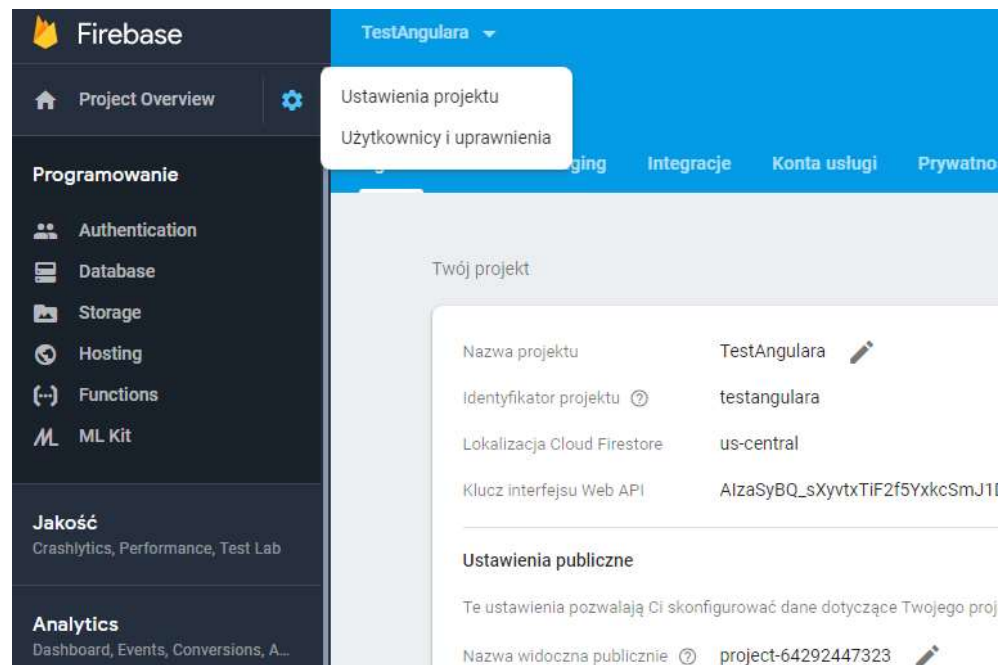
W okienku podajemy nazwę projektu, akceptujemy regulamin i klikamy „UTWÓRZ PROJEKT” i czekamy cierpliwie na powstanie projektu.

AngularFire

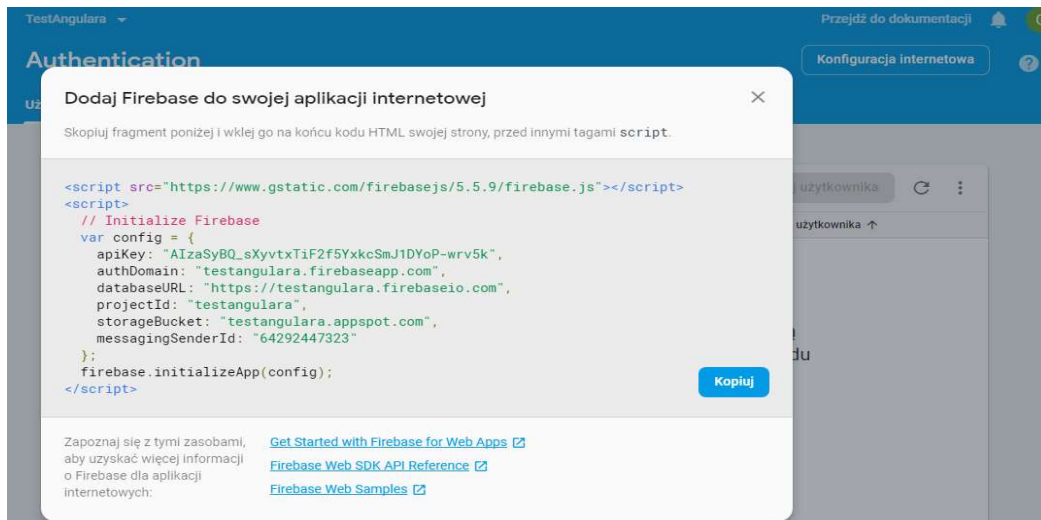
W celu komunikacji z naszym kontem w Firebase z poziomu tworzonej aplikacji webowej należy użyć dedykowanej biblioteki – [AngularFire](#), który jest wrapperem na bibliotekę [firebase.js](#). Pozwoli nam się zsynchronizować z danymi w czasie rzeczywistym, oraz dostarcza bardzo dobre API do logowania i monitorowania uwierzytelniania użytkownika. Rozpoczynamy od instalacji paczki:

```
npm install firebase @angular/fire --save
```

Po instalacji, wracamy do panelu Firebase i przechodzimy do ustawień projektu, klikając na zębatkę obok „PROJECT OVERVIEW” w lewym menu:



Scrollujemy nieco w dół i klikamy „Dodaj Firebase do swojej aplikacji internetowej”. Pojawia się okienko z danymi projektu:



Kopiujemy sam obiekt przypisany do „var config” i zapisujemy np. w jakimś pliku tekstowym. Reszta nas nie interesuje.

Podpięcie się pod Firebase

Wracamy do aplikacji angularowej i wklejamy plik konfiguracyjny do plików **environment.ts** oraz **environment.prod.ts** w katalogu `src/environments`. Jest to dobre miejsce na trzymanie takich globalnych ustawień. Obiekt możemy przypisać np. do pola „firebaseConfig”:

```
1 // The file contents for the current environment will overwrite these during build.
2 // The build system defaults to the dev environment which uses `environment.ts`, but if you do
3 // `ng build --env=prod` then `environment.prod.ts` will be used instead.
4 // The list of which env maps to which file can be found in `.angular-cli.json`.
5
6 export const environment = {
7   production: false,
8   firebaseConfig: {
9     apiKey: "AIzaSyBQ_sXyvtxTiF2f5YxkcSmJ1DYOp-wrv5k",
10    authDomain: "testangulara.firebaseio.com",
11    databaseURL: "https://testangulara.firebaseio.com",
12    projectId: "testangulara",
13    storageBucket: "testangulara.appspot.com",
14    messagingSenderId: "64292447323"
15  };
16 };
```

Następnie przechodzimy do `app.module.ts` i importujemy następujące moduły (**AngularFireModule**).

```
import { AngularFireModule } from "@angular/fire";
import { environment } from '../environments/environment';
```

.....

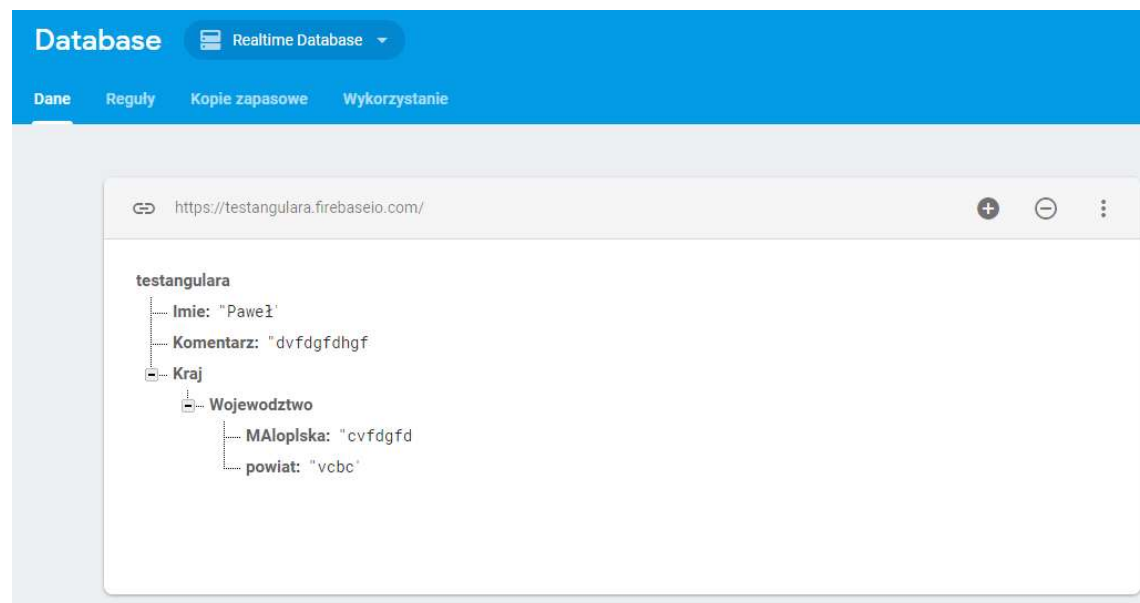
```
imports: [  
  AngularFireModule.initializeApp(environment.firebase),  
],
```

Zwróć uwagę na wywołanie metody `initializeApp` z obiektem konfiguracyjnym, który zapisaliśmy w pliku `environment.ts` i `environment.prod.ts`. Teraz nasza aplikacja staje się świadoma backendu dostarczonego przez Firebase.

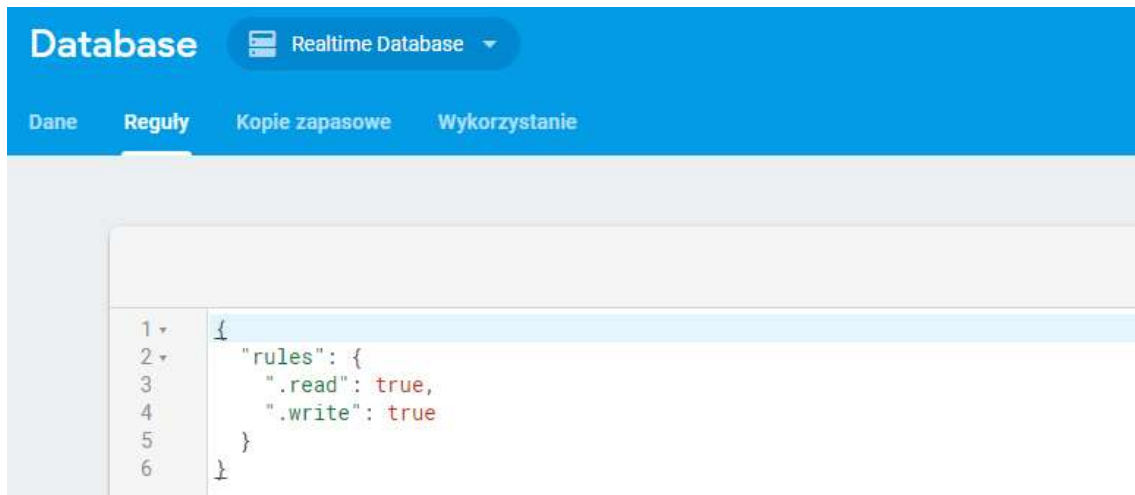
Obsługa bazy danych

Głównym celem użycia FireBase było skorzystanie z wbudowanej bazy danej, która miała za zadanie wspierać operacje CRUD na obiektach JASON.

Pierwszym krokiem jest stworzenie w bazie przykładowych danych. Możemy użyć RealTime DataBase lub Cloud Firestore. W obu przypadkach tworzenie danych w bazie jest banalnie proste.



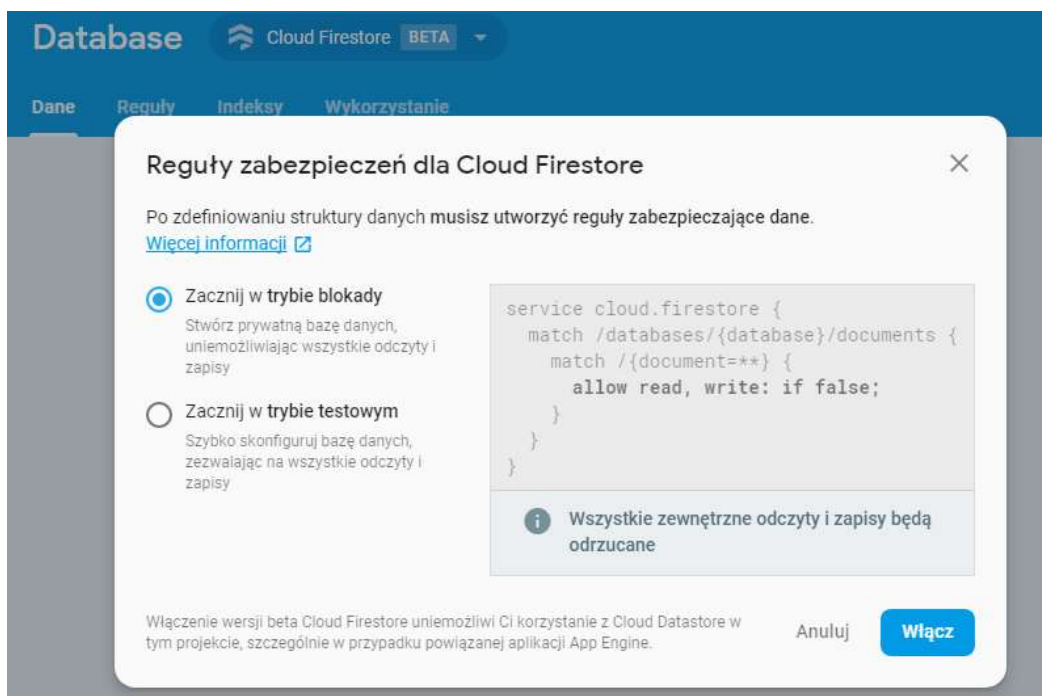
Nie wolno zapomnieć o ustawieniu reguł dostępowych – domyślnie obie wartości są ustawione na `false`.



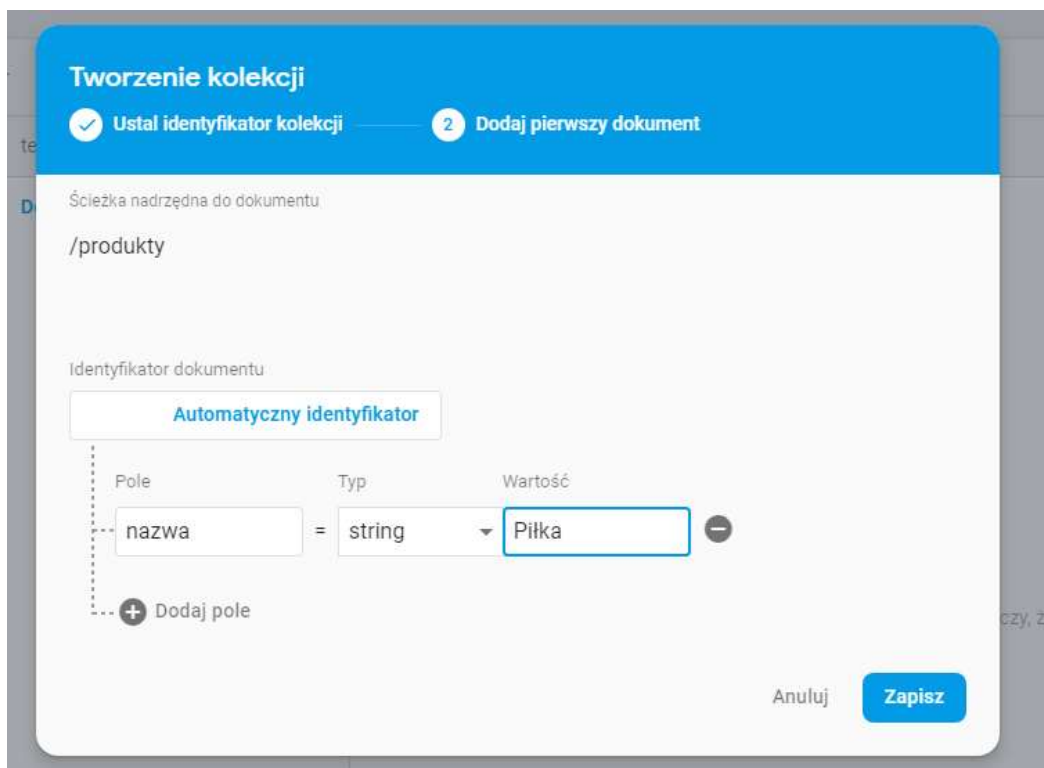
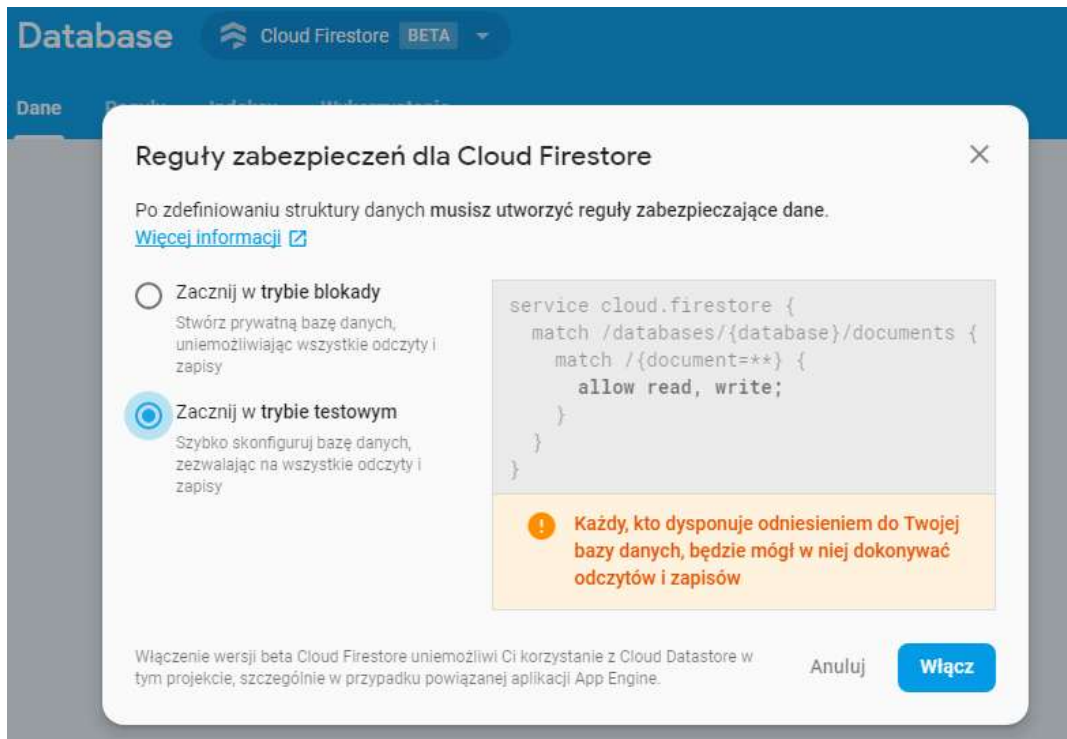
lub tak gdy zrezygnowaliśmy z autoryzacji

```
{  
  "rules": {  
    ".read": "auth == null",  
    ".write": "auth == null"  
  }  
}
```

W przypadku wybrania wersji Cloud



Lub



Po stworzeniu bazy w Firebase przechodzimy do Angulara.

W zależności od użytej bazy :

W module app.module.ts importujemy albo

```
import { AngularFireStore } from '@angular/fire/store';
```

albo

```
import { AngularFireDatabase } from '@angular/fire/database';
```

w zależności od tego, z której z baz chcemy korzystać.

...

```
@NgModule({
  imports: [
    AngularFireModule.initializeApp(environment.firebaseConfig),
    BrowserModule,
    AppRoutingModule,
    AngularFireDatabaseModule lub AngularFirestoreModule
  ],
  ...
})
```

Teraz w odpowiednim komponencie należy poprzez dedykowaną usługę dostarczyć zawartość bazy danych.

Przykładowa implementacja

```
import { AngularFireDatabase } from '@angular/fire/database';

export class GRFirestoreService {
  constructor(private db: AngularFireDatabase) { }
}
```

AngularFireDatabase

do obsługi pojedynczego Objektu

– wiązanie referencji z obiektem w bazie danych - odczyt pojedynczego

```
daneRef: AngularFireObject<any>;
```

```
this.daneRef = db.object('student');
```

```
// lub tak
```

```
Observable<any> dane = db.object('student').valueChanges();
```

– tworzenie obiektu w bazie:


```
const daneRef = db.object('student');  
daneRef.set({ name: 'GR' });
```

– edycja danych w bazie

```
const daneRef = db.object(' student');  
daneRef.update({ item2: 'cos tam' });
```

– Usuniecie obiektu:

```
const daneRef = db.object('student');  
daneRef.remove();
```

obsługa Listy (dane jako lista elementów)

Odczyt danych :

+ pobieranie danych jako tablica obiektów JSON bez metadanych (tylko same dane)

```
daneRef: Observable<any[]>;
```

```
this.daneRef = db.list('students').valueChanges();
```

+ pobieranie danych jako tablica obiektów JSON wraz z metadata
(DatabaseReference oraz dokument id, lub index tablicy):

```
daneRef: Observable<any[]>;
```

```
this.daneRef = db.list('students').snapshotChanges();
```

Dodanie nowego ellementu do listy:

```
const daneRef = db.list('students');  
daneRef.push({ name: 'GR', item: 'cos tam dodaje' });
```

Aktualizacja listy:

```
const daneRef = db.list('students');  
daneRef.set('key', { name: 'ktos inny', item: 'cos innego' });  
+ lub zmiana tylko specyficznej wartosci  
const daneRef = db.list('students');  
daneRef.update('key', { name: 'kolejny ktos' });
```

Usuniecie obiektu z listy:

```
const daneRef = db.list('students');  
daneRef.remove('key');
```

Usuniecie calej listy:

```
const daneRef = db.list('students');  
daneRef.remove();
```

AngularFirestore

dla documentu

odczyt danych:

```
daneRef: AngularFirestoreDocument<any>;  
this.daneRef = db.doc('students');  
// lub  
Observable<any> daneRef = db.doc('students').valueChanges();
```

– Dodanie/utworzenie dokumentu :

```
const daneRef = db.doc('students');
```

```
daneRef.set({ name: 'GR' });
```

– Aktualizacja

```
const daneRef = db.doc('students');  
daneRef.update({ item2: 'cos tam' });
```

– Usuniecie:

```
const daneRef = db.doc('students');  
daneRef.delete();
```

AngularFirestore

Kolekcja (Collection)

Odczyt danych :

+ pobieranie danych jako tablica obiektów JSON bez metadanych (tylko same dane)

```
daneRef: Observable<any[]>;  
this.daneRef = db.collection('students').valueChanges();
```

+ pobieranie danych jako tablica obiektów JSON wraz z metadata
(DatabaseReference oraz dokument id, lub index tablicy):

```
daneRef: Observable<any[]>;  
this.daneRef = db.collection('students').snapshotChanges();
```

Dodanie nowego elementu do kolekcji:

```
const daneRef = db.collection('students');  
const dane = { name: 'GR', item: 'cos tam dodaje' };  
daneRef.add({ ... dane });
```

Aktualizacja kolekcji:

```
const daneRef = db.collection('students');  
daneRef.doc('id').set({ name: 'ktos inny', item: 'cos innego' });  
+ lub zmiana tylko specyficznej wartosci  
const daneRef = db.collection('students');  
daneRef.doc('id').update({ name: 'kolejny ktos' });
```

Usuniecie obiektu z kolekcji:

```
const daneRef = db.collection('students');  
daneRef.doc('id').delete();
```

Zadanie 1. Podgląd wycieczki – widok szczegółowy. (3pkt)

Aktualnie wszystkie informacje o wycieczce dostępne są w widoku wyświetlającym listę wszystkich wycieczek. Nie jest to najwygodniejsze rozwiązanie dlatego proponuje stworzyć nowy komponent wyświetlający szczegóły dotyczące wycieczki. Przejście do szczegółów wycieczki odbywać się będzie z poziomu widoku listy wycieczek.

Po kliknięciu na zdjęcie wycieczki powinniśmy zostać przekierowani do widoku konkretnej wycieczki. Widok szczegółowy powinien w atrakcyjny sposób prezentować wszystkie szczegóły związane z informacją o wycieczce. Dodatkowo tylko z poziomu szczegółów wycieczki ma być możliwość oceniania. Przenosimy więc tę funkcjonalność z poziomu listy wycieczek do widoku szczegółowego. Duplikujemy również na tym poziomie możliwość rezerwacji i rezygnacji z wycieczki.

Na widoku pojedynczej wycieczki powinniśmy zobaczyć:

- nazwę wycieczki wraz z lokalizacją
- mapę prezentującą lokalizację destynacji
- lista zdjęć prezentującej wycieczkę – np. w postaci karuzeli lub innego slajdera
- Opis szczegółowy wycieczki

- Cena £/USD w zależności od ustaleń globalnego ustawienia
- sumaryczna ocena (ocena i liczba głosujących)
- link aby wrócić do widoku wszystkich z lista wszystkich wycieczek
- formularz recenzji i już dodane recenzje (2pkt)

Formularz recenzji powinien zawierać:

- Użyty NICK
- wprowadzenie nazwy (wymagane)
- obszar tekstu recenzji (wymagany, recenzja powinna być dłuższa niż 50 znaków, ale krótsza niż 500 znaków)
- data zakupu (typ wejścia="data") (opcjonalnie)

Po przesłaniu formularza zweryfikuj wszystkie pola. Wszystkie pola poza datą zakupu są wymagane. Gdyby coś nie jest ok np. opinia jest za krótka lub wymagany atrybut jest pusty, dodaj nowy błąd do tablicy błędów i wyświetlaj je. Po każdym przesłaniu formularza wyczyść tablicę błędów. Jeśli formularz jest poprawny, dodaj opinię do tablicy opinii (tylko lokalnie, nie wysyłaj żadnych zapytań) i wyczyść formularz pola. Powinniśmy być w stanie dodać tyle recenzji, ile chcemy.

Zadanie 2. Lista wycieczek – paginacja. (1pkt)

Lista wycieczek może zawierać wiele pozycji. Dodaj do widoku z lista wycieczek obsługę stronicowania (paginacji) danych wyświetlanych. Powinna umożliwiać zmiany ilość pozycji wyświetlanych na stronie. Możesz użyć gotowego komponentu lub napisać tę funkcjonalność samodzielnie.

Realizacja Backend

Możliwa jest realizacja backendu w wersji:

Łatwiejszej:

Zadanie 3. Biuro Turystyczne – Integracja z FireBase (4 pkt)

Na podstawie materiałów znajdujących się na początku lab zintegruj swoją aplikację z platformą Firebase. Niech dane o potrawach pochodzą z Firebase. Pozwoli to nam na rzeczywistą persystencję danych. Wybierz dowolny typ bazy danych znajdujący się w Firebase i zaimplementuj usługę pozwalającą na odczyt, dodawanie, modyfikacje lub usuwanie nowej pozycji z bazy.

Lub

Trudniejszej:

Zadanie 4. Restauracja Online – Własny serwer REST API (9 pkt)

W wersji trudniejszej wymagane jest samodzielne napisanie serwera RESTAPI z wykorzystaniem NodeJS/ExpressJS. Do przechowywania danych możesz użyć bazy w FireBase lub bazy MongoDB znajdującej się w chmurze pod adresem <https://www.mongodb.com/atlas/database>. Alternatywnie przy samodzielnej implementacji serwera możesz skorzystać z baz mySQL używać np. Squalize do komunikacji

Materiały potrzebne do realizacji znajdziesz w moich materiałach wykładowych. Po zaimplementowaniu serwera przetestuj jego funkcjonalność używając narzędzi typu np. curl lub Postman, ewentualnie zainstaluj rozszerzenie Thunder Client w Visual Studio Code.

W aplikacji klienckiej zaimplementuj usługę do komunikacji z serwerem aplikacyjnym. Wykorzystaj do tego celu moduł HttpClient. Następnie wywołaj usług w tych komponentach, które pozwolą Ci na zasilania danymi całej aplikacji oraz obsłużą metody do usuwania, dodawania i modyfikacji danych w aplikacji.

Punktacja szczegółowa:

2 pkt poprawne zdefiniowanie modelu + połączenie z bazami danych(wycieczka + recenzja)

2 pkt zdefiniowanie i implementacja routingu i endpointów (uwzględniając rest API)

5 pkt - zdefiniowanie kontrolerów dla poszczególnych endpoints

Zadanie 5. Biuro Turystyczne – dynamiczny wybór źródła danych (2 pkt)

Zintegruj w swojej aplikacji obie wersje realizacji backendu. Za pomocą np. opcji wyboru określ z którym wersja backendu chcesz współpracować. Odpowiednio zaprojektuj swoją aplikację tak aby można było dynamicznie zmieniać źródła danych.