Programowanie Aplikacji Webowych

laboratorium 11

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 wiec 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 realizacje tego zagadnienia:

- 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.

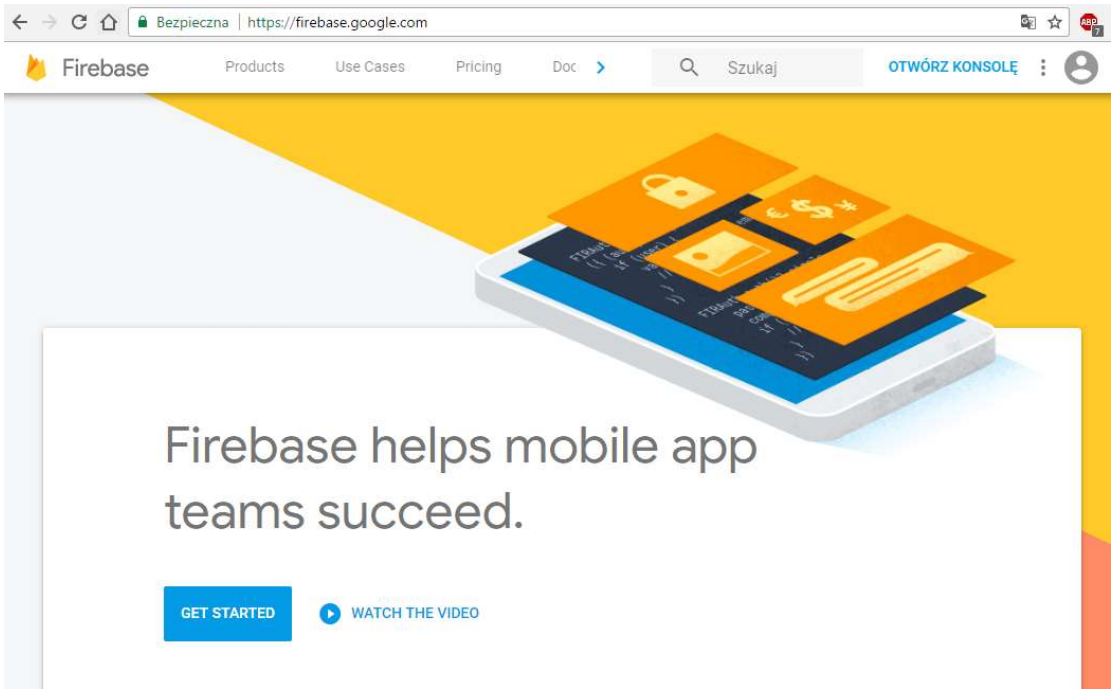
Ze względu na chęć przećwiczenia współpracy z obiektem httpClient jeden z projektów zrealizujemy w oparciu o współprace z fejkowym serwerem jasonplaceholder znajdującym się pod adresem https://jsonplaceholder.typicode.com/, a który dostarcza kilka przykładowych grup danych.

Informacje wstępne na temat Firebase.

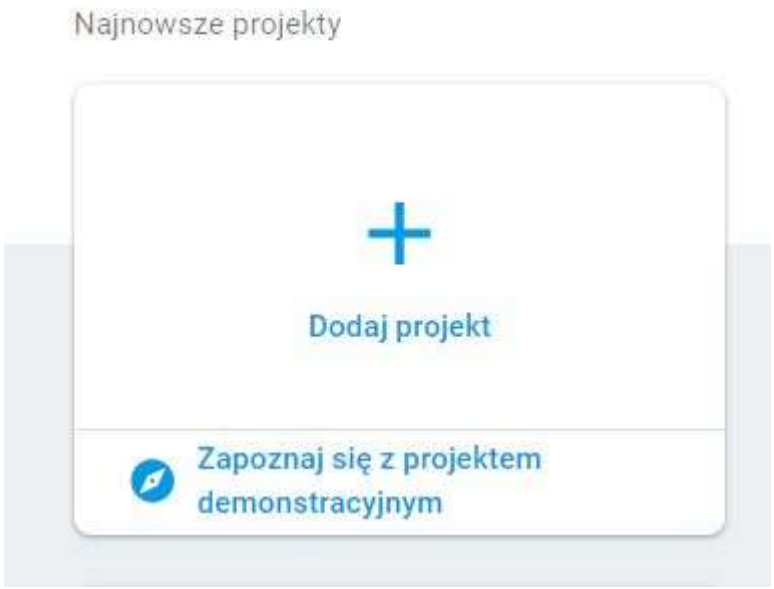
Firebase to tzw. **BaaS** (Backend as a Service), który umożliwia min. przechowywanie danych w formacie JSON oraz plików binarnych (np. jpg, mp4). Firebase dostarcza nam dwie bazy danych typu real-time database, gdzie komunikacja z serwerem jest oparta o Websockets,

dzięki czemu po aktualizacji danych, klient automatycznie dostaje najświeższe dane. Obsługa jest banalnie prosta, w podstawowym zakresie można niemalże wszystko wygenerować z panelu użytkownika! Dobre 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, założenia konta, a następnie po zalogowaniu klikamy przycisk „**OTWÓRZ KONSOLĘ**” w prawym górnym rogu:



Następnie w kolejnym widoku, klikamy „UTWÓRZ PROJEKT”:



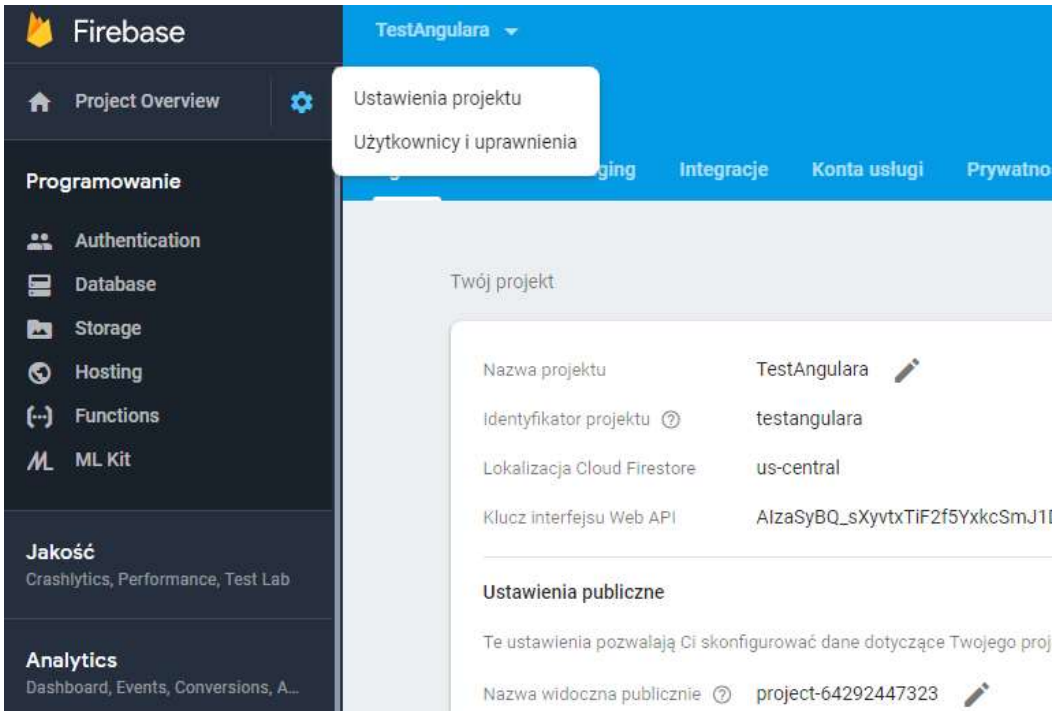
W okienku podajemy nazwę projektu, akceptujemy regulamin i klikamy „UTWÓRZ PROJEKT” i czekamy cierpliwie na powstanie projektu.

AngularFire

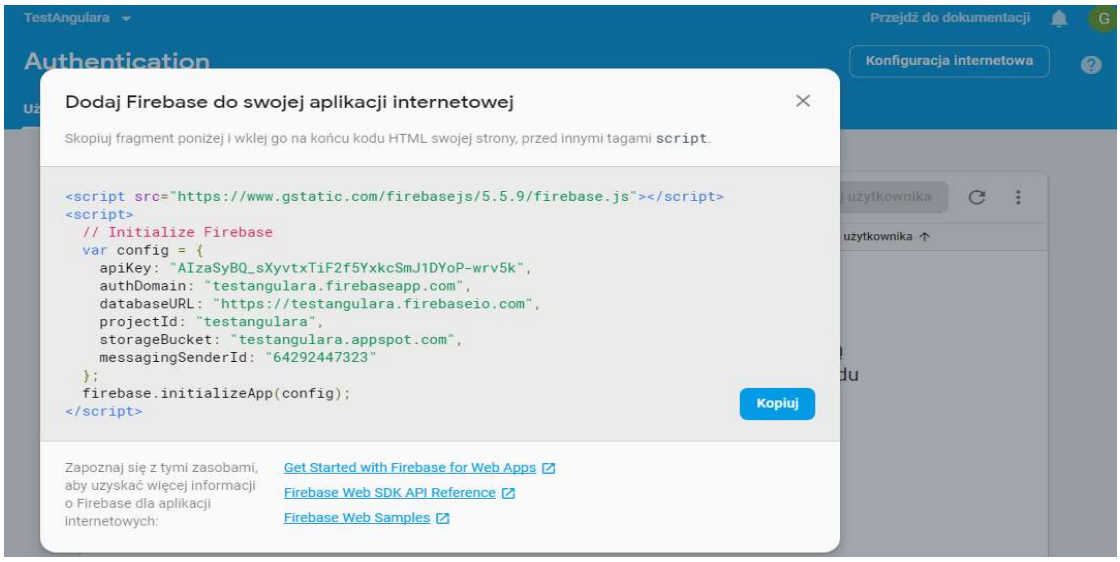
W celu komunikacji z naszym kontem w Firebase z poziomu tworzonej aplikacji webowej należy użyć dedykowanej biblioteki – **AngularFire**, który jest wrapperem na bliblioteką **firebase.js**. Pozwoli nam się zsynchronizować z danymi w czasie rzeczywistym, oraz dostarcza bardzo dobre API do logowania i 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":

```
// The file contents for the current environment will overwrite these during build.
// The build system defaults to the dev environment which uses `environment.ts`, but if you do
// `ng build --env=prod` then `environment.prod.ts` will be used instead.
// The list of which env maps to which file can be found in `.angular-cli.json`.

export const environment = {
    production: false,
    firebaseConfig: {
        apiKey: "AIzaSyBQ_sXyvtxTiF2f5YxkcSmJ1DYoP-wrv5k",
        authDomain: "testangulara.firebaseapp.com",
        databaseURL: "https://testangulara.firebaseio.com",
        projectId: "testangulara",
        storageBucket: "testangulara.appspot.com",
        messagingSenderId: "64292447323"
  };
};
```

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

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

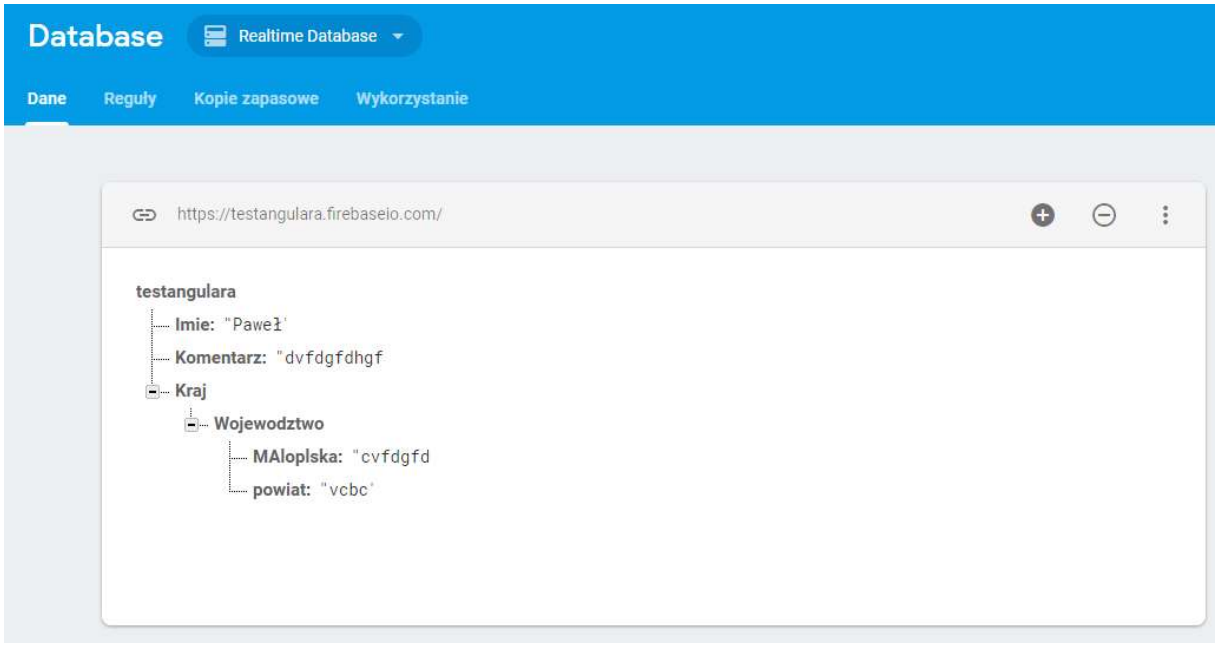
```
……..
imports: [
   AngularFireModule.initializeApp(environment.firebase),
],
```

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.

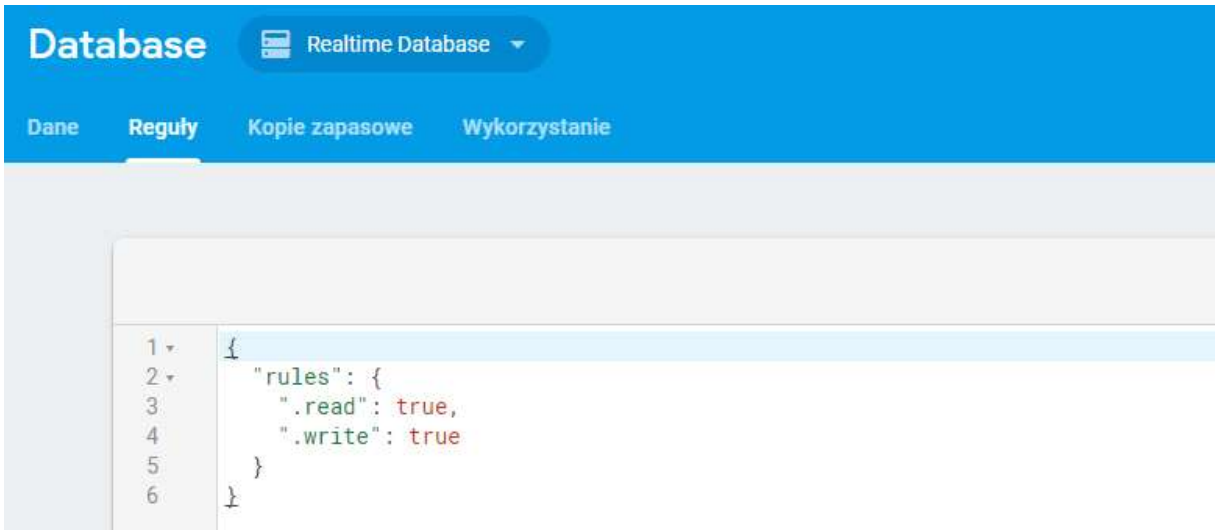
Obsługa bazy danych

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.



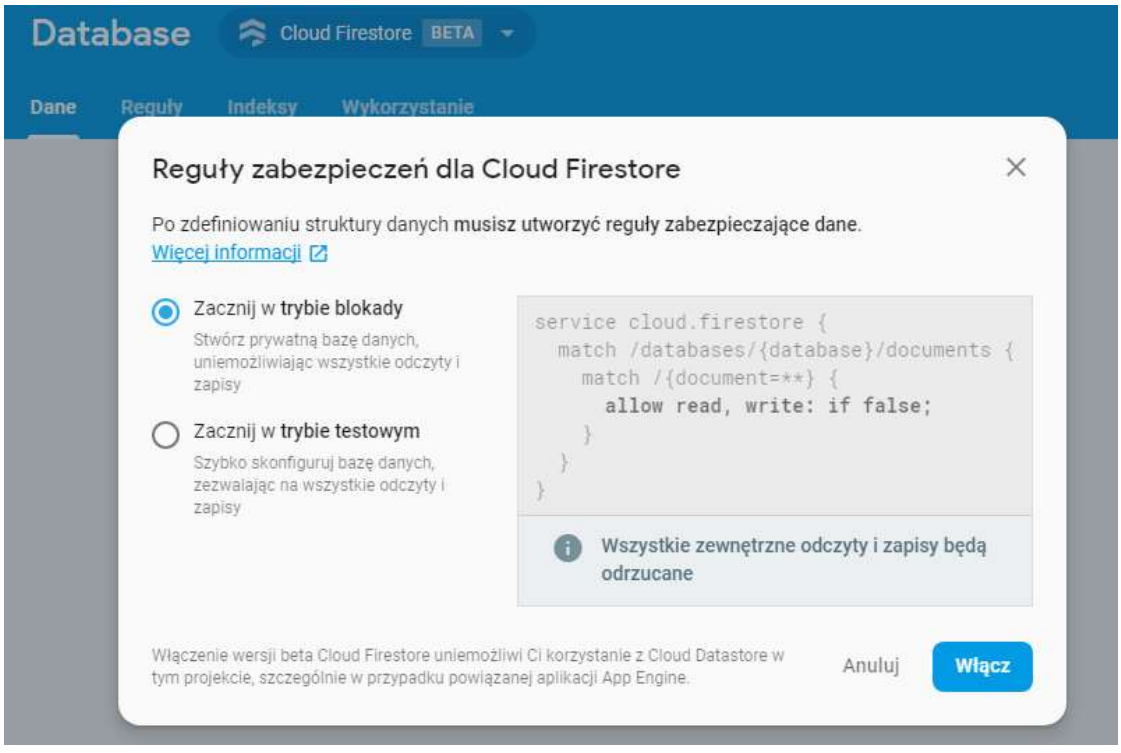
Nie wolno zapomnieć o ustawieni reguł dostępowych – domyślnie obie wartości są ustawione na false.



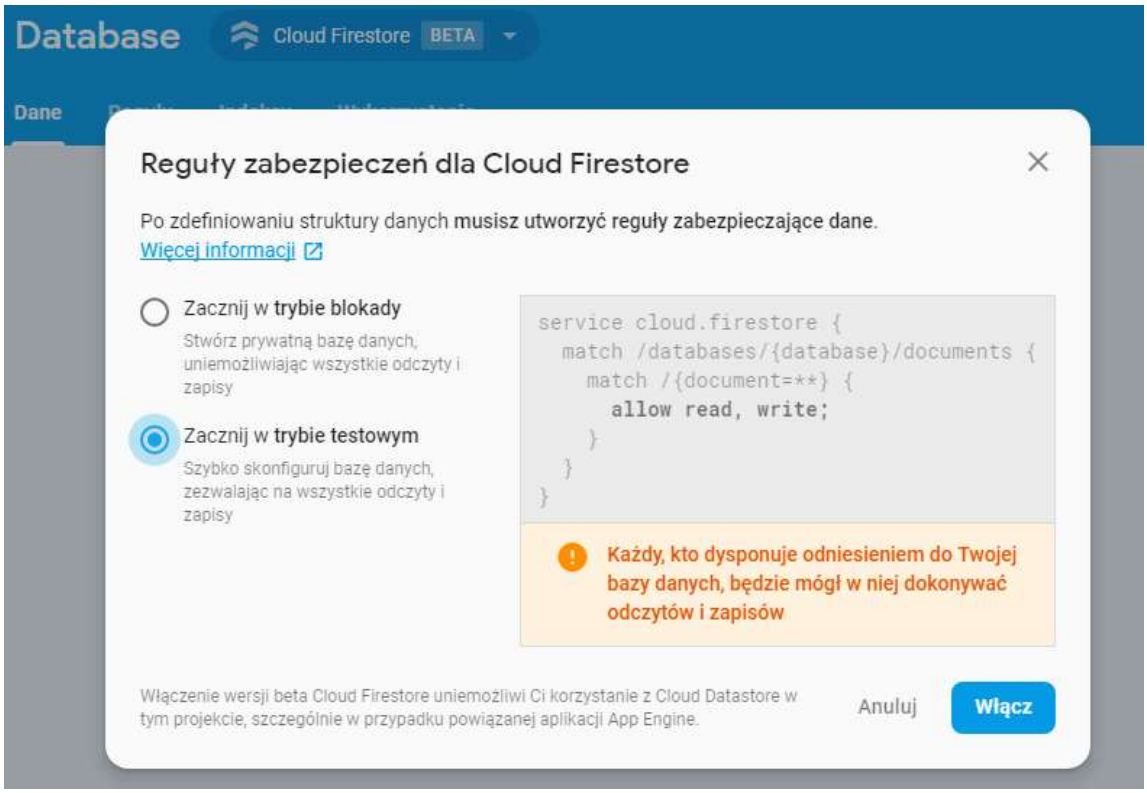
lub tak gdy zrezygnowaliśmy z autoryzacji

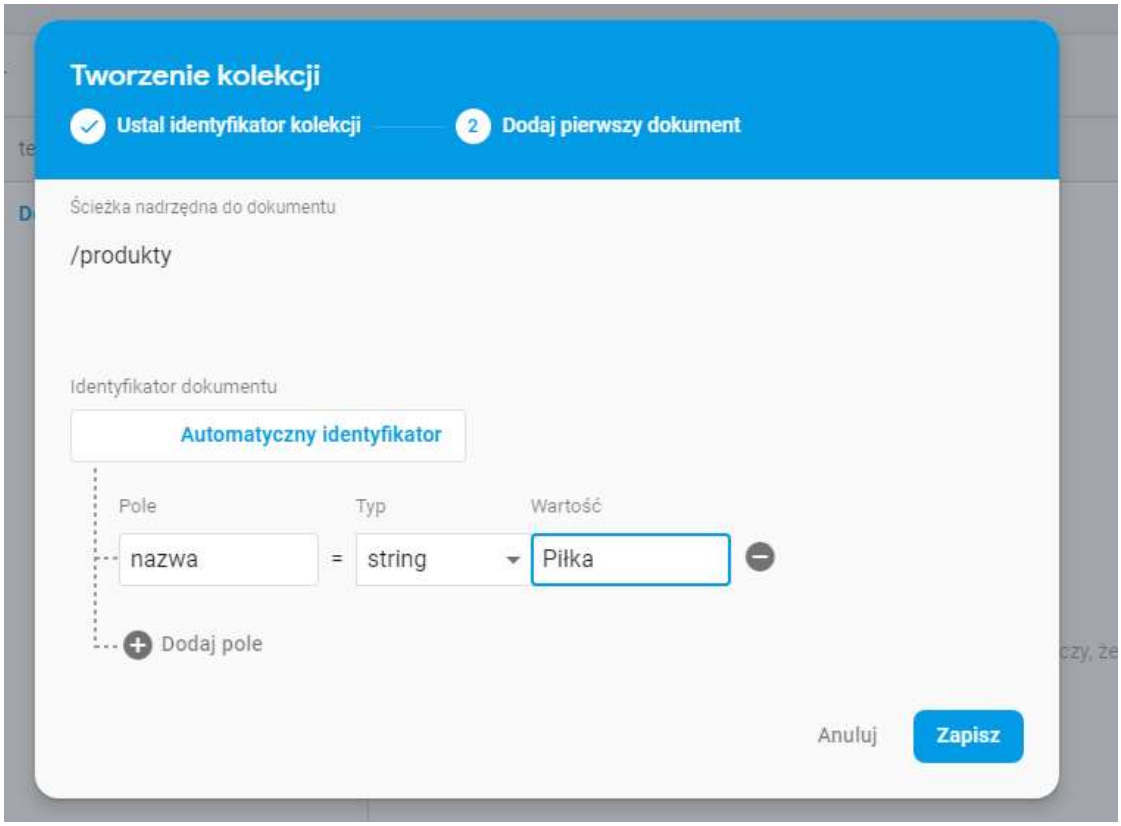
```
{
  "rules": {
    ".read": "auth == null",
    ".write": "auth == null"
  }
}
```

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

– wiazanie referencji z obiektem w bazie danych - odczyt pojedy

```
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 elelmentu 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();
```

Realizacja Backend

Możliwa jest realizacja backendu w wersji:

Łatwiejszej:

Zadanie 1. Aplikacja do zdalnych konsultacji lekarskich – Integracja z FireBase (3 pkt)

Na podstawie materiałów znajdujących się na początku lab zintegruj swoja aplikację z platformą Firebase. Niech dane o zaplanowanych terminach konsultacji oraz wszystkich ograniczeniach i zarezerwowanych konsultacjach pochodzą z Firebase. Pozwoli to nam na rzeczywistą persystencje danych. Wybierz dowolny typ bazy danych znajdujący się w Firebase i zaimplementuj usługę pozwalająca na odczyt, dodawanie, modyfikacje lub usuwanie nowej pozycji z bazy.

Lub

Trudniejszej:

Zadanie 2. Aplikacja do zdalnych konsultacji lekarskich – Własny serwer REST API (7 pkt)

W wersji trudniejszej wymagane jest samodzielne napisanie serwera RESTAPI z wykorzystaniem NodeJS/ExpressJS. Do przechowywania danych danych możesz użyć 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

2 pkt zdefiniowanie i implementacja routingu i endpointów (uwzględniając konwencje RestAPI)

3 pkt - zdefiniowanie kontrolerów dla poszczególnych endpoints

Zadanie 3. Aplikacja do zdalnych konsultacji lekarskich – dynamiczny wybór źródła danych (1 pkt)

Zintegruj w swojej aplikacji wszystkie wersje realizacji backendu na które się zdecydowałeś. Do wyboru masz albo wersje z dwoma opcjami: (Lokalny Json, Firebase) lub trzema (Lokalny Json, Firebase, Własny serwer). Za pomocą np. opcji wyboru określ z którą wersja backendu chcesz współpracować. Odpowiednio zaprojektuj swoja aplikację tak aby można było dynamicznie zmieniać zródla danych.