Zaawansowane Programowanie Webowe

Laboratorium nr 7

Celem laboratorium jest przećwiczenie zagadnień związanych z obsługą rzeczywistych danych produkcyjnych pochodzących z serwera z danymi i udostępnianych za pomocą REST API.

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. Jest to baza typu RealTime gwarantującą aktualizacje naszego Frontendu w przypadku modyfikacji danych w bazie.

Dodatkowo Firebase będziemy mogli wykorzystać jako serwer autentykacji w przypadku logowania się do panelu administracyjnego.

Strona projektu to https://firebase.google.com/

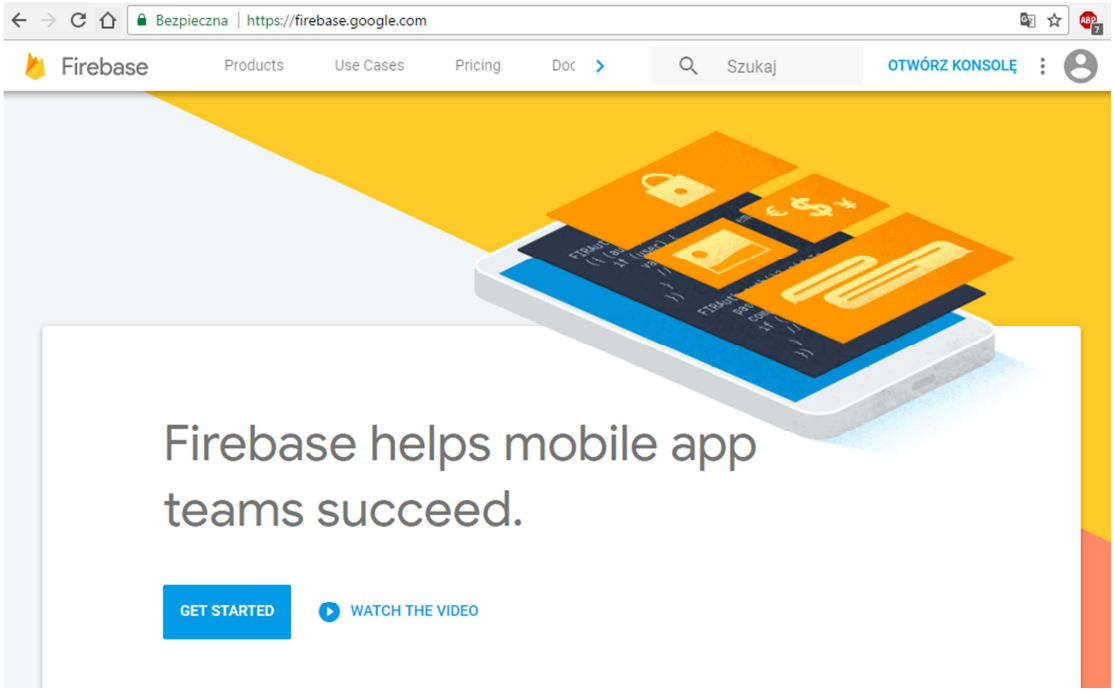
Przed rozpoczęciem pracy warto zapoznać się z dokumentacja projektowa https://firebase.google.com/docs/?authuser=0

Szczególnie interesująca będzie na nas sekcja:
https://firebase.google.com/docs/web/setup

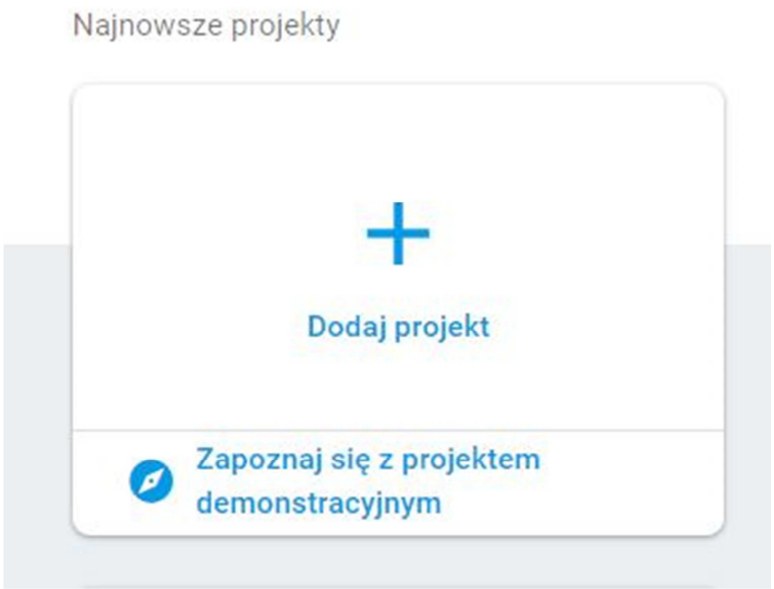
Firebase to tzw. **BaaS** (Backend as a Service), który umożliwia min. przechowywanie danych w formacie JSON oraz plików binarnych (np. jpg, mp4). Firebase dostarcza nam real-time database, gdzie komunikacja z serwerem jest oparta o Websockets, dzięki czemu po aktualizacji danych, klient automatycznie dostaje najświeższe dane. Obsługa jest banalnie prosta, w podstawowym zakresie można niemalże wszystko wygenerować z panelu użytkownika! Super rozwiązanie 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.

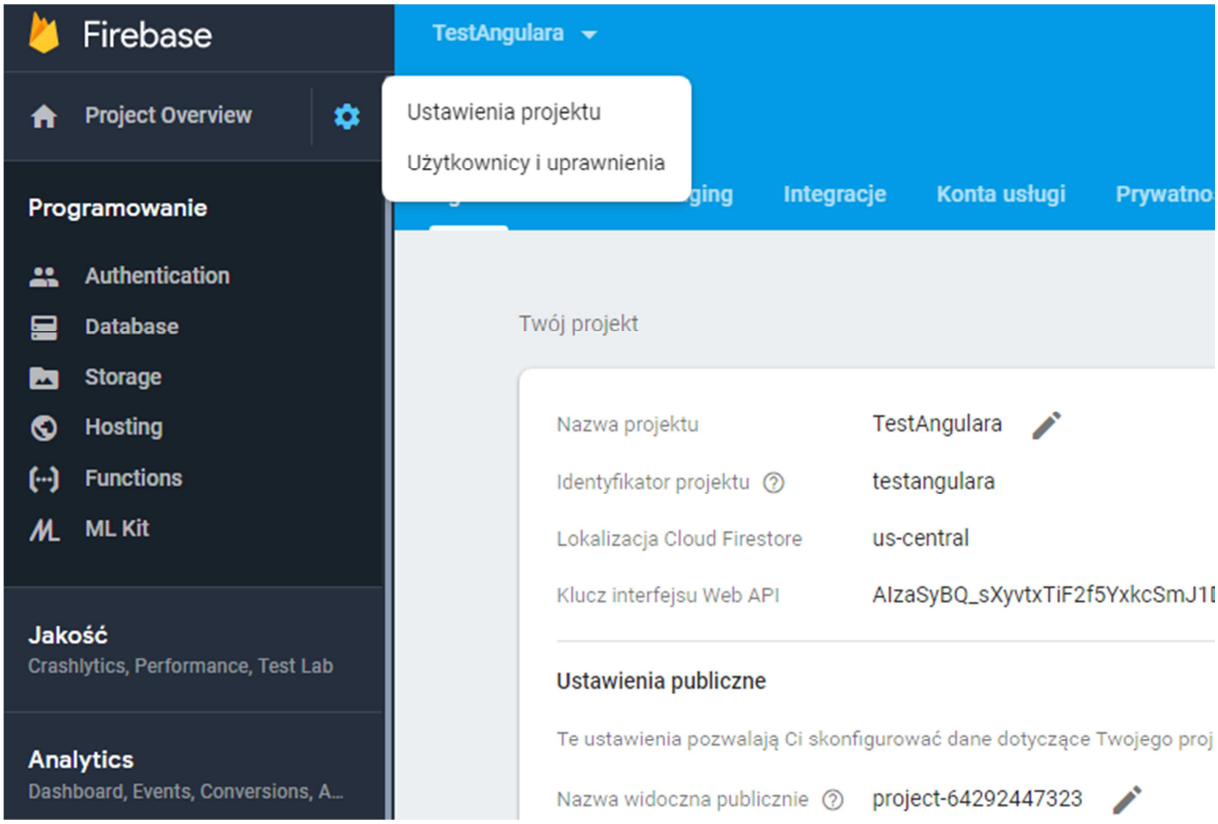
AngularFire

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

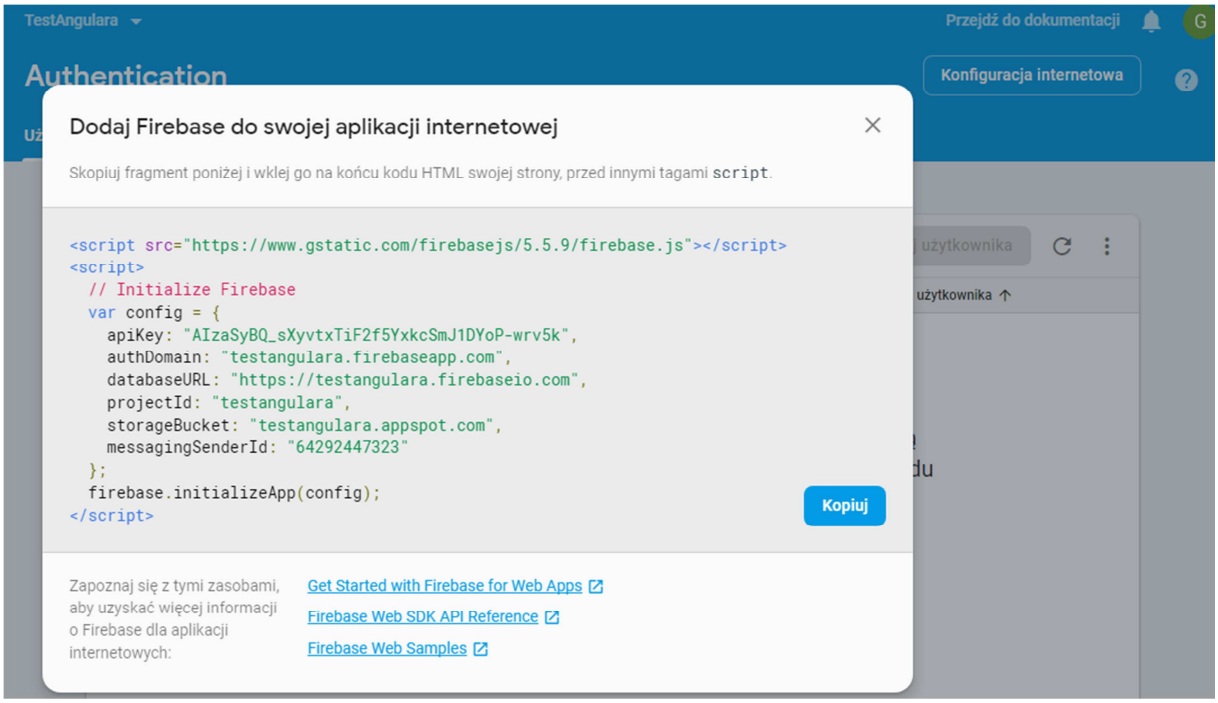
Rozpoczynamy od instalacji paczki:

```
npm install firebase angularfire2 --save
```

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

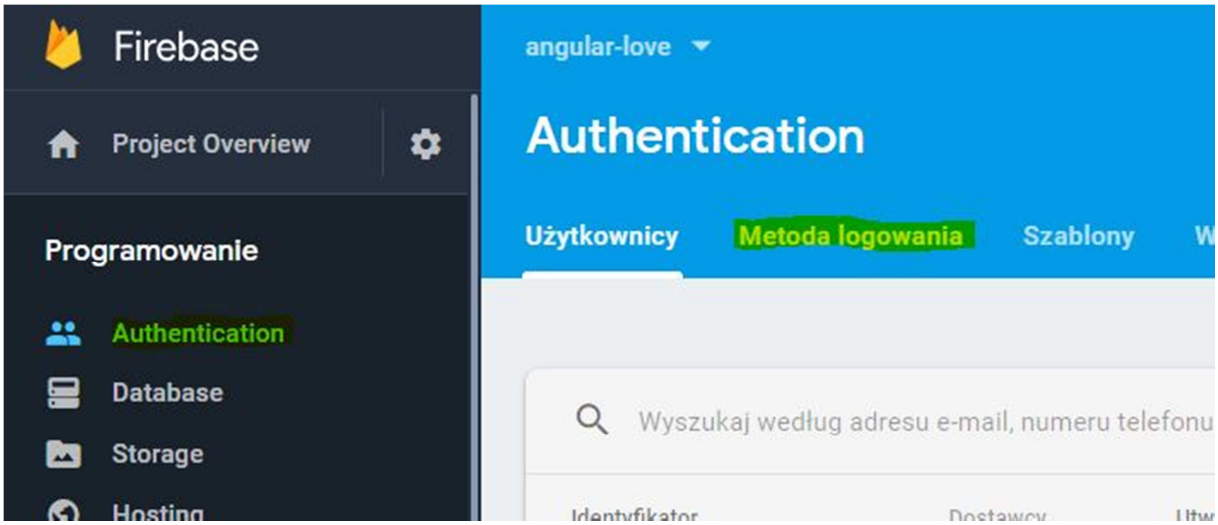


Scrollujemy nieco w dół i klikamy „Dodaj Firebase do swojej aplikacji internetowej". Pojawia się okienko z danymi projektu:

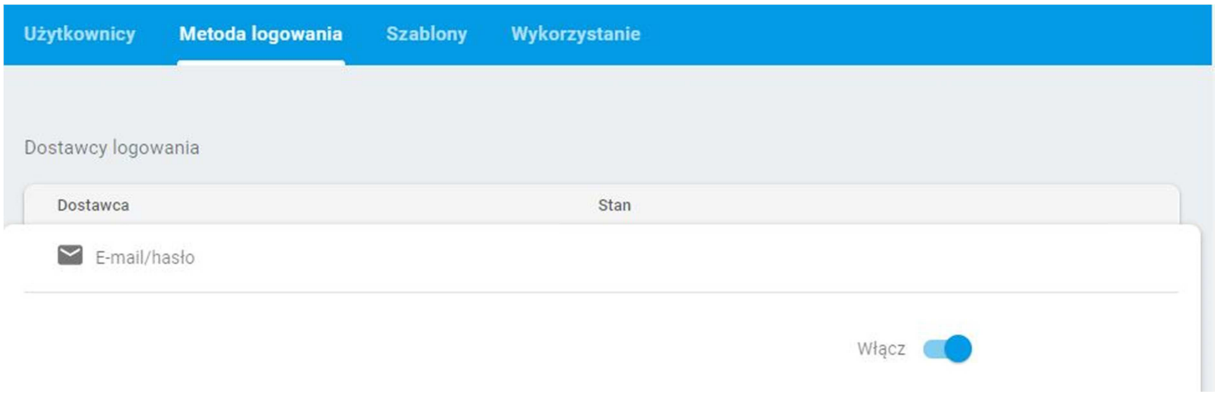


Kopiujemy sam obiekt przypisany do „var config” i zapisujemy np. w jakimś pliku tekstowym. Reszta nas nie interesuje.

Po utworzeniu projektu i wygenerowaniu danych dostępowych, w lewym menu przechodzimy do zakładki „AUTHENTICATION”, a następnie klikamy tab „METODA LOGOWANIA”:



Z listy klikamy na pierwszą pozycję – EMAIL / HASŁO i przestawiamy na „WŁĄCZ”:



Oczywiście możesz poeksperymentować z innymi metodami logowania. Zapisujemy ustawienia. Mamy już przygotowany serwer z możliwością przechowywania i rejestracji użytkowników

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 **(AngularFireAuthModule i AngularFireModule)**:

```
...
import { AngularFireAuthModule } from 'angularfire2/auth';
import { AngularFireModule } from 'angularfire2';
```

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

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.

AuthService

Stworzymy teraz usługę, która pozwoli nam rejestrować i logować użytkowników:

```
import { Injectable } from '@angular/core';
import { AngularFireAuth } from 'angularfire2/auth';
import { User } from 'firebase';
import { Observable } from 'rxjs/index';

export interface Credentials {
  email: string;
  password: string;
}

@Injectable({providedIn: 'root'})
export class AuthService {
  readonly authState$: Observable<User | null> = this.fireAuth.authState;

  constructor(private fireAuth: AngularFireAuth) {}

  get user(): User | null {
    return this.fireAuth.auth.currentUser;
  }

  login({email, password}: Credentials) {
    return this.fireAuth.auth.signInWithEmailAndPassword(email, password);
```

```
  }

  register({email, password}: Credentials) {
    return this.fireAuth.auth.createUserWithEmailAndPassword(email,
 password);
  }

  logout() {
    return this.fireAuth.auth.signOut();
  }
}
```

```
 readonly authState$: Observable<User | null> = this.fireAuth.authState;
```

AuthState jest strumieniem, który emituje zalogowanego użytkownika. Jeśli użytkownik zostaje wylogowany, strumień wyemituje null**.**

Obiekt użytkownika jest przechowuje również w polu currentUser:

```
 this.fireAuth.auth.currentUser;
```

Więcej na temat poszczególnych metod obiektu FireAuth należy poczytać w dokumentacji źródłowej.

Możemy manipulować stanem uwierzytelnienia w różnych scenariuszach, takich jak np. zamknięcie karty i powrót (wylogować użytkownika automatycznie czy może nie?). Otrzymujemy możliwość zmiany właściwości **PERSISTENCE**, w poniższy sposób:

```
 login({email, password}: Credentials) {
   const session = this.fireAuth.auth.Persistence.SESSION;
   return this.fireAuth.auth.setPersistence(session).then(() => {
     return this.fireAuth.auth.signInWithEmailAndPassword(email, password);
   });
 }
```

Możliwe 3 opcje do wyboru:

- **LOCAL (DOMYŚLNE)** – użytkownik nadal zostaje zalogowany po zamknięciu karty, trzeba jawnie użyć metody signOut aby wyczyścić stan zalogowania.

Przydatne, jeśli po zamknięciu karty i ponownym powrocie, chcemy użytkownikowi pozwolić pozostać zalogowanym.

- **SESSION** – stan zalogowanego użytkownika jest aktywny wyłącznie dla aktualnej sesji i zostanie wyczyszczony w przypadku zamknięcia okna/karty. Przydatne np. w aplikacjach, które są publicznie dostępne na komputerach, z których korzysta wielu użytkowników (np. w bibliotece).
- **NONE** – stan zalogowania jest przetrzymywany w pamięci i zostanie wyczyszczony po odświeżeniu okna.

Podgląd zarejestrowanych użytkowników

Po rejestracji, możemy zobaczyć założone konta w zakładce AUTHENTICATION:

Można również ręcznie dodać użytkownika, klikając „Dodaj użytkownika”.

AuthGuard

Obecnie w naszej aplikacji, gdy użytkownik odświeży stronę, to otrzymuje użytkownika z wartością null, gdyż currentUser będzie zaraz po odświeżeniu przechowywał wartość null, mimo, że stan uwierzytelnienia został zachowany.

Użytkownik może również przejść od razu na daną ścieżkę dowolnego widoku z pominięciem etapu logowania, oczywiście jeśli zna ścieżkę(za wiele mu to i tak nie da, bo bez zalogowania dane i tak nie zostanę pobrane).

Oba problemy może rozwiązać za pomocą guarda CanActivate. Podczas przejścia na widok inny niż login, sprawdzimy czy authState przechowuje stan użytkownika, jeśli tak, to go przepuścimy, natomiast jeśli wyemituje null, no to cofniemy go do widoku loginu:

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router }
from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from './auth.service';
import { map } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
```

```
  constructor(
    private authService: AuthService,
    private router: Router,
  ) {}

  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> {
    return this.authService.authState$.pipe(map(state => {
       if (state !== null) { return true; }

       this.router.navigate(['/login']);
       return false;
       }
     )
    );
  }
}
```

Guard **canActivate** zwraca strumień z true/false, w zależności czy **authState$** wyemitował stan użytkownika (emisja true) czy wyemitował null (emisja false). Strumień jest już obsługiwany przez mechanizm routingu w Angularze, także nie musimy się martwić o manualną subskrypcję w żadnym miejscu. Można oczywiście użytkownika przekierować np. na widok z informacją, że nie jest autoryzowany i prosimy o zalogowanie.

Teraz naszego guarda musimy nałożyć na odpowiednie ścieżki, np. na cały dashboard, który będzie sprawdzał również wszystkie ścieżki zawarte w „children”:

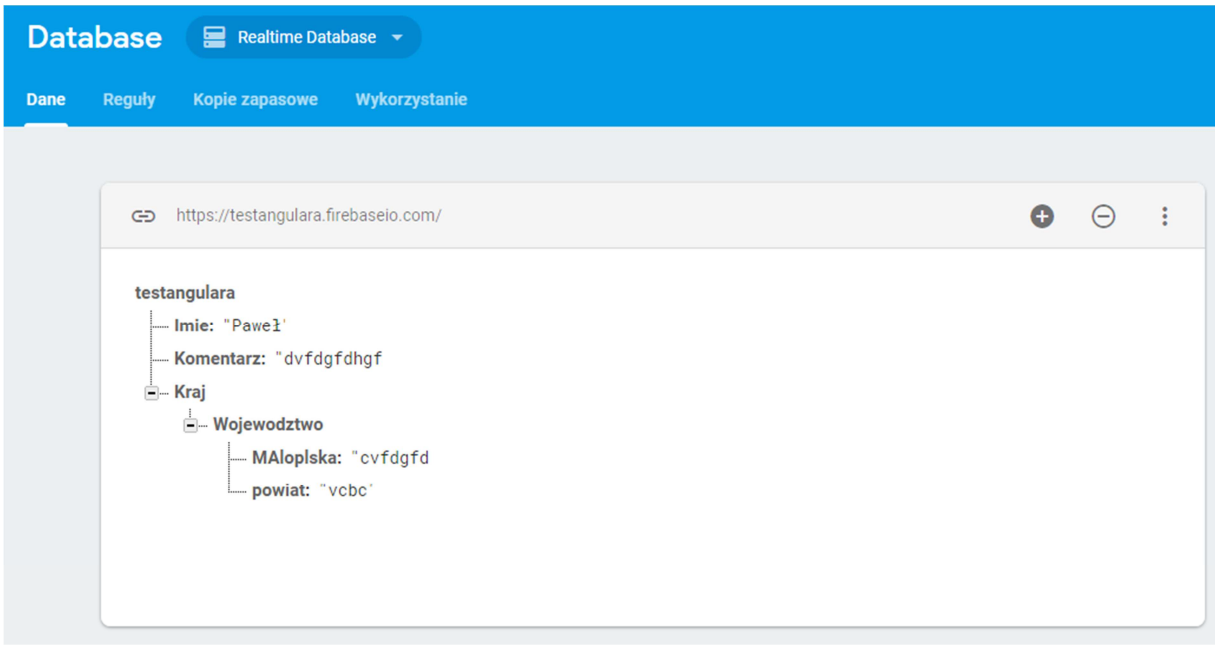
```
const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full'},
  { path: 'login', component: LoginComponent },
  {
    path: 'dashboard',
    component: DashboardComponent,
    canActivate: [AuthGuard],
    children: [...]
  },
];
```

Użytkownik jest sprawdzany na etapie przechodzenia między ścieżkami, oraz nie ma problemu, że użytkownik jest równy null na widoku po odświeżeniu aplikacji.

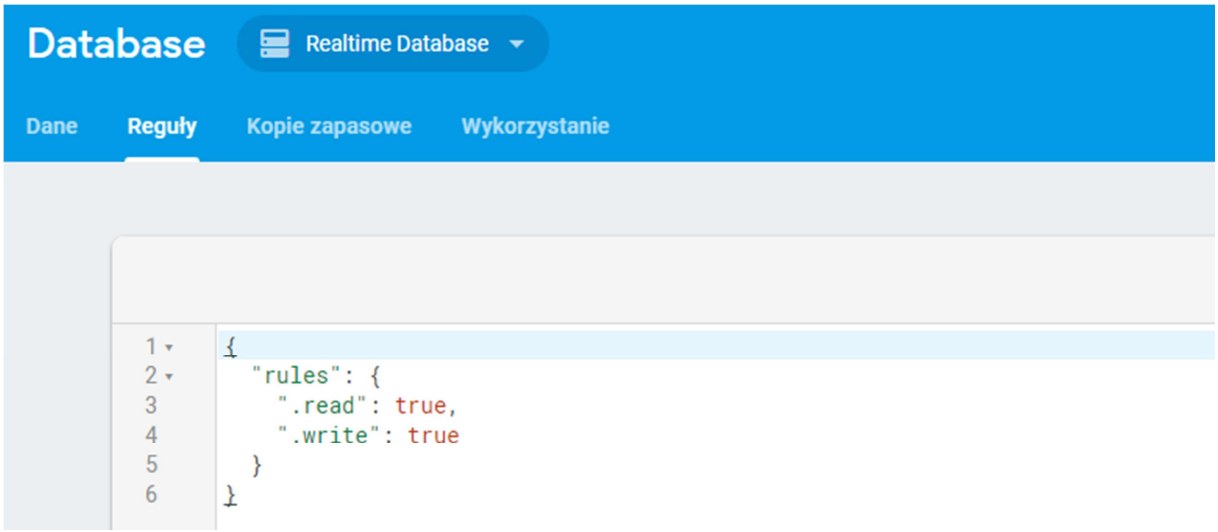
Obsługa bazy danych

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.



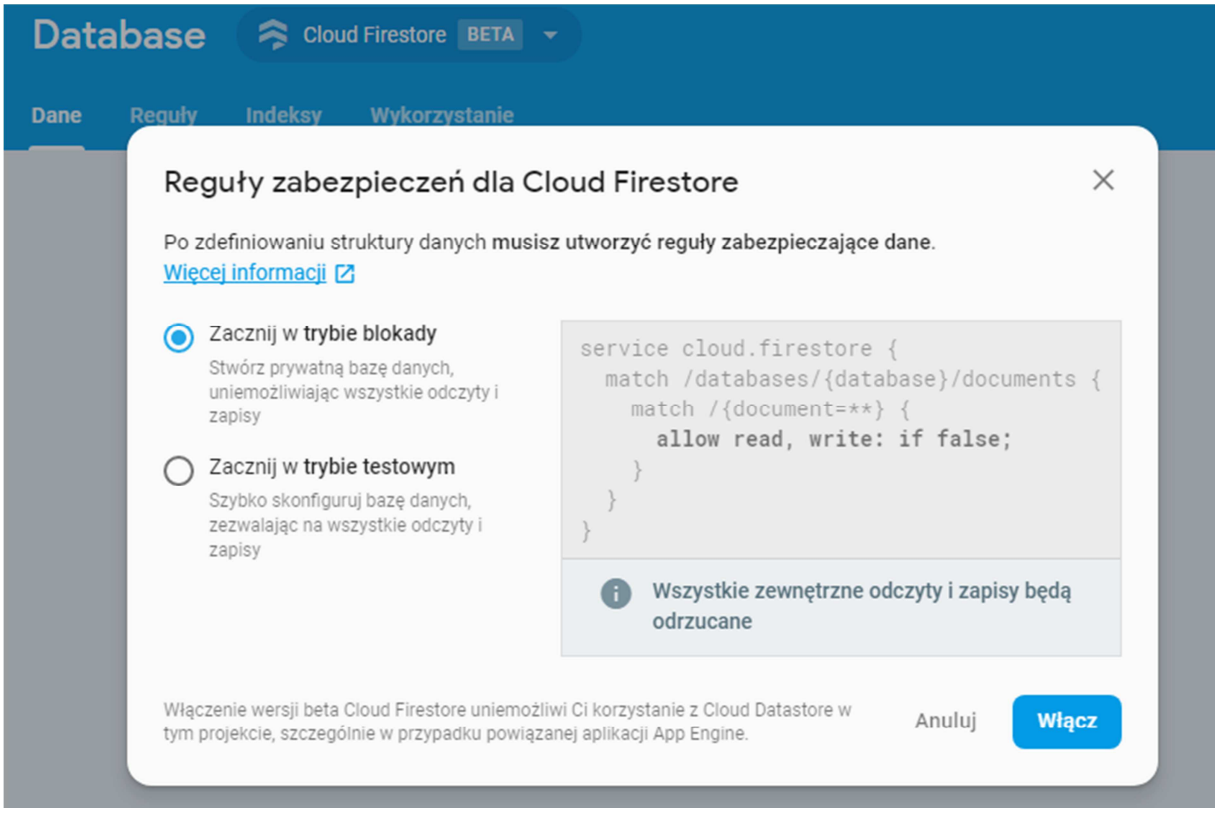
Nie wolno zapomnieć o ustawieni reguł dostępowych – domyślnie obie wartości są ustawione na false.



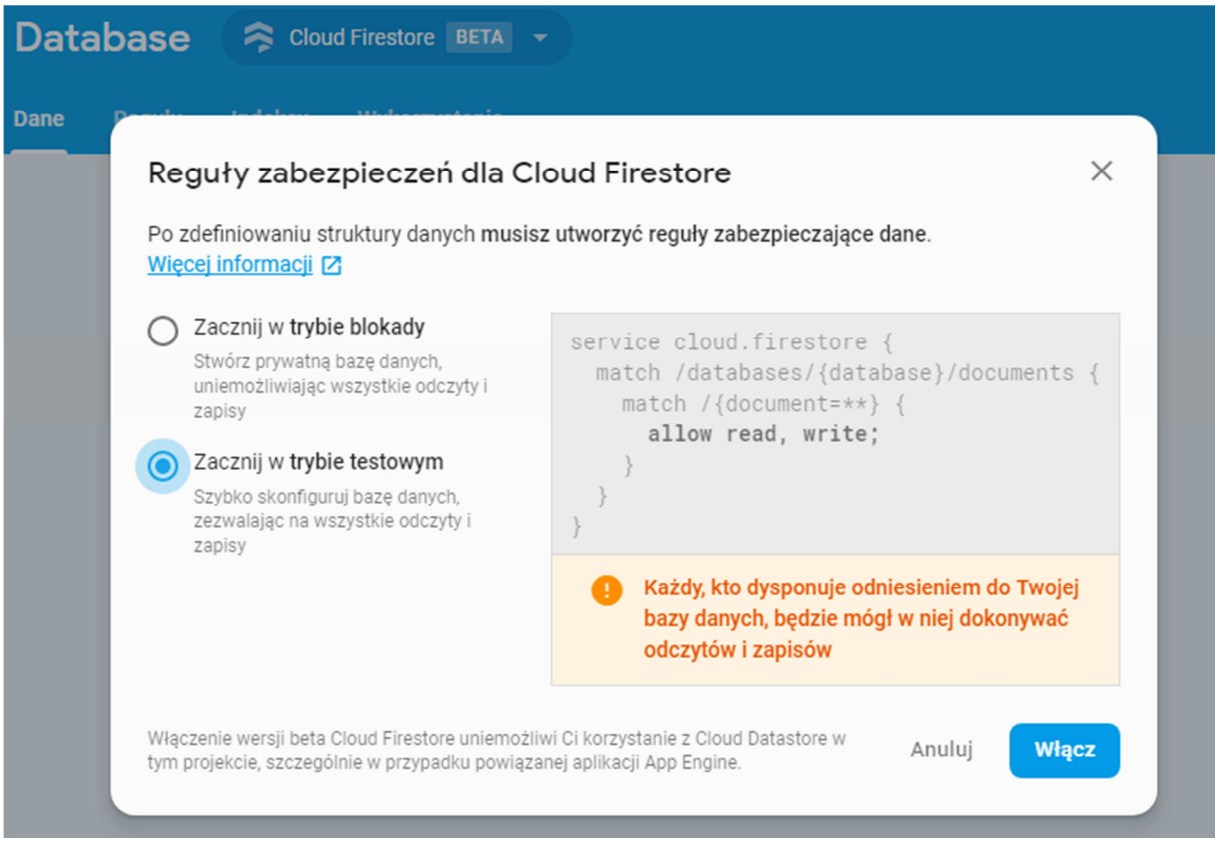
lub tak gdy zrezygnowaliśmy z autoryzacji

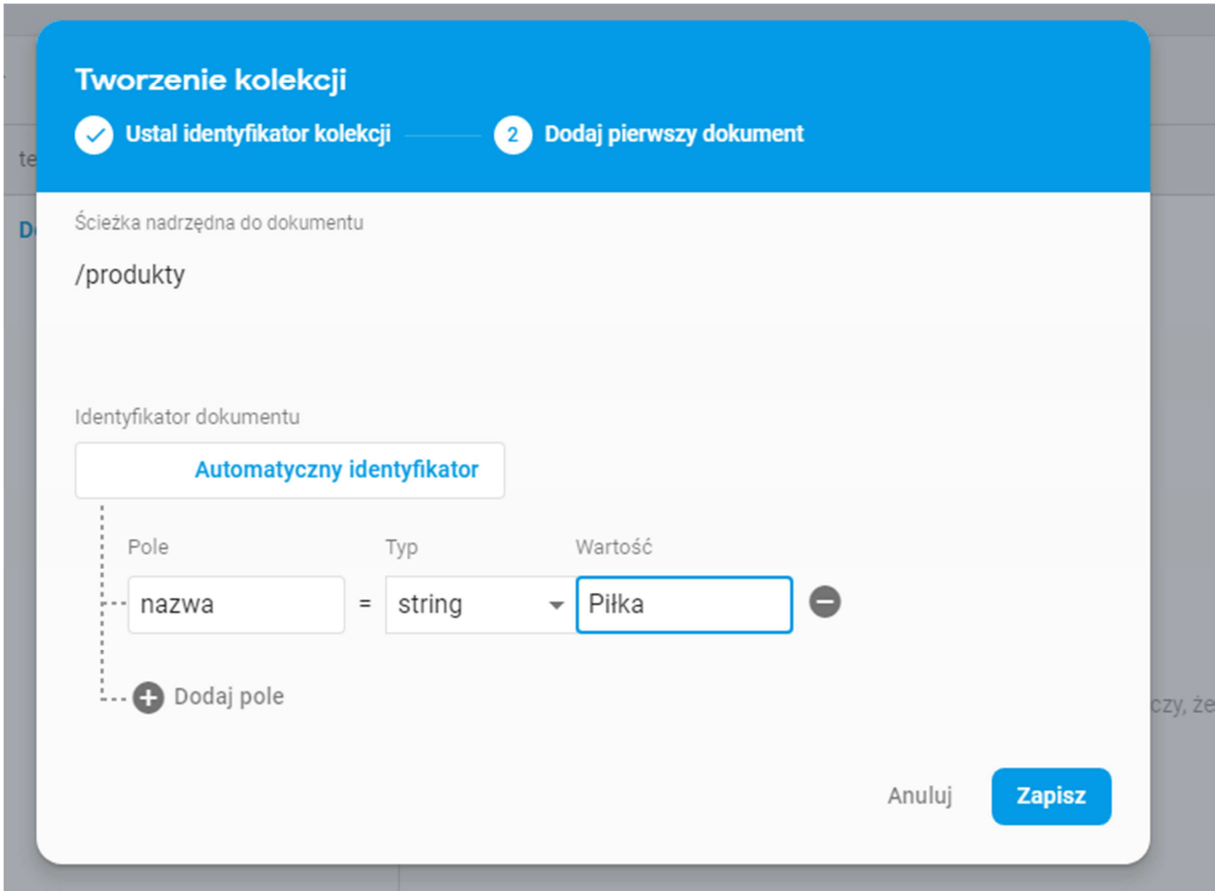
```
{
  "rules": {
    ".read": "auth == null",
    ".write": "auth == null"
  }
}
```

W przypadku wybrania wersji Cloud



Lub


Database
Cloud Firestore BETA
Dane
Reguły zabezpieczeń dla Cloud Firestore
Po zdefiniowaniu struktury danych musisz utworzyć reguły zabezpieczające dane.
Więcej informacji
Zacznij w trybie blokady
Stwórz prywatną bazę danych, uniemożliwiając wszystkie odczyty i zapisy
Zacznij w trybie testowym
Szybko skonfiguruj bazę danych, zezwalając na wszystkie odczyty i zapisy
service cloud.firestore {
match /databases/{database}/documents {
match /{document=**} {
allow read, write;
}
}
}
Każdy, kto dysponuje odniesieniem do Twojej bazy danych, będzie mógł w niej dokonywać odczytów i zapisów
Włączenie wersji beta Cloud Firestore uniemożliwi Ci korzystanie z Cloud Datastore w tym projekcie, szczególnie w przypadku powiązanej aplikacji App Engine.
Anuluj
Włącz


Tworzenie kolekcji
Ustal identyfikator kolekcji
2
Dodaj pierwszy dokument
Ścieżka nadrzędna do dokumentu
/produkty
Identyfikator dokumentu
Automatyczny identyfikator
Pole
Typ
Wartość
nazwa
=
string
Piłka
Dodaj pole
Anuluj
Zapisz

Po stworzeniu bazy w Firebase przechodzimy do Angulara.

W zależności od użytej bazy :

W module app.module.ts importujemy AngularFireDatabaseModule dla Real Time DataBase

```
...
import { AngularFireDatabaseModule } from 'angularfire2/database';
…

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

Lub dla Cloud FireStore

```
import { AngularFirestoreModule } from 'angularfire2/firestore';


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

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

Przykładowa implementacja

```
export class AppComponent {
    public data: FirebaseListObservable<any[]>;

    constructor(private db: AngularFireDatabase) {
    }
    ngOnInit() {
      this.data = this.db.list('/test');
    }


    // odczyt danych z bazy
    public getdata(listPath): Observable<any[]> {
        return this.db.list(listPath).valueChanges();
    }

}
```

Dodanie nowego wpisu w baize polega w zasadzie tylko na dodaniu obiektu do instancji AngularFireBaseList

Np.

```
adddata(value: string): void {

  this.data.push({ content: value, done: false });

}
```

Edycja bazy danych:

```
updatedata(cos: any): void {

  this.db.object('/test/' + cos.$key)

    .update({  //Jason // });

}
```

Usunięcie obiektu z bazy danych:

```
deletedata(cos: any): void {

  this.db.object('/test/' + cos.$key).remove();

}
```

Powróćmy teraz do kocowej specyfikacji projektu Sklep Internetowy. Została nam do omówienia panel administracyjny.

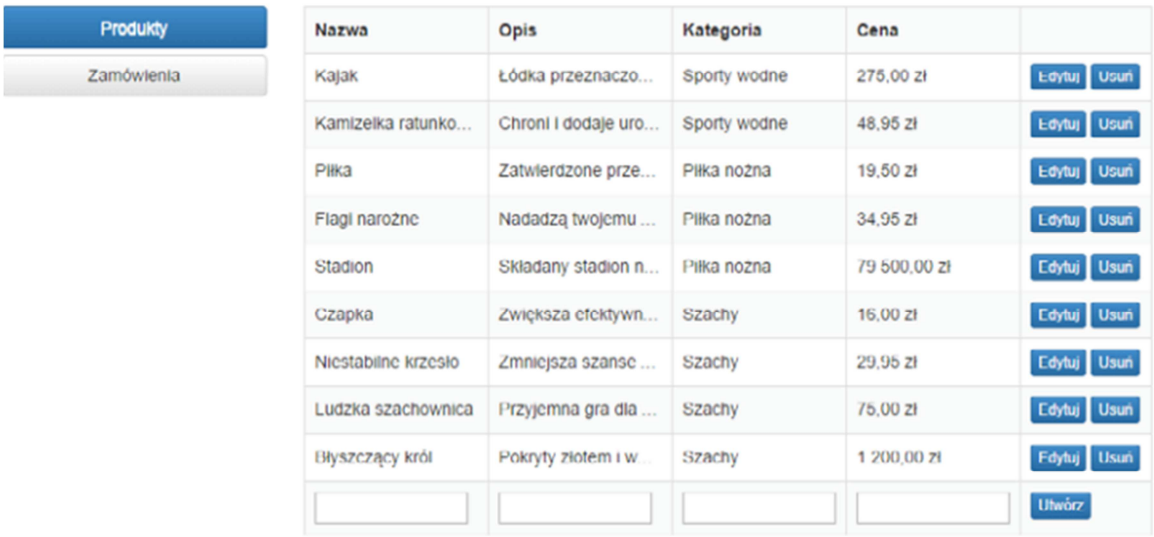
Dostęp do niego wymaga autentykacji. Jedna z możliwych rozwiązań to użycie Firebase jako serwera autentykacji. Alternatywa to GWT lub autentykacja oparta o protokół OAuth2.

Wyróżniamy dwie role: admina oraz pracownika zajmującego się realizacja zamówień.

Admin może wszystko, pracownik obsługuje tylko sekcje obsługi zamówień nie powinien mieć dostępu do sekcji zarzadzania produktami.

Sekcja zarzadzania produktami umożliwia przeglądanie asortymentu sklepu. Przy każdej pozycji można dokonać jej aktualizacji, usunąć lub zdefiniować promocje. Dodatkowo istnieje sekcja dodawania nowych produktów pozwalająca na wprowadzenie nowego produktu.

Przykładowa wersja (bardzo trywialna) panelu produktów:



Można (a nawet powinno się ją rozszerzyć o możliwość filtrowania oraz sortowania po rożnych kolumnach).

Sekcja dodawania promocji powinna umożliwić wskazanie produktów do promocji o ile procent będzie promocja oraz jak długo od zatwierdzenia promocja jest ważna (minimum 2 minuty – max 3 dni). Produkty objęte promocja powinny być odpowiednio odznaczone w panelu produktów.

Zamówienia

W tej zakładce mamy listę aktualnych zamówień w postaci danych teleadresowych oraz kwoty zamówienia. Wybór zamówienia powinien pozwolić na wejście do szczegółów zamówienia.
Pracownik powinien mieć możliwość kompletacji zamówienia zaznaczając które produkty zostały już przygotowane do wysyłki. Gdy wszystkie zostały zaznaczone zamówienie jest gotowe do wysyłki i jest możliwości zamiany jego statusu jako zrealizowane. Jest oczywiście możliwość zaznaczenie wszystkich produktów na raz.
W panelu tym można oglądać zamówienia po statusach: oddzielnie zrealizowane, oddzielnie w trakcie realizacji, oraz oczekujące. Przy zamówieniach zrealizowanych zapisujemy datę i czas wysyłki (czas zmiany statusu zamówienia na zrealizowany)