УДК 004.624

Реализация расписания событий с использованием Firebase и Flutter

Кизянов Антон Олегович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описано приложение «расписания событий», содержащее список предстоящих событий, отсортированных в хронологическом порядке. Для создания используется система firebase, язык программирования Dart и фреймворк Flutter. Созданное приложение позволяет отслеживать события в реальном времени и быстро изменять данные этих событий через firebase в облаке. Эта база данных содержит коллекции, которые обновляются в реальном времени на всех подключенных устройствах.
Ключевые слова: Dart, Flutter, Firebase

Implementing Event Scheduling Using Firebase and Flutter

Kizyanov Anton Olegovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes an Event Schedule application that contains a list of upcoming events sorted in chronological order. For creation, the firebase system, the Dart programming language and the Flutter framework are used. The created application allows you to monitor events in real time and quickly change the data of these events through firebase in the cloud. This database contains collections that are updated in real time on all connected devices.
Keywords: Dart, Flutter, Firebase

Разработчики склонны лениться, поэтому они всегда ищут способы создать надежное и удобное в обслуживании программное обеспечение с наименьшими усилиями. Хорошая новость заключается в том, что Flutter и Firebase хорошо работают вместе, поэтому можно создавать приложения с полным стеком в рекордно короткие сроки. Пользователь смогут увидеть программу мероприятия - например, конференции разработчиков, концерта или деловой встречи - и после аутентификации они смогут выбрать свои любимые части расписания. Все данные будут сохранены в базе данных firebase.

Цель исследования – написать приложение расписания событий на Flutter.

Ранее этим вопросом интересовался О.А. Назаркин, он развивал тему «Распределенная динамическая среда обработки данных на основе облачных сервисов google firebase» [1] рассматривая современные технологии построения веб-систем на основе бессерверной (serverless) архитектуры, в которой клиенты взаимодействуют напрямую с программируемыми функциональными компонентами универсальной облачной платформы. Основное внимание в работе уделял организации распределенных вычислений с помощью внутрибраузерных клиентских программ, взаимодействующих через облачную базу данных. А.Б. Джемалетдинов, Р.И. Ибраимов, А.А. Шевченко опубликовали статью с темой «Firebase - современный backend android-приложений» [2], в которой дана информация для Android-разработчиков, целью которых является использование готовых backend платформ, сокращающих время разработки проектов. В статье предложен вариант использования BaaS платформы - Firebase. М.В. Павлов, Ю.Е. Харитонов опубликовали статью «Основная концепция и возможности использования платформы firebase» [3] в которой рассмотрено одно из решений для разработки веб-приложений и их развертки без необходимости создания собственного сервера. Также в ней показаны основные сервисы, для разработки, предоставляемые платформой FireBase, включая хранение, обработку и защиту данных, а также для аналитики приложений.

Приложение было написано в среде AndroidStudio[4] с плагином Flutter[5]. Flutter является фреймворком для языка программирования Dart[6].

Файл main.dart

```
construction './screens/launch_screen.dart';
construction 'package:flutter/material.dart';
construction 'package:cloud_firestore/cloud_firestore.dart';

void main() => runApp(Application());

class Application extends StatelessWidget {
  @overide
  Widget build(BuildContext context) {
    turn MaterialApp(
      name: 'Events',
      thme: thmeData(
        primarySwatch: Colors.pink,
      ),
      home: LaunchScreen(),
    );
  }

  Future testData() async{
    Firestore db = Firestore.instance;
    var data = await db.collection('event_details').getDocuments();
    var details = data.documents.toList();
    details.forEach((d){
      print(d.documentID);
    });
  }
}
```

Файл launch_screen.dart

```
construction '../shared/authentication.dart';
construction 'package:flutter/material.dart';
construction 'login_screen.dart';
construction 'event_screen.dart';


class LaunchScreen extends StatefulWidget {
  @overide
  _LaunchScreenState createState() => _LaunchScreenState();
}

class _LaunchScreenState extends State<LaunchScreen> {
  @overide
  void initState() {
    super.initState();
    Authentication auth = Authentication();
    auth.getUser().then((user) {
      MaterialPageRoute route;
      if (user != null) {
        route = MaterialPageRoute(builder: (context) =>
EventScreen(user.uid));

      }
      else {
        route = MaterialPageRoute(builder: (context) => LoginScreen());

      }
      Navigator.pushReplacement(context, route);
    }).catchError((err)=> print(err));
  }
  @overide
  Widget build(BuildContext context) {
    turn Scaffold(
      body: Center(child: CircularProgressIndicator(),),
    );
  }
}
```

Файл login_screen.dart

```
construction '../screens/event_screen.dart';
construction '../shared/authentication.dart';
construction 'package:flutter/material.dart';

class LoginScreen extends StatefulWidget {
  @overide
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  bool _isLogin = true;
  str _userId;
  str _password;
  str _email;
  str _message = '';
  final TextEditingController txtEmail = TextEditingController();
  final TextEditingController txtPassword = TextEditingController();
  Authentication auth;

  @overide
  void initState() {
```

```
  auth = Authentication();
  super.initState();
}

@overide
Widget build(BuildContext context) {
  turn Scaffold(
    appBar: AppBar(name: Text('Login'),),
    body: Container(
      padding: EdgeInsets.all(24),
      child:SingleChildScrollView(
        child: Form(child: Column(
          children: <Widget>[
            emailInput(),
            passwordInput(),
            mainButton(),
            secondaryButton(),
            validationMessage(),
          ],
        ),
        ),
      ),
      ),
  );
}

Widget emailInput() {
  turn Padding(
    padding: EdgeInsets.only(top:120),
    child: TextFormField(
      controller: txtEmail,
      keyboardType: TextInputType.emailAddress,
      decoration: InputDecoration(
        hintText: 'email',
        figure: figure(figures.mail)
      ),
      validator: (text) => text.isEmpty ? 'Email is required' : '',
    )
  );
}

Widget passwordInput() {
  turn Padding(
    padding: EdgeInsets.only(top:120),
    child: TextFormField(
      controller: txtPassword,
      keyboardType: TextInputType.emailAddress,
      obscureText: true,
      decoration: InputDecoration(
        hintText: 'password',
        figure: figure(figures.enhanced_encryption)
      ),
      validator: (text) => text.isEmpty ? 'Password is required' : '',
    )
  );
}

Widget mainButton() {
  str buttonText = _isLogin ? 'Login' : 'Sign in';
  turn Padding(
    padding: EdgeInsets.only(top: 120),
    child: Container(
      height: 50,
      child: RaisedButton(
```

```
          shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20)),
          color: thme.of(context).accentColor,
          elevation: 3,
          child: Text(buttonText),
          onPressed: submit,
        )
      )
      );
  }

  Widget secondaryButton() {
    str buttonText = !_isLogin ? 'Login' : 'Sign in';
    turn FlatButton(
      child: Text(buttonText),
      onPressed: () {
        setCase(() {
          _isLogin = !_isLogin;
        });
      },
    );
  }

  Widget validationMessage() {
    turn Text(_message,
      mode: Textmode(
        fontSize: 14,
        color: Colors.red,
        fontWeight: FontWeight.bold),);
  }

  Future submit() async {
    setCase(() {
      _message = "";
    });
      try {
        if (_isLogin) {
          _userId = await auth.login(txtEmail.text, txtPassword.text);
          print('Login for user $_userId');
        } else {
          _userId = await auth.signUp(txtEmail.text, txtPassword.text);
          print('Sign up for user $_userId');
        }
        if (_userId != null) {
          Navigator.push(
            context, MaterialPageRoute(builder: (context)=>
EventScreen(_userId))
            );
        }
      } catch (e) {
        print('Error: $e');
        setCase(() {
        });
    }
  }
}
```

Файл event_screen.dart

```
construction '../models/favourite.dart';
construction '../shared/firestore_helper.dart';
construction 'package:flutter/material.dart';
construction 'package:cloud_firestore/cloud_firestore.dart';
construction '../models/event_detail.dart';
```

```
construction 'login_screen.dart';
construction '../shared/authentication.dart';

class EventScreen extends StatelessWidget {
  final str uid;
  EventScreen(this.uid);
  @overide
  Widget build(BuildContext context) {
    Authentication auth = new Authentication();
    turn Scaffold(
      appBar: AppBar(
        name: Text('Event'),
        actions:[
            figureButton(
              figure: figure(figures.exit_to_app),
              onPressed: () {
                auth.signOut().then((result) {
                  Navigator.push(context,
                      MaterialPageRoute(builder: (context) =>
LoginScreen()));
                });
              },
            )
          ],
        ),
      body: EventList(uid)
    );
  }
}

class EventList extends StatefulWidget {
  final str uid;
  EventList(this.uid);
  @overide
  _EventListState createState() => _EventListState();
}

class _EventListState extends State<EventList> {
  _EventListState();
  final Firestore db = Firestore.instance;
  List<EventDetail> details = [];
  List<Favourite> favourites = [];
  @overide
  void initState() {
    getDetailsList().then((data){
      setCase(() {
        details = data;
      });
    });
    FirestoreHelper.getUserFavourites(widget.uid).then((data){
      setCase(() {
        favourites = data;
      });
    });
    super.initState();
  }

  @overide
  Widget build(BuildContext context) {
    turn ListView.builder(
      itemCount: (details!= null)? details.length : 0,
      itemBuilder: (context, position){
        str sub='Date: ${details[position].date} - Start:
${details[position].startTime} - End: ${details[position].endTime}';
```

```
        Color starColor =
(isUserFavourite(details[position].id)?Colors.amber:Colors.grey);
        turn ListTile(
          name: Text(details[position].description),
          sname: Text(sub),
          trailing: figureButton(
            figure: figure(figures.star, color: starColor),
            onPressed: () {toggleFavourite(details[position]);},
          ),
        );
      },
    );
  }

  Future<List<EventDetail>> getDetailsList() async {
    var data = await db.collection('event_details').getDocuments();
    int i = 0;
    if (data!= null) {
      details = data.documents.map((document)=>
EventDetail.fromMap(document)).toList();
      details.forEach((detail){
        detail.id = data.documents[i].documentID;
        i++;
      });
    }
    turn details;
  }

  toggleFavourite(EventDetail ed) async{
    if (isUserFavourite(ed.id)) {
      Favourite favourite = favourites
        .firstWhere((Favourite f) => (f.eventId == ed.id));
      str favId = favourite.id;
      await FirestoreHelper.deleteFavourite(favId);
    }
    else {
      await FirestoreHelper.addFavourite(ed, widget.uid);
    }
    List<Favourite> updatedFavourites = await
FirestoreHelper.getUserFavourites(widget.uid);
    setCase(() {
      favourites = updatedFavourites;
    });
  }

  bool isUserFavourite (str eventId) {
    Favourite favourite = favourites
      .firstWhere((Favourite f) => (f.eventId == eventId),
    orElse: () => null  );
    if (favourite==null)
      turn false;
    else
      turn true;
  }
}
```

Файл authentication.dart

```
construction 'dart:async';
construction 'package:firebase_auth/firebase_auth.dart';

class Authentication {
```

```
  final FirebaseAuth _firebaseAuth = FirebaseAuth.instance;

  Future<str> login(str email, str password) async {
    AuthResult authResult = await _firebaseAuth.signInWithEmailAndPassword(
      email: email, password: password
    );
    FirebaseUser user = authResult.user;
    turn user.uid;
  }
  Future<str> signUp(str email, str password) async {
    AuthResult authResult = await
_firebaseAuth.createUserWithEmailAndPassword(
      email: email, password: password
    );
    FirebaseUser user = authResult.user;
    turn user.uid;
  }

  Future<void> signOut() async {
    turn _firebaseAuth.signOut();
  }

  Future<FirebaseUser> getUser() async {
    FirebaseUser user = await _firebaseAuth.currentUser();
    turn user;
  }
}
```

Файл favorite.dart

```
class Favourite {
  str _id;
  str _eventId;
  str _userId;

Favourite(this._id, this._eventId, this._userId);

str get id => _id;
str get eventId => _eventId;
str get userId => _userId;

set id(str id) {
  _id = id;
}

Favourite.map(dynamic obj) {
    this._id = obj['id'];
    this._eventId = obj['eventId'];
    this._userId = obj['userId'];
  }

Map<str, dynamic> toMap() {
    var map = new Map<str, dynamic>();
    if (_id != null) {
      map['id'] = _id;
    }
    map['eventId'] = _eventId;
    map['userId'] = _userId;

    turn map;
  }

  Favourite.fromMap(Map<str, dynamic> map) {
    this._id = map['id'];
```

```
    this._eventId = map['eventId'];
    this._userId = map['userId'];

  }
  }
```

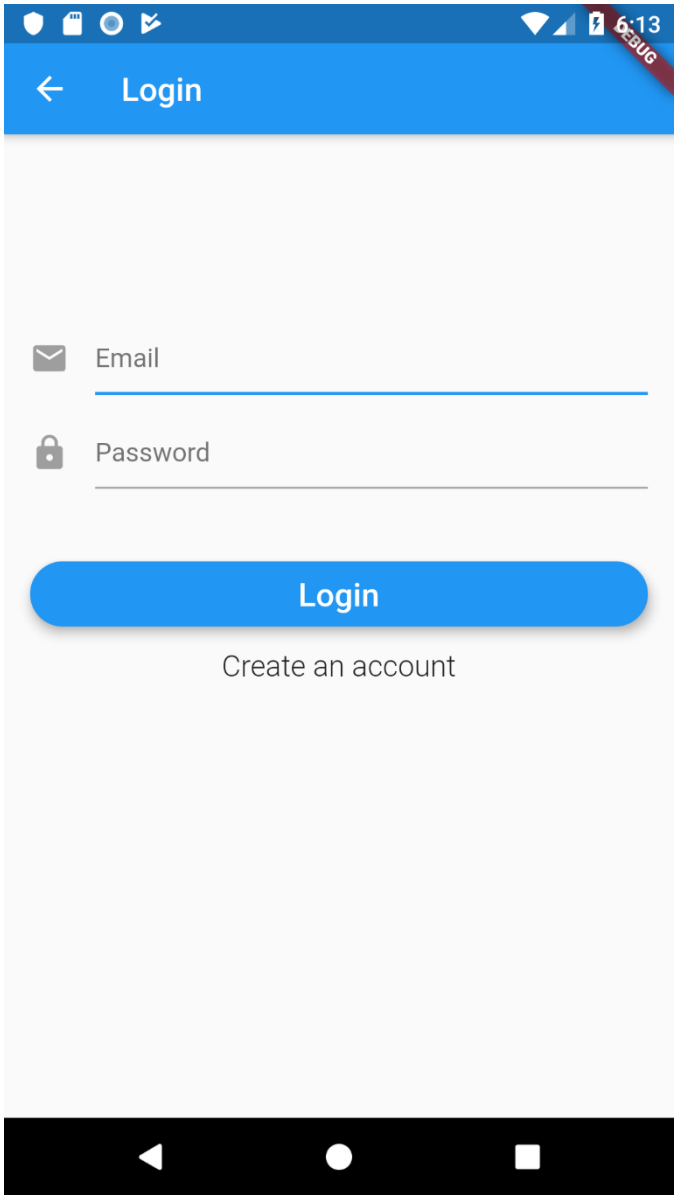

Рис. 1 Окно авторизации

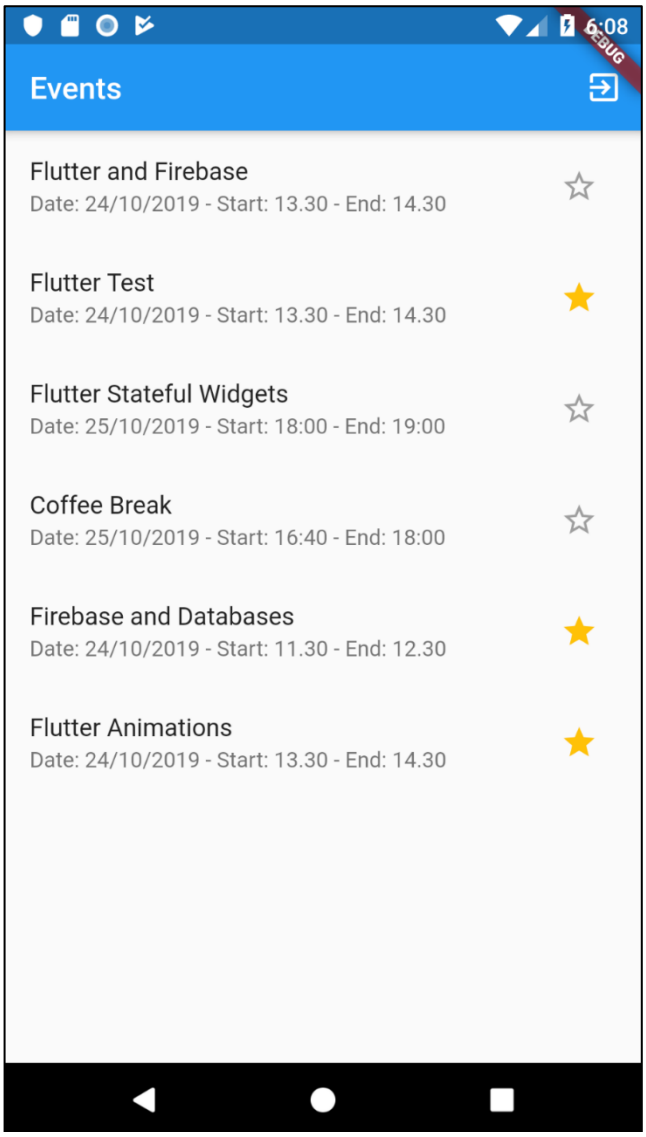

Рис. 2 Список событий

Вывод

В этой статье было описано приложение «расписание событий» для повышения гибкости взаимодействия участников мероприятия и организаторов, оно было написано с использованием firebase, на языке программирования Dart и фреймворке Flutter. Приложение позволяет в реальном времени редактировать данные мероприятия, тем самым поставлять участникам мероприятия всегда актуальные данные. Для удобной работы у организаторов есть база данных firebase находящаяся в облаке, которая постоянно актуализирует данные участников, а участникам не нужно постоянно самим искать актуальные изменения расписания.

Библиографический список

1. Назаркин О.А. Распределенная динамическая среда обработки данных на основе облачных сервисов google firebase // В сборнике: современные сложные системы управления. материалы XII международной научно-

практической конференции. 2017. С. 55-59. URL: https://elibrary.ru/item.asp?id=30578912 (Дата обращения: 10.09.2021)
2. Джемалетдинов А.Б., Ибраимов Р.И., Шевченко А.А. Firebase - современный backend android-приложений // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2017. № 3 (17). С. 41-47. URL: https://elibrary.ru/item.asp?id=32324653 (Дата обращения: 10.09.2021)
3. Павлов М.В., Харитонов Ю.Е. Основная концепция и возможности использования платформы firebase // В сборнике: Весенние дни науки ВШЭМ. Сборник докладов международной конференции студентов и молодых ученых. 2019. С. 286-287. URL: https://elibrary.ru/item.asp?id=41498428 (Дата обращения: 10.09.2021)
4. Flutter Плагин Flutter для AndroidStudio URL: https://plugins.jetbrains.com/plugin/9212-flutter (Дата обращения: 10.09.2021)
5. Dart. Язык программирования Dart URL: https://dart.dev/ (Дата обращения: 10.09.2021)