| | |
|---|---|
| ДИСЦИПЛИНА | Программирование корпоративных систем |
| ИНСТИТУТ | Институт перспективных технологий и индустриального программирования |
| КАФЕДРА | Кафедра индустриального программирования |
| ВИД УЧЕБНОГО МАТЕРИАЛА | Практические задание |
| ПРЕПОДАВАТЕЛЬ | Адышкин Сергей Сергеевич |
| СЕМЕСТР | 5 семестр, 2025-2026 гг. |

## Практическое занятие № 8

## Работа с базами данных. Подключение приложения к Firebase

### Цели занятия

- Подключить Flutter-приложение к Firebase через FlutterFire CLI.
- Освоить инициализацию `firebase_core` и работу с Cloud Firestore (`cloud_firestore`).
- Реализовать базовый CRUD (создание, чтение в реальном времени, обновление, удаление) для коллекции данных.
- Настроить минимальные правила безопасности Firestore для учебной среды.
- Сформировать практические навыки диагностики и устранения типовых ошибок подключения.

## Теоретическая часть

### Введение в Firebase и облачные базы данных

Firebase — это облачная платформа разработки мобильных и веб-приложений, принадлежащая компании **Google**, предоставляющая комплекс готовых сервисов для хранения данных, аутентификации, аналитики, уведомлений и хостинга.
Главная идея Firebase — **ускорить процесс создания приложений**, снимая с разработчика необходимость развертывать и администрировать серверную инфраструктуру.

Firebase представляет собой **Backend-as-a-Service (BaaS)** — то есть готовую серверную часть, доступную через SDK и REST API. Приложение разработчика взаимодействует с Firebase напрямую, минуя собственный сервер, что удобно для прототипов, учебных проектов и небольших коммерческих решений.

### Архитектура Firebase

Платформа Firebase состоит из модульных компонентов, которые можно подключать по мере необходимости:

1. **Firebase Authentication** — система аутентификации пользователей (email, Google, Apple, GitHub и др.).
2. **Cloud Firestore** — документо-ориентированная база данных нового поколения с поддержкой офлайн-доступа и синхронизации в реальном времени.
3. **Realtime Database** — древняя (до Firestore) JSON-база с мгновенной синхронизацией, но ограниченными возможностями запросов.
4. **Cloud Storage** — файловое хранилище (для изображений, документов, аудио и видео).
5. **Cloud Functions** — серверные функции на Node.js, вызываемые по событиям.
6. **Firebase Hosting** — хостинг для SPA и PWA-приложений.

7. **Firebase Analytics / Crashlytics** — аналитика и сбор ошибок.

Firebase тесно интегрирован с **Google Cloud Platform (GCP)**, что позволяет масштабировать приложение до промышленного уровня.

**Firebase в мобильной разработке**

Для мобильных приложений Firebase особенно популярен благодаря простоте подключения, наличию SDK для **Flutter, Android, iOS**, а также богатому визуальному интерфейсу в консоли. Платформа поддерживает:

- **реактивную модель данных** — приложение автоматически обновляется при изменении базы;
- **офлайн-доступ** — данные кэшируются и синхронизируются при восстановлении сети;
- **безопасность на уровне документа** — гибкие правила доступа;
- **кроссплатформенность** — единый код Flutter работает на Android, iOS, Web и Desktop.

Firebase особенно полезен для учебных и MVP-проектов, где нет необходимости писать сервер на Go, Node.js или Python — вся бизнес-логика может выполняться через Firestore и Cloud Functions.

**Cloud Firestore: особенности и модель данных**

**Cloud Firestore** — это **NoSQL документо-ориентированная база данных**, хранящая информацию в виде коллекций и документов:

- **Коллекция (Collection)** — логическая группа документов, аналог таблицы в SQL.
- **Документ (Document)** — отдельная запись в коллекции, аналог строки.
- **Поле (Field)** — отдельный атрибут документа.
- **Подколлекции (Subcollections)** — вложенные коллекции внутри документа.

Пример структуры:

```
users (коллекция)
 └── user123 (документ)
      ├── name: "Иван"
      ├── email: "ivan@mail.ru"
      └── notes (подколлекция)
            ├── note1 (документ)
            │     ├── title: "Первая заметка"
            │     └── content: "Привет, Firebase!"
            └── note2 (документ)
```

Firestore — **реактивная база**: если приложение подписано на коллекцию или документ через `Stream`, все изменения отображаются **в реальном времени** без повторных запросов к серверу.

**Отличия Cloud Firestore от традиционных SQL-баз**

| Характеристика | SQL (PostgreSQL, MySQL) | Cloud Firestore |
|---|---|---|
| Структура данных | Таблицы, строки, столбцы | Коллекции и документы |
| Тип данных | Строго типизированные | Динамические, JSON-подобные |
| Связи | JOIN, внешние ключи | Вложенные коллекции |
| Масштабирование | Вертикальное | Горизонтальное |
| Запросы | SQL, сложные фильтры | Ограниченные по условиям и индексам |
| Работа офлайн | Требует отдельной реализации | Поддерживается из коробки |
| Синхронизация | По инициативе клиента | Автоматическая в реальном времени |

Firestore — не замена SQL, а инструмент для проектов, где важны скорость разработки, масштабируемость и реактивность.

**Преимущества использования Firebase**

- **Быстрое подключение** — можно развернуть облачную БД за 5 минут.
- **Не требует сервера** — экономит время и ресурсы.

- **Синхронизация в реальном времени** — обновления видны мгновенно на всех устройствах.
- **Масштабируемость** — работает от прототипа до миллионов пользователей.
- **Безопасность** — гибкие правила доступа (Security Rules).
- **Интеграция с аналитикой и ML** — можно использовать Google Analytics и ML Kit.

**Ограничения Firebase**

- **Нет транзакций между коллекциями** (только в пределах одной).
- **Ограниченные запросы** (не поддерживаются сложные JOIN).
- **Платная модель хранения и чтения** — при большом объёме данных расходы растут.
- **Зависимость от Google Cloud** — нельзя развернуть локально.

Поэтому Firebase чаще применяют для:

- мобильных MVP-приложений;
- небольших SaaS-сервисов;
- учебных и исследовательских проектов;
- прототипов стартапов.

**Firebase и Flutter: взаимодействие**

В экосистеме Flutter для работы с Firebase используется набор пакетов **FlutterFire**, включающий:

- `firebase_core` — обязательный пакет для инициализации;
- `cloud_firestore` — работа с базой данных Firestore;
- `firebase_auth` — авторизация;
- `firebase_storage` — хранение файлов;
- `firebase_messaging` — push-уведомления;

- `firebase_crashlytics`, `firebase_analytics` — аналитика и отчёты об ошибках.

*Инициализация Firebase*

При запуске Flutter-приложения необходимо вызвать:

```
await Firebase.initializeApp(
  options: DefaultFirebaseOptions.currentPlatform,
);
```

Файл `firebase_options.dart` создаётся автоматически через CLI:

```
flutterfire configure
```

**Безопасность и Firestore Security Rules**

Firebase использует декларативный язык правил доступа:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId}/notes/{noteId} {
      allow read, write: if request.auth.uid == userId;
    }
  }
}
```

Эти правила определяют, кто и при каких условиях имеет доступ к документам.

На этапе практического занятия допускается **временное ослабление правил**:

```
allow read, write: if true;
```

Но в продакшене это **недопустимо**, поскольку открывает БД всему интернету.

Студенты должны осознавать, что **правила безопасности Firebase это аналог «прав доступа» в SQL** и являются критически важной частью архитектуры.

**Сравнение Firestore и Realtime Database**

| Параметр | Firestore | Realtime Database |
|---|---|---|
| Модель данных | Документы / коллекции | Дерево JSON |
| Запросы | Сложные, индексируемые | Простые, по узлам |
| Масштабирование | Автоматическое | Ограничено |
| Офлайн-режим | Да | Да |
| Обновления в реальном времени | Да | Да |
| Подходит для | Новых проектов | Старых / IoT решений |

Firestore считается более современным решением и рекомендуется к использованию.

**Типовая архитектура Flutter-приложения с Firebase**

```
Flutter UI
 ├── Экран (NotesPage)
 │     ├── StreamBuilder -> подписка на Firestore
 │     ├── Form -> создание/редактирование документов
 │     └── Snackbar / Dialog для UX
 ├── Firebase SDK
 │     ├── firebase_core
 │     ├── cloud_firestore
 │     └── firebase_auth (опционально)
 └── Firestore Database
       └── Коллекция "notes"
```

Все операции CRUD выполняются непосредственно из приложения, а Firestore обеспечивает синхронизацию и хранение данных в облаке.

**Рекомендации по архитектуре и безопасности**

1. Использовать отдельный проект Firebase для учебных целей.
2. Не хранить приватные ключи и токены в открытом доступе.
3. Ограничивать права пользователей через `Security Rules`.

4. Для продакшена обязательно подключать `firebase_auth` и правила вида:

```
allow read, write: if request.auth != null;
```

5. Для сложных проектов — выносить бизнес-логику на **Cloud Functions** или собственный бэкенд.

Firebase является мощным инструментом для быстрой интеграции базы данных в мобильное приложение.

Для Flutter-разработчиков он предоставляет простейший путь к созданию полноценных клиент-серверных решений без ручного написания API и развертывания серверов.

Понимание архитектуры Firebase, принципов работы Firestore и правил безопасности является фундаментальной компетенцией в области кроссплатформенной разработки мобильных приложений.

Ссылка на официальную документацию для подключение проекта Flutter к Firebase

## Практическая часть

### Общий алгоритм

1. Подготовка проекта Flutter
2. Создание проекта Firebase и привязка через FlutterFire CLI
3. Установка пакетов и инициализация Firebase в коде
4. Настройка Cloud Firestore и правил безопасности
5. Реализация CRUD (экран со списком документов + добавление/редактирование/удаление)
6. Скриншоты и отчёт

### 1) Подготовка проекта Flutter

Создайте новый проект (или используйте существующий):

```
flutter create firebase_notes_app
cd firebase_notes_app
```

**Контрольная точка 1:** проект компилируется и запускается на эмуляторе/устройстве (`flutter run`).

(Если среда ещё не настроена, используйте материал ПЗ №1 по установке SDK и эмулятора.)

### 2) Создание проекта Firebase и привязка FlutterFire

**Вариант А (рекомендуется): FlutterFire CLI**

1. Установите CLI (один раз на машину):

```
dart pub global activate flutterfire_cli
```

2. Авторизуйтесь в Google (откроется браузер):

```
flutterfire configure
```

3. Выберите/создайте проект Firebase и платформы (Android, iOS — при наличии macOS, Web — опционально).

В корне появится `lib/firebase_options.dart` и настроенные `google-services` для Android/iOS.

**Вариант Б (ручная привязка через консоль Firebase)** Создать проект в https://console.firebase.google.com → добавить приложение Android (пакет `com.example.firebase_notes_app`), скачать `google-services.json` в `android/app/`;

для iOS — `GoogleService-Info.plist` в `ios/Runner/`.

Затем вручную подключить Gradle-плагины. (Предпочтительно Вариант А.)

**Контрольная точка 2:** в проекте есть `firebase_options.dart`, сборка проходит без ошибок.

**3) Установка пакетов и инициализация**

Добавьте зависимости в `pubspec.yaml`:

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^3.6.0
  cloud_firestore: ^5.4.4
```

Инициализация в `lib/main.dart`:

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(const NotesApp());
}

class NotesApp extends StatelessWidget {
  const NotesApp({super.key});
  @override
```

```
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Firebase Notes',
      theme: ThemeData(useMaterial3: true),
      home: const NotesPage(),
    );
  }
}
```

**Контрольная точка 3:** приложение запускается без ошибок и не падает на старте.

**4) Cloud Firestore: коллекция и правила**

Создайте в Firestore коллекцию `notes` с полями:

`title` : string

`content` : string

`createdAt` : timestamp

`updatedAt` : timestamp

**Учебные правила (без аутентификации, только на время ПЗ!)**

```
// Firestore Security Rules (учебный режим!):
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      // Разрешить чтение/запись всем в рамках практики
      allow read, write: if true;
    }
  }
}
```

В отчёте обязательно отметьте, что в продакшене нужны строгие правила (см. «Дополнительно» ниже).

**Контрольная точка 4:** вы можете создавать документы через консоль Firebase.

**5) Экран CRUD (пример кода)**

Простой список заметок + форма добавления/редактирования. Создайте `lib/notes_page.dart` и вставьте:

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';

class NotesPage extends StatefulWidget {
  const NotesPage({super.key});
  @override
  State<NotesPage> createState() => _NotesPageState();
}

class _NotesPageState extends State<NotesPage> {
  final _db = FirebaseFirestore.instance;
  final _titleCtrl = TextEditingController();
  final _contentCtrl = TextEditingController();

  Future<void> _createNote() async {
    final title = _titleCtrl.text.trim();
    final content = _contentCtrl.text.trim();
    if (title.isEmpty) return;

    final now = Timestamp.now();
    await _db.collection('notes').add({
      'title': title,
      'content': content,
      'createdAt': now,
      'updatedAt': now,
    });

    _titleCtrl.clear();
    _contentCtrl.clear();
    if (mounted) Navigator.pop(context);
  }
```

```
        Future<void> _updateNote(DocumentReference ref, String
title, String content) async {
          await ref.update({
            'title': title,
            'content': content,
            'updatedAt': Timestamp.now(),
          });
        }

        Future<void> _deleteNote(DocumentReference ref) async {
          await ref.delete();
        }

        void _openCreateDialog() {
          showDialog(
            context: context,
            builder: (_) => AlertDialog(
              title: const Text('Новая заметка'),
              content: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                     TextField(controller: _titleCtrl, decoration:
const InputDecoration(labelText: 'Заголовок')),
                   TextField(controller: _contentCtrl, decoration:
const InputDecoration(labelText: 'Текст')),
                ],
              ),
              actions: [
                                    TextButton(onPressed:   ()   =>
Navigator.pop(context), child: const Text('Отмена')),
                 FilledButton(onPressed: _createNote, child: const
Text('Сохранить')),
              ],
            ),
          );
        }
```

```
      void _openEditDialog(DocumentSnapshot doc) {
        final data = doc.data() as Map<String, dynamic>? ?? {};
              final  titleCtrl  =  TextEditingController(text:
data['title'] ?? '');
             final  contentCtrl  =  TextEditingController(text:
data['content'] ?? '');

        showDialog(
          context: context,
          builder: (_) => AlertDialog(
            title: const Text('Редактировать'),
            content: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                   TextField(controller: titleCtrl, decoration:
const InputDecoration(labelText: 'Заголовок')),
                 TextField(controller: contentCtrl, decoration:
const InputDecoration(labelText: 'Текст')),
              ],
            ),
            actions: [
                                TextButton(onPressed:   ()   =>
Navigator.pop(context), child: const Text('Отмена')),
              FilledButton(
                onPressed: () async {
                                 await  _updateNote(doc.reference,
titleCtrl.text.trim(), contentCtrl.text.trim());
                  if (mounted) Navigator.pop(context);
                },
                child: const Text('Обновить'),
              ),
            ],
          ),
        );
      }
```

```
      @override
      Widget build(BuildContext context) {
                                  final       notesStream       =
_db.collection('notes').orderBy('createdAt',          descending:
true).snapshots();

        return Scaffold(
          appBar: AppBar(title: const Text('Firebase Notes')),
          floatingActionButton: FloatingActionButton(
            onPressed: _openCreateDialog,
            child: const Icon(Icons.add),
          ),
          body: StreamBuilder<QuerySnapshot>(
            stream: notesStream,
            builder: (context, snapshot) {
               if (snapshot.hasError) return const Center(child:
Text('Ошибка загрузки'));
               if (!snapshot.hasData) return const Center(child:
CircularProgressIndicator());

              final docs = snapshot.data!.docs;
                  if (docs.isEmpty) return const Center(child:
Text('Пока нет заметок'));

              return ListView.separated(
                padding: const EdgeInsets.all(12),
                itemCount: docs.length,
                          separatorBuilder: (_, __) => const
SizedBox(height: 8),
                itemBuilder: (context, i) {
                  final doc = docs[i];
                        final data = doc.data() as Map<String,
dynamic>? ?? {};
                         final title = data['title'] ?? '(без
названия)';
```

```
                  final content = data['content'] ?? '';

                  return Card(
                    child: ListTile(
                       title: Text(title, maxLines: 1, overflow:
TextOverflow.ellipsis),
                           subtitle: Text(content, maxLines: 2,
overflow: TextOverflow.ellipsis),
                       onTap: () => _openEditDialog(doc),
                       trailing: IconButton(
                         icon: const Icon(Icons.delete),
                                          onPressed: () =>
_deleteNote(doc.reference),
                       ),
                     ),
                   );
                 },
               );
             },
           ),
         );
       }
     }
```

**Контрольная точка 5:**

- добавление заметки создаёт документ в `notes`;
- список обновляется в реальном времени;
- редактирование меняет документ;
- удаление работает.

**6) Дополнительно (по желанию, +1 балл)**

1. **Аутентификация**

Подключите пакет `firebase_auth` и реализуйте анонимный вход:

```
import 'package:firebase_auth/firebase_auth.dart';
```

```
    await FirebaseAuth.instance.signInAnonymously();
```

И замените правила Firestore на:

```
    // Разрешаем доступ только аутентифицированным пользователям
    allow read, write: if request.auth != null;
```

2. **Владение данными пользователя**

Храните заметки внутри коллекции пользователя:

`users/{uid}/notes/{noteId}`, где `uid = FirebaseAuth.instance.currentUser!.uid`.

3. **Валидация и UX**

Не разрешайте пустые заголовки, выводите Snackbar при успехе/ошибке.

**Что сдаём (контрольные задания)**

1. Скриншот настроенного проекта Firebase (страница проекта или Firestore с коллекцией `notes`).
2. Скриншот запущенного приложения **с отображением списка** (пустого или с данными).
3. Скриншот **после добавления** заметки (элемент появился в списке).
4. Скриншот **после редактирования** (обновлённый заголовок/текст).
5. Скриншот **после удаления** (элемент исчез).
6. Весь отчёт в Git(Readme):

- кратко: как создавали и привязывали Firebase-проект (CLI/консоль);
- какие пакеты использовали и где инициализировали Firebase;
- структура коллекций/документов;
- какие правила безопасности установили (и почему они **недостаточны** для продакшена);
- с какими ошибками столкнулись и как их решили.

**Требования к отчёту**

- Объём: 2–4 страницы + все скриншоты контрольных этапов.
- Чёткая структура, нумерованные шаги, ссылки на ключевые файлы (`main.dart`, `notes_page.dart`).
- Отдельный раздел «Безопасность: что поменять в продакшене».

**Частые проблемы и быстрые решения**

- **com.google.gms.google-services не найден** → проверьте, что FlutterFire CLI обновил Gradle-скрипты Android и выполните `flutter clean && flutter pub get`.
- **Permission denied** при обращении к Firestore → проверьте актуальные **rules** и что вы правильно выбрали БД (Firestore vs Realtime Database).
- **iOS: приложение падает на старте** → убедитесь, что добавили `GoogleService-Info.plist` в `Runner` и включили `FirebaseApp.configure` через `Firebase.initializeApp`.
- **Не видит Android-устройство** → перепроверьте шаги из ПЗ №1 (AVD/драйверы/USB-отладка).