!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 № 9

Работа с базами данных. Подключение приложения к Supabase (Flutter)

Цели занятия

- Подключить Flutter-приложение к Supabase (Postgres + Auth + Realtime).
- Освоить инициализацию `supabase_flutter`, чтение/запись данных и потоковые обновления (`stream()`).
- Реализовать базовый CRUD (создание, чтение, обновление, удаление) с реактивным списком.
- Включить Row Level Security (RLS) и настроить безопасные политики доступа для аутентифицированных пользователей.

Теоретическая часть

Общие сведения о Supabase

Supabase — это открытая платформа (open-source backend-as-a-service), предназначенная для быстрого создания и управления приложениями, использующими базу данных **PostgreSQL**.

Её часто называют **открытой альтернативой Firebase**, поскольку она предоставляет схожий функционал:

- аутентификация пользователей,
- хранение данных,
- работа с файлами (storage),
- реактивные обновления (Realtime),
- серверные функции и API без необходимости писать собственный сервер.

При этом Supabase не является отдельной проприетарной базой — это надстройка над PostgreSQL, которая добавляет удобный слой API и интерфейсы управления.

Благодаря этому все данные физически хранятся в классической реляционной базе, что даёт преимущества SQL: сложные запросы, индексы, транзакции, связи, внешние ключи.

Архитектура Supabase

Архитектура Supabase состоит из нескольких ключевых компонентов:

1. **PostgreSQL** — основа хранения данных. Именно в ней создаются таблицы, схемы, триггеры и функции.
2. **PostgREST** — слой, автоматически генерирующий REST API для каждой таблицы БД.

Каждый эндпоинт доступен по HTTP-запросам (`GET`, `POST`, `PATCH`, `DELETE`) и использует SQL-запросы под капотом.

3. **Realtime сервер** — обеспечивает реактивное обновление данных. Он подписывается на WAL (Write-Ahead Log) PostgreSQL и транслирует изменения клиентам в реальном времени через WebSocket.

4. **Auth сервер (GoTrue)** — реализует систему аутентификации и управления пользователями, включая email-пароли, OAuth-провайдеров (Google, GitHub, Apple и др.) и Magic Link.

5. **Storage** — управляемое файловое хранилище для изображений, документов и других медиафайлов, с доступом по API и правилам безопасности.

6. **Edge Functions** — серверные функции, позволяющие запускать собственный код (TypeScript/Deno) на стороне Supabase.

Благодаря этому набору Supabase предоставляет полный backend без необходимости вручную писать серверную часть.

3. Преимущества Supabase по сравнению с традиционным backend

Возможность	**Supabase**	**Классический backend**
Развёртывание	Мгновенно (в облаке)	Требует сервер, CI/CD
База данных	PostgreSQL (готовая)	Нужно настраивать отдельно
REST API	Генерируется автоматически	Пишется вручную
Realtime	Встроен	Требует WebSocket-сервер
Аутентификация	Встроенная (GoTrue)	Нужно разрабатывать самостоятельно
Панель управления	Есть веб-интерфейс (Supabase Studio)	Обычно нет
Безопасность	RLS (Row-Level Security)	Зависит от реализации

Таким образом, Supabase идеально подходит для прототипирования, стартапов, учебных проектов и приложений, где требуется быстро связать frontend с реляционной базой данных.

Аутентификация и управление пользователями

Supabase Auth основан на библиотеке **GoTrue** — это лёгкий сервер авторизации с поддержкой:

- **Регистрации и входа по email и паролю**;
- **Passwordless (Magic Link)** — вход по одноразовой ссылке из письма;
- **OAuth 2.0 провайдеров** (Google, GitHub, Apple и т.д.);
- **Анонимных сессий и refresh-токенов**.

После успешной авторизации Supabase возвращает токен, который автоматически используется во всех последующих запросах. Метод `supabase.auth.currentUser` в Flutter позволяет получить объект текущего пользователя, включая его уникальный `id`, используемый для связи с таблицами.

Row Level Security (RLS)

Row Level Security (RLS) — это механизм PostgreSQL, который ограничивает доступ к отдельным строкам таблицы на уровне базы данных. В Supabase RLS **всегда должен быть включён**, если вы работаете напрямую из клиента Flutter или JavaScript.
Без него все пользователи смогут видеть и изменять все записи, что недопустимо.

Принцип работы RLS:

1. Включается защита на уровне таблицы:

```
alter table notes enable row level security;
```

2. Создаются **policies (политики)** — правила доступа. Пример: разрешить пользователю читать только свои записи:

```
create policy "Read own notes"
on public.notes
for select
to authenticated
using (user_id = auth.uid());
```

Здесь `auth.uid()` возвращает ID текущего аутентифицированного пользователя.

3. Если пользователь не удовлетворяет ни одной политике, операция отклоняется.

Таким образом, Supabase автоматически применяет эти правила при каждом REST или RPC-запросе.

Realtime и потоковые данные

Supabase Realtime — это сервис, который позволяет подписаться на изменения в таблицах PostgreSQL и получать уведомления о вставках, обновлениях и удалениях в реальном времени.

Как это работает:

- Realtime подписывается на WAL (журнал операций PostgreSQL);
- Когда таблица изменяется (например, новая запись), событие отправляется всем клиентам, подписанным на этот канал;
- В Flutter это реализуется через метод:
- `supabase.from('notes').stream(primaryKey: ['id']).listen(...)`

Типичные события:

- `INSERT` — добавлена новая запись;
- `UPDATE` — запись изменена;
- `DELETE` — запись удалена.

Эта технология позволяет обновлять интерфейс приложения без перезагрузки страницы — например, новые заметки появляются мгновенно на экране всех авторизованных пользователей.

Работа с базой данных (CRUD)

Как и в любом приложении, взаимодействие с базой данных состоит из четырёх базовых операций:

Операция	Метод Supabase Flutter	SQL-эквивалент
Create	insert()	INSERT INTO
Read	select()	SELECT * FROM
Update	update()	UPDATE SET ...
Delete	delete()	DELETE FROM

Пример чтения заметок:

```
final response = await supabase
  .from('notes')
  .select()
  .eq('user_id', supabase.auth.currentUser!.id);
```

Эти методы отправляют запросы напрямую к REST API Supabase (PostgREST), а Supabase сам конвертирует их в SQL-запросы и возвращает JSON-результат.

Инициализация Supabase в Flutter

Для работы с Supabase во Flutter используется официальный пакет `supabase_flutter`.

Он предоставляет упрощённую инициализацию и интеграцию с виджетами.

Шаги подключения:

1. Установить пакет через `pubspec.yaml`:

```
dependencies:
  supabase_flutter: ^2.0.0
```

2. Инициализировать Supabase в `main()` до запуска приложения:

```
await Supabase.initialize(
  url: 'https://your-project.supabase.co',
  anonKey: 'your-anon-key',
);
```

3. Получить экземпляр клиента:

```
final supabase = Supabase.instance.client;
```

4. Далее использовать его для аутентификации и запросов.

Безопасность и хранение данных

Supabase использует несколько уровней защиты:

1. **RLS** — основная защита данных.
2. **JWT-токен** — проверка подлинности каждого запроса.
3. **Политики на уровне API** — администратор может задавать, кто имеет доступ к REST, Realtime или Storage.
4. **Лимиты доступа по ключам:**

- `anon key` (публичный) используется в клиенте,
- `service_role key` (приватный) — только на сервере или в админ-интерфейсе.

Таким образом, вся логика безопасности централизована в самой базе данных, а не в приложении.

Supabase vs Firebase: сравнение концепций

Параметр	**Supabase**	**Firebase**
Тип БД	SQL (PostgreSQL)	NoSQL (Firestore / Realtime DB)
API	REST, GraphQL (через PostgREST)	Проприетарный SDK
Realtime	Через WAL	Через WebSocket
Хостинг кода	Edge Functions (Deno)	Cloud Functions (Node.js)
Open Source	Да	Нет
Локальный деплой	Возможен (Docker)	Нет

Вывод: Supabase ближе к традиционному подходу (SQL + строгие схемы), а Firebase — к документно-ориентированному. Для приложений, где важны связи, транзакции и SQL-запросы, Supabase предпочтительнее.

Применение Supabase в Flutter-проектах

Supabase идеально подходит для Flutter-приложений:

- Быстрая интеграция через `supabase_flutter`;
- Поддержка кроссплатформенности (Android, iOS, Web);
- Простая реализация аутентификации;
- Реактивное обновление интерфейса без написания серверной логики.

Типовые сценарии использования:

- Приложения заметок, ToDo, чатов;
- Прототипы MVP стартапов;
- Учебные лабораторные по мобильной разработке;
- Админ-панели и личные кабинеты.

Supabase — это современная облачная платформа, которая сочетает мощь PostgreSQL с удобством no-code API.

Для Flutter-разработчика это возможность создавать полнофункциональные приложения без написания отдельного backend.

При этом сохраняется полная совместимость с SQL и безопасность благодаря RLS и JWT.

Практическая часть

Подготовка проекта Flutter

```
flutter create supabase_notes_app
cd supabase_notes_app
```

Контрольная точка 0: проект собирается и запускается (`flutter run`).

1) Создание проекта Supabase и ключей

1. Зайдите в https://supabase.com → New project → сгенерируйте **Project URL** и **anon (public) key**.
2. В Dashboard откройте **Table editor** и создайте таблицу `notes` со схемой:
   - `id` uuid (PK, default: `gen_random_uuid()`)
   - `user_id` uuid (nullable = false)
   - `title` text
   - `content` text
   - `created_at` timestamp tz (default `now()`)
   - `updated_at` timestamp tz (default `now()`)

Включите **RLS** на таблице (`Enable RLS`). Создайте политики (SQL → **Policies**):

- Чтение только своих записей:

```
create policy "Read own notes"
on public.notes
for select
to authenticated
using (user_id = auth.uid());
```

- Вставка от имени текущего пользователя:

```
create policy "Insert own notes"
on public.notes
for insert
to authenticated
with check (user_id = auth.uid());
```

- Обновление/удаление только своих записей:

```
create policy "Update own notes"
on public.notes
for update
to authenticated
using (user_id = auth.uid());

create policy "Delete own notes"
on public.notes
for delete
to authenticated
using (user_id = auth.uid());
```

Пояснение: RLS **должна** быть включена для схем, доступных из клиента; политики ограничивают операции пользователем `auth.uid()`. (Supabase)

2) Установка пакета и инициализация в Flutter

В `pubspec.yaml` добавьте зависимости:

```
dependencies:
  flutter:
    sdk: flutter
  supabase_flutter: ^2.0.0
```

(версию уточняйте по pub.dev, можно оставить мажорный «2»). (Dart packages)

Далее — инициализация в `lib/main.dart`:

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

const supabaseUrl = 'https://YOUR-PROJECT-REF.supabase.co';
const supabaseAnonKey = 'YOUR-ANON-PUBLIC-KEY';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
    await Supabase.initialize(url: supabaseUrl, anonKey: supabaseAnonKey);
```

```
  runApp(const NotesApp());
}

class NotesApp extends StatelessWidget {
  const NotesApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Supabase Notes',
      theme: ThemeData(useMaterial3: true),
      home: const AuthGate(),
    );
  }
}
```

Пример инициализации соответствует официальной документации. (Supabase)

3) Простейшая аутентификация (email + magic link или email+password)

Для учебной работы используем **email+password** (самый быстрый старт). На экране входа вызовите:

```
final supabase = Supabase.instance.client;

await supabase.auth.signUp(email: email, password: password); // регистрация
await supabase.auth.signInWithPassword(email: email, password: password); // вход
```

После входа `auth.currentUser` содержит `id`, его будем писать в `user_id`. (Можно заменить на passwordless/magic link — см. Quickstart.) (Supabase)

4) Экран со списком и Realtime-стримом

Создайте `lib/notes_page.dart` и подключите поток из таблицы:

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

```
    final supabase = Supabase.instance.client;

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

    class _NotesPageState extends State<NotesPage> {
           late   final   Stream<List<Map<String,   dynamic>>>
_notesStream;

      @override
      void initState() {
        super.initState();
        final uid = supabase.auth.currentUser!.id;
        _notesStream = supabase
            .from('notes')
            .stream(primaryKey: ['id'])
             .eq('user_id', uid)                  // фильтр по
владельцу
            .order('created_at', ascending: false);
      }

       Future<void> _createNote(String title, String content)
async {
        final uid = supabase.auth.currentUser!.id;
        await supabase.from('notes').insert({
          'user_id': uid,
          'title': title,
          'content': content,
          'updated_at': DateTime.now().toIso8601String(),
        });
      }
```

```
       Future<void> _updateNote(String id, String title, String
content) async {
         await supabase.from('notes').update({
           'title': title,
           'content': content,
           'updated_at': DateTime.now().toIso8601String(),
         }).eq('id', id);
       }

       Future<void> _deleteNote(String id) async {
         await supabase.from('notes').delete().eq('id', id);
       }

       void _openCreateDialog() {
         final titleCtrl = TextEditingController();
         final contentCtrl = TextEditingController();
         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: () async {
                         await _createNote(titleCtrl.text.trim(),
contentCtrl.text.trim());
```

```
                  if (mounted) Navigator.pop(context);
                },
                child: const Text('Сохранить'),
              ),
            ],
          ),
        );
      }

      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(
            title: const Text('Supabase Notes'),
            actions: [
              IconButton(
                icon: const Icon(Icons.logout),
                onPressed: () async {
                  await supabase.auth.signOut();
                },
              ),
            ],
          ),
          floatingActionButton: FloatingActionButton(
            onPressed: _openCreateDialog,
            child: const Icon(Icons.add),
          ),
          body: StreamBuilder<List<Map<String, dynamic>>>(
            stream: _notesStream,
            builder: (context, snapshot) {
               if (snapshot.hasError) return const Center(child:
Text('Ошибка загрузки'));
               if (!snapshot.hasData) return const Center(child:
CircularProgressIndicator());
              final notes = snapshot.data!;
```

```
                  if (notes.isEmpty) return const Center(child:
Text('Пока нет заметок'));
               return ListView.separated(
                 padding: const EdgeInsets.all(12),
                 itemCount: notes.length,
                           separatorBuilder: (_, __) => const
SizedBox(height: 8),
                 itemBuilder: (context, i) {
                   final n = notes[i];
                   return Dismissible(
                     key: ValueKey(n['id']),
                                  background: Container(color:
Colors.red.withOpacity(.1)),
                     onDismissed: (_) => _deleteNote(n['id']),
                     child: Card(
                       child: ListTile(
                                 title: Text(n['title'] ?? '(без
названия)', maxLines: 1, overflow: TextOverflow.ellipsis),
                            subtitle: Text(n['content'] ?? '',
maxLines: 2, overflow: TextOverflow.ellipsis),
                         onTap: () {
                           final tc = TextEditingController(text:
n['title'] ?? '');
                           final cc = TextEditingController(text:
n['content'] ?? '');
                           showDialog(
                             context: context,
                             builder: (_) => AlertDialog(
                                                  title: const
Text('Редактировать'),
                               content: Column(
                                 mainAxisSize: MainAxisSize.min,
                                 children: [
                                       TextField(controller: tc,
decoration: const InputDecoration(labelText: 'Заголовок')),
```

```
                                        TextField(controller: cc,
decoration: const InputDecoration(labelText: 'Текст')),
                                    ],
                                ),
                                actions: [
                                      TextButton(onPressed: () =>
Navigator.pop(context), child: const Text('Отмена')),
                                  FilledButton(
                                    onPressed: () async {
                                        await _updateNote(n['id'],
tc.text.trim(), cc.text.trim());
                                                                if (mounted)
Navigator.pop(context);
                                    },
                                    child: const Text('Обновить'),
                                  ),
                                ],
                            ),
                        );
                    },
                                 trailing: IconButton(icon: const
Icon(Icons.delete_outline),        onPressed:        ()        =>
_deleteNote(n['id'])),
                    ),
                  ),
                );
              },
            );
          },
        ),
      );
    }
  }
```

Замечания:

- Метод `stream(primaryKey: ['id'])` обязателен: первичный ключ нужен клиенту для корректного маппинга приходящих событий. (Supabase)
- Если фильтруете поток, используйте чейнинг (`.eq('user_id', uid)`, `.order(...)`, `.limit(...)`). (Supabase)

5) Auth-гейт (минимум)

Добавьте простой экран `AuthGate` с формой email+password. После входа рендерьте `NotesPage`. (Логику можно взять из Flutter-квикстарта Supabase.) (Supabase)

6) Проверка

Контрольная точка 1: успешная инициализация Supabase (приложение не падает, нет ошибок сети).

Контрольная точка 2: регистрация/вход работают, в `auth.currentUser` есть `id`.

Контрольная точка 3: **CREATE** → вставка заметки создаёт запись в `notes` с `user_id = auth.uid()`.

Контрольная точка 4: **READ (Realtime)** → список обновляется без перезапуска (сразу видно новые/изменённые записи).

Контрольная точка 5: **UPDATE/DELETE** → изменения применяются и отражаются в списке.

Контрольная точка 6: RLS включает доступ только к своим заметкам (попытка читать/менять чужие — denied).

Что сдаём (контрольные задания)

1. Скриншот настроенного проекта Supabase (Dashboard: Database → `notes`, включён RLS, список Policies).
2. Скриншот экрана входа и экрана со списком (пустого и с данными).
3. Скриншот после добавления заметки (элемент появился).
4. Скриншот после редактирования и после удаления.

5. Файл `README` в репозитории: кратко

- как подключали Supabase (шаги, ключи, URL — без публикации приватных секретов),
- какие зависимости и где инициализация,
- структура таблицы и RLS-политики,
- с какими ошибками столкнулись и как решили (см. «Типовые ошибки» ниже).

Требования к отчёту

- Объём: 2–4 страницы + все скриншоты.
- Чёткие нумерованные шаги (инициализация, Auth, CRUD, Realtime).
- Раздел «Безопасность: RLS в продакшене» — что измените (только `authenticated`, политики по `auth.uid()`, валидация входных данных).

Контрольные вопросы

1. Зачем в `stream()` указывать `primaryKey` и что произойдёт, если его не задать? (Supabase)
2. Чем `stream()` (Realtime) отличается от обычного `select()`? Когда использовать каждый из подходов? (Supabase)
3. Что такое RLS и почему в Supabase она обязательна для клиентского доступа к БД? (Supabase)
4. Как связаны `auth.uid()` и поле `user_id` в таблице? Зачем политики `WITH CHECK`? (Supabase)
5. Как бы вы реализовали пагинацию и поиск по заголовку без перегрузки Realtime-канала?

Дополнительно (+1 балл)

- Замените email+password на **Passwordless (Magic Link)** или любой OAuth-провайдер из Supabase. (Supabase)

- Добавьте индексы (`created_at desc`) и сортировку на стороне БД для ускорения выборок.
- Подключите `storage` и загрузку изображений для заметок.
- Сделайте «Undo на удаление» через `SnackBar`.

Типовые ошибки и быстрые решения

- **`Permission denied` при чтении/записи:** проверьте, что пользователь **вошёл**, RLS включена, а политики разрешают действие (select/insert/update/delete) при `user_id = auth.uid()`. (Supabase)
- Реалтайм «не шевелится»: проверьте `primaryKey` в `stream()`, наличие RLS-политик на SELECT, стабильность фильтров; иногда проблема — в слишком узких фильтрах или неправильном ключе. (Supabase)
- Ошибка инициализации: убедитесь, что `Supabase.initialize(url, anonKey)` вызывается до `runApp`, и ключ/URL корректны (из Dashboard). (Supabase)