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 №11

Работа с базами данных. Основы работы с API (HTTP/REST) для Flutter

Цели занятия

- Понять базовые понятия HTTP/REST: методы, URL/эндпоинты, коды ответов, заголовки, тела запросов/ответов (JSON).
- Освоить основы интеграции Flutter-приложения с внешним API: `http`/`dio`, сериализация JSON, обработка ошибок и таймаутов.
- Научиться выстраивать **слой данных** с репозиторием и отделять его от UI (продолжаем архитектурную линию прошлых ПЗ).
- Реализовать список сущностей из публичного API + экран деталей + форму создания/редактирования (с демонстрацией запросов).
- Разобраться с пагинацией, фильтрацией, аутентификацией (Bearer), ретраями и UX при сетевых сбоях.

Теоретическая часть

Что такое API и как мы с ним общаемся

- **REST API**: набор HTTP-эндпоинтов, где ресурсы представлены URL'ами (например, `/posts`, `/posts/{id}`), а операции — методами:
  - `GET` (чтение), `POST` (создание), `PATCH/PUT` (изменение), `DELETE` (удаление).
- **Структура HTTP-запроса**: метод + URL + заголовки (Headers) + тело (Body, чаще JSON).
- **Статусы ответов**: `2xx` (успех), `4xx` (ошибка клиента: валидация, авторизация), `5xx` (ошибка сервера). Важные: `200 OK`, `201 Created`, `204 No Content`, `400/401/403/404/409/422`, `500/502/503`.
- **Заголовки**:
  - `Content-Type: application/json`
  - `Authorization: Bearer <JWT>` (при защищённых API).

JSON и сериализация

- Во Flutter преобразуем JSON ↔ Dart-модели: вручную (конструкторы `fromJson/toJson`) или через `json_serializable` (генерация).
- Рекомендация для учебных работ — начать вручную, затем освоить генерацию.

Клиенты для HTTP в Flutter

- `http` — простой, «батарейки в комплекте».
- `dio` — более «профессиональный»: перехватчики (interceptors), логирование, отмена запросов, FormData, удобные таймауты.
- В ПЗ используем **dio** как универсальный инструмент, но примеры легко переписать на `http`.

Архитектура слоя данных (повтор и усиление)

- Разделение ответственности: **UI ↔ Repository ↔ API client**. Это продолжение принципов разделения слоёв, применённых ранее (SQLite, списки, работа со стейтом).
- Плюсы: тестируемость, переиспользование, возможность заменить источник данных (локально/облако) без переписи UI.

Пагинация, фильтрация, поиск

- Пагинация: `?page=1&limit=20` или `?_page=1&_limit=20` (зависит от API).
- Фильтрация/поиск: `?q=...`, `?title_like=...`, `?sort=createdAt&order=desc`.
- В UI — подгрузка по скроллу или кнопка «Ещё» (см. практики списков из ПЗ №5).

Аутентификация и безопасность

- **Bearer JWT** в заголовке `Authorization`.
- Хранить токен безопасно (не коммитить в git), обновлять при истечении.
- В учебной работе используем открытый API (без auth), а раздел про токены — как задел для следующих занятий.

Ошибки, таймауты, повторные попытки

- Таймауты подключений/чтения (например, 10–15 сек).
- Ретраи на «сетевые» ошибки (экспоненциальная пауза 0.5s → 1s → 2s…).
- Идемпотентность: повтор `GET` безопасен, а `POST` — нет (в проде используют **idempotency keys**).
-

UX при сетевых операциях

- Состояния: loading / empty / error / data.
- Skeleton/placeholder при первичной загрузке; Pull-to-refresh; Snackbar для ошибок и Undo (паттерны списков у нас уже были).

Практическая часть

Итог задания

Мини-приложение **«API Notes Feed»**:

- Экран списка записей из публичного API с пагинацией.
- Экран деталей записи.
- Диалог создания/редактирования (демонстрация `POST/PATCH` на мок-API).
- Базовый слой данных с `dio`, репозиторием и обработкой ошибок.

Источник данных (2 варианта на выбор):

A. Только чтение:

`ttps://jsonplaceholder.typicode.com/posts` (стабильный, без аутентификации).

B. Полный CRUD: заведите свой ресурс на **mockapi.io** (или аналог), получите `baseUrl`, с которым будут работать `GET/POST/PATCH/DELETE`.

В отчёте укажите, какой вариант использовали. (Если B — приложите скрин студии mockapi.)

Подготовка проекта

```
flutter create api_notes_app
cd api_notes_app
```

Контрольная точка 0: проект собирается и запускается (эмулятор/устройство).

1) Зависимости

```
pubspec.yaml:
dependencies:
  flutter:
    sdk: flutter
  dio: ^5.7.0
```

(По желанию для логов: `logger` или `dio_logger`.)

2) Структура каталогов

```
lib/
  main.dart
  data/
     api_client.dart        // настройка dio, interceptors, baseUrl, таймауты
    notes_repository.dart// методы: list(), get(), create(), update(), delete()
  models/
    note.dart            // модель данных + fromJson/toJson
  pages/
     notes_page.dart       // список + пагинация + pull-to-refresh
    note_details_page.dart
```

Такой расклад согласуется с ранее применяемым принципом отделения слоя данных от UI.

3) Модель данных

`lib/models/note.dart`

```
class Note {
  final int id;            // для mockapi может быть String -> адаптируйте под свой API
  final String title;
  final String body;

   Note({required  this.id,  required  this.title,  required this.body});

  factory Note.fromJson(Map<String, dynamic> json) => Note(
    id: json['id'] is String ? int.tryParse(json['id']) ?? 0 : (json['id'] ?? 0),
    title: json['title'] ?? '',
    body: json['body'] ?? '',
  );

  Map<String, dynamic> toJson() => {
    'id': id,
```

```
        'title': title,
        'body': body,
      };
    }
```

4) API-клиент (dio)

lib/data/api_client.dart

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

class ApiClient {
  final Dio dio;

  ApiClient._(this.dio);

    factory ApiClient({required String baseUrl, String?
bearerToken}) {
      final dio = Dio(BaseOptions(
        baseUrl: baseUrl,
        connectTimeout: const Duration(seconds: 10),
        receiveTimeout: const Duration(seconds: 10),
        headers: {
          'Content-Type': 'application/json',
            if (bearerToken != null) 'Authorization': 'Bearer
$bearerToken',
        },
      ));

      dio.interceptors.add(InterceptorsWrapper(
        onRequest: (options, handler) {
          // Можно залогировать/прикрутить correlation-id
          handler.next(options);
        },
        onError: (e, handler) {
          // Глобальная обработка ошибок/ретраи (по желанию)
          handler.next(e);
        },
```

```
    ));
    return ApiClient._(dio);
  }
}
```

5) Репозиторий

lib/data/notes_repository.dart

```
import 'package:dio/dio.dart';
import '../models/note.dart';
import 'api_client.dart';

class NotesRepository {
  final ApiClient _client;
  NotesRepository(this._client);

  // A: jsonplaceholder (только чтение)
     // B: mockapi (полный CRUD) — эндпоинты могут отличаться: /notes, /notes/{id}

   Future<List<Note>> list({int page = 1, int limit = 20}) async {
    final resp = await _client.dio.get(
      '/posts',
       queryParameters: {'_page': page, '_limit': limit}, // для jsonplaceholder
    );
    final data = resp.data as List<dynamic>;
     return data.map((e) => Note.fromJson(e as Map<String, dynamic>)).toList();
  }

  Future<Note> get(int id) async {
    final resp = await _client.dio.get('/posts/$id');
    return Note.fromJson(resp.data as Map<String, dynamic>);
  }
```

```
       // Ниже — для варианта B (mockapi). В варианте A можно оставить заглушки.
      Future<Note> create(String title, String body) async {
        final resp = await _client.dio.post('/posts', data: {
          'title': title,
          'body': body,
        });
        return Note.fromJson(resp.data as Map<String, dynamic>);
      }

       Future<Note> update(int id, String title, String body) async {
        final resp = await _client.dio.patch('/posts/$id', data: {
          'title': title,
          'body': body,
        });
        return Note.fromJson(resp.data as Map<String, dynamic>);
      }

      Future<void> delete(int id) async {
        await _client.dio.delete('/posts/$id');
      }
    }
```

Примечание:

- Для jsonplaceholder `POST/PATCH/DELETE` возвращают фиктивные ответы (без реального сохранения). Этого достаточно, чтобы продемонстрировать формат запросов, обработку статусов и обновление UI.
- Для честного CRUD используйте mockapi.io и замените базовый URL + пути.

6) UI: список с пагинацией и деталями

`lib/pages/notes_page.dart` (эскиз — опустим импорты, оставим ключевую логику):

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

    class _NotesPageState extends State<NotesPage> {
      late final NotesRepository repo;
      final List<Note> _items = [];
      int _page = 1;
      bool _canLoadMore = true;
      bool _loading = false;

      @override
      void initState() {
        super.initState();
                    final client = ApiClient(baseUrl:
'https://jsonplaceholder.typicode.com');
        repo = NotesRepository(client);
        _refresh();
      }

      Future<void> _refresh() async {
            setState(() { _page = 1; _canLoadMore = true;
_items.clear(); });
        await _loadMore();
      }

      Future<void> _loadMore() async {
        if (!_canLoadMore || _loading) return;
        setState(() => _loading = true);
        try {
          final batch = await repo.list(page: _page, limit: 20);
          setState(() {
            _items.addAll(batch);
                    _canLoadMore = batch.isNotEmpty; // для
jsonplaceholder ~100 постов
            if (_canLoadMore) _page++;
```

```
          });
        } catch (e) {
          if (mounted) {
            ScaffoldMmessenger.of(context).showSnackBar(
              const SnackBar(content: Text('Ошибка загрузки')),
            );
          }
        } finally {
          if (mounted) setState(() => _loading = false);
        }
      }

      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(title: const Text('API Notes Feed')),
          floatingActionButton: FloatingActionButton(
            onPressed: () async {
                 // Для варианта B покажите диалог создания и
вызовите repo.create(...)
               // Для варианта A можно показать "демо-создание" и
локально добавить в список.
            },
            child: const Icon(Icons.add),
          ),
          body: RefreshIndicator(
            onRefresh: _refresh,
            child: _items.isEmpty && _loading
                                      ?   const   Center(child:
CircularProgressIndicator())
                : ListView.separated(
                    padding: const EdgeInsets.all(12),
                    itemCount: _items.length + 1,
                            separatorBuilder:  (_,  __)  =>  const
SizedBox(height: 8),
                    itemBuilder: (context, i) {
```

```
                        if (i == _items.length) {
                          // футер дозагрузки
                           if (_canLoadMore) { _loadMore(); return
const Center(child: Padding(
                               padding: EdgeInsets.all(16), child:
CircularProgressIndicator())); }
                          return const SizedBox.shrink();
                        }
                        final n = _items[i];
                        return Card(
                          child: ListTile(
                                title: Text(n.title, maxLines: 1,
overflow: TextOverflow.ellipsis),
                               subtitle: Text(n.body, maxLines: 2,
overflow: TextOverflow.ellipsis),
                             onTap: () => Navigator.push(context,
                                  MaterialPageRoute(builder: (_) =>
NoteDetailsPage(id: n.id, repo: repo))),
                             trailing: IconButton(
                                                    icon:  const
Icon(Icons.delete_outline),
                               onPressed: () async {
                                                 //  Вариант  B:  await
repo.delete(n.id);
                                  // В учебных целях можно удалить
локально:
                                                      setState(()  =>
_items.removeAt(i));
                                  ScaffoldMessenger.of(context).show
SnackBar(const SnackBar(content: Text('Удалено (демо)')));
                               },
                             ),
                          ),
                        );
                      },
                    ),
```

```
          ),
        );
      }
    }
```

lib/pages/note_details_page.dart:

```
    class NoteDetailsPage extends StatelessWidget {
      final int id;
      final NotesRepository repo;
         const  NoteDetailsPage({super.key,  required  this.id,
required this.repo});

      @override
      Widget build(BuildContext context) {
        return FutureBuilder<Note>(
          future: repo.get(id),
          builder: (context, snap) {
                if (snap.hasError) return const Scaffold(body:
Center(child: Text('Ошибка')));
                if (!snap.hasData) return const Scaffold(body:
Center(child: CircularProgressIndicator()));
            final n = snap.data!;
            return Scaffold(
              appBar: AppBar(title: Text('Запись #${n.id}')),
              body: Padding(
                padding: const EdgeInsets.all(16),
                child: Text(n.body),
              ),
            );
          },
        );
      }
    }
```

lib/main.dart:

```
    import 'package:flutter/material.dart';
    import 'pages/notes_page.dart';
```

```
void main() => runApp(const ApiNotesApp());

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

7) Контрольные точки

1. Проект запускается, зависимости установлены.
2. Отображается список записей, первый экран заполняется данными из API.
3. Пагинация работает: при прокрутке вниз подгружается следующая страница.
4. Экран деталей открывается и показывает содержимое.
5. Диалог «Создать запись»:
   - Вариант A (jsonplaceholder): демонстрация `POST`, локальное добавление в список, информирование, что на сервере не сохранится.
   - Вариант B (mockapi): реальная вставка и обновление списка после ответа 201.
6. Обработка ошибок: при сетевой ошибке — Snackbar/сообщение и повтор.

8) Что сдаём (контрольные задания)

- Скриншот экрана списка (первый экран, данные получены).
- Скриншот экрана деталей.

- Скриншот диалога создания и результата (элемент в списке).
- Краткий **отчёт (2–4 стр.)**:
  - Какой API использовали (A/B), базовый URL, примеры эндпоинтов.
  - Как устроены модель и репозиторий; ключевые фрагменты кода.
  - Как реализованы пагинация, ошибки/таймауты, UX (loading/empty/error).
  - Трудности и решения.
- (По желанию) Ссылка на репозиторий с Readme.

9) Требования к отчёту

- Структура: **Цели → Ход работы (шаги + код) → Результаты (скриншоты) → Выводы**.
- Чёткие ссылки на файлы `api_client.dart`, `notes_repository.dart`, `notes_page.dart`.
- Если выбран вариант B — приложить скрин настройки коллекции в mockapi.

10) Контрольные вопросы

1. Чем отличаются `GET`, `POST`, `PUT`, `PATCH`, `DELETE`? Приведите пример для каждой операции.
2. Для чего нужны коды ответов `201` и `204`, и когда они возвращаются?
3. Зачем отделять слой данных (репозиторий) от UI? Приведите два плюса такого подхода.
4. Что такое пагинация и чем отличается «кнопка Ещё» от «бесконечной прокрутки»? Какой UX выбрали вы и почему?
5. Как вы обрабатываете сетевые ошибки и как уведомляете пользователя? Какие статусы считаете «повторяемыми»?

11) Дополнительно (+1 балл)

- Добавьте **поиск** (фильтрация по `title`) с дебаунсом 250–300 мс (см. практики из ПЗ со списками).
- Включите **ретраи** на `DioErrorType.connectionError` (экспоненциальная пауза).
- Поддержите **pull-to-refresh** и **Undo на удаление** (Snackbar).
- Вынесите `baseUrl` и ключи в `.env` (через `--dart-define` или пакет конфигурации).