|  |  |
| --- | --- |
| 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 №5.**

**Работа со списками. Передача данных между модулями**

**Цели:**

* Научиться отображать коллекции данных с помощью ListView.builder.
* Освоить базовую навигацию Navigator.push / Navigator.pop и передачу данных через конструктор.
* Научиться добавлять, редактировать и удалять элементы списка без внешних пакетов и сложных архитектур.

**Теоретическая часть**

## Как Flutter рисует списки (важно для производительности)

* **Деревья:** Widget → Element → RenderObject. Рендерится не сам виджет, а соответствующий ему RenderObject.
* **Виртуализация:** ListView.builder/SliverList создают элементы «по требованию» — только те, что попали во вьюпорт + небольшой запас (cacheExtent). Это экономит память и время построения.
* **Slivers и viewport:** CustomScrollView управляет видимым окном; SliverList/SliverGrid — «куски» контента. Slivers позволяют смешивать: коллапсирующий AppBar, списки, секции, футеры — всё в одном скролле.

## Виды списков и когда их применять

* **ListView(children: ...)** — короткие, статические наборы (до ~20–30 элементов).
* **ListView.builder** — большие/динамические коллекции; минимальные затраты.
* **ListView.separated** — нужен разделитель между элементами.
* **GridView.builder** — плитка/карточки, каталог.
* **CustomScrollView + SliverList/SliverGrid** — сложные прокрутки (хэдеры, секции, футеры).
* **AnimatedList/SliverAnimatedList** — анимированные вставки/удаления (если важны красивые переходы).
* **ReorderableListView** — перетаскивание для изменения порядка.
* **ListTile** — готовый шаблон строки (leading/title/subtitle/trailing) для типичного списка.

### Ключевые параметры, влияющие на скорость

* **itemExtent** или **prototypeItem** — фиксированная высота строки → быстрее измерение.
* **cacheExtent** — сколько строить «с запасом». Не завышайте без нужды.
* **shrinkWrap** — пересчитывает высоту всего списка (дорого). Включать только когда список внутри другого скролла и элементов мало.
* **addRepaintBoundaries/addAutomaticKeepAlives** — по умолчанию true у ListView.builder; редко трогаем.

## Keys: стабильная идентичность элементов

* **Зачем:** Flutter переиспользует элементы. Без стабильного Key анимации/состояние «прыгают».

**Что выбирать:**

* + ValueKey(item.id) — лучший выбор для списков (стабильный бизнес-идентификатор).
  + ObjectKey(item) — если сам объект уникальный/иммутабельный.
  + UniqueKey() — только для временной уникальности, не для данных.
  + GlobalKey — не для списков (дорого, нужен в редких случаях).
* **Dismissible** требует уникальный и стабильный Key для корректной анимации удаления.

## Жесты и действия в строке

* **Свайп-удаление:** Dismissible (покажите подтверждение через SnackBar «Отменить»).
* **Меню действий:** PopupMenuButton, showModalBottomSheet, «долгое нажатие» (onLongPress).
* **Pull-to-refresh:** RefreshIndicator(onRefresh: ...) должен возвращать Future<void>.

## Управление состоянием списка

* **Локально (setState)**: храните массив элементов в состоянии экрана, обновляйте копиями (иммутабельность снижает баги).
* **Поднятие состояния:** передавайте колбэки в дочерние экраны/виджеты для изменения данных на родителе.
* **Сохранение табов/позиции скролла:** AutomaticKeepAliveClientMixin + PageStorageKey для сохранения состояния внутри вкладок.
* **Поиск с дебаунсом:** не фильтруйте на каждый символ — используйте Timer 250–300 мс.

## Асинхронность и загрузка данных

* **FutureBuilder** для первичной загрузки; учитывайте состояния: waiting, error, done.
* **Множественные запросы:** проверяйте mounted перед setState после await (чтобы не обновить размонтированный виджет).
* **Обновления:** для RefreshIndicator возвращайте Future окончания загрузки.
* **Исключения:** ловите ошибки и показывайте понятное сообщение + кнопку «Повторить».

## Передача данных между экранами (модулями)

### Navigator 1.0 (императивный, базовый)

**Вперёд:**

final result = await Navigator.push<T>(

context,

MaterialPageRoute(builder: (\_) => NextScreen(arg: item)),

);

Аргументы — через **конструктор** (предпочтительно) или RouteSettings(arguments: ...).

**Назад с результатом:**

Navigator.pop(context, updatedItem);

Вернувшись, вызывающая сторона получает result и обновляет список.

**Когда RouteSettings.arguments:** если не хотите «пробрасывать» много параметров через конструктор. На целевом экране:

final args = ModalRoute.of(context)!.settings.arguments as MyArgs;

### Navigator 2.0 / go\_router (современно)

**Именованные маршруты и параметры** в URL, удобны для deep-linking и веба.

**Возврат результата** также возможен: final res = await context.push<T>('/edit/$id');.

Для ПЗ достаточно **Navigator 1.0**; позже легко мигрировать на go\_router.

### Передавать объект или ID?

* **Малый/локальный проект:** можно передавать объект (иммутабельный, copyWith).
* **Сеть/БД:** передавайте **ID**, а на целевом экране актуализируйте данные из репозитория (исключает рассинхронизацию).

## Архитектурные соглашения для списков

* **Модель** (иммутабельная, copyWith, ==/hashCode при необходимости).
* **Репозиторий** (источник данных: память/SharedPreferences/API).
* **Экран списка** (загрузка, фильтрация, отображение, удаление).
* **Экран формы/деталей** (редактирование, валидация, Navigator.pop(result)).
* В дальнейшем: заменить локальное состояние на **Riverpod/BLoC** для тестируемости и «тонких» экранов.

## UX-штрихи для реальных списков

* **Пустое состояние:** иконка + текст + кнопка «Добавить».
* **Skeleton/placeholder:** вместо резких «прыжков» контента (в базовом виде — Container(height: …) серых оттенков).
* **Undo на удаление:** SnackBar(action: SnackBarAction(label: 'Отменить', ...)).
* **Большие тап-зоны:** минимум ~44 px по высоте строки, чтобы удобно нажималось.
* **Адаптация под масштаб текста:** проверьте при textScaleFactor = 1.3–1.5.
* **Доступность:** корректные semanticsLabel, понятные иконки, достаточный контраст.

## Пагинация и «бесконечная прокрутка»

* **Порог подгрузки:** реагируйте, когда position.pixels >= position.maxScrollExtent \* 0.9.
* **Футер-заглушка:** внизу CircularProgressIndicator через SliverToBoxAdapter/последний ListTile.
* **Идempotентность:** флаг «уже грузим» — чтобы не отправлять повторные запросы.

## Работа с изображениями в списках

* **Кэширование:** Image.network(..., cacheWidth: …, cacheHeight: …) для миниатюр — снижает память.
* **Плейсхолдер:** FadeInImage.memoryNetwork (или сначала локальную иконку/инициал).
* **Изоляция тяжёлых задач:** парсинг больших JSON — в compute(...).

**Практическая часть**

**Необходимо разработать** мини-приложение **«Simple Notes»** с одним списком заметок (заголовок + содержимое), добавлением/редактированием/удалением.

## 4.1. Подготовка проекта

flutter create simple\_notes

cd simple\_notes

## 4.2. Структура файлов (простая)

lib/

main.dart

edit\_note\_page.dart

models/note.dart

## 4.3. Модель данных

lib/models/note.dart

class Note {

final String id;

String title;

String body;

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

Note copyWith({String? title, String? body}) => Note(

id: id,

title: title ?? this.title,

body: body ?? this.body,

);

}

## 4.4. Главный экран со списком

lib/main.dart

import 'package:flutter/material.dart';

import 'models/note.dart';

import 'edit\_note\_page.dart';

void main() => runApp(const SimpleNotesApp());

class SimpleNotesApp extends StatelessWidget {

const SimpleNotesApp({super.key});

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Simple Notes',

theme: ThemeData(useMaterial3: true),

home: const NotesPage(),

);

}

}

class NotesPage extends StatefulWidget {

const NotesPage({super.key});

@override

State<NotesPage> createState() => \_NotesPageState();

}

class \_NotesPageState extends State<NotesPage> {

final List<Note> \_notes = [

Note(id: '1', title: 'Пример', body: 'Это пример заметки'),

];

Future<void> \_addNote() async {

final newNote = await Navigator.push<Note>(

context,

MaterialPageRoute(builder: (\_) => EditNotePage()),

);

if (newNote != null) {

setState(() => \_notes.add(newNote));

}

}

Future<void> \_edit(Note note) async {

final updated = await Navigator.push<Note>(

context,

MaterialPageRoute(builder: (\_) => EditNotePage(existing: note)),

);

if (updated != null) {

setState(() {

final i = \_notes.indexWhere((n) => n.id == updated.id);

if (i != -1) \_notes[i] = updated;

});

}

}

void \_delete(Note note) {

setState(() => \_notes.removeWhere((n) => n.id == note.id));

ScaffoldMessenger.of(context).showSnackBar(

const SnackBar(content: Text('Заметка удалена')),

);

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: const Text('Simple Notes')),

floatingActionButton: FloatingActionButton(

onPressed: \_addNote,

child: const Icon(Icons.add),

),

body: \_notes.isEmpty

? const Center(child: Text('Пока нет заметок. Нажмите +'))

: ListView.builder(

itemCount: \_notes.length,

itemBuilder: (context, i) {

final note = \_notes[i];

return ListTile(

key: ValueKey(note.id),

title: Text(note.title.isEmpty ? '(без названия)' : note.title),

subtitle: Text(

note.body,

maxLines: 1,

overflow: TextOverflow.ellipsis,

),

onTap: () => \_edit(note),

trailing: IconButton(

icon: const Icon(Icons.delete\_outline),

onPressed: () => \_delete(note),

),

);

},

),

);

}

}

## 4.5. Экран добавления/редактирования

lib/edit\_note\_page.dart

import 'package:flutter/material.dart';

import 'models/note.dart';

class EditNotePage extends StatefulWidget {

final Note? existing;

const EditNotePage({super.key, this.existing});

@override

State<EditNotePage> createState() => \_EditNotePageState();

}

class \_EditNotePageState extends State<EditNotePage> {

final \_formKey = GlobalKey<FormState>();

late String \_title = widget.existing?.title ?? '';

late String \_body = widget.existing?.body ?? '';

void \_save() {

if (!\_formKey.currentState!.validate()) return;

\_formKey.currentState!.save();

final result = (widget.existing == null)

? Note(

id: DateTime.now().millisecondsSinceEpoch.toString(),

title: \_title,

body: \_body,

)

: widget.existing!.copyWith(title: \_title, body: \_body);

Navigator.pop(context, result);

}

@override

Widget build(BuildContext context) {

final isEdit = widget.existing != null;

return Scaffold(

appBar: AppBar(title: Text(isEdit ? 'Редактировать' : 'Новая заметка')),

body: Padding(

padding: const EdgeInsets.all(16),

child: Form(

key: \_formKey,

child: Column(

children: [

TextFormField(

initialValue: \_title,

decoration: const InputDecoration(labelText: 'Заголовок'),

onSaved: (v) => \_title = v!.trim(),

),

const SizedBox(height: 12),

TextFormField(

initialValue: \_body,

decoration: const InputDecoration(labelText: 'Текст'),

minLines: 3,

maxLines: 6,

onSaved: (v) => \_body = v!.trim(),

validator: (v) => (v == null || v.trim().isEmpty)

? 'Введите текст заметки'

: null,

),

const Spacer(),

FilledButton.icon(

onPressed: \_save,

icon: const Icon(Icons.check),

label: const Text('Сохранить'),

),

],

),

),

),

);

}

}

D bnjut

## Контрольные задания

* Отобразить список заметок с использованием собственного дизайна.
* Кнопка **+** открывает экран создания заметки; после сохранения новая заметка появляется в списке.
* Тап по заметке открывает экран редактирования; изменения возвращаются на список.
* Удаление заметки по кнопке корзины в trailing.
* Свайп-удаление через Dismissible.
* Поле поиска в AppBar с фильтрацией по заголовку.

# Требования к отчёту

**Структура:**

* 1. Номер ПЗ, ФИО, группа
  2. Цели ПЗ
  3. Ход работы (шаги + 3–5 ключевых фрагментов кода с краткими пояснениями).
  4. Скриншоты: список, создание, редактирование, удаление. (по возможности видео или GIF)
  5. Выводы (что получилось/что было сложным).

# Контрольные вопросы

1. Зачем использовать ListView.builder вместо ListView(children: [...])?
2. Как передать объект на новый экран и вернуть обновлённый объект обратно?
3. Чем помогают Key в списках, и какой ключ выбрать для элементов?
4. Как реализовать удаление элемента самым простым способом?
5. Что произойдёт, если не вызвать setState после редактирования элемента?
6. Как быстро сгенерировать уникальный id без внешних пакетов?