МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практической работе №7

по дисциплине «Разработка кроссплатформенных мобильных приложений»

Выполнил:
Студент группы ИКБО-07-22 Савин А.С.

Проверил:
Старший преподаватель кафедры МОСИТ Шешуков Л.С.

Москва 2025 г.

Задачи практической работы:

Требуется выполнить следующие пункты:

— страничная навигация в приложениях;

— основные методы страничной навигации;

— навигация на маршрутах;

— выполнение практического задания;

1. Страничная навигации в приложении

В Flutter есть два основных способа навигации в приложении: страничная и маршрутизированная навигация. Страничная навигация работает на Navigator и Route. Navigator – это виджет, который управляет набором дочерних виджетов с помощью структуры Stack. Экраны в Flutter называются Route.

Многие приложения имеют Navigator в верхней части иерархии виджетов, чтобы отображать их логическую историю с помощью наложения с последними посещенными страницами, визуально поверх старых страниц. Использование этого шаблона позволяет навигатору визуально переходить с одной страницы на другую, перемещая виджеты в наложении. Аналогично, навигатор можно использовать для отображения диалога, разместив виджет диалога над текущей страницей.

В Flutter существует два основных подхода к организации навигации между экранами: Navigation 1.0 и Navigation 2.0. Navigation 1.0 представляет собой императивный способ управления навигацией, при котором разработчик вручную вызывает методы Navigator.push() и Navigator.pop() для перехода между маршрутами. Этот подход прост в освоении и подходит для небольших мобильных приложений, однако он не обеспечивает синхронизации состояния навигации с URL, что делает его неудобным для веб-платформ и ограничивает возможности по работе с deep links и восстановлением состояния.

Navigation 2.0, представленный в Flutter 2.0, реализует декларативный подход к навигации и основан на использовании компонентов Router, RouterDelegate и RouteInformationParser. Он позволяет полностью синхронизировать текущее состояние приложения с адресной строкой браузера, что особенно важно для веб-приложений, а также обеспечивает поддержку глубоких ссылок, корректную работу кнопок «назад» и «вперёд» в браузере и возможность восстановления состояния после перезагрузки. Хотя

реализация Navigation 2.0 вручную требует значительных усилий, на практике разработчики чаще используют такие пакеты, как go_router или auto_route, которые упрощают работу с декларативной навигацией, сохраняя при этом все её преимущества.

2. Основные методы страничной навигации

Для исполнения основных действий навигации Navigator предоставляет ряд методов для работы. Рассмотрим основные из них:

2.1 Метод push

В навигации часто требуется переходить между страницами с возможностью к ним в последующем вернуться. Данный способ навигации называется – вертикальной навигацией. Смысл вертикальной навигации заключается в сохранении состояний экранов, с которой был выполнен навигационный переход. По сути, навигационная структура реализуется на основе структуры Stack и вертикальная навигация – это добавление новой страницы в него. Для реализации добавления новой страницы в навигационный стек или реализации вертикального навигационного перехода класс Navigator предоставляет метод push. Сигнатура метода продемонстрирована на рисунке 1. Метод принимает в себя навигационную страницу, на которую требуется выполнить вертикальный навигационный переход.

```
2305    @optionalTypeArgs
2306    static Future<T?> push<T extends Object?>(BuildContext context, Route<T> route) {
2307      return Navigator.of(context).push(route);
2308    }
```

Рисунок 1 – Сигнатура метода push

Описание параметров:

— BuildContext context — контекст, через который извлекается Navigator.

— Route<T> route — маршрут, который будет добавлен в стек навигации.

— Возвращаемое значение: Future<T?> — завершится, когда добавленный маршрут будет удалён из стека, с возможным возвращаемым значением типа T.

2.2 Метод pop

В вертикальной навигации помимо самого перехода так же есть и обратное действие – вертикальный навигационный возврат. Когда мы добавляем страницу в навигационный Stack у пользователя должна быть возможность вернуться на страницу назад для просмотра предыдущей страницы. Для реализации возврата на предыдущую страницу в навигационном стеке или реализации вертикального навигационного возврата класс Navigator предоставляет метод pop.

Сигнатура метода продемонстрирована на рисунке 2. Метод может принимать в себя какие-либо данные, которые могут считаться в качестве результата навигационного перехода.

```
/// void _accept() {
///   Navigator.pop(context, true); // dialog returns true
/// }
/// ```
@optionalTypeArgs
static void pop<T extends Object?>(BuildContext context, [T? result]) {
  Navigator.of(context).pop<T>(result);
}
```

Рисунок 2 – Сигнатура метода pop

Описание параметров:

— BuildContext context — контекст, используемый для получения Navigator.

— [T? result] (необязательный) — значение, которое будет возвращено маршруту, ожидающему результата вызова push или pushReplacement.

— Возвращаемое значение: void — метод ничего не возвращает, но завершает текущий маршрут.

2.3 Метод pushReplacement

Часто, в ходе работы приложения только вертикальной навигации бывает недостаточно. Иногда в приложениях реализуется логика, когда пользователь должен перейти на страницу и при этом ему не нужна предыдущая страница в истории навигации. Такая логика часто реализуется в нижних и боковых меню, когда переход на самих реализован с сохранением навигационной истории, то есть вертикальной навигацией, а вот переходы между самими позициями в меню реализуются без сохранения предыдущего выбранного элемента меню. Данный способ навигации называется – горизонтальная навигация. Ее отличительным признаком является не наложение новой страницы на навигационный Stack системы, а снятием верхней страницы из него и заменой ее на новую, которую предоставляет пользователь горизонтальным переходом. Для реализации горизонтального навигационного перехода класс Navigator предоставляет метод pushReplacement. Сигнатура метода продемонстрирована на рисунке 3. Метод принимает в себя навигационную страницу, на которую требуется выполнить горизонтальный навигационный переход.

```
2396    @optionalTypeArgs
2397    static Future<T?> pushReplacement<T extends Object?, TO extends Object?>(
2398      BuildContext context,
2399      Route<T> newRoute, {
2400      TO? result,
2401    }) {
2402      return Navigator.of(context).pushReplacement<T, TO>(newRoute, result: result);
2403    }
```

Рисунок 3 – Сигнатура метода pushReplacement

Описание параметров:

— BuildContext context — контекст, используемый для поиска экземпляра Navigator, через который выполняется переход.

— Route<T> newRoute — маршрут, который будет добавлен вместо текущего.

— {TO? result} (необязательный) — результат, который будет передан предыдущему маршруту, если текущий удаляется.

— Возвращаемое значение: Future<T?> — завершится, когда новый маршрут будет удалён из стека, с возможным возвращаемым значением типа T.

3. Навигация на маршрутах

Если страничная навигация работает на передаваемых страницах в момент вызова методов push и pushReplacement, то маршрутизированная навигация более требовательная к вопросу начальной инициализации. Маршрутизированная навигация в фреймворке Flutter реализована абстрактным слоем функциональности, которую разработчик должен реализовать под приложение самостоятельно. В данном случае использовался пакет go_router — современное, официально поддерживаемое решение, которое упрощает реализацию Navigation 2.0 за счёт декларативного описания маршрутов, поддержки параметров, вложенной навигации и глубоких ссылок, а также обеспечивает корректную интеграцию с веб-платформой. Добавление go_router версии 14.2.7 в проект показано на рисунке 4.

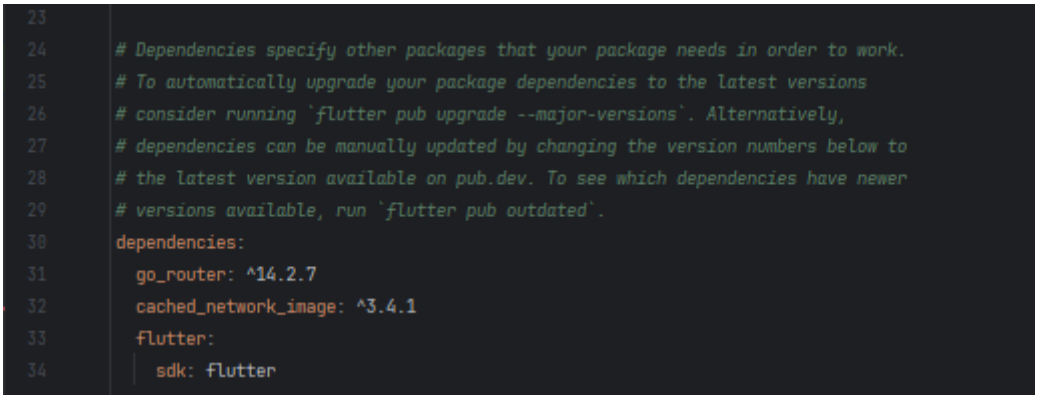


Рисунок 4 – Добавление go_router в проект

3.1 Маршрутная карта

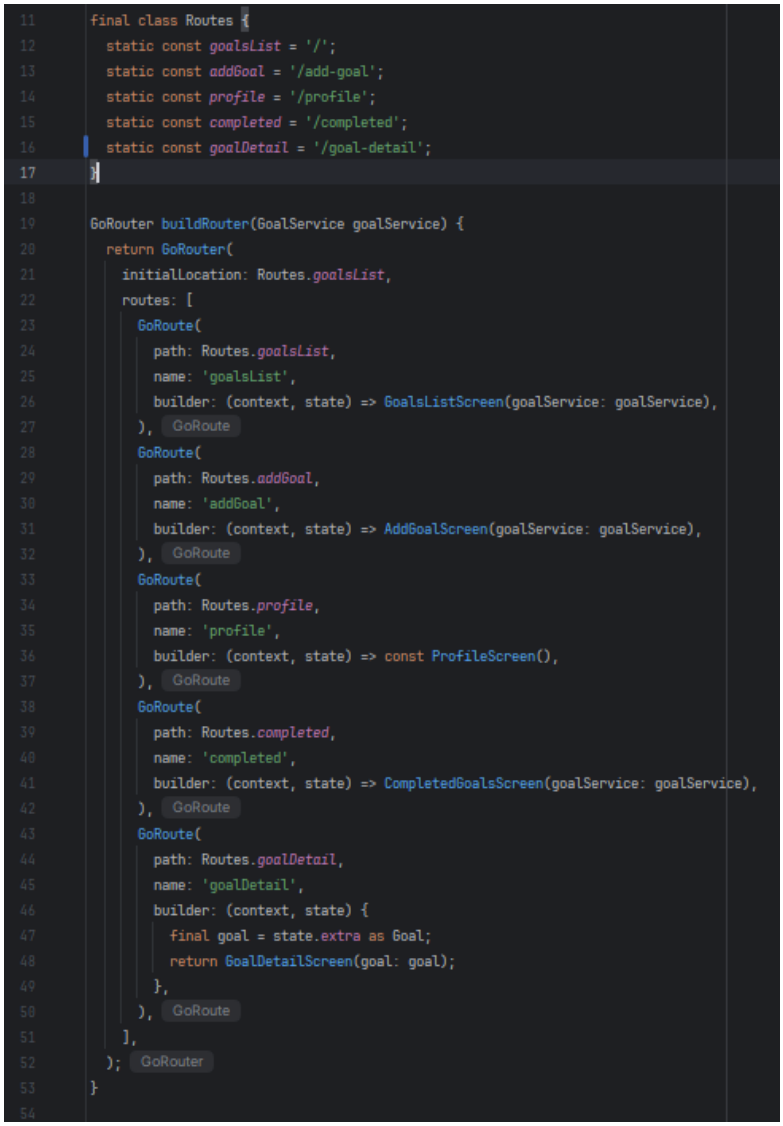
Маршрутная карта в приложении используется для определения навигационных маршрутов, которые разработчик может использовать при переходах между экранами. Она формируется на этапе инициализации приложения и не изменяется в ходе его работы. Маршрутная карта может содержать маршруты разных уровней и вложенности.

Маршрутная карта в GoRouter строится как список объектов GoRoute.

Каждый маршрут определяет:

- path: Строковый шаблон пути (например, '/details' или '/user/:id').
- builder: Функция, которая возвращает виджет-экран для данного пути. Функция получает контекст и состояние маршрута, что позволяет извлекать переданные параметры.
- routes: Список дочерних маршрутов, что позволяет создавать вложенные навигационные структуры.

В данном проекте маршрутная карта реализована в файле app_router.dart, где описаны маршруты всех пяти экранов приложения. Кодовая реализация маршрутной карты представлена на рисунке 5.



```dart
final class Routes {
  static const goalsList = '/';
  static const addGoal = '/add-goal';
  static const profile = '/profile';
  static const completed = '/completed';
  static const goalDetail = '/goal-detail';
}

GoRouter buildRouter(GoalService goalService) {
  return GoRouter(
    initialLocation: Routes.goalsList,
    routes: [
      GoRoute(
        path: Routes.goalsList,
        name: 'goalsList',
        builder: (context, state) => GoalsListScreen(goalService: goalService),
      ), GoRoute
      GoRoute(
        path: Routes.addGoal,
        name: 'addGoal',
        builder: (context, state) => AddGoalScreen(goalService: goalService),
      ), GoRoute
      GoRoute(
        path: Routes.profile,
        name: 'profile',
        builder: (context, state) => const ProfileScreen(),
      ), GoRoute
      GoRoute(
        path: Routes.completed,
        name: 'completed',
        builder: (context, state) => CompletedGoalsScreen(goalService: goalService),
      ), GoRoute
      GoRoute(
        path: Routes.goalDetail,
        name: 'goalDetail',
        builder: (context, state) {
          final goal = state.extra as Goal;
          return GoalDetailScreen(goal: goal);
        },
      ), GoRoute
    ],
  ); GoRouter
}
```

Рисунок 5 – Маршрутная карта

3.2 Делегат навигации

После того как маршрутная карта создана, ее необходимо подключить в наше приложение. Для этого используется специализированный конструктор Widget-а приложения – MaterialApp.router. В нем необходимо или по отдельности задать делегат навигации в приложение, или в целом задать конфигурацию навигации, как показано на рисунке 6.

```
void main() {
  final goalService = GoalService();
  runApp(MyApp(goalService: goalService));
}

class MyApp extends StatelessWidget {
  final GoalService goalService;
  const MyApp({super.key, required this.goalService});

  @override
  Widget build(BuildContext context) {
    final router = buildRouter(goalService);
    return MaterialApp.router(
      debugShowCheckedModeBanner: false,
      title: 'Goals',
      theme: ThemeData(useMaterial3: true, colorSchemeSeed: Colors.indigo),
      routerConfig: router,
    );  MaterialApp.router
  }
}
```

Рисунок 6 – Конфигурирование навигации в приложении

После того, как навигационная конфигурация в приложении добавлена, мы можем получить доступ к навигационному делегату в любом месте приложения, где у нас есть доступ к контексту нашего приложения. При обращении к навигационному делегату можно использовать ранее рассмотренные методы push, pop и pushReplacement, однако теперь они в качестве аргументов будут принимать не страницу навигации, а маршрут до требуемой страницы. Пример горизонтального навигационного перехода с главного экрана на другие в маршрутизированной навигации при помощи пакета GoRouter продемонстрирован на рисунке 7.

Рисунок 7 – Реализация горизонтального перехода

4. Выполнение практической работы

В рамках выполнения практической работы была проведена комплексная работа по реализации и тестированию системы навигации в мобильном приложении.

4.1 Реализация страничной навигации в проекте

Вертикальная страничная навигация

В разработанном приложении вертикальная навигация реализована при переходе от экрана списка целей к экрану добавления новой цели при помощи метода Navigator.push(). На экране списка целей пользователь видит список целей, строку поиска, статистическую карточку и кнопку «плюс» (FAB). При нажатии на FAB открывается экран создания цели.

Реализация вертикальной страничной навигации в файле goals_list_screen.dart показана на рисунке 8. Демонстрация навигации — на рисунках 9–10.

```
      floatingActionButton: FloatingActionButton.extended(
        onPressed: _addGoal,
        label: const Text('Новая цель'),
        icon: const Icon(Icons.add),
      ), FloatingActionButton.extended
    ); Scaffold
  }

  void _addGoal() {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (_) => AddGoalScreen(goalService: _goalService),
      ), MaterialPageRoute
    );
  }
}
```

Рисунок 8 – Реализация вертикального страничного перехода на экран добавления цели

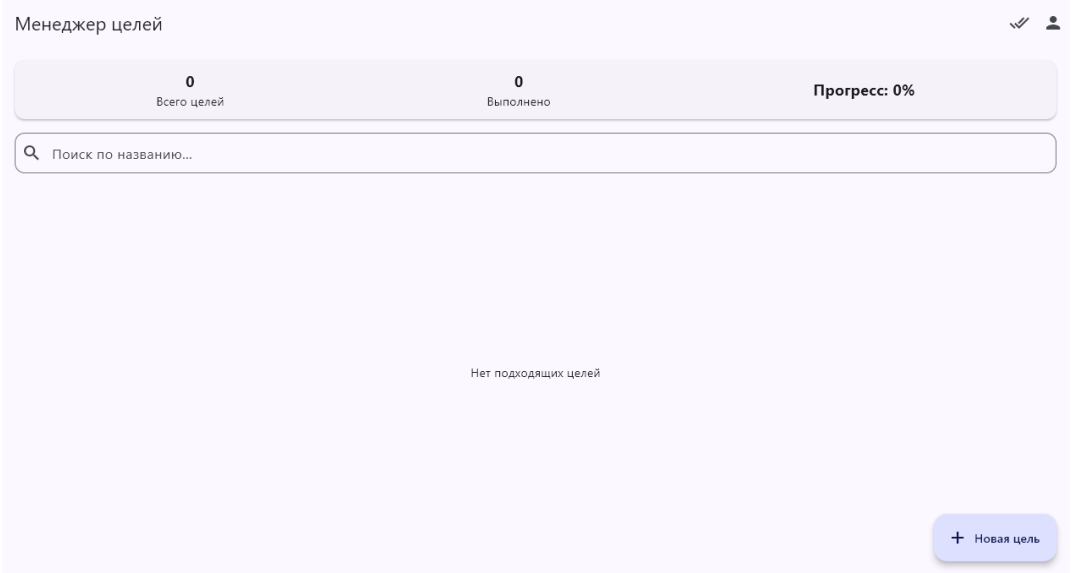


Рисунок 9 – Экран списка целей до выполнения вертикальной навигации на экран добавления цели

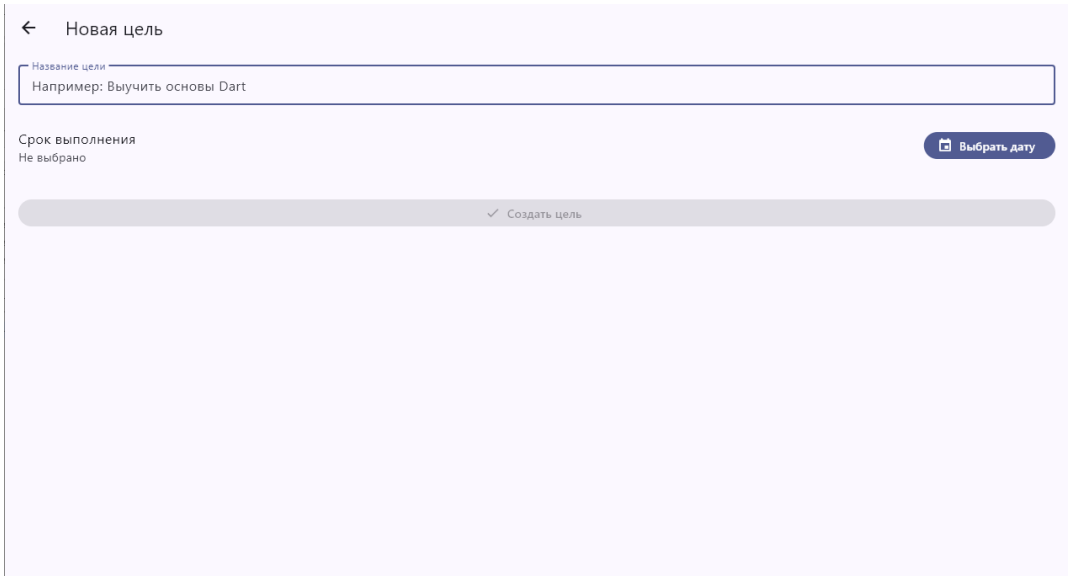


Рисунок 10 – Экран формы добавления цели после выполнения вертикальной навигации

Пользователь может вернуться, нажав на стрелку «Назад» в AppBar. Вертикальная навигация назад реализована при переходе от экрана добавления цели обратно к списку целей при помощи метода Navigator.pop(). На экране добавления цели с помощью кнопки в виде иконки стрелки влево, расположенной в левом верхнем углу. Реализация возврата в файле add_goal_screen.dart показана на рисунке 11. Демонстрация навигации — на рисунках 12–13.

```
return Scaffold(
  appBar: AppBar(
    title: const Text('Новая цель'),
    leading: BackButton(onPressed: () => Navigator.pop(context)),
  ), AppBar
```

Рисунок 11 – Реализация метода pop() на экране добавления цели

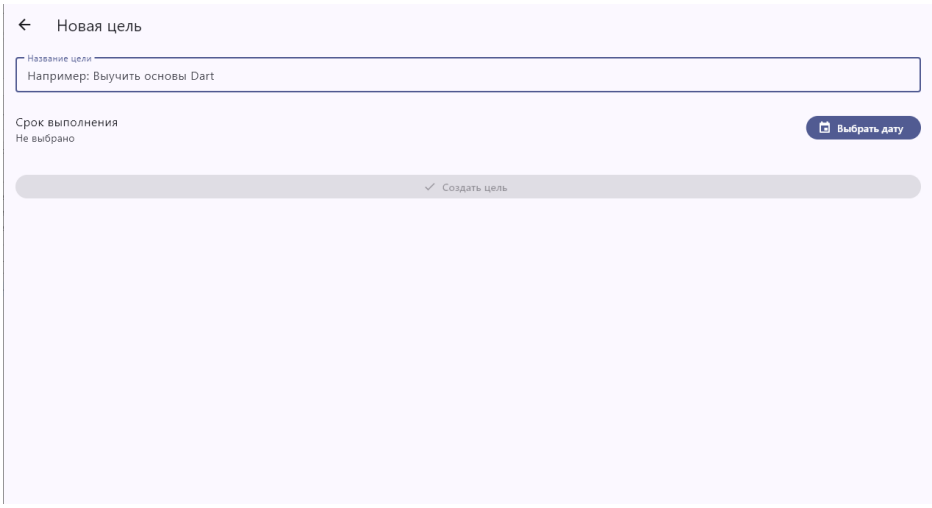


Рисунок 12 - Экран формы добавления цели до нажатия кнопки «Назад»

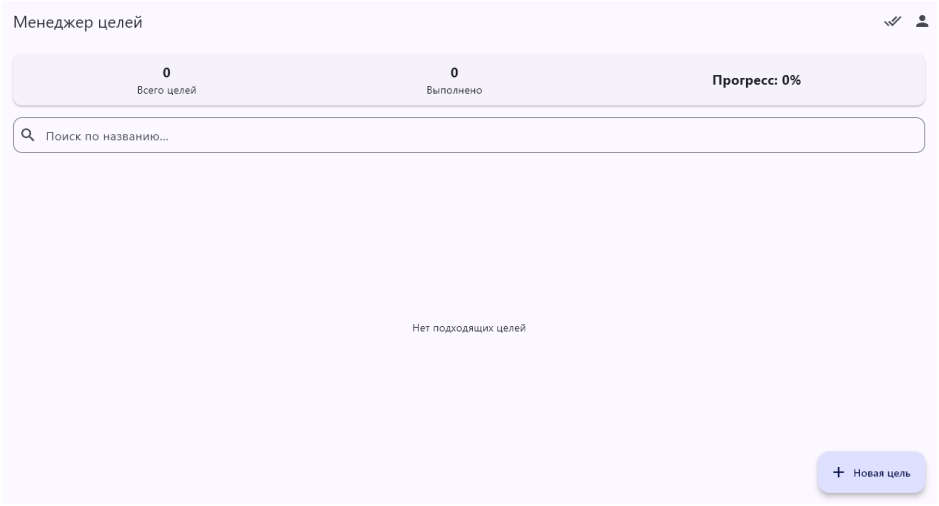


Рисунок 13 - Экран списка целей после выполнения нажатия на кнопку возврата

Реализация перехода на страницу профиля в файле goals_list_screen.dart показана на рисунке 14. Демонстрация работы навигации приведена на рисунках 15–16.

```
IconButton(
  tooltip: 'Профиль',
  icon: const Icon(Icons.person),
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (_) => const ProfileScreen()),
    );
  },
),
```

Рисунок 14 – Реализация вертикального страничного перехода на страницу профиля

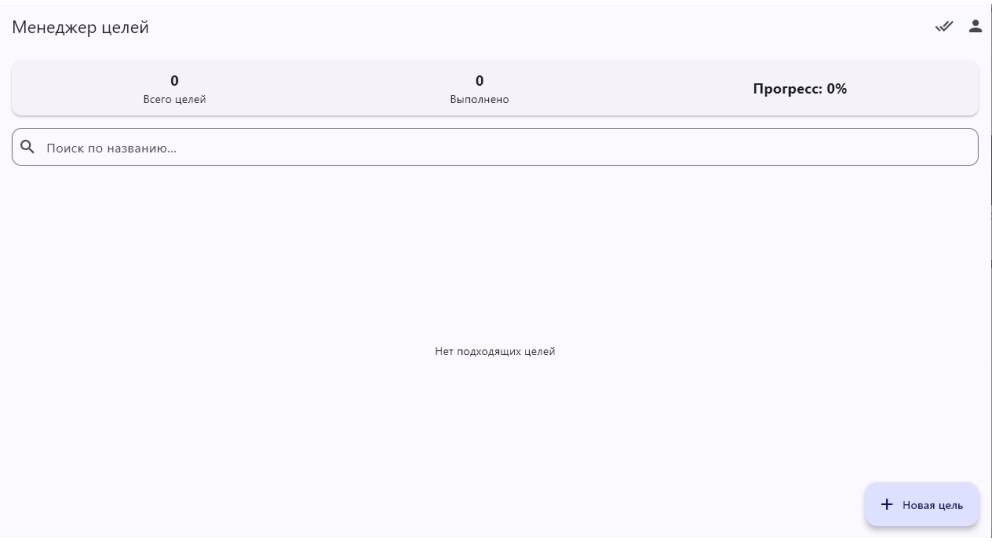


Рисунок 15 – Экран списка целей до выполнения вертикальной навигации на экран профиля

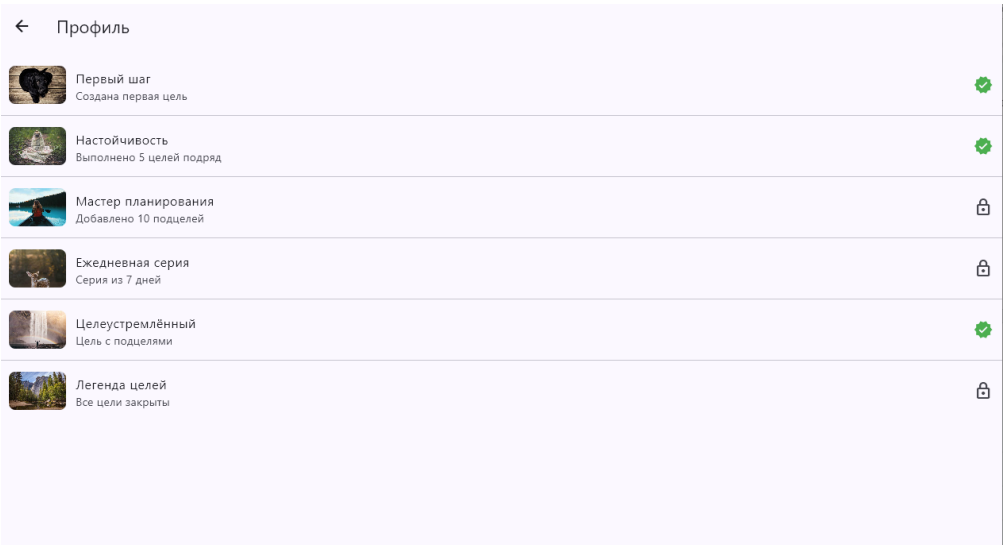


Рисунок 16 – Экран профиля после выполнения вертикальной навигации

Пользователь может вернуться обратно к списку целей, нажав на кнопку «Назад» в верхней панели приложения. Вертикальная навигация назад реализована с помощью метода context.pop(), который закрывает текущий экран профиля и возвращает пользователя на предыдущий экран — список целей.

Реализация возврата назад показана на рисунке 17. Демонстрация перехода обратно — на рисунках 18–19.

```
return Scaffold(
  appBar: AppBar(
    title: Text(goal.title),
    leading: BackButton(onPressed: () => Navigator.pop(context)),
  ), AppBar
```

Рисунок 17 – Реализация возврата назад

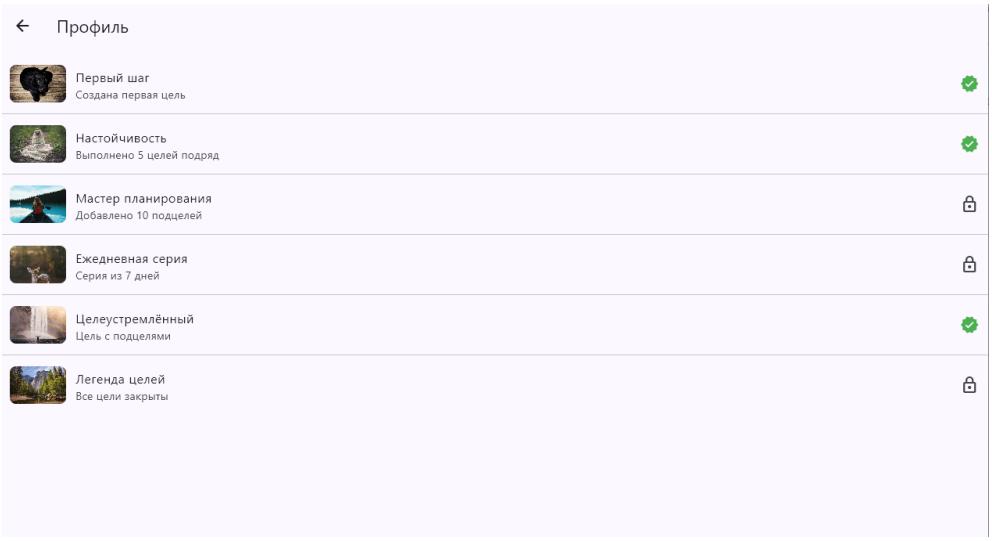


Рисунок 18 – Экран профиля до нажатия на кнопку возврата

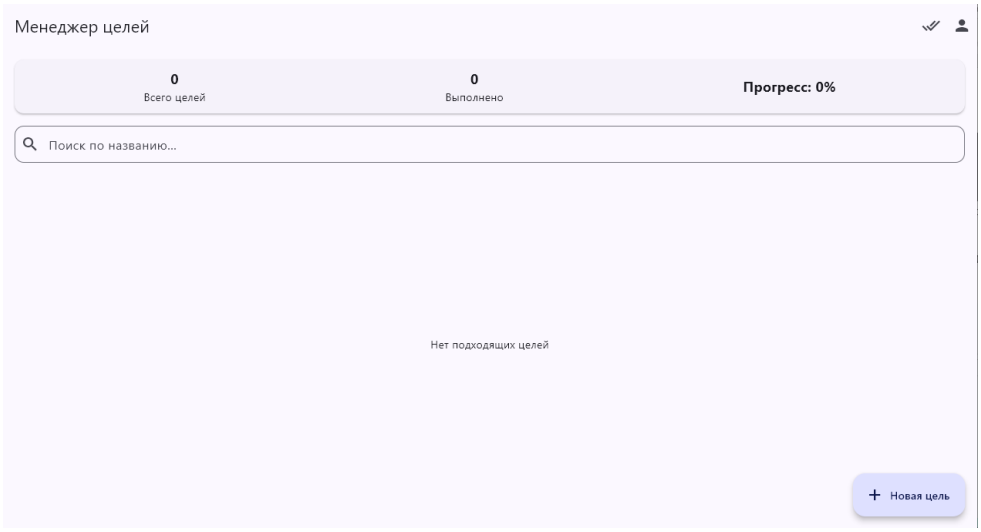


Рисунок 19 – Экран списка целей после выполнения нажатия на кнопку возврата

Реализация перехода на страницу выполненных целей в файле goals_list_screen.dart показана на рисунке 20. Демонстрация процесса перехода представлена на рисунках 21–22.

```
IconButton(
  tooltip: 'Выполненные',
  icon: const Icon(Icons.done_all),
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (_) => CompletedGoalsScreen(goalService: _goalService),
      ),
    );
```

Рисунок 20 – Реализация вертикального страничного перехода на страницу выполненных целей

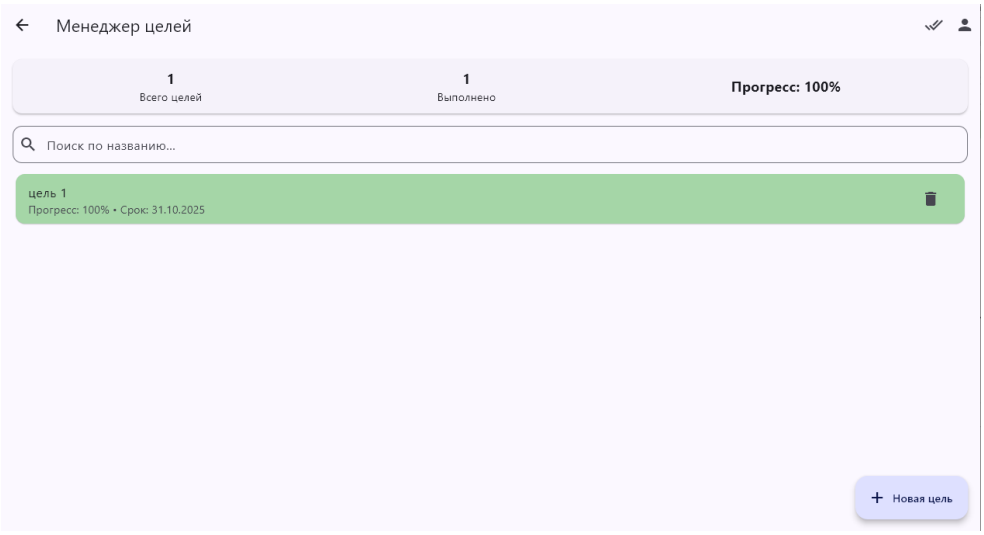


Рисунок 21 – Экран списка целей до выполнения вертикальной навигации на экран выполненных целей

Рисунок 22 – Экран выполненных целей после выполнения вертикальной навигации

Возврат на главный экран осуществляется с помощью кнопки «Назад» в верхней панели, реализованной методом context.pop(). После нажатия на кнопку приложение закрывает текущий экран выполненных целей и возвращает пользователя к списку всех целей.

Реализация возврата назад показана на рисунке 23. Демонстрация перехода обратно — на рисунках 24–25.

```
return Scaffold(
  appBar: AppBar(
    title: Text(goal.title),
    leading: BackButton(onPressed: () => Navigator.pop(context)),
  ), AppBar
```

Рисунок 23 – Реализация возврата назад

Рисунок 24 – Экран выполненных целей до выполнения нажатия на кнопку возврата

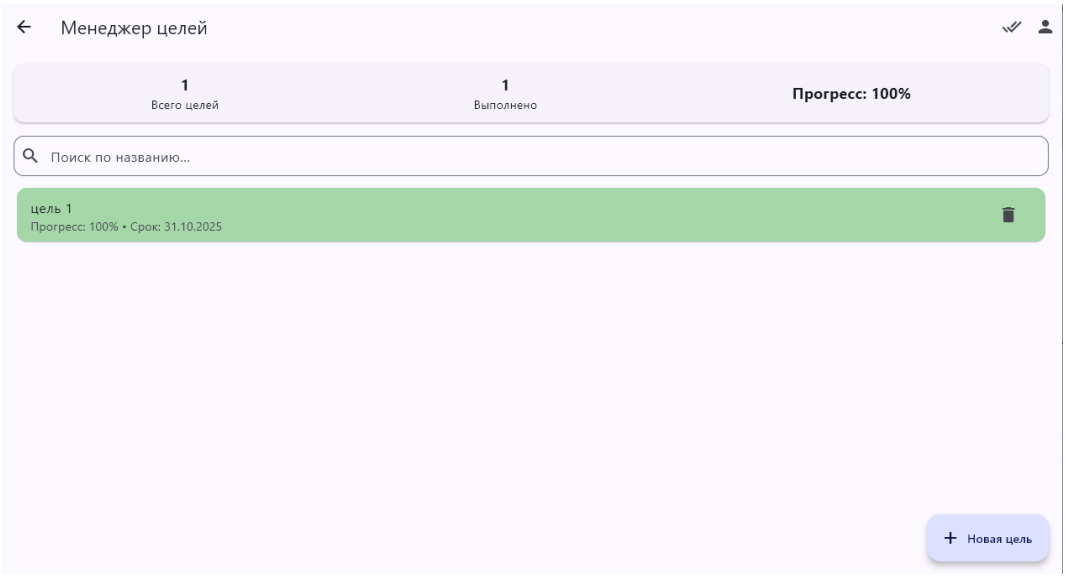


Рисунок 25 – Экран списка целей после выполнения нажатия на кнопку возврата

Реализация перехода на страницу подзадач цели в файле goals_list_screen.dart показана на рисунке 26. Демонстрация процесса перехода представлена на рисунках 27–28.

```
onTap: (goal) async {
  await Navigator.push(
    context,
    MaterialPageRoute(builder: (_) => GoalDetailScreen(goal: goal)),
  );
  setState(() {});
},
```

Рисунок 26 – Реализация вертикального страничного перехода на страницу подзадач цели

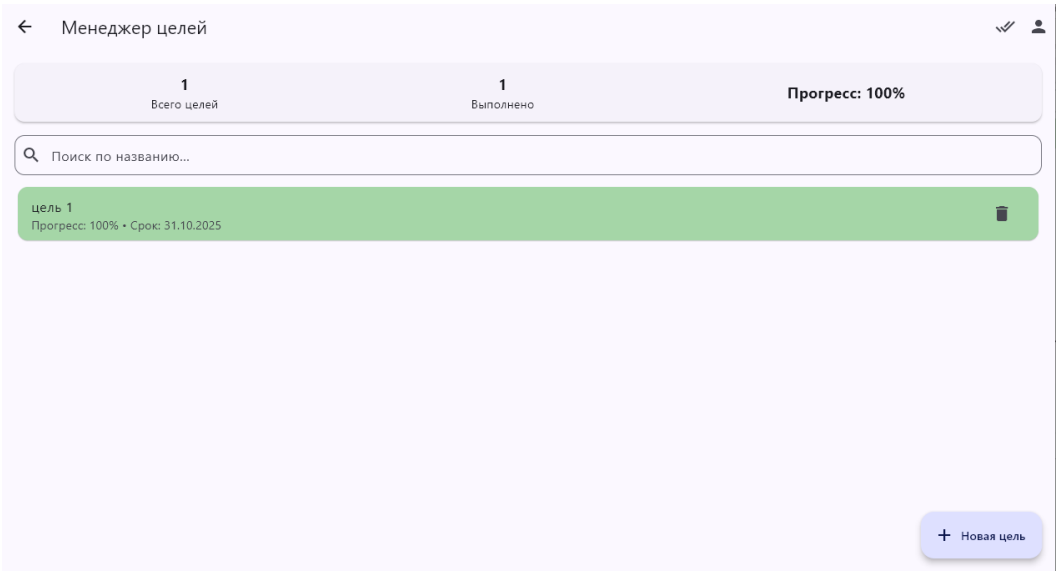


Рисунок 27 – Экран списка целей до выполнения вертикальной навигации на экран подзадач цели

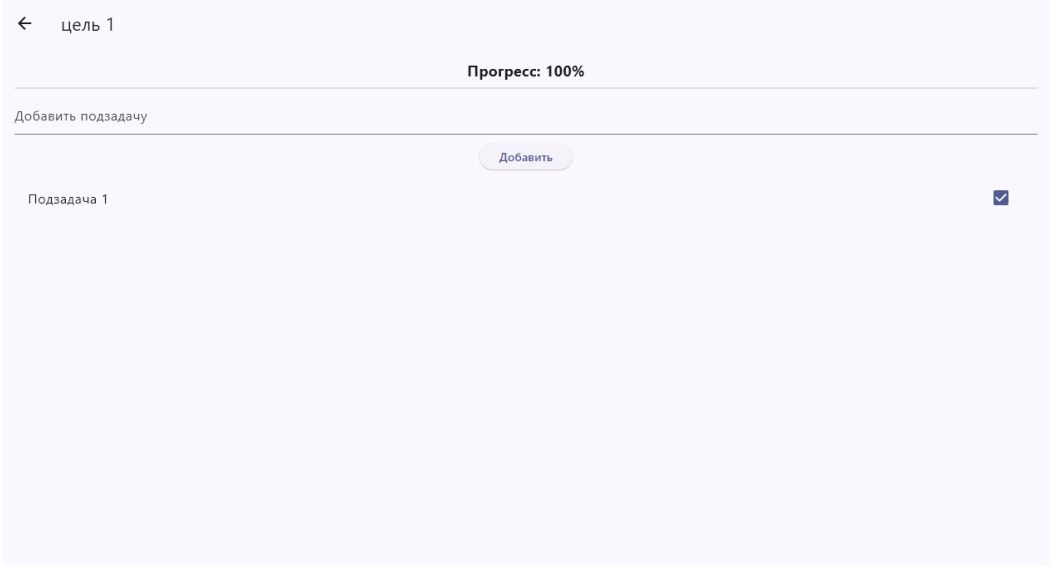


Рисунок 28 – Экран подзадач цели после выполнения вертикальной навигации

Возврат к списку целей осуществляется через кнопку «Назад» в верхней панели экрана. Навигация назад реализована при помощи метода context.pop(), который закрывает текущий экран подзадач и возвращает пользователя к предыдущему состоянию — списку всех целей.

Реализация возврата назад показана на рисунке 29. Демонстрация процесса возврата приведена на рисунках 30–31.

```
return Scaffold(
  appBar: AppBar(
    title: Text(goal.title),
    leading: BackButton(onPressed: () => Navigator.pop(context)),
  ), AppBar
```

Рисунок 29 – Реализация возврата назад

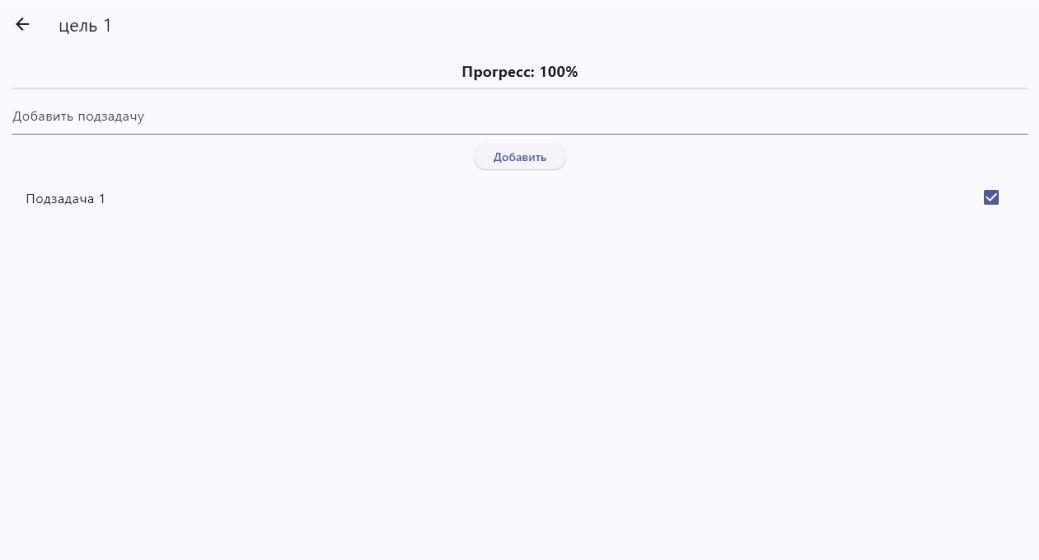


Рисунок 30 – Экран подзадач цели до выполнения нажатия на кнопку возврат

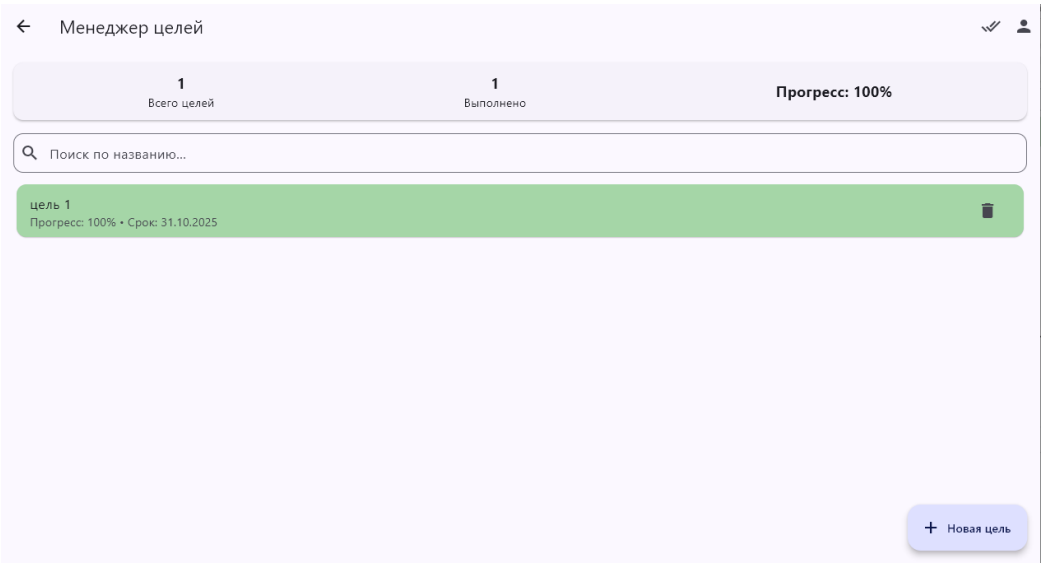


Рисунок 31 – Экран списка целей после выполнения нажатия на кнопку возврата

Горизонтальная страничная навигация

Горизонтальная страничная навигация реализована при помощи Navigator.pushReplacement() на экране добавления цели. После нажатия кнопки «Создать цель» выполняется навигационный переход с полной заменой текущего экрана на экран списка целей. Благодаря этому список сразу отображает добавленную цель без сохранения экрана добавления в стеке. Реализация показана на рисунке 32. Демонстрация — на рисунках 33–34. Ключевая особенность перехода — отсутствие возможности вернуться на предыдущий экран (экран добавления) по системной кнопке «Назад», так как он заменён.

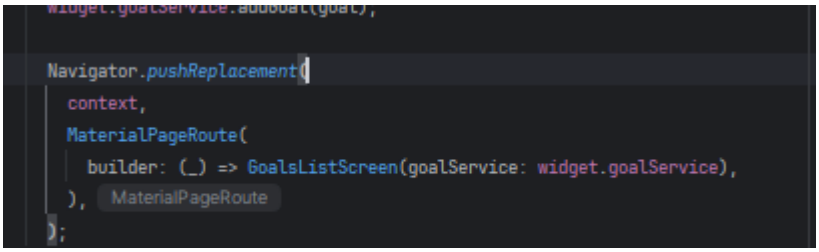


Рисунок 32 - Реализация горизонтальной страничной навигации с помощью метода pushReplacement

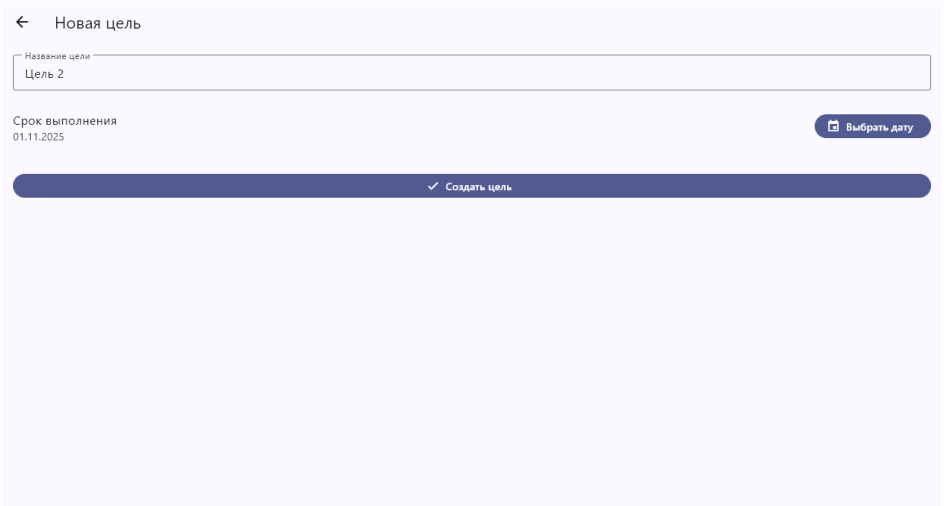


Рисунок 33 – Экран формы добавления цели до нажатия кнопки «Создать цель»

Рисунок 34 – Главный экран после выполнения горизонтальной страничной навигации с заменой экрана в стеке

4.2 Реализация маршрутизированной навигации в проекте

Для реализации маршрутизированной навигации в проекте был использован пакет go_router версии 14.2.7 — актуальная стабильная версия на момент разработки, предоставляющая удобный и современный API для декларативного описания маршрутов (рисунок 35). Выбор данной библиотеки обусловлен её надёжностью, совместимостью с последними обновлениями Flutter, а также активной поддержкой сообщества разработчиков. Добавление зависимости в проект представлено на рисунке.

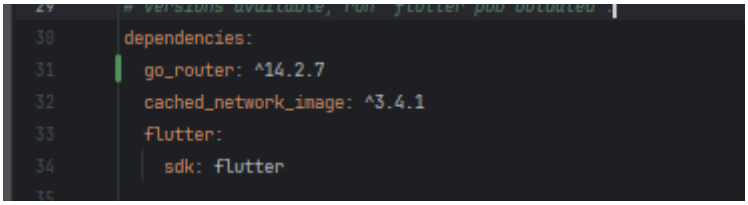


Рисунок 35 – Добавление зависимости go_router в файл pubspec.yaml

В данном приложении маршруты всех экранов определены в файле app_router.dart, который содержит основную карту навигации. В этом файле задаются пути ко всем основным разделам приложения и их взаимосвязи. Структура маршрутной карты показана на рисунке 36.

Рисунок 36 – Заполненная маршрутная карта

Используется MaterialApp.router в main.dart (рисунок 37).

```dart
class MyApp extends StatelessWidget {
  MyApp({super.key});

  final GoRouter _router = AppRouter().router;
  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      routerConfig: _router,
      theme: AppTheme.lightTheme,
    );
  }
}
```

Рисунок 37 – Встраивание MaterialApp.router

Вертикальная маршрутизированная навигация

Вертикальная маршрутизированная навигация реализована с использованием декларативных маршрутов GoRouter. Такой подход обеспечивает сохранение стека экранов и позволяет пользователю возвращаться на предыдущие страницы при помощи метода context.pop()

Реализация вертикального маршрутизированного перехода: Список целей → Добавление цели, при помощи context.push(Routes.addGoal) (рисунок 38). Процесс навигации можно увидеть на рисунках 39 – 40.

```dart
      ), Padding
      floatingActionButton: FloatingActionButton.extended(
        onPressed: _addGoal,
        label: const Text('Новая цель'),
        icon: const Icon(Icons.add),
      ), FloatingActionButton.extended
    ); Scaffold
  }

  void _addGoal() {
    context.push(Routes.addGoal);
  }
}
```

Рисунок 38 – Реализация вертикального маршрутизированного перехода на экран добавления цели

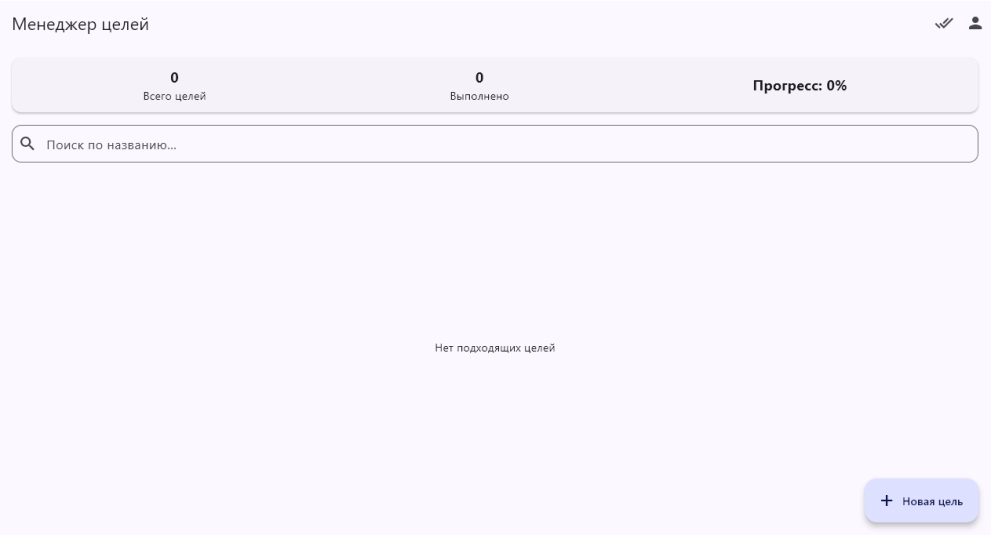


Рисунок 39 – Главный экран приложения до выполнения вертикальной маршрутизированной навигации на экран добавления цели

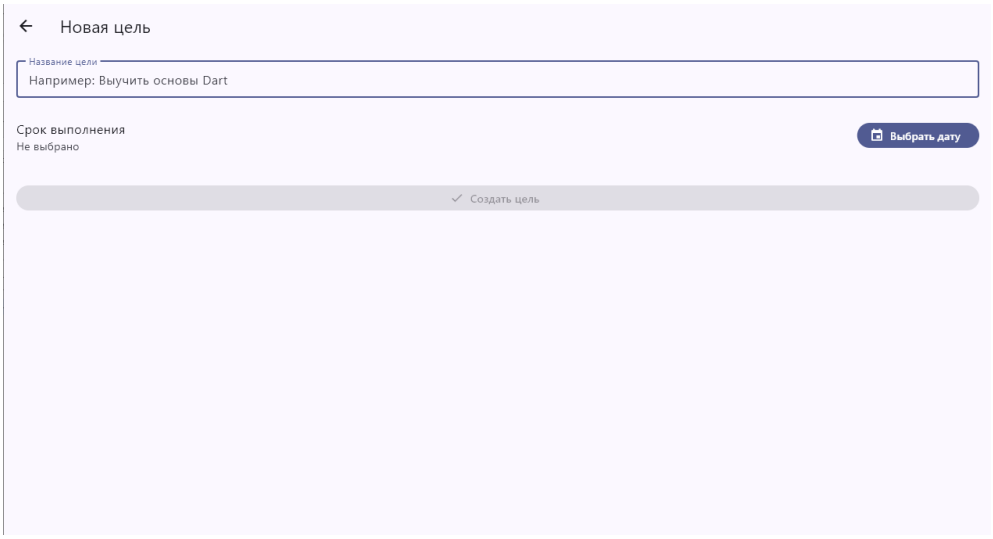


Рисунок 40 – Экран добавления цели после выполнения вертикальной маршрутизированной навигации

Реализация вертикального маршрутизированного перехода назад при помощи context.pop() (рисунок 41). Демонстрация навигации — на рисунках 42–43.

```
@override
Widget build(BuildContext context) {
  final canSave = _titleController.text.trim().isNotEmpty && _deadline != null;

  return Scaffold(
    appBar: AppBar(
      title: const Text('Новая цель'),
      leading: BackButton(onPressed: () => context.pop()),
    ), AppBar
    body: Form(
```

Рисунок 41 – Реализация метода pop()

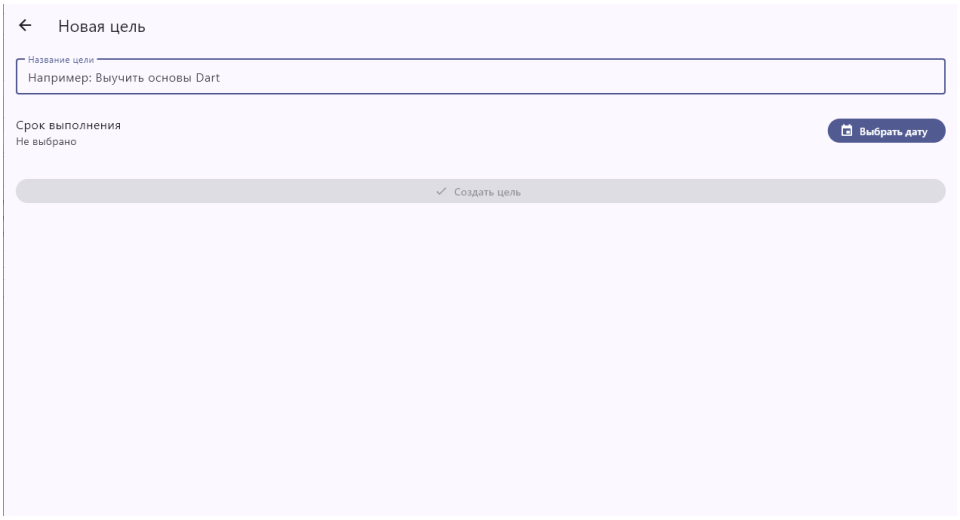


Рисунок 42 – Экран добавления цели до нажатия кнопки «Назад»

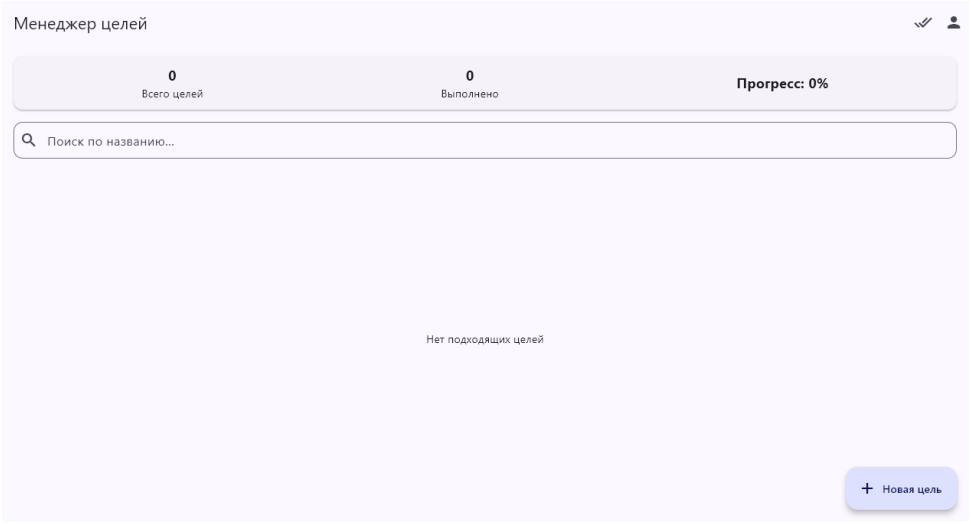


Рисунок 43 – Экран списка целей после выполнения нажатия на кнопку возврата

Реализация вертикального маршрутизированного перехода: Список целей → Профиль, выполняется при помощи метода context.push(Routes.profile) (рисунок 44). При нажатии на иконку профиля в верхней панели главного экрана приложение выполняет навигационный переход на страницу профиля пользователя. Процесс перехода показан на рисунках 45–46.

Рисунок 44 – Реализация вертикального маршрутизированного перехода на экран профиля

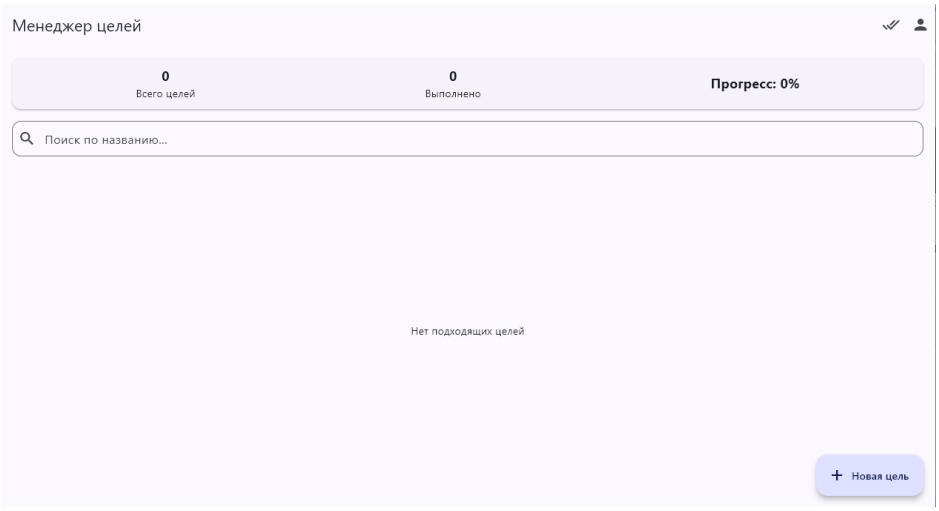


Рисунок 45 - Экран списка целей до выполнения вертикальной маршрутизированной навигации на экран профиля

Рисунок 46 – Экран профиля после выполнения вертикальной маршрутизированной навигации

Реализация вертикального маршрутизированного перехода назад осуществляется с помощью метода context.pop() (рисунок 47). После нажатия на кнопку «Назад» в верхней панели приложение закрывает экран профиля и возвращает пользователя к главному экрану со списком целей. Демонстрация процесса навигации представлена на рисунках 48–49.

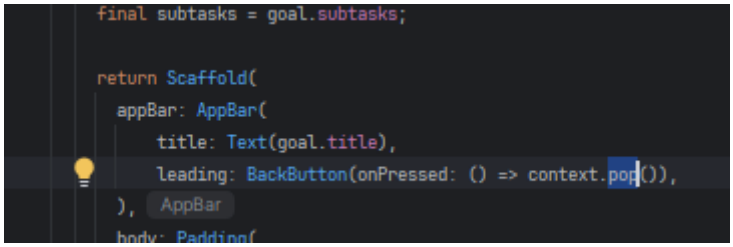


Рисунок 47 – Реализация метода context.pop()

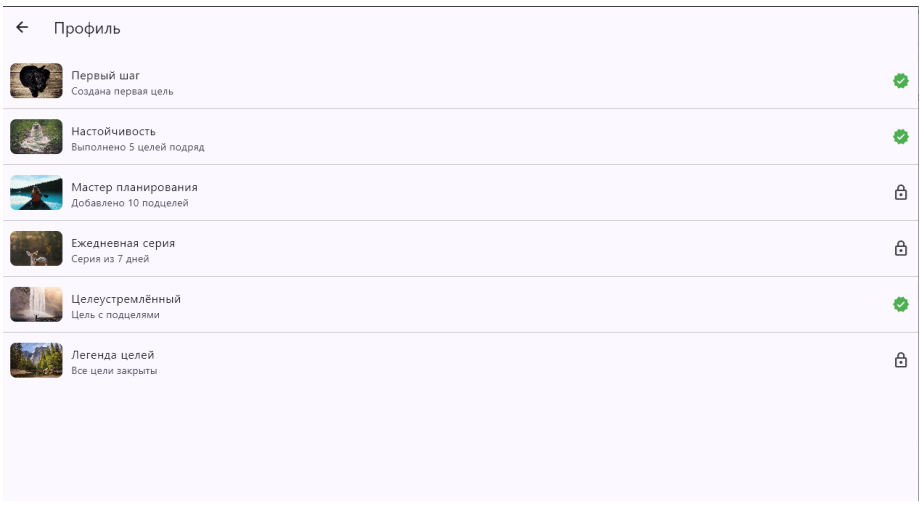


Рисунок 48 - Экран профиля до выполнения нажатия на кнопку возврата

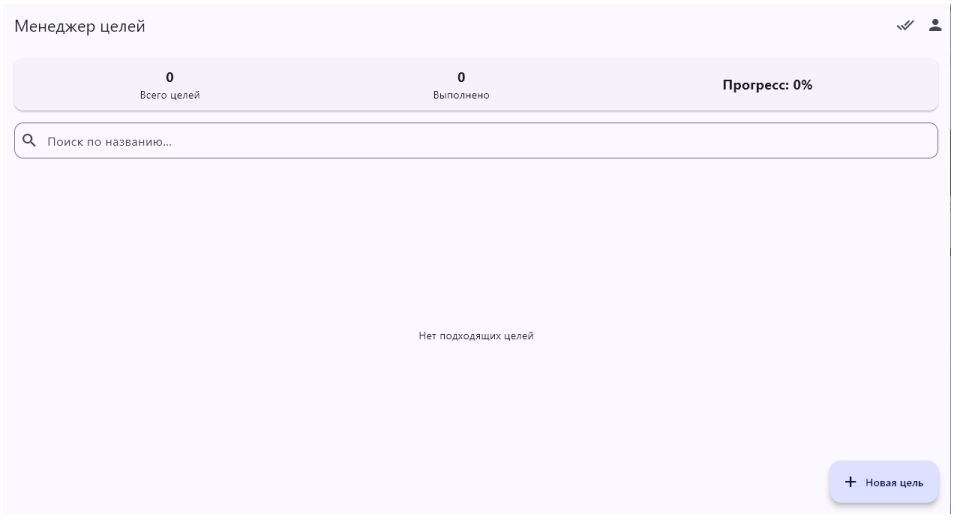


Рисунок 49 - Экран списка целей после выполнения нажатия на кнопку возврата

Реализация вертикального маршрутизированного перехода: Список целей → Выполненные цели, осуществляется при помощи метода context.push(Routes.completed) (рисунок 50). При нажатии на иконку «галочка» в верхней панели главного экрана пользователь переходит на страницу со списком завершённых целей. Процесс навигации продемонстрирован на рисунках 51–52.

```
IconButton(
  tooltip: 'Выполненные',
  icon: const Icon(Icons.done_all),
  onPressed: () {
    context.push(Routes.completed);
  },
),
```

Рисунок 50 – Реализация вертикального маршрутизированного перехода на экран выполненных целей

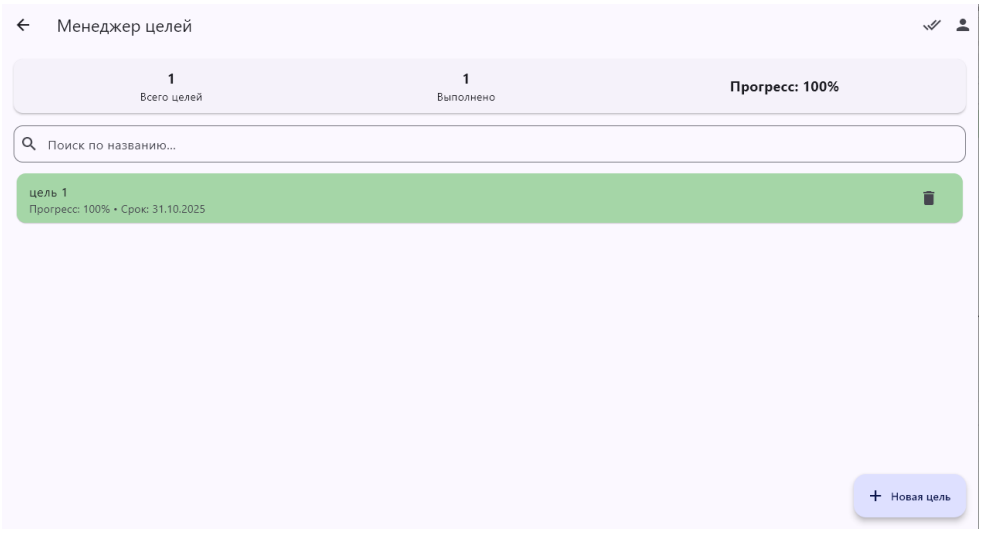


Рисунок 51 – Экран списка целей до выполнения вертикальной маршрутизированной навигации на экран выполненных целей

Рисунок 52 – Экран выполненных целей после выполнения вертикальной маршрутизированной навигации

Реализация вертикального маршрутизированного перехода назад выполняется с помощью метода context.pop() (рисунок 53). После нажатия на кнопку «Назад» в верхней панели приложение закрывает экран выполненных целей и возвращает пользователя к главному экрану со списком целей. Демонстрация процесса возврата представлена на рисунках 54–55.

Рисунок 53 – Реализация метода context.pop()

Рисунок 54 - Экран выполненных целей до выполнения нажатия на кнопку возврата

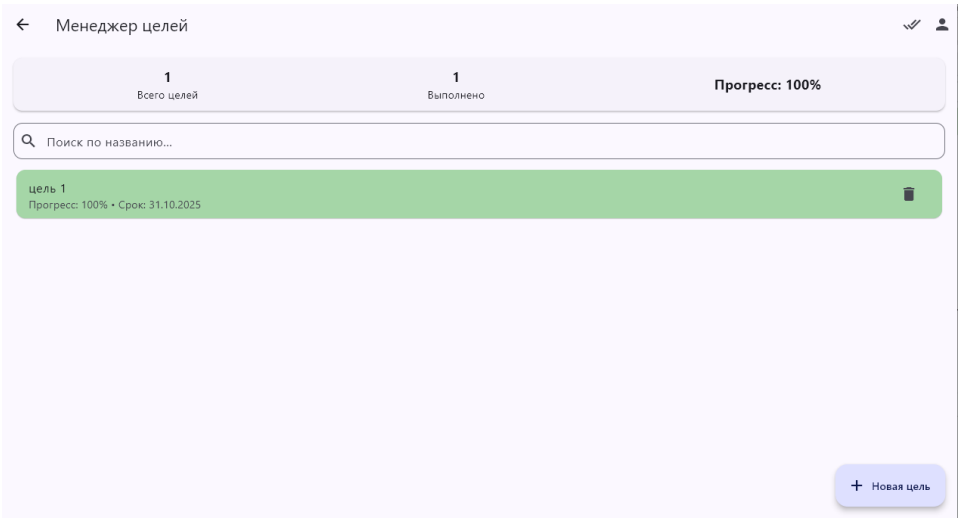


Рисунок 55 - Экран списка целей после выполнения нажатия на кнопку возврата

Реализация вертикального маршрутизированного перехода: Список целей → Подзадачи цели, выполняется при помощи метода context.push(Routes.goalDetail) (рисунок 56). При нажатии на карточку конкретной цели на главном экране пользователь переходит на страницу, где отображаются все подзадачи выбранной цели. Процесс навигации показан на рисунках 57–58.

```
child: GoalsListView(
  goals: goals,
  onDelete: _deleteGoal,
  onTap: (goal) async {
    await context.push(Routes.goalDetail, extra: goal);
    setState(() {});
  },
), GoalsListView
```

Рисунок 56 – Реализация вертикального маршрутизированного перехода на экран выполненных целей

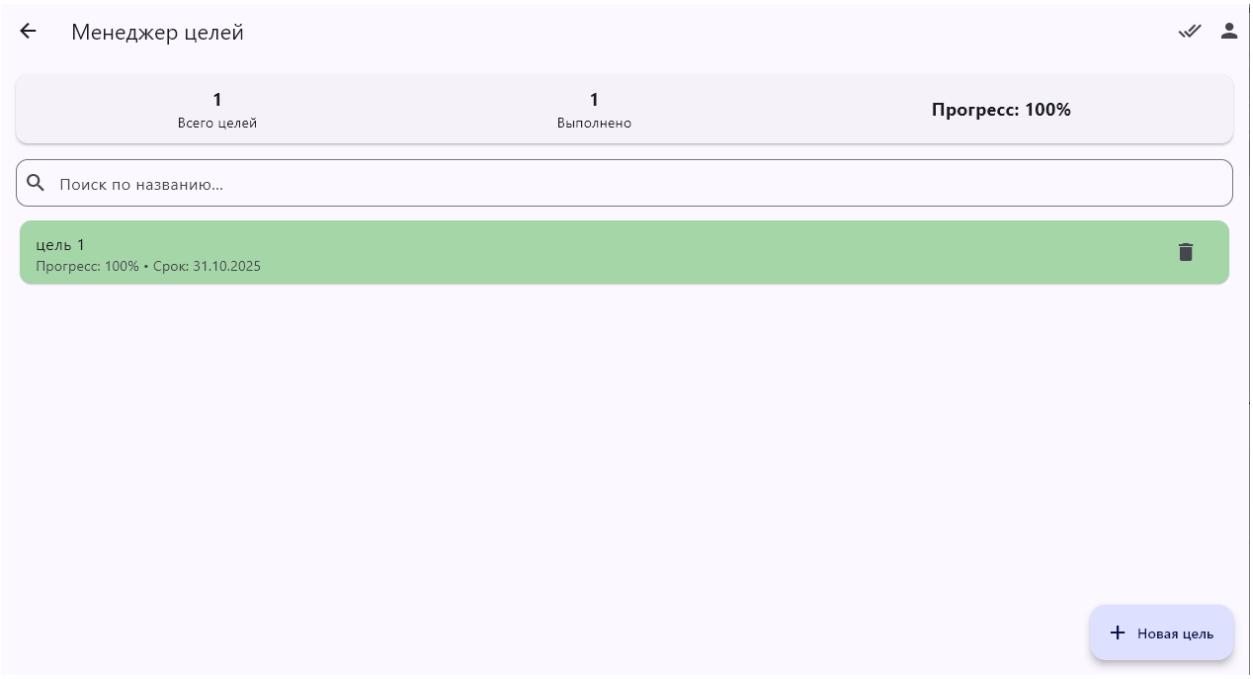


Рисунок 57 – Экран списка целей до выполнения вертикальной маршрутизированной навигации на экран подзадач цели

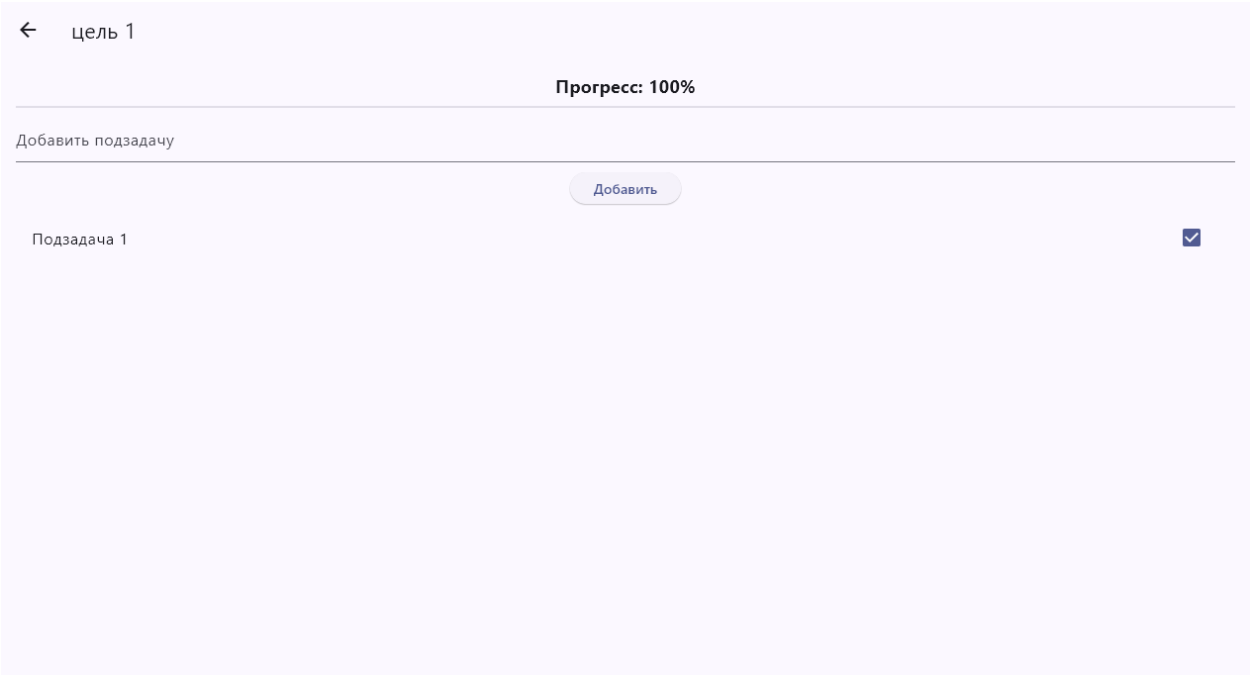


Рисунок 58 – Экран подзадач цели после выполнения вертикальной маршрутизированной навигации

Реализация вертикального маршрутизированного перехода назад осуществляется с помощью метода context.pop() (рисунок 59). После нажатия на кнопку «Назад» приложение закрывает экран подзадач и возвращает пользователя к списку всех целей. Демонстрация возврата представлена на рисунках 60–61.

```
return Scaffold(
  appBar: AppBar(
      title: Text(goal.title),
      leading: BackButton(onPressed: () => context.pop()),
  ),  // AppBar
  body: Padding(
```

Рисунок 59 – Реализация метода context.pop()

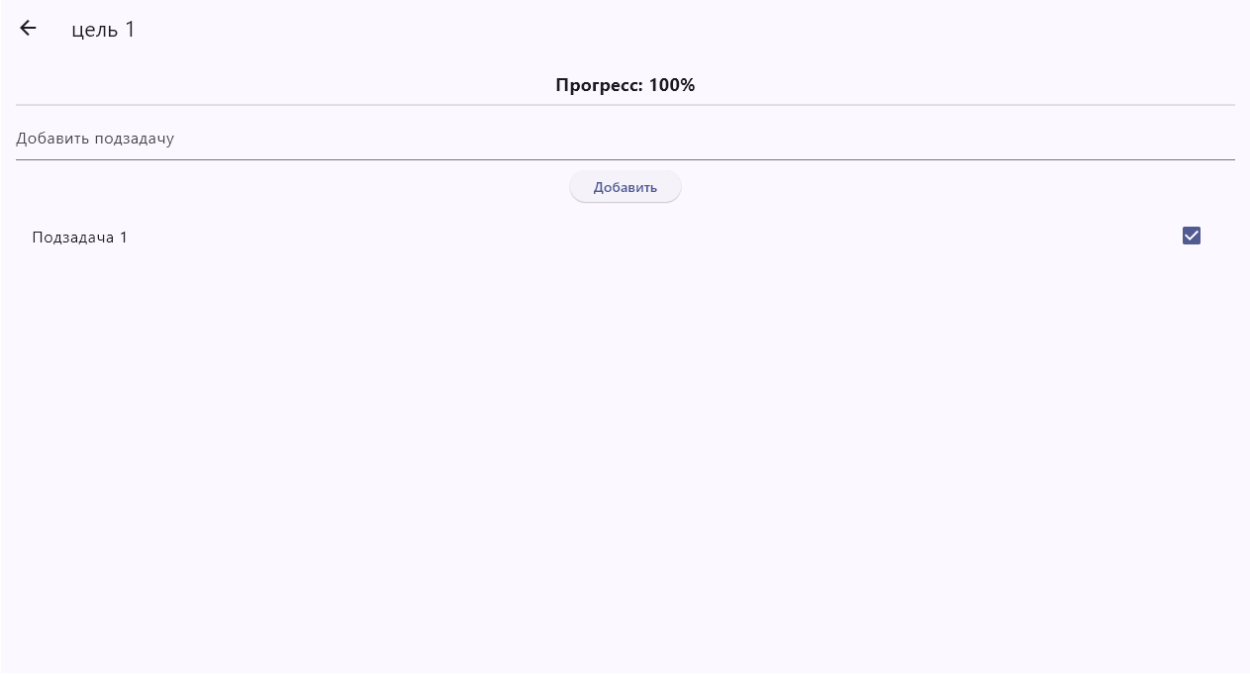


Рисунок 60 – Экран подзадач цели до выполнения нажатия на кнопку возврата

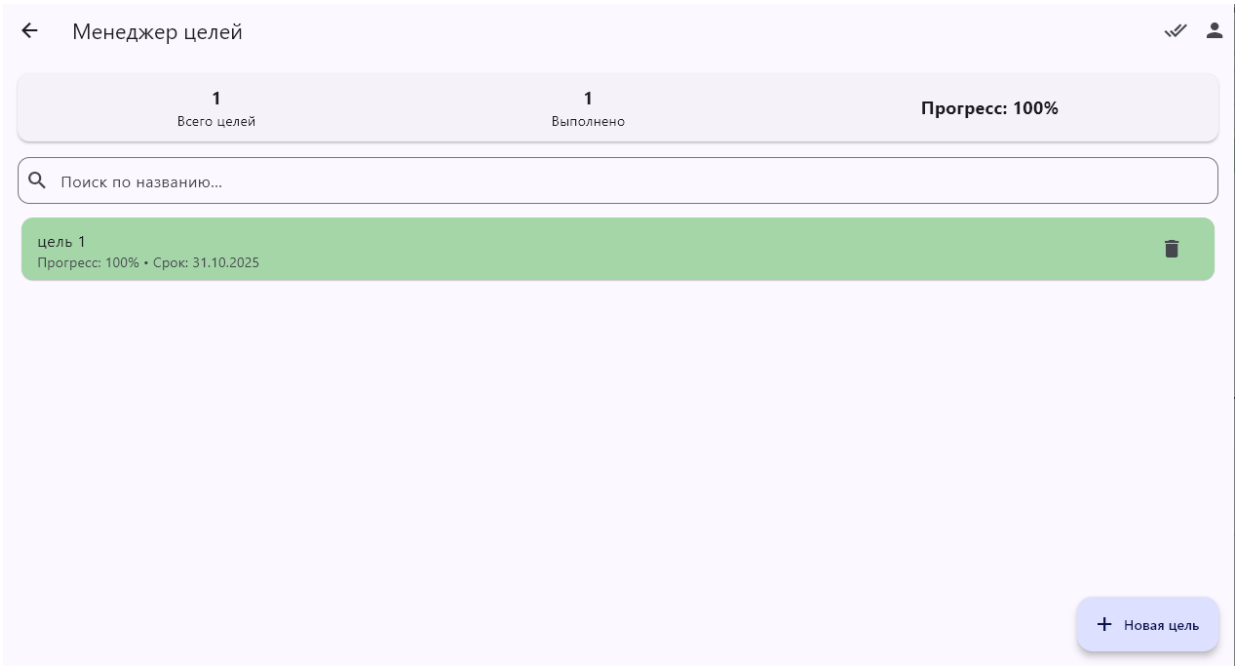


Рисунок 61 - Экран списка целей после выполнения нажатия на кнопку возврата

Горизонтальная маршрутизированная навигация

Горизонтальная страничная навигация реализована с использованием метода context.pushReplacement() (рисунок 62). В приложении данный механизм применяется на экране добавления цели: после нажатия кнопки «Создать цель» выполняется переход с полной заменой текущего экрана на главный экран со списком целей. Этот процесс иллюстрируют рисунки 63 – 64. Основная особенность такого перехода заключается в том, что после его выполнения пользователь не может вернуться на предыдущий экран, поскольку навигационный стек полностью обновляется.

```
    final goal = Goal(
      title: _titleController.text.trim(),
      deadline: _deadline!,
    );

    widget.goalService.addGoal(goal);

    context.pushReplacement(Routes.goalsList);
  }
```

Рисунок 62 - Реализация горизонтальной страничной навигации с помощью метода pushReplacement

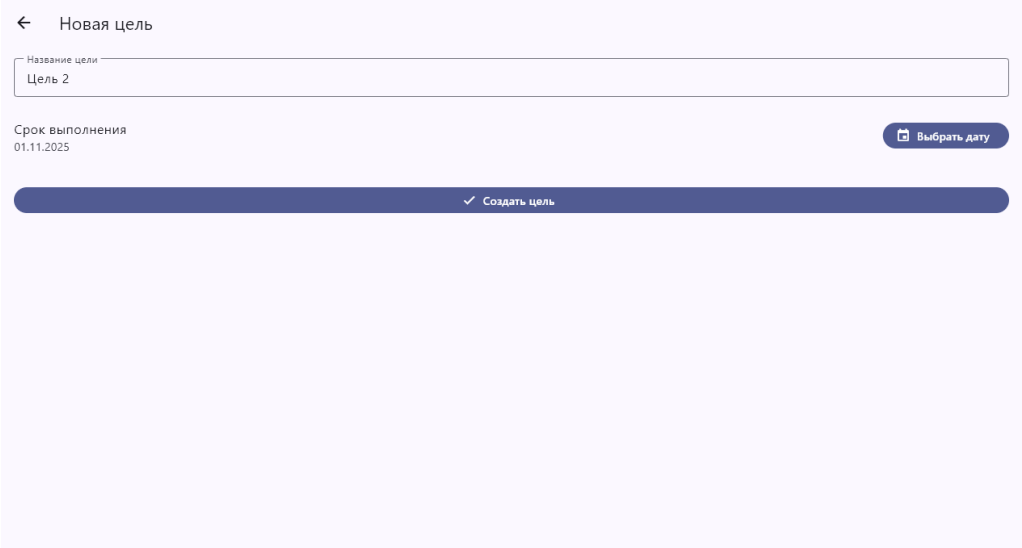


Рисунок 63 - Экран формы добавления цели до нажатия кнопки «Создать цель»

Рисунок 64 – Главный экран после выполнения горизонтальной страничной навигации с заменой верхнего экрана в стеке

Горизонтальная навигация с полной заменой стека

Ещё одним способом реализации горизонтальной навигации является использование метода go, который выполняет переход на указанный маршрут с полной очисткой и переинициализацией навигационного стека (рисунок 65). В данном приложении этот подход используется после успешного создания новой цели: сразу после сохранения данных осуществляется переход на главный экран со списком целей. Такой механизм исключает возможность возврата к экрану добавления через стандартные элементы навигации и обеспечивает логичное, завершённое поведение сценария.

```
    widget.goalService.addGoal(goal);

    context.go(Routes.goalsList);
}
```

Рисунок 65 – Реализация горизонтальной навигации с полной заменой стека с помощью метода go

Тестирование данного типа навигации выполнено на примере перехода с экрана добавления цели к основному экрану со списком целей. На рисунке 66 показан экран добавления цели перед подтверждением создания. После нажатия кнопки «Создать цель» выполняется горизонтальный навигационный переход методом go на корневой маршрут приложения с полной очисткой

стека, что исключает возможность возврата к предыдущему экрану. Итоговый результат перехода представлен на рисунке 67, где отображён главный экран приложения — список целей.

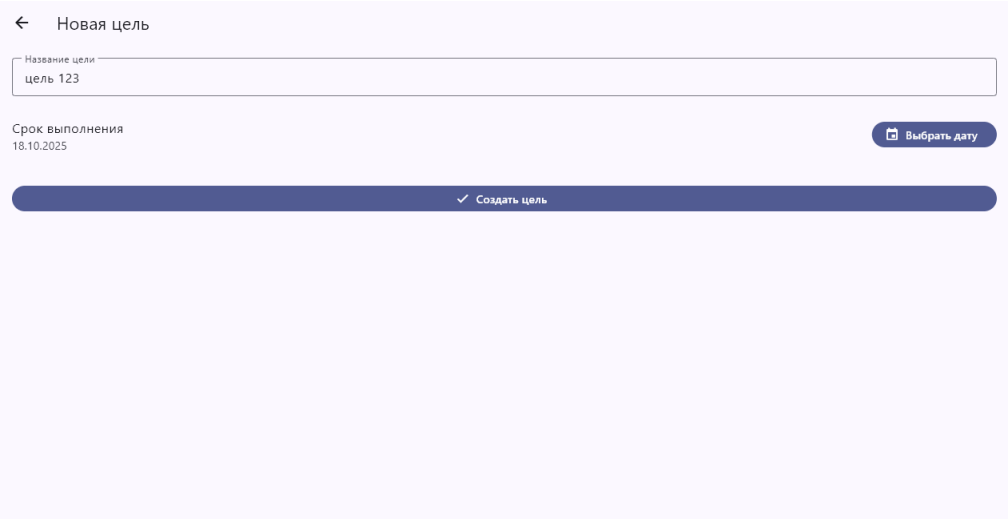


Рисунок 66 – Экран добавления цели до выполнения горизонтальной навигации с полной заменой стека

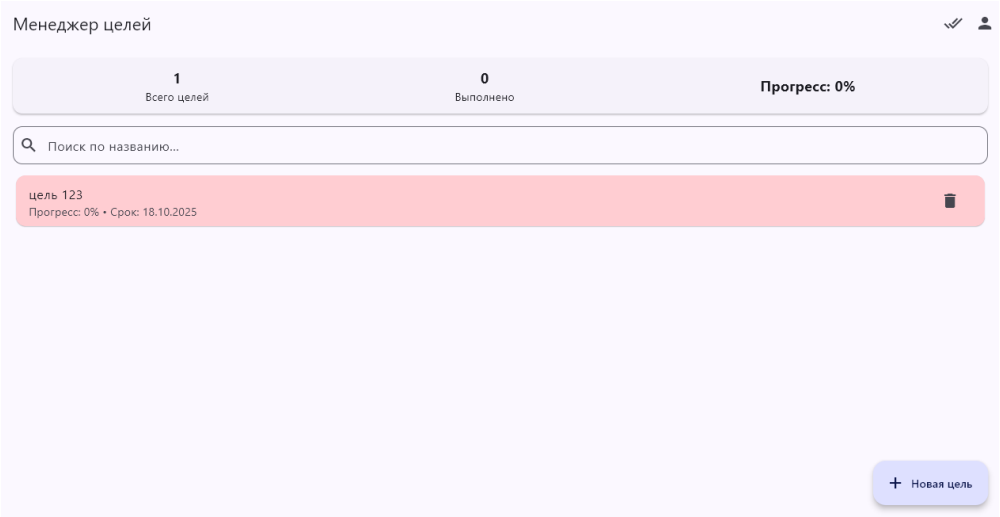


Рисунок 67 – Экран списка целей после выполнения горизонтальной навигации с полной заменой стека

В результате проведённой работы была успешно реализована и протестирована комплексная система навигации. Все виды навигационных переходов были тщательно протестированы в различных условиях и продемонстрировали стабильную работу. Результаты тестирования подтвердили, что разработанная система навигации обеспечивает высокую производительность, отзывчивость и соответствие ожиданиям пользователей.

Заключение

В ходе выполнения практической работы была разработана и внедрена система навигации для мобильного приложения включающая как страничную, так и маршрутизированную навигацию.

В процессе работы были реализованы основные виды навигационных переходов — вертикальные (push/pop) и горизонтальные (pushReplacement, go). Проведено тестирование переходов между всеми экранами приложения: списком целей, экраном добавления новой цели, профилем пользователя, выполненными целями и экраном деталей.

Использование GoRouter позволило значительно упростить навигационную структуру, сделать код более читаемым и поддерживаемым. Система навигации продемонстрировала высокую стабильность, отзывчивость и удобство для пользователя, что соответствует требованиям к современным мобильным приложениям.

Все изменения, выполненные в результате данной практической работы, были сохранены в удаленном репозитории github: https://github.com/Andrew-Savin-msk/prac7