DOI 10.37882/2223-2966.2025.06.35

АРХИТЕКТУРА КЛИЕНТ-СЕРВЕРНЫХ ВЗАИМОДЕЙСТВИЙ В МОБИЛЬНЫХ ПРИЛОЖЕНИЯХ НА DART ДЛЯ КОЛЛЕКТИВНОГО ОБСУЖДЕНИЯ ПЕШИХ МАРШРУТОВ

ARCHITECTURE OF CLIENT-SERVER INTERACTIONS IN MOBILE APPLICATIONS ON DART FOR COLLECTIVE DISCUSSION OF HIKING ROUTES

M. Mestnikova

Summary. Mobile applications focused on geolocation services and social interaction are becoming more and more in demand in the context of growing interest in an active lifestyle and digital recording of personal routes. Such applications allow users not only to track their own movements, but also to share routes with others, leave comments, share impressions and ratings. However, the development of an architecture capable of effectively supporting these functions in a multi-user environment requires a special approach to choosing a client-server model. To solve this problem, cross-platform development and cloud infrastructure tools have been used to ensure scalability, stability, and convenience of real-time data synchronization.

The Dart language and the Flutter framework were used as the technological basis for creating a mobile client, as well as Firebase cloud services. This architecture allows you to organize a reliable mechanism for publishing and displaying routes, comments, and multimedia content with support for offline access, and push notifications. Integration with Google Maps expands the possibilities of route visualization. The application structure includes modules for managing status, displaying cartographic data, and processing user actions, while synchronization is performed directly with cloud storage.

The developed prototype has passed modular, integration, load and offline testing with positive results. The experience gained demonstrates the practical importance of using architecture to create social geolocation services.

Keywords: mobile applications, deployment diagram, Android Studio, Dart-based development, Firebase.

Местникова Мария Олеговна
Северо-Восточный федеральный университет им. М.К. Аммосова, г. Якутск
nncn1436mestnik@gmail.com

Аннотация. Мобильные приложения, ориентированные на геолокационные сервисы и социальное взаимодействие, становятся все более востребованными в условиях растущего интереса к активному образу жизни и цифровой фиксации личных маршрутов. Такие приложения позволяют пользователям не только отслеживать собственные передвижения, но и делиться маршрутами с другими, оставлять комментарии, обмениваться впечатлениями и оценками. Однако разработка архитектуры, способной эффективно поддерживать эти функции в условиях многопользовательского взаимодействия, требует особого подхода к выбору клиент-серверной модели. Для решения этой задачи применены инструменты кроссплатформенной разработки и облачной инфраструктуры, обеспечивающие масштабируемость, устойчивость и удобство синхронизации данных в реальном времени.

В качестве технологической основы использован язык Dart и фреймворк Flutter для создания мобильного клиента, а также облачные сервисы Firebase. Такая архитектура позволяет организовать надёжный механизм публикации и отображения маршрутов, комментариев и мультимедийного контента с поддержкой офлайн-доступа, и push-уведомлений. Интеграция с Google Maps расширяет возможности визуализации маршрутов. Структура приложения включает модули для управления состоянием, отображения картографических данных и обработки пользовательских действий, при этом синхронизация осуществляется напрямую с облачными хранилищами. Разработанный прототип прошёл модульное, интеграционное, нагрузочное и оффлайн тестирование с положительными результатами. Полученный опыт демонстрирует практическую значимость использования архитектуры для создания социально-геолокационных сервисов.

Ключевые слова: мобильные приложения, диаграмма развертывания, Android Studio, разработка на языке Dart, Firebase.

Актуальность исследования обусловлена ограниченным числом современных мобильных решений, позволяющих пользователям фиксировать посещенные локации в формате визуальных постов. Также введение персональных записей пользователей, которым необходимо не только фиксировать посещенные места, но и прикреплять к ним текстовые описания, изображения и пользовательские оценки.

Объектом исследования выступают технологии мобильной разработки, ориентированные на использование геолокационных сервисов. Предмет исследования — разработка мобильного приложения на языке программирования Dart, предназначенного для создания геозаметок и планирования маршрутов.

Целью исследования является анализ архитектуры клиент-серверного мобильного приложения на Dart

с использованием Firebase, предназначенного для публикации, обсуждения и оценки пеших маршрутов пользователями. Для достижения этой цели в работе решаются следующие задачи:

1. формулировка и обоснование ключевых требований к функционалу мобильного приложения;
2. проектирование архитектуры приложения;
3. реализация рабочего прототипа;
4. проведение тестирования прототипа и анализ его соответствия поставленным требованиям.

Под требованиями к программному обеспечению понимаются функциональные и нефункциональные характеристики, которые должны быть реализованы в системе. Свойства, которыми обладает ПО для актуального определения функций и ограничений действий ПО, а также аппаратного обеспечения и среды разработки. Другими словами, это комплекс существенных свойств ПО [5].

Согласно международному стандарту ISO/IEC TR 19759:2015 (Software Engineering Body of Knowledge), требования к программному обеспечению охватывают весь спектр аспектов, влияющих на его проектирование, разработку и эксплуатацию. В рамках настоящего исследования на основе данного стандарта были определены следующие базовые требования к мобильному приложению:

1. система регистрации и аутентификации пользователей;
2. возможность создания и сохранения маршрутов в пользовательском профиле;
3. просмотр маршрутов других пользователей;
4. система оценки маршрутов (рейтинги, лайки);
5. фильтрация контента по заданным критериям;
6. хранение пользовательских данных и контента на серверной стороне (в облаке).

Архитектура приложения разрабатывалась в соответствии с классической моделью «клиент–сервер», где мобильное приложение выступает в роли клиента, а роль сервера выполняет облачная инфраструктура Firebase. Такой подход позволяет сосредоточить основную логику взаимодействия с пользователем и часть бизнес-логики на стороне мобильного клиента, в то время как хранение данных, синхронизация и авторизация вынесены на облачный бекенд.

Мобильное приложение устанавливает постоянные подключения (listeners) к облачной базе данных Firestore для отслеживания изменений данных в реальном времени. Благодаря этому новый комментарий или оценка маршрута, добавленные одним пользователем, мгновенно становятся видимы другим пользователям в приложении без необходимости вручную обновлять экран. Для обеспечения плавности работы в условиях нестабильного интернета применяются возможности офлайн-режима: локальное кеширование данных на устройстве и последующая синхронизация с облаком при восстановлении соединения [9]. При проектировании архитектуры особое внимание уделялось структурированию системы на модули. Мобильное приложение разделено на слои: презентационный слой, слой управления состоянием и бизнес-логики (например, provider/bloc, для отделения логики от UI) и слой доступа к данным (взаимодействие с API Firebase). Это обеспечивает поддержку принципов разделения ответственности, а хранение данных организовано в облачной NoSQL базе Firestore в виде структурированных документов. В целом, спроектированная архитектура охватывает все аспекты системы — от мобильного UI до облачного хранения и внешних API — и обеспечивает их согласованное взаимодействие.

Для наглядного описания спроектированной архитектуры в статье используется UML-диаграмма (диаграмма развертывания). Основное внимание уделено отображению клиентской и серверной частей, а также связям между ними в условиях облачной инфраструктуры. В качестве клиентской платформы выступают мобильные устройства пользователей. Приложение включает в себя модули пользовательского интерфейса, работы с геолокацией, управления состоянием, а также компоненты для взаимодействия с облачными сервисами через SDK Firebase. Серверная часть приложения представлена полностью облачной инфраструктурой Firebase, что отражает модель Backend-as-a-Service (BaaS).

Все клиентские устройства взаимодействуют с облачными сервисами по защищенным каналам связи (HTTPS), используя официальные SDK. Таким образом, архитектура исключает необходимость в собственных серверных мощностях и минимизирует затраты на инфраструктуру.

Диаграмма развертывания демонстрирует логическую связность между компонентами, что обеспечивает понимание того, где именно развернуты модули приложения и как они обмениваются данными.

Прототип приложения реализован с использованием языка Dart и фреймворка Flutter, что обеспечило запуск приложения на устройствах Android и iOS без изменения исходного кода. В процессе разработки использовался подход непрерывной интеграции Flutter с сервисами Firebase. Ниже перечислены основные технологии и решения, примененные при создании прототипа:

Firebase Cloud Firestore — облачная NoSQL база данных, выступающая в роли хранилища данных приложения. В Firestore были созданы коллекции для основных сущностей: сохраняются данные о каждом опубликованном маршруте, комментарии, пользователи. Такая

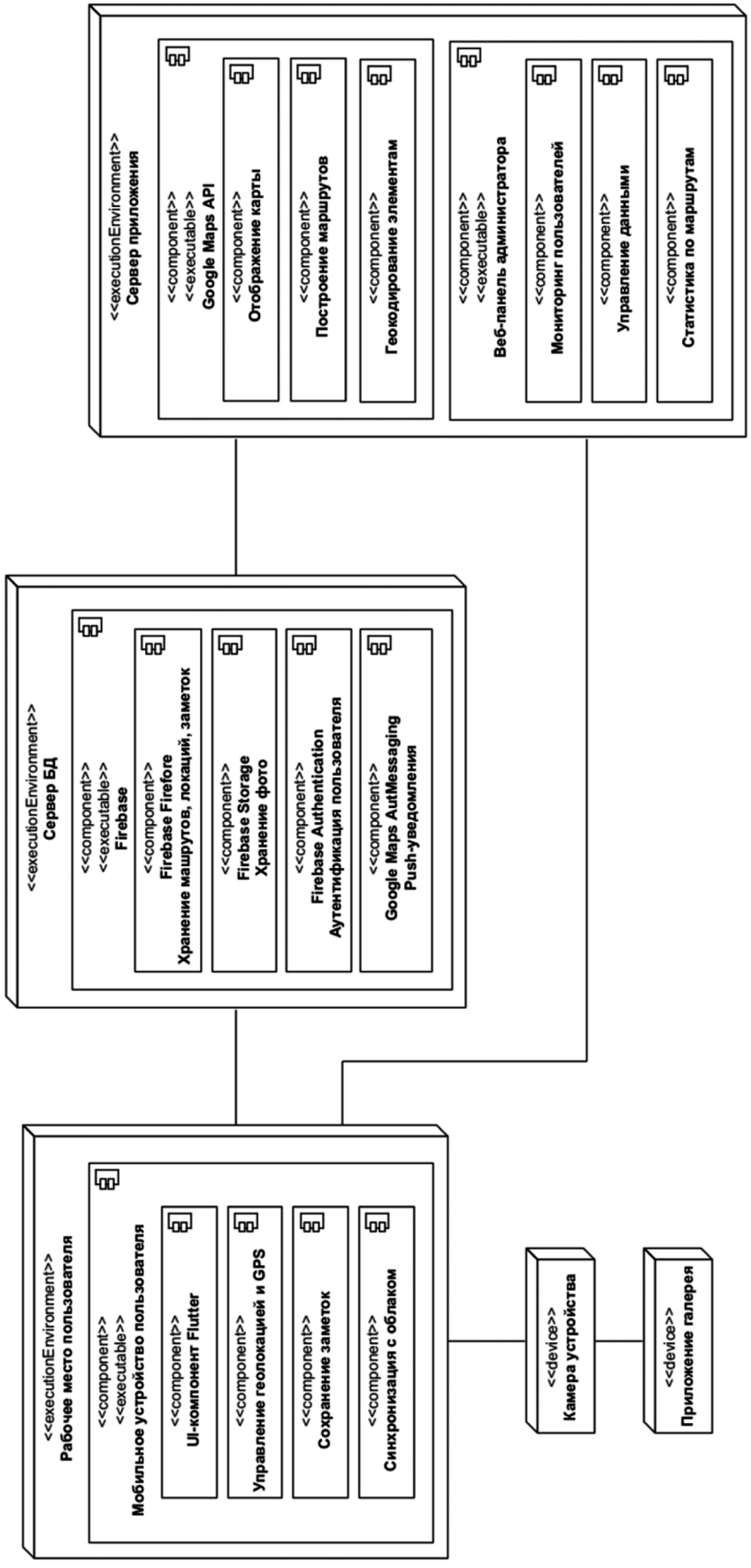

Рис. 1. Диаграмма развертывания

структура позволяет быстро выполнять запросы для получения всех комментариев к данному маршруту или всех маршрутов, опубликованные определенным пользователем. Firestore автоматически индексирует данные и поддерживает поиск с фильтрацией, что пригодилось для реализации функции поиска маршрутов по названию или тегам. Все операции с базой, а именно создание нового маршрута, добавление комментария, обновление рейтинга, выполняются напрямую с мобильного клиента через SDK Firebase, с мгновенной синхронизацией и локальным кешированием.

Firebase Storage — облачное хранилище файлов, интегрированное в приложение для работы с фотографиями. Маршруты прогулок зачастую сопровождаются фотографиями мест, картосхемами или треками в формате GPX (GPS eXchange Format)— эти данные сохраняются в Firebase Storage, а в документы Firestore заносятся только ссылки на соответствующие файлы. В прототипе реализована возможность прикреплять фото к маршруту: при публикации маршрута выбранное изображение загружается из памяти устройства или камеры и отправляется в Storage, после чего URL (Uniform Resource Locator) сохраненного файла автоматически добавляется в запись маршрута в Firestore. Такая схема разгружает базу данных и оптимизирует трафик, позволяя при загрузке ленты маршрутов сначала получать текстовую информацию, а изображения подгружать по ссылкам по мере необходимости.

Firebase Authentication — модуль аутентификации Firebase, отвечающий за регистрацию и вход пользователей. В приложении реализована система учетных записей: новый пользователь может создать аккаунт (с использованием email/пароля). Firebase Auth берет на себя хранение учетных записей и проверку входа, возвращая уникальный идентификатор пользователя (uid — unique identifier), который используется как ключ при привязке данных (например, маршрут содержит uid автора). Также настроены правила безопасности Firestore, привязывающие права доступа к uid пользователя: в результате каждый пользователь может редактировать или удалять только собственные маршруты и комментарии. Благодаря готовым решениям Firebase процесс управления пользователями в прототипе сводится к вызовам функций SDK (регистрация, логин, выход) [10].

Firebase Cloud Messaging (FCM) — сервис отправки push-уведомлений. Он используется для обеспечения обратной связи: приложение рассылает уведомления, когда происходят важные события, связанные с маршрутами. В прототипе настроены уведомления о новых комментариях: автор маршрута получит push-уведомление на устройство, если кто-то прокомментировал его запись; аналогично, комментарии в ответ могут генерировать уведомления для их автора. Технически это реализовано следующим образом: приложение при входе пользователя регистрирует токен устройства в FCM; при добавлении нового комментария в Firestore срабатывает облачная функция Firebase, которая формирует и отправляет уведомление через FCM нужному адресату (по токену). В результате пользователи оперативно получают оповещения, даже если приложение свернуто. Помимо комментариев, в будущем можно задействовать FCM и для других типов уведомлений — например, еженедельная подборка новых популярных маршрутов или оповещение друзей о том, что пользователь опубликовал новый маршрут [11].

Интеграция с Google Maps API. Поскольку основная тема приложения — географические маршруты, прототип интегрирован с картографическим сервисом для отображения маршрутов на карте. Использован пакет Google Maps API для Flutter (плагин, предоставляющий виджет карты и доступ к функциям Google Maps). При просмотре детали маршрута пользователь видит встроенную карту, на которой проложен трек прогулки (маркером отмечается старт и финиш, линия маршрута наносится по координатам). Для этого географические данные маршрутов (набор точек GPS) хранятся в Firestore, а при загрузке экрана маршрута эти данные передаются виджету карты. Кроме того, Google Maps API используется для геокодирования, то есть определяет координаты по заданному адресу или названию места, если пользователь при добавлении маршрута вводит текстовое описание начала маршрута. В прототипе показана базовая функциональность работы с картой; она подтверждает, что архитектура поддерживает подключение стороннего API и обработку связанных с ним данных.

На уровне клиентского приложения реализован полноценный интерфейс, включающий основные экраны: лента опубликованных маршрутов (с возможность фильтрации и поиска), экран создания нового маршрута, экран просмотра маршрута, экран профиля пользователя, форма входа и регистрации. Для навигации по приложению использован встроенный механизм маршрутизации Flutter (Navigator). Чтобы обеспечить интерактивность интерфейса при изменении данных, применены состояния и виджеты, подписанные на потоки данных Firestore (через StreamBuilder). Управление состоянием (state management) организовано с помощью Provider — этот подход позволил вынести бизнес-логику в отдельные провайдеры и ViewModel, которые взаимодействуют с репозиториями данных (Firebase). Таким образом, при реализации прототипа были соблюдены принципы, заложенные на этапе проектирования архитектуры: четкое разделение интерфейса, логики и доступа к данным, а также слабая связанность компонентов, что облегчает поддержку и масштабирование кода.

Средства разработки и инструменты. Прототип создавался с использованием официального SDK Flutter

и FlutterFire. В среде разработки (IDE Android Studio/ Visual Studio Code) настроена непрерывная сборка и отладка на устройствах-эмуляторах. Firebase проект настроен в режиме тестирования (эмуляторы Firebase использовались для прогона тестов). Код прототипа структурирован согласно принятым практикам. Для контроля версий использовался Git, что также способствовало отслеживанию изменений архитектуры в ходе итеративной доработки.

После разработки прототипа была проведена серия тестирований с целью проверки качества архитектуры и выявления возможных проблем. Тестирование носило поэтапный характер, охватывая разные уровни и аспекты системы. На первом этапе были написаны и выполнены модульные тесты для отдельных функций и компонентов клиентского приложения. Тестировались, в частности, утилиты обработки данных (корректность вычисления длины маршрута по списку координат, форматы дат и времени в комментариях). Результаты модульного тестирования показали корректность ключевых алгоритмов и методов: все проверочные сценарии прошли успешно, обнаружены и исправлены лишь незначительные ошибки, связанные с валидацией полей ввода (например, недопустимо пустое название маршрута).

Далее проводилось интеграционное тестирование, фокусирующееся на взаимодействии мобильного приложения с облачным бекендом Firebase. В специальной тестовой среде были развернуты Firebase Emulators, что позволило имитировать работу Firestore. Тестовые сценарии охватили типичные случаи: регистрация и вход пользователя, публикация маршрута, загрузка списка маршрутов из базы, добавление комментария и получение уведомления. Автоматические интеграционные тесты запускали приложение, выполняли описанные действия и проверяли ожидаемый результат (например, после добавления маршрута — количество документов в коллекции маршрутов увеличилось; при неавторизованном запросе к защищенной коллекции — получен отказ доступа). В результате было подтверждено, что все компоненты системы правильно связаны: данные успешно передаются от клиента к серверу и обратно, права доступа соблюдаются.

Отдельно проверялась способность приложения работать при отсутствии интернет-соединения и после его восстановления. Для этого на устройстве-тесте имитировалось отключение сети: пользователь пытался открыть ранее загруженные маршруты (они должны отображаться за счет кеша Firestore), благодаря офлайн-поддержке Firestore, приложение позволяет читать уже кеширован-

```
{
  TestWidgetsFlutterBinding.ensureInitialized();
  test('', () async {
    final routes = FirebaseFirestore.instance.collection('routes');
    for (int i = 0; i < 100; i++) {
      await routes.add({
        'title': 'Route $i',
        'description': 'Load testing route $i',
        'author': 'tester',
        'points': List.generate(10, (index) => {
              'lat': 55.7 + Random().nextDouble() / 100,
              'lon': 37.6 + Random().nextDouble() / 100,
            }),
      });
    }
    final count = (await routes.get()).docs.length;
    expect(count, greaterThanOrEqualTo(100));
  });
}
```

Рис. 2. Код модульного тестирования логики расчета маршрута

```
{
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();
  testWidgets('', (WidgetTester tester) async {
    app.main();
    await tester.pumpAndSettle();
    await FirebaseAuth.instance.createUserWithEmailAndPassword(
      email: "test1@gmail.com",
      password: "qwer123",
    );
    await tester.pumpAndSettle();
    expect(find.text('Create Route'), findsOneWidget);
  });
}
```

Рис. 3. Код интеграционного тестирования регистрации и экрана создания маршрута

```
void offline() async {
  TestWidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  test('', () async {
    FirebaseFirestore.instance.settings = const Settings(
      persistenceEnabled: true,
    );
    final commentsRef = FirebaseFirestore.instance.collection('comments');
    final localDoc = await commentsRef.add({
      "text": "Offline comment",
      "routeId": "offline_route_test",
    });
    expect(localDoc.id, isNotNull);
  });
}
```

Рис. 4. Код проверки сохранения комментария в офлайн-режиме

```
{
  TestWidgetsFlutterBinding.ensureInitialized();
  test('', () async {
    final routes = FirebaseFirestore.instance.collection('routes');
    for (int i = 0; i < 100; i++) {
      await routes.add({
        'title': 'Route $i',
        'description': 'Load testing route $i',
        'author': 'tester',
        'points': List.generate(10, (index) => {
              'lat': 55.7 + Random().nextDouble() / 100,
              'lon': 37.6 + Random().nextDouble() / 100,
            }),
      });
    }
    final count = (await routes.get()).docs.length;
    expect(count, greaterThanOrEqualTo(100));
  });
}
```

Рис. 5. Код нагрузочного тестирования с генерацией 100 маршрутов

ные данные. После подключения к интернету приложение автоматически синхронизировало изменения с сервером — отложенный комментарий успешно появился в облаке и стал виден другим пользователям. Этот тест подтвердил правильность архитектурного решения использовать Firestore с его встроенными возможностями офлайн-доступа: пользователи приложения не остаются «отрезанными» от информации во время временного отсутствия сети.

Для оценки масштабируемости архитектуры была проведена имитация нагрузки: с помощью скриптов и инструментов тестирования Firebase были сгенерированы десятки тестовых пользователей и сотни виртуальных маршрутов и комментариев. Необходимо проверить, как система справляется с возросшим объемом данных и множеством одновременных действий. Метрики, собранные в Firebase (например, время ответа на запрос к базе, задержка доставки уведомлений), оставались в пределах нормы при умеренной нагрузке. Приложение на стороне клиента также продолжило работать без заметного ухудшения производительности: лента маршрутов прокручивалась плавно, время загрузки данных не превысило нескольких секунд даже при заполненной базе. Конечно, полноценное тестирование на экстремальных объемах данных выходит за рамки прототипа, однако проведенные испытания показали, что выбранная архитектура обладает резервом масштабирования. Добавление новых пользователей или контента линейно увеличивает нагрузку, с которой Firebase-сервисы способны автоматически справляться за счет внутреннего масштабирования. Таким образом, прототип подтвердил свою работоспособность и устойчивость в приближенных к реальным условиям эксплуатации.

В результате проделанной работы имеется прототип мобильного приложения, который представлен на рисунке 6.

Заключение

В ходе выполнения работы разработан и исследован прототип мобильного приложения, предназначенного для коллективного обсуждения пеших маршрутов. Предложенная архитектура клиент-серверных взаимодействий продемонстрировала свою эффективность: она обеспечила оперативный обмен данными между пользователями в реальном времени, поддержку офлайн-режима и гибкое масштабирование под рост аудитории. За счет использования Flutter удалось значительно сократить время разработки интерфейса для разных платформ. Научная значимость работы заключается

Рис. 6. Пользовательский интерфейс

в обобщении опыта построения распределенных мобильных приложений на основе современных технологий кроссплатформенной разработки и облачных сервисов. В статье продемонстрировано, как сочетание Flutter и Firebase может быть использовано для решения задач социального взаимодействия пользователей, требующих надежной синхронизации и хранения данных.

Разработанная архитектура и диаграммы могут служить ориентиром для системных архитекторов и исследователей, занимающихся проектированием социальных мобильных сервисов на сходных технологических стеках. Практическая значимость проекта выражается в том, что созданный прототип может послужить основой для реального приложения, полезного сообществу любителей пеших прогулок. Архитектура уже прошла первичную проверку тестированием, что снижает риски при дальнейшем внедрении. Результаты нагрузки указывают, что приложение сможет обслуживать растущее число пользователей без ухудшения качества сервиса, что важно для любого социального продукта. Предложенные решения по интеграции геолокационных сервисов (Google Maps) расширяют пользовательский опыт, позволяя визуализировать маршруты и тем самым повышать наглядность и ценность публикуемой информации.

В дальнейшем планируется развивать проект в нескольких направлениях. Во-первых, расширение функциональности приложения: добавление новых социальных возможностей (система друзей, совместное планирование прогулок), внедрение рейтинговой системы и алгоритмов рекомендаций маршрутов на основе предпочтений пользователя. Во-вторых, усиление аналитической составляющей: разработка административной вебпанели для модерации контента, отслеживания активности пользователей и сбора статистики по популярности маршрутов. В-третьих, проведение более детального нагрузочного тестирования: с ростом базы данных может понадобиться оптимизация структуры хранилища или использовать кеширование на стороне клиента более активно. Подводя итог, разработанная архитектура показала свою жизнеспособность и соответствие поставленной задаче. Она сочетает научную новизну и практическую полезность. Дальнейшая работа над проектом будет направлена на превращение прототипа в полнофункциональное приложение, а также на исследование полученных данных использования для постоянного совершенствования архитектурных решений.

ЛИТЕРАТУРА

1. Дейтел П. Android для программистов: учебное пособие / П. Дейтел, Х. Дейтел, М. Морган. — Санкт-Петербург: Питер, 2013. — 557 с.
2. Голощапов А.Л. Google Android: книга / А.Л. Голощапов — Санкт-Петербург: БХВ-Петербург, 2013. — 268 с.
3. Мартин Р. Чистый Код: учебное пособие / Р. Мартин — Санкт-Петербург: Питер, 2013. — 590 с.
4. Анализ требований к программному обеспечению с примерами. [Электронный ресурс]. — Режим доступа: https://www.webursitet.ru/article/chto-takoe-trebovaniya-k-programmnomu-produktu.html (дата обращения 29.12.2024)
5. Области знаний программной инженерии и стандарты ЖЦ программного обеспечения — Текст: электронный // Интуит, Национальный открытый университет. — [Электронный ресурс]. — Режим доступа: https://intuit.ru/studies/courses/2190/237/lecture/6118
6. Спецификация формата GeoJSON. [Электронный ресурс]. — Режим доступа: http://gis-lab.info/docs/geojson_ru.html (дата обращения 11.01.2025)
7. Фаронов В.А. Программирование на языке высокого уровня: учебное пособие /В.А. Фаронов. — Издательство: Питер, 2006. — 640 с.
8. Советов Б.Я. Моделирование систем: учебное пособие / Б.Я. Советов — Москва: Вильямс, 2006. — 340 с.
9. Chatterjee N. Real-time Communication Application Based on Android Using Google Firebase // Computer Science and Management Studies / Nilanjan Chatterjee, Souvik Chakraborty, Aakash Decosta, Asoke Nath — 2018. — №6 (4). — С. 74–79 / [Электронный ресурс]. — Режим доступа: https://www.researchgate.net/publication/324840628_Real-time_Communication_Application_Based_on_Android_Using_Google_Firebase (дата обращения 17.03.2025)
10. Moroney L. Using Authentication in Firebase // The Definitive Guide to Firebase / Laurence Moroney — 2017. — С. 25–50 / [Электронный ресурс]. — Режим доступа: https://link.springer.com/chapter/10.1007/978-1-4842-2943-9_2 (дата обращения 20.03.2025)
11. Khawas C. Application of Firebase in Android App Development-A Study / Chunnu Khawas, Pritam Shah — 2018. — №179(46) — С. 49–53 / [Электронный ресурс]. — Режим доступа: https://www.researchgate.net/publication/325791990_Application_of_Firebase_in_Android_App_Development-A_Study (дата обращения 28.03.2025)
12. Silva R. Development and Evaluation of a Mobile Application with Augmented Reality for Guiding Visitors on Hiking Trails / Rute Silva, Rui Jesus, Pedro Jorge — 2023. — №6 — С. 58 / [Электронный ресурс]. — Режим доступа: https://www.mdpi.com/2414-4088/7/6/58 (дата обращения 03.04.2025)

Журнал «Современная наука: актуальные проблемы теории и практики»