

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ
КАЗАХСТАН

Казахский национальный исследовательский технический университет
имени К.И.Сатпаева

Институт автоматизации и информационных технологий

Кафедра «Программная инженерия»

Камиль Дильназ Руслановна

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

Образовательная программа: 6B06102 Computer Science

Алматы 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ
КАЗАХСТАН

Казахский национальный исследовательский технический университет
имени К.И.Сатпаева

Институт автоматизации и информационных технологий

Кафедра «Программная инженерия»

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой ПИ
канд. тех. наук, ассоц. профессор
Ф.Н. Абдолдина
« 30 » _____ 2025 г.



ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

На тему: «Разработка приложения по мониторингу и обратной связи по
качеству городской среды»

Образовательная программа: 6B06102 Computer Science

Выполнил

Камиль Д.Р.

Рецензент

Научный руководитель
PhD, ассоциированный профессор
Ж.Б. Кальпеева
" 4 " _____ 2025 г.

PhD, ассистент профессор в МУИТ
С.Б. Муханов
_____ 2025 г.



Алматы 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ
КАЗАХСТАН

Казахский национальный исследовательский технический университет
имени К.И.Сатпаева

Институт автоматизации и информационных технологий

Кафедра «Программная инженерия»

УТВЕРЖДАЮ

Заведующий кафедрой ПИ
канд. тех. наук, ассоц. профессор
Ф.Н. Абдолдина
«25» 05 2025 г.



ЗАДАНИЕ

на выполнение дипломного проекта

Обучающемуся: Камиль Дильназ Руслановне

Тема: Разработка приложения по мониторингу и обратной связи по качеству городской среды

Утверждена приказом проректора по академической работе:

№ 1804-90

	от « <u>24</u> » <u>10</u> 2024 г.
Срок сдачи законченного проекта	« <u>4</u> » <u>июня</u> 2025 г.

Исходные данные к дипломному проекту:

А) Анализ предметной области, анализ аналогичных проектов;

Б) Разработка технического задания;

В) Разработка архитектуры ПО;

Г) Проектирование и разработка понятного и интерактивного дизайна;

Д) Разработка и тестирование ПО;

Перечень подлежащих разработке в дипломном проекте вопросов: (с точным указанием обязательных чертежей): представлены 620 слайда презентации.

Рекомендуемая основная литература: из 13 наименований.

ГРАФИК

подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю и консультантам	Примечание
1. Анализ предметной области, разработка технического задания	26.03.2025	Выполнено
2. Разработка графического макета приложения	26.03.2025	Выполнено
3. Разработка серверной части приложения	02.06.2025	Выполнено
4. Разработка мобильного приложения	02.06.2025	Выполнено
5. Тестирование	02.06.2025	Выполнено
6. Написание пояснительной записки	02.06.2025	Выполнено

Подписи

консультантов и нормоконтролера на законченный дипломный проект с указанием относящихся к ним разделов проекта

Наименования разделов	Консультанты, И.О.Ф. (уч. степень, звание)	Дата подписания	Подпись	Оценка
Программное обеспечение	Шаханов Э.Р. преподаватель	05.06.2025		95
Нормоконтролер	Ердалиева З.У.	03.06.2025		

Научный руководитель  Ж.Б. Кальпеева

Задание приняла к исполнению обучающийся Камиль Д.Р.

Дата « 4 » июня 2025 г.

Алматы 2025

АНДАТПА

Бұл дипломдық жоба қалалық ортаның сапасын бағалау және тұрғындардан кері байланыс алу үшін мобильді қосымша әзірлеуге арналған. Қосымша пайдаланушыларға қалалық инфрақұрылымды бағалауға, шағымдарды фотосуреттермен және орналасқан жерімен бірге жіберуге мүмкіндік береді.

Сонымен қатар, қосымшада статистиканы көрсету және сауалнамалар жүргізу функциялары қарастырылған. Бұл құралдар қалалық билік органдарына қалалық ортаның мәселелерін жақсырақ түсінуге және оларды шешу үшін тиімді шаралар қабылдауға көмектеседі.

Пайдаланушыларды ынталандыру мақсатында қосымшада ішкі дүкенде тауарларды сатып алуға жұмсауға болатын ұпайлар түріндегі марапаттау жүйесі енгізілген.

Барлық деректер серверде сақталады және арнайы әзірленген API арқылы өңделеді. Серверлік бөлік Python (Django Rest Framework) негізінде жүзеге асырылған, ал мобильді қосымша Flutter фреймворкімен әзірленген.

АННОТАЦИЯ

Данный дипломный проект посвящён разработке мобильного приложения для мониторинга качества городской среды и обратной связи с жителями. Приложение позволяет пользователям оценивать городскую и отправлять жалобы с указанием местоположения и прикреплением фотографий.

Кроме того, в приложении реализованы функции отображения статистики и проведения опросов среди пользователей. Эти инструменты помогают городским властям лучше понимать проблемы городской среды и принимать более эффективные решения для её улучшения.

Для мотивации пользователей в приложении реализована система вознаграждений в виде баллов, которые могут быть использованы для приобретения товаров во внутреннем магазине.

Все данные сохраняются на сервере и обрабатываются с помощью специально разработанного API. Серверная часть проекта реализована на Python (Django Rest Framework), а мобильное приложение — на Flutter.

ABSTRACT

This diploma project focuses on developing a mobile application for monitoring the quality of the urban environment and providing feedback from residents. The

application allows users to evaluate urban infrastructure and submit complaints with photos and location data.

Additionally, the application includes features for displaying statistics and conducting user surveys. These tools help city authorities better understand the problems of the urban environment and make more effective decisions to improve it.

To motivate users, the application implements a rewards system in the form of points that can be spent on purchasing products in the internal store.

All data is stored on the server and processed using a specially developed API. The server-side is implemented using Python (Django Rest Framework), and the mobile application is built using Flutter.

СОДЕРЖАНИЕ

	Введение	8
1	Предпроектное исследование предметной области	10
1.1	Проблемы городской инфраструктуры Алматы и система цифрового мониторинга	10
1.1.1	Качество дорожного покрытия и транспортная система	10
1.1.2	Экологическая ситуация и загрязнение воздуха	11
1.1.3	Дефицит зелёных зон и плотная застройка	12
1.1.4	Проблемы водоотведения и наводнения	13
1.2	Обоснование потребности в системной оценке качества городской среды	13
1.2.1	Методология отбора метрик качества городской среды	13
1.2.2	Ключевые метрики оценки качества городской среды	14
1.3	Цифровой мониторинг как инструмент выявления и реагирования	14
1.4	Государственные инструменты мониторинга: ограничения и возможности	15
1.5	Анализ и сравнение аналогичных решений	16
1.6	Исследование общественного мнения	20
1.7	Обзор выбора технологий разработки приложения	21
1.7.1	Инструменты разработки мобильного приложения	21
1.7.2	Инструменты разработки серверной части приложения	22
1.7.3	Обоснование выбора системы управления базами данных	22
2	Проектирование приложения	24
2.1	Общая архитектура системы	24
2.2	Реализация серверной части приложения	24
2.2.1	Механизм аутентификации пользователей (JWT)	25
2.2.2	Модуль работы с жалобами (reports)	27
2.2.3	Модуль управления опросами (surveys)	29
2.2.4	Модуль аналитики данных (charts)	30
2.2.5	Модуль системы вознаграждений (rewards)	31
2.3	Подготовка проекта к развертыванию на облачном сервере	33
2.4	Развертывание API на облачном сервере	34
2.5	Разработка мобильного приложения	35
2.5.1	Структура проекта	35
2.5.1.1	Модели проекта	36
2.5.1.2	Сервис для управления аутентификацией и токенами	38
2.5.1.3	Сервис для взаимодействия с API	39
2.5.1.4	Структура экранов приложения	40
2.5.1.5	Разработка экранов приложения	42
2.5.2	Тестирование приложения	44

	Заключение	47
	Список использованной литературы	48
	Приложение А. Техническое задание	49
	Приложение Б. Результаты опроса	53
	Приложение В. Листинг (код) программы	59

ВВЕДЕНИЕ

В плане развития города Алматы на 2021–2025 годы [1], опубликованного на государственном портале Gov.kz, качество городской среды было выделено как важнейший фактор в формировании ключевого ресурса в развитии страны – качественного человеческого капитала.

Власти города предпринимают действия по устранению проблем городской среды, однако зачастую результаты остаются несбалансированными между районами. Городское пространство не реализует возможностей для равномерного развития всех частей города, что может быть связано в том числе с отсутствием эффективного инструмента для мониторинга и сравнения показателей качества между районами. Такая ситуация усложняет принятие обоснованных управленческих решений и замедляет развитие города.

Актуальность дипломного проекта заключается в необходимости повышения эффективности взаимодействия граждан и городских служб, а также создания платформы для мониторинга городской среды. В современных городах жители ежедневно сталкиваются с проблемами городской инфраструктуры, однако процесс их решения затруднён из-за недостаточной прозрачности работы ответственных служб и отсутствия единой системы мониторинга. Разрабатываемое мобильное приложение является актуальным, так как оно обеспечит удобный и быстрый способ подачи жалоб, отображения аналитических данных и получения обратной связи. Внедрение такой платформы позволит не только повысить эффективность работы городских служб, но и улучшить качество городской среды за счёт цифровизации процессов управления и активного вовлечения граждан.

Объектом исследования является процесс взаимодействия граждан с городскими службами с целью решения проблем городской среды и получения информации о её состоянии.

Предметом работы — мобильное приложение, предназначенное для мониторинга качества городской среды, подачи жалоб, а также обеспечения обратной связи с ответственными органами.

Цель дипломного проекта — разработка мобильного приложения, которое обеспечит удобный и эффективный процесс фиксации и отправки жалоб, предоставит доступ к статистике и аналитике городской среды, а также будет способствовать более прозрачной и результативной коммуникации между жителями и органами власти.

В соответствии с целью определены следующие задачи:

- 1) Выделить ключевые метрики оценки качества городской среды.
- 2) Провести анализ существующих решений и выявить их недостатки.
- 3) Разработать концепцию мобильного приложения, включая функциональные требования.
- 4) Разработать интерфейс (UI/UX), обеспечивающий удобство работы с приложением.

5) Реализовать основные модули приложения, включая подачу жалоб, их отслеживание и систему уведомлений.

6) Внедрить систему поощрений для пользователей в виде баллов и наград.

7) Протестировать работоспособность и удобство использования приложения.

1 Предпроектное исследование предметной области

1.1 Проблемы городской инфраструктуры Алматы и система цифрового мониторинга

Алматы – крупнейший мегаполис Казахстана, который сталкивается с комплексом инфраструктурных и экологических проблем. Среди них – неудовлетворительное состояние дорожной сети, загрязнение воздуха, нехватка зеленых зон и проблемы с водоотведением. Эти проблемы напрямую влияют на качество жизни горожан и требуют системного подхода к выявлению и решению. В данном разделе рассматриваются ключевые проблемы городской среды, предлагаются способы их мониторинга с использованием цифровых технологий и разрабатываются метрики для оценки состояния городской инфраструктуры.

1.1.1 Качество дорожного покрытия и транспортная система

Одной из наиболее острых проблем города является неудовлетворительное состояние дорожной инфраструктуры. Алматы испытывает хронические трудности, связанные с ямами, трещинами и разрушением асфальтового покрытия. В зимний период из-за частых перепадов температур ситуация усугубляется, создавая дополнительные риски для автомобилистов. В рейтинге качества дорог Всемирного экономического форума (WEF), проведенного в 2019 году (см. Рисунок 1), Казахстан занял 93-е место из 137 стран, получив оценку 3,6 балла из 7 возможных [2].



Рисунок 1.1 – Рейтинг качества дорог, предоставленный Всемирным экономическим форумом

Помимо этого, город страдает от высокой загруженности дорог. Согласно данным аналитических центров, средняя скорость движения в часы пик составляет менее 20 км/ч, что значительно увеличивает время в пути и негативно сказывается на повседневной жизни горожан. Основные причины пробок

связаны с несбалансированной системой общественного транспорта и нехваткой велосипедных дорожек.

В ряде городов Европы и США активно используются мобильные приложения, позволяющие гражданам оперативно сообщать о дорожных дефектах. Например, в США приложение SeeClickFix позволяет жителям отправлять фото и описание проблем, которые затем автоматически направляются в соответствующие службы.

Приложение CityVoice позволит жителям Алматы сообщать о повреждениях дорожного покрытия, приложив фото и геолокацию. Система уведомлений будет информировать пользователей о статусе их заявки. Аналитические данные помогут городским службам эффективнее планировать ремонтные работы и оперативно реагировать на критические проблемы.

1.1.2 Экологическая ситуация и загрязнение воздуха

Алматы входит в число самых загрязненных городов мира по данным Всемирной организации здравоохранения (ВОЗ). Основными источниками вредных выбросов являются автомобильный транспорт, промышленная деятельность и бытовое отопление.

В зимний период загрязнение воздуха значительно возрастает из-за использования угольного топлива на ТЭЦ и неблагоприятных погодных условий. Согласно данным мирового рейтинга по индексу качества воздуха (AQI) за 2024 год, в зимний период средняя концентрация PM2.5 в городе превышает допустимые нормы ВОЗ более чем в 7 – 10 раз [3] (см. Рисунок 2), что увеличивает риски развития заболеваний дыхательной системы и сердечно-сосудистых недугов [4].

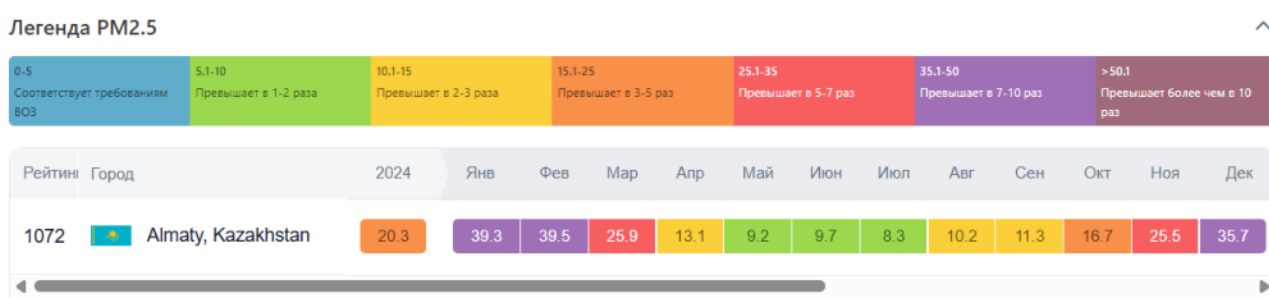


Рисунок 1.2 – график качества воздуха в Алматы за 2024 г. по данным IQAir

В Пекине успешно внедрена программа Blue Sky, которая использует данные от граждан для мониторинга загрязнения воздуха. В Лондоне приложение CityAir помогает жителям следить за качеством воздуха и сообщать

о проблемах, что позволяет местным властям более точно регулировать транспортные потоки и внедрять экологические инициативы.

При помощи CityVoice пользователи смогут сообщать о фактах сильного загрязнения воздуха, выбросах промышленных предприятий и сжигании отходов. Это позволит властям оперативно реагировать на нарушения экологических норм, а также формировать долгосрочную стратегию по улучшению качества воздуха.

1.1.3 Дефицит зеленых зон и плотная застройка

Алматы в прошлом был городом с большим количеством зеленых насаждений, однако в последние годы площадь парков и скверов значительно сократилась.

По данным Управления развития общественных пространств города Алматы [5], на 2024 год в городе насчитывается:

- 12 парков
- 141 сквер
- 31 пешеходная зона
- 21 бульвар
- 326 зеленых зон
- 11 набережных

Общая площадь зеленых территорий составляет 977 гектаров.

Эксперты и экологи отмечают, что несмотря на усилия по озеленению, в Алматы наблюдается сокращение площади зеленых насаждений. Основными причинами являются:

– Активная застройка: расширение дорожной инфраструктуры и строительство новых объектов часто сопровождаются вырубкой деревьев и уменьшением зеленых территорий.

– Уплотнение городской среды: увеличение площадей, покрытых асфальтом и бетоном, приводит к снижению естественного озеленения.

– Недостаточный уход за существующими насаждениями: отсутствие систематического полива и ухода за деревьями и кустарниками снижает их жизнеспособность.

Правительство города составляет стратегии по озеленению города и жители могут непосредственно помочь в предложении потенциальных зон для высадки деревьев через цифровые каналы.

К примеру, в Париже реализован проект "1000 деревьев", позволяющий жителям участвовать в выборе мест для озеленения. В Сингапуре работает приложение MyGreenSpace, через которое горожане могут предлагать участки для высадки деревьев, что повышает уровень вовлеченности населения в создание комфортной городской среды.

С помощью CityVoice жители смогут оставлять заявки на озеленение районов, предлагать места для высадки деревьев и голосовать за проекты по благоустройству. Это позволит властям учитывать мнение граждан при планировании развития городской среды.

1.1.4 Проблемы водоотведения и наводнения

Из-за особенностей рельефа Алматы подвержен подтоплениям после сильных осадков. Нехватка ливневой канализации приводит к образованию временных водоемов на дорогах и во дворах жилых домов, что затрудняет передвижение транспорта и пешеходов. На сайте gov.kz был опубликован анализ технического состояния водохозяйственных объектов, согласно которому основными причинами подтоплений являются арыки не соединенные с ливневой канализацией, износ старой арычной системы, вывод арыков на дороги и забитые коллекторы [6].

В Гамбурге действует система Regen–Check, позволяющая гражданам сообщать о подтоплениях в реальном времени. В результате городские службы оперативно направляют ресурсы на ликвидацию последствий осадков. В Токио применяются автоматизированные системы водоотведения, информация для которых также поступает от жителей через мобильные приложения.

CityVoice позволит жителям Алматы оперативно сообщать о проблемах с водоотведением. Интерактивная карта поможет властям выявлять уязвимые районы и разрабатывать стратегии по модернизации системы ливневой канализации.

1.2 Обоснование потребности в системной оценке качества городской среды

В условиях стремительной урбанизации устойчивое развитие невозможно без понимания и мониторинга городской среды. В этой части обосновывается необходимость перехода от разрозненных жалоб и наблюдений к структурированной системе метрик, которые отражают как объективное состояние инфраструктуры, так и субъективное восприятие горожан.

1.2.1 Методология отбора метрик качества городской среды

Для системной оценки городской среды необходимо определить метрики, отражающие объективное состояние инфраструктуры. Определение ключевых

показателей осуществлялось с учётом рекомендаций, основанных на локальном и международном опыте. Подробнее о них ниже.

UN–Habitat’s Urban Monitoring Framework (UMF) – международная система показателей, разработанная для комплексной оценки и мониторинга городского развития [7]. Этот фреймворк отражает глобальные приоритеты устойчивого развития, в частности цели, обозначенные в Целях устойчивого развития ООН (SDGs), включая ЦУР №11 – «Устойчивые города и населенные пункты» [8].

UMF учитывает важнейшие аспекты, которые напрямую влияют на качество городской жизни и устойчивость городской среды:

- Устойчивая мобильность – показатель оценивает доступность и эффективность транспортной системы, учитывая приоритет пешеходов, велосипедистов и общественного транспорта. Он необходим для снижения загруженности дорог, сокращения выбросов CO₂ и повышения удобства перемещений жителей.

- Экологические параметры – включают оценку уровня загрязнения воздуха, состояния водных ресурсов и зеленых зон. Эти факторы критически важны для здоровья населения и создания комфортной городской среды.

- Доступ к зеленым зонам – важнейший компонент городской инфраструктуры. Зеленые насаждения способствуют улучшению микроклимата, снижению уровня шума и обеспечивают психологический комфорт жителей. В развитых странах минимально рекомендуемая доля зеленых насаждений в городе – 10–15%, что соответствует рекомендациям ВОЗ и UMF.

- Транспортная связанность – показатель, позволяющий оценить, насколько эффективно и удобно связаны между собой различные районы города. Это особенно важно для равномерного социально–экономического развития и повышения мобильности горожан.

Дополнительно, при выборе ключевых показателей была проанализирована статья казахстанских ученых из Института экономики КН МОН РК «Качество городского пространства как фактор устойчивого экономического развития г. Алматы» [9]. В данной статье подчеркивается необходимость комплексного подхода к оценке качества городской среды, включающего такие ключевые показатели, как безопасность, комфорт, экологическая устойчивость и социальная справедливость. Авторы отмечают, что универсальная система индикаторов позволяет не только выявлять проблемные зоны, но и оценивать эффективность реализуемых городских инициатив.

1.2.2 Ключевые метрики оценки качества городской среды

На основании анализа вышеописанных источников были выделены ключевые метрики, которые отражают объективные характеристики

инфраструктуры и напрямую влияют на восприятие города жителями. Именно эта связка – физическое качество среды + субъективная оценка горожан – позволяет выстраивать реалистичную модель оценки городской среды и разрабатывать эффективные механизмы реагирования на проблемы.

Список выбранных метрик:

- 1) Транспортная доступность
- 2) Состояние дорожной инфраструктуры
- 3) Загруженность/пробки
- 4) Озеленение и благоустройство
- 5) Шумовое загрязнение
- 6) Водоотведение
- 7) Качество воздуха
- 8) Чистота территорий
- 9) Уличное освещение
- 10) Безопасность (общая)
- 11) Доступность городской среды
- 12) Развитость инфраструктуры досуга
- 13) Жилищные условия
- 14) Коммунальные услуги

Эти метрики охватывают ключевые инфраструктурные проблемы, описанные в предыдущем разделе, позволяют комплексно оценить качество городской среды и легко поддаются цифровой оценке через пользовательские опросы и обращения.

1.3 Цифровой мониторинг как инструмент выявления и реагирования

Цифровое приложение позволяет совмещать объективные и субъективные данные. Пользователи оценивают каждый из параметров по 10–балльной шкале и при необходимости оставляют жалобу (например, на плохое дорожное покрытие, шум или загрязнение). Такой подход позволяет выявлять зависимости:

Пример:

- Чистота улиц и дворов: оценка 7.8 / количество жалоб: 18
- Качество воздуха: оценка 3.2 / количество жалоб: 52
- Уровень шума: оценка 9.3 / количество жалоб: 3

Визуализируя эти данные, можно строить корреляционные графики, где низкий уровень субъективной оценки сопровождается высоким уровнем жалоб и наоборот. Это дает аналитикам и городским властям инструменты для приоритизации проблем. Кроме того, агрегированные данные можно анализировать по районам, времени суток и другим параметрам.

1.4 Государственные инструменты мониторинга: ограничения и возможности

В Казахстане вопросы мониторинга городской среды находятся в ведении нескольких государственных структур, таких как:

- Бюро национальной статистики – собирает обобщенные данные по жилищным условиям, транспорту, уровню загрязнения воздуха и водных ресурсов, но не фокусируется детально на городской инфраструктуре.

- Министерство экологии и природных ресурсов РК – предоставляет статистику по загрязнению воздуха, воде и почве, однако в отрыве от социальных факторов.

- Акиматы городов (например, Open Almaty) – принимают жалобы от граждан и ведут их учет, но, как правило, данные не агрегируются в открытом виде или публикуются нерегулярно.

- Центры урбанистики (например, Urban Forum Kazakhstan или Центр урбанистики Алматы) проводят периодические исследования и опросы, но такие инициативы, как правило, разовые и не масштабируются на постоянной основе.

Государственные структуры предоставляют разрозненные данные по параметрам городской среды. Жалобы граждан фиксируются, но не публикуются в систематизированном виде и не имеют визуального представления.

Предлагаемое приложение способно изменить эту ситуацию: оно позволяет преобразовать индивидуальные ощущения и замечания горожан в структурированные, визуализированные данные, которые могут быть полезны как для жителей, так и для городских администраций. Такая модель не только устраняет информационный разрыв между состоянием городской инфраструктуры и ее восприятием, но и создает основу для регулярного мониторинга, обратной связи и более прозрачного городского управления.

1.5 Анализ и сравнение аналогичных решений

Для того, чтобы разработать наиболее эффективное приложение, которое будет решать все поставленные проблемы, был проведен анализ конкурентов.

Анализ позволил выявить наиболее эффективные механизмы взаимодействия пользователей с государственными службами, определить ключевые преимущества и недостатки существующих платформ, а также учесть факторы, влияющие на уровень вовлеченности граждан в процессы управления городской инфраструктурой.

В рамках данного исследования были рассмотрены зарубежные (см. Таблица 1) и локальные (см. Таблица 2) аналоги, работающие по схожему принципу.

Таблица 1 – Аналогичные решения на международном рынке

Название	Описание	Плюсы	Минусы
FixMyStreet, Великобритания	Интернет–ресурс, позволяющий пользователям фиксировать инфраструктурные проблемы с последующей автоматической отправкой в соответствующие городские инстанции. Представлен в виде веб–сайта и мобильного приложения.	–простая процедура подачи жалобы: требуется лишь указать местоположение, описание проблемы и прикрепить фото; –интерактивная карта, на которой отображаются все поданные жалобы с возможностью фильтрации по категориям и статусу; –возможность подписки на обновления статуса уже существующей жалобы, что способствует вовлеченности пользователей; –автоматическая маршрутизация обращений в ответственные службы на основе геолокации.	–ответственные службы не обновляют статус жалобы, и пользователи вынуждены самостоятельно отслеживать изменения; –отсутствие личного кабинета для государственных органов, что ограничивает функциональность платформы с административной стороны.
SeeClickFix, США	Платформа для сообщений о проблемах городской инфраструктуры с возможностью отслеживания статуса.	–возможность комментирования и голосования за уже существующие жалобы, что помогает выявлять наиболее	требуется обязательная регистрация, что может снизить порог входа для новых пользователей.

		актуальные проблемы; –автоматическая передача жалоб в муниципальные службы; –интеграция с сgm–системами органов местного самоуправления, что ускоряет процесс обработки обращений; –предоставление статистики и аналитики по обращениям в разрезе отдельных районов.	
Наш город, Россия	Платформа, разработанная для взаимодействия жителей Москвы с органами исполнительной власти. Граждане могут сообщать о проблемах городской среды, а власти обязуются оперативно реагировать на обращения.	–возможность фиксировать широкий спектр городских проблем; –доступность через веб–версию и мобильное приложение.	–сложный пользовательский интерфейс, что может затруднять процесс подачи обращений; –отсутствие статистики по жалобам.

Таблица 2 – аналогичные решения на рынке Казахстана

Название	Описание	Плюсы	Минусы
eOtinish	Сервис, позволяющий гражданам подавать официальные	–официальный канал связи с государственными органами;	–сложный процесс подачи заявок, требующий

	обращения и жалобы в государственные органы. Доступен через веб-сайт и мобильное приложение eGov Mobile.	–возможность отслеживания статуса обращения; –доступность через различные платформы.	регистрации и подтверждения личности; –недостаточно интуитивный интерфейс, который может затруднять использование сервиса.
Open Almaty	Портал общественной приемной акимата города Алматы, предназначенный для приема обращений граждан и предоставления информации о государственных услугах.	–официальный канал связи с акиматом алматы; –возможность подачи обращений по широкому спектру городских проблем; –предоставление информации о государственных услугах и новостях города.	–возможны задержки в обработке обращений из-за высокой нагрузки на систему; –требуется регистрация для подачи обращения; –статистика по жалобам предоставляется только в ежегодном отчете на официальном сайте, нет возможности получить ее в реальном времени в приложении.
TazaQazBot	Telegram-бот, запущенный в рамках акции "Таза Қазақстан", предназначен для сообщений о проблемах города и экологической обстановке.	–простота использования через популярный мессенджер; –возможность отправки фото и описания проблемы; –автоматическая передача информации	–ограниченные возможности по отслеживанию статуса обращения; –отсутствие возможности просматривать обращения

		соответствующим службам.	других пользователей; –доступен только в telegram.
--	--	-----------------------------	---

На основе этого анализа в разработке CityVoice можно учесть следующие аспекты:

–упрощение процесса подачи жалобы – минимизация требований к регистрации и возможность подачи обращений в несколько кликов;

–автоматическая маршрутизация обращений – в перспективе интеграция с государственными системами для оперативного направления заявок в нужные службы;

–отслеживание статуса в реальном времени – предоставление пользователям возможности видеть динамику рассмотрения их обращения;

–вовлечение пользователей – система вознаграждения пользователей для поощрения их активности;

–предоставление аналитики и статистики – отображение статистических данных по количеству и характеру жалоб по каждому району города как для горожан, так и для ответственных лиц.

1.6 Исследование общественного мнения

Для более точного определения приоритетных проблем городской среды и подтверждения релевантности выбранных показателей был проведен собственный онлайн–опрос среди жителей Алматы (см. Приложение Б). В нем уточнялось, какие именно проблемы жители считают наиболее острыми (плохое состояние дорог, загрязнение воздуха, пробки, нехватка зеленых зон и т.д.), а также интерес к цифровым решениям для мониторинга качества городской среды.

Опрос был проведен среди 110 респондентов с использованием Google Форм и социальных сетей. Респонденты представляли разные возрастные группы и районы города.

Ключевые результаты опроса продемонстрированы в Таблице 3.

Таблица 3 – ключевые результаты опроса.

Вопрос	Ключевые результаты
Какие проблемы городской среды вы считаете наиболее актуальными?	88% – загрязнение воздуха 74% – пробки и загруженность 63% – нехватка зеленых насаждений 39% – шумовое загрязнение

Насколько интересны вам цифровые решения для мониторинга?	70% респондентов заинтересованы в цифровых решениях 25% – возможно воспользовались бы
Какие функции цифрового сервиса наиболее важны?	Отправка жалоб и предложений, просмотр статистики, получение вознаграждений

Результаты опроса подтвердили актуальность разработки мобильного приложения для цифрового мониторинга городской среды. Участие жителей и их заинтересованность в использовании подобных цифровых инструментов подчеркивают необходимость создания сервиса, который позволит улучшить качество жизни и повысить прозрачность работы городских властей.

1.7 Обзор выбора технологий разработки приложения

Исходя из цели проекта, был выбран стек технологий, который не только покрывает все необходимые функциональные аспекты, но и обеспечивает возможность дальнейшего масштабирования без кардинального изменения архитектуры. При выборе инструментов особое внимание уделялось производительности, удобству разработки и поддержке современных стандартов.

Разработка полнофункционального приложения включает следующие компоненты:

- бэкенд – серверная часть, отвечающая за бизнес–логику, обработку запросов и взаимодействие с базой данных.
- фронтенд – клиентские приложения для ios и android, обеспечивающие удобный пользовательский интерфейс.
- база данных – система хранения и управления структурированными данными приложения.

1.7.1 Инструменты разработки мобильного приложения

Для разработки клиентской части приложения выбран фреймворк Flutter, так как он предоставляет кроссплатформенное решение, позволяющее разрабатывать единый код для iOS и Android. Это значительно сокращает время на разработку, упрощает поддержку и снижает затраты, так как отпадает необходимость в создании двух отдельных нативных приложений.

Одним из ключевых преимуществ Flutter является его высокопроизводительный графический движок Skia, который обеспечивает быструю отрисовку интерфейса, плавную анимацию и высокую отзывчивость

даже на устройствах с ограниченными ресурсами. Это особенно важно для разрабатываемого приложения, в котором пользователи будут активно взаимодействовать с картами, загружать изображения и отслеживать изменения статусов жалоб.

Еще одним важным аспектом является гибкость UI-компонентов. Flutter позволяет разрабатывать кастомизированные пользовательские интерфейсы, следуя рекомендациям Material Design для Android и Cupertino для iOS. Это гарантирует, что приложение будет соответствовать платформенным стандартам и обеспечивать удобство работы для пользователей обоих типов устройств.

1.7.2 Инструменты разработки серверной части приложения

Для реализации серверной части приложения выбрана технология Django REST Framework (DRF). Этот фреймворк предоставляет мощную инфраструктуру для создания RESTful API, что делает его удобным для интеграции с мобильными приложениями. Гибкость и масштабируемость DRF позволяют реализовать сложную бизнес-логику и поддерживать высокую нагрузку, что особенно важно при увеличении числа пользователей и количества обрабатываемых жалоб.

Одним из ключевых преимуществ DRF является его встроенная система безопасности. Фреймворк автоматически защищает API от SQL-инъекций, и CSRF-атак, а также поддерживает различные механизмы аутентификации, включая JWT (JSON Web Token) и OAuth. Это гарантирует, что персональные данные пользователей останутся в безопасности, а доступ к различным функциям приложения будет ограничен в зависимости от роли пользователя (житель города, модератор, исполнитель).

DRF также предлагает гибкие механизмы сериализации данных, позволяя преобразовывать данные между JSON-форматом и объектами Python, что упрощает обработку запросов. Встроенные классы представлений (ViewSet) и маршрутизация делают API лаконичным и удобным в использовании. Фреймворк предоставляет возможность автоматической генерации документации облегчает интеграцию бэкенда с мобильным приложением.

1.7.3 Обоснование выбора системы управления базами данных

В качестве системы хранения данных выбрана PostgreSQL, поскольку она сочетает в себе производительность, надежность и гибкость. Эта СУБД поддерживает ACID-транзакции, что гарантирует целостность данных даже при высокой нагрузке.

Одним из ключевых факторов в пользу PostgreSQL является поддержка географических данных через расширение PostGIS. В приложении пользователи будут отправлять жалобы с указанием местоположения, а исполнители смогут фильтровать заявки по районам. Благодаря PostGIS можно оптимизировать работу с координатами, строить геозапросы и анализировать распределение жалоб на карте, что делает базу данных более функциональной для решения задач приложения.

Еще одним преимуществом PostgreSQL является эффективная работа с индексами, что ускоряет поиск данных. Это особенно важно при росте числа пользователей и увеличении объема хранимой информации. Система поддерживает настройку репликации и резервного копирования, что позволяет предотвратить потерю данных в случае сбоев.

2 Проектирование приложения

2.1 Общая архитектура системы

Проект построен на основе клиент–серверной архитектуры. Серверная часть и база данных размещены на облачном сервере Render. Мобильное приложение, выступающее в качестве клиента, взаимодействует с сервером посредством API. В зависимости от необходимой операции клиент отправляет HTTP–запрос на соответствующий endpoint (URL) сервера, а сервер возвращает ответ в формате JSON.

При GET–запросах (получение данных) JSON–сообщение содержит запрашиваемую клиентом информацию. При POST– и PUT–запросах сервер вызывает соответствующее представление, отвечающее за выполнение внутренней логики, и производит необходимые операции с данными: запись или изменение.

База данных служит центральным хранилищем информации, взаимодействует с серверным API и предоставляет данные по запросу.

На рисунке 2.1 представлена графическая схема архитектуры проекта, в которой показаны его основные компоненты и их взаимодействие.

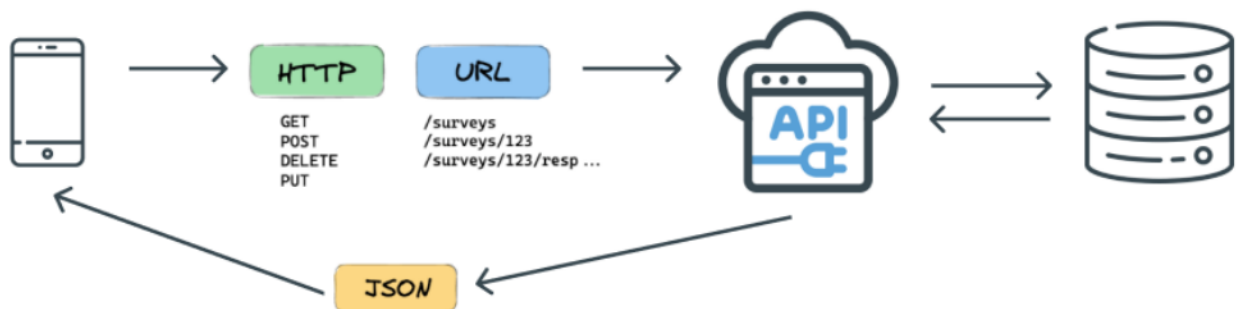


Рисунок 2.1 – Схема архитектуры приложения

2.2 Реализация серверной части приложения

Серверная часть приложения разработана с использованием фреймворка Django Rest Framework (DRF), который предоставляет удобные инструменты для построения RESTful API. Основная структура проекта включает следующие модули:

– core – центральный модуль проекта. В нем хранится файл settings.py, где задаются ключевые конфигурации: пути к статическим и медиа–файлам, параметры авторизации, настройки базы данных и список всех установленных приложений (INSTALLED_APPS). Здесь же осуществляется маршрутизация

URL-адресов проекта через файл `urls.py`, связывающий URL-адреса с соответствующими представлениями (`view`).

- `charts` – модуль для отображения статистики по данным, полученным от пользователей. Здесь реализованы представления для предоставления статистических данных по количеству и характеру жалоб и результатам опросов.

- `reports` – модуль, отвечающий за обработку жалоб пользователей. Включает модели, сериализаторы и представления для создания, чтения, обновления и удаления заявок.

- `rewards` – модуль, реализующий систему вознаграждений пользователей (баллы и магазин), включая логику накопления и списания баллов.

- `surveys` – модуль для работы с опросами пользователей, включая модели вопросов и ответов, а также API для проведения опросов.

На Рисунке 2.2 изображена схема структуры API.

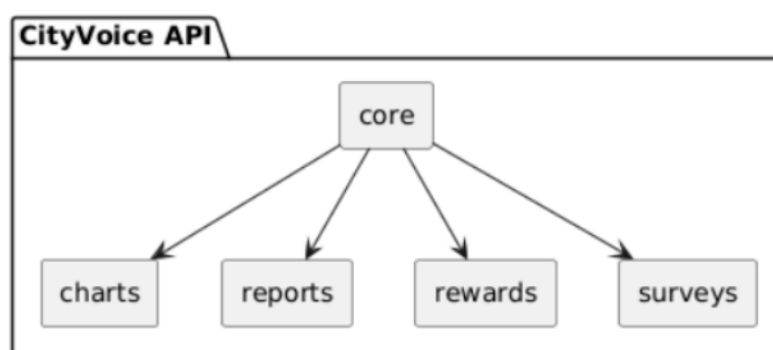


Рисунок 2.2 – структура API

2.2.1 Механизм аутентификации пользователей (JWT)

В модуле `core` был настроен механизм аутентификации при помощи JSON Web Token (JWT), реализованный с помощью библиотеки `django-rest-framework-simplejwt`.

JWT – это технология создания токенов доступа, при помощи которых происходит подтверждение подлинности аккаунта [10]. После аутентификации пользователь получает два токена, сгенерированных на сервере и подписанных секретным ключом. Первый токен – `access token`, с помощью которого происходит идентификация пользователя, второй – `refresh token`, данный токен используется для обновления `access`-токена. Для обеспечения безопасности и предотвращения перехвата, `access` токен действителен в рамках короткого времени, после которого он должен быть обновлен `refresh` токеном, который обладает более долгим сроком действия. После истечения `refresh`-токена, пользователь должен повторно авторизоваться для получения новой пары токенов.

Таким образом, при использовании JWT токенов, количество отправки идентификационных данных (логин, пароль) пользователем значительно сокращается, что обеспечивает более надежную защиту пользовательских данных.

Для подключения аутентификации через JWT, в файле settings.py была переопределена система аутентификации по умолчанию (см. Рисунок 2.3).

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend'],
}
```

Рисунок 2.3 – настройка аутентификации через JWT

Также были настроены параметры токена, такие как срок действия access-токена, поле в модели пользователя, используемое для идентификации, класс, используемый для генерации токена (см. Рисунок 2.4).

```
SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=60),
    "USER_ID_FIELD": "id",
    "USER_ID_CLAIM": "user_id",
    "AUTH_TOKEN_CLASSES": ("rest_framework_simplejwt.tokens.AccessToken",),
}
```

Рисунок 2.4 – настройка параметров JWT

В зависимости от роли пользователя в приложении – пользователь или представитель ответственной городской службы – ему доступен разный функционал. При аутентификации пользователя необходимо передать флаг, указывающий на его роль. Для того, чтобы реализовать данный функционал был переопределен стандартный сериализатор TokenObtainPairSerializer таким образом, чтобы после аутентификации вместе с базовой информацией о пользователе передавался флаг is_staff, разграничивающий функциональность между обычным пользователем и ответственным лицом (см. Рисунок 2.5).

```
class CustomTokenObtainPairSerializer(TokenObtainPairSerializer):
    @classmethod
    def get_token(cls, user):
        token = super().get_token(user)
        token['username'] = user.username
        token['first_name'] = user.first_name
        token['last_name'] = user.last_name
        token['is_staff'] = user.is_staff
        return token
```

Рисунок 2.5 – кастомный сериализатор Custom TokenObtainPairSerializer

2.2.2 Модуль работы с жалобами (reports)

Модуль reports в API отвечает за всю логику создания и манипуляции над пользовательскими заявками.

Файл models.py представляет собой список и настройки моделей приложения. Каждой модели в API соответствует таблица в базе данных. Модуль reports содержит следующие модели:

Таблица 4 – модели reports

Модель	Описание	Поля модели
Category	Модель, отвечающая за категории, по которым группируются и маршрутизируются заявки, а также составляются графики оценки городской среды.	Id – уникальное поле, первичный ключ, integer. name – название категории, varchar. unit – единица измерения категории, varchar. description – описание, varchar
District	Модель, представляющая административные районы города, с возможностью хранения геометрии (границ) и изображения района.	id – уникальное поле, первичный ключ, integer. name – название района, varchar. geometry – геометрия района, MultiPolygonField. image – изображение района, url, varchar.
Report	Модель, представляющая жалобы пользователей. Каждая	id – уникальное поле, первичный ключ, integer.

	жалоба относится к определенной категории и району.	user – внешний ключ к пользователю, ForeignKey. category – внешний ключ к категории, ForeignKey. name – название заявки, varchar. description – описание заявки, varchar. image – изображение, url, varchar. latitude – широта, double. longitude – долгота, double. district – внешний ключ к району, ForeignKey. status – статус заявки, varchar (с выбором из pending, in_progress, resolved, rejected). created_at – дата и время создания, datetime (автоматически). updated_at – дата и время последнего обновления, datetime (автоматически).
Comments	Модель, представляющая комментарии ответственных лиц к заявкам. Каждый комментарий относится к конкретной жалобе.	id – уникальное поле, первичный ключ, integer. user – внешний ключ к пользователю, ForeignKey. report – внешний ключ к заявке, ForeignKey. text – текст комментария, varchar. created_at – дата и время создания комментария, datetime (автоматически).

В файле views.py определены представления, реализующие логику манипуляции с данными моделей.

При запросе на создание жалобы, в теле запроса помимо остальных данных отправляются координаты (latitude, longitude) местоположения, api перенаправляет данные на представление Report, где осуществляется автоматическая локализация местоположения в рамках города. В результате при запросе информации о заявке, пользователь видит уже не координаты, а конкретный адрес внутри района в удобном формате.

Для реализации автоматической локализации был переопределен метод `perform_create` (см. Рисунок 2.6), который вызывается на моменте создания новой записи `Report`. В методе перехватываются значения координат и создается объект класса `Point` из библиотеки `GeoDjango`, которая представляет собой точку на карте, ранее загруженной в базу. Далее определяется район, в котором находится данный адрес, путем фильтрации записей из модели `Districts` по полю `geometry` (область на карте).

```
def perform_create(self, serializer):
    lat = self.request.data.get('latitude')
    lon = self.request.data.get('longitude')
    point = Point(float(lon), float(lat))
    district = District.objects.filter(geometry__contains=point).first()

    serializer.save(user=self.request.user, district=district)
```

Рисунок 2.6 – автоматическая локализация координат жалоб

2.2.3 Модуль управления опросами (surveys)

Модуль `surveys` отвечает за реализацию проведения опросов среди пользователей приложения.

Модуль содержит две модели – `SurveyQuestions` и `SurveyAnswers`:

Таблица 5 – модели `surveys`

Модель	Описание	Поля модели
<code>SurveyQuestion</code>	Модель, представляющая вопросы опроса. Каждый вопрос связан с определенной категорией.	<code>id</code> – уникальное поле, первичный ключ, <code>integer</code> . <code>text</code> – текст вопроса, <code>varchar</code> . <code>is_active</code> – флаг активности вопроса, <code>boolean</code> (по умолчанию <code>True</code>). <code>category</code> – внешний ключ к модели <code>Category</code> , <code>ForeignKey</code> .

SurveyResponse	Модель, представляющая ответы пользователей на вопросы. Содержит оценку (rating), которую пользователь поставил для конкретного вопроса. Модель содержит внешние ключи к моделям SurveyQuestion и District, что позволяет связать каждый ответ с конкретным вопросом и районом, обеспечивая точность сбора и анализа данных.	id – уникальное поле, первичный ключ, integer. user – внешний ключ к пользователю, ForeignKey. question – внешний ключ к вопросу, ForeignKey. district – внешний ключ к району, ForeignKey. rating – оценка, integer. created_at – дата и время создания ответа, datetime (автоматически).
----------------	--	---

В файле views.py реализованы представления, которые позволяют:

- Получить список доступных вопросов (SurveyQuestionListView).
- Отправить ответы пользователя с проверкой уникальности (SurveyResponseCreateView).
- Проверить доступность опроса для пользователя (SurveyAvailabilityAPIView).

При прохождении опроса, пользователь указывает район своего проживания, что позволяет отображать статистику качества городской среды как по всему городу, так и в разрезе выбранного района.

Таким образом, модуль surveys обеспечивает возможность сбора, хранения и анализа мнений пользователей о городской среде.

2.2.4 Модуль аналитики данных (charts)

Модуль charts в API отвечает за формирование и предоставление статистических данных. В рамках этого модуля осуществляется агрегирование данных из моделей Report, District, Category и SurveyResponse с целью построения статистики по количеству жалоб в разрезе категорий и районов, а также по результатам опросов.

Комбинирование этих данных позволяет получить расширенные аналитические показатели качества городской среды, учитывающие как субъективные мнения пользователей (результаты опросов), так и объективные данные о количестве обращений. Сравнение этих показателей позволяет

выявлять возможные корреляции между восприятием горожан и фактическим состоянием городской инфраструктуры.

Логика агрегирования и предоставления статистики реализуется в представлениях в `view.py`.

`DistrictCategoryStatsView` предоставляет статистику по количеству жалоб в указанном районе (`district_id`) с возможностью фильтрации по дате. Оно возвращает общее количество жалоб и их распределение по категориям.

`CitywideCategoryCorrelationView` возвращает сводные данные по всем категориям города: среднюю оценку пользователей (субъективное восприятие) и количество жалоб (объективные данные). Таким образом, позволяет выявить возможные закономерности между восприятием и реальным положением дел.

`DistrictCategoryCorrelationView` – это представление аналогично предыдущему, но для указанного района. Оно предоставляет среднюю оценку пользователей и количество жалоб по категориям в конкретном районе, что позволяет оценить качество городской среды локально.

2.2.5 Модуль системы вознаграждений (rewards)

Модуль `rewards` реализует систему вознаграждений пользователей за использование приложения. Пользователи получают баллы за успешно закрытые заявки, а затем могут тратить эти баллы во внутреннем магазине. Таким образом, модуль обеспечивает элемент геймификации и повышает мотивацию пользователей активно участвовать в улучшении городской среды.

Модуль содержит три модели, необходимых для реализации функционала награждений: `Reward`, `Product`, `Purchase`.

Таблица 6 – модели `rewards`

Модель	Описание	Поля модели
<code>Reward</code>	Модель, представляющая вознаграждения пользователей за выполнение заявок. Каждая запись – это начисление баллов пользователю за конкретный заявку.	<code>id</code> – первичный ключ, <code>integer</code> . <code>user</code> – внешний ключ к пользователю, <code>ForeignKey</code> . <code>report</code> – внешний ключ к заявке, <code>ForeignKey</code> . <code>amount</code> – количество начисленных баллов, <code>integer</code> (по умолчанию 100). <code>created_at</code> – дата и время начисления,

		datetime (автоматически). Ограничение уникальности – для одного пользователя и одной заявки может быть только одна запись.
Product	Модель, описывающая товары виртуального магазина, которые могут быть куплены за баллы.	id – первичный ключ, integer. name – название товара, varchar. description – описание, text. price – цена в баллах, integer. image – изображение товара, url, varchar. category – категория товара, varchar.
Purchase	Модель, фиксирующая покупки пользователей в магазине.	id – первичный ключ, integer. user – внешний ключ к пользователю, ForeignKey. product – внешний ключ к продукту, ForeignKey. purchased_at – дата и время покупки, datetime (автоматически).

Представления модуля описаны ниже.

RewardViewSet – представление, позволяющее пользователю просматривать свои вознаграждения.

check_reward – функция–представление, позволяющая проверить, есть ли уже вознаграждение у пользователя за конкретную заявку (по report_id).

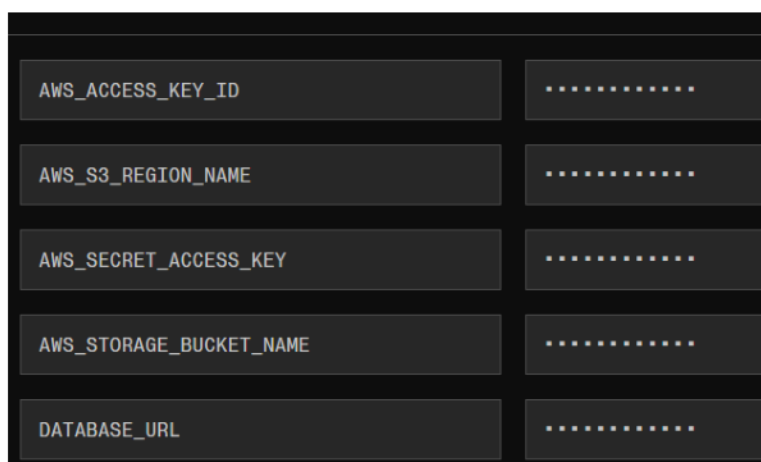
ProductViewSet – представление для получения списка всех доступных товаров в магазине.

ShopViewSet – представление, реализующее логику виртуального магазина: просмотр текущего баланса пользователя (сумма всех полученных баллов минус сумма всех потраченных), покупка товара за баллы с проверкой доступной суммы на балансе.

2.3 Подготовка проекта к развертыванию на облачном сервере

Для корректной работы серверной части приложения в облачной среде была произведена детальная настройка файла `settings.py` проекта.

Безопасное хранение переменных окружения. Поскольку проект размещен на GitHub для обеспечения безопасного хранения ключей, используемых в проекте, все они были вынесены в файл `.env.dev`. Данный файл содержит переменные окружения для локальной разработки. В `settings.py` указан путь к файлу `.env.dev`, переменные из файла подгружаются при помощи функции `load_env` библиотеки `dotenv`. В облачном сервере переменные окружения указываются напрямую в панели управления (см. Рисунок 2.7).



The image shows a screenshot of a server management interface with a dark theme. It displays a list of environment variables in a table-like structure. Each row has a label on the left and a masked value (dots) on the right.

AWS_ACCESS_KEY_ID	
AWS_S3_REGION_NAME	
AWS_SECRET_ACCESS_KEY	
AWS_STORAGE_BUCKET_NAME	
DATABASE_URL	

Рисунок 2.7 – хранение переменных окружения на сервере

Подключение к базе данных. Для подключения к облачной базе данных PostgreSQL используется библиотека `dj_database_url`, которая позволяет удобно указывать строку подключения (`DATABASE_URL`).

Кроме того, включен движок PostGIS, обеспечивающий работу с геоанализом. На Рисунке 2.8 представлено подключение проекта к базе данных.

```
DATABASES = {
    'default': dj_database_url.config(default=os.getenv("DATABASE_URL"))
}

DATABASES['default']['ENGINE'] = 'django.contrib.gis.db.backends.postgis'
```

Рисунок 2.8 – настройка подключения к базе данных в проекте

Подключение хранилища S3 для медиафайлов. При работе в продакшн-режиме (если `DEBUG = False`) для загрузки и хранения картинок используется

облачное хранилище S3, настроенное через библиотеку `django-storages` (см. Рисунок 2.9).

В переменных окружения указываются ключи доступа, имя бакета (хранилище), регион, а также параметры доступа к файлам.

```
# Хранилище для медиафайлов
if not DEBUG:
    STORAGES = {
        "default": {
            "BACKEND": "storages.backends.s3boto3.S3Boto3Storage",
        },
        "staticfiles": {
            "BACKEND": "django.contrib.staticfiles.storage.StaticFilesStorage",
        },
    }
```

Рисунок 2.9 – подключение облачного хранилища Amazon S3

Настройка логирования. Конфигурация логирования позволяет видеть основные ошибки и предупреждения в консоли на сервере, что упрощает отладку и сопровождение (см. Рисунок 2.10).

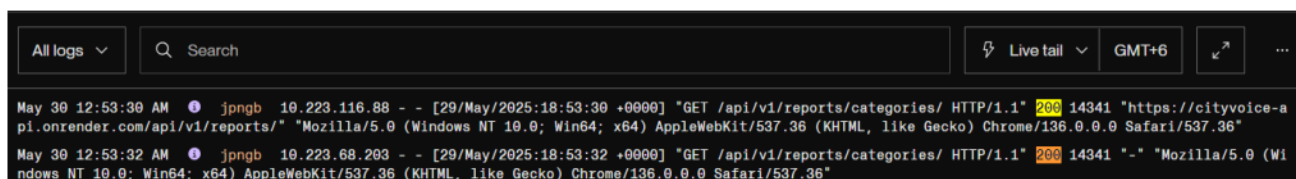


Рисунок 2.10 – панель с логами на сервере

2.4 Развертывание API на облачном сервере

Для развертывания сервиса (Django API) был выбран облачный сервер Render. Render предоставляет простую и удобную платформу для размещения веб-приложений, обеспечивая их стабильную работу и минимизируя затраты на настройку инфраструктуры.

Этапы развертывания описаны ниже.

Подготовка зависимостей. В корневой директории проекта был создан файл `requirements.txt`, в котором указаны все необходимые зависимости Python.

Настройка базы данных. Для хранения всех данных приложения используется отдельный экземпляр базы данных PostgreSQL, предоставляемый Render. Дополнительно на сервере было установлено расширение PostGIS для работы с гео Данными.

Загрузка проекта на сервер. Исходный код приложения размещен в репозитории GitHub. В панели управления Render репозиторий подключается к

проекту на GitHub, после чего запускается автоматическая сборка и развертывание приложения.

Мониторинг и поддержка работы API. На панели управления Render можно в реальном времени отслеживать статус приложения, просматривать логи и быстро реагировать на возможные ошибки или сбои. Это существенно упрощает администрирование и обеспечивает высокую доступность приложения.

API-сервис был успешно развернут на облачном сервере Render и доступен по адресу: <https://cityvoice-api.onrender.com/api/v1>.

2.5 Разработка мобильного приложения

2.5.1 Структура проекта

Исходный код мобильного приложения размещен в директории `lib/`, которая содержит основные модули и файлы, необходимые для реализации пользовательского интерфейса и взаимодействия с сервером. Ключевые каталоги внутри `lib/`:

- `models` – содержит модели данных, которые описывают структуру объектов, получаемых от API (например, жалобы, районы, опросы).
- `services` – сервисный слой, отвечающий за сетевые запросы и взаимодействие с API (авторизация, получение и отправка данных).
- `pages` – страницы приложения (экраны), которые реализуют пользовательский интерфейс и логику взаимодействия с пользователем.

В корне проекта расположен файл `pubspec.yaml`, в котором указаны зависимости (пакеты Flutter, такие как `http`, `flutter_map`, `jwt_decoder`). Эти зависимости позволяют подключить библиотеки для отображения карт, работы с HTTP-запросами, хранения токенов и других функций.

Кроме того, в проекте присутствуют конфигурационные файлы для платформы Android (папка `android/`), которые содержат:

- настройки сборки проекта (Gradle),
- разрешения для доступа к камере и интернету (`AndroidManifest.xml`),
- параметры, необходимые для корректного взаимодействия с API и сервером.

Эта структура позволяет удобно организовать исходный код и упрощает поддержку и масштабирование приложения. На Рисунке 2.11 изображена схема структуры проекта.

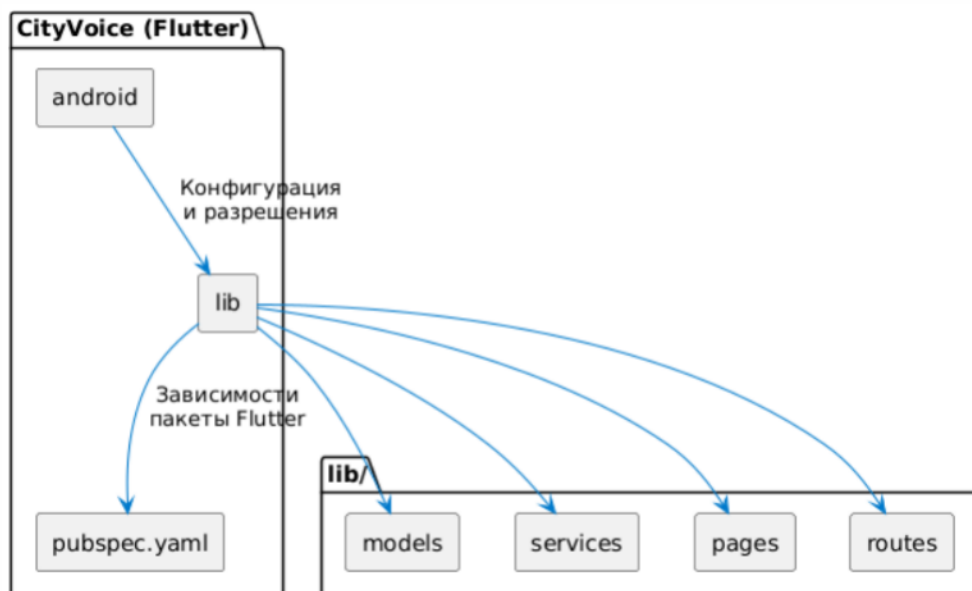


Рисунок 2.11 – структура проекта на flutter

2.5.1.1 Модели проекта

При разработке приложения особое внимание было уделено правильной организации и использованию моделей данных. Модели представляют собой структуры, которые описывают данные, получаемые от API и используемые внутри приложения.

Каждая модель описывает конкретный объект (например, жалобу, район, товар в магазине) с набором свойств, соответствующих данным, которые возвращает сервер. Это позволяет легко и удобно работать с данными, не привязываясь напрямую к JSON-формату.

Использование моделей в проекте обеспечивает приведение данных с API к удобному формату для использования в приложении. Применение моделей является общепринятой практикой в разработке на Flutter и описано в официальной документации [11].

В таблице 7 описаны модели, используемые в проекте.

Таблица 7 – модели приложения

Модель	Описание	Источник API
Report	Модель, представляющая жалобу. Включает данные о проблеме, ее статусе, местоположении, авторе и комментариях.	/api/v1/reports/ – получение списка жалоб и подробной информации о них.

	Используется для отображения подробной информации о каждой жалобе в мобильном приложении.	
District	Модель, представляющая административный район. Содержит идентификатор, название и изображение района.	/api/v1/reports/districts / – получение списка всех районов города.
DistrictStats	Модель, представляющая сводную статистику по количеству жалоб внутри района. Содержит идентификатор и название района, общее количество жалоб и список категорий с их количеством.	/api/v1/charts/districts/{district_id}/stats/ – получение статистики жалоб по конкретному району.
ComplaintStats	Вложенная модель, представляющая статистику по каждой категории внутри района (количество жалоб по каждой категории).	Используется как часть ответа вышеуказанного API.
SurveyQuestion	Модель, представляющая вопрос опроса. Содержит идентификатор и текст вопроса.	/api/v1/statistics/survey/questions/ – получение списка доступных вопросов для опроса.
CategoryCorrelationStat	Модель, представляющая объединенную статистику по категориям. Содержит идентификатор категории, название, среднюю оценку пользователей (субъективное восприятие) и	/api/v1/charts/statistics/citywide/correlation/ – получение сводной статистики по всем категориям в рамках всего города. /api/v1/charts/statistics/survey/statistics/by–

	количество жалоб (объективные данные).	district/\$districtId/ – получение сводной статистики по всем категориям в рамках выбранного района.
Product	Модель, представляющая товар в виртуальном магазине приложения. Содержит идентификатор, название, описание, цену (в баллах), ссылку на изображение и категорию товара.	/api/v1/rewards/products/ – получение списка доступных товаров в магазине.

2.5.1.2 Сервис для управления аутентификацией и токенами

В рамках мобильного приложения для корректного взаимодействия с сервером (API) реализован отдельный сервис – AuthService. Этот сервис отвечает за выполнение всех задач, связанных с аутентификацией и авторизацией пользователя, а также управлением токенами.

Основные функции AuthService:

Управление токенами (access и refresh). Сервис использует библиотеку flutter_secure_storage для безопасного хранения токенов на устройстве пользователя.

–Access–токен (access_token) необходим для авторизации запросов к API.

–Refresh–токен (refresh_token) используется для получения нового access–токена в случае его истечения.

Проверка валидности access–токена. Перед выполнением любого запроса AuthService проверяет, не истек ли срок действия access–токена (с помощью библиотеки jwt_decoder). Если срок действия токена истек, выполняется обновление (refresh) токена через эндпоинт сервера.

Обновление access–токена (refresh). Если access–токен просрочен, AuthService отправляет запрос на эндпоинт API /api/v1/token/refresh/, передавая refresh–токен. В случае успешного обновления новый access–токен сохраняется в хранилище.

Проверка статуса авторизации. С помощью метода isLoggedIn сервис может определить, авторизован ли пользователь в текущий момент. Это достигается за счет проверки наличия refresh–токена и его срока действия.

Запросы с авторизацией. AuthService предоставляет вспомогательные методы `authorizedGet` и `authorizedPost`, которые автоматически добавляют заголовок `Authorization: Bearer <access_token>` при отправке HTTP-запросов.

Выход пользователя из приложения. Метод `logout` удаляет все сохраненные токены, полностью разлогинивая пользователя.

Дополнительные функции:

- Метод `getUsername` позволяет получить имя пользователя, декодируя данные из `access`-токена.

- Метод `isStaff` проверяет наличие прав администратора (по полю `is_staff` в токене).

Принципы работы AuthService полностью соответствуют рекомендациям по реализации JWT-аутентификации, описанным в официальной документации JWT.io [10] и в разделе по авторизации API в документации Flutter [12].

AuthService выполняет ключевую роль в обеспечении безопасного и корректного взаимодействия мобильного приложения с API. Он реализует хранение, проверку и обновление токенов, а также предоставляет удобный интерфейс для авторизованных запросов к серверу.

2.5.1.3 Сервис для взаимодействия с API

Для обеспечения взаимодействия мобильного приложения с серверной частью в проекте реализован отдельный сервис – ApiService. Этот сервис служит «единым центром» для выполнения всех HTTP-запросов к API, отвечая за получение, отправку и обновление данных, а также за их преобразование в удобные для использования в приложении модели.

Основные задачи ApiService:

Получение и отправка данных. Сервис обеспечивает отправку авторизованных запросов (с использованием `access`-токена, полученного через AuthService) к различным эндпоинтам API. В зависимости от типа запроса (GET, POST, PUT) используются соответствующие методы `http`.

Работа с данными пользователей:

- Получение списка заявок пользователя (`getUserReports`).

- Отправка новой заявки, включая возможность загрузки изображения (`submitReport`).

- Обновление статуса заявки (например, перевод в статус «Решено»).

- Получение статистики по заявкам пользователя (количество заявок по статусам и категориям).

Работа с данными города и опросами:

- Получение списка районов (`getDistricts`).

- Получение статистики по жалобам для конкретного района (`getDistrictCategoryStats`).

–Получение и отправка ответов на вопросы опроса (loadQuestions, submitResponses).

–Проверка доступности опроса для пользователя.

Сбор и анализ статистики:

–Получение сводной статистики по категориям – корреляция субъективных и объективных данных – в рамках города (getCategoryCorrelationStats).

–Получение сводной статистики в конкретном районе (getDistrictCategoryCorrelationStats).

–Работа с моделями статистики: DistrictStats, CategoryCorrelationStat.

Работа с наградами и виртуальным магазином

–Проверка наличия вознаграждения за заявку (checkRewardExists).

–Получение списка доступных товаров (getProducts).

–Проверка баланса пользователя (getBalance).

–Совершение покупки в магазине (purchaseProduct).

ApiService является важной частью мобильного приложения, обеспечивая надежное и структурированное взаимодействие с сервером. Он упрощает работу с данными, сокращает дублирование кода и повышает надежность приложения при обмене данными с API.

2.5.1.4 Структура экранов приложения

Исходя из функциональных требований, в приложении были созданы соответствующие окна (экраны), каждый из которых отвечает за отдельный пользовательский сценарий: авторизация, просмотр и создание жалоб, аналитика, работа с вознаграждениями и др.

В зависимости от роли пользователя (обычный пользователь или представитель городской службы) доступный функционал различается. Для обычного пользователя помимо основных страниц предусмотрены дополнительные страницы для создания новых заявок и работы с магазином вознаграждений. На Рисунке 2.12 представлена диаграмма, демонстрирующая доступные пользователю экраны и навигацию между ними.

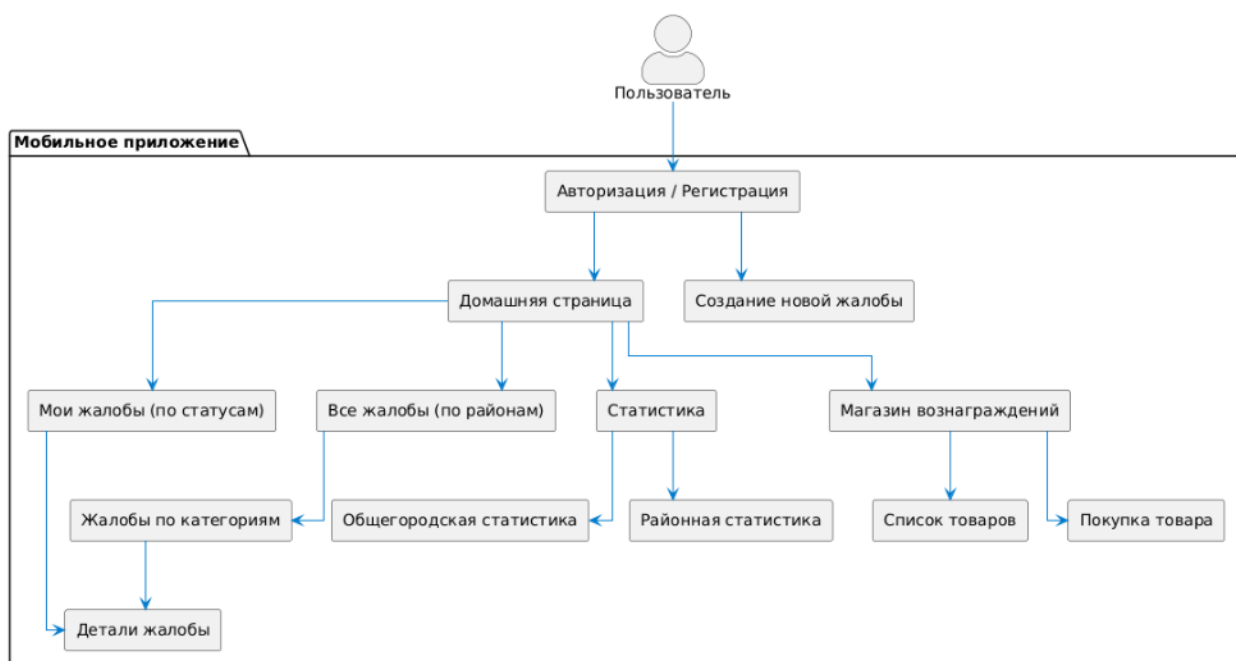


Рисунок 2.12 – список страниц, доступных пользователю, и навигация между ними

Представитель городской службы имеет доступ к специализированной странице для обработки поступивших жалоб, но не может создавать собственные заявки или использовать магазин вознаграждений. На Рисунке 2.13 представлена диаграмма, демонстрирующая экраны доступные представителю городских служб.

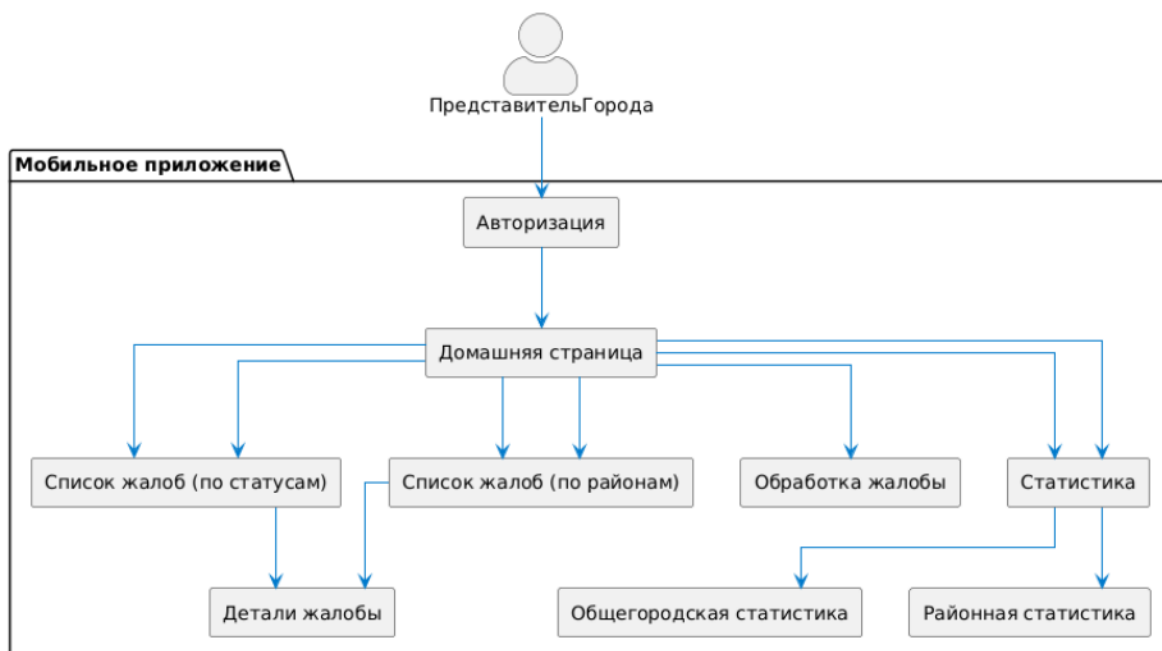


Рисунок 2.13 – список страниц, доступных представителям городских служб, и навигация между ними

Такая организация интерфейса позволяет адаптировать приложение под различные потребности пользователей и обеспечивает удобство работы с системой.

2.5.1.5 Разработка экранов приложения

В качестве примера реализации экранов в мобильном приложении рассмотрим страницу создания новой жалобы. Данный экран позволяет пользователю отправить жалобу, указав ключевую информацию: заголовок, описание, категорию, местоположение и при необходимости – фотографию.

На Рисунке 2.14 представлен скриншот окна создания новой жалобы пользователем.

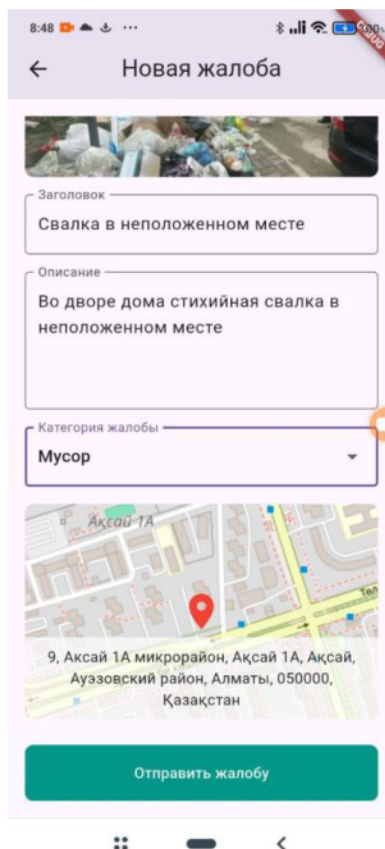


Рисунок 2.14 – окно создания новой жалобы

Основные элементы и функции страницы:

- Форма ввода с проверкой обязательных полей (заголовок, описание, категория).
- Выбор фотографии.

–Выбор точки на карте, с автоматическим получением адреса (с использованием внешнего API OpenStreetMap). На Рисунке 2.15 демонстрируется виджет для выбора местоположения жалобы на карте.

–Валидация формы – если пользователь не указал обязательные данные, отправка жалобы блокируется и отображается сообщение об ошибке.

–Отправка данных осуществляется через вызов метода submitReport() сервиса ApiService, который обрабатывает передачу данных на сервер (включая изображение, если оно выбрано).

–Обратная связь – при успешной отправке отображается сообщение с подтверждением, а при ошибке – подробное описание возникшей ошибки.

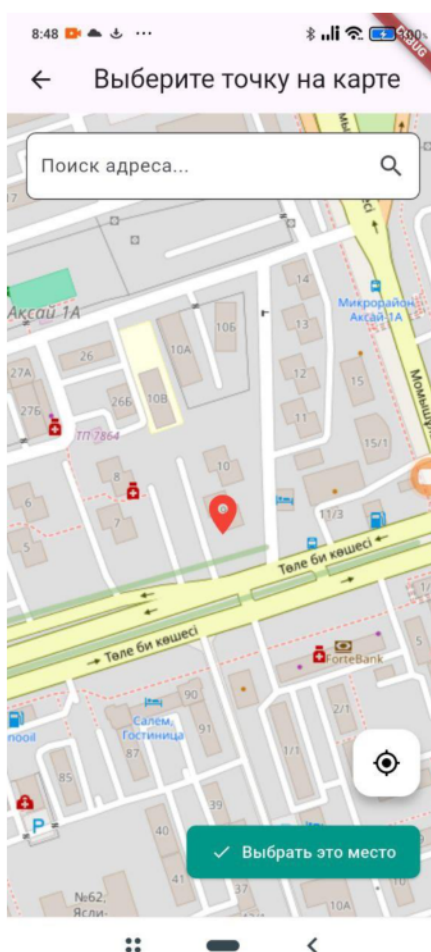


Рисунок 2.15 – виджет с выбором местоположения жалобы на карте

Во Flutter интерфейс приложения строится из виджетов – основных строительных блоков, которые описывают, как должен выглядеть интерфейс [13]. Виджеты во Flutter являются декларативными – они описывают, как должен выглядеть экран, а Flutter сам обновляет их при изменении состояния.

Виджеты бывают двух типов:

1) StatelessWidget – виджет, который не хранит внутреннее состояние. Пример: статичная кнопка или текст.

2) `StatefulWidget` – виджет, который может изменяться во времени (например, форма с вводом текста, анимация). Такие виджеты используют объект `State`, где хранится текущее состояние виджета.

Ключевой особенностью реализации этой страницы является использование `StatefulWidget` (см. Рисунок 2.16), что позволяет динамически обновлять интерфейс при выборе категории, загрузке изображения и указании местоположения.

```
class NewReportScreen extends StatefulWidget {
  const NewReportScreen({Key? key}) : super(key: key);

  @override
  State<NewReportScreen> createState() => _NewReportScreenState();
}

class _NewReportScreenState extends State<NewReportScreen> {
  final _formKey = GlobalKey<FormState>();
  final _titleController = TextEditingController();
  final _descController = TextEditingController();
}
```

Рисунок 2.16 – класс создания новой жалобы

Для упрощения взаимодействия с API и локальными данными в модуле используется сервисный слой (`ApiService`), который инкапсулирует логику отправки запроса, а в интерфейсе остается только вызов этого метода.

Такой подход демонстрирует общий принцип построения экранов в мобильном приложении – разделение логики и интерфейса, а также гибкость в обработке данных, что особенно важно для реализации надежного пользовательского опыта.

2.5.2 Тестирование приложения

После завершения разработки всех ключевых функций мобильного приложения было проведено ручное (мануальное) тестирование с целью выявления возможных ошибок и проверки корректной работы всех функций.

Тестирование проводилось в следующем порядке:

- 1) Были подготовлены тестовые сценарии.
- 2) Приложение запускалось на: эмуляторе Android (Pixel 4, API 33) и реальном устройстве Redmi 9C (Android 10).
- 3) Проверялись разные сценарии взаимодействия пользователя: ввод корректных и некорректных данных, загрузка изображений, отправка запросов, работа с картой.

В таблице 8 представлены результаты проведения тестирования.

Таблица 8 – Результаты тестирования

№	Название теста	Действия пользователя	Ожидаемый результат	Статус
1	Регистрация нового пользователя	Ввести корректные данные (имя, e-mail, пароль) и нажать «Зарегистрироваться».	Пользователь успешно зарегистрирован и переходит на «Домашнюю страницу».	пройдено
2	Регистрация с пустыми полями	Оставить обязательные поля пустыми и нажать «Зарегистрироваться».	Появляется сообщение об ошибке: «Заполните обязательные поля».	пройдено
3	Авторизация	Ввести корректные данные для входа и нажать «Войти».	Пользователь успешно авторизован.	пройдено
4	Авторизация с неверным паролем	Ввести правильный e-mail и неправильный пароль, нажать «Войти».	Выводится сообщение об ошибке: «Неверный логин или пароль».	пройдено
5	Отправка жалобы с фото	Заполнить форму создания жалобы, прикрепить фотографию, указать местоположение.	Жалоба успешно отправлена, появляется уведомление об успехе.	пройдено
6	Отправка жалобы без выбора местоположения	Заполнить форму, но не указать местоположение.	Выводится сообщение об ошибке: «Пожалуйста, выберите местоположение».	пройдено
7	Отправка жалобы без выбора категории	Заполнить форму, но не указать категорию.	Выводится сообщение об ошибке: «Пожалуйста, выберите категорию».	пройдено

8	Просмотр списка жалоб	Перейти на страницу «Список жалоб» после авторизации.	Отображается список всех жалоб пользователя.	пройдено
9	Просмотр деталей жалобы	Нажать на элемент списка жалоб.	Открывается страница с подробной информацией о жалобе.	пройдено
10	Обновление статуса жалобы (роль is_staff)	Представитель городской службы заходит в жалобу и меняет статус.	Статус жалобы успешно обновлен, отображается новый статус.	пройдено
11	Отображение аналитики	Перейти в раздел «Аналитика» с домашней страницы.	Отображаются графики общей и районной статистики.	пройдено
12	Покупка товара при достаточном балансе	Перейти в «Магазин вознаграждений», выбрать товар, нажать «Купить».	Покупка совершается успешно, баланс уменьшается.	пройдено
13	Покупка товара при недостаточном балансе	Попытаться купить товар, если баланса не хватает.	Выводится сообщение: «Недостаточно монет». Покупка не совершается.	пройдено
14	Выбор местоположения на карте	Перейти в режим выбора точки на карте, отметить точку.	Выбранная точка отображается на карте, адрес подгружается автоматически.	пройдено

В результате проведения тестирования были выявлены незначительные ошибки при валидации полей формы и при загрузке фотографий. Ошибки были оперативно устранены. Все функции приложения работают корректно. Интерфейс остается удобным и понятным даже при разных разрешениях устройств.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта было разработано мобильное приложение CityVoice, предназначенное для повышения качества городской среды посредством эффективного взаимодействия между жителями и городскими службами. Актуальность проекта обоснована необходимостью устранения информационного разрыва между гражданами и государственными структурами, который препятствует своевременному решению проблем городской инфраструктуры, таких как плохое состояние дорог, нехватка зеленых зон и проблемы с водоотведением.

На этапе исследования были изучены существующие отечественные и международные решения, выявлены их ограничения и сильные стороны. Также были проанализированы реальные проблемы городской среды Алматы, а на основе опроса 110 респондентов подтверждена высокая заинтересованность граждан в создании удобного, прозрачного и действенного инструмента мониторинга и подачи жалоб и предложений.

С технической стороны проект реализован в виде клиент–серверного приложения. Клиентская часть разработана с использованием фреймворка Flutter, обеспечивающего высокую производительность и гибкость интерфейса. Серверная часть построена на Django REST Framework с использованием PostgreSQL и расширения PostGIS для обработки пространственных данных. Архитектура backend–части организована по принципу модульности и включает в себя отдельные микроприложения для управления жалобами, опросами, вознаграждениями, пользователями и аналитикой.

В результате была создана система, которая позволяет пользователям:

- быстро фиксировать проблемы на карте с прикреплением фото;
- отслеживать статус обращений;
- участвовать в опросах и вносить вклад в оценку городской среды;
- получать баллы за активность и использовать их в системе вознаграждений.

Городские службы, в свою очередь, получают возможность оперативно реагировать на обращения граждан, отслеживать проблемы по районам, приоритизировать ресурсы и планировать системные меры на основе агрегированных данных.

Таким образом, CityVoice не только решает задачу цифровизации обратной связи между гражданами и администрацией, но и формирует основу для устойчивого и прозрачного управления городской средой. В перспективе проект может быть масштабирован на другие города, интегрирован с государственными порталами.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- 1 ПЛАН РАЗВИТИЯ ГОРОДА АЛМАТЫ НА 2021-2025 ГОДЫ // Электронная версия на сайте https://www.gov.kz/uploads/2023/1/12/11fe6e6a80b80f7a894a58d876523dfd_origin.al.1289333.pdf
- 2 TheGlobalEconomy. Roads quality in Kazakhstan // Электронный ресурс https://www.theglobaleconomy.com/Kazakhstan/roads_quality
- 3 IQAir. Индекс качества воздуха в Алматы, 2025 г. // Электронный ресурс <https://www.iqair.com/ru/world-most-polluted-cities>
- 4 World Health Organization 2021, WHO global air quality guidelines. Particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide. // Электронная версия на сайте <https://iris.who.int/bitstream/handle/10665/345329/9789240034228-eng.pdf>
- 5 Данные о количестве зеленых насаждений в Алматы на 2024 год от Управления развития общественных пространств города Алматы // Электронный ресурс <https://www.inform.kz/ru/parki-skveri-i-naberezhnie-kakie-obekti-blagoustroyat-v-2025-godu-v-almati-18bf21>
- 6 The Global Urban Monitoring Framework, 2022 // Электронный ресурс <https://unhabitat.org/the-global-urban-monitoring-framework>
- 7 ООН, цели в области устойчивого развития. Цель 11: Обеспечение открытости, безопасности, жизнестойкости и экологической устойчивости городов и населенных пунктов // Электронный ресурс <https://www.un.org/sustainabledevelopment/ru/cities/>
- 8 Статья на сайте gov.kz «Причины постоянных подтоплений в Алматы» // Электронный ресурс <https://www.gov.kz/memleket/entities/almaty/press/news/details/361403>
- 9 Нурланова Н.К., Альжанова Ф.Г., Сатпаева З.Т. Качество городского пространства как фактор устойчивого экономического развития г. Алматы // Электронная версия на сайте <https://esp.ieconom.kz/jour/article/view/262/265>
- 10 Introduction to JSON Web Tokens // Электронный ресурс <https://jwt.io/introduction>
- 11 Flutter documentation — JSON and serialization // Электронный ресурс <https://docs.flutter.dev/data-and-backend/serialization/json>
- 12 Flutter documentation – Make authenticated requests // Электронный ресурс <https://docs.flutter.dev/cookbook/networking/authenticated-requests>
- 13 Flutter - Stateless and stateful widgets // Электронный ресурс <https://docs.flutter.dev/development/ui/interactive>

Приложение А

(обязательное)

Техническое задание

А.1.1 Техническое задание на разработку мобильного приложения для мониторинга и обратной связи по качеству городской среды

Настоящее техническое задание распространяется на разработку мобильного приложения для мониторинга и обратной связи по качеству городской среды. Приложение позволит пользователям сообщать о проблемах в городской среде, проходит опросы по оценке качества городской среды, а ответственным органам – оперативно реагировать на обращения, анализировать статистику по наиболее частым проблемам и планировать ремонтные и модернизационные работы. Приложение предоставит эффективное решение для построения комфортной городской среды посредством простого и удобного инструмента, который станет посредником между горожанами и местными городскими властями и службами.

А.1.2 Основание для разработки

Приложение разрабатывается на основании согласия дипломного руководителя выбранной дипломником темой.

А.1.3 Назначение

Приложение предназначено для улучшения взаимодействия жителей с городскими службами путем упрощения процесса подачи жалоб на инфраструктурные проблемы и повышения прозрачности их обработки.

А.1.4 Требование к функциональным характеристикам

Приложение должно включать две роли: пользователь (житель города) и исполнитель (представитель ответственного органа). В зависимости от роли пользователя в приложении система должна обеспечивать определенные функции.

Должны быть обеспечены удобный ввод, хранение и обработка данных, а также предоставление пользователям информативных выходных данных.

А.1.4.1 Входные данные и способы их ввода

Пользователи вводят данные через формы в мобильном приложении. Данные включают текстовые поля, изображения и геолокационные метки.

Для пользователей:

- Регистрация и аутентификация: ввод имени пользователя, email, пароля.
- Отправка жалобы: ввод текста с описанием проблемы, загрузка фото, автоматическое определение или ручной ввод геолокации.
- Комментирование жалоб: ввод текстового комментария.
- Прохождение опроса: оценка метрики по 10-балльной шкале, выбор из списка, ввод текста.

Для исполнителей:

- Фильтрация жалоб: выбор района, категории, статуса.
- Изменение статуса жалобы: обновление статуса (на рассмотрении, в работе, выполнена), добавление комментария.
- Аналитика: отображение статистики в виде графиков в разрезе районов и временных промежутков.

А.1.4.2 Выходные данные

При нормальном функционировании:

- Подтверждение успешной регистрации и входа.
- Сообщение об успешной отправке жалобы.
- Отображение списка жалоб с фильтрацией и сортировкой.
- Вывод текущего статуса жалобы и комментариев исполнителей.
- Отображение статистики по жалобам.

Сообщения об ошибках:

- Некорректный формат данных (недопустимый email).
- Ошибка загрузки изображения (превышение допустимого размера).
- Ошибка сети при отправке жалобы или изменении статуса.

Приложение должно обеспечивать корректное взаимодействие между всеми ролями, минимизировать ошибки ввода данных и предоставлять пользователям подробную информацию о статусе их обращений.

А.1.5 Требования к составу и параметрам технических средств

Для обеспечения стабильной работы мобильного приложения необходимо учитывать аппаратные и программные требования, а также совместимость с различными устройствами.

Приложение должно поддерживать версию Android не ниже 7.0 (Nougat), так как более ранние версии имеют ограниченную поддержку актуальных технологий и библиотек. Для корректной работы приложения устройство пользователя должно соответствовать следующим характеристикам:

- ОЗУ: минимум 2 ГБ, рекомендуется 3 ГБ и выше для стабильной работы и быстрого отклика интерфейса.

- Свободное место в памяти: минимум 150 МБ для установки приложения, с учетом кэширования данных.

- Приложение должно корректно функционировать при следующих условиях сетевого соединения:

- Минимальная скорость интернет-соединения для базового использования — 1 Мбит/с.

- Оптимальная скорость для загрузки фото и взаимодействия с картами — 5 Мбит/с и выше.

- Поддержка сетей: Wi-Fi, 3G, 4G (LTE), 5G.

- Приложение будет активно взаимодействовать с сервером для обработки обращений граждан, поэтому:

- Сервер должен поддерживать REST API для эффективной передачи данных.

- Шифрование данных (HTTPS, TLS 1.2/1.3) для защиты пользовательских обращений.

- Минимальная задержка серверного отклика — до 1 секунды для комфортного взаимодействия.

А.1.6 Требования к надежности

Приложение должно производить валидацию вводимой пользователем информации, обеспечивать целостность и конфиденциальность хранимой информации. Должно быть предусмотрено разграничение функциональных возможностей приложения для пользователей с разными ролями.

А.1.7 Требования к безопасности

Для обеспечения конфиденциальности пользовательских данных приложение должно использовать защищенные механизмы аутентификации, для предотвращения несанкционированного доступа. Передача данных должна осуществляться по защищенным протоколам HTTPS с использованием TLS для обеспечения конфиденциальности и целостности информации.

Приложение Б

Результаты опроса

Опрос для исследования интереса к цифровому мониторингу городской среды в Алматы

110 ответов

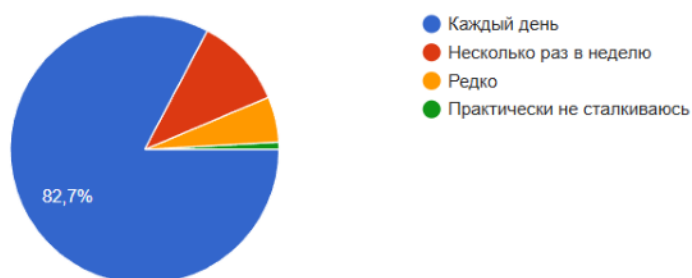
[Опубликовать статистику](#)

Блок 1. Ваш опыт и участие

1. Как часто вы сталкиваетесь с проблемами городской инфраструктуры (плохие дороги, загрязнение воздуха, нехватка зелёных зон и др.)?

[Копировать](#)

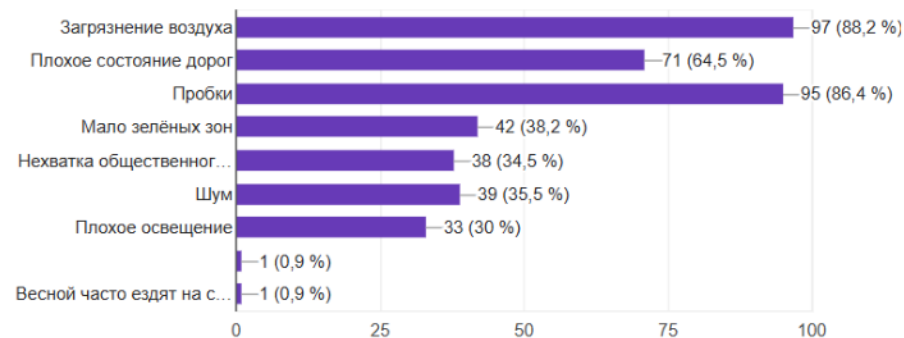
110 ответов



2. Какие проблемы городской среды вы считаете наиболее острыми в Алматы? (Можно выбрать несколько)

[Копировать](#)

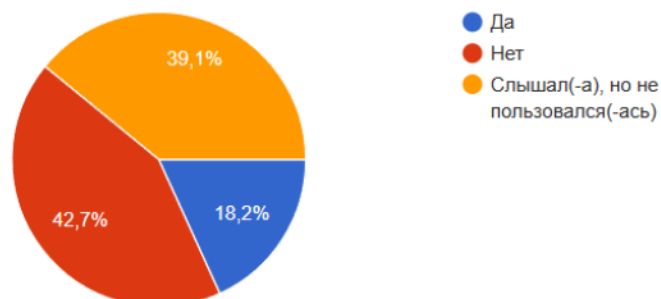
110 ответов



3. Приходилось ли вам когда-либо подавать жалобу или обращение в государственные сервисы (например, через Open Almaty или eOtinish)?

 Копировать

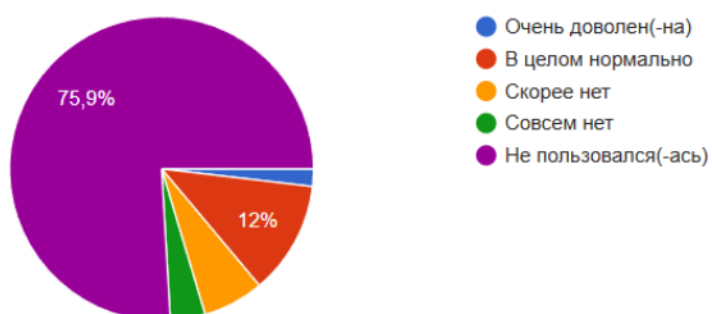
110 ответов



4. Если пробовали — насколько вы довольны результатом?

 Копировать

108 ответов



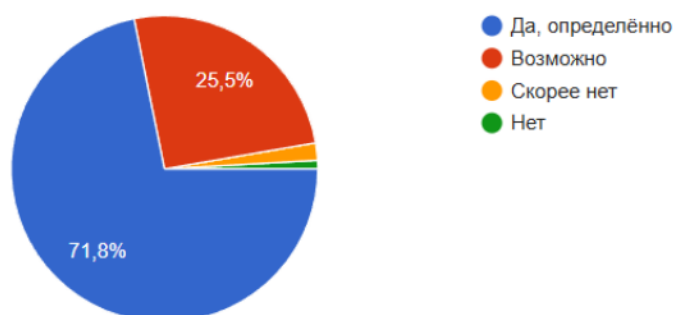


5. Если бы у вас было приложение, в котором вы могли бы:

- Оценивать чистоту улиц, уровень шума, дороги и транспорт по 10-балльной шкале,
- Оставлять жалобы и пожелания,
- Видеть рейтинг районов по качеству среды,
- Следить за реакцией властей и динамикой улучшений,
- Получать поощрения за вклад в улучшение города

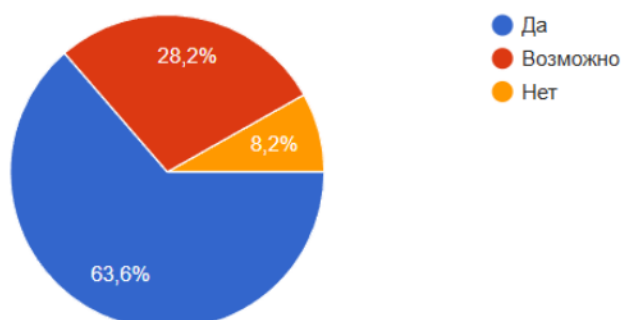
— пользовались бы вы им?

110 ответов



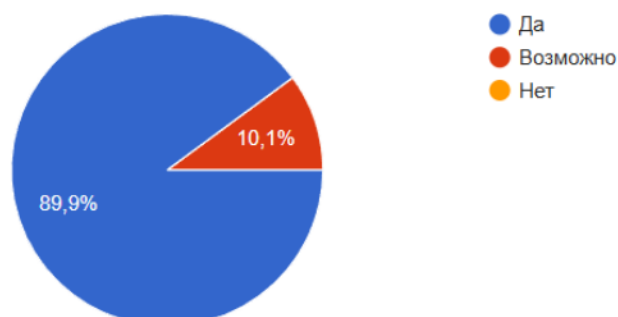
6. Интересно ли вам видеть на карте города жалобы и предложения от других жителей?

110 ответов



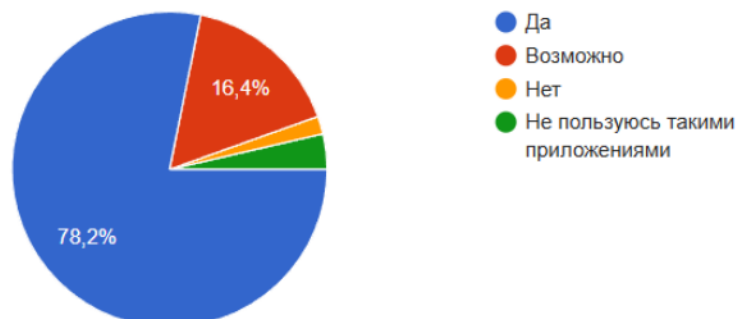
7. Полезно ли видеть статистику районов по качеству городской среды и количеству решенных проблем?

109 ответов



8. Если бы такое приложение было бесплатным, вы бы установили его на телефон?

110 ответов

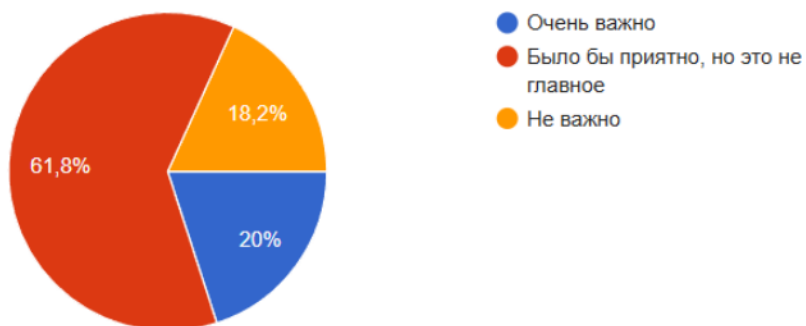


Блок 3. Мотивация и поощрение

 Копировать

9. Насколько важно для вас, чтобы за участие в улучшении города вы получали признание или поощрение?

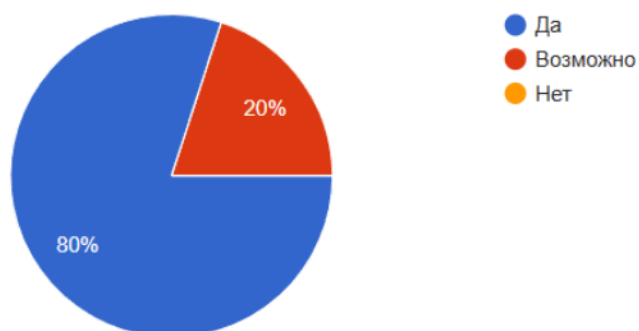
110 ответов



11. Если бы вы знали, что ваш отзыв или жалоба действительно учитывается и помогает сделать город лучше — стали бы вы активнее участвовать?

 Копировать

110 ответов

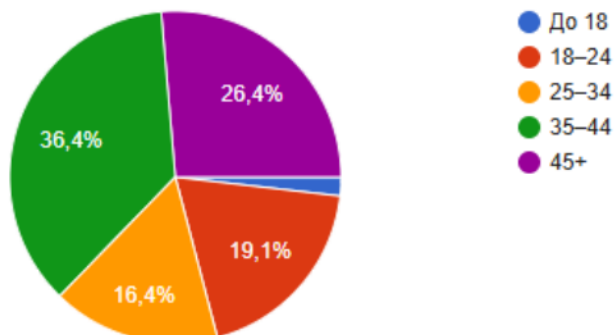


Блок 4. Немного о вас (анонимно)

 Копировать

12. Ваш возраст:

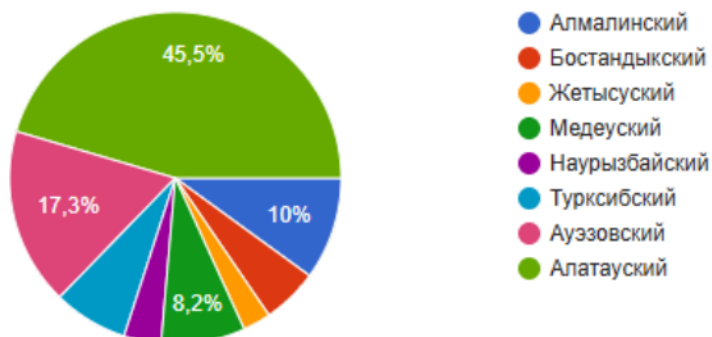
110 ответов



 Копировать

13. Ваш район проживания

110 ответов



Приложение В

Листинг (код) программы

Полный исходный текст программы доступен на Github.

API: <https://github.com/DKamil12/cityvoice-api>

Mobile: <https://github.com/DKamil12/cityvoice-mobile>

Ниже приведен код особенно важных модулей API:

1) Представления модуля Reports

```
# Представление для взаимодействия с районами
# HTTP действия: GET
class DistrictViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = District.objects.all()
    serializer_class = DistrictSerializer

# Представление для взаимодействия с категориями
# HTTP действия: GET, POST, PUT, DELETE
class CategoryViewSet(viewsets.ModelViewSet):
    serializer_class = CategorySerializer
    queryset = Category.objects.all()

# Представление для взаимодействия с жалобами
# HTTP действия: GET, POST, PUT, DELETE
class ReportViewSet(viewsets.ModelViewSet):
    queryset = Report.objects.all()
    filter_backends = [DjangoFilterBackend, filters.OrderingFilter]
    filterset_fields = ['category', 'district', 'status']
    ordering_fields = ['created_at']
# функция для сериализации модели в зависимости от HTTP-метода
def get_serializer_class(self):
    if self.action == 'create':
        return ReportOnCreateSerializer
    elif self.action == 'update':
        return ReportOnUpdateSerializer
    return ReportOnReadSerializer

# функция для фильтрации жалоб по дате
def get_queryset(self):
    queryset = super().get_queryset()
    start_date = self.request.query_params.get('start_date')
    end_date = self.request.query_params.get('end_date')
    if start_date:
```



```

        queryset = queryset.filter(created_at__gte=parse_date(start_date))
    if end_date:
        queryset = queryset.filter(created_at__lte=parse_date(end_date))
    return queryset

# функция, переопределяющая создание объекта, определяет район
жалобы, в зависимости от выбранных координат
def perform_create(self, serializer):
    lat = self.request.data.get('latitude')
    lon = self.request.data.get('longitude')
    point = Point(float(lon), float(lat))
    district = District.objects.filter(geometry__contains=point).first()
    serializer.save(user=self.request.user, district=district)

# функция, переопределяющая обновление объекта, позволяет обновлять
статус жалобы исполнителю
def update(self, request, *args, **kwargs):
    report = self.get_object()
    user = request.user
    status = request.data.get('status')
    comment_text = request.data.get('comment')

    if not user.is_staff:
        raise PermissionDenied('Только исполнители могут изменять статус
жалобы!')
    if not comment_text:
        raise ValidationError('Перед обновлением статуса оставьте
комментарий!')

    was_completed = report.status == 'resolved'
    report.status = status
    report.save()
    Comments.objects.create(user=user, report=report, text=comment_text)

    if status == 'resolved' and not was_completed:
        Reward.objects.get_or_create(
            user=report.user,
            report=report,
            defaults={'amount': 100}
        )
    return Response(ReportOnReadSerializer(report).data)

# Представление для взаимодействия с комментариями.

```

```

# HTTP действия: GET, POST, PUT, DELETE
class CommentViewSet(viewsets.ModelViewSet):
    queryset = Comments.objects.all()
    permission_classes = [IsAuthenticatedOrReadOnly]

    def get_serializer_class(self):
        if self.action in ['create', 'update']:
            return CommentSerializerOnWrite
        else:
            return CommentSerializerOnRead

    def perform_create(self, serializer):
        report = serializer.validated_data['report']
        user = self.request.user

        if user != report.user or not user.is_staff:
            raise PermissionDenied("Только исполнитель и владелец жалобы
могут оставлять комментарии!")
        serializer.save(user=user)

```

2) Представления модуля *Surveys*

```

# Представление для получения списка вопросов
# HTTP действия: GET
class SurveyQuestionListView(generics.ListAPIView):
    queryset = SurveyQuestion.objects.all()
    serializer_class = SurveyQuestionSerializer

# Представление для отправки ответа на вопрос
# HTTP действия: POST
class SurveyResponseCreateView(APIView):
    permission_classes = [IsAuthenticated]

    def post(self, request):
        question_id = request.data.get("question")
        rating = request.data.get("rating")
        district_id = request.data.get("district")

        if not all([question_id, rating, district_id]):
            raise ValidationError("Не все поля заполнены")

        if SurveyResponse.objects.filter(user=request.user,
question_id=question_id).exists():

```

```

        raise ValidationError("Вы уже проходили этот опрос")

    SurveyResponse.objects.create(
        user=request.user,
        question_id=question_id,
        rating=rating,
        district_id=district_id,
    )
    return Response({"success": True})

# Представление для проверки доступности опроса для пользователя
# HTTP действия: GET
class SurveyAvailabilityAPIView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        total_questions = SurveyQuestion.objects.count()
        user_answers =
SurveyResponse.objects.filter(user=request.user).values('question_id').distinct().count()

        available = user_answers < total_questions
        return Response({"available": available})

```

3) Представления модуля Charts

```

# Представление для получения статистики по количеству и характеру
жалоб в разрезе района
# HTTP действия: GET
class DistrictCategoryStatsView(APIView):
    def get(self, request, district_id, format=None):
        # поиск района
        try:
            district = District.objects.get(pk=district_id)
        except District.DoesNotExist:
            return Response({"detail": "District not found"}, status=404)

        # базовый queryset по району
        qs = Report.objects.filter(district=district)

        # фильтры по дате
        start = request.GET.get("start_date")
        end = request.GET.get("end_date")
        if start:
            qs = qs.filter(created_at__date__gte=parse_date(start))

```

```

if end:
    qs = qs.filter(created_at__date__lte=parse_date(end))

# общее число жалоб
total = qs.count()

# разбивка по категориям
raw = (
    qs
    .values("category__id", "category__name")
    .annotate(count=Count("id"))
    .order_by("-count")
)
categories = [
    {"id": item["category__id"], "name": item["category__name"], "count":
item["count"]}
    for item in raw
]

return Response({
    "district_id": district.id,
    "district_name": district.name,
    "total": total,
    "categories": categories
})

```

Представление для получения статистики по оценке показателей качества городской среды и количество жалоб по каждому показателю (категории) по всему городу

HTTP действия: GET

class CitywideCategoryCorrelationView(APIView):

def get(self, request):

Получаем все категории

categories = Category.objects.all()

result = []

for cat in categories:

Средняя оценка по вопросам этой категории

avg_rating = SurveyResponse.objects.filter(

question__category=cat

).aggregate(avg=Avg("rating"))["avg"]

Кол-во жалоб в этой категории

complaint_count = Report.objects.filter(

```

        category=cat
    ).count()

    result.append({
        "category_id": cat.id,
        "category_name": cat.name,
        "average_rating": round(avg_rating or 0, 2),
        "complaint_count": complaint_count,
    })
    return Response(result)

```

Ниже приведен код особенно важных модулей мобильного приложения:

1) Сервис для осуществления аутентификации через JWT токен

```

class AuthService {
    static const _baseUrl = 'https://cityvoice-api.onrender.com/api/v1';
    static const _accessKey = 'access_token';
    static const _refreshKey = 'refresh_token';
    final _storage = const FlutterSecureStorage();

    # асинхронный метод для хранения токенов доступа на стороне клиента
    Future<void> saveTokens(String access, String refresh) async {
        await _storage.write(key: _accessKey, value: access);
        await _storage.write(key: _refreshKey, value: refresh);
    }

    # класс для получения access-токена
    Future<String?> getValidAccessToken() async {
        String? token = await _storage.read(key: _accessKey);
        if (token == null || JwtDecoder.isExpired(token)) {
            await _refreshAccessToken();
            token = await _storage.read(key: _accessKey);
        }
        return token;
    }

    # класс для обновления access-токена
    Future<void> _refreshAccessToken() async {
        final refresh = await _storage.read(key: _refreshKey);
        if (refresh == null || JwtDecoder.isExpired(refresh)) {
            throw Exception('Refresh token expired');
        }

        final response = await http.post(

```

```

        Uri.parse('${_baseUrl}/token/refresh/'),
        headers: {'Content-Type': 'application/json'},
        body: jsonEncode({'refresh': refresh}),
    );

    if (response.statusCode == 200) {
        final data = jsonDecode(response.body);
        await _storage.write(key: _accessKey, value: data['access']);
    } else {
        throw Exception('Failed to refresh access token');
    }
}

# проверка авторизации пользователя
Future<bool> isLoggedIn() async {
    final refresh = await _storage.read(key: _refreshKey);
    return refresh != null && !JwtDecoder.isExpired(refresh);
}

# метод для выхода из приложения
Future<void> logout() async {
    await _storage.deleteAll();
}

# метод для авторизованного доступа к ресурсам API
Future<http.Response> authorizedGet(String endpoint) async {
    final token = await getValidAccessToken();
    return http.get(
        Uri.parse('${_baseUrl}/$endpoint'),
        headers: {'Authorization': 'Bearer $token'},
    );
}

# метод для получения данных о пользователе из access-токена
Future<String?> getUsername() async {
    final token = await getValidAccessToken();
    if (token != null) {
        final decoded = JwtDecoder.decode(token);
        return "${decoded['first_name']} ${decoded['last_name']}";
    }
    return null;
}

# метод для проверки роли пользователя

```

```

Future<bool> isStaff() async {
  final token = await getValidAccessToken();
  if (token == null) return false;
  final decoded = JwtDecoder.decode(token);
  return decoded['is_staff'] == true;
}
}

```

2) Сервис для взаимодействия с API – центральный класс, инкапсулирующий всю логику отправки запросов на сервер.

```

class ApiService {
  final String _baseUrl = 'https://cityvoice-api.onrender.com/api/v1';
  final String _host = 'cityvoice-api.onrender.com';
  final AuthService _auth = AuthService();

```

отправка новой жалобы

```

Future<Result<void>> submitReport({
  required String title,
  required String description,
  required String categoryId,
  required double latitude,
  required double longitude,
  File? image,
}) async {
  final url = Uri.parse('$_baseUrl/reports/reports/');
  final token = await _auth.getValidAccessToken();

  final request = http.MultipartRequest('POST', url);

  request.headers['Authorization'] = 'Bearer $token';
  request.fields['name'] = title;
  request.fields['description'] = description;
  request.fields['category'] = categoryId;
  request.fields['latitude'] = latitude.toString();
  request.fields['longitude'] = longitude.toString();

  if (image != null) {
    request.files.add(
      await http.MultipartFile.fromPath(
        'image',
        image.path,
        contentType: MediaType('image', 'jpeg'),
      ),
    ),
  }
}

```



```

    );
}

try {
    final streamedResponse = await request.send();
    final response = await http.Response.fromStream(streamedResponse);

    if (response.statusCode == 201) {
        return Result.success(null);
    } else {
        return Result.failure(
            'Ошибка: ${response.statusCode}, ${response.body}',
        );
    }
} catch (e) {
    return Result.failure('Сетевая ошибка: $e');
}
}

```

получение статистических данных по оценке показателей качества городской среды в разрезе района и корреляции между оценкой и количеством жалоб по показателю

```

Future<List<CategoryCorrelationStat>> getDistrictCategoryCorrelationStats(
    int districtId,
) async {
    final token = await _auth.getValidAccessToken();
    final uri = Uri.parse(
        '$_baseUrl/charts/statistics/survey/statistics/by-district/$districtId/',
    );

    final response = await http.get(
        uri,
        headers: {
            'Authorization': 'Bearer $token',
            'Content-Type': 'application/json',
        },
    );

    if (response.statusCode == 200) {
        final List<dynamic> data = jsonDecode(utf8.decode(response.bodyBytes));
        return data.map((e) => CategoryCorrelationStat.fromJson(e)).toList();
    } else {
        throw Exception('Не удалось загрузить данные по району');
    }
}

```