

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Клиентское приложение для информационной
системы контроля и анализа потребления
электроэнергии**

Студент

(подпись, дата)

Пресняков Артём Дмитриевич
(ФИО полностью)

Руководитель

(подпись, дата)

Гурьянов Л. В.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Пресняков Артём Дмитриевич гр. 22ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Клиентское приложение для информационной системы контроля и анализа потребления электроэнергии

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы Гурьянов Лев Вячеславович

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Разработать программные компоненты клиентской части информационной системы контроля и анализа учета электроэнергии

2) Среда выполнения приложения: семейство ОС UNIX

Языковые средства разработки: TypeScript

Инструментальные средства разработки: Visual Studio Code

Технологии разработки: React, Capacitor, Redux Toolkit, Storybook, Docker

3) Основные функции программного обеспечения:

а) управление бригадами и задачами;

б) ведение абонентов и объектов;

в) фиксация результатов проверок;

г) проверка фотоматериалов;

д) формирование отчетов.

5. Перечень подлежащих разработке вопросов

- 1) Анализ предметной области и требований к системе
- 2) Проектирование программных средств
- 3) Реализация программных средств
- 4) Тестирование программных средств
- 5) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 116 листов, 53 рисунка, 22 таблицы, 20 использованных источников, 2 приложения.

КЛИЕНТСКОЕ ПРИЛОЖЕНИЕ, ИНФОРМАЦИОННАЯ СИСТЕМА, REACT, NODE.JS, TYPESCRIPT, JAVASCRIPT, CAPACTIOR, PWA, RTK QUERY, UML

Объектом разработки являются функциональные компоненты клиентской части информационной системы контроля и анализа потребления электроэнергии.

Предметом разработки является подсистема пользовательского взаимодействия, обеспечивающая получение задач, сопровождение проверок, фотофиксацию, оформление актов, просмотр отчетов и управление задачами, бригадами, абонентами, проверками и пользователями.

Цель работы – разработка клиентского приложения информационной системы контроля и анализа потребления электроэнергии.

Средства разработки: язык программирования TypeScript, среды разработки Visual Studio Code.

Среда исполнения – семейство ОС UNIX.

Результаты работы: проведен обзор и анализ существующих информационных систем контроля и анализа потребления электроэнергии, определены требования к клиентскому приложению, выполнено проектирование мобильного АРМ инспектора и панели администратора для диспетчеров, разработаны и протестированы интерфейсы получения задач, оформления актов, фотофиксации, просмотра отчетов и управления задачами, бригадами, абонентами, проверками и пользователями

					ПГУ 09.03.04 – 08ВР221.13 ПЗ						
Изм.	Лист	№ докум.	Подп.	Дата	Клиентское приложение информационной системы контроля и анализа потребления электроэнергии. Пояснительная записка.	Лит.		Лист		Листов	
Разраб.		Пресняков А.Д.						4		116	
Пров.		Гурьянов Л. В.									
Реценз.											
Н. контр.		Такташкин Г.В.									
УтверГ.		Козлов А.Ю.									
						Группа 22ВП1					

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку клиентского приложения.....	9
1.1 Анализ предметной области	9
1.1.1 Понятие контроля и анализа потребления электроэнергии.....	9
1.1.2 Участники предметной области.....	10
1.1.3 Организация информационных потоков.....	13
1.1.4 Документооборот предметной области	14
1.1.5 Проблемы текущего бизнес-процесса.....	16
1.2 Анализ информационных систем аналогичного назначения	17
1.3 Постановка задачи на разработку клиентского приложения	22
1.3.1 Общая концепция клиентского приложения	22
1.3.2 Функциональные требования.....	24
1.3.3 Нефункциональные требования.....	29
2 Проектирование клиентского приложения для информационной системы контроля и анализа потребления электроэнергии	31
2.1 Выбор общей архитектуры клиентской части	31
2.2 Проектирование маршрутов и навигации	35
2.3 Проектирование доменной структуры и данных.....	36
2.4 Проектирование сценария работы инспектора с актом	38
2.5 Проектирование интеграции с серверной информационной системой ..	40
2.6 Проектирование пользовательского интерфейса	40
2.6.1 Проектирование пользовательского интерфейса АРМ диспетчера...	40
2.6.2 Проектирование пользовательского интерфейса АРМ инспектора...	42
2.7 Требования к программному и аппаратному окружению	45
2.8 Обоснование технологических решений и вариантов поставки.....	46
3 Разработка программного обеспечения клиентского приложения.....	48
3.1 Организация разработки и структура клиентской части	48

3.2	Реализация мобильного АРМ инспектора.....	49
3.3	Реализация формы акта, фотофиксации и результатов работы	53
3.4	Реализация административной панели диспетчера	58
3.5	Автоматическая генерация API-клиента и обмен данными.....	70
3.6	Сборка PWA и Android-приложения через Capacitor.....	71
4	Контроль качества программного обеспечения.....	75
4.1	Обоснование подхода к тестированию	75
4.2	Функциональное тестирование пользовательских сценариев	77
4.3	Проверка интерфейсов административной панели в Storybook	86
4.4	Ручная проверка интерфейса	90
4.5	Оценка сложности программного кода по метрикам Холстеда.....	91
4.6	Функционально-стоимостной анализ	93
	Заключение	97
	Список использованных источников	99
	Приложение А (справочное) Глоссарий	102
	Приложение Б (обязательное) Листинг приложения	104

Введение

Энергосбытовые организации работают с большим объемом данных о потребителях, договорах, объектах энергопотребления, приборах учета, пломбах, проверках и результатах выездной работы. От оперативности обработки этих данных зависит качество контроля потребления электроэнергии, своевременность выявления нарушений и полнота документального подтверждения действий инспекторов [1].

Контроль потребления электроэнергии связан не только с расчетами и ведением абонентской базы. На практике он включает планирование выездов, назначение задач бригадам, проверку приборов учета на объекте, фотофиксацию, оформление актов ограничения, возобновления, проверки введенного ограничения и фиксации самовольного подключения. Если эти операции выполняются с опорой на бумажные документы и разрозненные каналы связи, между фактическим выполнением проверки и появлением результата в информационной системе возникает задержка [2]. Особенно заметна эта проблема в работе выездных бригад. Инспектору необходимо получить актуальный список задач, быстро найти объект на карте, открыть сведения об абоненте, внести результаты проверки и прикрепить фотографии прибора учета и пломб. Диспетчеру, в свою очередь, нужно видеть состояние задач, управлять составом бригад, назначать работы и формировать отчетность по результатам дня. Отсутствие единого клиентского приложения приводит к дублированию ввода, ручной передаче статусов и повышенному риску ошибок в актах.

Актуальность выпускной квалификационной работы определяется необходимостью разработки клиентского приложения, которое связывает мобильную работу инспектора на объекте и офисную работу диспетчера с серверной информационной системой контроля и анализа потребления электроэнергии. Такое приложение должно обеспечивать единый пользовательский контур для задач, проверок, актов, фотографий, бригад и

отчетов.

Целью выпускной квалификационной работы является разработка клиентского приложения для информационной системы контроля и анализа потребления электроэнергии, обеспечивающего работу выездных бригад инспекторов и диспетчеров при планировании, выполнении и контроле задач проверки объектов энергопотребления.

Для достижения поставленной цели необходимо решить следующие задачи: проанализировать предметную область контроля и анализа потребления электроэнергии; определить участников процесса, документы и информационные потоки; выявить проблемы текущего бизнес-процесса; провести анализ аналогичных программных решений; сформулировать функциональные и нефункциональные требования; спроектировать архитектуру клиентского приложения; определить структуру мобильного АРМ инспектора и административной панели диспетчера; спроектировать сценарии работы с задачами, актами, фотографиями и отчетами.

Объектом исследования является процесс контроля и анализа потребления электроэнергии в части планирования и выполнения выездных проверок. Предметом исследования является клиентское программное обеспечение, обеспечивающее взаимодействие инспекторов и диспетчеров с серверной информационной системой.

Особое внимание уделяется пользовательским интерфейсам, механизмам обмена данными с серверным API, разграничению функциональности по ролям пользователей и обеспечению удобной работы как в стационарных условиях, так и при выполнении выездных задач.

Разрабатываемое клиентское приложение рассматривается как монорепозитарий клиентских модулей. В его состав входят мобильный веб-клиент инспектора с возможностью Android-сборки и административная панель диспетчера. Клиентская часть взаимодействует с серверным API, использует общие типы и сгенерированный API-клиент-клиент, а пользовательские сценарии разделены по ролям и устройствам.

1 Анализ предметной области и постановка задачи на разработку клиентского приложения

1.1 Анализ предметной области

1.1.1 Понятие контроля и анализа потребления электроэнергии

Контроль потребления электроэнергии представляет собой совокупность организационных и технических мероприятий, направленных на проверку фактического состояния объектов энергопотребления, приборов учета, средств визуального контроля и документов, связанных с предоставлением коммунальной услуги по электроснабжению. В рамках рассматриваемой системы контроль выполняется не в одном офисном рабочем месте, а в распределенном процессе: часть действий выполняется диспетчером, часть – выездной бригадой на объекте, часть – специалистом по работе с абонентами.

На объекте инспектор сверяет сведения об адресе и абоненте с реестром, проверяет наличие доступа к прибору учета, оценивает состояние пломб, фиксирует показания, выявляет признаки нарушения режима потребления и при необходимости оформляет акт. Важным результатом является не только текстовое описание проверки, но и набор доказательных материалов: фотографии прибора учета, пломб, места установки, а также сведения о времени и статусе выполнения задачи.

Анализ потребления электроэнергии опирается на накопленные результаты таких проверок. Для диспетчера и руководителей отдела важны сведения о количестве назначенных, выполняемых и завершенных задач, выявленных нарушениях, качестве фотофиксации, полноте сформированных актов и равномерности загрузки бригад. Поэтому клиентское приложение должно не только отображать данные, но и обеспечивать корректный сбор первичной информации на объекте.

Особенность предметной области состоит в том, что ошибка в первичных данных приводит к последствиям сразу в нескольких контурах. Неверно указанный адрес затрудняет поиск объекта, потерянная фотография снижает

доказательную силу акта, несвоевременное обновление статуса мешает диспетчеру перераспределять нагрузку, а ручной ввод сведений из бумажных актов увеличивает вероятность расхождений между фактической проверкой и данными информационной системы.

1.1.2 Участники предметной области

Предметная область системы связана с деятельностью отдела контроля и анализа потребления электроэнергии. Его работа включает подготовку исходных данных, формирование бригад, назначение задач, выполнение проверок, оформление результатов и последующий анализ. Структурная организация предметной области показана на рисунке 1.

Для клиентского приложения такое разделение ролей важно потому, что каждый участник работает с разным набором данных и принимает решения на своем этапе процесса. Диспетчер отвечает за планирование и контроль, инспектор фиксирует фактическое состояние объекта, а специалист по абонентам обеспечивает достоверность исходной информации. Поэтому интерфейсы должны не дублировать друг друга, а поддерживать общий поток данных между участниками проверки.

Такое распределение ответственности также определяет требования к составу экранов. Инспектору необходимо быстро получить задачу и зафиксировать результат на объекте, диспетчеру – видеть состояние работ и управлять назначениями, а общие данные должны оставаться согласованными независимо от того, через какой клиентский модуль они были изменены.

Клиентское приложение поэтому должно показывать пользователю уже известные сведения и оставлять ручной ввод только для результата проверки. Такой подход снижает вероятность ошибок при заполнении данных, ускоряет работу пользователя и обеспечивает единообразие информации между мобильным клиентом инспектора и административной панелью диспетчера. Кроме того, сокращение объема ручного ввода уменьшает влияние человеческого фактора и позволяет снизить количество допускаемых ошибок.

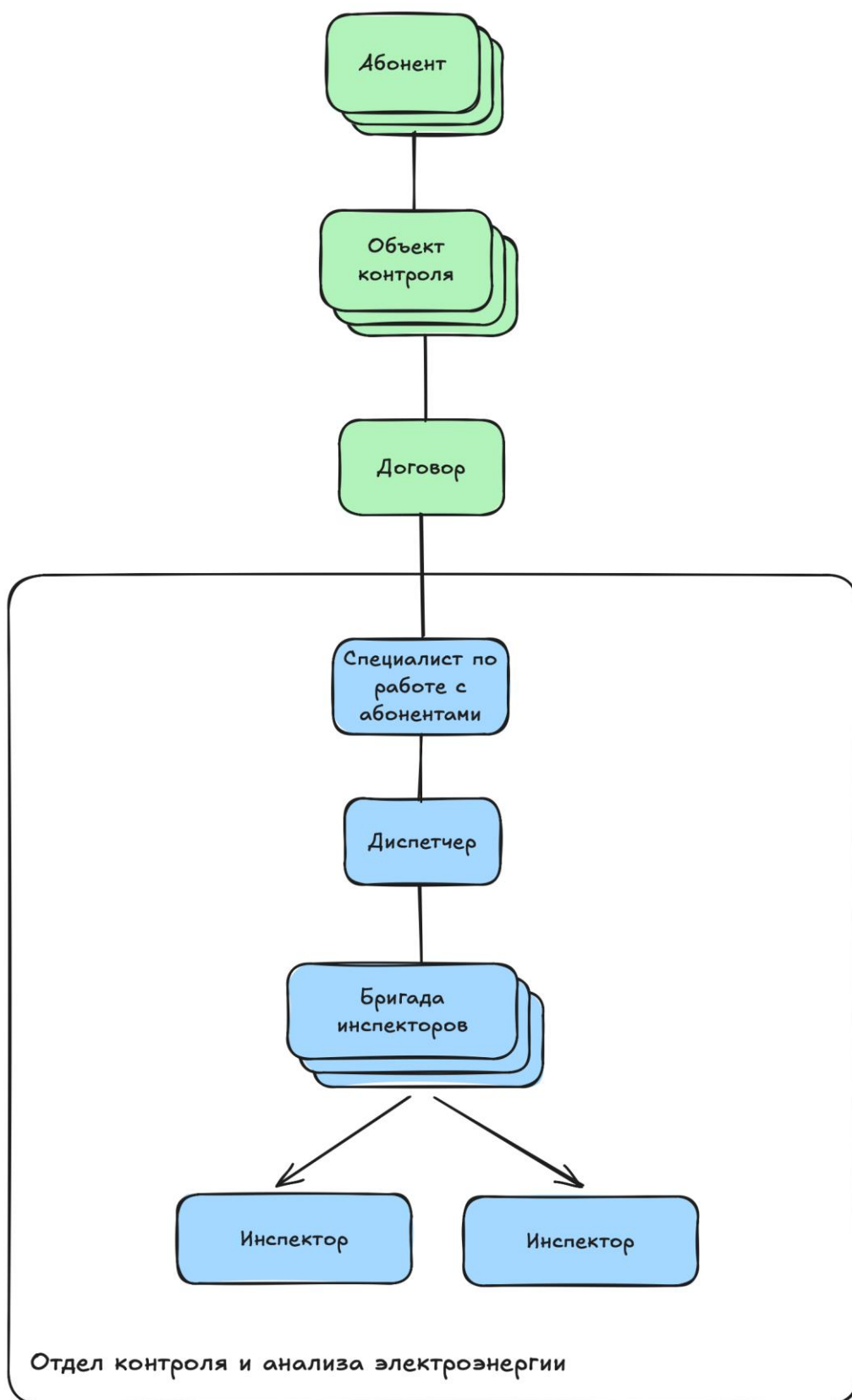


Рисунок 1 – Структурная организация предметной области

На рисунке 1 показано, что проверка объекта является результатом взаимодействия нескольких ролей. Специалист по работе с абонентами отвечает

за реестр и исходные сведения, диспетчер превращает эти сведения в задачи и распределяет их между бригадами, а инспекторы возвращают в систему результат фактической проверки.

Основные участники и поддерживаемые ими функции представлены в таблице 1.

Таблица 1 – Участники предметной области и их функции

Категория участника	Роль в предметной области	Функции, поддерживаемые клиентским приложением
Инспектор	Выполняет проверку объекта энергопотребления на месте	Просмотр задач, работа с карточкой объекта, фотофиксация, заполнение формы акта, отправка результата проверки
Бригада инспекторов	Совместно выполняет назначенные диспетчером задачи	Получение общего списка задач, перевод задачи в работу, фиксация выполнения
Диспетчер	Планирует и контролирует работу бригад	Формирование бригад, назначение задач, контроль статусов, просмотр результатов, формирование отчетов
Специалист по работе с абонентами	Ведет сведения об абонентах, договорах, объектах и приборах учета	Предоставление исходных данных для задач и карточек объектов
Абонент	Потребитель электроэнергии, объект которого проверяется	Его сведения используются при отображении задачи и формировании акта
Серверная информационная система	Хранит предметные данные и принимает результаты клиентских операций	API для задач, бригад, проверок, фотографий, отчетов и справочных данных

Из таблицы 1 следует, что клиентское приложение должно поддерживать не один пользовательский интерфейс, а два взаимосвязанных рабочих места: мобильное рабочее место инспектора и административную панель диспетчера. При этом данные специалиста по абонентам также должны использоваться в карточке задачи, даже если самостоятельный клиентский модуль специалиста выносится за рамки текущей версии.

1.1.3 Организация информационных потоков

Информационные потоки строятся вокруг задачи на проверку. Сначала в серверной информационной системе формируются сведения об абоненте, объекте, договоре, приборе учета и пломбах. Эти сведения поступают из реестра, который ведется специалистом по работе с абонентами. На основании данных реестра диспетчер создает задачу, выбирает бригаду и назначает плановое время выполнения.

После назначения бригада видит задачу в мобильном приложении. Инспектор открывает список работ, просматривает карточку объекта, определяет расположение адреса на карте и при необходимости строит маршрут. При начале выполнения задача переводится в состояние «в работе», что позволяет диспетчеру видеть фактический статус и не ждать устного подтверждения от бригады.

На объекте инспектор заполняет многошаговую форму акта. В зависимости от ситуации выбирается тип акта: ограничение, возобновление, проверка введенного ограничения или фиксация самовольного подключения. Фотографии прибора учета и пломб прикрепляются к проверке и передаются в серверную часть вместе с результатами формы.

После завершения проверки серверная система сохраняет акт, вложения и итоговый статус задачи. Диспетчер использует эти сведения в административной панели для контроля выполнения, анализа работы бригад и подготовки отчетности. Контекстная диаграмма информационной системы приведена на рисунке 2.

В этой схеме особенно важна своевременность передачи статусов. Пока задача находится только в бумажном акте или в устном сообщении бригады, диспетчер не может корректно оценить загрузку исполнителей и завершенность работ. Клиентское приложение сокращает этот разрыв: результат проверки фиксируется в момент выполнения и сразу становится частью общей информационной модели.

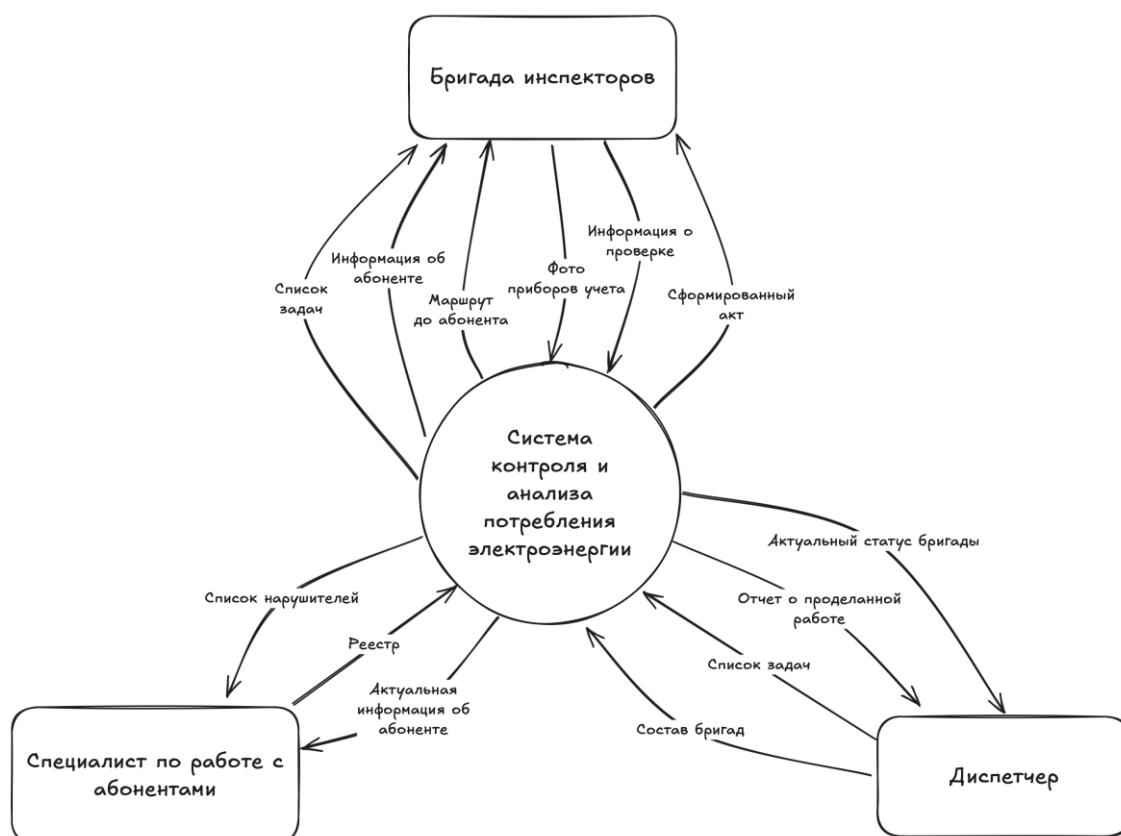


Рисунок 2 – Контекстная диаграмма информационной системы

На рисунке 2 показано, что клиентское приложение является промежуточным звеном между пользователями и серверной информационной системой. Поэтому в проектировании важно отделять пользовательские сценарии от низкоуровневых API-запросов и хранить бизнес-логику отображения в доменном слое клиента.

1.1.4 Документооборот предметной области

Документооборот отдела контроля складывается из документов, подтверждающих факт проверки и ее результат. К основным документам относятся акт о введении ограничения коммунальной услуги по электроснабжению, акт о возобновлении предоставления коммунальной услуги, акт проверки введенного ограничения режима потребления и акт о самовольном подключении.

Дополнительно в процессе используются фотографии приборов учета и пломб, реестр абонентов, объектов и приборов учета, а также отчет о

проделанной работе. Для клиентского приложения важно сохранять связь между задачей, результатами проверки, фотографиями и итоговым актом, иначе диспетчер не сможет быстро проверить полноту материалов.

Пример фотографии прибора учета, используемой как доказательный материал проверки, представлен на рисунке 3.



Рисунок 3 – Пример фотографии прибора учета

Фотография на рисунке 3 важна не как отдельный файл, а как часть записи о проверке. Она должна быть привязана к задаче, типу вложения, прибору учета или пломбе, а также быть доступной диспетчеру при проверке результата. Поэтому клиентский модуль загрузки файлов должен быть встроен в сценарий акта, а не существовать как отдельная галерея.

Реестр и отчетность являются вторым важным контуром документооборота. Реестр содержит исходные сведения для назначения задач, а отчет фиксирует выполненную работу за период. В клиентском приложении эти документы отражаются через карточки абонентов, задачи, проверки и отчеты.

1.1.5 Проблемы текущего бизнес-процесса

Анализ текущего бизнес-процесса показывает, что основные сложности возникают из-за разрыва между местом выполнения проверки и системой, где эти данные должны быть обработаны. Пока инспектор работает на объекте, диспетчер может не видеть фактического статуса, а фотографии и акт могут попадать в информационную систему позже. Основные проблемы и способы их решения приведены в таблице 2.

Таблица 2 – Проблемы текущего бизнес-процесса и способы их решения в клиентском приложении

Проблема текущего процесса	Решение в мобильном АРМ инспектора	Решение в админ-панели диспетчера
Задачи передаются и уточняются вне единого интерфейса	Инспектор получает назначенные задачи в приложении	Диспетчер назначает задачи бригадам и видит распределение
Статус выполнения становится известен с задержкой	Бригада переводит задачу в работу и завершает ее через приложение	Статусы отображаются в панели управления
Акты заполняются вручную и могут содержать ошибки	Форма акта разбита на последовательные шаги с обязательными полями	Диспетчер получает структурированный результат проверки
Фотографии хранятся отдельно от проверки	Фото прибора учета и пломбы прикрепляются при заполнении акта	Фотографии доступны в карточке проверки
Отчетность формируется вручную	Результаты проверки сразу отправляются в систему	Отчеты формируются по данным задач и проверок
Нет единого контроля факта выезда	Используются карта, геолокация и время изменения статуса	Диспетчер видит ход выполнения и может анализировать отклонения

Решение перечисленных проблем требует разработки клиентского приложения, ориентированного на реальный порядок работы инспектора и диспетчера, а не только на хранение справочных данных.

1.2 Анализ информационных систем аналогичного назначения

Для оценки актуальности разработки были рассмотрены решения, связанные с учетом электроэнергии, управлением проверками и организацией полевых работ. Сравнение выполнялось по критериям, которые важны именно для клиентского приложения: мобильная работа с задачами, карта, фотофиксация, акты, административное управление бригадами, отчетность, интеграция с серверной системой и ориентация на энергосбытовую предметную область.

Первым аналогом является сервис СТЕК-Контролер. Официальное описание указывает, что сервис состоит из программы «Работа контролера» и мобильного Android-приложения «Контролер» для выполнения обходов с применением мобильных устройств [3]. В составе решения заявлены реестр назначенных работ, карточка абонента, фотофиксация показаний и пломб, GPS-контроль, фиксация недопуска, обмен по API, онлайн- и оффлайн-режимы, а также защита от потери данных при неподтвержденной отправке [3]. Скриншот страницы аналога представлен на рисунке 4.

При сравнении с разрабатываемым приложением данный аналог полезен как пример специализированного мобильного рабочего места. Однако для рассматриваемой системы требуется не только обходной мобильный сценарий, но и единый административный контур диспетчера, связанный с задачами, проверками, бригадами, абонентами и отчетами. Диспетчер должен иметь возможность не только просматривать факт выполнения работ, но и управлять их распределением, отслеживать изменение статусов, контролировать загрузку бригад и получать данные для последующего формирования отчетов.

При оценке аналога учитывалось, насколько полевой результат может быть связан с диспетчерским контролем и общей моделью задач, а также позволяет ли решение проследить весь путь выполнения работы от назначения задачи до фиксации результата. Дополнительно рассматривалась согласованность различных участников действий во время выполнения процесса.

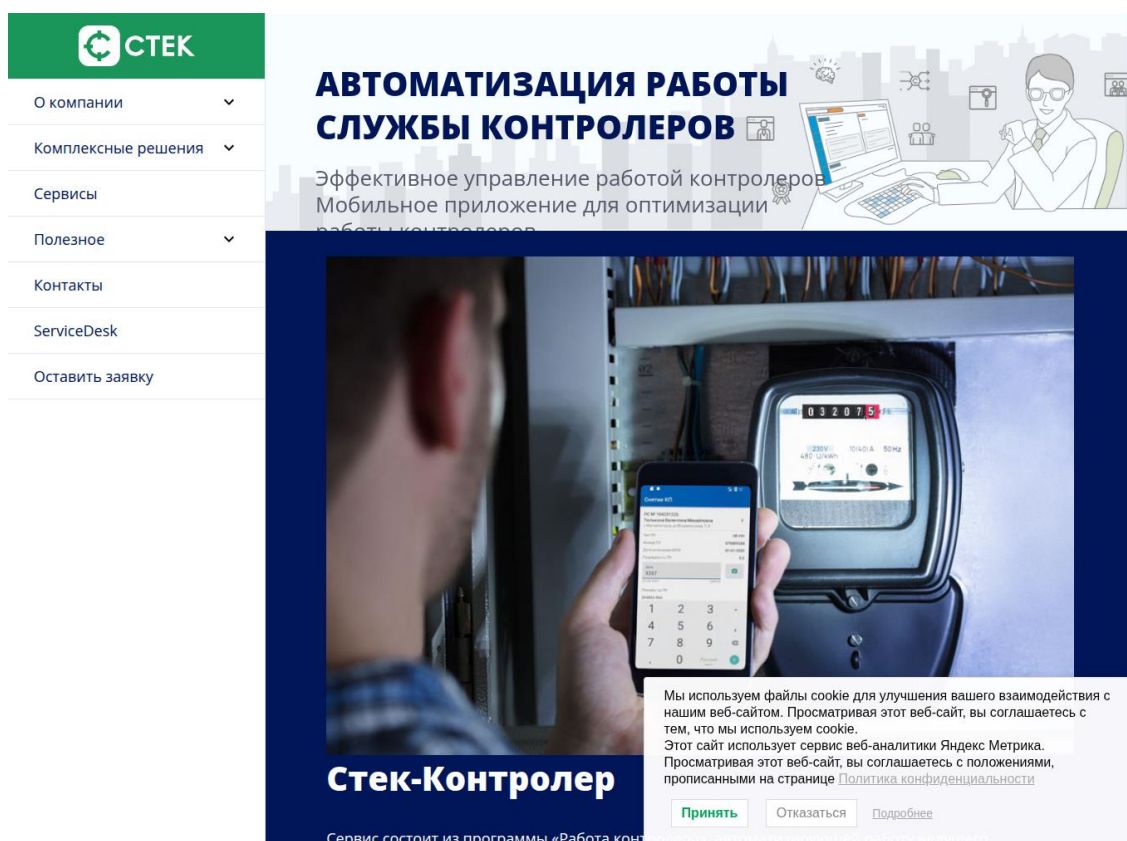


Рисунок 4 – Страница сервиса СТЕК-Контролер

СТЕК-Контролер является наиболее близким аналогом для мобильного контура инспектора. Его сильные стороны – предметная близость к коммунальному и энергосбытовому учету, поддержка обходов, данных абонента, фотофиксации, GPS и интеграции с учетной системой. Ограничение для рассматриваемой работы состоит в том, что решение является готовым внешним продуктом со своей учетной экосистемой и не описывает архитектуру клиентского приложения, интегрированного с разрабатываемым серверным API, формой конкретных актов и текущим монорепозиторием.

Вторым аналогом является ПАК АСУ ПКПЭ НИИАС. Официальная страница описывает систему как программно-аппаратный комплекс для учета, контроля потребления электроэнергии, формирования отчетных документов и управленческой отчетности [4]. В перечне функций присутствуют бюджетирование, контроль потерь электроэнергии, отчетность по объемам и тарифам передачи, расчет индекса технического состояния оборудования и интеграционные модули [4]. Скриншот страницы приведен на рисунке 5.

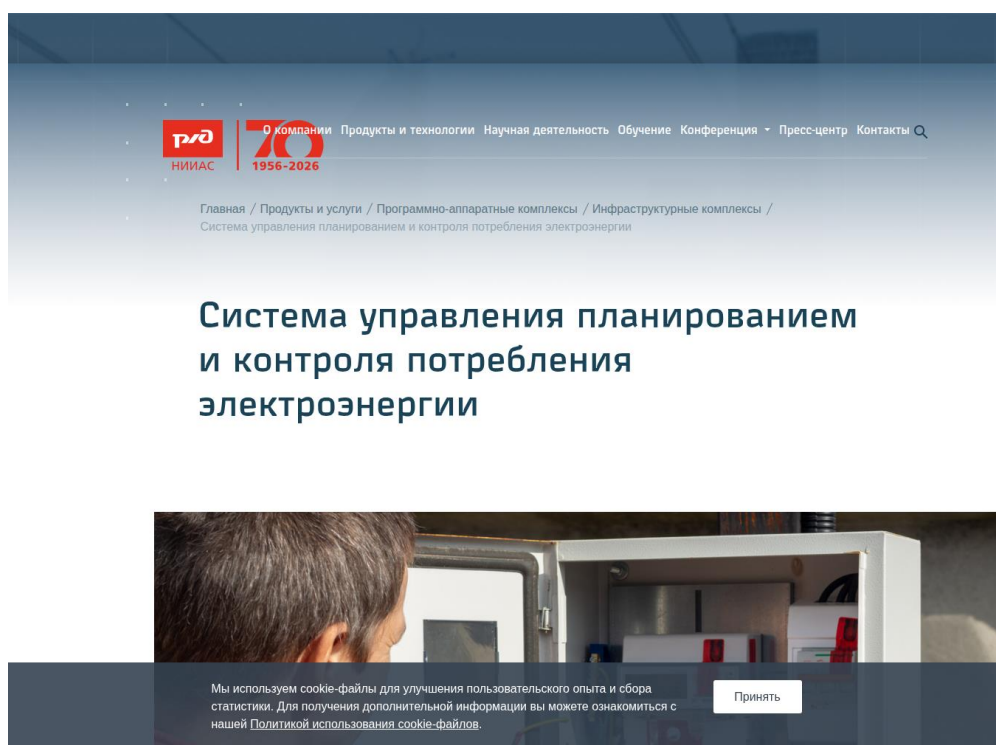


Рисунок 5 – Страница ПАК АСУ ПКПЭ НИИАС

ПАК АСУ ПКПЭ полезен как пример крупной отраслевой системы, ориентированной на управленческий и аналитический контур. Его сильная сторона – широкий охват процессов контроля потребления электроэнергии и формирование отчетности. Для темы ВКР это решение показывает, что контроль электроэнергии требует не только мобильного сбора данных, но и последующей аналитики. При этом оно не является специализированным клиентским приложением для мобильной бригады и диспетчерской панели в малом контуре задач: акцент смещен к корпоративной отчетности, бюджетам, потерям и инфраструктурным показателям.

Третьим аналогом является SafetyCulture. В справочном центре SafetyCulture описана как платформа операционной деятельности, доступная через web и мобильные приложения. Среди функций указаны шаблоны, inspections, schedules, actions, issues, documents, assets, analytics и другие модули [5]. Система позволяет выполнять проверки, заполнять формы и формировать отчеты, а также назначать действия ответственным пользователям [5]. Скриншот страницы приведен на рисунке 6.

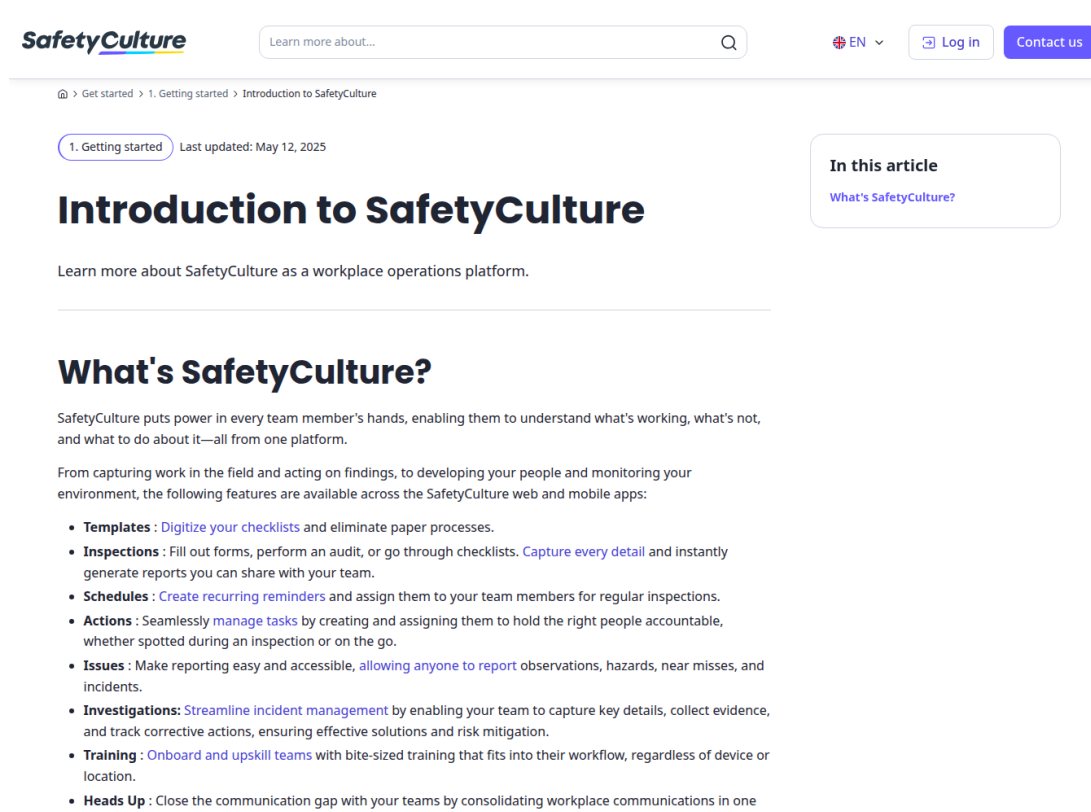


Рисунок 6 – Страница справочного центра SafetyCulture

SafetyCulture является сильным примером универсальной мобильной платформы для проверок. Ее преимущества – развитая работа с шаблонами, мобильными проверками, действиями, документами и аналитикой. Однако система не ориентирована на российский энергосбытовой документооборот, не содержит готовой модели бригад инспекторов, задач проверки приборов учета, типов актов ограничения и возобновления электроснабжения. Для рассматриваемой ВКР SafetyCulture показывает ценность мобильной формы и фотофиксации, но не заменяет специализированное клиентское приложение.

Отдельно в качестве базового варианта необходимо учитывать ручной или полуручной процесс. Он не требует внедрения готового продукта, но сохраняет основные недостатки: бумажные акты, задержку обновления статусов, раздельное хранение фотографий, отсутствие карты и сложность формирования отчетности. Поэтому ручной процесс рассматривается как минимальный уровень, относительно которого оценивается эффект клиентского приложения.

Результаты сравнения аналогов представлены в таблице 3.

Таблица 3 – Сравнительный анализ информационных систем аналогичного назначения

Критерий	СТЕК-Контроль	ПАК АСУ ПКП Э	SafetyCulture	Ручной процесс	Разрабатываемое приложение
Мобильный доступ инспектора к задачам	2	0	2	0	2
Работа с картой, адресом и маршрутом	1	0	1	0	2
Фотофиксация прибора учета и пломб	2	0	2	1	2
Связь фото с конкретной проверкой и актом	2	0	1	0	2
Заполнение отраслевых актов на объекте	1	0	1	1	2
Работа с задачами бригад и статусами	1	0	1	0	2
Административное управление бригадами	2	0	1	0	2
Формирование отчетов по задачам и проверкам	1	2	2	1	2
Ориентация на энергосбытовую предметную область	2	2	0	1	2

Продолжение таблицы 3

Критерий	СТЕК-Контролер	ПАК АСУ ПКПЭ	SafetyCulture	Ручной процесс	Разрабатываемое приложение
Интеграция с существующим серверным API проекта	0	0	0	0	2
Соответствие текущим сценариям инспектора и диспетчера	0	0	0	0	2

Для оценки использована шкала: 0 баллов – критерий не поддерживается, 1 балл – поддерживается частично, 2 балла – поддерживается полностью. СТЕК-Контролер является самым близким функциональным аналогом, но он не решает задачу проектирования клиентского приложения внутри текущей информационной системы и не может быть использован как готовый модуль без изменения архитектуры проекта. ПАК АСУ ПКПЭ подтверждает важность отчетности и аналитики, но относится к более крупному корпоративному уровню. SafetyCulture демонстрирует сильную мобильную механику проверок, однако является универсальной платформой без отраслевой модели энергосбыта.

Таким образом, разработка собственного клиентского приложения обоснована. Оно должно объединить сильные стороны аналогов: мобильную работу с задачами и фотофиксацией, диспетчерское управление, отчетность, а также предметную привязку к приборам учета, пломбам, актам ограничения, возобновления и проверок.

1.3 Постановка задачи на разработку клиентского приложения

1.3.1 Общая концепция клиентского приложения

Разрабатываемое клиентское приложение предназначено для работы с информационной системой контроля и анализа потребления электроэнергии. Оно должно предоставить пользователям интерфейсы для выполнения

повседневных операций без прямого обращения к базе данных или внутренним сервисам.

В составе клиентской части выделяются два основных модуля: мобильный АРМ инспектора и административная панель диспетчера. Мобильный модуль используется бригадой на объекте и рассчитан на небольшой экран, быстрый ввод данных, работу с камерой и геолокацией. Административная панель используется диспетчером на рабочем месте и рассчитана на таблицы, фильтры, карточки сущностей и контроль статусов.

Состав клиентских модулей представлен в таблице 4.

Таблица 4 – Состав клиентских модулей и классов пользователей

Клиентский модуль	Класс пользователя	Назначение модуля	Основные функции
Мобильный АРМ инспектора	Инспектор, бригада инспекторов	Выполнение проверок на объектах энергопотребления	Список задач, карта, карточка задачи, фотофиксация, заполнение акта, просмотр актов и отчетов
Административная панель диспетчера	Диспетчер	Планирование и контроль работы бригад	Управление бригадами, назначение задач, контроль статусов, просмотр проверок, отчеты
Общий API-клиент	Клиентские модули	Единый типизированный доступ к серверной информационной системе	Запросы к задачам, бригадам, абонентам, проверкам, файлам и отчетам

Оба модуля работают с одними и теми же предметными сущностями, но отображают их в разном контексте. Инспектор воспринимает задачу как объект выполнения проверки, а диспетчер – как элемент плана работ, связанный с бригадой, сроком, статусом и результатом.

1.3.2 Функциональные требования

Функциональные требования определяют, каким должно быть поведение программного средства в заданных условиях: какие действия должны быть доступны пользователю, какие данные должна принимать система и какой результат должен быть получен после выполнения операции [6]. Такие требования описывают не внутреннюю реализацию приложения, а ожидаемые функции с точки зрения пользователей и предметной области.

Для клиентского приложения информационной системы контроля и анализа потребления электроэнергии функциональные требования связаны с двумя основными группами пользователей. Первая группа – инспекторы и бригады инспекторов, которые работают на объекте энергопотребления и фиксируют результат проверки. Вторая группа – диспетчеры, которые планируют работу бригад, распределяют задачи и контролируют ход выполнения проверок.

Разрабатываемое приложение должно поддерживать полный цикл работы с задачами: получение назначения, просмотр сведений об объекте, выполнение проверки, фотофиксацию, заполнение акта, передачу результата на сервер и последующий контроль выполненной работы. Поэтому требования сформулированы не как отдельные экранные элементы, а как пользовательские возможности, необходимые для выполнения бизнес-процесса.

Функциональные требования к разрабатываемому клиентскому приложению целесообразно разделить по пользовательским приложениям, поскольку мобильный АРМ инспектора и АРМ диспетчера решают разные задачи в одном бизнес-процессе.

К функциональным требованиям мобильного АРМ инспектора относятся:

- выполнение входа в мобильное приложение по идентификатору бригады или учетным данным пользователя;
- проверка корректности введенных данных авторизации перед отправкой запроса на сервер;

- отображение основной информации о задаче: адрес объекта энергопотребления, данные абонента, контактный телефон, лицевой счет, плановое время визита и комментариев;
- открытие карточки задачи из общего списка задач или с карты;
- отображение плановых задач на карте с привязкой к координатам объекта;
- перевод выбранной задачи в статус выполнения перед началом работы на объекте;
- сохранение основных данных задачи в форме акта, чтобы инспектор не вводил повторно адрес, лицевой счет и сведения об абоненте;
- последовательное заполнение многошаговой формы акта на мобильном устройстве;
- ввод контактной информации абонента и проверка обязательных полей формы;
- выбор типа оформляемого акта в зависимости от результата проверки;
- ввод сведений, специфичных для выбранного типа акта;
- прикрепление фотографии прибора учета с камеры мобильного устройства;
- прикрепление фотографии пломбы или другого средства визуального контроля;
- передача изображений на сервер с привязкой к адресу объекта и данным прибора учета;
- создание записи о проверке на основе введенных инспектором данных;
- просмотр детальной информации по созданному акту или выполненной задаче;
- отображение отчетов, доступных инспектору по результатам выполненных работ.

К функциональным требованиям АРМ диспетчера относятся:

- выполнение входа в административную панель пользователем с ролью диспетчера;
- просмотр списка активных и архивных бригад;
- создание новой бригады из доступных инспекторов;
- изменение состава существующей бригады;
- перевод бригады в архив, если она больше не используется в планировании;
- просмотр общего списка задач отдела контроля;
- назначение задачи на выбранную бригаду;
- изменение планового времени выполнения задачи;
- просмотр результатов выполненных проверок, приложенных фотографий и сформированных актов;
- получение статистики по выполненным задачам за выбранный период;
- формирование отчета о проделанной работе на основе агрегированных данных;
- получение данных для информационной панели диспетчера, отражающей состояние задач, бригад и результатов проверок.

Диаграмма вариантов использования показывает набор пользовательских сценариев, действующих лиц и связей между ними. Она относится к статическому представлению системы и позволяет увидеть, какие функции доступны каждой категории пользователей [7].

Для рассматриваемого приложения были выделены две диаграммы вариантов использования: для мобильного АРМ инспектора и для административной панели диспетчера. Такое разделение необходимо, потому что инспектор выполняет проверку непосредственно на объекте, а диспетчер управляет распределением задач и анализирует результаты работы отдела. Диаграмма вариантов использования для мобильного АРМ инспектора представлена на рисунке 7.

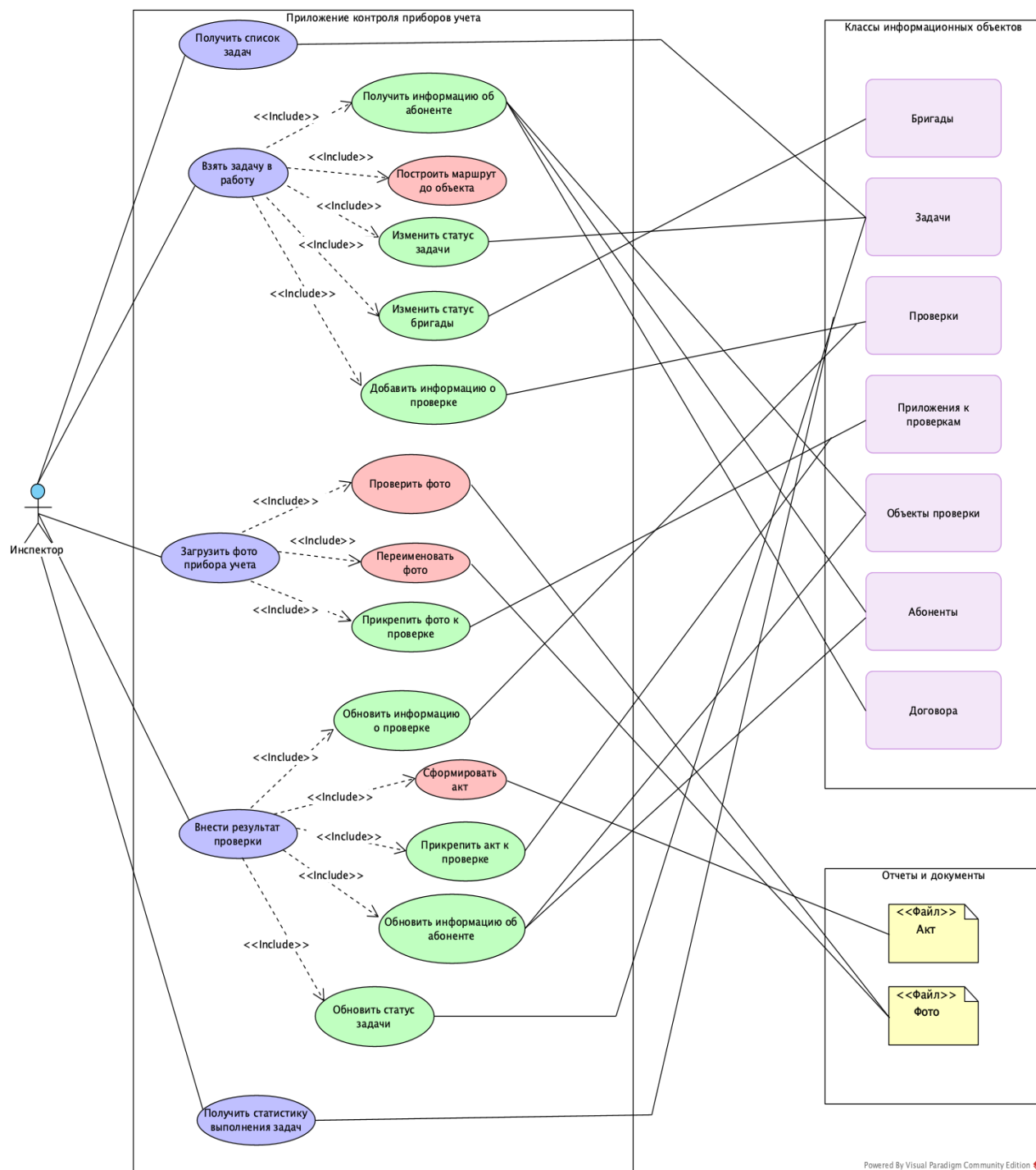


Рисунок 7 – Диаграмма вариантов использования для АРМ инспектора

Диаграмма вариантов использования АРМ диспетчера дополняет мобильный сценарий инспектора. Она показывает, что кабинетный пользователь работает не с отдельной проверкой, а с набором взаимосвязанных операций: созданием задач, назначением бригад, просмотром результатов, управлением пользователями и формированием отчетности. Такое разделение сценариев подтверждает необходимость двух клиентских модулей в составе одной

системы.

Диаграмма вариантов использования для административной панели диспетчера приведена на рисунке 8.

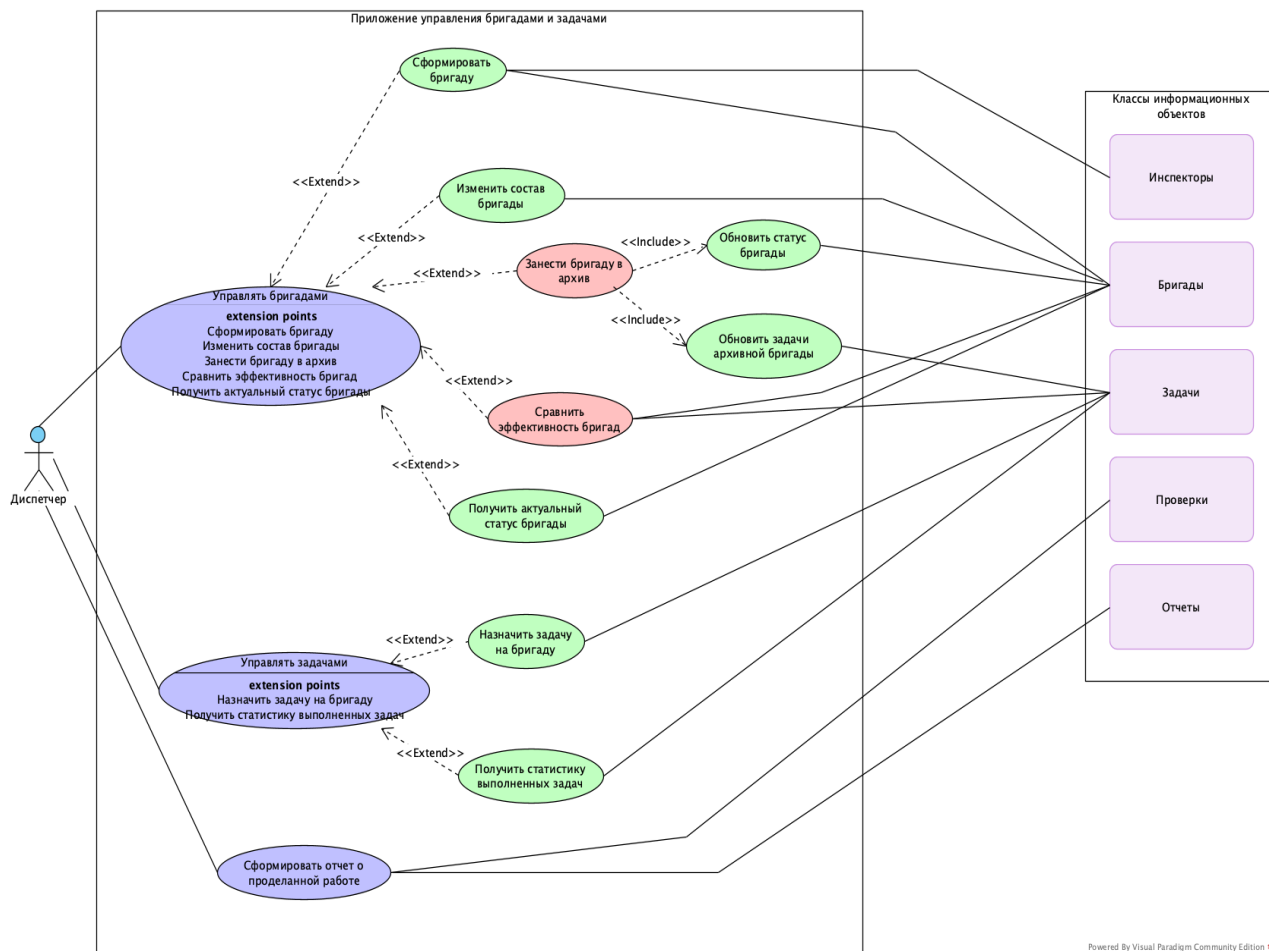


Рисунок 8 – Диаграмма вариантов использования для АРМ диспетчера

Сопоставление двух диаграмм позволяет выделить общие сущности, которые должны одинаково интерпретироваться в обоих интерфейсах. К ним относятся задача, бригада, абонент, объект электроснабжения, проверка, вложение и отчет. Поэтому дальнейшее проектирование клиентской части строится вокруг общей доменной модели, а не вокруг независимых экранов, что позволяет поддерживать единый источник истины.

Такой подход снижает дублирование бизнес-логики, упрощает дальнейшее сопровождение приложения, а также обеспечивает согласованность данных между различными пользовательскими сценариями.

1.3.3 Нефункциональные требования

Нефункциональные требования описывают свойства и ограничения программного средства, которые не сводятся к отдельным пользовательским действиям, но определяют качество работы системы в целом [6]. К таким свойствам относятся удобство использования, надежность, безопасность, производительность, совместимость, сопровождаемость и возможность дальнейшего развития приложения.

Для клиентского приложения информационной системы контроля и анализа потребления электроэнергии нефункциональные требования особенно важны, потому что система используется в разных условиях. Инспектор работает с мобильным устройством на объекте, где требуется быстрый ввод данных и стабильная отправка фотографий. Диспетчер работает с большим количеством задач и должен быстро получать общую картину по бригадам и проверкам.

К нефункциональным требованиям мобильного АРМ инспектора относятся:

- интерфейс должен быть удобен для использования на небольшом экране и не требовать сложной навигации при выполнении проверки;
- основные действия инспектора – просмотр задачи, переход к карте, начало работы, добавление фотографии и заполнение формы – должны быть доступны в логически последовательном порядке;
- приложение должно корректно работать как в мобильном браузере, так и в Android-сборке через Capacitor;
- экранные формы должны быть пригодны для заполнения во время выезда на объект энергопотребления;
- пользователь не должен иметь возможность отправить акт с незаполненными обязательными полями;
- временная недоступность сервера или ошибка загрузки фотографии не должна приводить к потере уже введенных данных формы;
- приложение должно показывать понятные сообщения при ошибках

загрузки задач, отправки фотографий и создания проверки;

- фотографии прибора учета и пломбы должны сохранять связь с задачей, адресом объекта и данными прибора учета;

- сведения об абоненте должны отображаться только в объеме, необходимом инспектору для выполнения проверки;

- инспектор должен видеть только задачи, акты и результаты, относящиеся к его бригаде;

- изменение статуса задачи должно быть отражено в интерфейсе без повторного прохождения всего сценария;

- мобильный модуль должен оставаться сопровождаемым при добавлении новых типов актов, полей проверки и вариантов фотофиксации.

К нефункциональным требованиям АРМ диспетчера относятся:

- административная панель должна быть рассчитана на настольный экран и поддерживать удобную работу с большим количеством записей;

- интерфейс диспетчера должен обеспечивать быстрый просмотр состояния бригад, задач и результатов проверок;

- списки задач, бригад, проверок и отчетов должны открываться без заметных задержек при штатном объеме данных;

- диспетчер должен получать понятные сообщения об ошибках при загрузке данных, назначении задач и формировании отчетов;

- доступ к функциям управления бригадами, задачами и отчетами должен предоставляться только пользователям с соответствующей ролью;

- административная панель должна корректно отображаться в современных настольных браузерах;

- отчетные представления должны сохранять согласованность с данными задач, проверок, бригад и актов;

- интерфейс должен оставаться удобным при увеличении количества бригад, задач и выполненных проверок;

- добавление новых статусов задач, отчетных представлений и

административных разделов не должно требовать полной переработки панели;

– код административной панели должен быть разделен на модули, соответствующие предметным сущностям и пользовательским сценариям диспетчера.

Перечисленные нефункциональные требования определяют ограничения, которые необходимо учитывать при проектировании мобильного АРМ инспектора и АРМ диспетчера. Таким образом, функциональные требования задают состав сценариев каждого приложения, а нефункциональные требования определяют качество, надежность и удобство их выполнения.

2 Проектирование клиентского приложения для информационной системы контроля и анализа потребления электроэнергии

2.1 Выбор общей архитектуры клиентской части

Клиентская часть проектируется как монорепозитарий, содержащий несколько приложений и общий пакет типизированного доступа к серверному API. Такое решение соответствует предметной области: мобильный инспекторский сценарий и диспетчерский сценарий имеют разные интерфейсы, но работают с общей моделью задач, бригад, проверок, абонентов и отчетов.

В проекте выделяются три основные части: приложение инспектора, административная панель и общий пакет обмена с сервером. Приложение инспектора рассчитано на мобильный веб-интерфейс с возможностью Android-сборки. Административная панель ориентирована на настольный браузер диспетчера. В основе системы стоит принцип модульности, что позволяет гибко масштабировать функциональность системы без потери управляемости.

Общий пакет содержит сгенерированные функции серверных запросов и общие типы данных, которые не должны дублироваться в приложениях с целью избежать дублирование логики при росте, масштабировании и поддержке клиентского модуля системы.

Ключевые проектные решения приведены в таблице 5.

Таблица 5 – Основные проектные решения клиентского приложения

Проектное решение	Причина выбора	Следствие для реализации
Монорепозиторий на <code>nrpm</code> и <code>Turborepo</code>	Нужно сопровождать мобильное и административное приложение в одной базе исходного кода	Сборка и установка зависимостей выполняются из корня проекта, приложения разделены по рабочим пакетам
Отдельные приложения инспектора и диспетчера	Сценарии инспектора и диспетчера отличаются устройством, навигацией и плотностью интерфейса	Мобильный интерфейс не смешивается с настольной административной панелью
Общий пакет обмена с сервером	Оба приложения обращаются к одним серверным данным	Описание запросов обновляется централизованно по серверному контракту
Слой предметных сущностей внутри каждого приложения	Экранные компоненты не должны зависеть от технического формата ответа сервера	Страницы получают типы, функции загрузки и подготовленные данные через предметные модули
Маршрутизация <code>React Router</code>	Нужна явная схема переходов между экранами приложения	Маршруты разделены на мобильный и административный контуры
<code>Capacitor</code> для <code>Android</code> -сборки инспектора	Инспектору нужен мобильный сценарий с камерой и геолокацией	Один и тот же интерфейс поставляется как веб-версия и <code>Android</code> -приложение

Выбранный подход уменьшает дублирование описаний серверных запросов и позволяет развивать разные пользовательские интерфейсы независимо. При изменении серверного контракта сгенерированный API-клиент обновляется в общем пакете, а приложения получают доступ к нему через собственные слои предметных сущностей.

2.1.1 Архитектура клиентских веб-приложений на основе FSD

Для организации клиентского кода применен адаптированный подход `Feature-Sliced Design`. Он используется не как формальное переименование всех каталогов, а как правило распределения ответственности: верхний уровень собирает приложение и маршруты, страницы описывают пользовательские сценарии, предметные сущности отвечают за данные и операции предметной

области, а общие элементы интерфейса и утилиты не зависят от конкретного экрана. Такой подход соответствует официальной модели FSD, где код делится на слои, предметные срезы и внутренние сегменты [8].

В проекте это выражено в разделении на мобильное приложение инспектора, административную панель и общий пакет обмена с сервером. Внутри приложений страницы отделены от предметных модулей, базовых элементов интерфейса и прикладных блоков. Направление зависимостей сохраняется сверху вниз: экран может использовать предметную сущность и общий компонент, но предметная сущность не должна знать о конкретной странице.

Компонентное устройство клиентской части, в котором показаны приложения инспектора и диспетчера, общий пакет запросов, Android-контейнер и серверная система, представлено на рисунке 9.

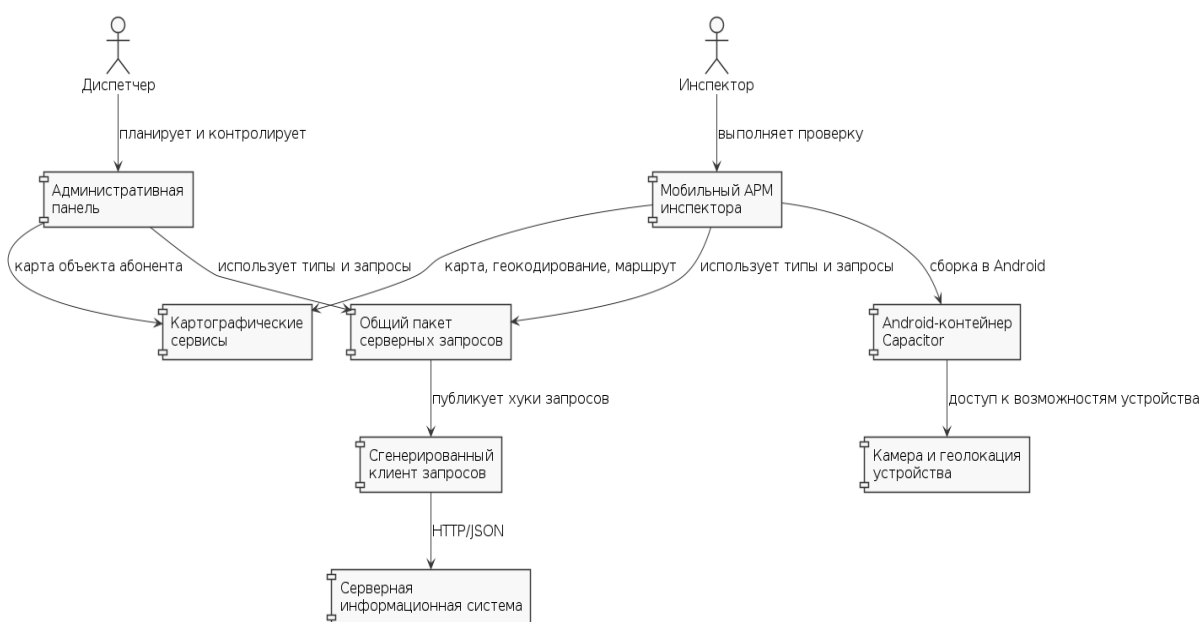


Рисунок 9 – Диаграмма компонентов клиентских приложений

Из диаграммы компонентов следует, что оба пользовательских рабочих места используют единый типизированный доступ к данным, но имеют разные интерфейсные оболочки. Это позволяет развивать мобильные сценарии инспектора и кабинетные сценарии диспетчера без копирования описания серверных запросов.

Среда развертывания клиентских приложений представлена на рисунке 10.

Такое развертывание сохраняет разные формы поставки, но оставляет единый контур обмена данными между клиентскими модулями и серверной частью.

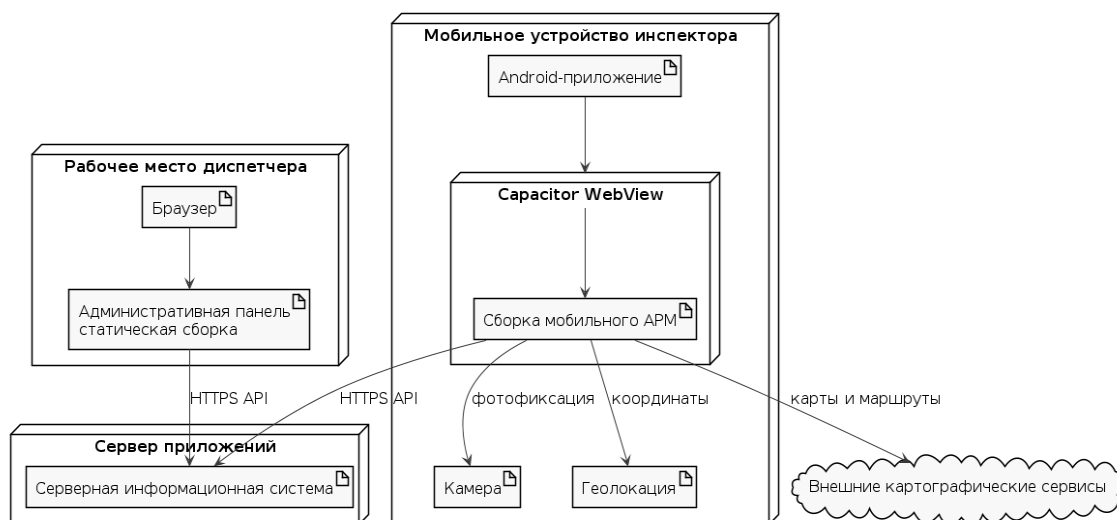


Рисунок 10 – Диаграмма развертывания клиентской части

На диаграмме развертывания видно различие поставки двух клиентских рабочих мест: административная панель работает в браузере диспетчера, а мобильный АРМ может запускаться внутри Android-приложения через WebView Capacitor и обращаться к камере и геолокации устройства [9].

Коммуникация диспетчера, инспектора и серверной системы при выполнении проверки показана на рисунке 11.

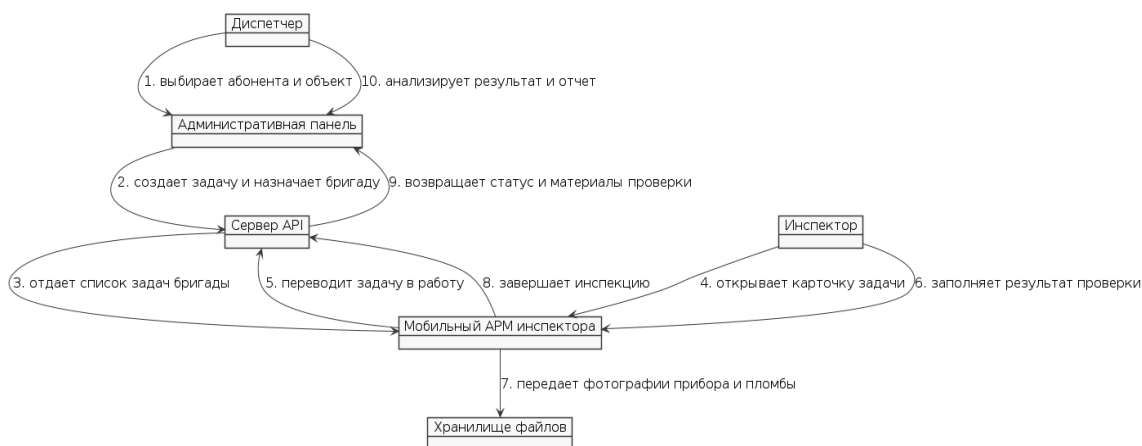


Рисунок 11 – Диаграмма коммуникации планирования и выполнения проверки

Диаграмма коммуникации фиксирует основной предметный контур: диспетчер создает и назначает задачу, инспектор выполняет проверку на объекте, а результат возвращается в административную панель вместе с фотоматериалами.

2.2 Проектирование маршрутов и навигации

Маршруты мобильного приложения должны соответствовать рабочему циклу инспектора. После входа пользователь попадает к списку задач, может перейти к карте, карточке задачи, форме создания акта, списку актов и отчетам. Все рабочие маршруты защищаются проверкой авторизации, а маршрут входа остается доступным без защиты. При проектировании маршрутизации учитывались возможности React Router по описанию маршрутов и клиентской навигации [10].

Административная панель имеет отдельный префикс /admin и собственную компоновку страниц. Диспетчер работает со сводным экраном, бригадами, задачами, проверками, абонентами и отчетами. Навигация проектируется как постоянное боковое меню, потому что основной экран диспетчера рассчитан на таблицы и карточки, а не на нижнюю мобильную навигацию.

Структура маршрутов представлена в таблице 6.

Таблица 6 – Проектируемая структура маршрутов клиентского приложения

Контур	Маршрут	Назначение
Мобильный АРМ инспектора	/	Список задач бригады
Мобильный АРМ инспектора	/map	Карта плановых задач
Мобильный АРМ инспектора	/acts/:id	Карточка задачи и переход к составлению акта

Продолжение таблицы 6

Контур	Маршрут	Назначение
Мобильный АРМ инспектора	/create	Многошаговая форма акта
Мобильный АРМ инспектора	/acts	Список созданных актов и проверок
Мобильный АРМ инспектора	/report	Отчеты, доступные бригаде
Мобильный АРМ инспектора	/login	Вход в мобильное приложение
Административная панель	/admin	Дашборд диспетчера
Административная панель	/admin/brigades	Список бригад
Административная панель	/admin/tasks	Список и назначение задач
Административная панель	/admin/inspections	Просмотр результатов проверок
Административная панель	/admin/subscribers	Просмотр абонентов и объектов
Административная панель	/admin/reports	Формирование отчетов

Разделение маршрутов позволяет сохранить мобильную эргономику инспекторского приложения и одновременно спроектировать административный интерфейс как рабочий инструмент диспетчера.

2.3 Проектирование доменной структуры и данных

Предметная область приложения содержит сущности, которые используются обоими клиентскими модулями. Чтобы экранный слой не зависел от технических типов серверного контракта напрямую, для каждой значимой сущности проектируется локальный слой предметных сущностей. В нем размещаются публичные типы, преобразование данных, функции запросов и вспомогательные функции.

Для мобильного приложения наиболее важны сущности бригады, задачи, проверки, файла, геоданных, отчета и абонента. Для административной панели дополнительно важны расширенные представления бригад, задач, проверок, абонентов и отчетов. При этом прямые обращения к сгенерированному API-

клиенту должны оставаться внутри слоя предметных сущностей, а страницы должны получать уже подготовленные данные для отображения.

Концептуальная модель данных информационной системы, подготовленная на этапе анализа предметной области, представлена на рисунке 12.

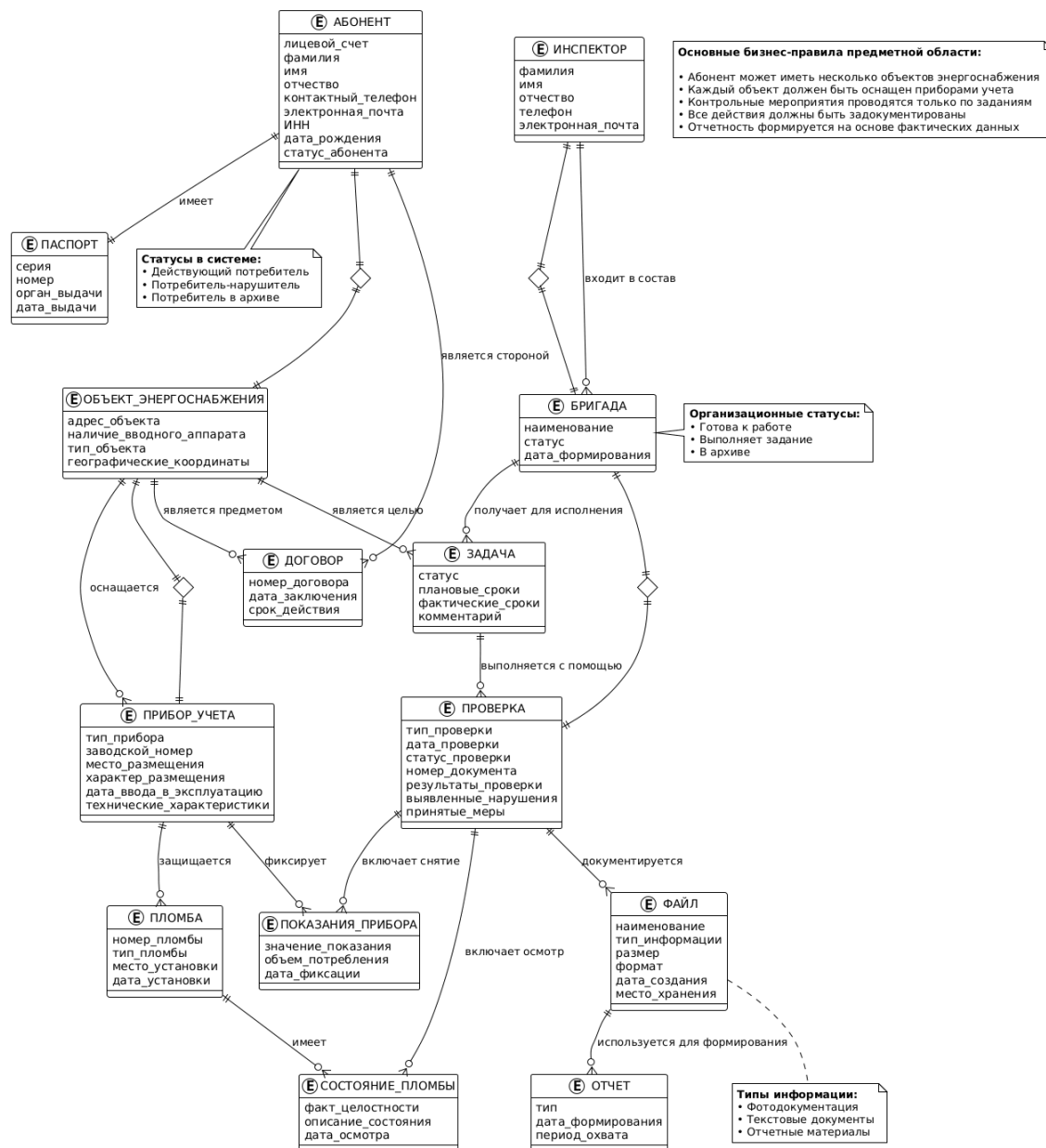


Рисунок 12 – Концептуальная модель данных информационной системы

Модель на рисунке 12 показывает, что клиентское приложение работает не с изолированными экранами, а со связанными сущностями. Задача связана с бригадой и объектом, проверка связана с задачей, приборы и пломбы связаны с объектом и результатами проверки, а файлы являются вложениями к проверке.

Основные сущности, их назначение, а также использование в клиентских модулях в приложении приведены в таблице 7.

Таблица 7 – Основные доменные сущности клиентского приложения

Сущность	Назначение	Использование в клиентских модулях
Бригада	Группа инспекторов, выполняющая назначенные задачи	Вход инспектора, список бригад, назначение задач
Задача	Плановая или выполняемая работа по объекту энергопотребления	Список задач, карта, карточка задачи, статусы диспетчера
Проверка	Результат выполнения задачи и основа для акта	Заполнение формы, просмотр результатов, завершение задачи
Абонент	Потребитель, связанный с объектом проверки	Карточка задачи, административный справочник
Объект проверки	Адрес и место установки прибора учета	Карта, карточка задачи, формирование акта
Прибор учета	Устройство, состояние которого проверяет инспектор	Ввод номера, типа, показаний и фотографий
Пломба	Средство визуального контроля	Фиксация номера, места установки и фото
Файл	Фотография или сформированный документ	Загрузка фото, просмотр вложений проверки
Отчет	Сводный документ по выполненным работам	Просмотр и формирование отчетов

Такая структура данных позволяет сохранить единый предметный язык в обоих приложениях и при этом адаптировать отображение под разные роли пользователей.

2.4 Проектирование сценария работы инспектора с актом

Ключевым сценарием мобильного приложения является оформление результата проверки. Для мобильного экрана выбрана многошаговая форма, так как она уменьшает когнитивную нагрузку на инспектора и позволяет проверять обязательные поля постепенно.

Сценарий начинается не с произвольного создания акта, а с карточки

назначенной задачи. Инспектор открывает задачу, переводит ее в работу, после чего приложение переносит известные сведения в форму: лицевой счет, адрес, данные потребителя и другую информацию, доступную из карточки. Это снижает объем ручного ввода и уменьшает риск ошибок.

Дальнейший набор шагов зависит от выбранного типа акта. Общими являются ввод лицевого счета, контактных данных, загрузка фотографий и выбор типа акта. После выбора типа добавляются шаги, связанные с причиной ограничения, состоянием электроснабжения, способом выполнения действия, местоположением прибора, типом прибора, показаниями, нарушением ограничения и финальной отправкой.

Проектируемый порядок шагов формы приведен в таблице 8.

Таблица 8 – Структура многошаговой формы акта

Группа шагов	Шаг формы	Назначение
Общие шаги	Лицевой счет	Проверка или ввод идентификатора абонента
Общие шаги	Контактные данные	Фиксация телефона, ФИО и адреса
Общие шаги	Фотографии прибора и пломбы	Фотофиксация ключевых материалов проверки
Общие шаги	Тип акта	Выбор сценария: ограничение, возобновление, проверка или самовольное подключение
Специальные шаги	Причина ограничения	Указание основания для ограничения или проверки
Специальные шаги	Статус электроснабжения	Фиксация состояния услуги, времени и даты
Специальные шаги	Способ выполнения действия	Описание способа ограничения или возобновления
Специальные шаги	Местоположение и тип прибора	Уточнение характеристик прибора учета
Специальные шаги	Показания и пломбы	Ввод показаний, номеров и мест установки пломб
Завершение	Отправка акта	Передача результата проверки и вложений в серверную систему

Многошаговая форма должна хранить текущий шаг, историю переходов и выбранный тип акта. Это необходимо для корректной работы кнопки возврата и для ветвления сценария после выбора типа акта.

2.5 Проектирование интеграции с серверной информационной системой

Клиентское приложение не должно содержать собственную независимую бизнес-базу. Его задача состоит в получении данных от серверной информационной системы, отображении этих данных пользователю и передаче обратно результатов действий. Поэтому интеграция с API проектируется как отдельный слой поверх сгенерированного клиента серверных запросов.

Для новых и изменяемых серверных методов страницы и компоненты не должны обращаться к адресам запросов напрямую. Сначала сгенерированный API-клиент размещается в общем пакете, затем в приложении создается предметная обертка. Такой подход позволяет менять внутренний контракт API без массового переписывания пользовательского интерфейса.

Жизненный цикл проверки проектируется следующим образом: диспетчер создает или назначает задачу; инспектор запускает задачу; сервер создает проверку; мобильное приложение получает проверку по задаче; инспектор прикрепляет фотографии; после заполнения формы проверка завершается; связанная задача получает итоговый статус. Такой порядок исключает создание новых задач из мобильного акта и закрепляет планирование за административным контуром.

2.6 Проектирование пользовательского интерфейса

2.6.1 Проектирование пользовательского интерфейса АРМ диспетчера

Интерфейс АРМ диспетчера проектируется как настольная административная панель. Основной принцип такого интерфейса – быстро показать пользователю состояние отдела контроля: какие задачи назначены, какие бригады доступны, какие проверки уже выполнены и какие материалы

требуют просмотра. Поэтому в основе интерфейса находятся таблицы, фильтры, боковая навигация и карточки подробной информации.

Главная навигация административной панели должна обеспечивать переход между разделами задач, бригад, проверок, абонентов, пользователей и отчетов. Для диспетчера важно не проходить длинный мастер действий, а быстро переключаться между связанными сущностями. Например, при создании задачи он должен выбрать абонента и объект, назначить бригаду, указать плановое время, затем вернуться к списку задач и контролировать изменение статуса.

Пользовательский сценарий работы диспетчера с задачей представлен на диаграмме активности на рисунке 13. Диаграмма показывает, что интерфейс должен поддерживать как создание новой задачи, так и контроль уже назначенной проверки.

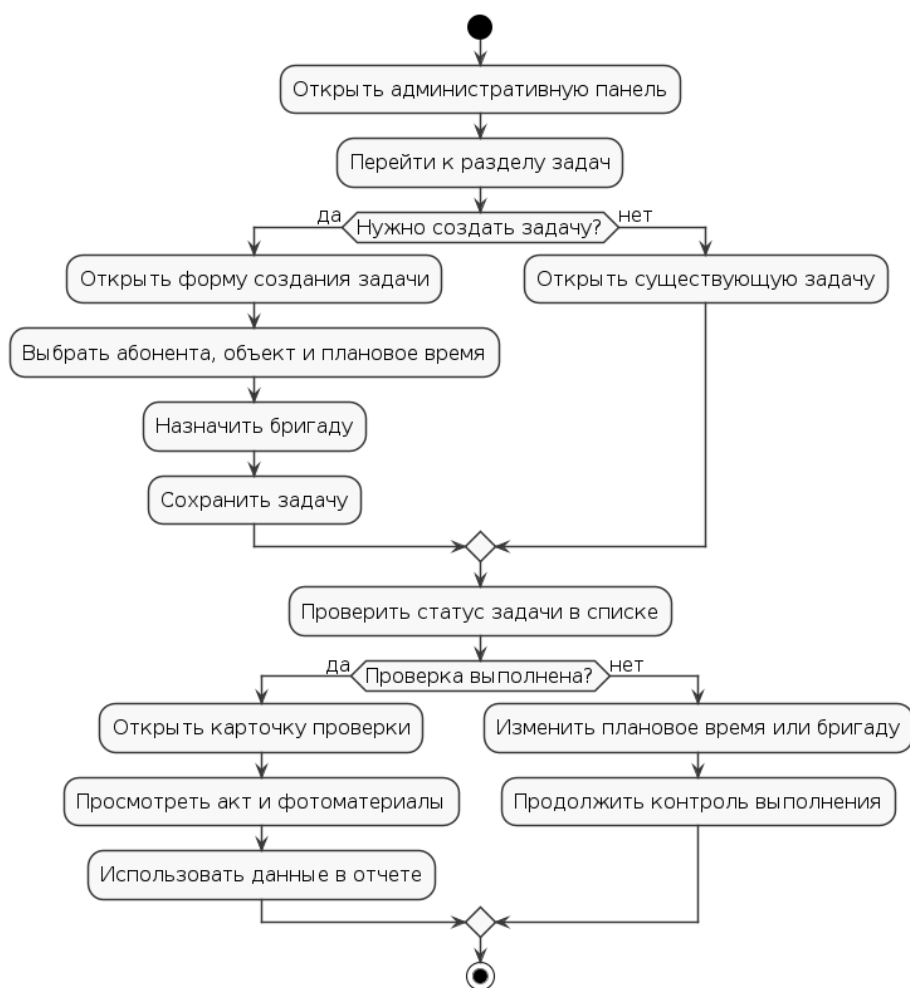


Рисунок 13 – Диаграмма активности работы диспетчера с пользовательским интерфейсом

В качестве примера спроектированного интерфейса АРМ диспетчера на рисунке 14 приведен экран управления задачами. Он демонстрирует табличную структуру, фильтрацию, действия по созданию задачи и переход к деталям записи.

Для диспетчера такой экран является рабочей точкой смены: из него удобно контролировать статусы, назначенную бригаду и переходить к деталям задачи. При проектировании статус, плановое время и назначенная бригада рассматривались как основные признаки быстрого контроля задачи.

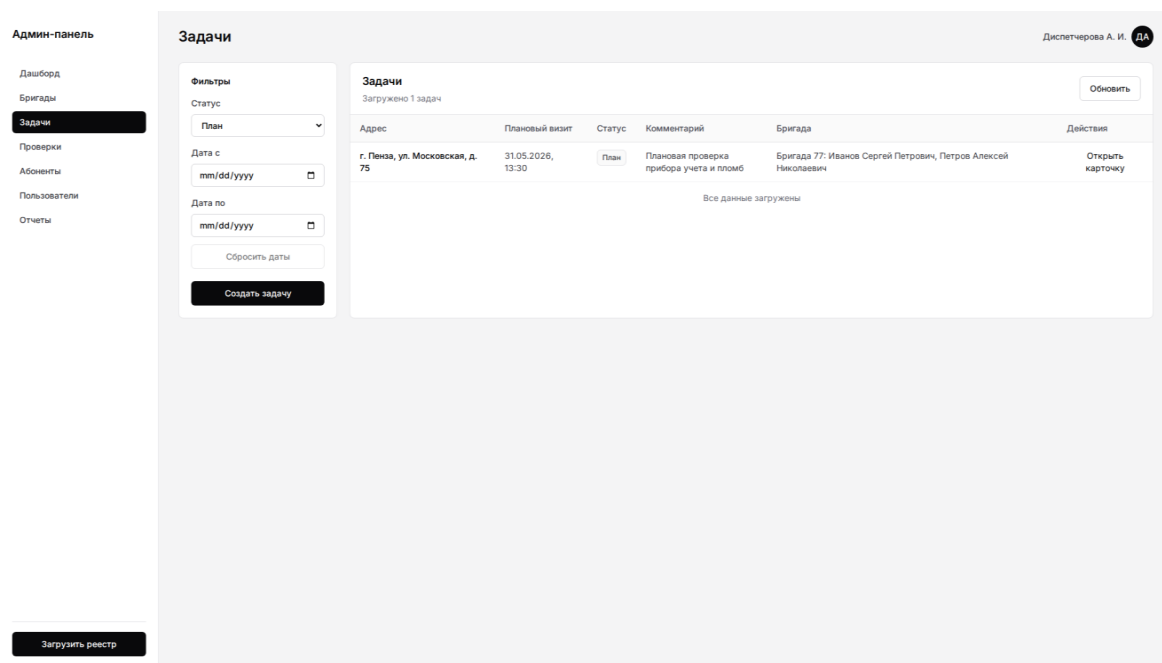


Рисунок 14 – Пример спроектированного интерфейса АРМ диспетчера

Такой подход позволяет использовать административную панель как рабочий инструмент планирования. Диспетчер видит не один изолированный экран, а взаимосвязанную систему разделов, где задача связана с бригадой, абонентом, объектом проверки, результатом инспекции и отчетом.

2.6.2 Проектирование пользовательского интерфейса АРМ инспектора

Интерфейс мобильного АРМ инспектора проектируется как полноэкранный мобильное приложение. Его основная задача – провести пользователя по линейному сценарию от получения назначенной задачи до

завершения проверки. В отличие от диспетчера, инспектор работает на объекте энергопотребления, поэтому интерфейс должен быть компактным, предсказуемым и пригодным для работы одной рукой.

Основные разделы мобильного АРМ должны быть доступны через нижнюю навигацию: список задач, карта, создание акта, список актов и отчеты. Список задач показывает состояние работы бригады, карточка задачи объединяет адрес, данные абонента и карту, а форма акта разбивается на последовательные шаги, что позволяет снизить нагрузку на инспектора и уменьшает вероятность пропуска обязательных сведений.

Состояния формирования акта представлены на диаграмме состояний на рисунке 15.

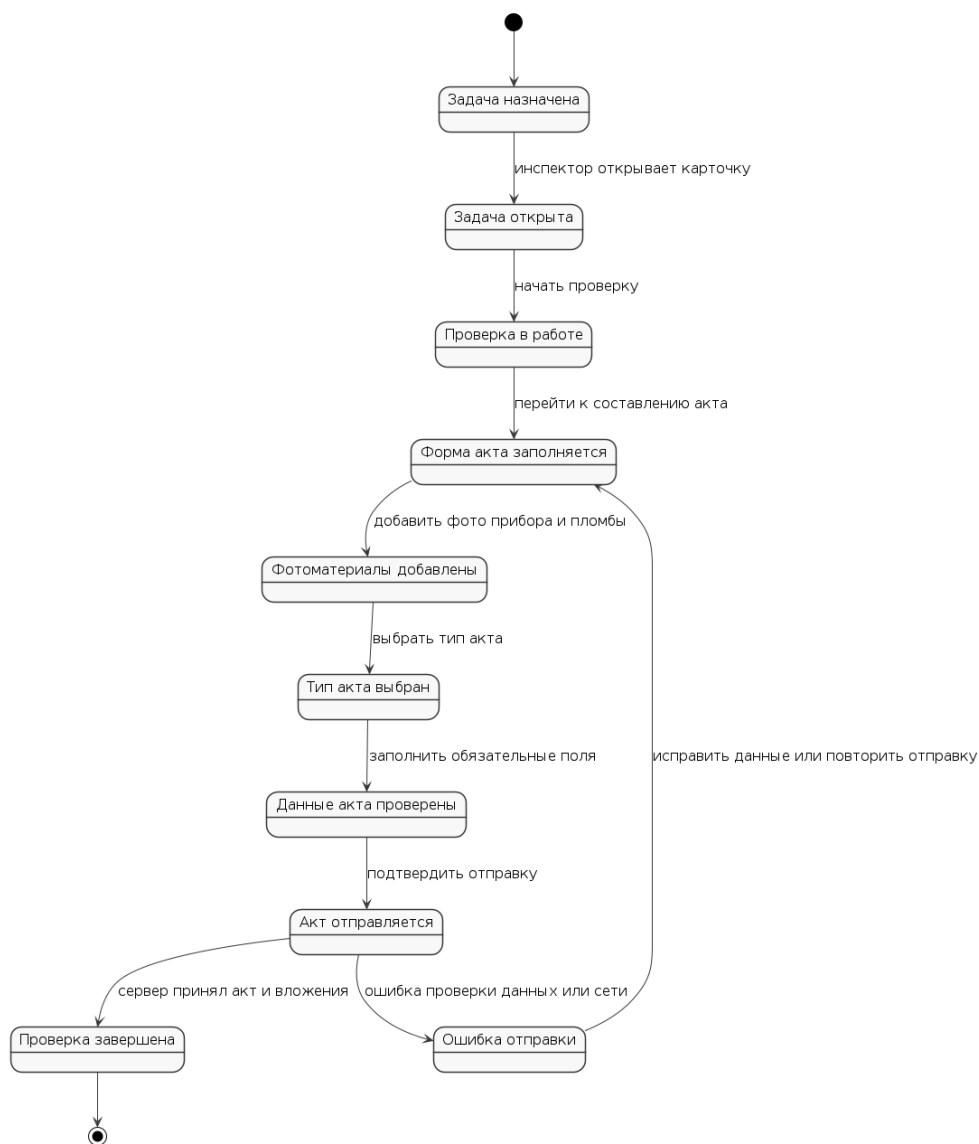


Рисунок 15 – Диаграмма состояний формирования акта и завершения проверки

Диаграмма показывает, что результат проверки проходит несколько этапов: задача назначена, карточка открыта, проверка переведена в работу, форма акта заполняется, фотографии добавлены, тип акта выбран, данные проверены и отправлены на сервер. Завершающим состоянием является успешное завершение проверки, а при ошибке пользователь возвращается к заполнению или повторной отправке.

Пример спроектированного мобильного интерфейса инспектора представлен на рисунке 16. Экран показывает карточку задачи, которая связывает сведения об объекте, карту и действие перехода к составлению акта.

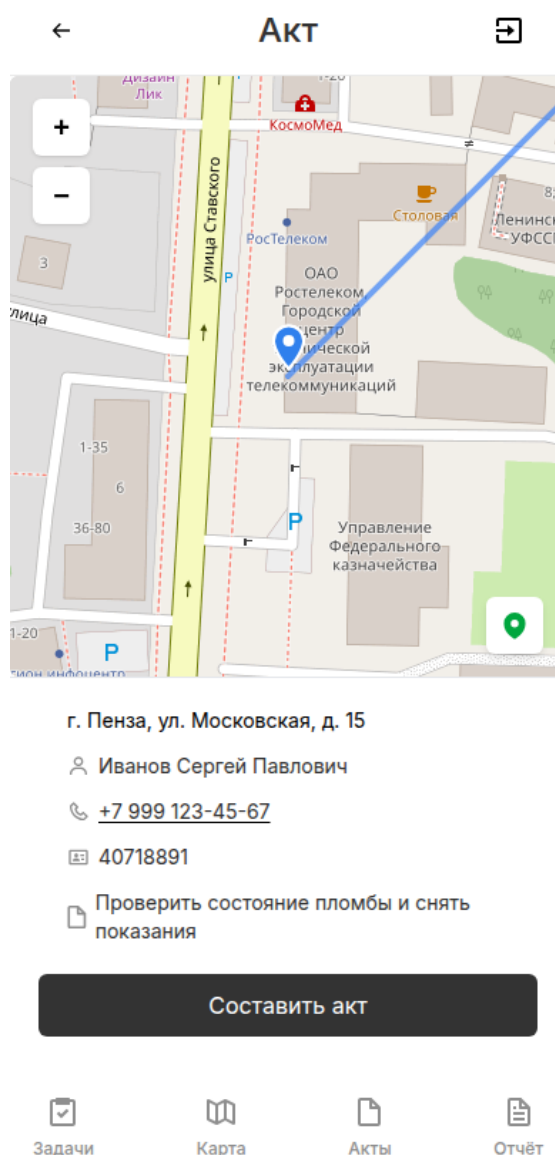


Рисунок 16 – Пример спроектированного интерфейса мобильного АРМ инспектора

В результате проектирования интерфейса мобильного АРМ инспектора определены основные принципы: короткие экраны, пошаговая форма, минимум ручного ввода, явные статусы задачи, поддержка камеры и сохранение связи между актом, фотоматериалами и задачей. Эти решения соответствуют условиям выездной работы и требованиям к надежному завершению проверки.

2.7 Требования к программному и аппаратному окружению

Проектируемое приложение должно запускаться в окружениях, соответствующих его пользовательским сценариям. Требования к окружению представлены в таблице 9.

Таблица 9 – Требования к программному и аппаратному окружению

Компонент	Требование
Мобильное устройство инспектора	Android-устройство с камерой, доступом к геолокации и интернет-соединением
Браузерный режим инспектора	Современный браузер с поддержкой JavaScript, доступа к сети и отображения карт
Рабочее место диспетчера	Настольный браузер с экраном, достаточным для работы с таблицами и фильтрами
Сборка клиентских приложений	Node.js, npm, Turborepo, Rsbuild
Android-сборка	Capacitor-проект с webDir мобильного приложения и Android target SDK
Серверная интеграция	Доступ к серверному API задач, бригад, проверок, абонентов, файлов и отчетов

Указанные требования отражают различие условий эксплуатации двух рабочих мест. Инспектору важны мобильность, камера, геолокация и устойчивый доступ к серверу, а диспетчеру – удобная работа с таблицами и карточками на настольном экране. Поэтому одно и то же клиентское приложение поставляется в разных формах: как веб-интерфейс, PWA и Android-сборка [11].

При этом требования к окружению не рассматриваются отдельно от пользовательских сценариев. Камера и геолокация нужны именно в момент выполнения проверки, широкая область экрана важна при диспетчерской работе

со списками, а воспроизводимая сборка необходима для передачи одинаковой версии приложения всем пользователям системы.

2.8 Обоснование технологических решений и вариантов поставки

При проектировании клиентского приложения было необходимо выбрать такой набор технологий, который позволит одновременно поддерживать мобильное рабочее место инспектора и административную панель диспетчера. Основным ограничением являлась необходимость быстро развивать пользовательские сценарии, не создавая отдельные независимые реализации для браузера и Android-устройства.

Для мобильного АРМ выбран подход веб-приложения с возможностью установки и упаковки в Android. На уровне браузера приложение имеет manifest-файл с режимом отдельного окна, стартовым адресом, цветами и набором иконок. Это соответствует модели PWA: приложение может запускаться как отдельное окно без видимой браузерной оболочки. При этом для доступа к возможностям устройства, таким как камера и геолокация, используется Capacitor, который переносит собранную веб-часть в нативный Android-проект и предоставляет доступ к платформенным возможностям через плагины.

Обоснование основных технологических решений приведено в таблице 10.

Таблица 10 – Обоснование технологических решений клиентского приложения

Решение	Причина выбора	Результат для проекта
React и TypeScript	Нужно было реализовать два интерактивных интерфейса с большим количеством состояний, форм и списков	Экраны построены как переиспользуемые компоненты, а типизация снижает риск ошибок при работе с данными сервера
Tailwind CSS	Для интерфейсов необходима быстрая сборка экранов без отдельной громоздкой системы стилей	В проекте используются единые утилитарные классы, CSS-переменные темы и адаптивные ограничения размеров

Продолжение таблицы 10

Решение	Причина выбора	Результат для проекта
Rsbuild	Нужна была быстрая сборка React-приложений и отдельный каталог результата для последующей поставки	Мобильное приложение и админка собираются в каталог build, который можно отдавать как статический веб-интерфейс
Автоматическая генерация API-клиента	Серверный контракт содержит много сущностей и операций, ручное описание запросов повышало бы вероятность расхождений	На основе OpenAPI создаются типизированные функции запросов и изменений, которые затем используются в доменных слоях приложений
PWA-подход	Инспектору важен запуск мобильного интерфейса как отдельного приложения и наличие иконок на устройстве	Веб-версия получает manifest со стартовым адресом, режимом отдельного окна, цветами и набором иконок
Capacitor	Нужно было сохранить единую веб-кодую базу и получить Android-приложение с доступом к камере, геолокации и файловым разрешениям	Собранная веб-часть синхронизируется с Android-проектом, после чего APK собирается штатными средствами Gradle

Таким образом, проектирование было ориентировано не только на внешний вид экранов, но и на способ поставки приложения. Мобильный интерфейс может работать как веб-приложение и как Android-приложение, а административная панель остается обычным веб-интерфейсом для сотрудников офиса. Выбранные технологические решения согласуются между собой: React обеспечивает компонентную модель интерфейса, общий пакет API уменьшает дублирование запросов, а Capacitor позволяет использовать мобильные возможности без разработки отдельного нативного клиента. Это делает архитектуру пригодной для дальнейшего расширения функций инспектора и диспетчера, а также позволяет сохранить согласованность подходов к разработке и повторно использовать значительную часть кода между различными клиентскими модулями.

3 Разработка программного обеспечения клиентского приложения

3.1 Организация разработки и структура клиентской части

Разработка велась в монорепозитории, где отдельно размещены мобильное приложение инспектора, административная панель диспетчера и общий пакет обмена данными с сервером. Такой способ организации позволил развивать пользовательские рабочие места независимо, но не дублировать описание серверных запросов и общих типов данных.

Мобильное приложение находится в каталоге `apps/inspector`. В нем реализованы вход бригады, список задач, карта, карточка задачи, заполнение акта, просмотр актов и отчетов. Административная панель находится в каталоге `apps/admin` и содержит разделы диспетчера: дашборд, бригады, задачи, проверки, абоненты, пользователи и отчеты. Общий пакет `packages/shared` содержит сгенерированный клиент серверного API.

В реализации FSD-подход проявляется в том, что страницы не содержат собственных описаний серверных адресов. Например, экран задач административной панели работает с предметными функциями задач, бригад и объектов, а мобильная карточка задачи получает подготовленные сведения о задаче, абоненте, приборе учета и пломбах. Общие элементы, такие как кнопки, поля ввода, таблицы, состояния загрузки и навигационные обертки, вынесены из конкретных страниц и повторно используются в разных сценариях.

Для разработки применялись React [12] и TypeScript [13]. React использован для построения экранов и управления состоянием интерфейса, TypeScript – для статической проверки структуры данных. Маршруты описаны отдельно от страниц, поэтому добавление нового раздела не требует изменения корневого компонента приложения. Состояния многошаговой формы акта и навигации мобильного приложения хранятся в Redux Toolkit [14], а данные, приходящие от сервера, проходят через слой предметных сущностей. Выбор React соответствует компонентной модели построения пользовательских интерфейсов.

Состав основных частей разработанного клиентского приложения приведен в таблице 11.

Таблица 11 – Состав разработанных частей клиентского приложения

Часть проекта	Назначение	Что было реализовано
Мобильный АРМ инспектора	Работа бригады на объекте проверки	Вход по бригаде, получение задач, карта, карточка объекта, форма акта, фотографии, акты и отчеты
Административная панель диспетчера	Планирование и контроль работы бригад	Разделы задач, бригад, проверок, абонентов, пользователей, отчетов и общей статистики
Общий пакет обмена данными	Единая связь с серверной информационной системой	Сгенерированный по OpenAPI клиент и типы данных, используемые обоими приложениями
Android-проект Capacitor	Поставка мобильного АРМ как Android-приложения	Нативная оболочка, разрешения Android, синхронизация веб-сборки и сборка APK

При разработке интерфейса использовался Tailwind CSS [15]. В мобильном приложении через CSS-тему заданы цвета, шрифт Inter и текстовые размеры, а экранные компоненты собираются из компактных утилитарных классов. Это особенно важно для мобильных форм: расстояния, высота кнопок, нижняя навигация и безопасные отступы экрана задаются единообразно во всех шагах. В административной панели тот же подход позволил быстро собрать таблицы, фильтры, панели и формы без ручного описания отдельного CSS-класса для каждого элемента.

3.2 Реализация мобильного АРМ инспектора

Работа инспектора начинается с экрана входа. Пользователь вводит идентификатор бригады, после чего приложение запрашивает сведения о бригаде и сохраняет необходимые данные в сеансе браузера. Экран входа показан на рисунке 17.

Войти

Бригада

Введите бригаду

Войти

Рисунок 17 – Экран входа бригады в мобильное приложение

После входа инспектор попадает на экран задач. На нем отображается состав бригады и список назначенных объектов. Задачи разделены по статусам, поэтому бригада видит плановые проверки, уже начатые работы и завершённые выезды. Экран списка задач представлен на рисунке 18.

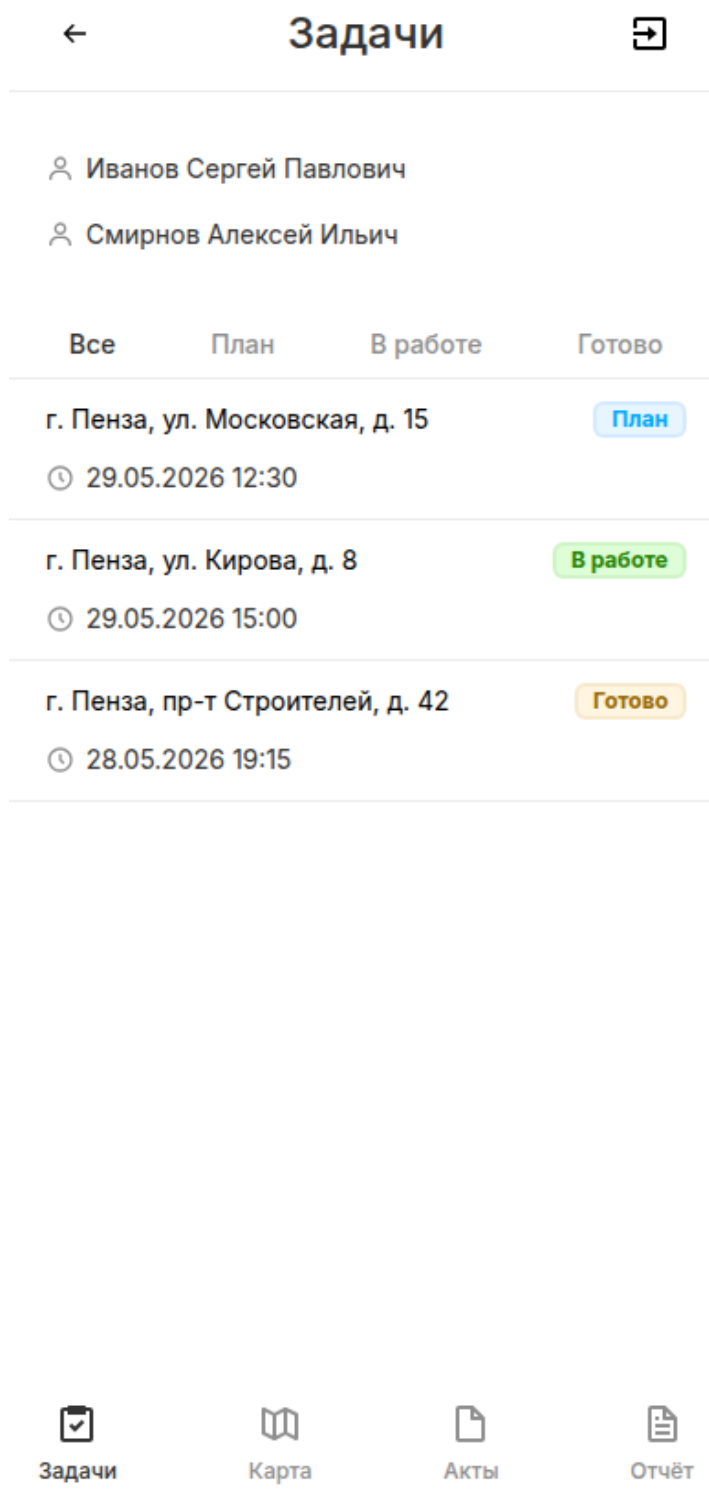


Рисунок 18 – Список задач бригады в мобильном АРМ инспектора

Карточка задачи объединяет сведения, которые нужны инспектору перед началом проверки: карту, адрес объекта, данные абонента, номер лицевого счета и комментарий диспетчера. Карта строится на основе адреса объекта и текущей геолокации устройства. Карточка задачи показана на рисунке 19.

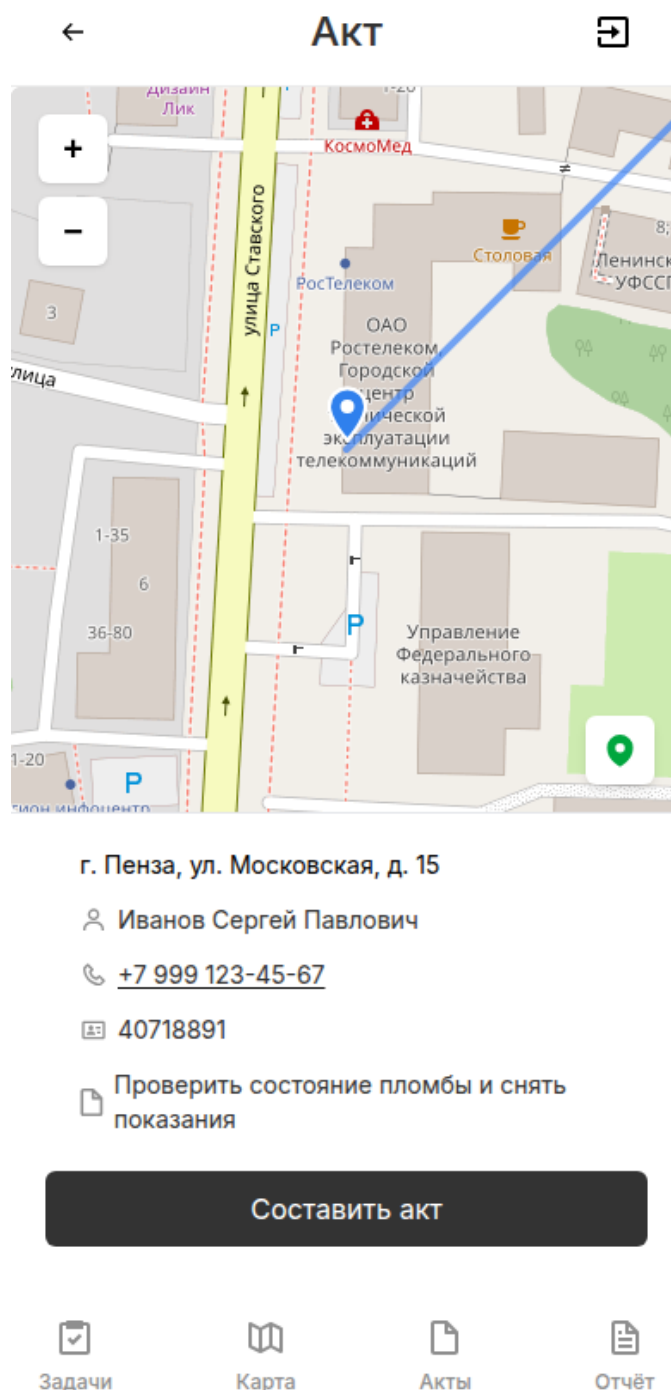


Рисунок 19 – Карточка задачи с картой объекта проверки

Из карточки инспектор переходит к составлению акта. При этом приложение переводит задачу в работу и переносит в форму уже известные сведения: идентификатор задачи, номер инспекции, лицевой счет, телефон, ФИО абонента, адрес, идентификаторы прибора учета и пломбы. Это уменьшает количество ручного ввода и снижает вероятность ошибки при оформлении результата проверки.

3.3 Реализация формы акта, фотофиксации и результатов работы

Форма акта реализована как последовательность коротких шагов. На первом шаге инспектор проверяет лицевой счет, затем вводит контактные данные, прикрепляет фотографии, выбирает тип акта и заполняет технические сведения о состоянии электроснабжения, приборе учета и пломбах. Начальный шаг формы показан на рисунке 20.

← **Акт** ☰

Лицевой счёт

40718891

Продолжить

Задачи Карта Акты Отчёт

Рисунок 20 – Начальный шаг заполнения акта в мобильном приложении

Фотофиксация выполняется с использованием возможностей устройства. Приложение позволяет получить снимок с камеры или выбрать изображение из галереи, после чего фотографии отправляются как вложения к инспекции. Это

важно для предметной области, потому что изображения прибора учета и пломбы подтверждают фактическое состояние объекта на момент проверки.

После завершения проверки созданные материалы доступны в разделе актов. В нем инспектор видит список выполненных проверок и прикрепленные файлы, что показано на рисунке 21.

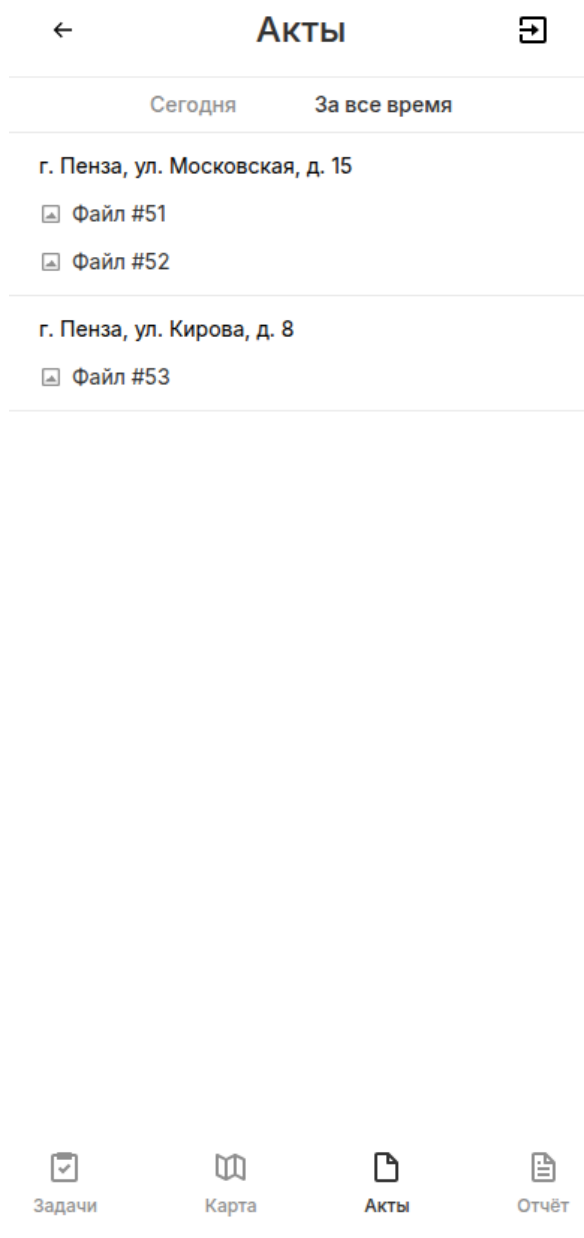


Рисунок 21 – Список актов и приложенных материалов в мобильном приложении

Отдельный экран отчетов позволяет бригаде просматривать сформированные итоговые файлы. Такой экран нужен для контроля результатов выполненной работы и доступа к отчетным материалам без перехода в

административную панель. Экран отчетов мобильного приложения представлен на рисунке 22.

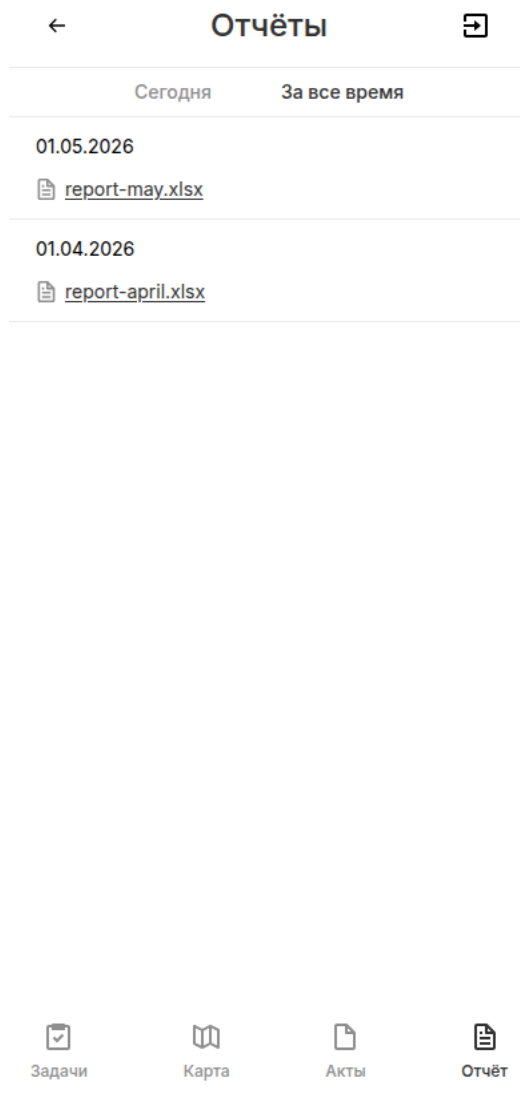


Рисунок 22 – Экран отчетов в мобильном приложении инспектора

Логика основных действий инспектора приведена в таблице 12.

Таблица 12 – Основные действия инспектора в мобильном приложении

Действие	Что делает пользователь	Результат в системе
Вход бригады	Вводит идентификатор бригады	Приложение получает состав бригады и открывает рабочие экраны
Просмотр задач	Выбирает задачу из списка или через карту	Открывается карточка конкретного объекта проверки

Продолжение таблицы 12

Действие	Что делает пользователь	Результат в системе
Начало работы	Нажимает кнопку составления акта	Задача переводится в работу, данные переносятся в форму
Заполнение акта	Вносит показания, сведения о пломбах, тип акта и результат проверки	Формируется результат инспекции
Фотофиксация	Делает фотографии прибора учета и пломбы	Файлы прикрепляются к инспекции как доказательные материалы
Просмотр итогов	Открывает разделы актов и отчетов	Проверяет созданные материалы и отчетные файлы

Формирование результата проверки в мобильном приложении имеет несколько явно различимых состояний. Сначала задача находится в плановом состоянии, затем инспектор переводит ее в работу, открывает форму результата, заполняет исходные данные, прикрепляет фотографии, выбирает тип результата и отправляет сведения на сервер. Если отправка или загрузка вложений завершается ошибкой, пользователь возвращается к заполненным данным и повторяет действие без потери уже введенной информации. Эти состояния представлены на рисунке 23.

Разделение сценария на состояния помогает снизить риск потери данных. Пользователь видит, на каком этапе находится результат проверки, а приложение может сохранять введенные сведения между шагами формы. Это особенно важно для полевой работы, где связь с сервером может быть нестабильной, а повторный сбор фотографий и показаний увеличивает время проверки.

Кроме того, состояние формы обеспечивает возврат к предыдущим шагам без повторного заполнения всего результата проверки.

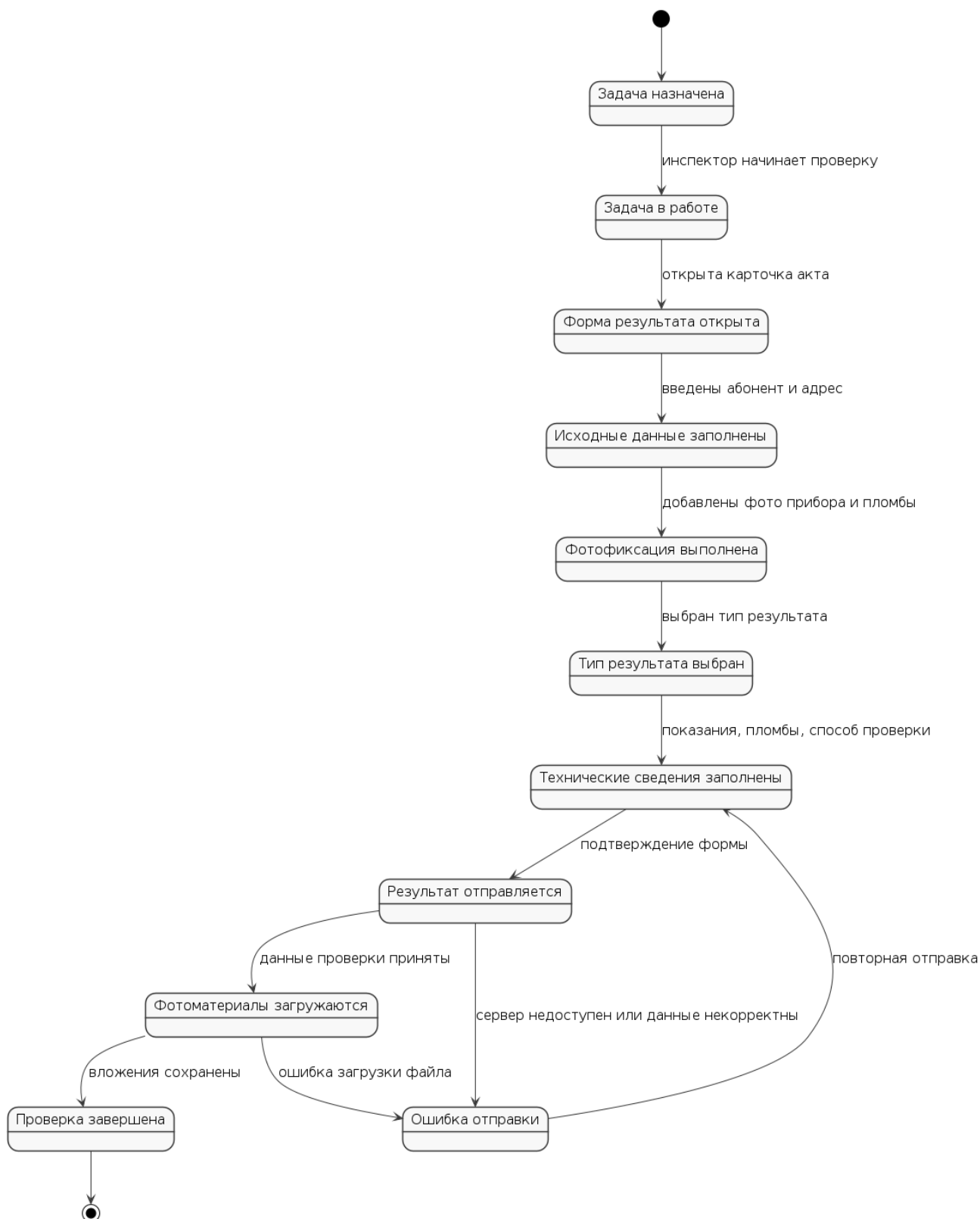


Рисунок 23 – Диаграмма состояний формирования результата проверки

Состояния на диаграмме соответствуют пользовательскому смыслу результата проверки, а не только техническим запросам. Для инспектора важны понятные этапы: задача начата, данные собраны, фото добавлены, результат

отправлен, проверка завершена или требуется повторная отправка.

Последовательность обмена данными между мобильным АРМ, серверными сервисами и административной панелью при завершении проверки показана на рисунке 24.

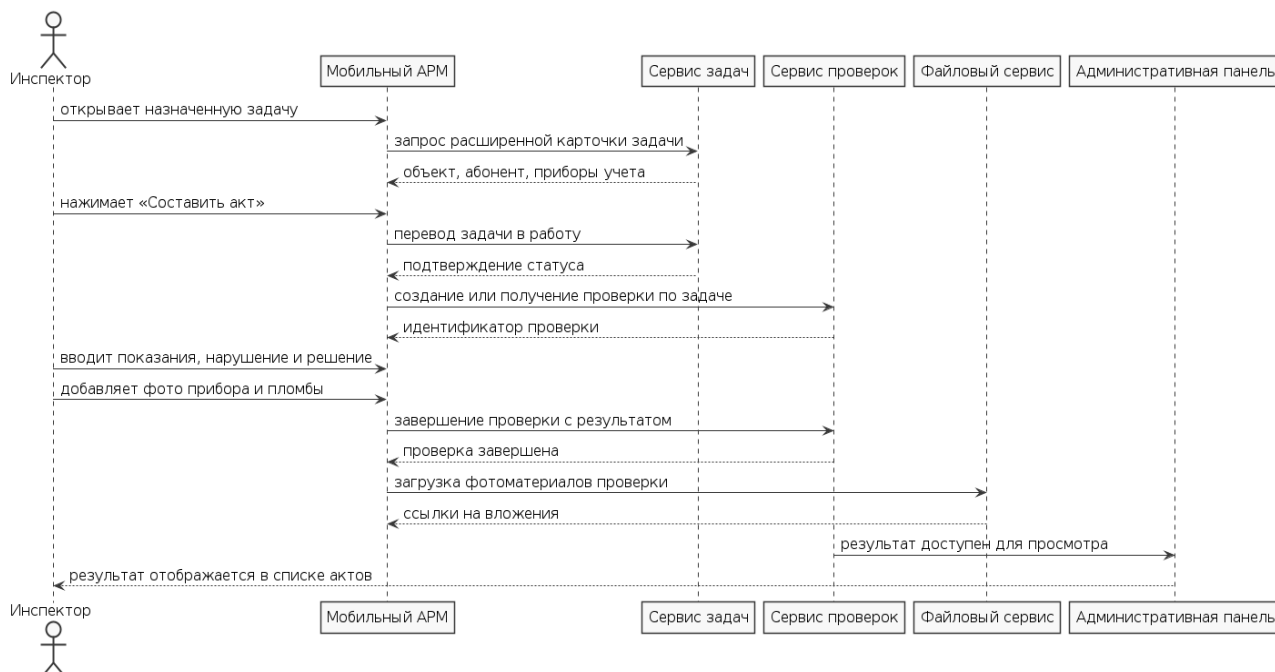


Рисунок 24 – Диаграмма последовательности формирования результата проверки

Диаграмма последовательности показывает, что итог проверки формируется не одной кнопкой, а цепочкой действий: получение карточки задачи, перевод в работу, создание или получение проверки, заполнение результата, завершение проверки и загрузка фотоматериалов. После этого диспетчер видит результат в административной панели.

3.4 Реализация административной панели диспетчера

Административная панель разрабатывалась как рабочее место для сотрудника, который планирует проверки и контролирует выполнение задач. В отличие от мобильного приложения, здесь важны табличные представления, фильтры и быстрый переход к карточкам записей. Основной раздел задач показан на рисунке 25.

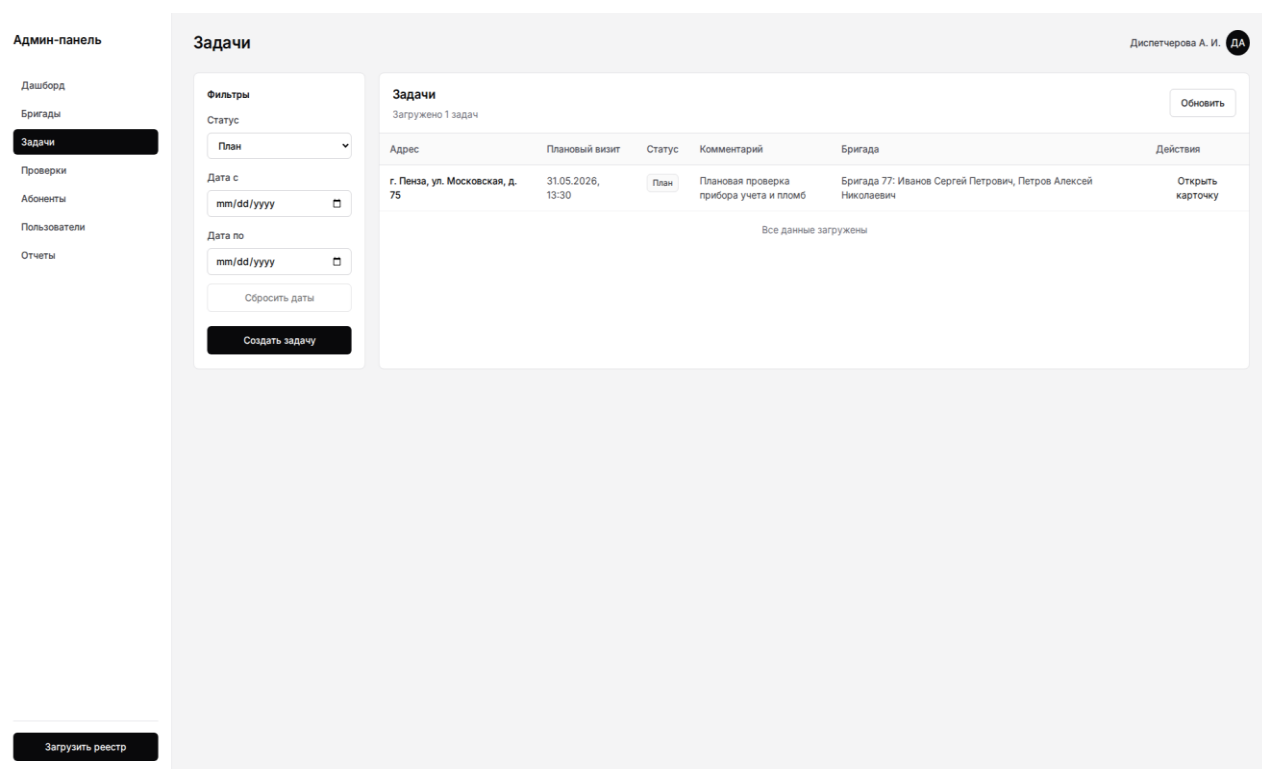


Рисунок 25 – Управление задачами в административной панели диспетчера

В разделе задач диспетчер видит адрес, плановый визит, статус, комментарий и назначенную бригаду. Через фильтры можно ограничить список по статусу и датам, а через кнопку создания добавить новую задачу. Это закрывает основной кабинетный сценарий: подготовить проверку и назначить ее исполнителям.

Раздел бригад используется для контроля состава исполнителей и их текущего состояния, а также дополняет раздел задач, так как позволяет рассматривать исполнителей не изолированно, а в контексте выполняемых ими проверок. Управление бригадами является важным бизнес-процессом в рамках АРМ диспетчера, так как в соответствии с требованиями диспетчер должен иметь возможность легко и быстро в реальном времени отредактировать состав бригады. Раздел бригад представлен на рисунке 26.

Связь задач с бригадами помогает диспетчеру учитывать текущую загрузку исполнителей при новом назначении. За счет этого диспетчер может не только выбрать исполнителя для задачи, но и оценить, насколько назначение соответствует текущему распределению работ.

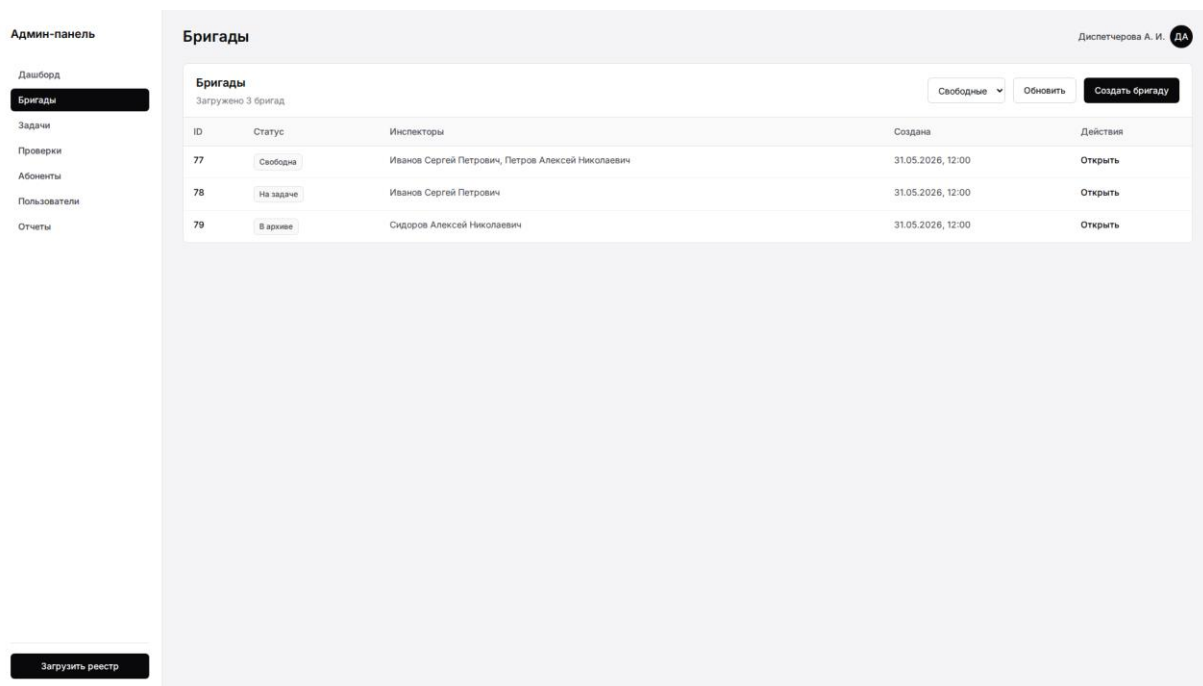


Рисунок 26 – Раздел управления бригадами в административной панели

Раздел проверок позволяет диспетчеру видеть результаты выполненных и выполняемых инспекций: номер связанной задачи, тип проверки, статус, дату и количество приложенных материалов. Экран проверок приведен на рисунке 27.

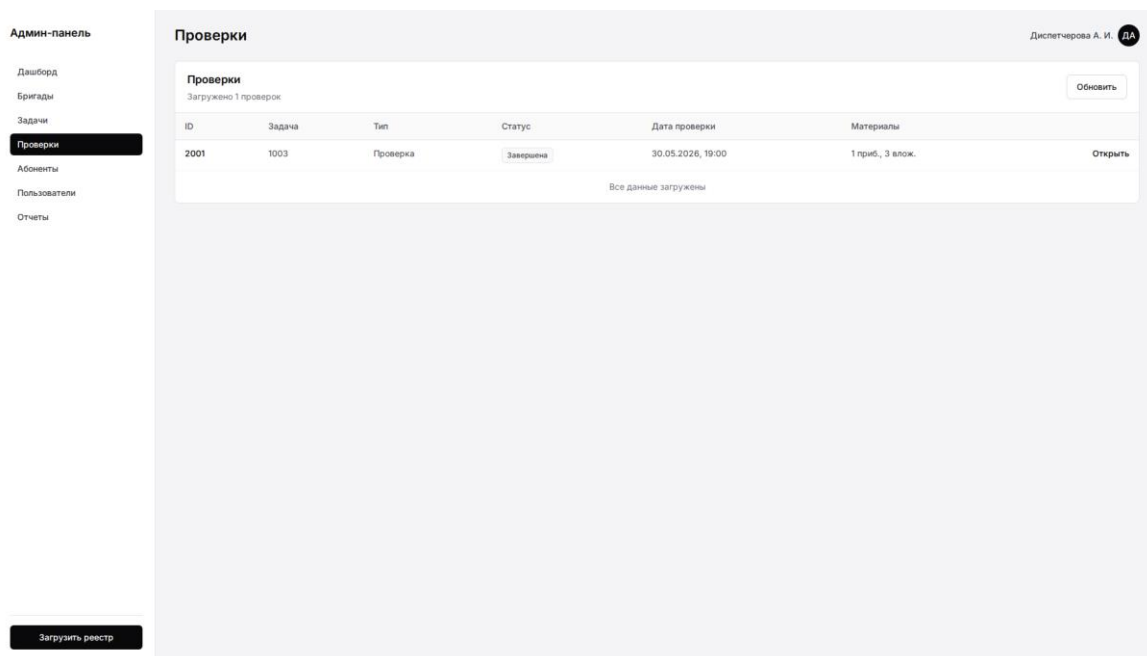


Рисунок 27 – Раздел проверок в административной панели

Раздел отчетов предназначен для просмотра сформированных отчетных файлов и создания нового отчета за выбранный период. Он показан на рисунке 28.

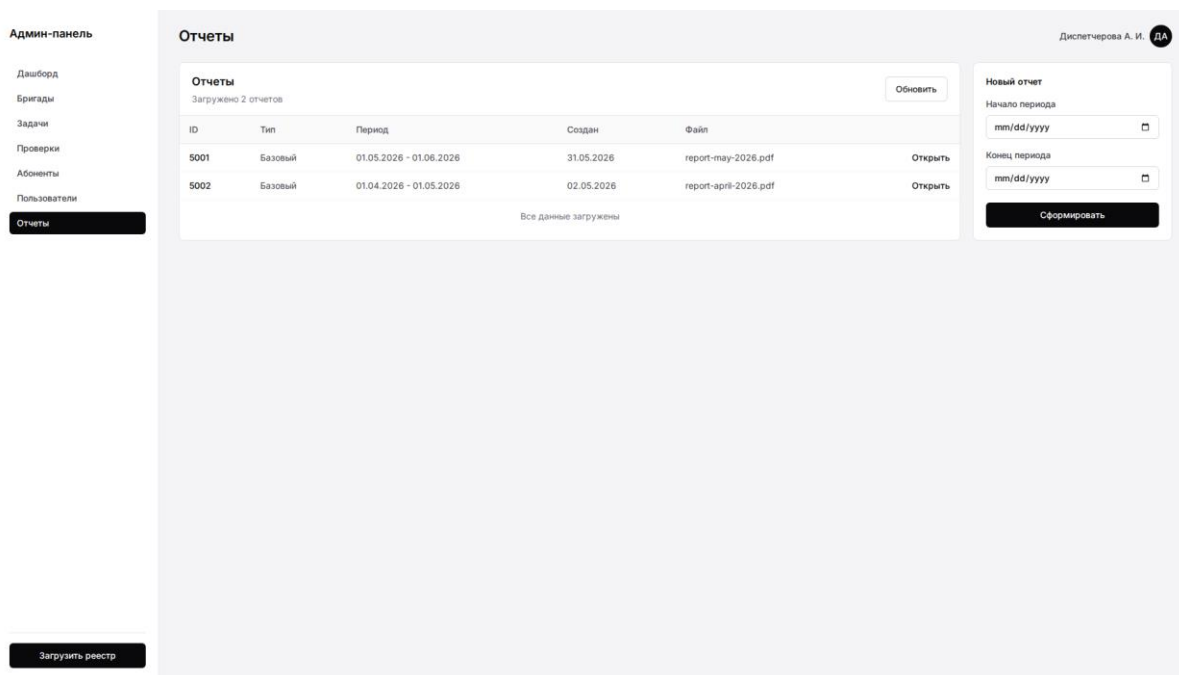


Рисунок 28 – Раздел отчетов в административной панели

Полный состав реализованных экранов административной панели приведен в таблице 13.

Таблица 13 – Основные функции административной панели

Экран или маршрут	Что реализовано	Назначение для диспетчера
/admin/login	Форма входа администратора с сохранением сеанса и защитой внутренних разделов	Ограничивает доступ к управлению задачами, бригадами и отчетами
/register	Экран завершения регистрации по приглашению	Позволяет подключать новых сотрудников через предварительно созданную ссылку
/admin	Сводный экран с карточками показателей, графиками по задачам, проверкам и бригадам, списками последних задач и инспекций	Дает диспетчеру быстрый обзор текущей загрузки и результатов работы
/admin/brigades	Список бригад, фильтр по статусу, обновление данных и переход к созданию или карточке бригады	Помогает контролировать доступные и занятые бригады

Продолжение таблицы 13

Экран или маршрут	Что реализовано	Назначение для диспетчера
/admin/brigades/create	Форма создания бригады по идентификаторам инспекторов	Позволяет подготовить состав исполнителей для назначения задач
/admin/brigades/:id	Карточка бригады со статусом, составом инспекторов, датами создания и обновления, действием архивирования	Позволяет проверить состав бригады и вывести ее из активной работы
/admin/tasks	Таблица задач, фильтры по статусу и датам, бесконечная подгрузка, обновление, открытие модального окна карточки задачи	Используется для ежедневного планирования и контроля выездов
/admin/tasks/create	Форма выбора объекта, бригады, планового времени визита и комментария	Создает новую задачу проверки для передачи в мобильное приложение инспектора
/admin/inspections	Список проверок с номером задачи, типом, статусом, датой и количеством материалов	Позволяет контролировать ход и результаты инспекций
/admin/inspections/:id	Карточка проверки с причиной, решением, признаками нарушений, приборами, вложениями и описанием нарушения	Дает возможность проверить содержание выполненного акта и доказательные материалы
/admin/subscribers	Поиск абонентов по ФИО, лицевому счету, телефону или адресу, таблица результатов и подгрузка списка	Используется для уточнения данных потребителя перед созданием задачи
/admin/subscribers/:id	Карточка абонента с редактированием данных абонента, договоров и объектов, а также картой объекта	Позволяет поддерживать актуальность адресной и контактной информации
/admin/users	Список пользователей, обновление, модальное окно создания приглашения и ссылка для регистрации	Поддерживает управление сотрудниками, которым нужен доступ к административной панели

Продолжение таблицы 13

Экран или маршрут	Что реализовано	Назначение для диспетчера
/admin/reports	Список отчетов, файл отчета, форма выбора периода и запуск формирования нового отчета	Используется для получения итоговых материалов по выполненным работам

Такое перечисление показывает, что административная панель реализована не только как список задач, а как полный рабочий контур диспетчера: от входа и управления пользователями до подготовки задач, контроля инспекций, ведения абонентских данных и формирования отчетов.

Помимо основных списков задач, бригад, проверок и отчетов, в административной панели разработаны дополнительные экраны, которые закрывают полный цикл работы диспетчера и администратора. На рисунке 29 приведен сводный экран административной панели. Отдельно показан раздел управления пользователями, который используется для администрирования доступа.

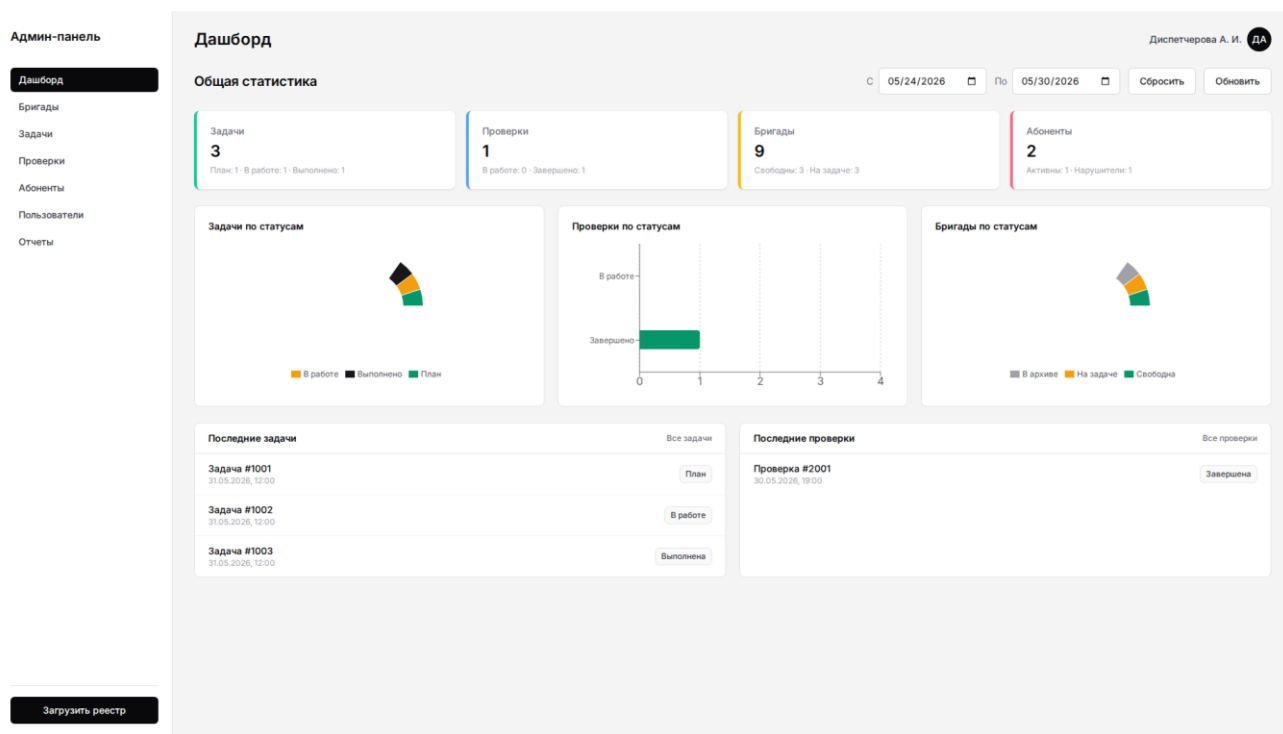


Рисунок 29 – Сводный экран административной панели

Сводный экран дает диспетчеру общий контекст работы отдела: количество задач, проверок и активных бригад видно до перехода к детальным разделам. Он используется как краткая точка контроля перед переходом к подробным спискам задач, проверок и бригад, за счет чего сводный экран снижает количество лишних переходов при первичной оценке состояния отдела.

Раздел управления пользователями связан с безопасностью административного контура. Доступ к созданию задач, изменению бригад и просмотру результатов проверок должен выдаваться контролируемо, поэтому приглашение нового сотрудника оформляется через роль и одноразовую ссылку регистрации. Раздел управления пользователями представлен на рисунке 30.

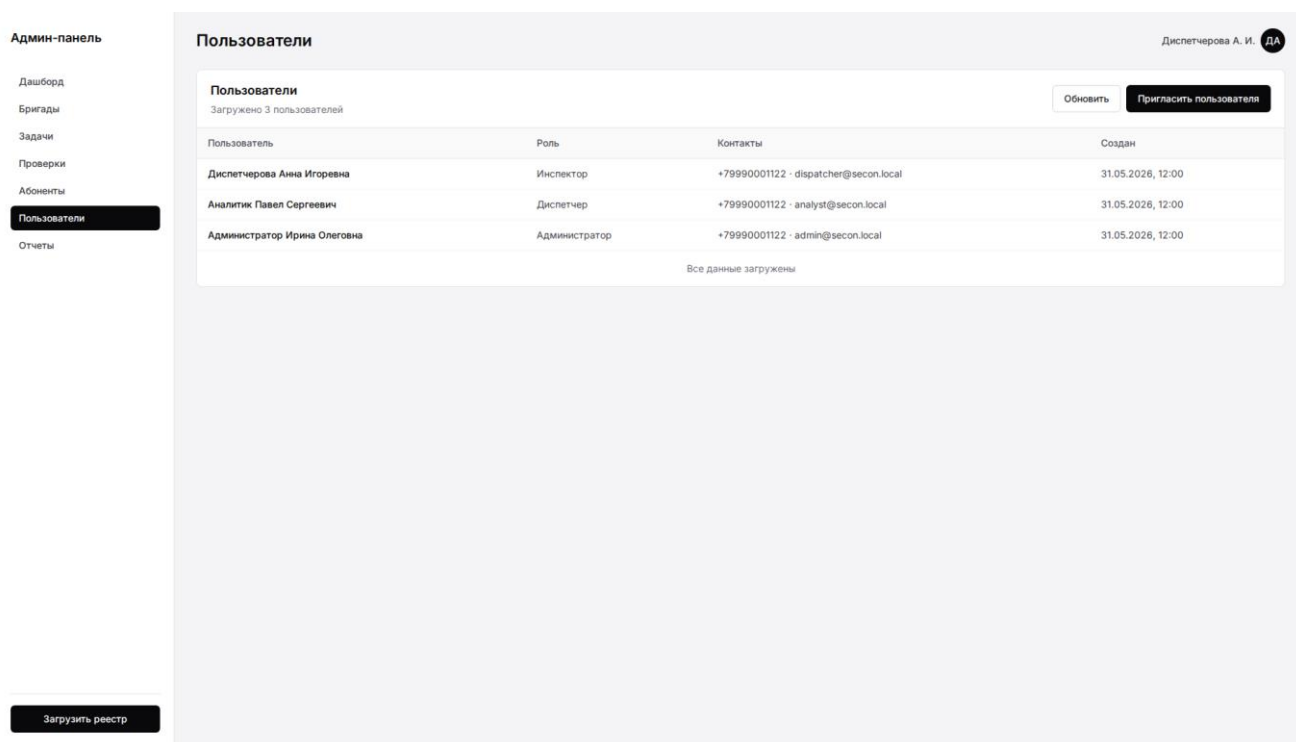


Рисунок 30 – Раздел управления пользователями административной панели

Такой подход отделяет рабочие операции диспетчера от администрирования учетных записей. В результате пользовательский доступ можно развивать независимо от экранов задач и отчетов, не изменяя основной сценарий планирования проверок.

Сводный экран используется для быстрого контроля числа задач, проверок и бригад, а раздел пользователей нужен для выдачи доступа сотрудникам. В нем

администратор видит роли пользователей и может сформировать ссылку приглашения.

Доступ к административной панели начинается с авторизации сотрудника. Пользователь должен ввести свой телефон или адрес электронной почты и пароль. Экран входа показан на рисунке 31.

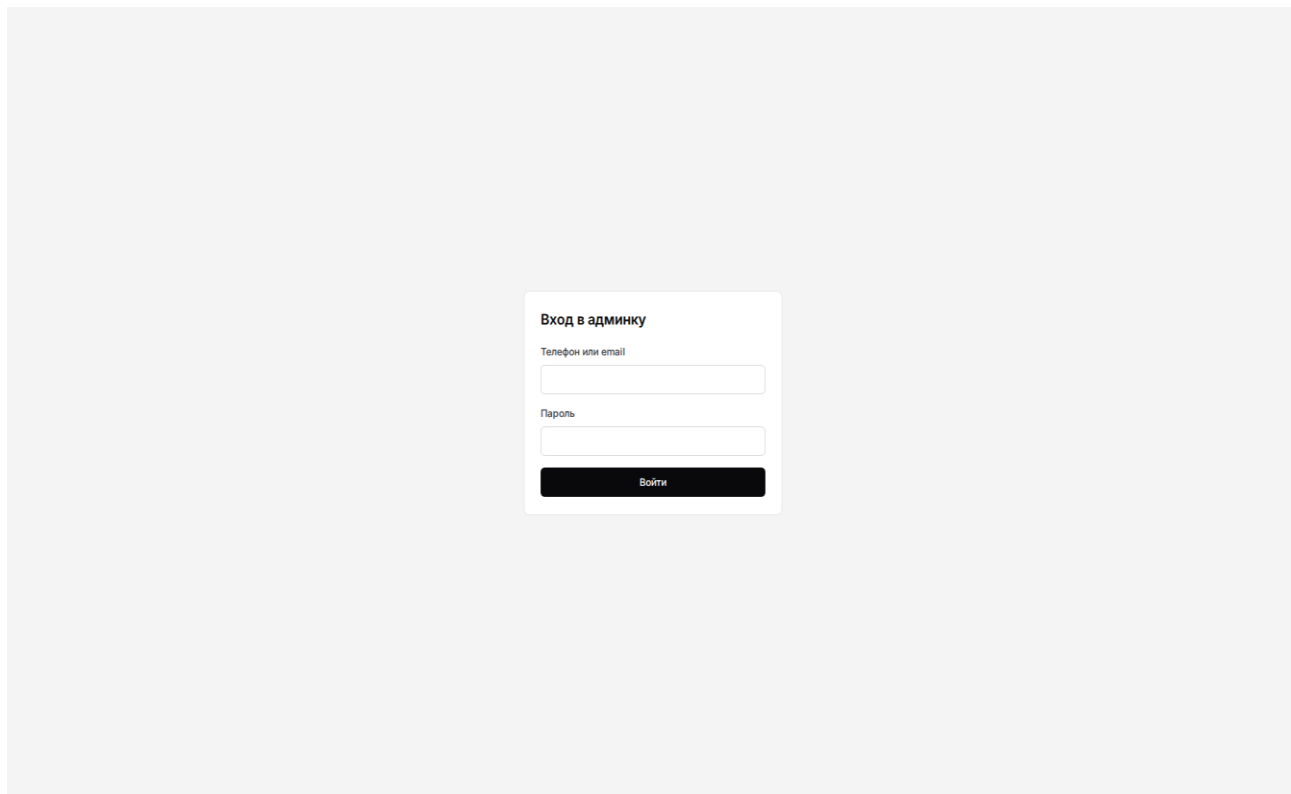


Рисунок 31 – Экран входа в административную панель

После входа администратор может открыть раздел пользователей и сформировать приглашение для нового сотрудника. В приглашении выбирается роль: инспектор, диспетчер или администратор. Сервер возвращает одноразовую ссылку, которую можно передать сотруднику для завершения регистрации. Одноразовая ссылка позволяет обеспечить возможность безопасно реализовать процесс регистрации новых пользователей. Новый сотрудник может зарегистрироваться только по ссылке-приглашению. Полученную одноразовую ссылку можно удобно скопировать в буфер обмена по нажатию на кнопку «Скопировать ссылку» для дальнейшей передачи приглашаемому пользователю. Следует учитывать, что у каждого пользователя может быть активна только одна ссылка-приглашение. Процесс ее создания показан на рисунке 32.

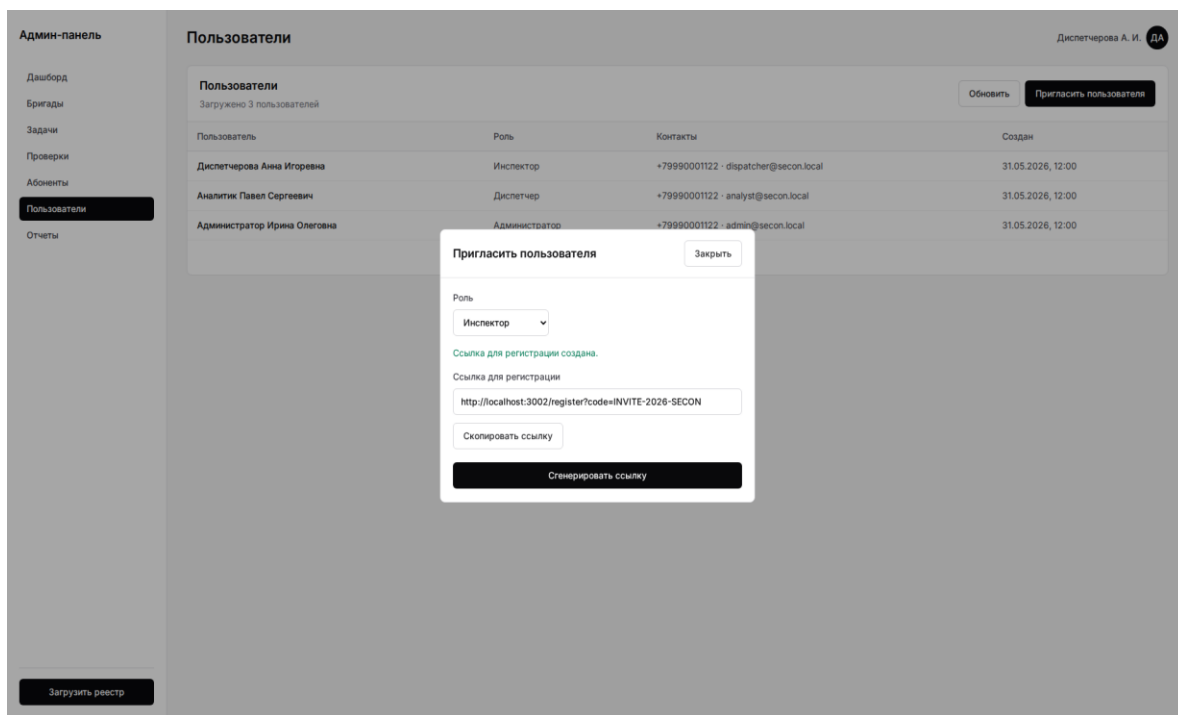


Рисунок 32 – Формирование ссылки приглашения пользователя

Новый сотрудник открывает полученную ссылку, заполняет фамилию, имя, отчество, электронную почту, телефон и пароль. Экран завершения регистрации по приглашению представлен на рисунке 33.

Рисунок 33 – Регистрация пользователя по приглашению

Таким образом, пользовательский доступ в административной панели

строится не через свободную регистрацию, а через контролируемое приглашение. Это соответствует предметной области: права на управление задачами, бригадами и проверками должны выдавать только сотрудники с административной ролью.

Работа с абонентской базой вынесена в отдельный раздел, так как задача диспетчера часто начинается не с бригады, а с поиска потребителя или объекта электроснабжения. На рисунке 34 приведен список абонентов. Этот экран используется как начальная точка поиска потребителя перед созданием или уточнением задачи на проверку.

Такое разделение позволяет проверить исходные сведения об объекте до создания задачи и уменьшить риск ошибочного назначения.

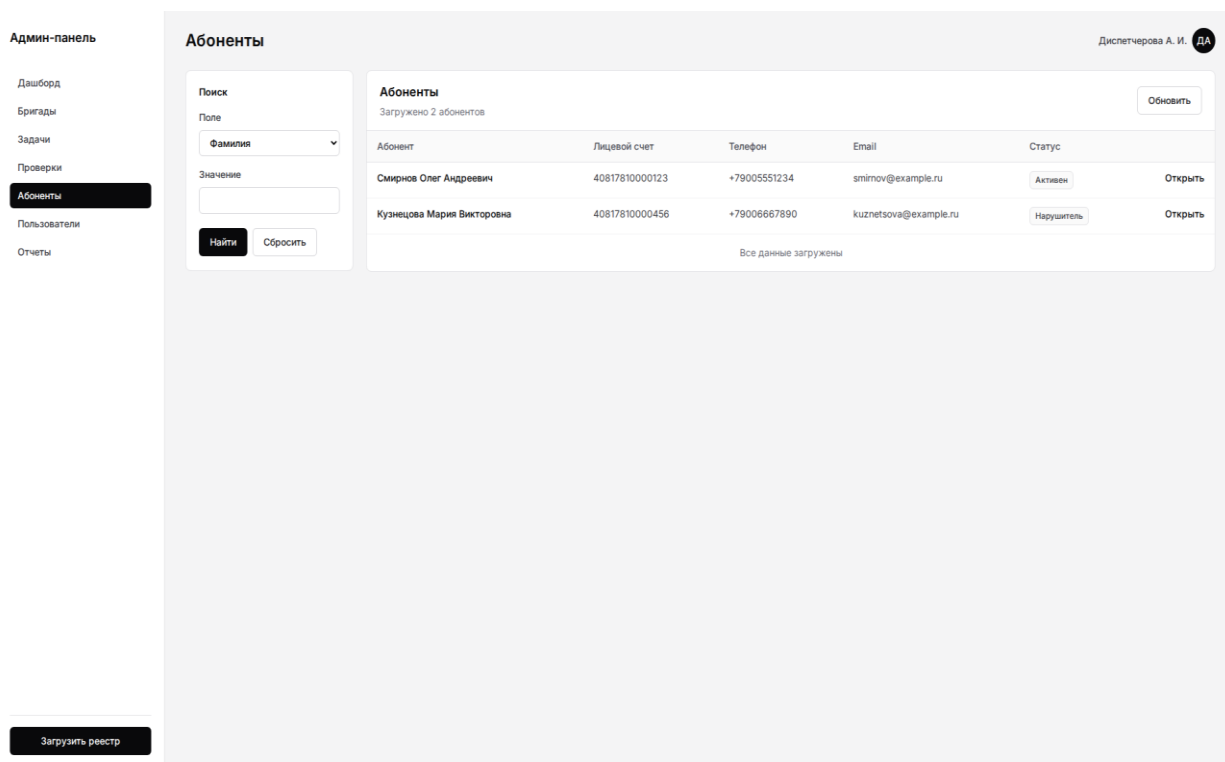


Рисунок 34 – Раздел поиска и просмотра абонентов

Карточка абонента объединяет персональные данные, договоры, объекты электроснабжения и карту, которые диспетчер при необходимости может отредактировать.

Карточка абонента представлен на рисунке 35.

Админ-панель

Дашборд
Бригады
Задачи
Проверки
Абоненты
Пользователи
Отчеты

Абонент Смирнов Олег Андреевич

Диспетчерова А. И. **ДА**

Назад к списку

Абонент: Смирнов Олег Андреевич
Лицевой счет: 40817810000123
Статус: Активен
Телефон: +79005551234
Email: smirnov@example.ru
Договоры: 2
Объекты: 2

Данные абонента

Сохранить

Фамилия: Смирнов
Имя: Олег
Отчество: Андреевич
Лицевой счет: 40817810000123
Телефон: +79005551234
Email: smirnov@example.ru
ИНН: 583601234567
Дата рождения: 03/12/1984
Статус: Активен

Договоры

Номер: ЛС-40817810000123
Дата подписания: 09/01/2024
Сохранить
Объект: г. Пенза, ул. Московс...

Объекты

Адрес: г. Пенза, ул. Московская, д. 75
Сохранить
☒ Есть автомат
Приборов: 1
Адрес: г. Пенза, ул. Пушкина, д. 12
Сохранить
☐ Есть автомат
Приборов: 0

Загрузить реестр

Рисунок 35 – Карточка абонента с объектом электроснабжения и картой

На рисунке 36 показана форма создания задачи. Диспетчер при создании задачи заполняет адрес, плановую дату визита, назначает бригаду и при необходимости может указать комментарий к задаче.

Админ-панель

Дашборд
Бригады
Задачи
Проверки
Абоненты
Пользователи
Отчеты

Новая задача

Диспетчерова А. И. **ДА**

Адрес: Начните вводить адрес

Плановый визит: mm/dd/yyyy, --:--:--

Бригада: Назначить позже

Комментарий

Отмена Создать

Загрузить реестр

Рисунок 36 – Форма создания задачи в административной панели

После заполнения формы задача связывается с объектом электроснабжения и бригадой, поэтому диспетчер может перейти к проверке

назначенного исполнителя. На рисунке 37 представлена карточка бригады, которая содержит информацию о ее статусе и инспекторах, которые в нее входят.

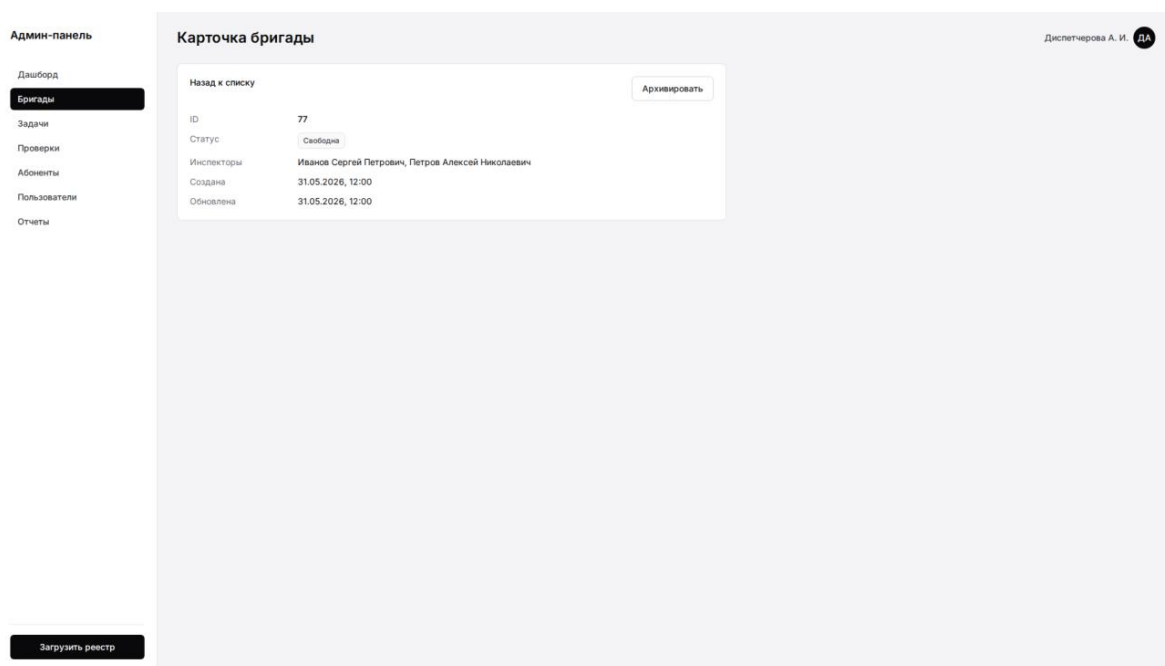


Рисунок 37 – Карточка бригады в административной панели

На рисунке 38 представлена карточка выполненной проверки с приложенными фотоматериалами

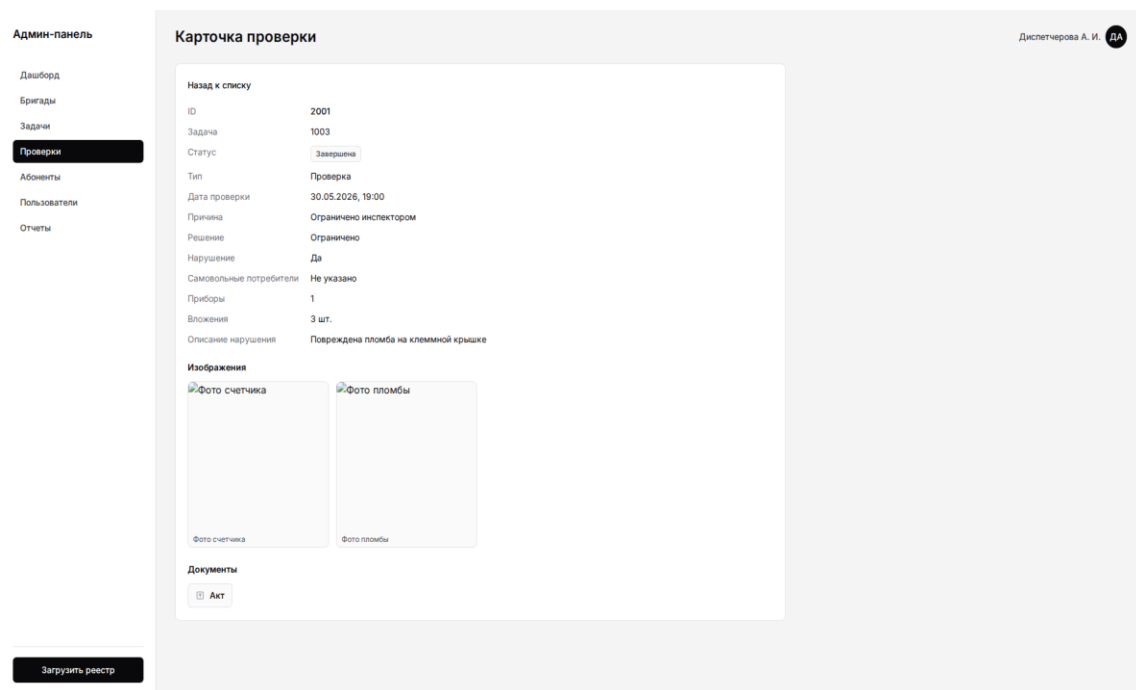


Рисунок 38 – Карточка выполненной проверки с фотоматериалами

Таким образом, эти экраны показывают завершенность административной

части: диспетчер может создать новую задачу по объекту, проверить состав бригады, открыть результат инспекции и увидеть вложенные фотографии прибора учета и пломбы.

3.5 Автоматическая генерация API-клиента и обмен данными

Для связи с серверной информационной системой был реализован общий пакет `packages/shared`. В нем находится базовая настройка запросов и файл, автоматически создаваемый по OpenAPI-описанию серверного API. Генерация запускается командой `pnpm generate-api`, а в конфигурации указаны адрес OpenAPI-схемы, базовый файл API, выходной файл и необходимость создавать функции для запросов и изменений.

После генерации приложения получают типизированные операции для задач, бригад, инспекций, отчетов, абонентов, объектов и файлов. На страницах эти операции используются не напрямую, а через локальные предметные модули приложений. Такой слой делает код ближе к языку предметной области: экран задач работает с задачей, бригадой и объектом, а не с техническим именем серверного пути. Процесс обмена данными в приложении представлен в таблице 14.

Таблица 14 – Организация обмена данными с серверной системой

Этап	Что происходит	Зачем это нужно
Получение OpenAPI-схемы	Конфигурация генератора указывает адрес серверного описания API	Клиентская часть использует актуальный контракт серверной системы
Генерация клиентского кода	Создаются типы данных и функции запросов для чтения и изменения сущностей	Снижается объем ручного кода и риск ошибки в адресах или параметрах
Предметная адаптация	В каждом приложении создаются модули для задач, бригад, проверок и других сущностей	Страницы получают данные в понятной для интерфейса форме
Использование на экранах	Компоненты вызывают функции загрузки, создания, изменения и удаления данных	Пользовательские действия сразу отражаются в серверной информационной системе

Такой подход особенно полезен при развитии системы. Если серверный контракт меняется, клиентский код можно обновить через повторную генерацию, а затем точно адаптировать предметные модули и экраны. Это делает сопровождение двух клиентских рабочих мест более предсказуемым.

3.6 Сборка PWA и Android-приложения через Capacitor

Мобильное приложение инспектора разрабатывается как веб-приложение, но поставляется двумя способами: как статическая веб-сборка и как Android-приложение. Сначала Rsbuild собирает React-код в каталог build [16]. Именно этот каталог указан в `capacitor.config.ts` как `webDir`, поэтому Capacitor знает, какие файлы нужно перенести в Android-проект.

После веб-сборки команда `cap sync android` копирует содержимое build в Android-проект и обновляет нативные зависимости. Далее Android-проект собирается через Gradle. В конфигурации Android задан пакет `ru.secon.hack`, а в `AndroidManifest.xml` указаны разрешения на интернет, точную и приблизительную геолокацию, а также чтение и запись внешнего хранилища. Эти разрешения нужны для карты, определения местоположения и работы с фотографиями.

Процесс сборки Android-приложения приведен в таблице 15.

Таблица 15 – Этапы сборки мобильного приложения для Android

Этап сборки	Команда или файл проекта	Результат
Сборка веб-части	<code>pnpm build:inspector</code> или <code>rsbuild build</code>	Создается каталог <code>apps/inspector/build</code> со статическими файлами приложения
Синхронизация Capacitor	<code>pnpm android:sync</code> или <code>cap sync android</code>	Собранные веб-файлы переносятся в Android-проект, обновляются зависимости Capacitor
Нативная сборка	<code>apps/inspector/android/gradlew assemble</code>	Gradle формирует APK-файл Android-приложения

Продолжение таблицы 15

Этап сборки	Команда или файл проекта	Результат
Автоматическая сборка	<code>.github/workflows/build-android.yaml</code>	На ветке <code>mobile</code> устанавливаются зависимости, выполняется <code>build:android</code> , собирается APK и публикуется выпуск с файлом приложения
PWA-поставка	<code>public/manifest.json</code> и статическая веб-сборка	Приложение может запускаться как отдельное веб-приложение в режиме отдельного окна

Последовательность сборки Android-приложения через Rsbuild, Capacitor и Gradle представлена на рисунке 39.

Выбранная схема сборки позволяет использовать единый исходный код мобильного интерфейса для веб-режима и Android-поставки. Это уменьшает стоимость сопровождения: изменения в форме акта, списке задач или карте вносятся в React-приложение один раз, после чего результат может быть собран как статический веб-артефакт и синхронизирован с Android-проектом.

Отдельный этап синхронизации через Capacitor нужен для переноса собранных файлов в нативный контейнер и обновления платформенных зависимостей. После этого Gradle формирует APK, пригодный для установки на мобильное устройство инспектора. Таким образом использование Capacitor и технологий PWA позволяет из веб-приложения получить нативное мобильное Android приложение, что удовлетворяет нефункциональные требования реализованной системы.

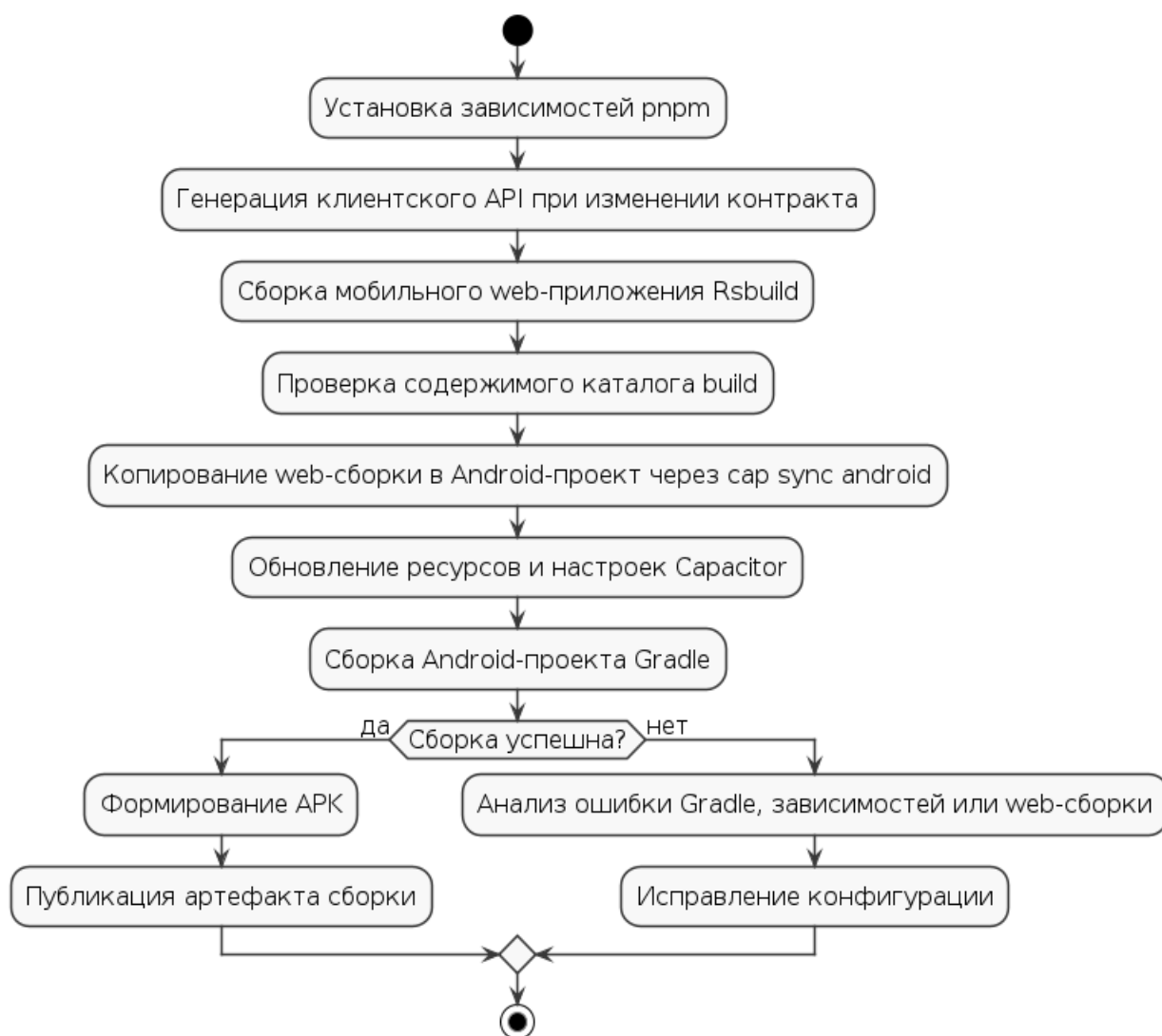


Рисунок 39 – Диаграмма активности сборки Android-приложения

Диаграмма активности показывает, что Android-приложение не пишется отдельно от веб-интерфейса. Сначала собирается мобильное React-приложение, затем Capacitor переносит полученные файлы в Android-проект, после чего Gradle формирует APK-артефакт. Автоматический конвейер сборки настроен через GitHub Actions. Согласно документации GitHub Actions, сценарий сборки описывается YAML-файлом, запускается по событию репозитория, состоит из заданий, выполняемых на выбранном исполнителе, и набора последовательных шагов [17]. В проекте этот подход используется для двух задач: сборки Android-приложения и поставки веб-версии через Docker.

Файл `.github/workflows/build-android.yaml` запускается при отправке изменений в ветку `mobile`. В задании используется виртуальный

исполнитель Ubuntu, затем выполняются получение исходного кода, настройка JDK 21, настройка Gradle, установка Node.js 20, включение `pnpm` через `Corespack`, установка зависимостей с проверкой файла фиксации зависимостей, команда `pnpm build:android` и сборка APK через Gradle. После сборки APK переименовывается с контрольной суммой, создается GitHub Release и в него загружается готовый файл приложения [17].

Файл `.github/workflows/deploy-docker.yaml` запускается при отправке изменений в ветку `master` на собственном исполнителе. Он собирает Docker-образ мобильного веб-приложения, останавливает старые контейнеры того же приложения, запускает новый контейнер с тегом текущего коммита и очищает устаревшие образы. Такой конвейер отделяет ручную разработку от воспроизводимой поставки: результат сборки можно связать с конкретным коммитом и повторить при необходимости.

PWA-часть не заменяет Android-сборку, а дополняет ее. Manifest-файл задает стартовый адрес, режим отдельного окна, цветовую схему и набор иконок. Это позволяет использовать мобильный интерфейс как установленное веб-приложение, если пользователю не требуется нативная упаковка. Android-сборка через `Capacitor` нужна тогда, когда приложение должно распространяться как APK и стабильно обращаться к возможностям устройства.

Итоговая схема поставки клиентского приложения включает несколько вариантов: административная панель собирается и размещается как веб-интерфейс, мобильный АРМ инспектора может быть размещен как устанавливаемое веб-приложение и дополнительно упакован в Android-приложение. Благодаря этому разработанная система подходит как для офисной работы диспетчера, так и для выездной работы инспектора на мобильном устройстве.

В результате веб-версия подходит для централизованного обновления, а Android-сборка – для закрепленных мобильных устройств инспекторов. Такое разделение вариантов поставки позволяет учитывать разные условия эксплуатации системы.

4 Контроль качества программного обеспечения

Контроль качества клиентских приложений выполнен для подтверждения того, что разработанные средства диспетчера и инспектора соответствуют требованиям, сформулированным в разделе 1, и проектным решениям, приведенным в разделах 2 и 3. Проверка охватывала корректность пользовательских сценариев, устойчивость преобразования данных, взаимодействие с серверным API, сборку веб-приложений и пригодность интерфейсов к работе на целевых экранах.

Особенность проекта состоит в наличии двух клиентских контуров: административной панели диспетчера и мобильного АРМ инспектора. Поэтому контроль качества был организован как сочетание автоматизированных проверок доменной логики и ручной проверки пользовательских сценариев. Такой подход позволяет отделить ошибки бизнес-правил и преобразования данных от дефектов интерфейса, которые проявляются только при работе с реальными формами, таблицами, картой и фотографиями.

4.1 Обоснование подхода к тестированию

Для выбора методов проверки были выделены основные риски клиентской части. Они связаны не только с ошибками отображения, но и с потерей данных при заполнении акта, неверным назначением задач, некорректной интерпретацией статусов и зависимостью от сервисов геокодирования и построения маршрута. Риски и способы их контроля представлены в таблице 16.

Таблица 16 – Риски клиентских приложений и способы проверки

Риск	Возможное последствие	Способ проверки
Ошибка авторизации диспетчера или бригады	Пользователь не получает доступ к своим функциям либо открывает чужой сценарий работы	Функциональная проверка входа и проверка helper-функций сессии
Некорректная загрузка задач, бригад, абонентов, проверок и отчетов	Диспетчер принимает решение по некорректным данным	Модульные тесты преобразования DTO, ручная проверка списков и карточек

Продолжение таблицы 16

Риск	Возможное последствие	Способ проверки
Неверное отображение статусов задач и проверок	Задача может быть ошибочно назначена, удалена или считаться выполненной	Тесты правил статусов и проверка таблиц административной панели
Сбой сценария создания задачи диспетчером	Бригада не получает плановую работу или получает задачу с неверными реквизитами	Функциональный сценарий создания задачи и проверка формируемого запроса
Потеря данных при заполнении акта инспектором	Материалы проверки становятся неполными и требуют повторного выезда	Проверка последовательности шагов формы и значений по умолчанию
Ошибка загрузки фотографий счетчика и пломбы	Фотоматериалы не прикрепляются к акту и не подтверждают результат проверки	Ручная проверка шага фотофиксации и интеграционного сценария завершения инспекции
Проблемы адаптации мобильного интерфейса	Инспектор не может использовать приложение на экране смартфона в полевых условиях	Ручная проверка мобильного viewport, навигации и отсутствия наложений элементов

На основании указанных рисков в проекте применялись несколько уровней контроля. Их назначение приведено в таблице 17.

Таблица 17 – Виды тестирования, примененные в проекте

Вид проверки	Назначение	Объекты проверки
Интеграционное тестирование	Проверка взаимодействия клиентского слоя с внешним API или RTK Query сервисом	Геокодирование, маршрутизация, поиск инспекции по задаче
Функциональное тестирование	Проверка пользовательских сценариев от входа до получения результата	Создание задачи, просмотр карточек, заполнение акта, просмотр отчетов
Ручная UI-проверка	Оценка читаемости, адаптивности и доступности основных действий	Desktop-интерфейс диспетчера и мобильный интерфейс инспектора
Контрольная сборка	Подтверждение, что исходный код собирается в поставляемые веб-артефакты	Rsbuild-сборка административной панели и мобильного приложения

Таким образом, выбранная схема контроля качества направлена на проверку наиболее рискованных участков приложения: доменной логики, API-интеграций и интерфейсных сценариев, которые непосредственно влияют на работу диспетчера и инспектора.

4.2 Функциональное тестирование пользовательских сценариев

Функциональное тестирование выполнялось по основным сценариям, которые соответствуют вариантам использования, описанным в разделе 2. Для административной панели проверялись сценарии диспетчера, связанные с задачами, бригадами, абонентами, проверками, пользователями и отчетами. Для мобильного АРМ инспектора проверялись вход бригады, работа со списком задач, открытие карточки задачи и заполнение акта. Сводные тестовые сценарии приведены в таблице 18.

Таблица 18 – Функциональные тестовые сценарии клиентских приложений

№	Сценарий	Предусловие	Тестовые действия	Ожидаемый результат	Фактический результат
1	Вход диспетчера в панель администратора	Пользователь зарегистрирован и имеет роль диспетчера	Открыть экран входа, ввести учетные данные, подтвердить вход	Открывается защищенная часть административной панели	Рисунок 40
2	Просмотр дашборда диспетчера	Диспетчер авторизован	Перейти на главную страницу административной панели	Отображаются сводные показатели, последние задачи и проверки	Рисунок 41
3	Просмотр и фильтрация задач	В системе имеются задачи разных статусов	Открыть раздел задач, переключить фильтр статуса и периода	Список задач обновляется в соответствии с выбранными условиями	Рисунок 42
4	Создание задачи и назначение бригады	В системе есть абонентский объект и доступная бригада	Открыть форму создания задачи, выбрать объект, бригаду и плановую дату	Формируется запрос создания задачи с идентификаторами объекта и бригады	Рисунок 43

Продолжение таблицы 18

№	Сценарий	Предусловие	Тестовые действия	Ожидаемый результат	Фактический результат
5	Просмотр списка бригад и карточки бригады	В системе зарегистрированы бригады	Открыть раздел бригад и перейти в карточку выбранной бригады	Отображаются состав бригады, контакты и связанные задачи	Рисунок 44
6	Просмотр абонентов и карточки абонента	В системе есть абоненты и объекты электроснабжения	Выполнить поиск абонента и открыть его карточку	Показываются реквизиты абонента, договор, объект и карта	Рисунок 45
7	Просмотр выполненной проверки	По задаче сформирована инспекция с вложениями	Открыть раздел проверок и карточку выполненной проверки	Показываются параметры проверки и фотоматериалы	Рисунок 46
8	Формирование ссылки приглашения пользователя	Диспетчер имеет право управлять пользователями	Открыть раздел пользователей и создать приглашение	Система отображает ссылку регистрации пользователя	Рисунок 47
9	Вход бригады в мобильное приложение	Идентификатор бригады существует в backend	Ввести идентификатор бригады на экране входа	Сохраняются данные бригады, открывается список задач	Рисунок 48
10	Работа инспектора с задачей и актом	У бригады есть плановая задача	Открыть задачу, перевести ее в работу, пройти шаги акта и приложить фотографии	Создается результат проверки, задача переходит к завершению сценария	Рисунок 49
11	Просмотр актов и отчетов инспектором	Для бригады созданы материалы проверок	Открыть разделы актов и отчетов в нижней навигации	Пользователь видит сформированные материалы и может открыть карточку	Рисунок 50

Результаты функциональной проверки показывают, что ключевые действия пользователя покрывают как диспетчерский контур, так и мобильный контур инспектора. Проверка сценария с фотографиями зависит от доступности backend-сервиса загрузки файлов, поэтому данный риск дополнительно

учитывается при интеграционных испытаниях.

Ниже приведены экраны, полученные после выполнения функциональных тестовых сценариев. Каждый экран соответствует отдельному сценарию из таблицы 18 и фиксирует состояние приложения после проверяемого действия.

На рисунке 40 представлен результат теста входа диспетчера в административную панель.

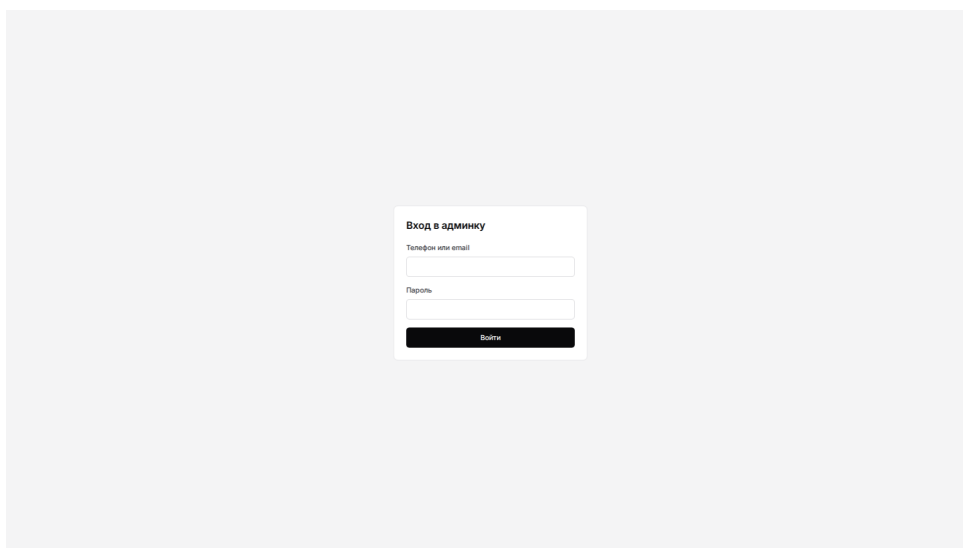


Рисунок 40 – Результат теста входа диспетчера в административную панель

После входа проверялось отображение начального рабочего состояния диспетчера. На рисунке 41 представлен тест просмотр дашборда диспетчера.

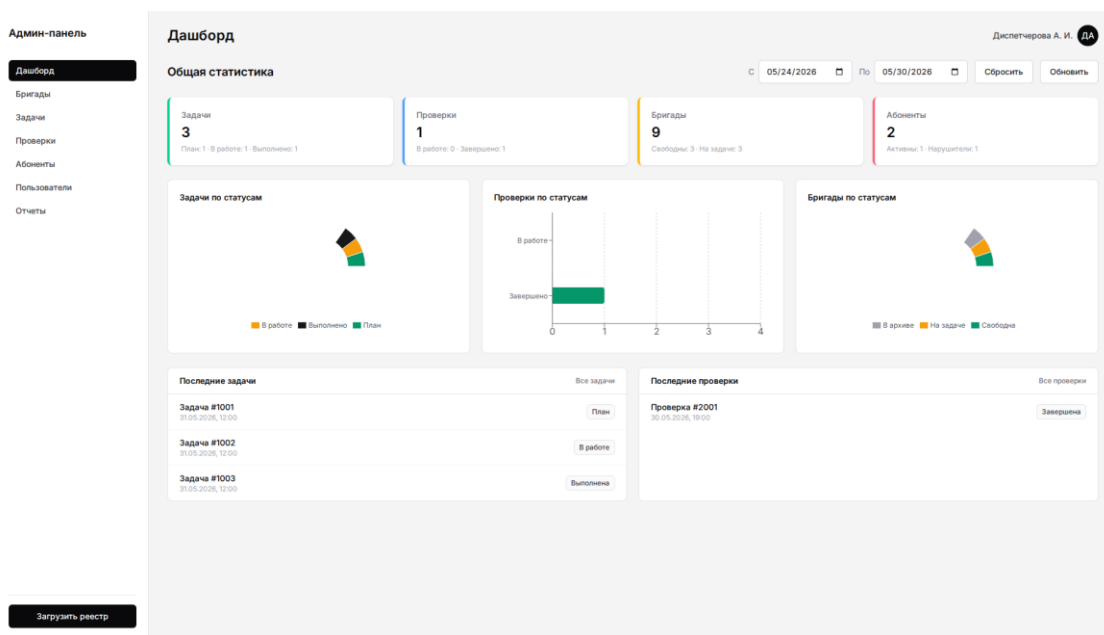


Рисунок 41 – Результат теста просмотра дашборда диспетчера

Далее проверялся переход от сводного состояния к списочным данным административной панели. На рисунке 42 представлен результат теста просмотра и фильтрации задач.

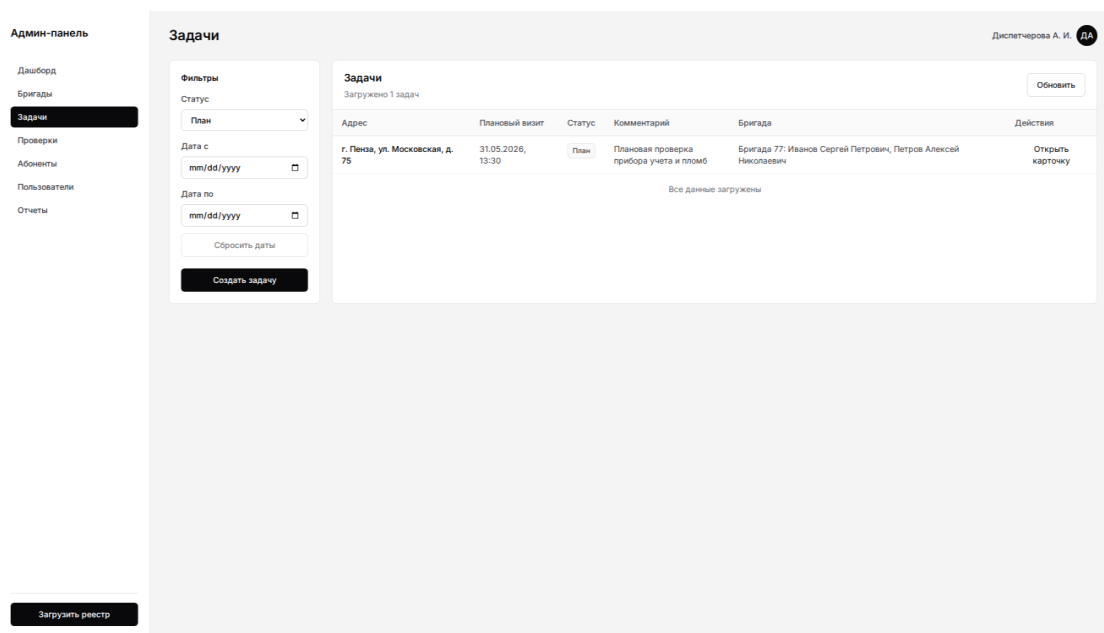


Рисунок 42 – Результат теста просмотра и фильтрации задач

Проверка создания задачи фиксировала корректность формы и выбора связанных сущностей. На рисунке 43 представлен результат теста создания задачи и назначения бригады.

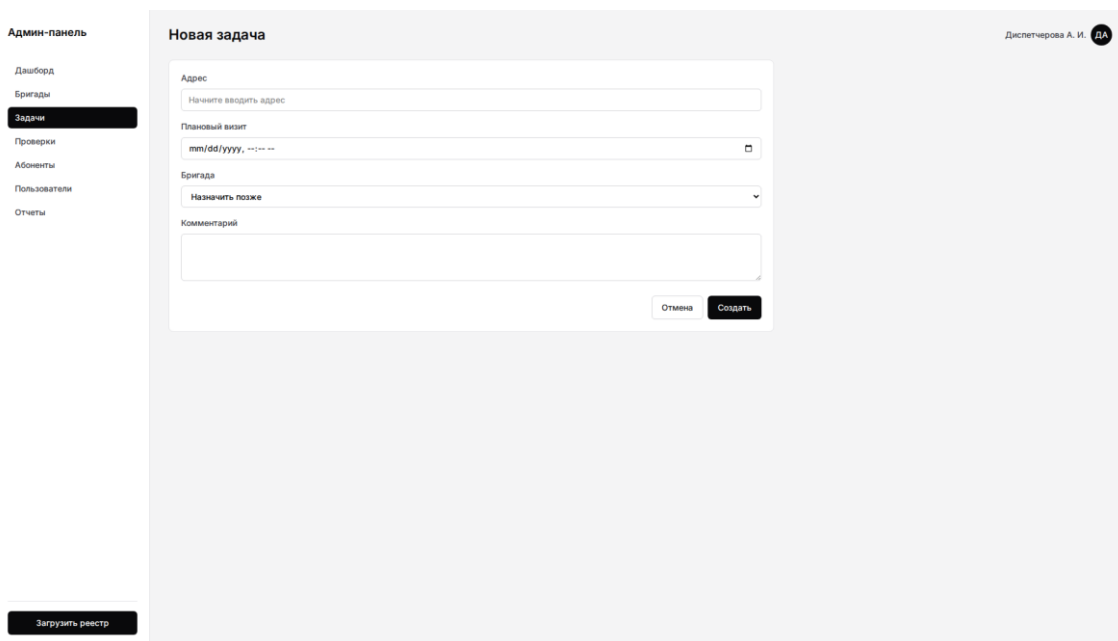


Рисунок 43 – Результат теста создания задачи и назначения бригады

После создания задачи контролировался просмотр данных бригады,

которой назначаются работы. На рисунке 44 представлен результат теста просмотра карточки бригады.

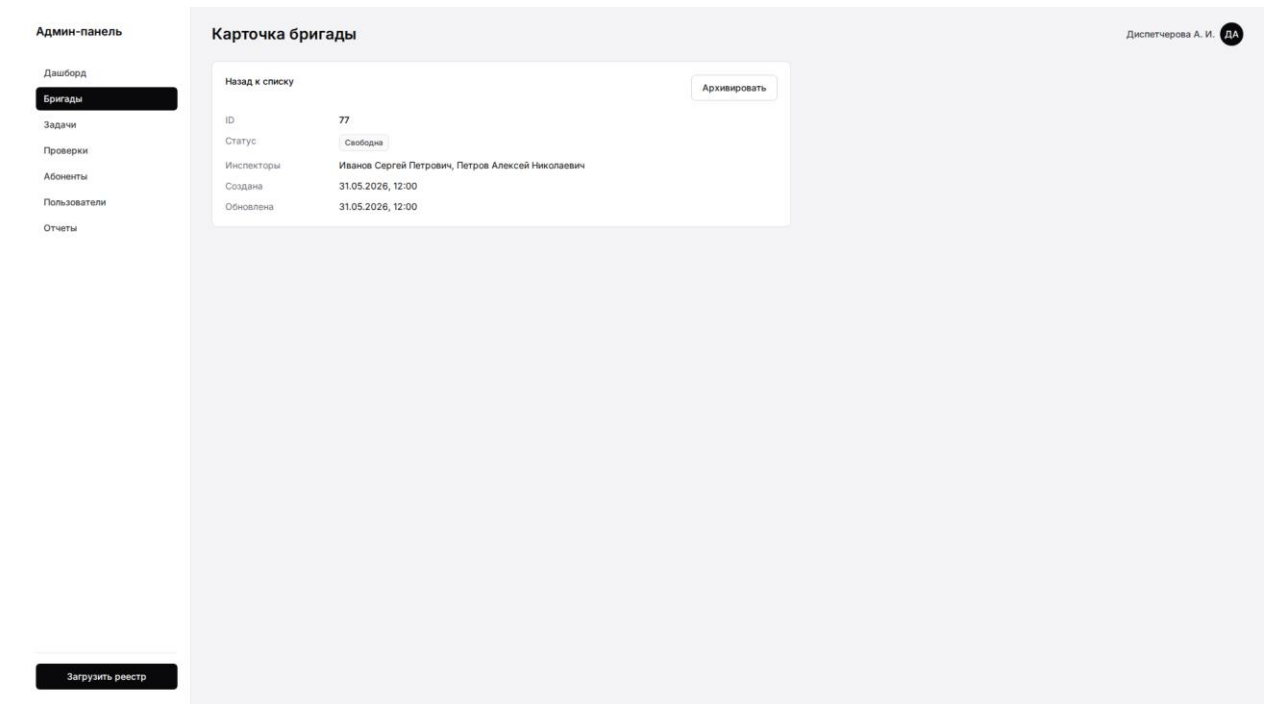


Рисунок 44 – Результат теста просмотра карточки бригады

Следующий сценарий проверял переход к абонентским данным, используемым при планировании проверки. На рисунке 45 представлен результат теста просмотра карточки абонента.

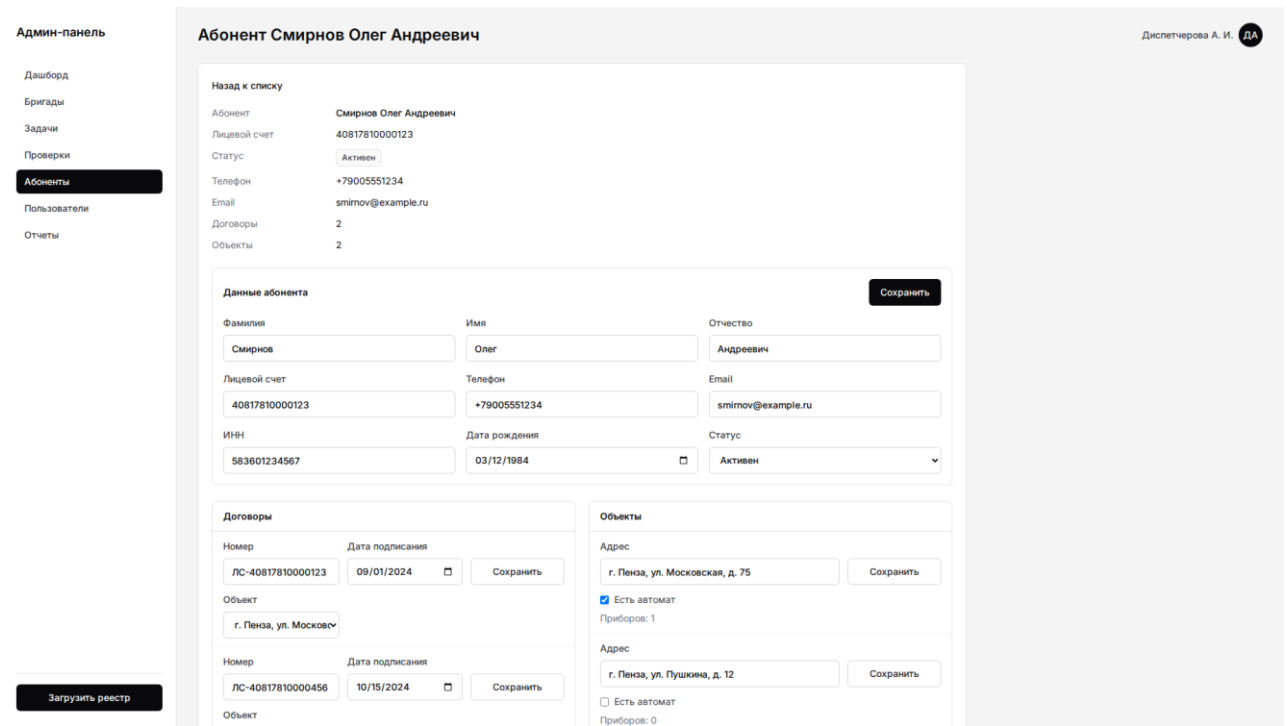


Рисунок 45 – Результат теста просмотра карточки абонента

После проверки абонентской карточки контролировалось отображение результата выполненной инспекции. На рисунке 46 представлен результат теста просмотра выполненной проверки.

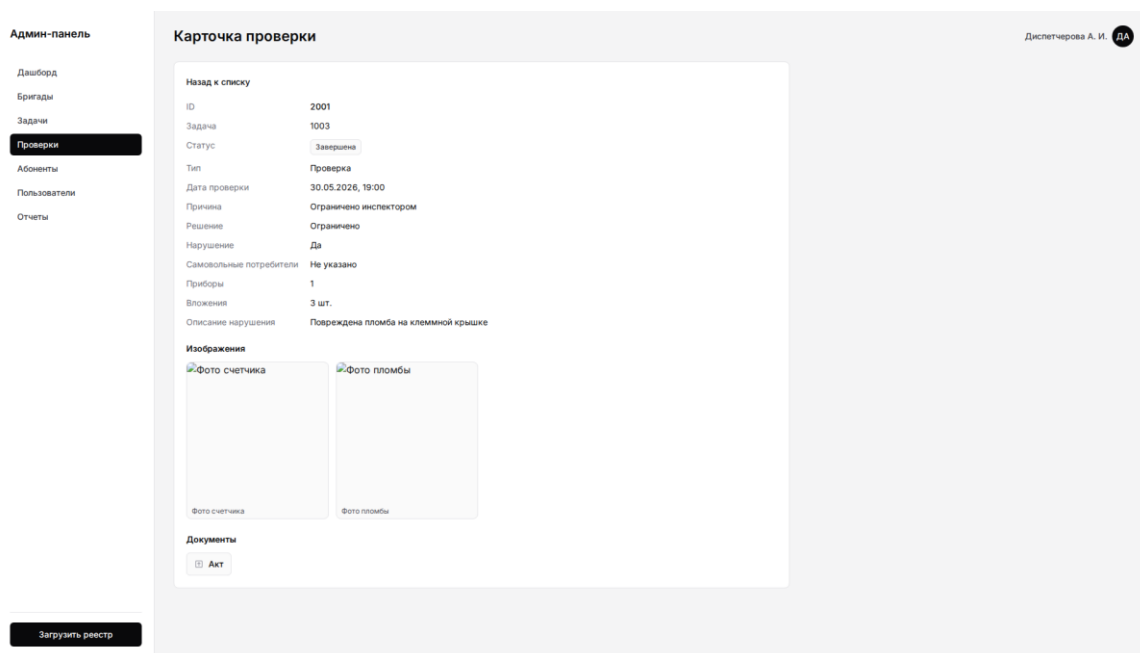


Рисунок 46 – Результат теста просмотра выполненной проверки

Сценарий приглашения пользователя проверял административную часть управления доступом. На рисунке 47 представлен результат теста формирования ссылки приглашения.

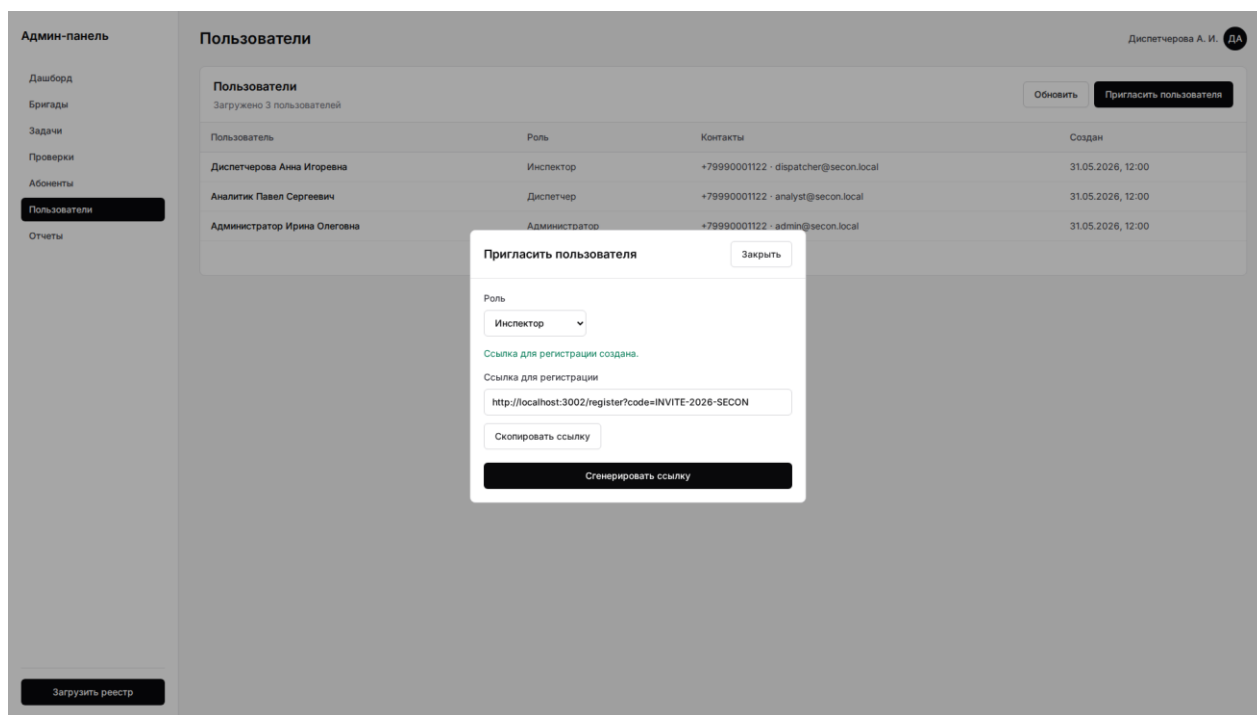


Рисунок 47 – Результат теста формирования ссылки приглашения пользователя

Затем проверялся отдельный мобильный контур входа бригады. На рисунке 48 представлен результат теста входа бригады в мобильное приложение.

Войти

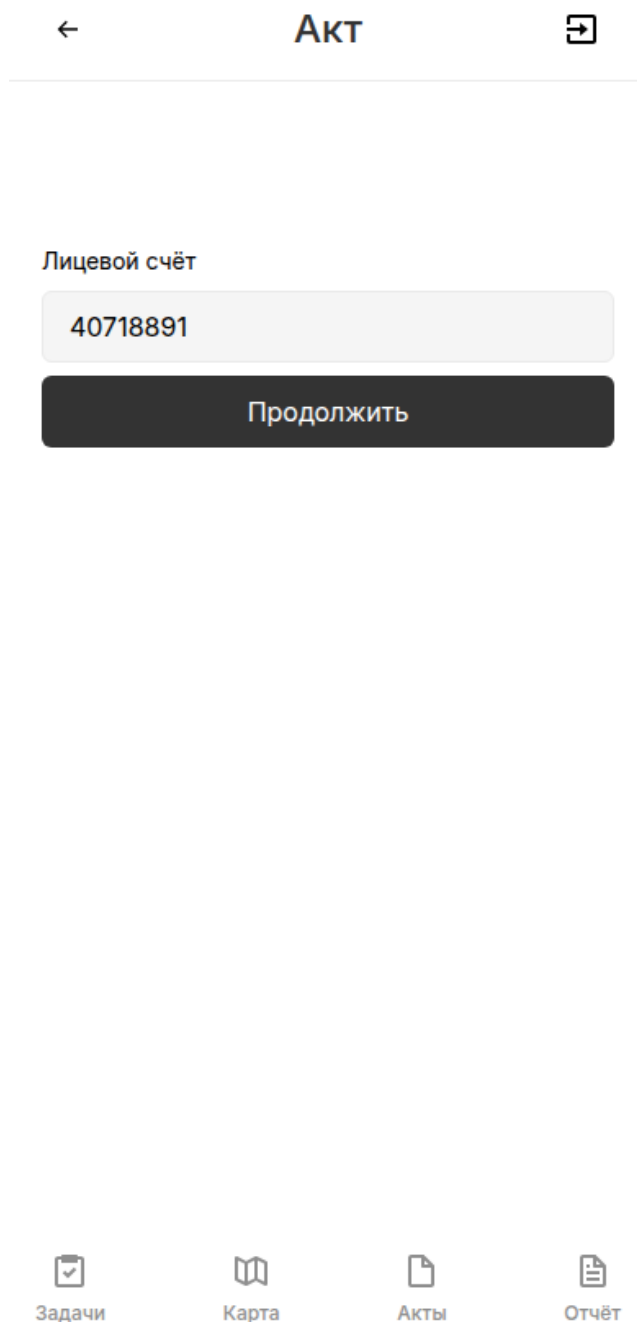
Бригада

Войти

Рисунок 48 – Результат теста входа бригады в мобильное приложение

Проверка мобильного входа подтверждает, что бригада может начать работу без отдельной учетной записи сотрудника. Для полевого сценария это снижает количество действий перед началом смены и позволяет сразу перейти к списку назначенных задач.

После входа бригады проверялся основной полевой сценарий заполнения акта. Результат выполнения теста заполнения акта инспектором представлен на рисунке 49.



← **Акт** →

Лицевой счёт

40718891

Продолжить

Задачи Карта Акты Отчёт

Рисунок 49 – Результат теста заполнения акта инспектором

Сценарий заполнения акта является наиболее критичным для мобильного приложения. В ходе проверки оценивалось, что пользователь может последовательно пройти шаги формы, сохранить сведения об абоненте и объекте, прикрепить фотоматериалы и завершить результат проверки без

возврата к бумажному документообороту.

Полученный экран подтверждает работоспособность многошагового подхода: инспектор видит только текущий набор полей, а уже введенные данные сохраняются в состоянии формы. Это уменьшает вероятность пропуска обязательных сведений и делает оформление результата более предсказуемым.

Завершающий сценарий подтверждал доступ инспектора к созданным материалам и отчетам. Результат теста просмотра отчетов инспектором представлен на рисунке 50.

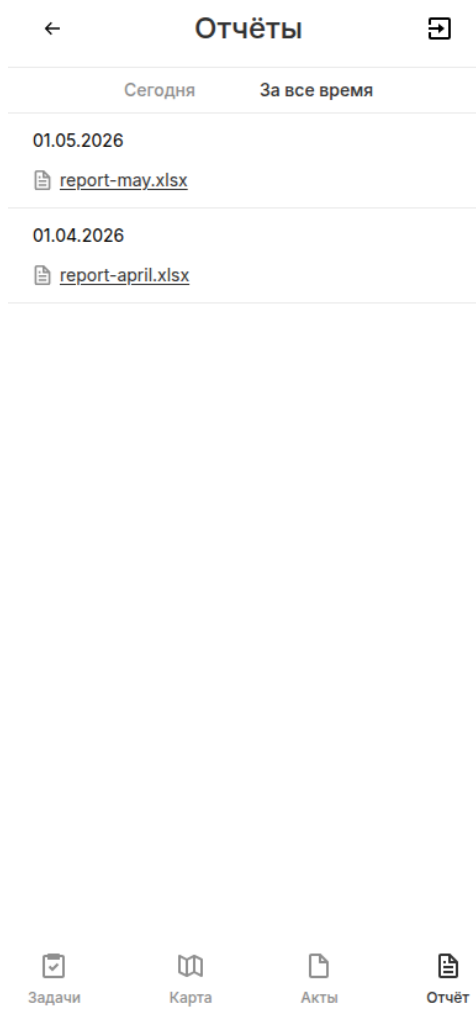


Рисунок 50 – Результат теста просмотра отчетов инспектором

Просмотр отчетов завершает мобильный сценарий контроля результатов. Бригада может убедиться, что сформированные материалы доступны в приложении, а диспетчерский контур получает основание для последующего анализа выполненных работ.

Таким образом, серия функциональных проверок показывает не только корректность отдельных экранов, но и связность всего пользовательского цикла: от входа и получения задачи до оформления результата и просмотра итоговых материалов.

4.3 Проверка интерфейсов административной панели в Storybook

Для проверки пользовательского интерфейса административной панели был использован Storybook. Storybook представляет собой среду изолированной разработки, демонстрации и тестирования интерфейсных компонентов и страниц [18]. В отличие от запуска всего приложения, Storybook позволяет открыть отдельный экран с заранее заданным набором входных данных, состоянием авторизации, маршрутом и ответами API. Это особенно важно для административной панели, где один и тот же экран должен корректно работать в состояниях загрузки, ошибки, пустого списка и заполненных данных.

В проекте Storybook подключен к приложению админ-панели и использует тот же стек, что и основная административная панель: React, React Router, Redux Toolkit Query и Rsbuild. Конфигурация находится в каталоге `apps/admin/.storybook`. Файл `main.ts` задает поиск stories и сборщик `storybook-react-rsbuild`, а файл `preview.tsx` подключает общие провайдеры приложения: Redux store, MemoryRouter, стили административной панели и загрузчик MSW. Для каждой story создается отдельный Redux store, поэтому данные RTK Query, ошибки и состояние загрузки не переносятся между сценариями.

Для имитации серверной части используется Mock Service Worker. Моки описаны в каталоге `apps/admin/.storybook/mocks` и разделены по доменным областям: `auth`, `brigade`, `task`, `inspection`, `subscriber`, `report`, `user` и `geo`. Такой подход позволяет проверять интерфейс без обращения к реальному backend и воспроизводить трудноуловимые состояния: бесконечную загрузку, ошибку запроса, пустой ответ, карточку существующей сущности и успешное выполнение мутации. Отдельное внимание уделено страницам, зависящим от параметров маршрута. Карточки бригады, абонента и проверки используют

useParams, поэтому для них в stories задаются initialEntries и шаблон маршрута. Благодаря этому Storybook открывает страницу в том же контексте, что и реальное приложение, например /admin/inspections/1 или /admin/subscribers/1. Это устраняет ситуацию, при которой компонент получает пустой идентификатор и вместо проверяемой карточки показывает сообщение о некорректном ID.

Каждая story описывает не только внешний вид экрана, но и проверяемое состояние. Для списочных страниц используются варианты WithData (данные загружены), Empty (данные отсутствуют), Loading (загрузка) и Error (ошибка на сервере). Для карточек сущностей дополнительно проверяется корректная загрузка объекта по идентификатору. Для форм создания задачи и бригады задаются моки зависимых справочников, чтобы проверить готовое состояние формы, загрузку справочников и ошибку их получения. Набор проверяемых stories приведен в таблице 19.

Таблица 19 – Проверка экранов административной панели в Storybook

Группа stories	Проверяемые экраны	Моки и контекст	Проверяемое поведение
Auth	LoginPage, RegisterPage	Сессия администратора, ответы /auth/me, login, register и init-registration	Вход пользователя, регистрация по приглашению
Dashboard	DashboardPage	Задачи, проверки, бригады и абоненты с разными статусами	Отображение сводной статистики, графиков и последних записей без обращения к реальному API
Brigades	BrigadesPage, BrigadeCreatePage, BrigadeDetailsPage	Список бригад, список пользователей-инспекторов, карточка бригады и архивирование	Список, пустое состояние, загрузка, ошибка, заполнение формы создания и открытие карточки по route id

Продолжение таблицы 19

Группа stories	Проверяемые экраны	Моки и контекст	Проверяемое поведение
Tasks	TasksPage, TaskCreatePage	Задачи, объекты электроснабжения, доступные бригады, extended-данные задачи	Фильтрация списка, открытие модального окна задачи, создание задачи и проверка зависимых справочников
Inspections	InspectionsPage, InspectionDetailsPage	Список проверок, карточка проверки, вложения, поиск проверки по задаче	Отображение типов и статусов проверок, карточки с фотоматериалами, состояния загрузки и ошибки
Subscribers	SubscribersPage, SubscriberDetailsPage	Список абонентов, расширенная карточка, договоры, объекты и геокодирование адреса	Поиск абонента, открытие карточки по route id, редактирование данных абонента, договора и объекта
Reports	ReportsPage	Список отчетов и ответ создания базового отчета	Отображение файлов отчетов, пустое состояние, ошибка загрузки и форма формирования отчета за период
Users	UsersPage	Список пользователей и ответ создания приглашения	Просмотр пользователей, состояние загрузки, ошибок и отсутствия данных

Проверка stories выполняется двумя способами. Первый способ используется во время разработки: разработчик запускает команду `pnpm --filter @secon/admin storybook`, открывает нужную story в браузере и визуально проверяет конкретное состояние экрана. Такой режим удобен для согласования интерфейса, так как не требует подготовки данных на сервере и позволяет быстро переключаться между вариантами Loading, Error, Empty и WithData.

Внешний вид панели Storybook с открытой story дашборда административной панели и нижней панелью проверок показан на рисунке 51.

Панель проверок рядом со story позволяет сразу увидеть, прошел ли

сценарий отображения без ошибки маршрута, моков или ожидаемых элементов.

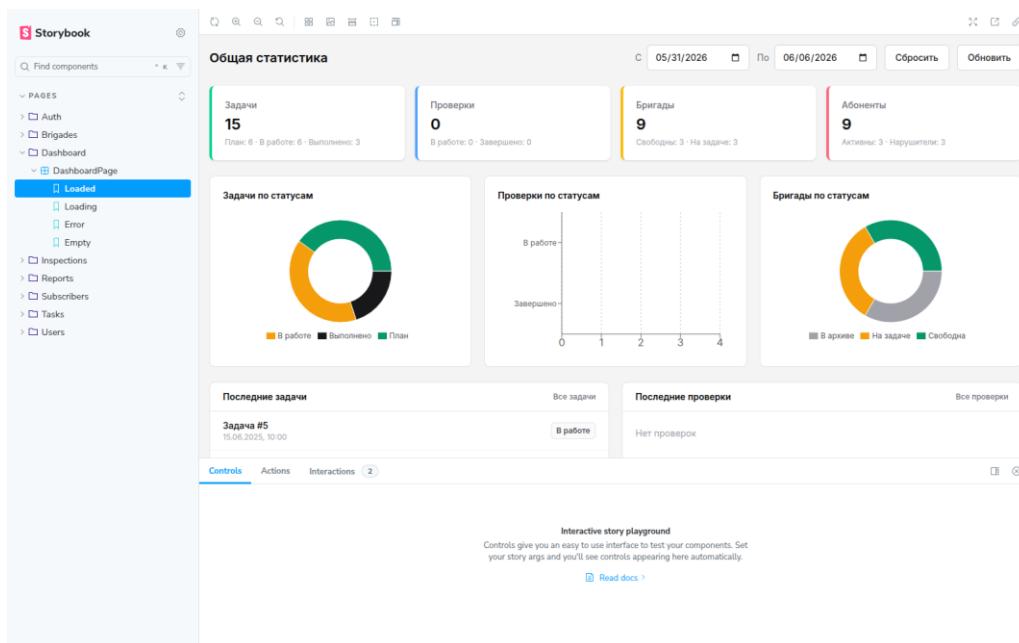


Рисунок 51 – Панель Storybook со story дашборда административной панели

В рамках автоматизированной проверки были написаны Storybook interaction tests для экранов административной панели. Stories описывают состояния WithData, Empty, Loading и Error, а play-функции из `@storybook/test` проверяют появление ключевых элементов после рендеринга страницы, работу маршрутов с параметрами и корректность моков API.

Второй способ является автоматизированным. В документации Storybook interaction tests описываются как проверки, в которых story задает начальное состояние компонента или страницы, а play-функция имитирует действия пользователя и выполняет утверждения после рендеринга [18]. Для stories, где необходимо подтвердить корректную загрузку ключевого элемента, используются play-функции из `@storybook/test`. Эти функции проверяют наличие конкретных элементов интерфейса: заголовков разделов, кнопки архивирования, изображения вложения проверки, заполненного поля фамилии абонента и других признаков успешной загрузки состояния. Тем самым Storybook используется не только как витрина компонентов, но и как средство браузерного smoke-тестирования страниц. Контрольный запуск выполнялся на собранной версии Storybook. В результате было пройдено 52 браузерные проверки. Успешный

запуск подтверждает, что описанные состояния страниц административной панели отображаются с корректными моками, не уходят в бесконечную загрузку и не показывают ошибку некорректного идентификатора там, где должна отображаться карточка сущности.

Использование Storybook снижает риск регрессий интерфейса при изменении клиентского API, структуры данных и маршрутизации. Если изменяется DTO проверки, карточка абонента или логика защищенного маршрута, соответствующая story сразу показывает проблему в изолированном сценарии. Поэтому Storybook в данной работе применяется как средство документирования интерфейсных состояний, проверки моков API и автоматизированного контроля основных экранов административной панели.

4.4 Ручная проверка интерфейса

Автоматизированные тесты не позволяют полностью оценить пригодность интерфейса к работе, поэтому дополнительно была выполнена ручная проверка экранов. Для административной панели использовался desktop viewport, соответствующий рабочему месту диспетчера. Для мобильного АРМ инспектора проверялась компоновка на мобильном экране, так как приложение предназначено для использования на Android-устройстве. Чек-лист ручной проверки приведен в таблице 20.

Таблица 20 – Чек-лист ручной проверки пользовательского интерфейса

Проверяемый аспект	Административная панель	Мобильное приложение инспектора	Результат
Публичные и защищенные экраны	Проверены экран входа и защищенные разделы после авторизации	Проверен экран входа бригады и защищенная навигация	Критических отклонений не выявлено
Читаемость таблиц и форм	Проверены таблицы задач, бригад, абонентов, проверок, пользователей и отчетов	Проверены карточки задач и формы акта	Данные читаемы на целевых экранах
Отсутствие наложения элементов	Проверены панели, карточки и модальные окна	Проверены шаги формы, навигация и заголовки	Наложения ключевых элементов не выявлены

Продолжение таблицы 20

Проверяемый аспект	Административная панель	Мобильное приложение инспектора	Результат
Состояния загрузки и ошибок	Проверены страницы состояния при загрузке и ошибке запроса	Проверены состояния ожидания данных и пустых списков	Состояния отображаются пользователю
Доступность основных действий	Создание задачи, открытие карточек и переходы доступны из навигации	Переход к карте, акту, отчетам и карточке задачи доступен из нижнего меню	Основные действия доступны без скрытых переходов
Соответствие реализованным сценариям	Экраны соответствуют сценариям диспетчера, показанным на рисунках 25–39	Экраны соответствуют сценариям инспектора, показанным на рисунках 17–22	Интерфейс подтверждает проектные сценарии

Ручная проверка подтвердила, что разработанные экраны соответствуют сценариям, представленным в проектном и реализационном разделах. Наиболее важным для мобильного приложения является отсутствие перегрузки экрана на шагах акта, так как инспектор работает с формой в полевых условиях и должен быстро перейти от карточки задачи к фотофиксации и завершению проверки.

4.5 Оценка сложности программного кода по метрикам Холстеда

Для количественной оценки сложности, а также приблизительной трудоемкости понимания и сопровождения кода были рассчитаны метрики Холстеда для нескольких репрезентативных модулей клиентской части.

Вычисление словаря программы производится по формуле:

$$n = n_1 + n_2,$$

где n_1 – число уникальных операторов;

n_2 – число уникальных операндов.

Вычисление длины программы производится по формуле:

$$N = N_1 + N_2,$$

где N_1 – общее число операторов;

N_2 – общее число операндов.

Вычисление объема программы производится по формуле:

$$V=N*\log_2 n,$$

где N – длина программы;

n – словарь программы.

Вычисление сложности программы производится по формуле:

$$D=\frac{n_1N_2}{2n_2},$$

где N_2 – общее число операндов;

n_1 – число уникальных операторов;

n_2 – число уникальных операндов.

Вычисление усилия реализации производится по формуле:

$$E=D*V,$$

где D – сложность программы;

V – объем программы.

Результаты расчета приведены в таблице 21.

Таблица 21 – Расчет метрик Холстеда для модулей клиентского приложения

Модуль	n1	n2	N1	N2	n	N	V	D	E
Навигация шагов акта	26	66	384	205	92	589	3842,4	40,4	155150,6
Завершение инспекции	29	125	451	289	154	740	5377,4	33,5	180272,7
Преобразование задач админ-панели	32	130	531	322	162	853	6260,9	39,6	248124,0
Преобразование абонентов	28	56	164	94	84	258	1649,2	23,5	38756,6

Наибольшее расчетное усилие получено для модуля преобразования задач административной панели. Это связано с большим числом проверок статусов, подготовкой параметров пагинации, форматированием дат и сборкой запросов создания или обновления задачи. Модули, связанные с актом инспектора, также имеют повышенную сложность, поскольку в них учитываются разные типы актов и последовательность шагов формы. Риск ошибок в этих участках снижается за счет вынесения правил в доменные helper-функции и покрытия их

автоматизированными тестами.

4.6 Функционально-стоимостной анализ

Функционально-стоимостной анализ (ФСА) представляет собой метод исследования системы, позволяющий сопоставить полезные функции программного продукта с затратами на их реализацию [19]. В рамках данной работы ФСА используется для оценки того, какие этапы разработки формируют основную стоимость клиентской части и какие функции обеспечивают практический эффект для диспетчера и инспектора.

Основная идея ФСА состоит в том, что пользователь приобретает не сам программный код, а набор функций, решающих его задачи. Для рассматриваемой системы такими функциями являются планирование проверок, назначение бригад, сопровождение работы инспектора на объекте, фотофиксация, формирование материалов проверки и просмотр отчетов. Повысить ценность продукта можно двумя способами: снижением трудозатрат на реализацию и увеличением числа полезных для пользователя функций.

Для расчета стоимости использована оценка оплаты труда программиста-разработчика в Пензе по данным сервиса Finder [20]. Динамика зарплаты приведена на рисунке 52.

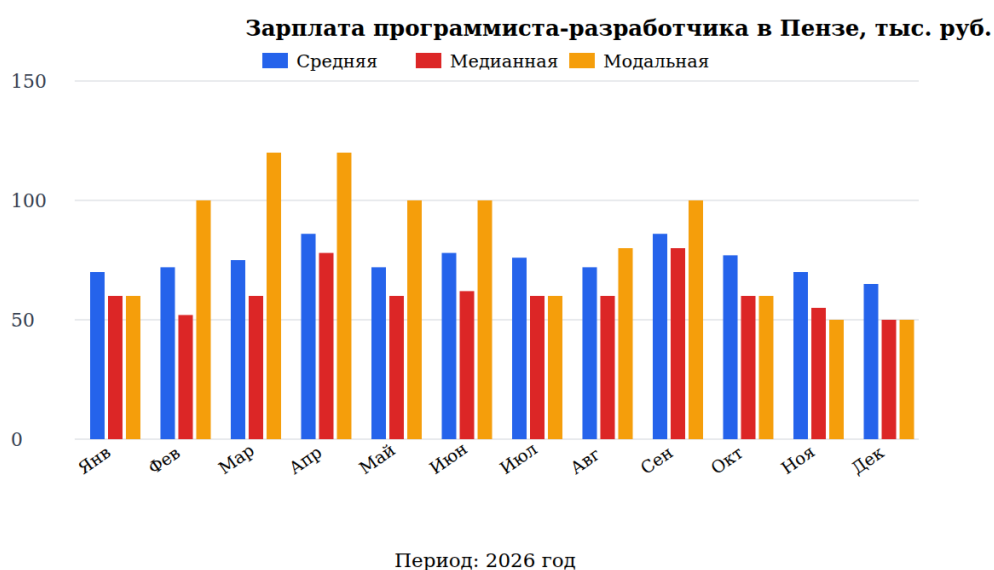


Рисунок 52 – Диаграмма зарплаты программиста-разработчика в Пензе

Исходя из диаграммы и опубликованных данных по вакансиям, средняя зарплата программиста-разработчика в Пензе составляет около 74000 рублей в месяц [19]. При стандартной 40-часовой рабочей неделе расчетная ставка составляет приблизительно 460 рублей в час. Эта ставка использована для условной оценки трудозатрат проекта.

Распределение функций и трудозатрат по этапам разработки приведено в таблице 22.

Таблица 22 – Распределение функций и трудозатрат проекта

Этап	Основная функция	Трудозатраты, ч	Результат этапа
Анализ предметной области	Выявление процессов контроля электроэнергии и ролей участников	32	Сформулированы требования и ограничения
Проектирование клиентской части	Определение архитектуры, маршрутов, данных и сценариев	40	Подготовлены проектные решения
Разработка мобильного АРМ инспектора	Получение задач, карта, карточка задачи, заполнение акта	86	Создано мобильное приложение инспектора
Разработка административной панели	Управление задачами, бригадами, абонентами, проверками и отчетами	96	Создано рабочее место диспетчера
Интеграция с API	Подключение generated API, адаптация DTO, внешние сервисы карты	42	Клиентская часть связана с backend
Контроль качества	Тестирование, сборка, проверка интерфейсов	38	Подтверждены основные сценарии
Оформление документации	Подготовка ВКР и описания результатов	30	Оформлены материалы проекта

План разработки, сформированный по этапам ФСА, представлен на рисунке 53. Диаграмма Ганта показывает параллельность разработки мобильного АРМ инспектора и административной панели после завершения проектирования.

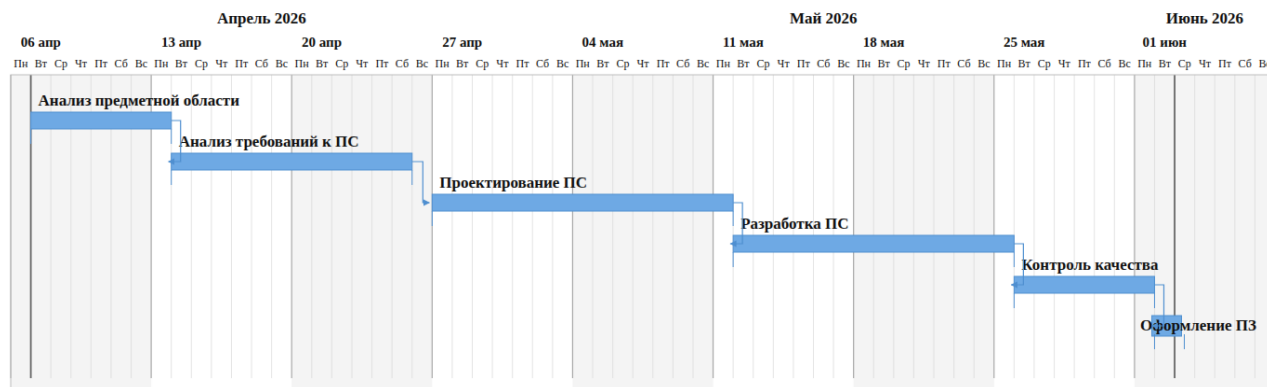


Рисунок 53 – Диаграмма Ганта разработки клиентской части

На основе трудоемкости из таблицы 23 рассчитана условная стоимость реализации клиентской части. Стоимость этапа рассчитывалась по формуле:

$$C=T*S,$$

где C – стоимость этапа разработки;

T – трудоемкость этапа, ч;

S – расчетная ставка специалиста, руб./ч.

Расчет приведен в таблице 23.

Таблица 23 – Затраты на реализацию проекта

Этап	Трудозатраты	Ставка	Стоимость
Анализ предметной области	32 ч	460 руб./ч	14720 руб.
Проектирование клиентской части	40 ч	460 руб./ч	18400 руб.
Разработка мобильного АРМ инспектора	86 ч	460 руб./ч	39560 руб.
Разработка панели администратора	96 ч	460 руб./ч	44160 руб.
Интеграция с API	42 ч	460 руб./ч	19320 руб.
Контроль качества	38 ч	460 руб./ч	17480 руб.

Продолжение таблицы 23

Этап	Трудозатраты	Ставка	Стоимость
Оформление документации	30 ч	460 руб./ч	13800 руб.
Итого	364 ч	460 руб./ч	167440 руб.

В результате проведения функционально-стоимостного анализа рассчитано, что условная стоимость разработки клиентской части составляет 167440 рублей без учета серверной инфраструктуры, промышленной эксплуатации и публикации Android-приложения. Наибольшую долю стоимости формируют реализация административной панели и мобильного АРМ инспектора, поскольку эти этапы содержат основные пользовательские функции системы.

Таким образом контроль качества клиентских приложений выполнен по нескольким направлениям: проверены функциональные сценарии диспетчера и инспектора, автоматизированные тесты доменной логики, ручная пригодность интерфейса и production-сборка обоих приложений. Функционально-стоимостной анализ показал, что основная часть трудозатрат связана с разработкой двух пользовательских контуров, а оставшиеся ограничения касаются настройки единого тестового запуска для мобильного приложения и устранения проблем со сборкой. Полученные результаты показывают, что дальнейшее развитие проекта целесообразно связывать не с изменением выбранной архитектуры, а с расширением набора проверяемых сценариев и повышением автоматизации контроля качества. В первую очередь это относится к стабильной проверке мобильного сценария акта, загрузки фотографий и согласованности данных между административной панелью и мобильным приложением.

Заключение

В ходе выполнения выпускной квалификационной работы разработано клиентское приложение для информационной системы контроля и анализа потребления электроэнергии. В составе решения рассмотрены предметная область, роли участников, информационные потоки, требования к мобильному рабочему месту инспектора и панели диспетчера. В результате проектирования определены маршруты, доменная структура клиентской части, сценарий формирования акта, интеграция с серверной системой и варианты поставки приложения. При выборе архитектурных решений учитывались подходы к построению информационных систем и программной архитектуры [1].

В процессе разработки реализованы мобильный веб-клиент инспектора, административная панель диспетчера, типизированный обмен данными с сервером, поддержка фотофиксации, работа с картой, сборка PWA и Android-приложения. Использование Capacitor [9], Tailwind CSS [15], RTK Query [14], Rsbuild [16], PWA-подхода [11] и GitHub Actions [17] позволило связать пользовательские сценарии с воспроизводимой сборкой и поставкой клиентского приложения.

Поставленная цель достигнута: разработанное клиентское приложение обеспечивает получение задач бригадой, сопровождение проверки на объекте, оформление результата, просмотр актов и отчетов, а также административное управление задачами и пользователями. Контроль качества показал, что основные пользовательские сценарии диспетчера и инспектора подтверждены функциональными проверками, доменная логика административной панели покрыта модульными тестами, а состояния интерфейса административной панели дополнительно зафиксированы и проверены в Storybook [18]. Оба клиентских приложения успешно собираются для поставки, поэтому разработанное решение может использоваться как основа для дальнейшей интеграции с серверной частью и опытной эксплуатации.

Практическая значимость разработанного клиентского приложения состоит в том, что оно переводит основные действия инспектора и диспетчера в

единый цифровой контур. Инспектор получает назначенные задачи, использует карту и форму результата проверки, а диспетчер видит состояние работ и связанные материалы без ожидания ручной передачи сведений. Это снижает риск расхождений между фактическим выполнением проверки и данными информационной системы.

Дальнейшее развитие решения может быть связано с расширением набора типов актов, усилением автоматизированной проверки мобильного сценария, улучшением обработки ошибок при загрузке фотографий и развитием отчетных представлений для диспетчера. При этом выбранная структура клиентской части позволяет добавлять новые функции без переработки всего приложения, поскольку пользовательские сценарии разделены по ролям, а общий обмен данными вынесен в отдельный пакет и доменные модули.

При развитии серверного API такая структура позволит обновлять общий клиент и доменные модули без полной переработки экранов.

Это особенно важно для отчетных функций: новые показатели и представления могут добавляться постепенно, не нарушая уже реализованный цикл работы инспектора и диспетчера. Дополнительным результатом работы является подготовка основы для дальнейшей эксплуатации клиентской части: структура маршрутов, доменные модули и общий API-клиент позволяют расширять функциональность без нарушения уже реализованных пользовательских сценариев.

Таким образом цель выполнения выпускной квалификационной работы была достигнута – было разработано готовое к эксплуатации, удобное, а также масштабируемое клиентское приложение (веб-приложение для диспетчеров и PWA приложение для инспекторов) для информационной системы контроля и анализа потребления электроэнергии. Такое клиентское приложение позволит упростить работу как диспетчеров, так и выездных инспекторов, а также устранит трудности поддерживания и проблемы с качеством заполненных актов и приложенных фотографий. Фрагменты кода разработанного клиентского приложения представлены в приложении Б.

Список использованных источников

1. Информационные системы: определение и методологии создания // Otus : образовательный портал. – [Электронный ресурс]. – URL: <https://otus.ru/journal/informacionnye-sistemy-opredelenie-i-metodologii-sozdaniya/> (дата обращения: 27.05.2026).
2. Контроль учета электроэнергии // Cometa : блог. – [Электронный ресурс]. – URL: <https://cometa.ru/blog/detail/kontrol-ucheta-elektroenergii/> (дата обращения: 27.05.2026).
3. Автоматизация работы контролеров // СТЕК-Контролер : официальный сайт. – [Электронный ресурс]. – URL: <https://stack-it.ru/avtomatizaciya-raboty-kontrollerov/> (дата обращения: 27.05.2026).
4. Система управления планированием и контроля потребления электроэнергии // НИИАС : официальный сайт. – [Электронный ресурс]. – URL: <https://niias.ru/products-and-services/products/infrastrukturnye-kompleksy/programmno-apparatnyy-kompleks-tsentrov-planirovaniya-i-kontrolya-potrebleniya-elektroenergii-oao-rzh/> (дата обращения: 27.05.2026).
5. Introduction to SafetyCulture // SafetyCulture Help Center. – [Электронный ресурс]. – URL: <https://help.safetyculture.com/004053> (дата обращения: 27.05.2026).
6. Вигерс, К. Разработка требований к программному обеспечению / К. Вигерс, Д. Битти ; пер. с англ. – 3-е изд., доп. – М. : Русская редакция ; СПб. : БХВ-Петербург, 2014. – 736 с.
7. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон ; пер. с англ. – 2-е изд. – М. : ДМК Пресс, 2006. – 496 с.
8. Feature-Sliced Design // Официальная документация архитектурной методологии. – [Электронный ресурс]. – URL: <https://feature-sliced.design/> (дата обращения: 30.05.2026).
9. Build workflow and Android platform // Capacitor Documentation. – [Электронный ресурс]. – URL: <https://capacitorjs.com/docs> (дата обращения: 30.05.2026).

10. Маршрутизация для React-приложений // React Router Documentation : официальная документация. – [Электронный ресурс]. – URL: <https://reactrouter.com/> (дата обращения: 01.06.2026).
11. Progressive web apps // MDN Web Docs. – [Электронный ресурс]. – URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (дата обращения: 30.05.2026).
12. React. Официальная документация библиотеки React для построения пользовательских интерфейсов // React Documentation. – [Электронный ресурс]. – URL: <https://react.dev/> (дата обращения: 01.06.2026).
13. TypeScript. Официальная документация // TypeScript Documentation. – [Электронный ресурс]. – URL: <https://typescriptlang.org/docs> (дата обращения: 30.05.2026).
14. RTK Query // Redux Toolkit Documentation. – [Электронный ресурс]. – URL: <https://redux-toolkit.js.org/rtk-query/overview> (дата обращения: 30.05.2026).
15. Tailwind CSS v4 and CSS-first configuration // Tailwind CSS Documentation. – [Электронный ресурс]. – URL: <https://tailwindcss.com/docs> (дата обращения: 30.05.2026).
16. React plugin and production build configuration // Rsbuild Documentation. – [Электронный ресурс]. – URL: <https://rsbuild.dev/> (дата обращения: 30.05.2026).
17. Workflow syntax and build automation // GitHub Actions Documentation. – [Электронный ресурс]. – URL: <https://docs.github.com/en/actions> (дата обращения: 30.05.2026).
18. Interaction tests and play function // Storybook Documentation. – [Электронный ресурс]. – URL: <https://storybook.js.org/docs/writing-tests/interaction-testing> (дата обращения: 01.06.2026).
19. Функционально-стоимостный анализ // Рувики : Интернет-энциклопедия. – [Электронный ресурс]. – URL: https://ru.ruwiki.ru/wiki/Функционально-стоимостный_анализ (дата обращения:

31.05.2026).

20. Зарплата программиста в Пензе в 2026 году, средняя и медианная // Finder : сервис по поиску работы. – [Электронный ресурс]. – URL: <https://finder.work/salaries/programmist/penza> (дата обращения: 30.05.2026).

Приложение А
(справочное)
Глоссарий

Абонент – потребитель электроэнергии, сведения о котором используются при создании задачи и оформлении результата проверки.

Административная панель – веб-интерфейс диспетчера для управления задачами, бригадами, абонентами, проверками, пользователями и отчетами.

АРМ инспектора – мобильное рабочее место бригады для просмотра задач, работы с картой, заполнения акта и прикрепления фотографий.

Бригада – группа исполнителей, которой назначаются задачи на проверку объектов энергопотребления.

Диспетчер – пользователь административной панели, выполняющий планирование проверок, назначение бригад и контроль результатов.

Задача на проверку – запись, содержащая объект, адрес, плановое время, назначенную бригаду, статус и данные для инспектора.

Инспекция – результат выполнения проверки, связанный с задачей, актом, фотоматериалами и итоговым статусом.

Клиентское приложение – мобильный АРМ инспектора, административная панель и общий пакет доступа к серверному API.

Объект проверки – адрес или объект энергопотребления, по которому выполняется выездная проверка.

Проверка – процесс контроля объекта, включающий просмотр исходных данных, фотофиксацию, заполнение формы и передачу результата.

Фотофиксация – прикрепление фотографий прибора учета, пломб или объекта для подтверждения состояния на момент выезда.

API – программный интерфейс серверной части для получения и изменения данных задач, бригад, абонентов, проверок и отчетов.

Приложение Б
(обязательное)
Листинг приложения

Листинг Б.1 – Исходный код страницы создания акта (АРМ инспектора)

```

import { useEffect } from 'react'
import type { ComponentType } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { FormProvider, useForm } from 'react-hook-form'
import { useNavigate } from 'react-router'
import { RootState } from '../store/store'
import { resetForm, setFormStep } from '../store/navigationSlice'
import AccountNumberStep from './act-create/AccountNumberStep'
import ActSubmitStep from './act-create/ActSubmitStep'
import ActTypeStep from './act-create/ActTypeStep'
import ConsumerContactStep from './act-create/ConsumerContactStep'
import
    EnergyActionMethodStep
    from
    './act-
create/EnergyActionMethodStep'
import MeterLocationStep from './act-create/MeterLocationStep'
import MeterPhotosStep from './act-create/MeterPhotosStep'
import MeterReadingStep from './act-create/MeterReadingStep'
import MeterTypeStep from './act-create/MeterTypeStep'
import
    RestrictionReasonStep
    from
    './act-
create/RestrictionReasonStep'
import
    RestrictionViolationStep
    from
    './act-
create/RestrictionViolationStep'
import SupplyStatusStep from './act-create/SupplyStatusStep'
import { getActCreateStep, getActCreateSteps } from './act-
create/actFlow'
import type { ActCreateStepId } from './act-create/actFlow'
import { canCreateActFromTask, createActFormDefaultValues } from
'./act-create/actFormTypes'
import type { ActCreateFormValues } from './act-create/actFormTypes'
import { Button } from '../components/ui/button'

const STEP_COMPONENTS: Record<ActCreateStepId, ComponentType> = {
  accountNumber: AccountNumberStep,
  consumerContact: ConsumerContactStep,
  meterPhotos: MeterPhotosStep,
  actType: ActTypeStep,
  restrictionReason: RestrictionReasonStep,
  supplyStatus: SupplyStatusStep,
  energyActionMethod: EnergyActionMethodStep,
  meterLocation: MeterLocationStep,
  meterType: MeterTypeStep,
  meterReading: MeterReadingStep,
  restrictionViolation: RestrictionViolationStep,
  actSubmit: ActSubmitStep,
}

function ActCreateFallback() {
  const dispatch = useDispatch()

  return (
    <div className="flex h-full flex-col justify-center gap-4
px-5">

```

```

        <p className="text-14-20-regular">
            Невозможно открыть текущий шаг акта. Начните
заполнение заново.
        </p>
        <Button          type="button"          onClick={ ()          =>
dispatch(resetForm())}>
            Вернуться к началу
        </Button>
    </div>
)
}

function ActCreate() {
    const dispatch = useDispatch()
    const navigate = useNavigate()
    const { currentStep, actType, formState } = useSelector(
        (state: RootState) => state.navigation.formSteps,
    )

    const steps = getActCreateSteps(actType)
    const step = getActCreateStep(currentStep, actType)
    const StepComponent = step ? STEP_COMPONENTS[step.id] :
undefined

    const fm = useForm<ActCreateFormValues>({
        defaultValues: createActFormDefaultValues(formState),
    })

    useEffect(() => {
        if (currentStep > steps.length && steps.length > 0) {
            dispatch(setFormStep(steps.length))
        }
    }, [currentStep, dispatch, steps.length])

    if (!canCreateActFromTask(formState)) {
        return (
            <div className="flex h-full flex-col justify-center gap-
4 px-5">
                <p className="text-14-20-regular">
                    Акт можно составить только из карточки
назначенной задачи.
                </p>
                <Button          type="button"          onClick={ ()          =>
navigate('/')}>
                    Перейти к задачам
                </Button>
            </div>
        )
    }

    return (
        <div className="flex h-[80vh] flex-col">
            <FormProvider {...fm}>

```

```

        <form onSubmit={ (event) => event.preventDefault() }>
            {StepComponent ? <StepComponent /> :
<ActCreateFallback />}
        </form>
    </FormProvider>
</div>
)
}

export default ActCreate

```

Листинг Б.2 – Исходный код формы создания задачи (АРМ диспетчера)

```

import { useMemo, useState, type FormEvent } from 'react'
import { formatBrigadeName, type AdminBrigade } from
'../../../../entities/brigade'
import { formatObjectLabel, type AdminObject } from
'../../../../entities/object'
import type { AdminTaskCreateForm } from '../../../../entities/task'
import { useDebounceValue } from '../../../../lib/useDebounceValue'

const INITIAL_FORM: AdminTaskCreateForm = {
  objectId: '',
  brigadeId: '',
  planVisitAt: '',
  comment: ''
}

type TaskCreateFormProps = {
  brigades: AdminBrigade[]
  objects: AdminObject[]
  isSubmitting: boolean
  onCreate: (form: AdminTaskCreateForm) => Promise<void>
  onCancel: () => void
}

export const TaskCreateForm = ({
  brigades,
  objects,
  isSubmitting,
  onCreate,
  onCancel,
}: TaskCreateFormProps) => {
  const [form, setForm] =
  useState<AdminTaskCreateForm>(INITIAL_FORM)
  const [addressQuery, setAddressQuery] = useState('')
  const [isAddressListOpen, setAddressListOpen] = useState(false)
  const [error, setError] = useState('')
  const debouncedAddressQuery = useDebounceValue(addressQuery,
250)
  const addressSuggestions = useMemo(() => {
    const query
    debouncedAddressQuery.trim().toLocaleLowerCase('ru-RU')
    const source = query

```

```

        ? objects.filter((object) =>
            formatObjectLabel(object).toLocaleLowerCase('ru-
RU').includes(query),
        )
        : objects

        return source.slice(0, 20)
    }, [debouncedAddressQuery, objects])

const updateField = <TKey extends keyof AdminTaskCreateForm>(
    key: TKey,
    value: AdminTaskCreateForm[TKey],
) => {
    setForm((current) => ({ ...current, [key]: value }))
    setError('')
}

const handleSubmit = async (event: FormEvent<HTMLFormElement>)
=> {
    event.preventDefault()

    if (!form.objectId) {
        setError('Выберите адрес из списка объектов.')
        return
    }

    try {
        await onCreate(form)
        setForm(INITIAL_FORM)
        setAddressQuery('')
        setAddressListOpen(false)
    } catch {
        setError('Не удалось создать задачу. Проверьте данные и
повторите попытку.')
    }

    return (
        <form onSubmit={handleSubmit}>
            <label className="block text-sm font-medium text-zinc-
700">
                Адрес
                <input
                    value={addressQuery}
                    onFocus={() => setAddressListOpen(true)}
                    onBlur={() => {
                        window.setTimeout(()
setAddressListOpen(false), 120)
                    }}
                    onChange={(event) => {
                        setAddressQuery(event.target.value)
                        updateField('objectId', '')
                        setAddressListOpen(true)

```

```

        setError('')
      }}
      className="mt-2 w-full rounded-md border
border-zinc-300 bg-white px-3 py-2 text-sm text-zinc-950 outline-
none focus:border-zinc-950"
      placeholder="Начните вводить адрес"
    />
  </label>
  {isAddressListOpen ? (
    <div className="mt-2 max-h-56 overflow-auto
rounded-md border border-zinc-200">
      {addressSuggestions.length > 0 ? (
        addressSuggestions.map((object) => {
          const label = formatObjectLabel(object)
          const objectId = object.ID === undefined
? '' : String(object.ID)
          const isSelected = objectId.length > 0
&& form.objectId === objectId

          return (
            <button
              key={object.ID ?? label}
              type="button"
              disabled={!objectId}
              onClick={() => {
                setAddressQuery(label)
                updateField('objectId',
objectId)

                setAddressListOpen(false)
              }}
              className={[
                'block w-full px-3 py-2
text-left text-sm hover:bg-zinc-50 disabled:cursor-not-allowed
disabled:text-zinc-400',
                isSelected
                  ? 'bg-zinc-950 text-
white hover:bg-zinc-950'
                  : '',
              ].join(' ')}
            >
              {label}
            </button>
          )
        })
      ) : (
        <div className="px-3 py-2 text-sm text-
zinc-500">Адрес не найден</div>
      )}
    </div>
  ) : null}
  <label className="mt-4 block text-sm font-medium text-
zinc-700">
    Плановый визит
  </label>

```

```

        <input
            value={form.planVisitAt}
            onChange={(event) => updateField('planVisitAt',
event.target.value)}
            className="mt-2 w-full rounded-md border
border-zinc-300 bg-white px-3 py-2 text-sm text-zinc-950 outline-
none focus:border-zinc-950"
            type="datetime-local"
        />
    </label>
    <label className="mt-4 block text-sm font-medium text-
zinc-700">
        Бригада
        <select
            value={form.brigadeId}
            onChange={(event) => updateField('brigadeId',
event.target.value)}
            className="mt-2 w-full rounded-md border
border-zinc-300 bg-white px-3 py-2 text-sm text-zinc-950 outline-
none focus:border-zinc-950"
        >
            <option value="">Назначить позже</option>
            {brigades.map((brigade) => (
                <option
                    key={brigade.ID
formatBrigadeName(brigade)} value={brigade.ID}>
                    {formatBrigadeName(brigade)}
                </option>
            ))}
        </select>
    </label>
    <label className="mt-4 block text-sm font-medium text-
zinc-700">
        Комментарий
        <textarea
            value={form.comment}
            onChange={(event) => updateField('comment',
event.target.value)}
            className="mt-2 min-h-20 w-full resize-y
rounded-md border border-zinc-300 bg-white px-3 py-2 text-sm text-
zinc-950 outline-none focus:border-zinc-950"
        />
    </label>
    {error ? <div className="mt-3 text-xs text-red-
700">{error}</div> : null}
    <div className="mt-5 flex justify-end gap-2">
        <button
            type="button"
            onClick={onCancel}
            className="admin-button admin-button-secondary"
        >
            Отмена
        </button>
        <button

```

```

        type="submit"
        disabled={isSubmitting}
        className="admin-button admin-button-primary"
      >
        {isSubmitting ? 'Создание...' : 'Создать'}
      </button>
    </div>
  </form>
)
}

```

Листинг Б.3 – Исходный код страницы деталей задачи (АРМ инспектора)

```

import { useParams, useNavigate } from 'react-router'
import { Button } from '../components/ui/button'
import { Label } from '../components/ui/label'
import { useEffect, useRef, useState } from 'react'
import { Spinner } from '../components/ui/spinner'
import InteractiveMap from '../components/core/map'
import LocateControl from '../components/core/LocateControl'
import { getFullName } from '../utils/strings'
import { useDispatch } from 'react-redux'
import { updateFormState } from '../store/navigationSlice'
import { LatLngTuple, Map } from '@secon/shared'
import { Geolocation } from '@capacitor/geolocation'
import { toast } from 'react-hot-toast'
import {
  TaskStatus,
  toTaskView,
  useGetTaskByIdExtendedQuery,
  useStartTaskMutation,
} from '../entities/task'
import {
  fetchInspectionByTaskIdWithRetry,
  useLazyGetInspectionByTaskIdQuery,
} from '../entities/inspection'
import { useLazyGeocodeAddressQuery } from '../entities/geo'

export const ActDetails = () => {
  const { id } = useParams<{ id: string }>()
  const navigate = useNavigate()
  const dispatch = useDispatch()
  const [userLocation, setUserLocation] = useState<LatLngTuple | null>(null)
  const [marker, setMarker] = useState<{ id: string; position: LatLngTuple } | null>(null)
  const [isActive, setIsActive] = useState(false)
  const mapRef = useRef<Map | null>(null)

  const {
    data: taskExtended,
    error,
    isLoading,
  } = useGetTaskByIdExtendedQuery({

```

```

        id: Number(id),
    })
    const task = taskExtended ? toTaskView(taskExtended) : undefined
    const [startTask] = useStartTaskMutation()
    const [fetchInspectionByTaskId] = useLazyGetInspectionByTaskIdQuery()
    const [geocodeAddress] = useLazyGeocodeAddressQuery()

    const handleCreateAct = async () => {
        if (task) {
            try {
                if (task.status === TaskStatus.StatusDone) {
                    toast.error('Задача уже завершена')
                    return
                }

                if (task.status !== TaskStatus.StatusInWork) {
                    await startTask({ id: Number(task.id)
                })}.unwrap()

                const inspection = await fetchInspectionByTaskIdWithRetry(
                    fetchInspectionByTaskId,
                    Number(task.id),
                )

                dispatch(
                    updateFormState({
                        taskId: task.id,
                        inspectionId: String(inspection.ID ?? ''),
                        accountNumber: task.accountNumber,
                        phoneNumber: task.consumer.phoneNumber,
                        consumer: getFullName(task.consumer),
                        address: task.address,
                        deviceId: task.deviceId,
                        sealId: task.sealId,
                    })
                )

                navigate('/create')
            } catch (e) {
                console.error('Ошибка при старте задачи:', e)
                toast.error('Не удалось начать работу по задаче')
            }
        }
    }

    const handleCenterOnUser = () => {
        if (mapRef.current && userLocation) {
            mapRef.current.setView(userLocation, 16)
            setIsActive(true)
            setTimeout(() => setIsActive(false), 1500)
        }
    }

```

```

        } else {
            toast.error('Не удалось определить ваше
местоположение')
        }
    }

    // Получаем геолокацию один раз при загрузке
    useEffect(() => {
        const fetchGeolocation = async () => {
            try {
                const position = await
Geolocation.getCurrentPosition({
                    enableHighAccuracy: true,
                    timeout: 10000,
                })
                const coords: LatLngTuple =
[position.coords.latitude, position.coords.longitude]
                setUserLocation(coords)
            } catch {
                // Ничего не делаем
            }
        }

        fetchGeolocation()
    }, [])

    useEffect(() => {
        const fetchMarker = async () => {
            if (!task) return
            const parts = task.address.split(',')
            if (parts.length >= 3) {
                const [rawCity, rawStreet, rawHouse] = parts.map((p)
=> p.trim())
                const city = rawCity.replace(/^\r\.\s*/i, '')
                const street = rawStreet.replace(/^\ул\.\s*/i, '')
                const house = rawHouse.replace(/^\д\.\s*/i, '')
                const formattedAddress = `${street} ${house},
${city}, Пензенская область, Россия`
                const coord = await
geocodeAddress(formattedAddress).unwrap()
                if (coord) {
                    setMarker({ id: task.id, position: coord })
                }
            }
        }

        fetchMarker()
    }, [geocodeAddress, task])

    if (isLoading) return <Spinner />
    if (error) return <div className="p-4 text-center text-red-
500">Ошибка при загрузке задачи</div>
    if (!task) return <div className="p-4 text-center">Задача не

```

найдена</div>

```

    return (
      <div className="flex h-full flex-col">
        <div className="relative h-[50vh] w-full">
          <InteractiveMap
            enableRouting={true}
            userLocation={userLocation}
            markers={marker ? [marker] : []}
            className="h-full w-full rounded-md"
            mapRef={mapRef}
          />

          <div className="absolute bottom-4 right-4 z-
[1001]">
            <LocateControl onClick={handleCenterOnUser}
isLocating={isActive} />
          </div>
        </div>

        <div className="border-t border-gray-200 px-5 pb-5">
          <div className="m-5 flex flex-col gap-3">
            <p className="text-14-20-regular select-
none">{task.address}</p>
            <Label text={getFullName(task.consumer)}
icon="user" />
            <Label
              text={task.consumer.phoneNumber}
              className="underline-black-1 underline
underline-offset-2"
              icon="phone"
            />
            <Label text={task.accountNumber} icon="idCard"
/>
            <Label text={task.comment} icon="fileBlank" />
          </div>
          <Button className="w-full"
onClick={handleCreateAct}>
            Составить акт
          </Button>
        </div>
      </div>
    )
  }
}

```

Листинг Б.4 – Исходный код страницы списка пользователей (АРМ диспетчера)

```

import { useCallback, useMemo, useState } from 'react'
import { Button, InfiniteLoadTrigger, useInfiniteList } from
'../../components/ui'
import { type AdminUsersQuery, type AdminUser, useLazyGetUsersQuery
} from '../../entities/user'
import {
  buildRegistrationLink,

```

```

    type AdminInitRegistrationForm,
    useInitRegistrationMutation,
  } from '../../entities/auth'
import { USERS_PAGE_SIZE } from '../lib/constants'
import { UserInviteModal } from '../ui/UserInviteModal'
import { UsersTable } from '../ui/UsersTable'

export const UsersPage = () => {
  const [getUsers] = useLazyGetUsersQuery()
  const [initRegistration, initRegistrationState] =
    useInitRegistrationMutation()
  const [isInviteModalOpen, setInviteModalOpen] = useState(false)
  const [invitationLink, setInvitationLink] = useState('')
  const [message, setMessage] = useState<{ type: 'success' |
'error'; text: string } | null>(null)
  const baseQuery = useMemo(() => ({}), [])
  const loadPage = useCallback((query: AdminUsersQuery) =>
getUsers(query).unwrap(), [getUsers])
  const usersList = useInfiniteList<AdminUser, AdminUsersQuery>({
    baseQuery,
    getItemKey: (user) => user.id,
    loadPage,
    pageSize: USERS_PAGE_SIZE,
  })

  const inviteUser = async (form: AdminInitRegistrationForm) => {
    setMessage(null)
    setInvitationLink('')

    try {
      const invitation = await initRegistration({
        userServiceInitRegistrationDto: {
          role: form.role,
        },
      }).unwrap()
      setInvitationLink(
        buildRegistrationLink(window.location.href,
invitation.invitationCode),
      )
      setMessage({ type: 'success', text: 'Ссылка для
регистрации создана.' })
      return true
    } catch {
      setMessage({ type: 'error', text: 'Не удалось создать
приглашение.' })
      return false
    }
  }

  return (
    <div>
      <section className="min-w-0 rounded-lg border border-
zinc-200 bg-white">

```

```

        <div className="flex items-center justify-between
border-b border-zinc-200 px-5 py-4">
            <div>
                <h2 className="text-lg font-semibold text-
zinc-950">Пользователи</h2>
                <p className="mt-1 text-sm text-zinc-500">
                    Загружено {usersList.items.length}
пользователей
                </p>
            </div>
            <div className="flex items-center gap-2">
                <Button
                    type="button"
                    variant="secondary"
                    onClick={usersList.refresh}
                    disabled={usersList.isFetching}
                >
                    {usersList.isFetching ? 'Обновление...'
: 'Обновить'}
                </Button>
                <Button
                    type="button"
                    variant="primary"
                    onClick={() => {
                        setMessage(null)
                        setInvitationLink('')
                        setInviteModalOpen(true)
                    }}
                >
                    Пригласить пользователя
                </Button>
            </div>
        </div>
        <UsersTable users={usersList.items} />
        <InfiniteLoadTrigger
            hasMore={usersList.hasMore}
            isLoading={usersList.isFetching}
            onLoadMore={usersList.loadMore}
        />
    </section>

    {isInviteModalOpen ? (
        <UserInviteModal
            invitationLink={invitationLink}
            isSubmitting={initRegistrationState.isLoading}
            message={message}
            onClose={() => setInviteModalOpen(false)}
            onInvite={inviteUser}
        />
        ) : null}
    </div>
)
}

```