

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

**Серверное приложение информационной
системы контроля и анализа потребления
электроэнергии**

Студент

(подпись, дата)

Хрунин Дмитрий Алексеевич
(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 Хрунин Дмитрий Алексеевич гр. 22ВП1 факультета ФВТ
<Фамилия, имя, отчество полностью>

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

Тема утверждена приказом ПГУ № _____ от « _____ » _____ 20 _____

3. Руководитель работы Гурьянов Л.В.

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

1) Разработать программные компоненты подсистемы, обеспечивающие
автоматизацию учета и анализа потребления электроэнергии

2) Среда выполнения приложения: семейство ОС UNIX

Языковые средства разработки: Go, PostgreSQL, ClickHouse, MinIO, UML

Инструментальные средства разработки: JetBrains GoLand, JetBrains

DataGrip, Visual Paradigm, Postman

Технологии разработки: Docker, Apache Kafka, Prometheus, Grafana

3) Основные функции программного обеспечения:

а) управление бригадами инспекторов;

б) управление абонентами;

в) управление объектами проверки;

г) управление договорами с абонентами;

- д) управление реестрами абонентов;*
- е) управление задачами на проверку объектов;*
- ж) проведение и фиксация результатов проверок;*
- з) проверка качества фотографий с целью выявления искажений;*
- и) аналитика и отчетность о работе бригад инспекторов.*

5. Перечень подлежащих разработке вопросов

- 1) Анализ предметной области и постановка задачи*
- 2) Проектирование*
- 3) Реализация*
- 4) Контроль качества программного обеспечения*
- 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 146 листов, 40 рисунков, 13 таблиц, 21 использованных источников, 8 приложений.

МИКРОСЕРВИСЫ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, ИНФОРМАЦИОННАЯ СИСТЕМА, МОНИТОРИНГ, РАСПРЕДЕЛЕННАЯ СИСТЕМА, REST API, GO, POSTGRESQL, MINIO, APACHE KAFKA, CLICKHOUSE, POSTMAN, PROMETHEUS, GRAFANA, UML

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

Предметом разработки является подсистема обеспечения механизма регистрации, хранения, обработки и представления данных о задачах, проверках, приборах учета, абонентах и отчетах.

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

Средства разработки: язык программирования Go, среды разработки JetBrains GoLand, JetBrains DataGrip, СУБД PostgreSQL, ClickHouse, объектная система хранения MinIO, брокер сообщений Apache Kafka.

Среда исполнения – семейство ОС UNIX.

Результаты работы: приведен обзор и анализ существующих информационных систем контроля и анализа потребления электроэнергии, определены функциональные и нефункциональные требования к информационной системе, выбран архитектурный стиль и выполнено проектирование программных средств, разработано и протестировано программное обеспечение механизма регистрации, хранения, обработки и представления данных о задачах, проверках, приборах учета, абонентах, аналитических показателях и отчетах.

					ПГУ 09.03.04 – 08ВР221.26 ПЗ						
Изм.	Лист	№ докум.	Подп.	Дата	Серверное приложение информационной системы контроля и анализа потребления электроэнергии. Пояснительная записка.	Лит.		Лист	Листов		
Разраб.		Хрунин Д.А.						4	146		
Пров.		Гурьянов Л.В.									
Реценз.											
Н. контр.		Такташкин Д.В.									
УтверГ.		Козлов А.Ю.									
						Группа 22ВП1					

Содержание

Введение	7
1 Анализ предметной области и постановка задачи на разработку	9
1.1 Анализ предметной области.....	9
1.1.1 Понятие контроля и анализа потребления электроэнергии	9
1.1.2 Описание предметной области и ее участников	10
1.1.3 Организационная структура информационных потоков.....	12
1.1.4 Документооборот предметной области.....	12
1.1.5 Проблемы в текущем бизнес-процессе	13
1.2 Анализ информационных систем аналогичного назначения.....	14
1.2.1 Платформа СТЕК	14
1.2.2 Программно-аппаратный комплекс АСУ ПКПЭ	15
1.2.3 Платформа SafetyCulture	17
1.3 Постановка задачи на разработку программных средств.....	19
1.3.1 Общая концепция системы.....	19
1.3.2 Основные функции и преимущества системы	20
1.3.3 Отличия от текущего бизнес-процесса	22
1.4 Функционально-стоимостной анализ	23
2 Анализ требований к разрабатываемым средствам	26
2.1 Анализ функциональных требований	26
2.2 Анализ нефункциональных требований	32
3 Проектирование программных средств	34
3.1 Общая архитектура программных средств	34
3.2 Проектирование структур данных.....	37
3.2.1 Концептуальная модель баз данных.....	37
3.2.2 Логическая модель баз данных	41
3.3 Общая архитектура серверного приложения системы	45
3.4 Обобщенная структура разрабатываемой системы	48
3.5 Требования к аппаратному обеспечению серверного приложения.....	51
4 Реализация программных средств	53

4.1	Физическая модель баз данных.....	53
4.2	Разработка архитектурных слоев внутри микросервисов.....	61
4.2.1	Разработка слоя представления данных в микросервисах	62
4.2.2	Разработка слоя бизнес-логики в микросервисах	65
4.2.3	Разработка слоя доступа к данным в микросервисах	68
4.3	Разработка аналитических дашбордов	71
5	Контроль качества программного обеспечения	76
5.1	Обоснование выбора модели тестирования и метрик кода.....	76
5.2	Тестирование серверного приложения информационной системы	78
5.2.1	Функциональное тестирование обновления статуса абонента по результатам проверки	78
5.2.2	Модульное тестирование метода создания бригады	82
5.2.3	Метрики Холстеда для функции валидации лицевого счета абонента	84
	Заключение.....	88
	Список использованных источников.....	91
	Приложение А (справочное) Глоссарий.....	94
	Приложение Б (рекомендуемое) Спецификация атрибутов таблиц.....	97
	Приложение В (рекомендуемое) Объекты обращения к базам данных.....	115
	Приложение Г (рекомендуемое) SQL-скрипты	121
	Приложение Д (рекомендуемое) Структуры слоя представления данных.....	132
	Приложение Е (рекомендуемое) Структуры слоя бизнес-логики	135
	Приложение Ж (рекомендуемое) Структуры слоя доступа к данным	140
	Приложение И (рекомендуемое) Листинг модульных тестов	144

Введение

С развитием технологий информационные системы стали неотъемлемой частью многих сфер деятельности. Эти системы создаются для эффективного управления данными: их хранения, обработки и поиска. Они опираются на организационные ресурсы, которые обеспечивают доступ к информации и её распространение. Основным результатом их работы – это создание информационных продуктов, таких как базы данных, IT-услуги, документы и массивы данных [1].

В современных условиях развития жилищно-коммунального хозяйства (ЖКХ) вопросы эффективного учета и контроля потребления электроэнергии приобретают особую актуальность. Электроэнергия является одним из ключевых ресурсов, обеспечение оплаты которого требует строгого учета, своевременного выявления нарушений потребления и контроля за состоянием приборов учета.

Существующие методы работы инспекторов энергосбытовых компаний, основанные на бумажном документообороте и ручной обработке данных, перестают отвечать требованиям оперативности и достоверности. Ручное заполнение актов, отсутствие оперативной связи с диспетчерской, сложности с фотофиксацией нарушений и долгая обработка результатов проверок приводят к ошибкам, потере данных и снижению общей эффективности работы отдела контроля.

В связи с этим возникает необходимость разработки специализированных программных средств, обеспечивающих автоматизацию деятельности инспекторов и ведение данных в единой информационной системе.

Целью выпускной квалификационной работы является разработка серверного приложения информационной системы контроля и анализа потребления электроэнергии, обеспечивающей автоматизацию ключевых бизнес-процессов отдела контроля энергосбытовой организации. Работа включает в себя анализ предметной области, обзор существующих аналогов,

анализ функциональных и нефункциональных требований к программным средствам, проектирование, реализацию и контроль качества программного обеспечения.

Для разработки системы используются современные технологии и инструменты: язык программирования Go обеспечивает высокую производительность и надежность системы; объектно-реляционная СУБД PostgreSQL используется для хранения структурированной информации; колоночная СУБД ClickHouse – для эффективной обработки аналитических запросов; брокер сообщений Apache Kafka обеспечивает надежную передачу сообщений между сервисами; объектное хранилище MinIO отвечает за хранение файлов с возможностью масштабирования; а Prometheus и Grafana обеспечивают наблюдаемость системы с помощью метрик.

1 Анализ предметной области и постановка задачи на разработку

1.1 Анализ предметной области

1.1.1 Понятие контроля и анализа потребления электроэнергии

Контроль потребления электроэнергии представляет собой комплекс мероприятий, включающих мониторинг, измерение и учет фактических расходов электрической энергии потребителями в соответствии с нормативами и договорами между абонентами и энергосбытовыми организациями. Комплекс мероприятий включает в себя технические и организационные мероприятия.

В энергосбытовой области контроль потребления электроэнергии является ключевым процессом по обеспечению справедливости расчетов между абонентами и энергосбытовыми организациями. В его задачи входит выявление несанкционированного потребления электроэнергии, проверка соблюдения ограничений абонентами и сбор доказательной базы нарушений в потреблении электроэнергии.

Автоматизированная система контроля учета электроэнергии — это система, которая позволяет контролировать и учитывать количество отпущенной и потребленной электроэнергии (мощности, энергоресурсов) на разных объектах. Она используется для взаиморасчетов за электроэнергию между поставщиками и потребителями, для оптимизации энергопотребления и для предотвращения хищений электроэнергии [2].

Анализ потребления электроэнергии является аналитическим процессом по обработке и изучению данных, полученных при выполнении мероприятий контроля и учета электроэнергии. Он направлен на своевременное выявление аномалий, расчет ключевых показателей и прогнозирование расходов электроэнергии. Анализ включает в себя статистическую обработку показаний приборов учета, сопоставление фактического потребления электроэнергии и плановых значений, а также выявление закономерностей и трендов.

1.1.2 Описание предметной области и ее участников

Предметной областью разрабатываемой информационной системы является автоматизация процессов отдела контроля и анализа потребления электроэнергии энергосбытовой компании.

В задачи отдела входит:

- обеспечение достоверного учета отпущенной электроэнергии;
- выявление нарушений потребления и несанкционированных вмешательств в работу приборов учета;
- контроль технического состояния приборов учета;
- ограничение энергоснабжения при выявлении нарушений;
- контроль соблюдения введенных ограничений;
- документальная фиксация произведенных мероприятий.

Рассматриваемый отдел представляет собой многоуровневую организационную структуру, включающую в себя три основных категории участников предметной области.

Инспектор – сотрудник, выполняющий выезды на объекты энергоснабжения в целях проверки и контроля потребления электроэнергии. В его обязанности входит:

- получение задач от диспетчера с назначенными объектами, их адресами и информацией об абоненте;
- прибытие на объект к установленному времени;
- визуальный осмотр приборов учета электроэнергии и средств визуального контроля (пломб);
- снятие текущих показаний приборов учета;
- проверка целостности пломб;
- фотофиксация приборов учета, их показаний и пломб;
- выявление фактов несанкционированного доступа и иных нарушений;

- составление актов о проведенных мероприятиях на основе полученных данных;
- загрузка документов и фотографий в систему.

Инспекторы работают в составе бригад по два человека в каждой. Такое решение обусловлено требованиями техники безопасности и нормативными актами энергосбытовой организации.

Диспетчер – сотрудник, занимающийся управлением бригадами инспекторов и распределением задач. В зону его ответственности входит:

- планирование деятельности инспекторов на основе перечня необходимых проверок;
- формирование и управление составом бригад инспекторов;
- эффективное распределение задач между бригадами;
- отслеживание и контроль за выполнением задач;
- формирование отчетов о выполненных работах за определенный период (день, месяц, квартал);
- анализ эффективности бригад инспекторов.

Специалист по работе с абонентами – сотрудник, отвечающий за ведение реестра и связь с абонентами. Его функции включают:

- загрузку и актуализацию реестра абонентов;
- ведение информации о договорах энергоснабжения;
- обновление данных об объектах энергоснабжения на основе данных с проверок инспекторов;
- формирование списков нарушителей и лиц с задолженностью;
- подготовку уведомлений абонентам о выявленных нарушениях.

Абонент (потребитель электроэнергии) – владелец объекта энергоснабжения, на котором установлены приборы учета со средствами визуального контроля. Энергоснабжающая компания заключает с ним договор о предоставлении услуг энергоснабжения объекта. Абонент напрямую не взаимодействует с системой, но его данные обрабатываются внутри нее.

1.1.3 Организационная структура информационных потоков

В общем случае процесс контроля и анализа потребления электроэнергии выглядит как последовательное выполнение следующих действий:

- подготовка к проверкам;
- выполнение проверок;
- обработка и анализ результатов;
- архивирование и статистический анализ.

Специалист по работе с абонентами формирует реестр абонентов, который содержит в себе всю связанную с ними информацию: ФИО абонента, номер договора, объекты энергоснабжения, информация о приборах учета и т.д. На основе этих данных диспетчер создает список задач на проверку объектов и планирует примерную дату и время этих проверок. Затем он распределяет эти задачи между свободными бригадами инспекторов.

Инспекторы получают список назначенных им задач, прибывают на объекты согласно плану и выполняют проверку приборов учета. В процессе проверки инспекторы выполняют фотофиксацию показаний приборов учета и состояния средств визуального контроля. В результате бригада формирует акт по одной из форм энергосбытовой организации. В конце рабочего дня инспекторы выгружают фотографии на сервер и переименовывают их по шаблону. Сформированные акты отправляются в энергосбытовую компанию.

Диспетчер формирует отчеты на основе полученных данных и анализирует эффективность работы бригад. Выявленные нарушения обрабатываются в особом порядке, фотографии проверяются на искажения и размытость, список нарушителей передается специалисту по работе с абонентами для дальнейшего их уведомления.

1.1.4 Документооборот предметной области

В процессе работы отдела создаются следующие основные документы:

- акт о введении ограничения (приостановлении) коммунальной услуги по электроснабжению – составляется при выявлении оснований для ограничения энергоснабжения (задолженность, нарушение договора, несанкционированное подключение);
- акт о возобновлении предоставления коммунальной услуги по электроснабжению – составляется при устранении абонентом причин ограничения;
- акт об осуществлении проверки введенного ограничения режима потребления – составляется при дежурной проверке объекта энергоснабжения и отсутствии нарушений;
- акт о самовольном подключении к электрическим сетям – составляется при выявлении фактов несанкционированного подключения абонента к электрическим сетям;
- фотографии приборов учета и пломб – визуальная документация состояния оборудования и нарушений, фиксируемая бригадой инспекторов для подтверждения выявленных фактов;
- отчет о проделанной работе – ежедневный отчет, формируемый диспетчером на основе выполненных задач.

Все документы хранятся на сервере в виде обычных файлов и обрабатываются вручную.

1.1.5 Проблемы в текущем бизнес-процессе

Подробный анализ текущего бизнес-процесса отдела контроля и анализа потребления электроэнергии выявил ряд проблем.

Ручное заполнение наименований файлов фотографий приборов учета имеет повышенный риск человеческого фактора ошибки. Инспекторы неправильно именуют файлы или вовсе игнорируют требования.

При выполнении фотофиксации отсутствует контроль качества фотографий. По этой причине возникает возможность загрузки дефектных фотографий, из-за которых информация может быть искажена или утеряна.

Бригады инспекторов вручную заполняют акты проверок. Это существенно увеличивает время проведения проверочных мероприятий и влияет на количество ошибок при составлении актов. Неэффективное распоряжение ресурсами инспекторов ведет к увеличению затрат энергосбытовой компании.

Ежедневное составление отчетов создает дополнительную нагрузку на диспетчера. Их формирование и выгрузка на сервер занимает большое количество времени, которое можно было бы потратить на более приоритетные задачи.

У диспетчера отсутствует актуальная информация о бригадах инспекторов и задачах, над которыми они работают. Это снижает прозрачность всего бизнес-процесса. У инспекторов появляется возможность для злоупотребления этим, что снижает общую эффективность отдела.

Специалист по работе с абонентами получает информацию о нарушителях только на следующий рабочий день, что снижает скорость доставки уведомлений до абонентов.

Все перечисленные проблемы напрямую влияют на эффективность работы отдела, а, следовательно, и на всю энергосбытовую организацию. Наличие данных проблем негативно сказывается на прибыли компании и показывает, насколько актуальным может быть решение.

1.2 Анализ информационных систем аналогичного назначения

Проведем обзор и анализ существующих программных продуктов со схожим функционалом. На рынке есть несколько аналогов, отечественных и зарубежных. По результатам анализа будут выявлены преимущества и недостатки каждого рассмотренного решения.

1.2.1 Платформа СТЕК

Платформа СТЕК представляет собой отечественное решение для энергосбытовых организаций. Данная платформа позиционирует себя как

комплексное решение автоматизации бизнес-процессов по учету и продаже абонентам электроэнергии. Логотип компании представлен на рисунке 1.



Рисунок 1 – Логотип компании СТЕК

СТЕК реализует учет электроэнергии и расчеты за нее, интеграцию с внешними и внутренними системами, расчеты и взаимодействие с поставщиками услуг. Компания также сообщает об организации взаимодействия с потребителями непосредственно в программе для энергосбыта.

В качестве преимуществ данного решения можно выделить полный цикл расчетов за электроэнергию и интеграцию с платежными системами. Абонентам предоставлена удобная возможность взаимодействия с энергосбытовой организацией посредством платформы. Продукт сосредоточен в основном на финансовом аспекте проблем, возникающих у компании.

Это является и недостатком платформы в контексте требуемой системы. Платформа СТЕК специализируется на обработке платежей и управлении дебиторской задолженностью, однако не располагает специализированным решением для организации работы бригад инспекторов. Таким образом, она закрывает только часть проблем отдела контроля и анализа потребления электроэнергии и не подходит для внедрения в энергосбытовую организацию.

1.2.2 Программно-аппаратный комплекс АСУ ПКПЭ

ПАК АСУ ПКПЭ решает большой комплекс задач, связанных с деятельностью ТЭ по учету, контролю потребления электроэнергии,

формированию множества отчетных документов и форм, в том числе и управленческой отчетности. В подсистеме управленческой отчетности реализована работа с бюджетами ТЭ и НТЭ, контролем их исполнения, планирование доходов от передачи электроэнергии, расчет индекса технического состояния основного технологического оборудования подстанций ОАО «РЖД», анализ тарифов на электроэнергию, контроль потерь электроэнергии и другие функции [3]. Логотип института-разработчика изображен на рисунке 2.



Рисунок 2 – Логотип научно-исследовательского и проектно-конструкторского института информатизации, автоматизации и связи на железнодорожном транспорте

В качестве преимуществ программно-аппаратного комплекса можно выделить автоматизацию управленческой отчетности, возможность контроля потерь электроэнергии и удобное ведение бюджета. Но данное решение направлено в первую очередь на крупные предприятия, для небольших организаций оно не подходит. Также стоит отметить отсутствие мобильного приложения для проведения проверок и документирования нарушений.

1.2.3 Платформа SafetyCulture

SafetyCulture – это зарубежная платформа для управления рабочим процессом. Ее универсальность позволяет интегрироваться в разные сферы промышленности, в том числе и в энергосбытовые организации. Интерфейс платформы изображен на рисунке 3.

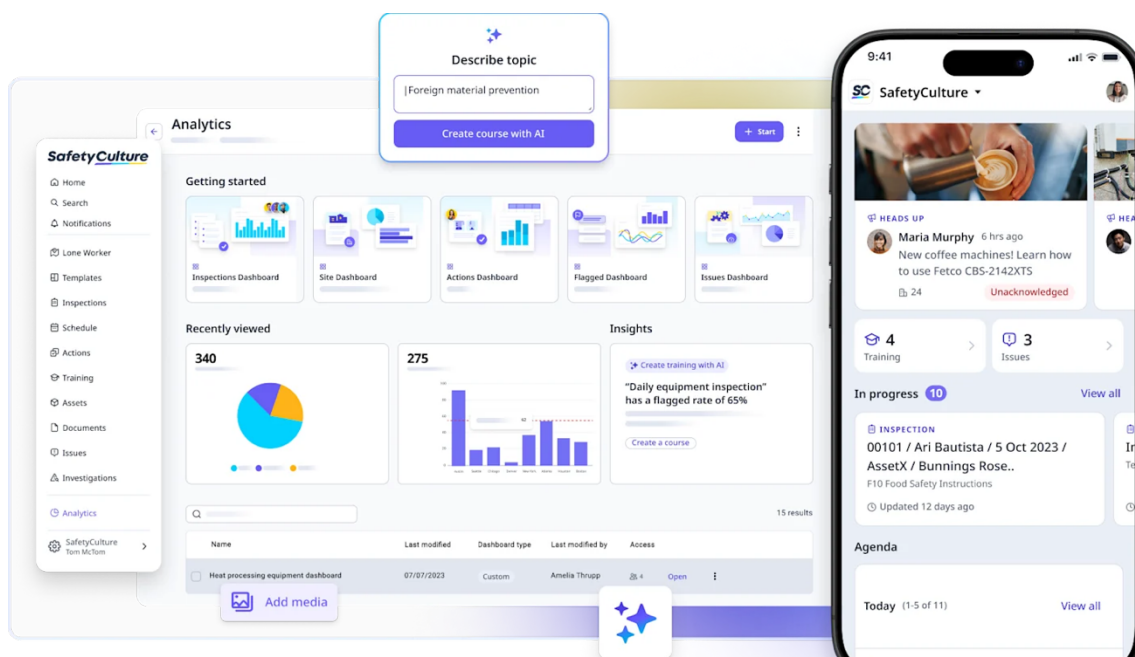


Рисунок 3 – Интерфейс платформы SafetyCulture

Платформа имеет мобильное приложение, через которое инспектор может взаимодействовать с проверками и чеклистами в оффлайн-режиме. Удобный пользовательский интерфейс упрощает использование платформы, а готовые шаблоны проверок позволяют экономить время.

Но универсальность платформы создает дополнительные издержки. Отсутствие специализации на энергосбыте заметно сказывается на функционале приложений. Например, необходимость взаимодействия с реестром абонентов и приборами учета никак не предусмотрена разработчиками.

Для проведения сравнительного анализа было выделено несколько ключевых критериев, которым должна соответствовать система. Данные

критерии нужны для оценки аналогов разрабатываемой системы. Для оценки каждого из критериев была предложена следующая система оценивания:

- 0 баллов – полностью не соответствует критерию;
- 1 балл – частично соответствует критерию;
- 2 балла – полностью соответствует критерию.

Результаты проведения сравнительного анализа приведены в таблице 1.

Таблица 1 – Сравнительный анализ информационных систем аналогичного назначения

Критерий	Платформа СТЕК	ПАК АСУ ПКПЭ	Платформа SafetyCulture	Разрабатываемая система
Автоматическая генерация актов на основе данных с проверки	1	0	1	2
Фотофиксация приборов учета и средств визуального контроля	0	0	1	2
Формирование отчетов о проделанной работе	2	2	2	2
Анализ фотографий на искажения	0	0	0	2
Мониторинг работы бригад инспекторов в реальном времени	2	1	2	2
Итого	5	3	6	10

Таким образом, по итогам сравнительного анализа, ни одна рассмотренная информационная система аналогичного назначения не соответствует критериям и требованиям отдела контроля и анализа потребления электроэнергии энергосбытовой организации. Следовательно, разрабатываемая в данной выпускной квалификационной работе система актуальна.

1.3 Постановка задачи на разработку программных средств

1.3.1 Общая концепция системы

В рамках выпускной квалификационной работы необходимо разработать серверное приложение информационной системы контроля и анализа потребления электроэнергии. Оно относится к категории корпоративных информационных систем, предназначенных для автоматизации процессов отдела контроля и анализа потребления электроэнергии в энергосбытовых компаниях.

Целевой аудиторией информационной системы являются сотрудники отдела контроля и анализа потребления электроэнергии – инспекторы, диспетчеры и специалисты по работе с абонентами.

На основе проведенного анализа предметной области система должна удовлетворять следующие ключевые потребности пользователей:

- проверка показаний приборов учета электроэнергии и целостности средств визуального контроля;
- фиксация нарушений и фактов несанкционированного вмешательства в работу приборов учета;
- автоматизированное формирование актов и отчетов;
- анализ динамики потребления электроэнергии и выявление аномальных отклонений;
- предоставление диспетчеру сводного обзора операций, задач, проверок и результатов работы бригад;
- управление ресурсами инспекторов и бригад;
- обеспечение оперативного взаимодействия бригад инспекторов с диспетчерской.

Удовлетворение вышеописанных потребностей позволит повысить удобство работы пользователей, достоверность данных и оперативность выполнения контрольных мероприятий.

1.3.2 Основные функции и преимущества системы

Взяв за основу результаты анализа предметной области и выявленные потребности пользователей системы, можно выделить ее основные функции:

- формирование актов установленного образца;
- фотофиксация приборов учета и средств визуального контроля;
- формирование отчетов о проделанной работе;
- аналитика потребления электроэнергии, выявление аномалий и обзор операций отдела контроля.

Серверное приложение информационной системы должно предоставлять возможность автоматической генерации актов на основе данных с проверки объекта энергоснабжения. Всего существует четыре типа актов:

- акт о введении ограничения;
- акт о возобновлении энергоснабжения;
- акт о самовольном подключении;
- акт о проверке ограничения.

Фотографии приборов учета и средств визуального контроля должны надежно храниться в системе, а также привязываться к соответствующей проверке. При сохранении файлы фотографий должны автоматически переименовываться в соответствии с шаблоном.

Система должна иметь возможность составления отчетов о выполненных работах на основе агрегированных данных с проверок. Отчеты можно формировать за любой период, и они должны содержать информацию о выполненных задачах, времени их выполнения и выявленных нарушениях.

Отдельной ключевой функцией является аналитический контур системы. Он предназначен не только для выгрузки отчетных документов, но и для оперативного анализа накопленных данных. На основе сведений о проверках, показаниях приборов учета, выявленных нарушениях и статусах задач формируются аналитические представления для веб-портала: дашборд аномалий потребления и дашборд обзора операций. Дашборд аномалий

потребления позволяет выделять объекты энергоснабжения с нетипичной динамикой расхода, резкими изменениями показаний или повторяющимися признаками нарушений. Дашборд обзора операций отражает состояние текущей работы отдела контроля: количество задач по статусам, загрузку бригад, число завершенных проверок, долю выявленных нарушений и динамику выполнения работ за выбранный период.

Наличие аналитических дашбордов позволяет диспетчеру и специалистам по работе с абонентами не ограничиваться просмотром отдельных задач и отчетов, а видеть обобщенную картину работы системы. Благодаря этому информационная система выполняет не только учетную, но и аналитическую функцию, что соответствует назначению системы контроля и анализа потребления электроэнергии.

Основные преимущества разрабатываемой системы:

- автоматизация документооборота – сокращение времени на составление актов и отчетов и минимизация ошибок при их заполнении;
- ускорение проведения проверок – бригада может сосредоточиться на самой проверке и не отвлекаться на ручное заполнение актов;
- прозрачность работы бригад инспекторов – диспетчер видит статус выполнения задач в реальном времени и может оперативно перераспределить нагрузку;
- улучшение качества доказательной базы – фотографии приборов учета и средств визуального контроля предварительно автоматически проверяются на искаженность и размытость;
- аналитика и отчетность – система накапливает данные для выявления тенденций, обнаружения аномалий потребления и контроля операций отдела.

Обладая этими преимуществами, система сильно облегчит и ускорит работу отдела контроля и анализа потребления электроэнергии.

1.3.3 Отличия от текущего бизнес-процесса

Анализ существующего бизнес-процесса отдела выявил ряд проблем. В таблице 2 предложены решения данных проблем с помощью разрабатываемой системы.

Таблица 2 – Проблемы в текущем бизнес-процессе и их решения

Проблема в текущем процессе	Решение в разрабатываемой системе
Инспекторы допускают ошибки в именовании фотографий или игнорируют требования по их переименованию	Автоматическая генерация названия файла по единому шаблону с привязкой к проверке
Низкое качество фотографий, размытые или искаженные изображения приборов учета и средств визуального контроля	Автоматический анализ загруженных фотографий на предмет искажений и размазанности; система может отклонить некачественные снимки
Трудоемкое ручное заполнение актов на объекте приводит к потере времени	Автоматическая генерация актов на основе введенных данных; бригада инспекторов вводит данные, остальную работу выполняет система
Отсутствие оперативного контроля – диспетчер узнает о проблеме с задержкой	Система предоставляет в реальном времени информацию о статусе каждой задачи и бригады
Ручное формирование отчетов и их выгрузка в сеть требуют много времени и внимания	Полностью автоматизированная генерация отчетов и выгрузка документов и фотографий
Отсутствие единого аналитического представления не позволяет быстро обнаруживать аномалии потребления и оценивать состояние операций отдела	Дашборды аналитики показывают объекты с нетипичной динамикой потребления, распределение задач по статусам, загрузку бригад и результаты проверок за выбранный период
Информация из реестра абонентов быстро устаревает	База данных автоматически обновляется в соответствии с новыми данными с проверок
Инспекторы могут отклоняться от маршрута или игнорировать обязанности	Автоматическая фиксация времени начала и окончания задачи для контроля и дальнейшего анализа и оптимизации

В результате разработки должно быть создано серверное приложение информационной системы контроля и анализа потребления электроэнергии, удовлетворяющее ключевые потребности и решающее основные проблемы выявленной целевой аудитории. Таким образом, разработанное приложение станет универсальным инструментом, который повысит скорость работы, уменьшит затраты и издержки, а также улучшит прозрачность внутри отдела

контроля и анализа потребления электроэнергии. Совокупность этих факторов повысит прибыль энергосбытовой организации.

1.4 Функционально-стоимостной анализ

Функционально-стоимостной анализ (ФСА) – это метод исследования системы, позволяющий выявить возможные точки снижения затрат с помощью анализа функционала системы и его стоимости.

Основная идея, лежащая в основе ФСА, состоит в том, что покупатель приобретает не товар, а, скажем так, набор функций, который может быть реализован с его помощью. Аналогичным образом, в сфере информационных технологий пользователь приобретает не установочный файл с красивой иконкой, а функционал, призванный решить определенные задачи [4].

Повысить ценность продукта возможно двумя способами: снижением себестоимости и затрат на его реализацию, а также увеличением количества полезных для пользователя функций.

Выполним функционально-стоимостной анализ, чтобы узнать рациональность и стоимость разрабатываемого серверного приложения информационной системы. Для этого оценим продолжительность и стоимость каждого этапа реализации проекта. Чтобы оценка была более точной, проанализируем зарплаты программистов-разработчиков. Диаграмма зарплат программистов-разработчиков в Пензе за 2025 год представлена на рисунке 4 [5].

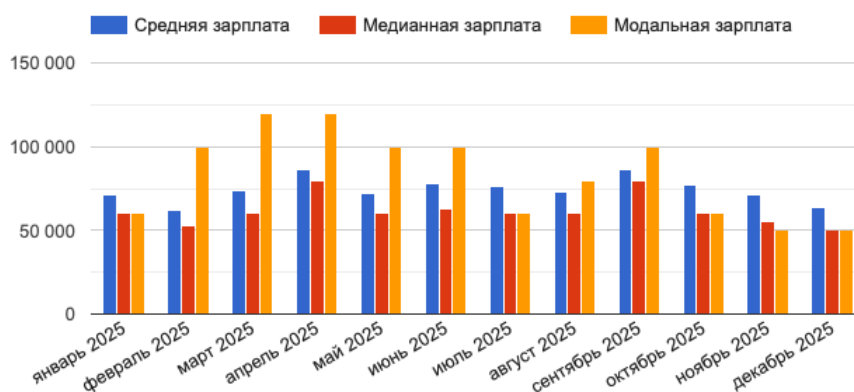


Рисунок 4 – Диаграмма зарплат программиста-разработчика в Пензе

Исходя из диаграммы, в среднем зарплата программиста-разработчика в Пензе составляет 74 195 рублей в месяц. С учетом стандартной 40-часовой рабочей недели, за час программист-разработчик в среднем получает 460 рублей. Возьмем эти данные за основу ФСА.

Следующим шагом ФСА будет декомпозиция проекта на этапы. Декомпозиция требований и процессов проектирования при создании конкретных компонентов и программных продуктов основывается на разбиении общей цели проекта на несколько промежуточных целей и этапов, каждый из которых также можно разделить [6]. В результате декомпозиции были выделены основные этапы:

- анализ предметной области;
- анализ требований к ПС;
- проектирование ПС;
- разработка ПС;
- контроль качества;
- оформление ПЗ.

Для удобства проведения ФСА данные о проекте были занесены в приложение Microsoft Project. С помощью данной программы был составлен план разработки проекта, представленный на рисунке 5.

	Режим задачи	Название задачи	Длительность	Начало	Окончание	Предшественн
1		Подготовка ВКР	50 дней	Пн 06.04.26	Пт 12.06.26	
2		Анализ предметной области	7 дней	Пн 06.04.26	Вт 14.04.26	
3		Анализ требований к ПС	4 дней	Ср 15.04.26	Пн 20.04.26	2
4		Проектирование ПС	10 дней	Вт 21.04.26	Пн 04.05.26	3
5		Разработка ПС	21 дней	Вт 05.05.26	Вт 02.06.26	4
6		Контроль качества	5 дней	Ср 03.06.26	Вт 09.06.26	5
7		Оформление ПЗ	3 дней	Ср 10.06.26	Пт 12.06.26	6

Рисунок 5 – План разработки проекта

Также с помощью данной программы была составлена диаграмма Ганта, представленная на рисунке 6.

2 Анализ требований к разрабатываемым средствам

2.1 Анализ функциональных требований

Функциональные требования (functional requirements) определяют, каким должно быть поведение продукта в тех или иных условиях. Они определяют, что разработчики должны создать, чтобы пользователи смогли выполнить свои задачи (пользовательские требования) в рамках бизнес-требований [7].

В рамках выпускной квалификационной работы необходимо реализовать серверное приложение информационной системы контроля и анализа потребления электроэнергии. Оно должно сосредоточить в себе весь необходимый функционал и стать ядром отдела контроля и анализа потребления электроэнергии. Целью этого решения является автоматизация процессов и снижение издержек в работе энергосбытовых организаций.

Выделим функциональные требования к разрабатываемому приложению:

- получение списка задач с актуальными статусами;
- отображение полной информации о задаче (адрес объекта энергоснабжения, ФИО абонента, запланированное время);
- изменение статуса задачи;
- ввод и проверка достоверности результатов проверки в систему;
- интеграция с системой распознавания показаний прибора учета на фото (OCR);
- автоматическое формирование всех видов актов на основе введенных данных в формате DOCX;
- поддержка загрузки фотографий приборов учета и средств визуального контроля в форматах JPG и PNG;
- автоматическая проверка качества изображения при загрузке;
- автоматическое переименование файлов по шаблону;
- агрегирование данных о выполненных задачах и проверках;

- формирование отчетов за различные периоды на основе агрегированных данных;
- получение данных для дашборда аномалий потребления;
- получение данных для дашборда обзора операций отдела контроля.

Диаграмма вариантов использования (use case diagram) – диаграмма, показывающая набор вариантов использования, действующих лиц и их связей. Относится к статическому представлению системы с точки зрения вариантов использования [8].

Рассмотрим функциональные требования к системе на диаграммах вариантов использования. Для каждой категории пользователей была разработана отдельная диаграмма.

Диаграмма вариантов использования для инспектора представлена на рисунке 7.

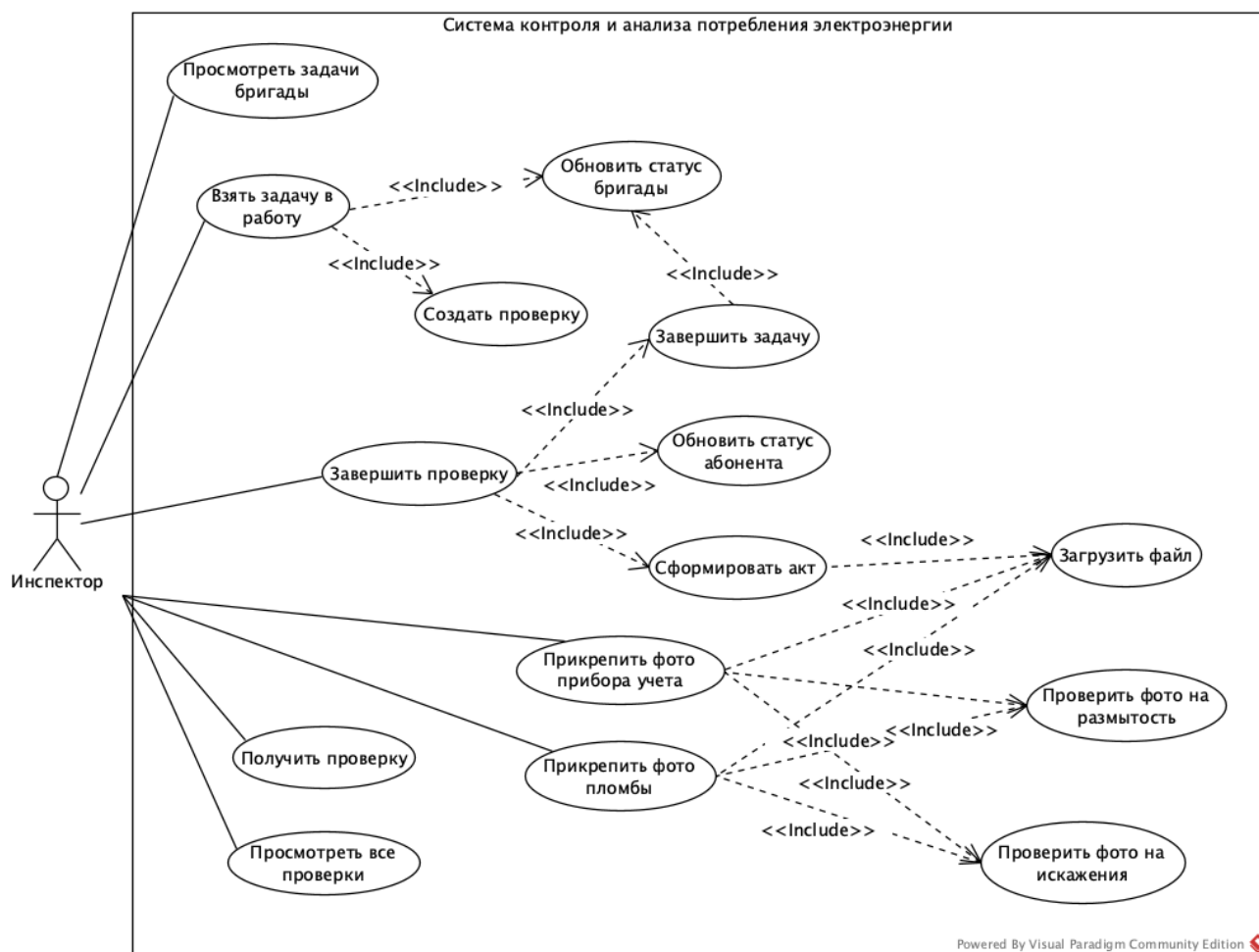


Рисунок 7 – Диаграмма вариантов использования для актера Инспектор

Диаграмма вариантов использования для диспетчера представлена на рисунке 8.

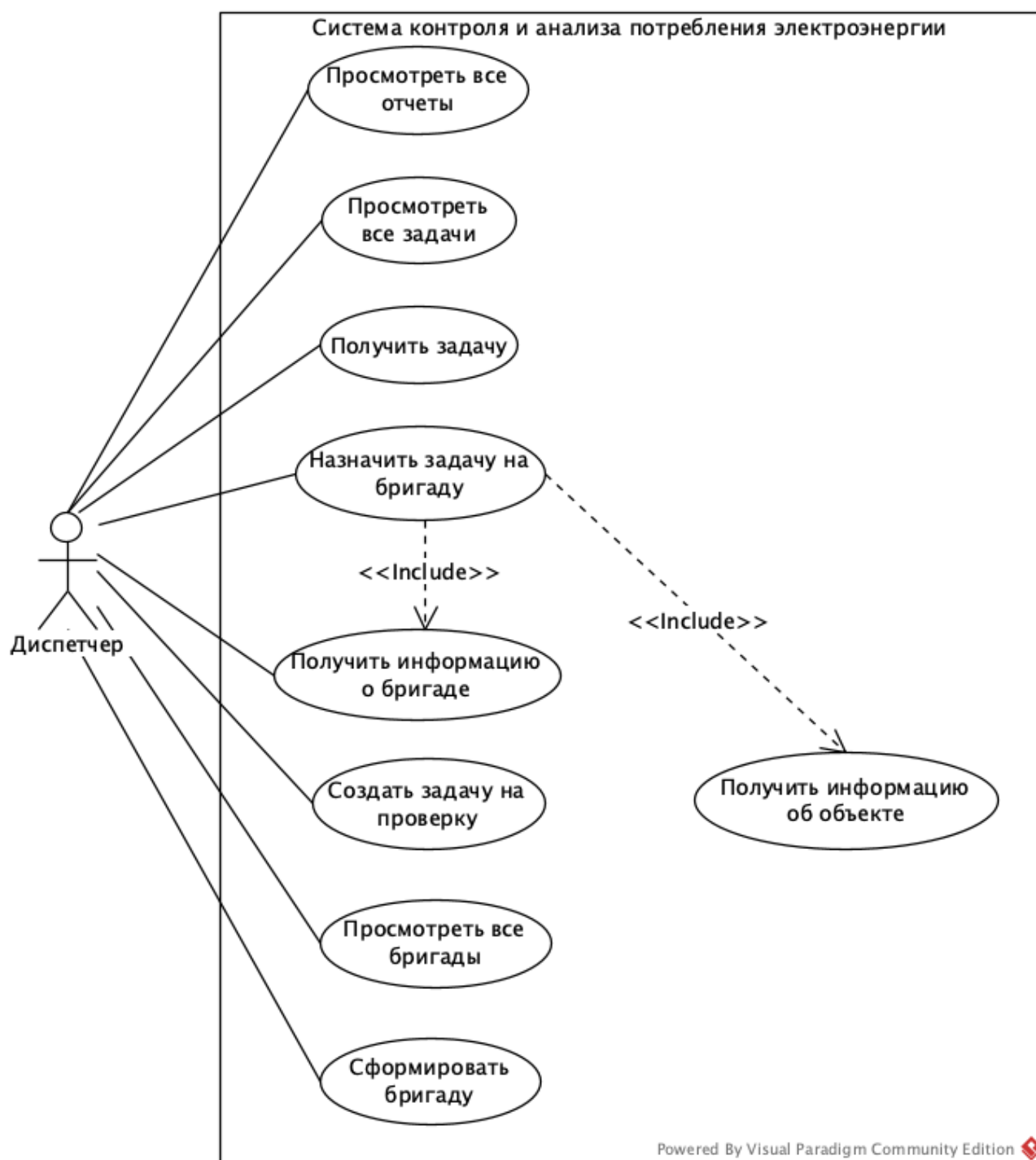


Рисунок 8 – Диаграмма вариантов использования для актера Диспетчер

Диаграмма вариантов использования для специалиста по работе с абонентами представлена на рисунке 9.



Рисунок 9 – Диаграмма вариантов использования для актера Специалист по работе с абонентами

На рисунке 10 приведена спецификация варианта использования «Взять задачу в работу».

Наименование: взять задачу в работу
ID: 1
Краткое описание: инспектор выбирает задачу и берет ее в работу
Главные актеры: инспектор, система
Второстепенные актеры: нет
Предусловия: 1. Список задач получен. 2. Инспектор выбрал задачу. 3. Задача в статусе «Запланирована».
Основной поток: 1. Инспектор выбирает задачу и нажимает «Взять в работу». 2. Система проверяет доступность задачи. 3. Система обновляет статус задачи на «В работе». 4. Система автоматически обновляет статус бригады на «На задаче». 5. Система автоматически создает проверку в статусе «В работе». 6. Система отображает подтверждение.
Постусловия: задача в работе бригады
Альтернативные потоки: 2а. Задача не прошла проверку, система выдает сообщение об ошибке.

Рисунок 10 – Описание сценария варианта использования «Взять задачу в работу»

На рисунке 11 приведена спецификация варианта использования «Прикрепить фото прибора учета».

Наименование: прикрепить фото прибора учета
ID: 2
Краткое описание: инспектор прикрепляет фотографию прибора учета или пломбы к проверке
Главные актеры: инспектор, система
Второстепенные актеры: нет
Предусловия: 1. Задача в статусе «В работе». 2. Проверка есть в системе. 3. Проверка в статусе «В работе». 4. У инспектора есть доступ к приложению.
Основной поток: 1. Инспектор фотографирует прибор учета. 2. Инспектор выбирает опцию «Загрузить фото». 3. Система принимает и проверяет фото. 4. Система сохраняет фото. 5. Система отображает успешное сохранение.
Постусловия: фотография прибора учета прикреплена к проверке
Альтернативные потоки: 3а. Фотография не прошла проверку, система выдает сообщение об ошибке и предлагает повторить попытку.

Рисунок 11 – Описание сценария варианта использования «Прикрепить фото прибора учета»

На рисунке 12 приведена спецификация варианта использования «Сформировать ежедневный отчет о работе бригад».

Наименование: сформировать ежедневный отчет о работе бригад
ID: 3
Краткое описание: система ежедневно формирует отчет о работе бригад
Главные актеры: система
Второстепенные актеры: нет
Предусловия: в системе есть данные о задачах и проверках
Основной поток: 1. Система начинает формирование отчета в определенное время. 2. Система собирает данные о выполненных задачах за день. 3. Система формирует отчет в формате Excel. 4. Система сохраняет сформированный отчет.
Постусловия: сформирован ежедневный отчет
Альтернативные потоки: нет

Рисунок 12 – Описание сценария варианта использования «Сформировать ежедневный отчет о работе бригад»

На рисунке 13 приведена спецификация варианта использования «Назначить задачу на бригаду».

Наименование: назначить задачу на бригаду
ID: 4
Краткое описание: диспетчер выбирает задачу и назначает ее на бригаду инспекторов
Главные актеры: диспетчер, система
Второстепенные актеры: нет
Предусловия: 1. Список задач получен. 2. Диспетчер выбрал задачу. 3. Задача не в статусе «В работе».
Основной поток: 1. Диспетчер выбирает задачу и нажимает «Назначить бригаду». 2. Система проверяет доступность задачи. 3. Диспетчер выбирает бригаду из списка доступных бригад. 4. Система привязывает задачу к выбранной бригаде. 5. Система отображает подтверждение.
Постусловия: задача назначена на бригаду инспекторов
Альтернативные потоки: 2а. Задача не прошла проверку, система выдает сообщение об ошибке.

Рисунок 13 – Описание сценария варианта использования «Назначить задачу на бригаду»

На рисунке 14 приведена спецификация варианта использования «Загрузить реестр из Excel».

Наименование: загрузить реестр из Excel
ID: 5
Краткое описание: специалист по работе с абонентами загружает реестр абонентов в систему
Главные актеры: специалист по работе с абонентами, система
Второстепенные актеры: нет
Предусловия: реестр сформирован виде Excel-файла соответствующего формата
Основной поток: 1. Специалист по работе с абонентами загружает файл с реестром в систему. 2. Система проверяет правильность составления реестра. 3. Система считывает информацию об абонентах из реестра и обновляет информацию в базе данных. 4. Система считывает информацию об объектах энергоснабжения из реестра и обновляет информацию в базе данных. 5. Система считывает информацию о приборах учета из реестра и обновляет информацию в базе данных. 6. Система считывает информацию о средствах визуального контроля из реестра и обновляет информацию в базе данных. 7. Система считывает информацию о договорах из реестра и обновляет информацию в базе данных.
Постусловия: база данных системы обновлена информацией из реестра
Альтернативные потоки: 2а. Реестр не прошел проверку, система выдает сообщение об ошибке.

Рисунок 14 – Описание сценария варианта использования «Загрузить реестр из Excel»

Приведенные диаграммы вариантов использования точно отражают функциональные требования, которые были предъявлены к серверному приложению.

2.2 Анализ нефункциональных требований

Нефункциональное требование – это описание свойства или особенности, которым должна обладать система, или ограничение, которое должна соблюдать система [6].

Учитывая, что соблюдение нефункциональных требований существенно влияет на проектирование архитектуры системы, перечислим основные

характеристики системы, основанные на эксплуатационных, структурных и сквозных свойствах архитектуры [9]:

- серверное приложение информационной системы должно оперативно обрабатывать большие объемы данных о задачах, проверках и абонентах и быстро формировать все возможные варианты документов (производительность);
- сбои в одной части приложения никаким образом не должны влиять на работоспособность другой его части (надежность и безопасность);
- система должна быть готова к повышению нагрузки из-за сезонности или кратного прироста абонентов энергосбытовой организации (масштабируемость);
- система должна обеспечивать простоту добавления новых типов документов без необходимости существенной переработки существующих модулей (сопровожаемость);
- программные средства должны использовать средства аутентификации для проверки подлинности личности пользователя (аутентифицируемость);
- система должна контролировать доступ к определенному функционалу на основе системы ролей пользователей (авторизируемость).

Перечисленные нефункциональные требования послужат основой для проектирования серверного приложения информационной системы контроля и анализа потребления электроэнергии и обеспечат высокое качество разработанного программного обеспечения.

3 Проектирование программных средств

3.1 Общая архитектура программных средств

На этапе анализа требований к разрабатываемым средствам были выявлены функциональные и нефункциональные требования. В соответствии с анализом этих требований было принято решение использовать микросервисный архитектурный стиль при проектировании и реализации системы. Микросервисы – это независимо выпускаемые сервисы, которые моделируются вокруг предметной области бизнеса. Сервис инкапсулирует функциональность и делает ее доступной для других сервисов через сети [10].

В результате проектирования информационной системы контроля и анализа потребления электроэнергии была создана диаграмма компонентов. Диаграмма компонентов – это структурная диаграмма языка унифицированного моделирования, она описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами [11]. Также данная диаграмма упрощает понимание структуры и взаимосвязей системы и в целом представляет собой общую картину архитектуры системы.

При выделении компонентов серверного приложения учитывались границы предметных областей системы. Такой подход позволяет снизить связность между частями системы и упростить дальнейшее сопровождение программного комплекса. Каждый компонент отвечает за ограниченный набор функций и взаимодействует с другими компонентами через явно определенные программные интерфейсы. Благодаря этому изменение логики одного сервиса не требует полной переработки остальных частей системы, что повышает масштабируемость и сопровождаемость разрабатываемого приложения.

Диаграмма компонентов серверного приложения информационной системы контроля и анализа потребления электроэнергии представлена на рисунке 15.

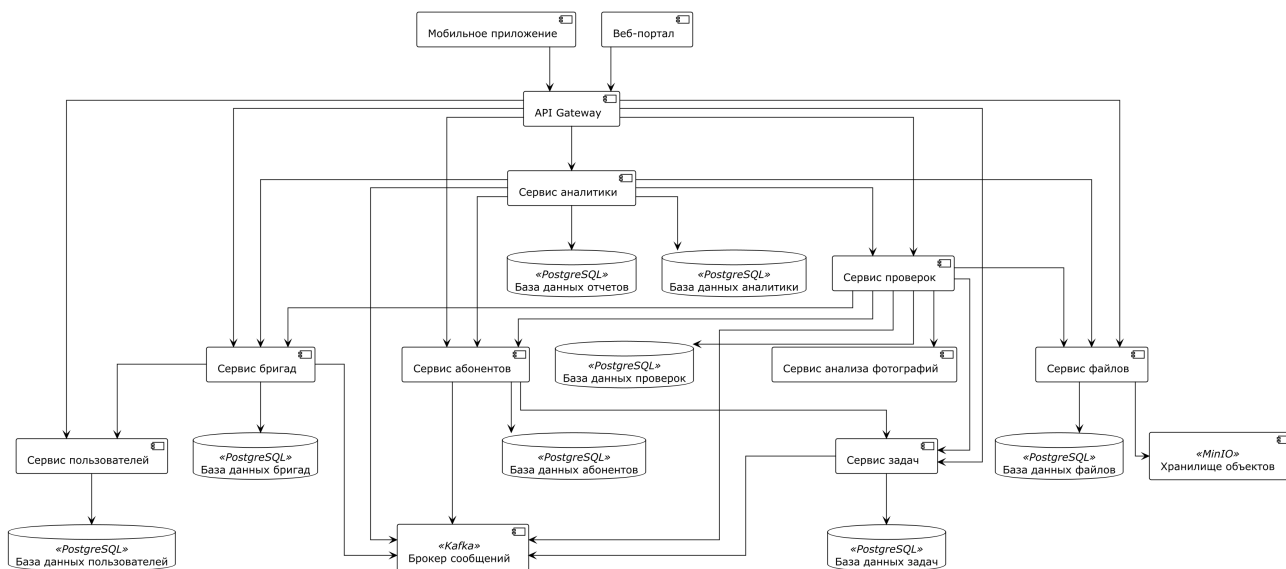


Рисунок 15 – Диаграмма компонентов серверного приложения информационной системы контроля и анализа потребления электроэнергии

Каждый из представленных на диаграмме микросервисов отвечает за определенную предметную область:

- API Gateway – общая точка входа в систему для интеграции с клиентскими приложениями;
- сервис пользователей – аутентификация, авторизация и управление пользовательской информацией;
- сервис бригад – формирование бригад инспекторов и управление ими;
- сервис абонентов – работа с реестром абонентов, объектами энергоснабжения, приборами учета и договорами;
- сервис задач – управление жизненным циклом задач на проверку объектов энергоснабжения;
- сервис проверок – управление процессом проведения проверок и формирование актов;
- сервис файлов – централизованное управление файлами в системе;

- сервис аналитики – формирование отчетов о работе бригад инспекторов, подготовка данных для дашбордов аномалий потребления и обзора операций отдела контроля;
- сервис анализа фотографий – проверка качества фотографий приборов учета и средств визуального контроля.

Такое разделение по предметным областям обеспечивает высокую степень модульности, масштабируемости и отказоустойчивости системы, позволяя каждому микросервису независимо развиваться и выполнять свои функции.

Обмен информацией между микросервисами происходит посредством двух подходов – REST API и Apache Kafka.

REST (Representational State Transfer, передача репрезентативного состояния) – архитектурный стиль построения распределенной системы, в основе которого лежит понятие ресурса и его состояния. Ресурс имеет свой универсальный идентификатор (URI, unified resource identifier), используя который, над данным ресурсом можно совершать различные действия, например, CRUD (create, read, update, delete) [12].

REST используется для простого обмена сообщениями между микросервисами в реальном времени. В работе используются следующие методы:

- GET для получения данных;
- POST для создания или изменения ресурса;
- PUT и PATCH для частичного и полного обновления;
- DELETE для удаления.

Apache Kafka – это распределенное хранилище данных, оптимизированное для приема и обработки потоковых данных в режиме реального времени. Потоковые данные – это данные, непрерывно генерируемые тысячами источников данных, которые, как правило, передают записи данных

одновременно. Поточковая платформа должна справляться с таким постоянным притоком данных и обрабатывать их последовательно и поэтапно [13].

Apache Kafka используется для асинхронного обмена сообщениями, например, для оповещения всех микросервисов о смене статуса проверки объекта энергоснабжения. Это позволяет ускорить и упростить работу системы путем отправки сообщений в очередь и продолжением выполнения операций без блокировки основного потока выполнения сервиса.

3.2 Проектирование структур данных

3.2.1 Концептуальная модель баз данных

Концептуальная модель – это модель, отображающая данные и связи предметной области с учетом их семантического содержания, независимо от представления в компьютере. Такая модель представления данных строится для отображения вида хранения данных в общем виде и лучшего понимания какие данные необходимы для хранения в системе [14].

Концептуальная модель базы данных позволяет взглянуть на модель данных на высоком уровне абстракции и не зависеть от конкретной СУБД. Она особенно полезна на ранних этапах проектирования и позволяет выделить основные сущности, их атрибуты и взаимосвязи.

Для разработки концептуальной модели базы данных используется модель «сущность-связь» (Entity-Relation Model, ER-модель), позволяющая наглядно представить структуру базы данных и взаимодействие ее компонентов. На рисунке 16 представлена диаграмма типа «сущность-связь» для серверного приложения информационной системы контроля и анализа потребления электроэнергии.

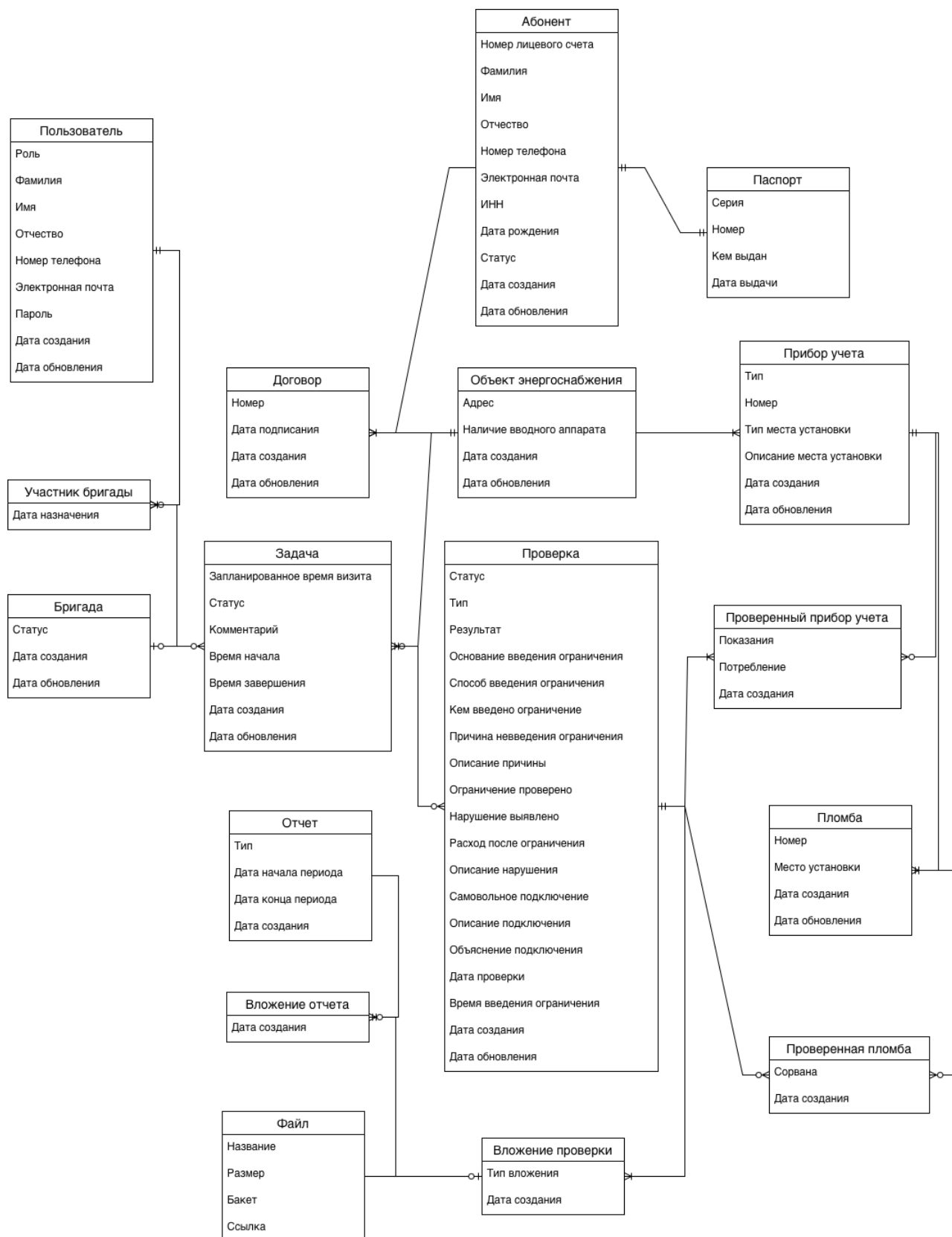


Рисунок 16 – Концептуальная модель базы данных

В результате разработки концептуальной модели базы данных серверного приложения информационной системы контроля и анализа потребления электроэнергии был определен первоначальный набор сущностей, спецификация которого представлена в таблице 4.

Таблица 4 – Спецификация первоначального набора сущностей

№ п/п	Наименование	Идентификатор	Описание
1	Пользователь	user	Содержит информацию о пользователях системы (инспекторы, диспетчеры, специалисты), включая личные данные, контакты и авторизационные данные
2	Бригада	brigade	Содержит информацию о бригадах инспекторов, выполняющих проверки
3	Участник бригады	brigade_member	Содержит информацию о пользователях, состоящих в бригаде
4	Абонент	subscriber	Содержит информацию об абонентах энергосбытовой компании, которые являются владельцами объектов энергоснабжения
5	Паспорт	passport	Содержит информацию о паспортах абонентов
6	Объект энергоснабжения	object	Содержит информацию об объектах энергоснабжения, на которых проводятся проверки
7	Прибор учета	device	Содержит информацию о приборах учета, установленных на объектах энергоснабжения
8	Пломба	seal	Содержит информацию о средствах визуального контроля (пломбах), установленных на приборах учета
9	Договор	contract	Содержит информацию о договорах между энергосбытовой компанией и абонентами
10	Задача	task	Содержит информацию о задачах для бригад инспекторов
11	Файл	storage_file	Содержит информацию о файлах, загруженных в систему
12	Проверка	inspection	Содержит информацию о проверках объектов энергоснабжения
13	Проверенный прибор учета	inspected_device	Содержит информацию о приборах учета, изученных в рамках проверки, включая показания и расход электроэнергии
14	Проверенная пломба	inspected_seal	Содержит информацию о пломбах, изученных в рамках проверки, включая запись о повреждении

Продолжение таблицы 4

№ п/п	Наименование	Идентификатор	Описание
15	Вложение проверки	inspection_attachment	Содержит информацию о файлах, прикрепленных к проверке (фотографии приборов учета и пломб, акты)
16	Отчет	report	Содержит информацию об отчетах о выполненных работах за определенный период
17	Вложение отчета	report_attachment	Содержит информацию о файлах, прикрепленных к отчету

В таблице 5 представлено описание всех возможных связей первоначального набора сущностей.

Таблица 5 – Спецификация связей между сущностями концептуальной схемы данных

№ п/п	Наименование	Семантика	Мощность
1	user-brigade_member	Пользователь может являться участником бригады	1:M
2	brigade-brigade member	Бригада состоит из участников бригады	1:M
3	brigade-task	Бригада выполняет задачи	1:M
4	subscriber-passport	У абонента есть паспорт	1:1
5	subscriber-contract	Абонент заключает договор с энергосбытовой компанией	1:M
6	object-contract	Объект энергоснабжения может входить в несколько договоров за разное время	1:M
7	object-task	Объект энергоснабжения может проверяться в рамках нескольких задач	1:M
8	object-device	На объекте энергоснабжения может быть установлено несколько приборов учета	1:M
9	device-seal	На приборе учета может быть установлено несколько пломб	1:M
10	device-inspected_device	Прибор учета может быть проверен несколько раз	1:M
11	seal-inspected_seal	Пломба может быть проверена несколько раз	1:M
12	task-inspection	В рамках задачи может быть проведено несколько проверок	1:M
13	storage_file-inspection_attachment	Информация о файле может быть привязана к вложению проверки	1:1
14	storage_file-report_attachment	Информация о файле может быть привязана к вложению отчета	1:1
15	inspection-inspected_device	В рамках проверки может быть проверено несколько приборов учета	1:M

Продолжение таблицы 5

№ п/п	Наименование	Семантика	Мощность
16	inspection-inspected_seal	В рамках проверки может быть проверено несколько пломб	1:M
17	inspection-inspection_attachment	К проверке может быть прикреплено несколько файлов	1:M
18	report-report_attachment	К отчету может быть прикреплено несколько файлов	1:M

Концептуальная модель базы данных была разработана в нотации IDEF1X [15] с помощью приложения для построения графиков draw.io.

3.2.2 Логическая модель баз данных

Логической моделью данных называют усовершенствованную версию концептуальной модели. В ней схематически представлены ограничения данных, имена сущностей и связи, которые нужно будет реализовать независимо от платформы [16]. Логическая модель строится на основе концептуальной и представляет собой базу для построения физической модели базы данных.

Целью данного этапа моделирования является преобразование концептуальной ER-модели в реляционную модель данных. Она включает в себя:

- преобразование сущностей в отношения (таблицы);
- преобразование связей во внешние ключи и связующие таблицы;
- проведение нормализации отношений для устранения избыточности и аномалий обновления.

Одним из ключевых аспектов проектирования качественной базы данных является нормализация – процесс приведения отношений к нормальным формам (НФ) [17]. В рамках данной работы проектирование ведется с целью достижения Третьей нормальной формы (3НФ), которая требует, чтобы:

- в базе данных не было дубликатов и составных данных (1НФ);
- у каждой записи в базе данных был первичный ключ (2НФ);

– в записи не было столбцов с неключевыми значениями, которые зависят от других неключевых значений (ЗНФ).

Для серверного приложения информационной системы контроля и анализа потребления электроэнергии было принято решение разделить базы данных по предметным областям. В результате было выделено 7 баз данных: «Пользователи», «Аналитика», «Бригады», «Файлы», «Проверки», «Абоненты» и «Задачи».

Разделение базы данных по предметным областям обладает следующими преимуществами:

- изоляция и независимость бизнес-доменов;
- снижение зацепления и повышение связности;
- масштабируемость и производительность;
- упрощение миграций и развертывания;
- соответствие принципам микросервисной архитектуры.

На рисунке 17 представлена логическая модель базы данных «Пользователи».



Рисунок 17 – Логическая модель базы данных «Пользователи»

На рисунке 18 представлена логическая модель базы данных «Аналитика».

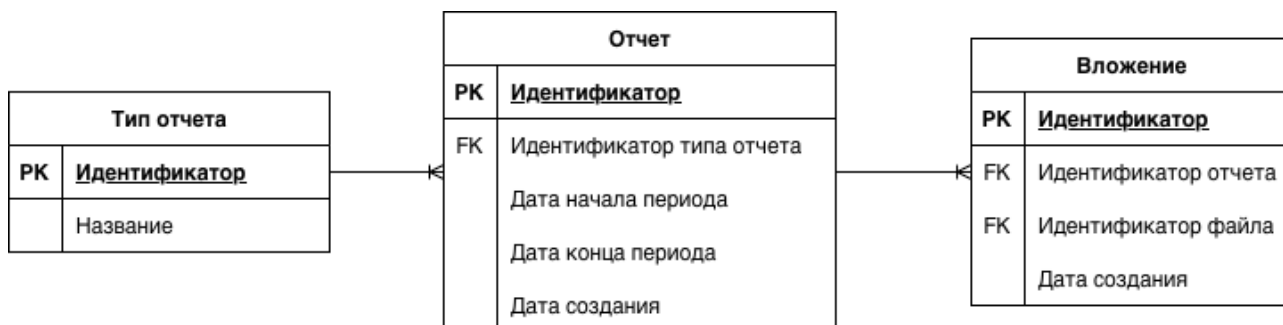


Рисунок 18 – Логическая модель базы данных «Аналитика»

На рисунке 19 представлена логическая модель базы данных «Бригады».

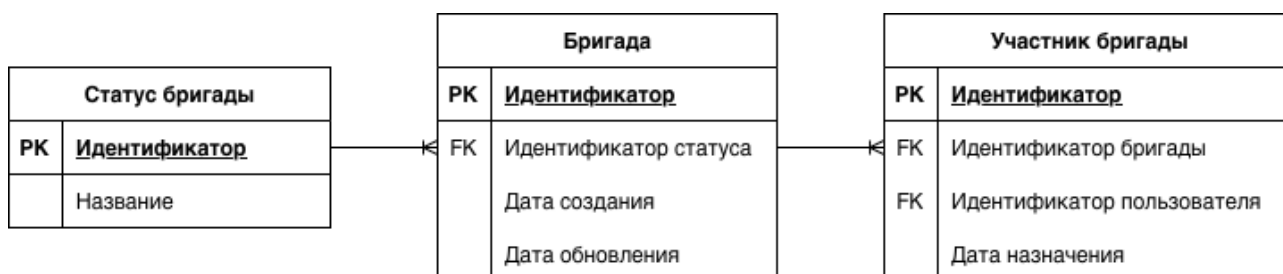


Рисунок 19 – Логическая модель базы данных «Бригады»

На рисунке 20 представлена логическая модель базы данных «Файлы».

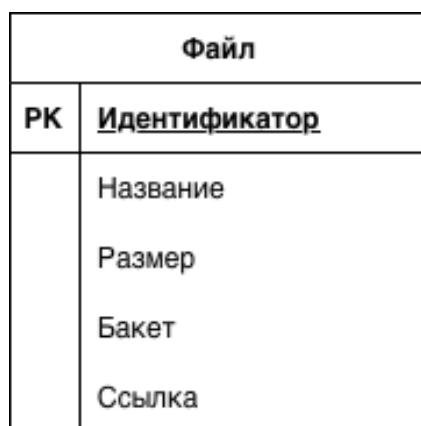


Рисунок 20 – Логическая модель базы данных «Файлы»

На рисунке 21 представлена логическая модель базы данных «Проверки».

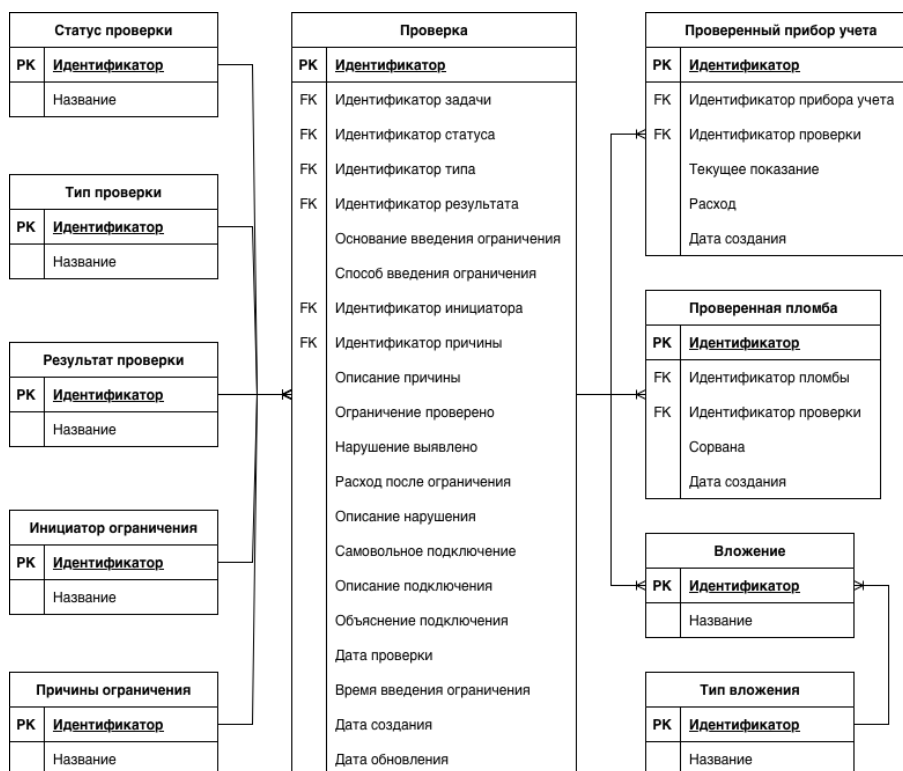


Рисунок 21 – Логическая модель базы данных «Проверки»

На рисунке 22 представлена логическая модель базы данных «Абоненты».

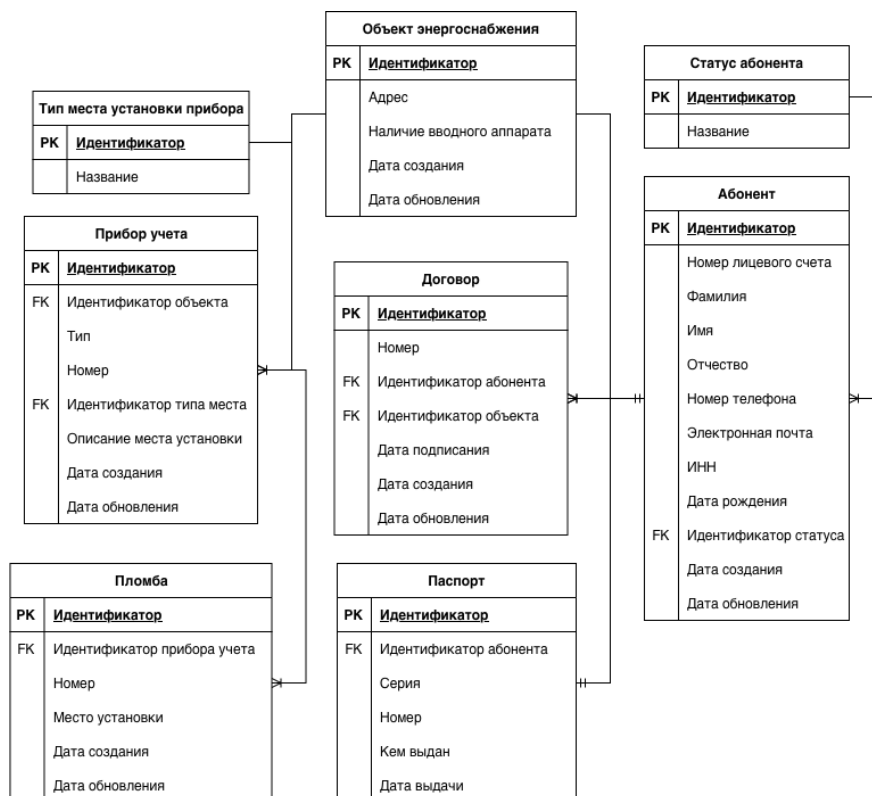


Рисунок 22 – Логическая модель базы данных «Абоненты»

На рисунке 23 представлена логическая модель базы данных «Задачи».



Рисунок 23 – Логическая модель базы данных «Задачи»

Разработанные модели баз данных основаны на концептуальной модели и будут использованы при разработке физической модели базы данных.

3.3 Общая архитектура серверного приложения системы

Для реализации серверного приложения информационной системы контроля и анализа потребления электроэнергии принята микросервисная архитектура. Каждый микросервис фокусируется на своей предметной области (управление абонентами и объектами учета, бригадами, задачами на проверки, проверками на объектах, файлами, аналитическими отчетами и др.) и развертывается независимо. Взаимодействие между сервисами и клиентскими приложениями осуществляется по протоколу HTTP на основе REST-контрактов, а для асинхронного обмена событиями используется брокер сообщений Apache Kafka. Хранение данных разделено по ответственности: доменные сущности сохраняются в PostgreSQL, файловые вложения – в MinIO, агрегированные аналитические данные – в ClickHouse.

Серверное приложение системы построено вокруг единой точки входа для внешних клиентов в виде прокси-сервера Nginx, выполняющего функции API-шлюза. Через него веб-портал и мобильное приложение инспектора обращаются к конечным точкам REST отдельных микросервисов, не взаимодействуя с ними напрямую. Такой подход позволяет централизованно

описать контракты, выполнять маршрутизацию запросов, применять механизмы аутентификации и авторизации, а также скрывать внутреннюю структуру системы от потребителей API.

Каждый из микросервисов реализован по принципам многослойной архитектуры, что обеспечивает четкое разделение ответственности между обработкой HTTP-запросов, бизнес-логикой и работой с данными. На рисунке 24 представлена типовая архитектура микросервиса, используемая в системе.

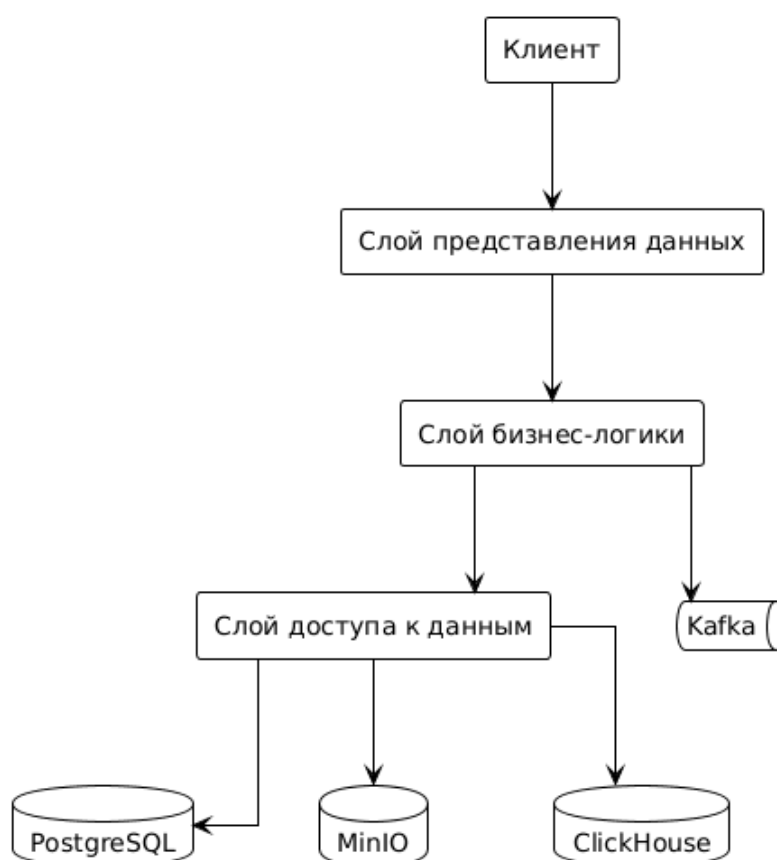


Рисунок 24 – Типовая многослойная архитектура микросервисов серверного приложения информационной системы контроля и анализа потребления электроэнергии

В верхний слой архитектуры входит клиент, под которым понимаются веб-портал, мобильное приложение или другие микросервисы, обращающиеся к сервису по HTTP. Полученные от клиента запросы обрабатываются слоем представления данных (маршрутизатором и обработчиками), отвечающим за

прием и разбор запросов, валидацию входных данных, формирование стандартных ответов и преобразование внешних DTO во внутренние представления. После успешной валидации запрос передается на уровень бизнес-логики.

Слой бизнес-логики инкапсулирует предметные правила и операции, выполняемые над доменными объектами сервиса (абонент, объект учета, задача проверки, проверка, файл, отчет и т.п.). Здесь реализуются сценарии создания и изменения сущностей, проверки ограничений, регистрация событий и вызовы асинхронной обработки через Kafka. При необходимости получить или сохранить данные в постоянное хранилище слой бизнес-логики использует следующий уровень – слой доступа к данным.

Слой доступа к данным реализует паттерн репозитория и определяет, как микросервис взаимодействует с базами данных и другими хранилищами. В нем инкапсулируются детали работы с SQL-запросами к PostgreSQL, операции с объектным хранилищем MinIO, а также преобразование между структурами хранения (модели БД, объекты файлового хранилища) и доменными сущностями. Это позволяет изолировать бизнес-логику от конкретных технологий хранения и упростить последующую эволюцию схем данных.

На уровне баз данных и внешних хранилищ собираются и сохраняются все данные, относящиеся к конкретному микросервису: записи о задачах и проверках, сведения об абонентах и объектах учета, метаданные файлов и сформированных отчетов, агрегированные аналитические показатели. Использование отдельного аналитического хранилища ClickHouse позволяет разгрузить СУБД PostgreSQL от тяжелых отчетных запросов и обеспечить быструю выборку данных для дашбордов аномалий потребления и обзора операций. Объектное хранилище MinIO обеспечивает надежное и масштабируемое хранение файловых вложений. В совокупности многослойная организация микросервисов и разделение хранилищ по типам нагрузок повышают сопровождаемость, масштабируемость и отказоустойчивость серверной части системы.

3.4 Обобщенная структура разрабатываемой системы

В обобщенном виде разрабатываемая информационная система контроля и анализа потребления электроэнергии представляет собой распределенное клиент-серверное приложение, реализованное в виде набора специализированных микросервисов. Клиентской частью выступают веб-портал для диспетчеров и специалистов по работе с абонентами и мобильное приложение для инспекторов. Через них пользователи получают доступ к функциям учета абонентов и объектов, планирования и выполнения проверок, фиксации результатов, формирования отчетности и просмотра аналитических дашбордов. Серверная часть разворачивается в виде нескольких взаимосвязанных сервисов, обменивающихся данными по HTTP и через брокера сообщений Kafka, а также использующих общие хранилища данных и файлов.

К преимуществам такой архитектуры можно отнести:

- масштабируемость: отдельные сервисы (например, сервис задач или сервис проверок) могут быть независимо масштабированы по горизонтали при росте нагрузки;
- изолированность отказов: сбой одного сервиса не приводит к полной недоступности системы при сохранении работоспособности остальных компонентов;
- гибкость развития: функциональность по учету абонентов, планированию проверок, аналитике и работе с файлами может эволюционировать независимо за счет отдельных репозиторий и циклов развертывания;
- возможность подбора оптимального технологического стека для разных задач.

Вместе с тем использование микросервисной архитектуры накладывает определенные ограничения и сложности:

- возрастает сложность инфраструктуры: помимо самих сервисов необходимо поддерживать работу брокера сообщений, объектного и аналитического хранилищ, системы мониторинга и обратного прокси;
- увеличивается сетевое взаимодействие между компонентами, что требует внимания к вопросам производительности, отказоустойчивости и трассировки межсервисных вызовов;
- появляются особенности, связанные с согласованностью данных при событийном обмене, что необходимо учитывать при проектировании бизнес-процессов;
- для поддержки эксплуатации требуются развитые средства наблюдаемости (метрики, логи, трассировки) и централизованный мониторинг.

Архитектура приложения в общем виде представлена на диаграмме развертывания на рисунке 25.

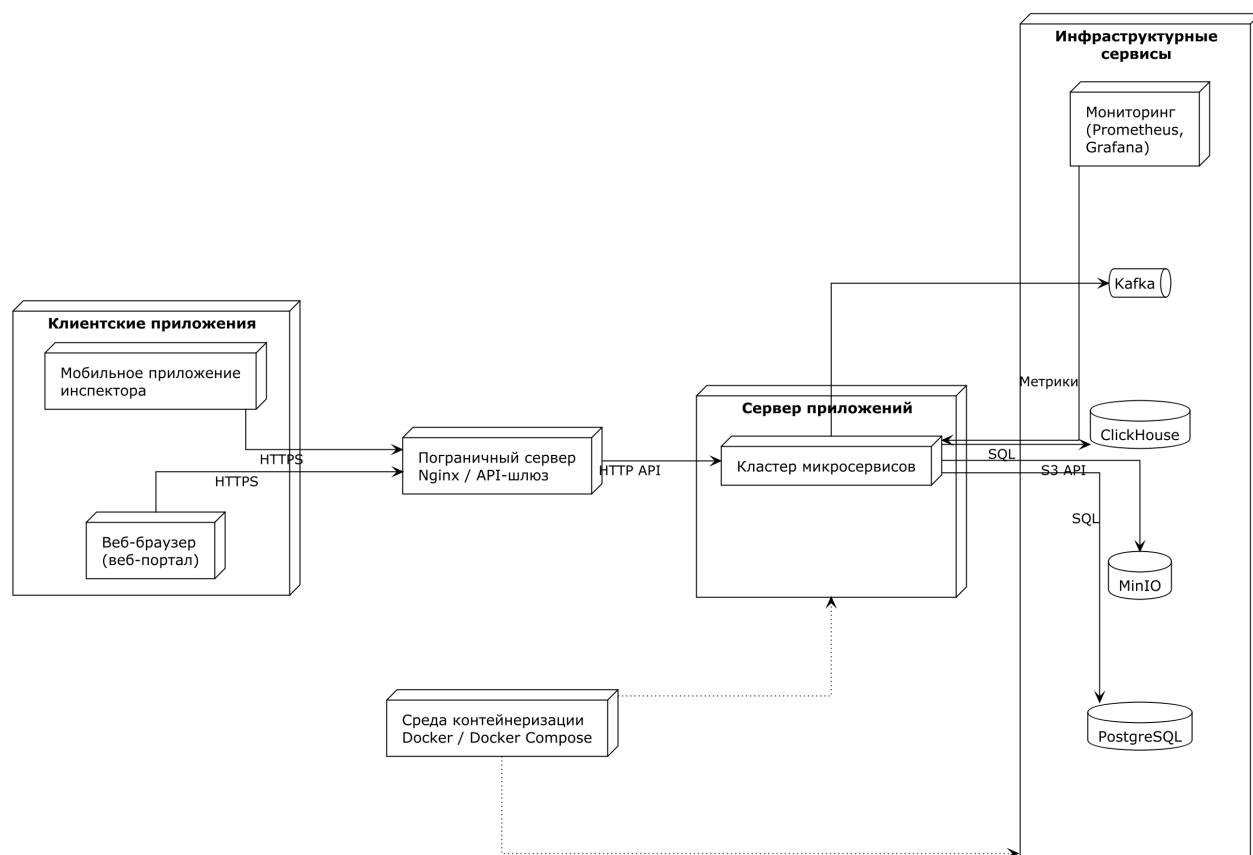


Рисунок 25 – Диаграмма развертывания информационной системы контроля и анализа потребления электроэнергии

На диаграмме отражены основные узлы, обеспечивающие работу системы: клиентские приложения (веб-браузер и мобильное приложение), пограничный узел с HTTP-прокси (Nginx/API-шлюз), набор микросервисов, а также инфраструктурные компоненты для хранения и обработки данных. Микросервисы используют СУБД PostgreSQL для хранения сущностей (абоненты, объекты, задачи, проверки и др.), объектное хранилище MinIO для файловых вложений (фотографии приборов учета, акты, отчеты) и аналитическое хранилище ClickHouse для агрегированных сведений о выполненных работах, динамике потребления и операционных показателях отдела контроля. Для асинхронного взаимодействия между сервисами применяется брокер сообщений Kafka, через который распространяются события жизненного цикла задач и проверок.

Пользователи взаимодействуют с системой посредством выполнения HTTP-запросов из веб-портала или мобильного приложения к API-шлюзу, который маршрутизирует обращения к соответствующим микросервисам. При выполнении операций, связанных с планированием и проведением проверок, сервис задач создает записи в базе данных и публикует события в Kafka, сервис проверок фиксирует результаты проверок и инициирует генерацию актов, а аналитический сервис на основе поступающих событий формирует отчеты, рассчитывает показатели для дашбордов и сохраняет их в аналитическом хранилище. При необходимости каждый из сервисов обращается к централизованному файловому хранилищу для загрузки или получения связанных документов и фотографий.

Аналитический сервис получает из Kafka события о создании и изменении задач, завершении проверок, фиксации показаний приборов учета и выявлении нарушений. Эти события преобразуются в агрегированные записи, удобные для построения дашбордов. Для дашборда аномалий потребления рассчитываются отклонения текущего расхода от предыдущих значений, признаки резкого роста или падения потребления, а также повторяемость нарушений по объектам энергоснабжения. Для дашборда обзора операций

формируются показатели количества активных и завершенных задач, распределения проверок по результатам, средней длительности выполнения задач и загрузки бригад. Благодаря этому веб-портал может предоставлять пользователю не только реестр операций, но и управленческую картину состояния отдела контроля.

Для обеспечения развертывания и обновления всех компонентов на серверной стороне используется контейнеризация. Микросервисы, брокер сообщений, системы хранения и мониторинга упакованы в контейнеры и поднимаются с помощью Docker и Docker Compose. Такой подход позволяет единообразно описать состав окружения, упростить запуск системы во время разработки и перенести конфигурацию на производственные серверы, а также обеспечивает возможность масштабирования отдельных компонентов за счет изменения параметров оркестрации без модификации прикладного кода.

3.5 Требования к аппаратному обеспечению серверного приложения

Для корректного функционирования серверного приложения разрабатываемой информационной системы контроля и анализа потребления электроэнергии необходимо обеспечить определенный уровень вычислительных ресурсов на стороне инфраструктуры, где будут запускаться контейнеры с микросервисами, брокером сообщений, системами хранения данных и мониторинга. Требования к аппаратному обеспечению определяются с учетом одновременной работы нескольких сервисов, обработки входящих запросов, обмена сообщениями между компонентами, записи и чтения данных из хранилищ, а также накопления журналов и метрик работы системы. Дополнительный запас вычислительных ресурсов необходим для обеспечения стабильной работы приложения при увеличении количества пользователей, росте объема данных о потреблении электроэнергии и выполнении аналитических операций. Минимальные и рекомендуемые требования к

аппаратному обеспечению серверных узлов, предназначенных для размещения данной системы, сведены в таблицу 6.

Таблица 6 – Требования к аппаратному обеспечению серверного приложения

Параметр	Минимальная конфигурация	Рекомендуемая конфигурация
Процессор	x64, 4 ядра с тактовой частотой от 2.0 ГГц	x64, 8 и более ядер с тактовой частотой от 3.0 ГГц
Оперативная память	8 ГБ	16 ГБ и более
Тип и объем дискового пространства	HDD объемом не менее 200 ГБ	SSD объемом 1 ТБ и более
Пропускная способность сети	1 Гбит/с	1 Гбит/с и выше, стабильный канал связи с внешними системами
Операционная система	Ubuntu Server 20.04 LTS или эквивалентная Linux-система	Ubuntu Server 22.04 LTS и выше
Обязательное ПО	Docker и Docker Compose актуальных версий	Docker и Docker Compose актуальных версий

Выбранные параметры ориентированы на типовой сценарий эксплуатации, при котором на одном сервере одновременно развернуты все основные компоненты системы: микросервисы, брокер сообщений Kafka, СУБД PostgreSQL, объектное хранилище MinIO, аналитическое хранилище ClickHouse и средства мониторинга. При росте нагрузки или объемов данных возможно горизонтальное масштабирование за счет распределения микросервисов и инфраструктурных сервисов по нескольким физическим или виртуальным серверам; в этом случае приведенные требования могут рассматриваться как базовые для каждого отдельного узла с последующей адаптацией под фактический профиль нагрузки.

4 Реализация программных средств

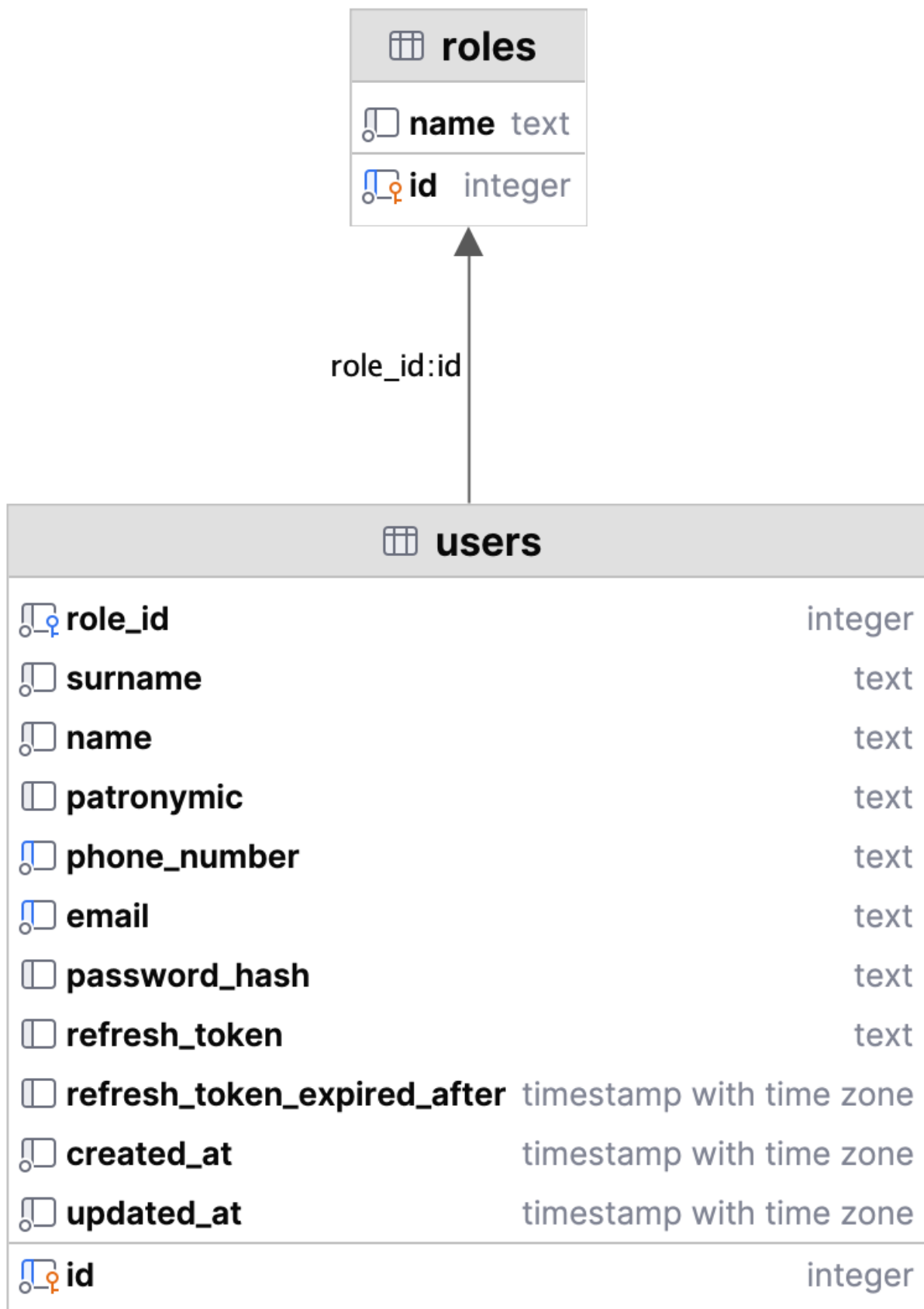
4.1 Физическая модель баз данных

Физическая модель базы данных представляет собой заключительный этап проектирования хранилища данных и формируется на основе логической модели, уточненной под конкретную выбранную СУБД. На данном этапе окончательно определяются состав таблиц, столбцов и связей, типы данных, ограничения целостности, индексы и другие свойства, влияющие на производительность, надежность и масштабируемость хранилища. Для рассматриваемого серверного приложения информационной системы контроля и анализа потребления электроэнергии в качестве основной СУБД выбрана PostgreSQL, на основе которой реализованы отдельные базы данных для микросервисов.

Такое разделение соответствует выбранному микросервисному архитектурному стилю и позволяет изолировать данные каждой предметной области. Каждый сервис работает только со своей частью хранилища, что снижает связанность между компонентами и упрощает сопровождение структуры данных при дальнейшем развитии системы. При необходимости изменение таблиц одного микросервиса может выполняться независимо от остальных подсистем, если при этом сохраняются внешние программные интерфейсы взаимодействия. Для управления изменениями структуры баз данных используются миграции, позволяющие последовательно создавать таблицы, ограничения, связи и индексы в среде развертывания.

Физические модели отдельных подсистем представлены на следующих рисунках. На диаграммах отражены ключевые таблицы, их атрибуты, первичные и внешние ключи, а также связи между сущностями внутри каждой предметной области.

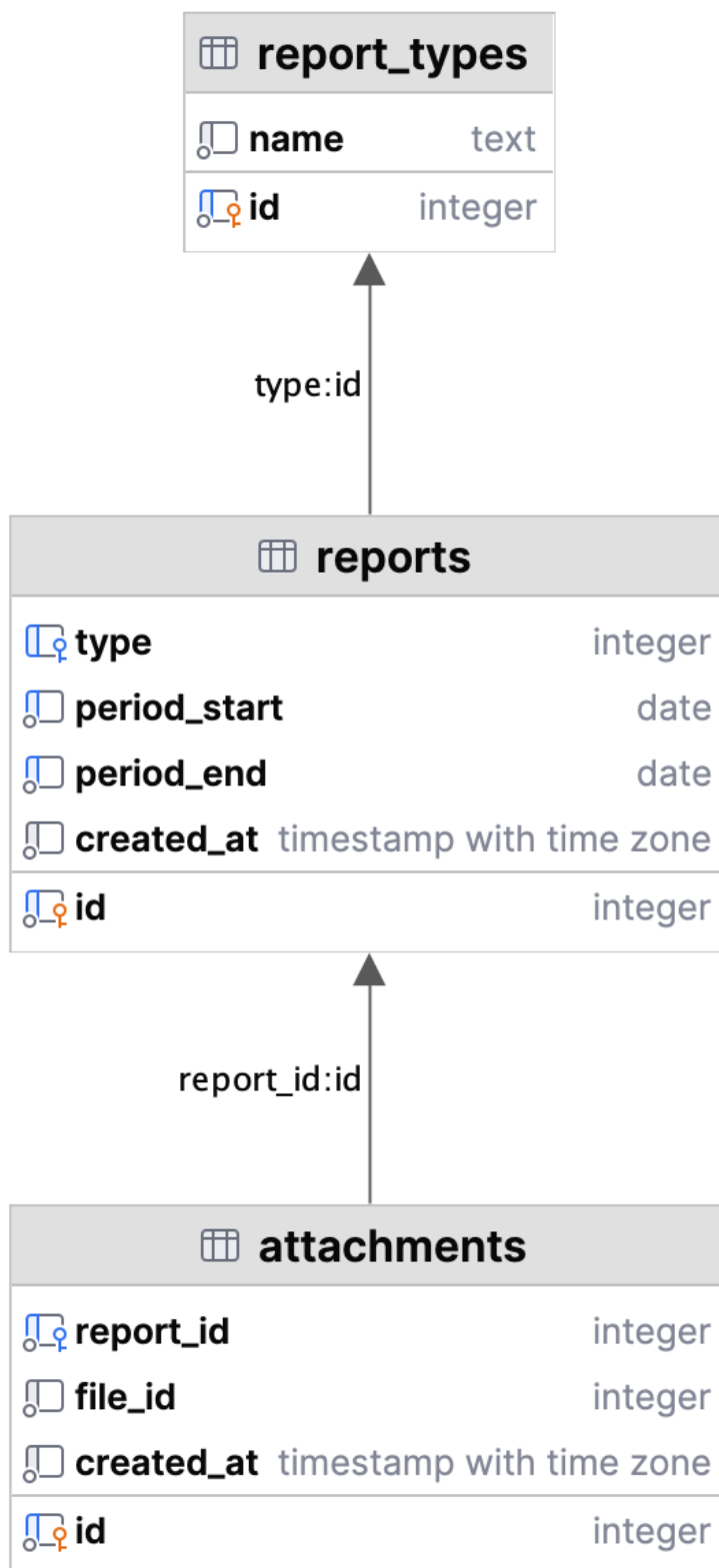
На рисунке 26 представлена физическая модель базы данных «Пользователи».



Powered by yfiles

Рисунок 26 – Физическая модель базы данных «Пользователи»

На рисунке 27 представлена физическая модель базы данных «Аналитика».



Powered by yFiles

Рисунок 27 – Физическая модель базы данных «Аналитика»

На рисунке 28 представлена физическая модель базы данных «Бригады».

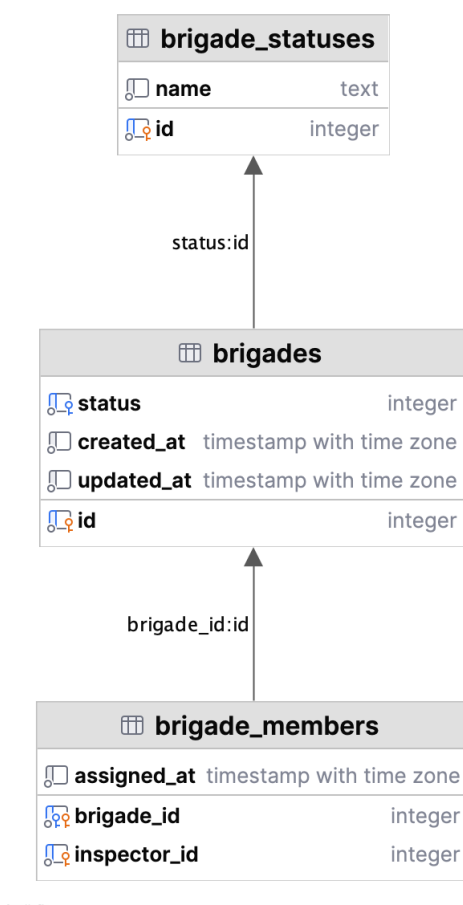


Рисунок 28 – Физическая модель базы данных «Бригады»

На рисунке 29 представлена физическая модель базы данных «Файлы».

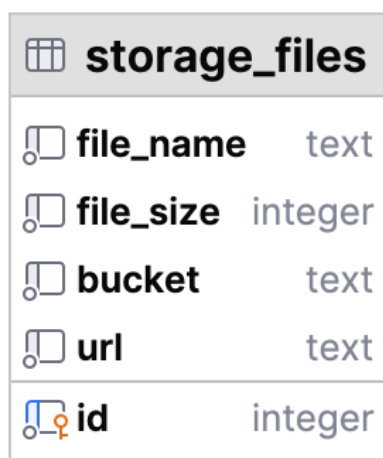


Рисунок 29 – Физическая модель базы данных «Файлы»

На рисунке 30 представлена физическая модель базы данных «Проверки».

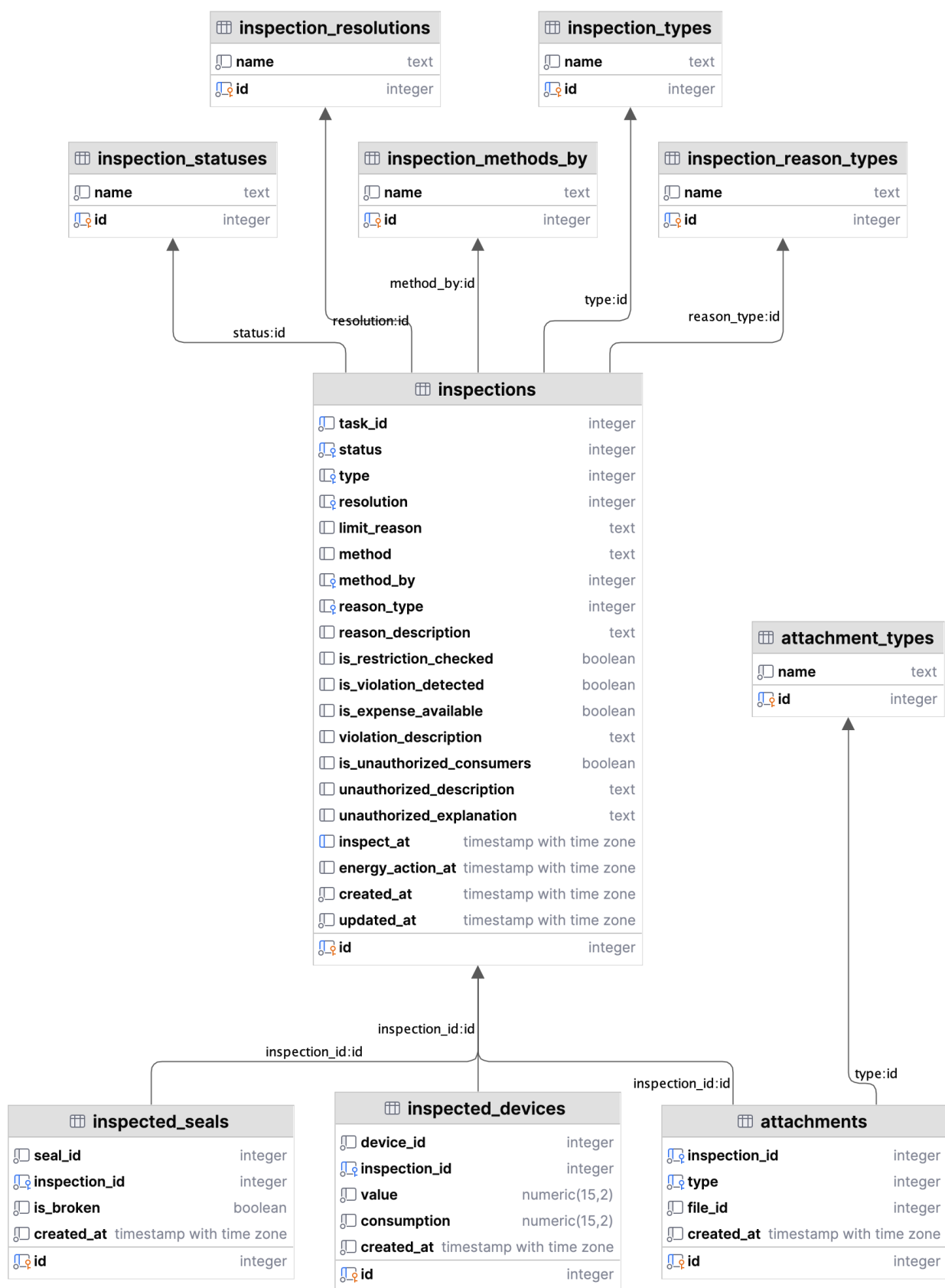


Рисунок 30 – Физическая модель базы данных «Проверки»

На рисунке 31 представлена физическая модель базы данных «Абоненты».

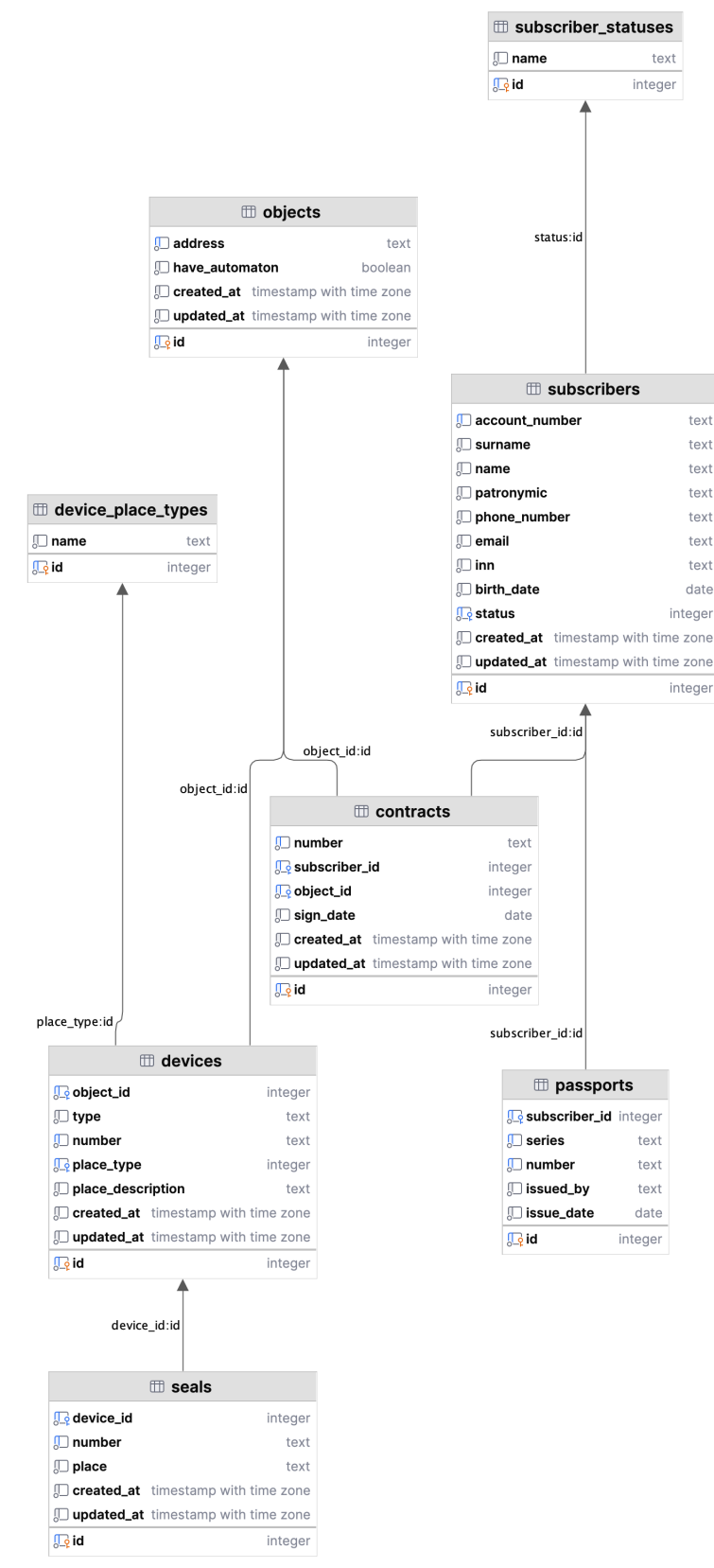


Рисунок 31 – Физическая модель базы данных «Абоненты»

На рисунке 32 представлена физическая модель базы данных «Задачи».

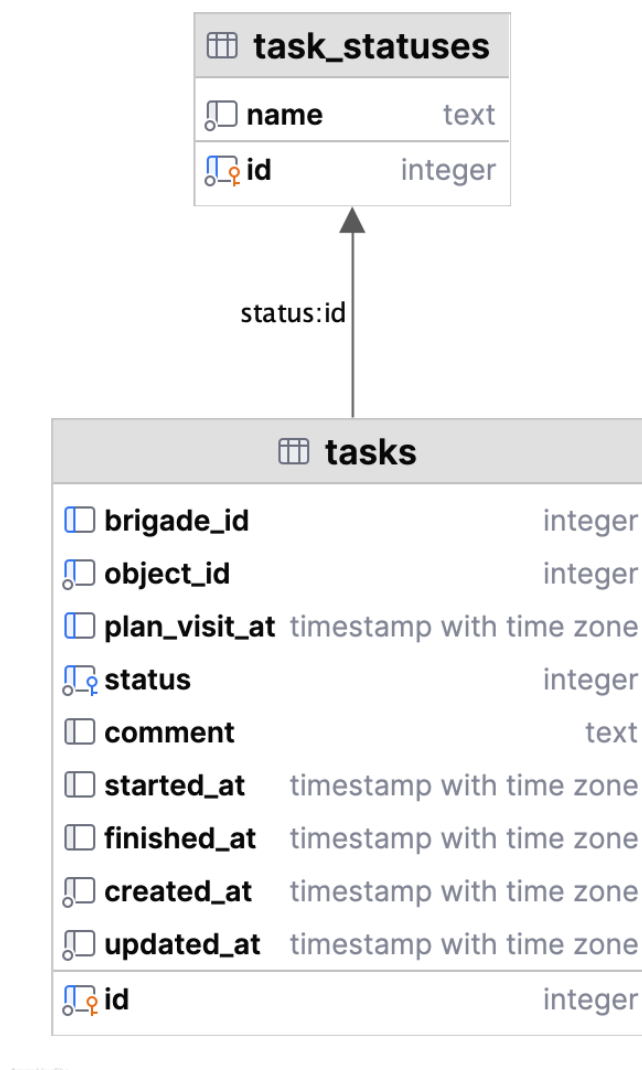


Рисунок 32 – Физическая модель базы данных «Задачи»

В таблице 7 приведен перечень основных сущностей, реализованных в реляционных базах данных системы, с указанием их назначения и принадлежности к одной из категорий: стержневые (ключевые доменные таблицы), кодовые (справочники и статусы), характеристические (подробные данные и измерения) и ассоциативные (связующие таблицы для реализации связей «многие-ко-многим» и хранения присоединенных объектов).

Таблица 7 – Описание сущностей баз данных серверного приложения информационной системы контроля и анализа потребления электроэнергии

№ п/п	Наименование	Описание	Категория
1	users	Информация о пользователях	Стержневая
2	roles	Справочник ролей пользователей	Кодовая
3	subscriber_statuses	Справочник статусов абонентов	Кодовая
4	subscribers	Список абонентов с персональными и контактными данными	Стержневая
5	passports	Паспортные данные абонентов	Характеристическая
6	objects	Перечень объектов учета с адресами и характеристиками подключения	Стержневая
7	device_place_types	Справочник типов мест установки приборов учета	Кодовая
8	devices	Сведения о приборах учета, установленных на объектах	Стержневая
9	seals	Информация о пломбах, установленных на приборах учета	Характеристическая
10	contracts	Договоры между абонентами и энергосбытовой организацией	Стержневая
11	brigade_statuses	Справочник статусов бригад	Кодовая
12	brigades	Список бригад инспекторов с текущими статусами	Стержневая
13	brigade_members	Состав бригад, связи между бригадами и инспекторами	Ассоциативная
14	task_statuses	Справочник статусов задач на проверки	Кодовая
15	tasks	Задачи на проведение выездных проверок по объектам учета	Стержневая
16	inspection_statuses	Справочник статусов проверок	Кодовая
17	inspection_types	Справочник типов проверок	Кодовая
18	inspection_resolutions	Справочник возможных результатов проверок	Кодовая
19	inspection_methods_by	Справочник, определяющий, кем произведено действие по ограничению/возобновлению	Кодовая
20	inspection_reason_types	Справочник типов оснований и причин по результатам проверки	Кодовая
21	inspections	Список проверок	Стержневая
22	inspected_devices	Измеренные показания приборов учета и рассчитанный расход по конкретной проверке	Характеристическая
23	inspected_seals	Результаты осмотра пломба в рамках проведенной проверки	Характеристическая
24	attachment_types	Справочник типов вложений к проверкам	Кодовая

Продолжение таблицы 7

№ п/п	Наименование	Описание	Категория
25	attachments	Связи между проверками и файлами, хранящимися во внешнем файловом хранилище	Ассоциативная
26	storage_files	Метаданные файлов	Стержневая
27	report_types	Справочник типов аналитических отчетов и представлений дашбордов	Кодовая
28	reports	Список сформированных аналитических отчетов с указанием периода и параметров анализа	Стержневая
29	attachments	Связи между отчетами и файлами отчетных документов	Ассоциативная

Подробные спецификации атрибутов сущностей, их типов данных и ограничений приведены в приложении Б.

Описание структур данных, используемых в микросервисах на уровне доступа к базам данных, представлено в приложении В.

Полные SQL-скрипты инициализации и наполнения таблиц вынесены в приложение Г.

4.2 Разработка архитектурных слоев внутри микросервисов

Обеспечение строгого разделения обязанностей между элементами каждого микросервиса представляет собой одну из ключевых задач при проектировании серверного приложения информационной системы контроля и анализа потребления электроэнергии. Многоуровневая архитектура применяется во всех сервисах приложения для обеспечения возможности масштабирования, простоты тестирования и удобства сопровождения программного кода. Каждый уровень такой архитектуры выполняет определенный набор функций и обменивается данными с соседними уровнями посредством явно определенных интерфейсов.

Микросервис управления проверками (inspection-service) выбран в качестве базового примера для подробного анализа организации уровней и механизмов их взаимодействия. Остальные микросервисы системы построены

по аналогичным принципам, а функциональность `inspection-service` достаточно полно отражает все основные архитектурные подходы, применяемые в системе: прием и обработку HTTP-запросов, реализацию бизнес-правил с интеграцией внешних сервисов, операции с базами данных и генерацию асинхронных событий.

Реализация каждого уровня для остальных микросервисов системы детально описана в приложениях:

- описание слоя представления данных в приложении Д;
- описание слоя бизнес-логики в приложении Е;
- описание слоя доступа к данным в приложении Ж.

В рамках данного раздела последовательно рассматриваются все перечисленные архитектурные уровни микросервиса проверок.

4.2.1 Разработка слоя представления данных в микросервисах

На уровне представления данных определяются контракты взаимодействия, которые позволяют клиентским приложениям понять, какие действия доступны через конкретный микросервис. Функции данного архитектурного уровня оформляются как обработчики HTTP-запросов и регистрируются в общем маршрутизаторе в процессе инициализации серверного приложения. Методы этого уровня принимают HTTP-запросы различных типов (GET, POST, PUT, PATCH, DELETE), осуществляют проверку корректности входящих данных и направляют их на уровень бизнес-логики для выполнения требуемых операций.

Листинг 1 демонстрирует процесс регистрации маршрутов в маршрутизаторе для микросервиса проверок.

Листинг 1 – Инициализация маршрутизатора и подключение маршрутов

```
func (s *ServerBuilder) AddInspections(service
*inspection.Service) {
    r := s.router.SubRouter("/inspections")
    r.HandleGet("", handler.GetAllInspections(service))
    r.HandleGet("/task/{taskID}",
handler.GetInspectionByTaskID(service))
```

```

        r.HandlePost("/{id}/photo",
handler.AttachPhotoToInspection(service))
        r.HandlePatch("/{id}/finish",
handler.FinishInspection(service))
    }

```

В представленном фрагменте кода формируется подроутер с базовым путем `/inspections`, к которому привязываются обработчики для различных операций: получения перечня всех проверок, поиска проверки по номеру задачи, добавления фотографии к проверке и завершения процесса проверки. Обработчики получают ссылку на объект сервиса бизнес-логики, который выполняет основную работу по обработке запроса.

Листинг 2 содержит реализацию обработчика запроса на добавление фотографии к проверке.

Листинг 2 – Реализация функции обработки запроса на привязку фотографии

```

func AttachPhotoToInspection(s *inspection.Service)
gorouter.Handler {
    return func(c gorouter.Context) error {
        var vars inspectionIDVars
        c.Vars(&vars)
        files, err := c.FormFiles("Photo")
        attachmentTypes, err := c.FormValues("AttachmentType")
        attachmentTypeRaw, err :=
strconv.Atoi(attachmentTypes[0])
        attachmentType :=
inspection.AttachmentType(attachmentTypeRaw)
        var deviceID, sealID int
        switch attachmentType {
        case inspection.AttachmentTypeDevicePhoto:
            deviceIDs, fErr := c.FormValues("DeviceID")
            deviceID, fErr = strconv.Atoi(deviceIDs[0])
        case inspection.AttachmentTypeSealPhoto:
            sealIDs, fErr := c.FormValues("SealID")
            sealID, fErr = strconv.Atoi(sealIDs[0])
        default:
            return fmt.Errorf("invalid attachment type: %d",
attachmentType)
        }
        response, err := s.AttachPhoto(c.Ctx(),
c.Log().WithTags("AttachPhoto"), inspection.AttachPhotoRequest{
            InspectionID: vars.ID,
            Type:         attachmentType,
            DeviceID:     deviceID,
            SealID:       sealID,
            FileHeader:   files[0],

```

```

    })
    return c.WriteJson(http.StatusOK, response)
}
}

```

Обработчик выполняет последовательность операций: извлекает идентификатор проверки из параметров маршрута, читает файл изображения и дополнительные атрибуты запроса из данных формы, проверяет корректность полученных значений и формирует структуру запроса для передачи на уровень бизнес-логики. После успешной обработки запроса сервисом результат сериализуется в формат JSON и возвращается клиенту с HTTP-кодом 200 OK.

Рисунок 33 иллюстрирует схему взаимодействия между маршрутизатором и обработчиками HTTP-запросов в микросервисе проверок.

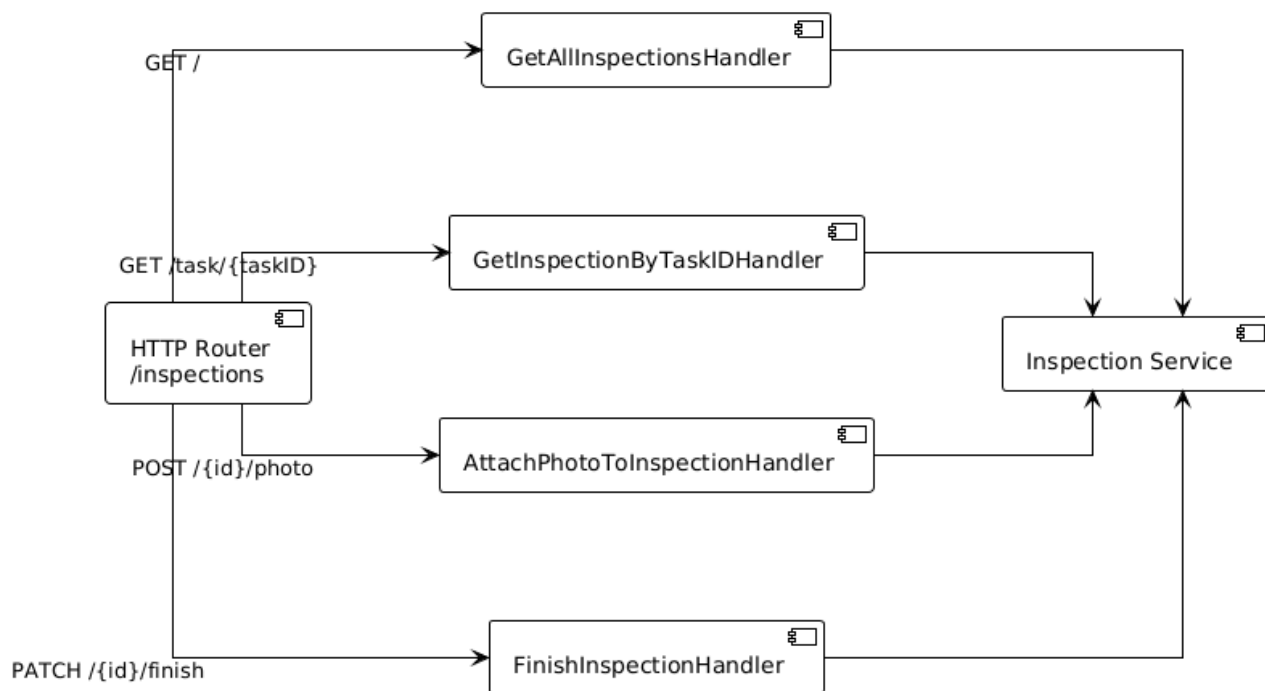


Рисунок 33 – Связь между маршрутизатором и функциями обработки запросов микросервиса проверок

На диаграмме показан механизм распределения входящих HTTP-запросов по соответствующим обработчикам в зависимости от используемого метода или пути запроса. Обработчики получают контекст выполнения запроса, извлекают нужные параметры, проверяют их корректность и передают управление

сервисному уровню для выполнения бизнес-операций, после чего формируют и возвращают ответное сообщение клиенту.

4.2.2 Разработка слоя бизнес-логики в микросервисах

Уровень бизнес-логики реализует операции, характерные для конкретной предметной области микросервиса. Структуры данных, включающие функции, описывающие бизнес-процессы, обозначаются суффиксом Service для указания их принадлежности к уровню бизнес-логики.

Листинг 3 показывает процесс создания экземпляра сервиса и его структуру.

Листинг 3 – Инициализация сервиса и описание его структуры

```
type Service struct {
    repository      Repository
    publisher        *Publisher
    analyzerService AnalyzerService
    subscriberService SubscriberService
    fileService      FileService
    taskService      TaskService
    brigadeService   BrigadeService
    templates        config.Templates
}

func NewService(repository Repository, publisher *Publisher,
analyzerService AnalyzerService, subscriberService
SubscriberService, fileService FileService,
    taskService TaskService, brigadeService BrigadeService,
templates config.Templates) *Service {
    return &Service{
        repository:      repository,
        publisher:        publisher,
        analyzerService:  analyzerService,
        subscriberService: subscriberService,
        fileService:      fileService,
        taskService:      taskService,
        brigadeService:   brigadeService,
        templates:        templates,
    }
}
```

Структура сервиса объединяет необходимые зависимости: репозиторий для операций с базой данных, компонент публикации событий для отправки

сообщений в Kafka, клиентские модули внешних сервисов для запроса дополнительных сведений и настройки шаблонов для создания документов. Подобная структура зависимостей упрощает процесс тестирования сервиса путем замены реализаций интерфейсов на тестовые заглушки и повышает адаптивность системы при модификации внешних компонентов.

Обработка информации на данном уровне выполняется с применением доменных моделей предметной области, которые служат основой для всей логики обработки. Уровень бизнес-логики также взаимодействует с уровнем доступа к данным для чтения и записи необходимой информации.

Листинг 4 содержит пример реализации метода уровня бизнес-логики для добавления фотографии к проверке.

Листинг 4 – Пример функции слоя бизнес-логики для привязки фотографии

```
func (s *Service) AttachPhoto(ctx goctx.Context, log golog.Logger,
request AttachPhotoRequest) (Attachment, error) {
    f, _ := request.FileHeader.Open()
    var fileBuffer bytes.Buffer
    io.Copy(&fileBuffer, f)
    processedImage, _ := s.analyzerService.ProcessImage(ctx,
request.FileHeader.Filename, bytes.NewReader(fileBuffer.Bytes()))
    if processedImage.IsBlurred || processedImage.HasError {
        return Attachment{}, ErrBlurredPhoto
    }
    var object subscriber.Object
    switch request.Type {
    case AttachmentTypeDevicePhoto:
        object, _ = s.subscriberService.GetObjectByDeviceID(ctx,
request.DeviceID)
    case AttachmentTypeSealPhoto:
        object, _ = s.subscriberService.GetObjectBySealID(ctx,
request.SealID)
    }
    number, _ := attachmentNumber(request, object)
    fileName := fmt.Sprintf(
        "%s - %s №%s от %s%s",
        object.Address,
        attachmentName(request.Type),
        number,
        gotime.MoscowNow().Format("02.01.2006 15.04.05"),
        filepath.Ext(request.FileHeader.Filename),
    )
    uploadedFile, _ := s.fileService.Upload(ctx, fileName,
bytes.NewReader(fileBuffer.Bytes()))
}
```

```

        attachment, _ := s.repository.AddAttachment(ctx,
request.InspectionID, uploadedFile.ID, request.Type)
        return attachment, nil
    }

```

Метод `AttachPhoto` реализует последовательность шагов обработки фотографии: открытие файла из запроса, перенос содержимого в буфер памяти, передача изображения в сервис оценки качества для анализа на размытость и дефекты, запрос сведений об объекте учета из микросервиса абонентов, генерация имени файла с учетом адреса объекта и типа вложения, размещение файла в централизованном хранилище через `file-service` и регистрация связи между проверкой и файлом в базе данных через репозиторий. Подобная последовательность операций гарантирует контроль качества фотографий на стадии загрузки и создание осмысленных имен файлов для удобного поиска и использования в дальнейшем.

Рисунок 34 содержит диаграмму классов сервиса уровня бизнес-логики микросервиса проверок.

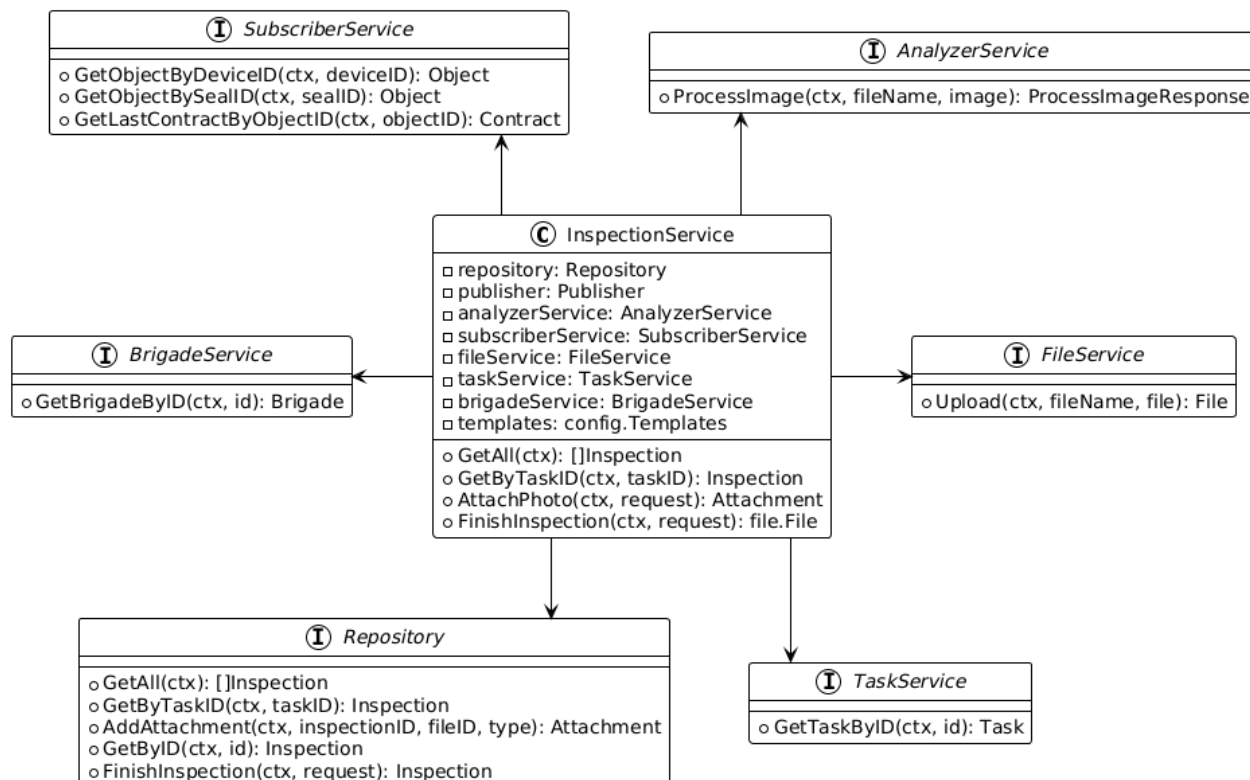


Рисунок 34 – Диаграмма классов сервиса слоя бизнес-логики микросервиса проверок

Диаграмма показывает организацию сервиса вместе с его зависимостями: репозиторием для работы с базой данных, клиентскими модулями внешних сервисов для получения дополнительных сведений и компонентом публикации событий для асинхронного обмена сообщениями. Каждый метод сервиса реализует конкретный бизнес-процесс, комбинируя обращения к репозиторию и внешним сервисам для получения необходимого результата.

4.2.3 Разработка слоя доступа к данным в микросервисах

Уровень доступа к базе данных обеспечивает установку соединения и выполнение операций с базой данных через реализованные функции и SQL-скрипты. Структуры данных, именуемые репозиториями и содержащие функции работы с базой, обозначаются суффиксом Repository для указания их принадлежности к данному уровню. Этот уровень не включает сложной логики обработки данных и предназначен исключительно для выполнения операций над данными в БД: выборка, добавление, изменение и удаление записей.

Листинг 5 демонстрирует создание экземпляра репозитория и его структуру.

Листинг 5 – Инициализация репозитория и описание его структуры

```
type Repository struct {
    db *sqlx.DB
}

func NewRepository(db *sqlx.DB) *Repository {
    return &Repository{
        db: db,
    }
}
```

Репозиторий скрывает детали подключения к базе данных за счет использования библиотеки sqlx, расширяющей стандартный пакет database/sql дополнительным функционалом для работы со структурами языка Go. В момент создания репозиторий получает установленное соединение с базой данных, которое было настроено при запуске приложения.

Листинг 6 содержит пример метода уровня доступа к данным для добавления вложения к проверке.

Листинг 6 – Пример метода слоя доступа к данным

```
//go:embed sql/add_attachment.sql
var addAttachmentsSQL string

func (r *Repository) AddAttachment(ctx context.Context,
inspectionID, fileID int, attachmentType
inspection.AttachmentType) (inspection.Attachment, error) {
    var a Attachment
    err := r.db.GetContext(ctx, &a, addAttachmentsSQL,
inspectionID, attachmentType, fileID)
    if err != nil {
        return inspection.Attachment{},
fmt.Errorf("r.db.GetContext: %w", err)
    }

    return MapAttachmentFromDB(a), nil
}
```

Метод `AddAttachment` выполняет SQL-запрос для создания записи о связи между проверкой и файлом в таблице `attachments` и возвращает созданную запись. SQL-скрипт внедряется в код посредством директивы `go:embed`, что дает возможность хранить запросы в непосредственной близости от использующего их кода и обеспечивает контроль синтаксиса на этапе компиляции. После выполнения запроса полученный результат трансформируется из модели базы данных в доменную модель при помощи функции преобразования `MapAttachmentFromDB`.

Взаимодействие всех компонентов и уровней системы рассмотрено на примере процесса добавления фотографии к проверке. Для визуализации этого процесса разработана диаграмма последовательности, представленная на рисунке 35.

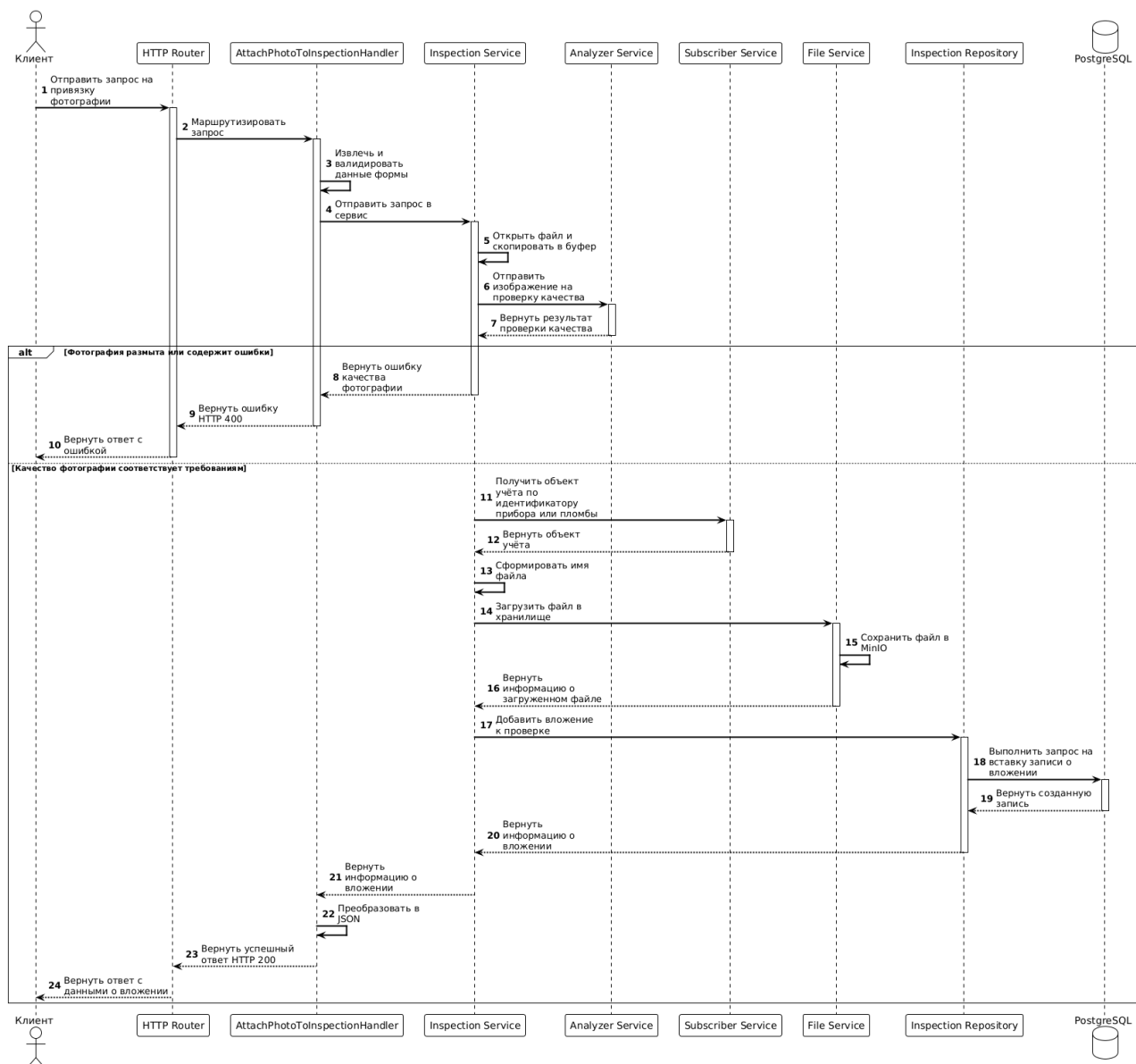


Рисунок 35 – Диаграмма последовательности процесса привязки фотографии к проверке

Процесс добавления фотографии инициируется отправкой клиентом POST-запроса с файлом изображения и метаданными на адрес `/inspections/{id}/photo`. Обработчик `AttachPhotoToInspection` принимает запрос, извлекает информацию из формы запроса, проверяет ее корректность и создает объект запроса для передачи на уровень бизнес-логики. В методе `AttachPhoto` сначала выполняется открытие файла и перенос его содержимого в буфер памяти для дальнейшей обработки. Далее изображение направляется в сервис оценки качества фотографий (`analyzer-service`) посредством HTTP-запроса для

анализа на размытость и наличие дефектов. В случае несоответствия качества фотографии установленным требованиям процесс прекращается с возвратом ошибки клиенту. При успешной проверке качества сервис запрашивает информацию об объекте учета из микросервиса абонентов (subscriber-service) по идентификатору прибора учета или пломбы, указанному в запросе. На основе полученных данных генерируется имя файла, включающее адрес объекта, категорию вложения, номер прибора учета или пломбы и временную метку. Сформированный файл размещается в централизованном файловом хранилище через микросервис файлов (file-service), который сохраняет файл в MinIO и возвращает идентификатор сохраненного файла. После успешного размещения файла репозиторий выполняет SQL-запрос для создания записи в таблице attachments, устанавливающей связь между проверкой и загруженным файлом. Результат операции передается через все уровни обратно клиенту в виде JSON-ответа с информацией о созданном вложении. Подобный многошаговый процесс гарантирует контроль качества фотографий, автоматическое создание имен файлов и надежное сохранение связей между проверками и их вложениями.

4.3 Разработка аналитических дашбордов

Аналитические дашборды являются пользовательским представлением данных, которые подготавливаются серверной частью системы на основе сведений о задачах, проверках, абонентах, объектах энергоснабжения и показаниях приборов учета. Их назначение состоит в том, чтобы предоставить диспетчеру и специалистам отдела контроля не только список отдельных записей, но и обобщенную картину состояния процессов. Такой подход позволяет быстрее выявлять проблемные объекты, оценивать загруженность бригад и контролировать выполнение проверок за выбранный период.

Серверная часть обеспечивает дашборды агрегированными данными через аналитический сервис. В качестве источников используются события жизненного цикла задач и проверок, поступающие через Kafka, а также данные

доменных сервисов: сведения об абонентах, объектах энергоснабжения, приборах учета, результатах проверок и выявленных нарушениях. Агрегированные показатели сохраняются в аналитическом хранилище ClickHouse, что позволяет выполнять тяжелые выборки без нагрузки на основные транзакционные базы данных PostgreSQL.

Первым аналитическим представлением является дашборд обзора операций. Он отображает общее состояние работы отдела контроля: количество задач по статусам, число завершенных проверок, распределение результатов проверок, текущую загрузку бригад и динамику выполнения работ. Данный экран используется диспетчером для оперативной оценки текущей ситуации и принятия решений о перераспределении задач между бригадами. Пример дашборда обзора операций представлен на рисунке 36.

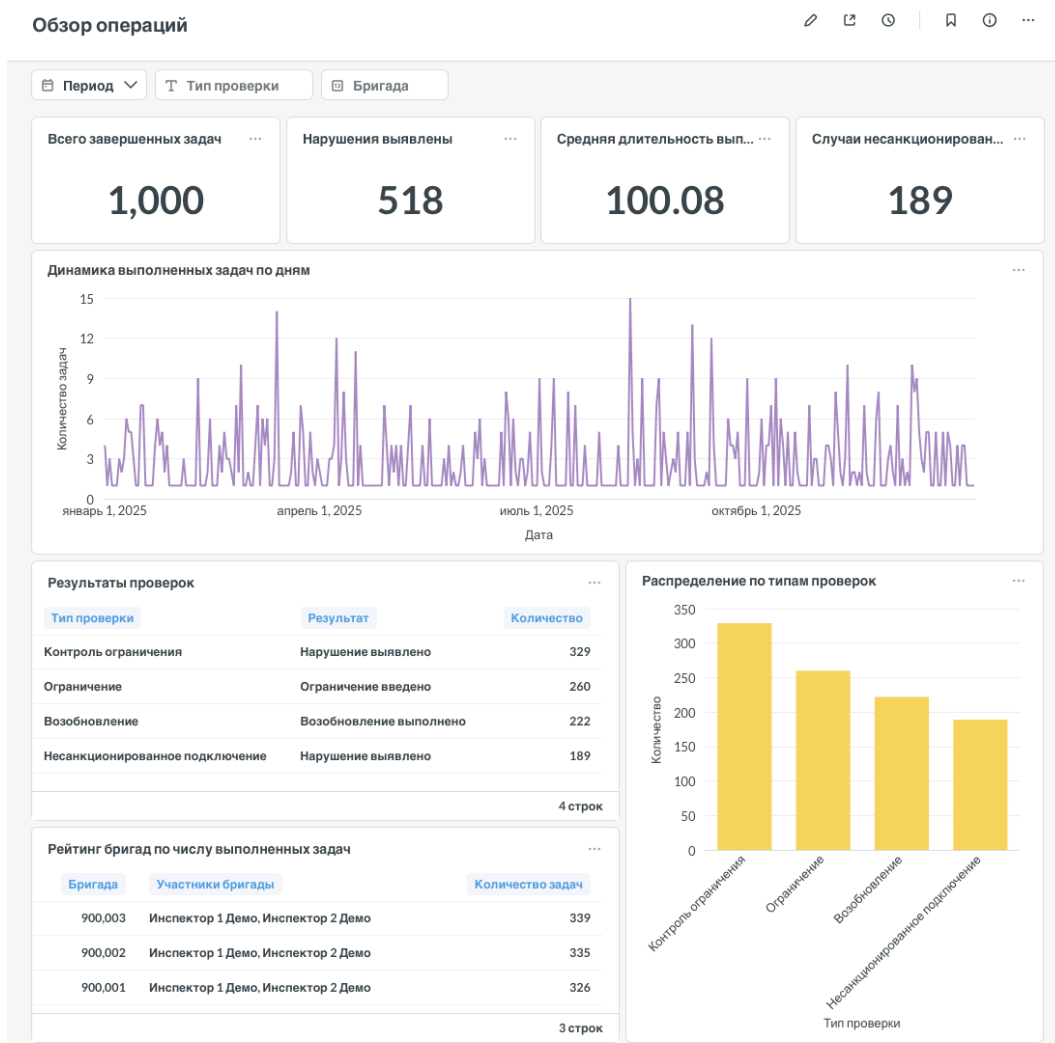


Рисунок 36 – Дашборд обзора операций отдела контроля

Вторым аналитическим представлением является дашборд аномалий потребления. Его задача заключается в выявлении объектов энергоснабжения с нетипичной динамикой расхода электроэнергии. Для этого аналитический сервис сопоставляет текущие и предыдущие значения показаний приборов учета, рассчитывает отклонения и выделяет объекты, по которым наблюдаются резкие изменения потребления или повторяющиеся признаки нарушений. Такой дашборд помогает специалистам по работе с абонентами и диспетчеру быстрее определить объекты, требующие дополнительной проверки. Пример дашборда аномалий потребления представлен на рисунке 37.

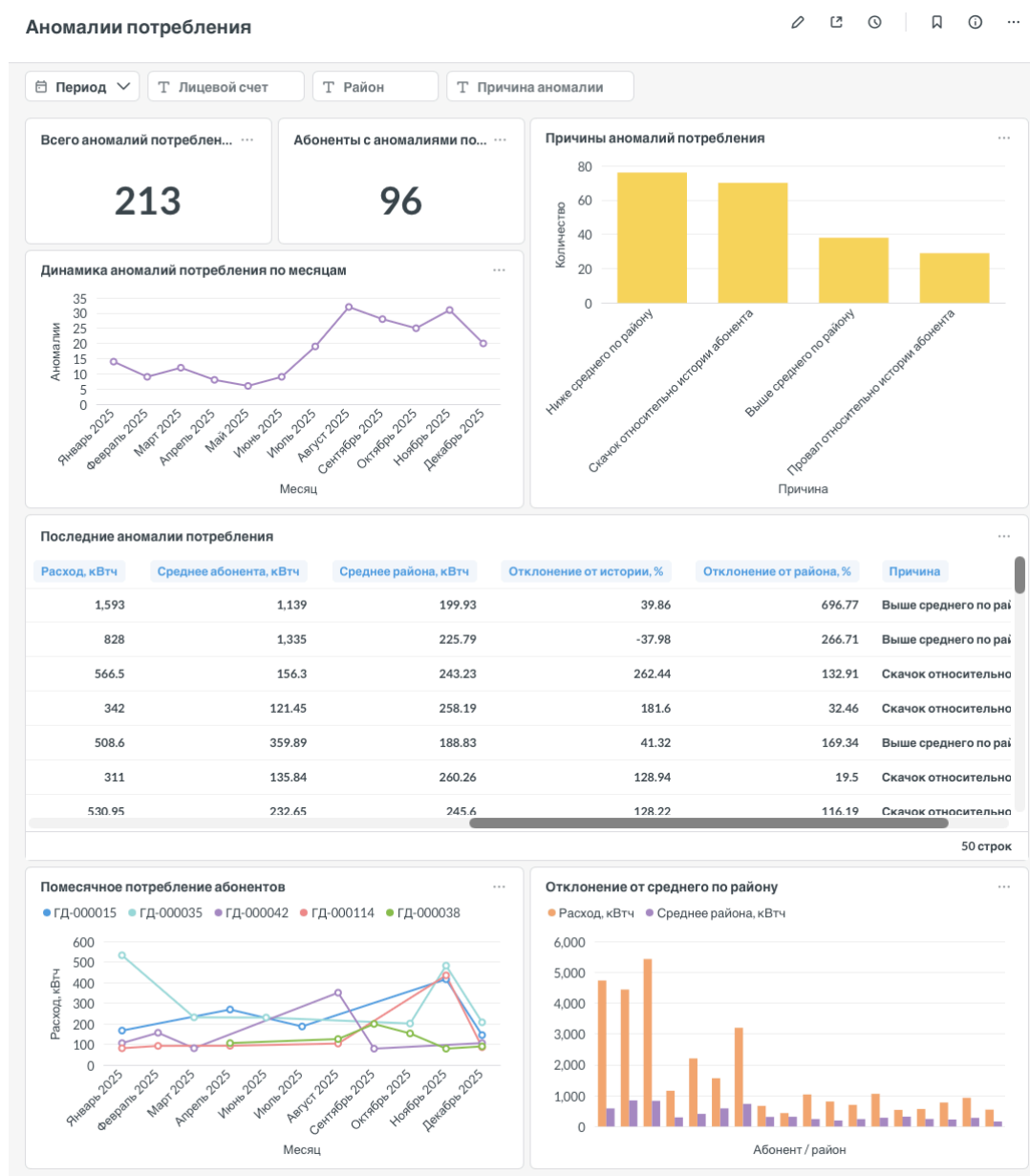


Рисунок 37 – Дашборд аномалий потребления электроэнергии

Для детализации аналитических показателей используется дашборд абонентов и объектов энергоснабжения. Он позволяет перейти от агрегированных показателей к конкретным сущностям: абоненту, объекту, прибору учета, договору и истории проверок. Такая связь между аналитическими дашбордами и предметными данными важна для практической работы отдела контроля, поскольку выявленная аномалия должна быть проверяема по исходным данным. Дашборд абонентов и объектов представлен на рисунке 38.

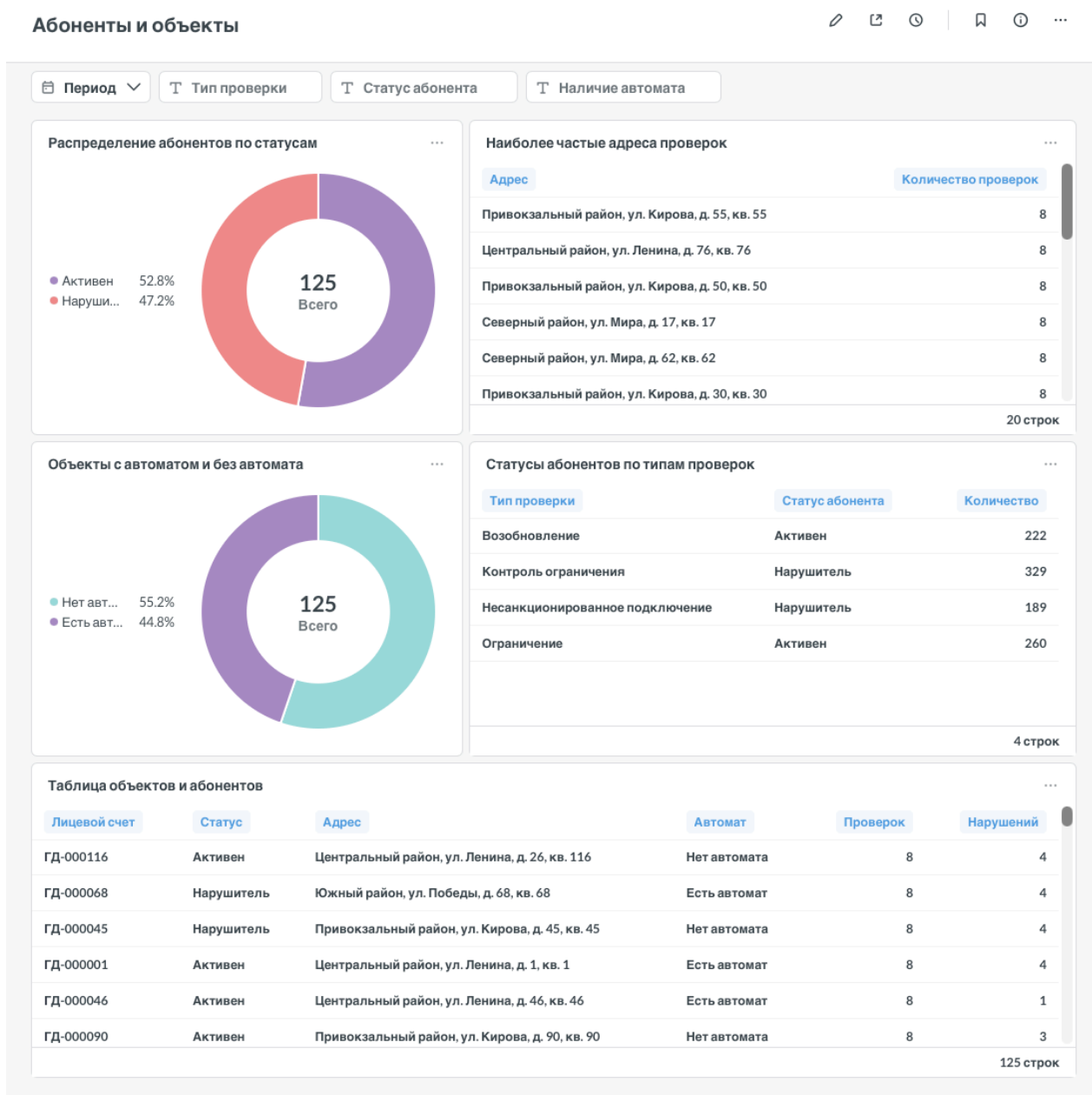


Рисунок 38 – Дашборд абонентов и объектов энергоснабжения

Таким образом, разработанные аналитические дашборды дополняют функции учета и документооборота средствами анализа. Дашборд обзора операций обеспечивает контроль выполнения работ, дашборд аномалий потребления помогает выявлять подозрительные изменения расхода электроэнергии, а дашборд абонентов и объектов связывает агрегированные показатели с конкретными данными предметной области. Совокупность этих представлений подтверждает аналитическую составляющую информационной системы и обосновывает ее назначение как системы контроля и анализа потребления электроэнергии.

5 Контроль качества программного обеспечения

5.1 Обоснование выбора модели тестирования и метрик кода

При создании серверного приложения информационной системы контроля и анализа потребления электроэнергии значительное внимание было уделено вопросам обеспечения качества разрабатываемого программного кода. Применение системного подхода к тестированию и оценке характеристик кода позволяет обеспечить стабильную и безопасную работу каждого микросервиса, входящего в состав системы, а также своевременно обнаруживать дефекты на различных стадиях жизненного цикла разработки.

Для серверной части системы выделены следующие основные виды тестирования:

- модульное тестирование – верификация корректного поведения отдельных функций и компонентов микросервисов в изолированной среде, без обращения к внешним зависимостям [18];
- интеграционное тестирование – проверка правильности взаимодействия между микросервисами, брокером сообщений Kafka, СУБД PostgreSQL и объектным хранилищем MinIO в условиях, приближенных к реальной эксплуатации [19];
- тестирование производительности – измерение времени отклика конечных точек REST, пропускной способности при обработке событий Kafka и устойчивости системы при повышенных нагрузках;
- функциональное тестирование – проверка соответствия реализованного поведения системы заявленным требованиям [20].

Помимо тестирования, важную роль в обеспечении качества играют количественные метрики кода, позволяющие объективно оценить характеристики разработанного программного обеспечения. В рамках данной выпускной квалификационной работы для анализа качества кода микросервисов были выбраны метрики SLOC и метрика Холстеда.

Метрика SLOC (Source Lines of Code) определяет количество строк исходного кода программы и применяется для:

- оценки объема выполненной разработки по каждому микросервису;
- прогнозирования трудозатрат, необходимых для сопровождения и развития отдельных компонентов системы;
- обнаружения участков кода с избыточной сложностью, требующих рефакторинга.

Метрика Холстеда представляет собой набор показателей, вычисляемых на основе подсчета уникальных и общих операторов и операндов в исходном коде. Данная метрика позволяет:

- количественно оценить алгоритмическую сложность реализованных модулей;
- спрогнозировать трудоемкость разработки и сопровождения программного продукта;
- проанализировать потенциальную надежность системы на основе расчетного числа ошибок [21].

Выбор указанных метрик продиктован их способностью обеспечить объективную и воспроизводимую оценку качества кода микросервисов, выявить потенциальные архитектурные проблемы и контролировать рост сложности системы в процессе ее эволюции. Систематический расчет данных показателей на протяжении цикла разработки способствует поддержанию высокого уровня качества серверного приложения информационной системы контроля и анализа потребления электроэнергии.

Совокупность перечисленных подходов к тестированию и анализу кода формирует целостную модель контроля качества, гарантирующую соответствие разработанного программного обеспечения предъявляемым функциональным и нефункциональным требованиям и обеспечивающую его надежное функционирование в условиях промышленной эксплуатации.

5.2 Тестирование серверного приложения информационной системы

5.2.1 Функциональное тестирование обновления статуса абонента по результатам проверки

По завершении разработки серверного приложения информационной системы было проведено функциональное тестирование, направленное на подтверждение корректности реализованного поведения системы. Данный вид тестирования имеет особое значение для обеспечения качества разрабатываемого продукта, поскольку позволяет убедиться в том, что приложение соответствует заявленным требованиям и корректно реагирует на допустимые и недопустимые входные данные.

Для проведения функционального тестирования был выбран процесс обновления статуса абонента при обработке событий о завершении проверки в микросервисе абонентов (subscriber-service). Данный процесс реализован в методе `SubscriberOnInspectionEvent`, который принимает события из `Kafka`, десериализует их и в зависимости от типа и результатов проверки определяет, требуется ли изменение статуса абонента. Выбор данного функционала обусловлен тем, что он охватывает несколько ветвей принятия решений: определение типа события, проверку статуса завершенной проверки, анализ типа проверки и определение нового статуса абонента в зависимости от выявленного нарушения.

Для данного процесса были сформулированы следующие требования:

- система должна игнорировать событие, если его тип не является типом «Завершение»;
- система должна сообщить об ошибке, если тип события не распознан;
- система должна сообщить об ошибке, если статус проверки не равен «Выполнена»;
- система должна сообщить об ошибке, если тип проверки не задан;

- система не должна изменять статус абонента, если тип проверки не относится к верификации или расследованию несанкционированного подключения;
- система должна установить статус абонента «Нарушитель» при обнаружении нарушения;
- система должна установить статус абонента «Активный» при отсутствии нарушения.

Для описания поведения системы были определены 6 причин и 7 следствий. Причинами являются условия, проверяемые в ходе обработки события, а следствиями – результаты, которые система формирует в ответ на определенные комбинации входных условий. Причины и следствия представлены в таблице 8.

Таблица 8 – Причины и следствия

П1	Тип события – «Начало»
П2	Тип события – «Завершение»
П3	Статус проверки – «Выполнена»
П4	Тип проверки задан
П5	Тип проверки релевантен
П6	Обнаружено нарушение
С1	Событие проигнорировано
С2	Ошибка: неизвестный тип события
С3	Ошибка: недопустимый статус проверки
С4	Ошибка: тип проверки не задан
С5	Статус абонента не изменился (тип проверки нерелевантен)
С6	Статус абонента установлен – «Нарушитель»
С7	Статус абонента установлен – «Активный»

На основе таблицы 7 разработана функциональная диаграмма, описывающая поведение системы с учетом всех возможных комбинаций входных условий и возникающих ошибок. Функциональная диаграмма представлена на рисунке 39.

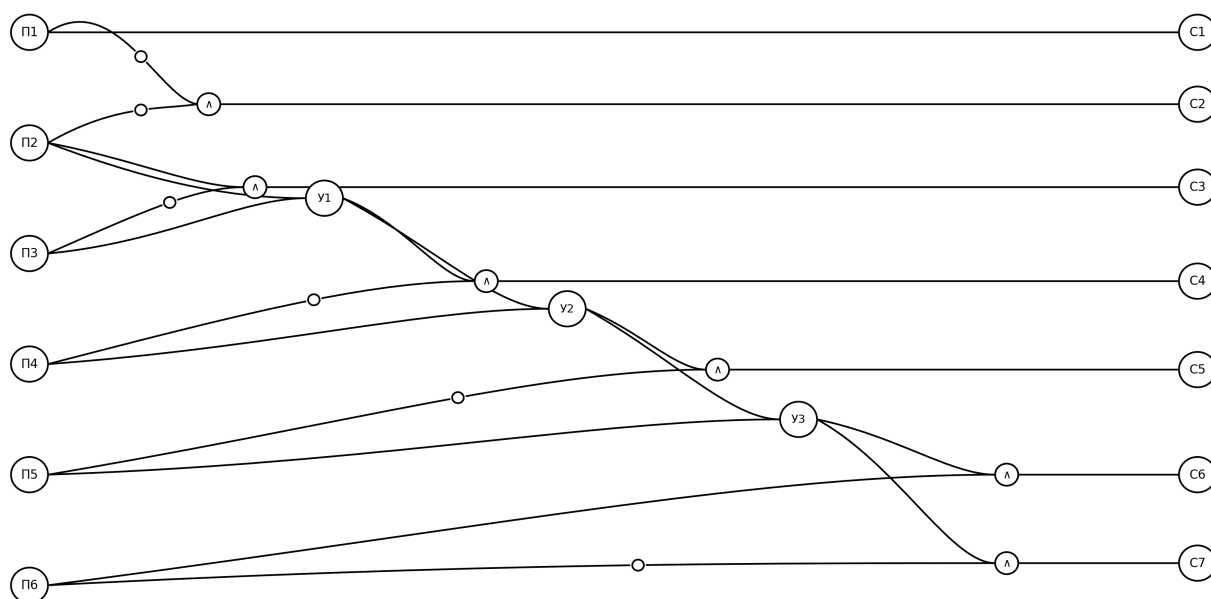


Рисунок 39 – Функциональная диаграмма обновления статуса абонента по результатам проверки

На основе функциональной диаграммы составлена таблица решений, отражающая все допустимые комбинации причин и соответствующие им следствия. Каждый столбец таблицы описывает один вариант поведения системы при определенном наборе входных условий. Таблица решений представлена в таблице 9.

Таблица 9 – Таблица решений

	1	2	3	4	5	6	7
П1	1	0	0	0	0	0	0
П2	0	0	1	1	1	1	1
П3	-	-	0	1	1	1	1
П4	-	-	-	0	1	1	1
П5	-	-	-	-	0	1	1
П6	-	-	-	-	-	0	1
С1	1	0	0	0	0	0	0
С2	0	1	0	0	0	0	0
С3	0	0	1	0	0	0	0
С4	0	0	0	1	0	0	0
С5	0	0	0	0	1	0	0
С6	0	0	0	0	0	0	1
С7	0	0	0	0	0	1	0

В таблице решений символ «-» означает, что значение данной причины не влияет на результат, поскольку решение уже принято на предшествующем этапе проверки условий.

На основе таблицы решений разработаны тест-кейсы, описывающие конкретные входные данные и ожидаемые результаты для каждого варианта поведения системы. Тест-кейсы представлены в таблице 10.

Таблица 10 – Тест-кейсы

№ п/п	Тип события	Статус проверки	Тип проверки	Обнаружено нарушение	Ожидаемый результат
1	Начало	-	-	-	Событие проигнорировано
2	Неизвестный	-	-	-	Ошибка: неизвестный тип события
3	Завершение	В работе	-	-	Ошибка: недопустимый статус
4	Завершение	Выполнена	Не задан	-	Ошибка: тип проверки не задан
5	Завершение	Выполнена	Ограничение	-	Статус абонента не изменился
6	Завершение	Выполнена	Верификация	Нет	Статус абонента – «Активный»
7	Завершение	Выполнена	Верификация	Да	Статус абонента – «Нарушитель»

В результате выполнения функционального тестирования все 7 тест-кейсов были успешно пройдены. Поведение системы в каждом из рассмотренных сценариев полностью соответствует сформулированным требованиям: события типа «Начало» корректно игнорируются, нераспознанные типы событий приводят к регистрации ошибки, проверки с недопустимым статусом или отсутствующим типом отклоняются, а статус абонента обновляется в соответствии с наличием или отсутствием выявленного нарушения.

5.2.2 Модульное тестирование метода создания бригады

С целью верификации корректности работы отдельных компонентов серверного приложения информационной системы было выполнено модульное тестирование. Данный вид тестирования направлен на изолированную проверку логики конкретного метода без обращения к реальным внешним зависимостям – базе данных и сторонним сервисам. Замена зависимостей обеспечивается механизмом подстановки имитационных объектов (mock), что позволяет контролировать возвращаемые значения и моделировать различные сценарии, включая ошибочные.

В качестве объекта модульного тестирования выбран метод CreateBrigade пакета brigade микросервиса brigade-service, реализованного на языке Go. Метод отвечает за формирование бригады инспекторов: он принимает список идентификаторов пользователей, запрашивает их данные через интерфейс UserService, проверяет, что каждый пользователь обладает ролью «Инспектор», и передает сформированный список в репозиторий для сохранения. Выбор данного метода обоснован тем, что он содержит несколько точек принятия решений и взаимодействует с двумя внешними зависимостями, что создает достаточное количество ветвей для покрытия тестами.

Тестирование выполнялось с применением стандартного модуля testing языка Go. Для имитации зависимостей были созданы структуры mockUserService и mockRepository, реализующие интерфейсы UserService и Repository соответственно. Имитационные объекты позволяют задавать заранее определенные ответы и ошибки, благодаря чему каждый тестовый сценарий проверяет строго изолированную ветвь логики метода. Подробный исходный код теста представлен в приложении И.

Пример модульного теста приведен в листинге 7.

Листинг 7 – Пример модульного теста метода CreateBrigade

```
func TestCreateBrigade_Success(t *testing.T) {
    svc := NewService(
```

```

        &mockRepository{brigade: Brigade{ID: 1, Status:
StatusIdle}},
        &mockUserService{users: []user.User{
            {ID: 10, Role: user.RoleInspector, Surname:
"Иванов", Name: "Иван"},
            {ID: 20, Role: user.RoleInspector, Surname:
"Петров", Name: "Пётр"},
        }},
    )

    b, err := svc.CreateBrigade(gotctx.Wrap(context.Background()),
CreateBrigadeRequest{
    InspectorIDs: []int{10, 20},
})
    if err != nil {
        t.Fatalf("unexpected error: %v", err)
    }
    if b.ID != 1 {
        t.Fatalf("expected brigade ID = 1, got %d", b.ID)
    }
    if len(b.Inspectors) != 2 {
        t.Fatalf("expected 2 inspectors, got %d",
len(b.Inspectors))
    }
}

```

Результаты проведения тестов представлены в таблице 11.

Таблица 11 – Результаты модульного тестирования метода CreateBrigade

№ п/п	Входные данные	Ожидаемый результат	Фактический результат	Статус теста
1	Два пользователя с ролью Inspector, корректные идентификаторы	Бригада успешно создана, содержит двух инспекторов	Бригада создана с ID = 1, число инспекторов равно 2	Пройден
2	UserService возвращает ошибку «user service unavailable»	Метод возвращает ошибку, бригада не создается	Получена ошибка «get users: user service unavailable»	Пройден
3	Запрошены два идентификатора, UserService возвращает одного пользователя	Метод возвращает ошибку несовпадения количества	Получена ошибка «want 2 users, got 1»	Пройден
4	Один пользователь с ролью Inspector, второй – с ролью Dispatcher	Метод возвращает ошибку: пользователь не является инспектором	Получена ошибка «user 20 is not inspector: role = 2»	Пройден

Продолжение таблицы 11

№ п/п	Входные данные	Ожидаемый результат	Фактический результат	Статус теста
5	Корректный пользователь, Repository возвращает ошибку «database error»	Метод возвращает ошибку сохранения в репозитории	Получена ошибка «create brigade in repository: database error»	Пройден

Результаты выполнения тестов в среде разработки приведены на рисунке 40.

Test Results		359 ms
✓	TestCreateBrigade_Success	0 ms
✓	TestCreateBrigade_UserServiceError	0 ms
✓	TestCreateBrigade_UserCountMismatch	0 ms
✓	TestCreateBrigade_UserNotInspector	0 ms
✓	TestCreateBrigade_RepositoryError	0 ms

Рисунок 40 – Результаты выполнения модульных тестов

Все пять модульных тестов были успешно пройдены. Методы CreateBrigade корректно обрабатывает как штатные, так и аварийные ситуации: при недоступности сервиса пользователей возвращается соответствующая ошибка, при расхождении числа запрошенных и полученных пользователей формируется диагностическое сообщение, попытка включить в бригаду пользователя с неподходящей ролью отклоняется, а сбой на уровне репозитория корректно транслируется вызывающему коду.

5.2.3 Метрики Холстеда для функции валидации лицевого счета абонента

Метрики Холстеда представляют собой набор количественных характеристик программного кода, вычисляемых на основе подсчета синтаксических элементов – операторов и операндов. Эти метрики позволяют оценить объем, сложность и трудоемкость сопровождения программы без

необходимости ее запуска, что делает их пригодными для статического анализа качества кода.

В качестве объекта для расчета метрик выбрана функция `ValidateAccountNumber` пакета `subscriber` микросервиса `subscriber-service`. Данная функция выполняет валидацию лицевого счета абонента в соответствии с набором бизнес-правил: проверяет длину строки, допустимость символов, наличие обязательных компонентов (прописных русских букв, цифр и дефисов), а также отсутствие последовательных дефисов. Код функции представлен в приложении Е.

Для проведения анализа необходимо определить количество уникальных операторов и операндов, а также общее число их вхождений в исходном коде функции. Перечень операторов с указанием частоты встречаемости приведен в таблице 12.

Таблица 12 – Операторы функции `ValidateAccountNumber`

Оператор	Количество
<code>return</code>	8
<code>if</code>	6
<code>:=</code>	4
<code>=</code>	4
<code>!</code>	3
<code>case</code>	3
<code>isUpperRussianLetter</code>	2
<code>isDigit</code>	2
<code>[]</code> (индексация)	2
<code>[]rune</code> (преобразование типа)	1
<code>len</code>	1
<code>for range</code>	1
<code>switch</code>	1
<code>default</code>	1
<code><</code>	1
<code>></code>	1
<code>==</code>	1
<code>-</code> (вычитание)	1
<code>n1 = 18</code>	<code>N1 = 43</code>

Перечень операндов с указанием частоты встречаемости приведен в таблице 13.

Таблица 13 – Операнды функции ValidateAccountNumber

Операнд	Количество вхождений
prevHyphen	5
runes	4
length	4
r	4
false	4
hasHyphen	3
true	2
accountNumber	1
accountNumberMinLength	1
accountNumberMaxLength	1
0	1
1	1
‘_’	1
nil	1
ErrAccountNumberTooShort	1
ErrAccountNumberTooLong	1
ErrAccountNumberStartNotLetter	1
ErrAccountNumberEndNotDigit	1
ErrAccountNumberConsecutiveHyphens	1
ErrAccountNumberInvalidChar	1
ErrAccountNumberNoHyphen	1
n2 = 21	N2 = 40

На основании полученных данных вычисляется длина программы по формуле:

$$N=N1+N2=43+40=83$$

Словарь программы определяется как сумма уникальных элементов:

$$n=n1+n2=18+21=39$$

Объем программы (Halstead Volume) рассчитывается по формуле:

$$HV=N \times \log_2 n = 83 \times 5,285 = 438,66$$

Для вычисления комплексного показателя качества кода – индекса сопровождаемости (Maintainability Index) – необходимы дополнительные характеристики: цикломатическая сложность и число строк кода. Цикломатическая сложность функции ValidateAccountNumber составляет $CC = 11$, что обусловлено наличием шести условных операторов if, трех ветвей case в конструкции switch и одного цикла for. Число строк исполняемого кода (без пустых строк и комментариев) составляет $LOC = 38$.

Индекс сопровождаемости вычисляется по формуле Олдена-Коулмана:

$$MI=171-5,2\times \ln HV -0,23\times CC-16,2\times \ln LOC =77,9$$

Полученное значение индекса сопровождаемости $MI = 77,9$ превышает пороговое значение 65, что свидетельствует о высоком уровне сопровождаемости анализируемого кода. Функция обладает умеренной цикломатической сложностью, обусловленной характером задачи (многоуровневая валидация), однако каждая ветвь логики реализована линейно и не содержит вложенных вызовов. Объем программы также находится в допустимых границах, что подтверждает прозрачность кодовой базы и невысокую трудоемкость ее модификации.

Заключение

В ходе выполнения выпускной квалификационной работы было спроектировано и реализовано серверное приложение информационной системы контроля и анализа потребления электроэнергии, предназначенной для автоматизации деятельности отдела контроля энергосбытовой организации.

Серверное приложение информационной системы реализовано на языке программирования Go с применением микросервисной архитектуры. Приложение включает в себя семь сервисов на Go и один сервис на Python с использованием фреймворка FastAPI для анализа качества фотографий. Межсервисное взаимодействие организовано по протоколу HTTP на основе архитектурного стиля REST, а асинхронный обмен событиями обеспечивается посредством брокера сообщений Apache Kafka. Для хранения данных применяются СУБД PostgreSQL, аналитическое хранилище ClickHouse и объектное хранилище MinIO. Единая точка входа в систему реализована с помощью прокси-сервера Nginx в роли API-шлюза, а средства наблюдаемости представлены связкой Prometheus и Grafana. Контейнеризация и развертывание компонентов осуществляются при помощи Docker и Docker Compose.

На этапе анализа предметной области было исследовано понятие контроля и анализа потребления электроэнергии, описаны участники бизнес-процесса и их функциональные обязанности, выявлены проблемы в текущем бизнес-процессе отдела контроля. Проведен сравнительный анализ существующих информационных систем аналогичного назначения – платформы СТЕК, программно-аппаратного комплекса АСУ ПКПЭ и платформы SafetyCulture, - в результате которого было установлено, что ни одно из рассмотренных решений не удовлетворяет требованиям отдела контроля. Были сформулированы функциональные и нефункциональные требования к разрабатываемым программным средствам, определены три категории пользователей системы: инспекторы, диспетчеры и специалисты по работе с абонентами.

На стадии проектирования была определена микросервисная архитектура в качестве основы системы, включающая девять микросервисов, каждый из которых отвечает за отдельную предметную область. Спроектированы концептуальная модель базы данных в нотации IDEF1X, включающая 17 сущностей и 18 связей, а также логические модели семи баз данных, разделенных по предметным областям. Разработаны диаграммы компонентов и развертывания, определены минимальные и рекомендуемые требования к аппаратному обеспечению серверных узлов.

Реализация серверной части включала разработку многослойной архитектуры внутри каждого микросервиса с выделением слоя представления данных, слоя бизнес-логики и слоя доступа к данным. Реализованы механизмы обработки HTTP-запросов, валидации входных данных, взаимодействия с базами данных через библиотеку `sqlx`, публикации и потребления событий Kafka, автоматической генерации документов в формате DOCX, проверки качества загружаемых фотографий, автоматического формирования имен файлов, генерации аналитических отчетов в формате Excel и подготовки агрегированных данных для дашбордов аномалий потребления и обзора операций. Разработаны физические модели баз данных, включающие 29 сущностей различных категорий.

В рамках контроля качества программного обеспечения было проведено функциональное тестирование процесса обновления статуса абонента с применением метода причинно-следственного графа и таблиц решений, по итогам которого все семь тест-кейсов были успешно пройдены. Выполнено модульное тестирование метода создания бригады инспекторов с использованием имитационных объектов, все пять тестовых сценариев подтвердили корректную работу метода. Произведен расчет метрик Холстеда для функции валидации лицевого счета абонента, индекс сопровождаемости которой составил 77,9, что превышает пороговое значение 65 и свидетельствует о высоком уровне сопровождаемости кода.

В ходе функционально-стоимостного анализа были рассчитаны затраты на разработку проекта. Суммарные трудозатраты составили 400 часов, а итоговая стоимость разработки – 184 000 рублей. Работа над проектом запланирована на период с 8 апреля по 3 июня текущего года.

Таким образом, по итогам выполнения выпускной квалификационной работы разработано серверное приложение информационной системы контроля и анализа потребления электроэнергии, в полной мере удовлетворяющее поставленным целям и задачам. Разработанная система обеспечивает автоматизацию документооборота, ускорение проведения проверок, повышение прозрачности работы бригад инспекторов, контроль качества фотографий приборов учета, формирование аналитической отчетности, выявление аномалий потребления и оперативный обзор операций отдела контроля, что в совокупности направлено на повышение эффективности работы отдела контроля и анализа потребления электроэнергии энергосбытовой организации.

Список использованных источников

1. Информационные системы: определение и методологии создания [Электронный ресурс]. – URL: <https://otus.ru/journal/informacionnye-sistemy-opredelenie-i-metodologii-sozdaniya/> (дата обращения: 12.04.2026).
2. Контроль учета электроэнергии [Электронный ресурс]. – URL: <https://cometa.ru/blog/detail/kontrol-ucheta-elektroenergii/> (дата обращения: 12.04.2026).
3. Система управления планированием и контроля потребления электроэнергии [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: 15.04.2026).
4. Князькова В.С. Развитие функционально-стоимостного анализа и особенности его применения при проектировании программных продуктов // *π-Economy*. – 2022. – Т. 15, № 5. – С. 111–121. – DOI: <https://doi.org/10.18721/JE.15507> (дата обращения: 14.04.2026).
5. Статистика рынка труда в Пензе – «Программист-разработчик» [Электронный ресурс] // ГородРабот.ру. – URL: <https://penza.gorodrabot.ru/salaries/programmist-razrabotchik> (дата обращения: 14.04.2026).
6. Липаев В.В. Программная инженерия сложных заказных программных продуктов: Учебное пособие. – М.: МАКС Пресс, 2014. – 312 с.
7. Вигерс К., Битти Д. Разработка требований к программному обеспечению. 3-е изд., дополненное / Пер. с англ. — М. : Издательство «Русская редакция» ; СПб. : БХВ-Петербург, 2014. — 736 стр.
8. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс. – 496 с.

9. Ричардс М., Форд Н. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы. — СПб.: Питер, 2023. — 448 с.
10. Ньюмен С. Создание микросервисов. 2-е изд. — СПб.: Питер, 2023. — 624 с.
11. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2 // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения: 17.04.2026).
12. Дергачев А.М., Кореньков Ю.Д., Логинов И.П., Сафронов А.Г. Технологии веб-сервисов — СПб: Университет ИТМО, 2021. — 100 с.
13. Что такое Apache Kafka? // <https://aws.amazon.com> [Электронный ресурс] — URL: <https://aws.amazon.com/ru/what-is/apache-kafka/> (дата обращения 17.04.2026).
14. Просто о сложном. Концептуальная модель. // <https://mlkv.ru> [Электронный ресурс] — URL: <https://mlkv.ru/sample-page/материалы-по-базам-данных/концептуальная-модель/> (дата обращения: 18.04.2026).
15. Верников Г. Основы методологии IDEF1X // Проект «Корпоративный менеджмент» [Электронный ресурс] — URL: <https://www.cfin.ru/vernikov/idef/idef1x.shtml> (дата обращения: 18.04.2026).
16. В чем разница между логической моделью данных и физической моделью данных? // <https://aws.amazon.com> [Электронный ресурс] — URL: <https://aws.amazon.com/ru/compare/the-difference-between-logical-and-physical-data-model/> (дата обращения: 20.04.2026).
17. Как привести данные в форму: что такое нормализация и зачем она нужна // <https://practicum.yandex.ru> [Электронный ресурс] — URL: <https://practicum.yandex.ru/blog/chto-takoe-normalizaciya-dannyh/> (дата обращения: 20.04.2026).
18. Юнит-тестирование для чайников // <https://habr.com> [Электронный ресурс] — URL: <https://habr.com/ru/articles/169381/> (дата обращения: 23.04.2026).

19. Интеграционное тестирование: виды, примеры и инструменты // <https://habr.com> [Электронный ресурс] – URL: <https://habr.com/ru/articles/843534/> (дата обращения: 23.04.2026).

20. Фундаментальная теория тестирования // <https://habr.com> [Электронный ресурс] – URL: <https://habr.com/ru/articles/549054/> (дата обращения: 23.04.2026).

21. Программный код и его метрики // <https://habr.com> [Электронный ресурс] – URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 23.04.2026).

Приложение А
(справочное)
Глоссарий

Микросервис – это компактный автономный компонент информационной системы, отвечающий за выполнение одной конкретной функции. Он взаимодействует с другими сервисами через четко определенные интерфейсы, обеспечивая независимую разработку, развертывание и масштабирование.

API (Application Programming Interface) – это программный интерфейс, который определяет правила и методы взаимодействия между различными приложениями или компонентами системы. Он позволяет одной программе запрашивать данные или функции у другой через стандартизированные вызовы.

API шлюз – это централизованный компонент системы, служащий единой точкой входа для внешних запросов к множеству серверных сервисов. Он выполняет маршрутизацию трафика, аутентификацию пользователей, ограничение скорости запросов и преобразование данных между клиентами и внутренними сервисами.

Серверное приложение – это программное обеспечение, развернутое на серверной инфраструктуре и предназначенное для обработки запросов от клиентов, выполнения бизнес-логики и управления данными. Оно обеспечивает централизованный доступ к функциям системы через сеть, поддерживая взаимодействие с базами данных и другими сервисами.

Предметная область – это сфера реальной деятельности, включающая ключевые объекты, процессы и взаимосвязи, которые подлежат моделированию в информационной системе. Она определяет контекст и границы автоматизируемых задач, служа основой для проектирования функциональности.

Дашборд – это интерактивное аналитическое представление данных, предназначенное для быстрого контроля ключевых показателей системы. В рамках информационной системы контроля и анализа потребления электроэнергии дашборды используются для отображения аномалий потребления, статусов задач, загрузки бригад, результатов проверок и других агрегированных показателей работы отдела контроля.

Контейнеризация – это технология виртуализации на уровне операционной системы, позволяющая упаковывать приложение вместе со всеми его зависимостями в легковесный изолированный контейнер. Такой подход обеспечивает одинаковую работу ПО на разных средах без конфликтов версий библиотек и настроек.

Бизнес-логика – это совокупность правил, процессов и операций, определяющих поведение приложения в соответствии с задачами предметной области. Она реализует ключевые бизнес-процессы, такие как проверки, расчеты и координацию данных между компонентами системы.

Репозиторий – это архитектурный шаблон проектирования, создающий прослойку между бизнес-логикой приложения и хранилищем данных для абстрагирования операций доступа. Он предоставляет удобный интерфейс, скрывая детали работы с базой данных и упрощая тестирование и замену конкретных технологий хранения.

Приложение Б
(рекомендуемое)
Спецификация атрибутов таблиц

Таблица Б.1 – Спецификация атрибутов таблицы roles (user-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор роли	int	PK	Да	generated always as identity	
2	name	Наименование роли	text		Да		

Таблица Б.2 – Спецификация атрибутов таблицы users (user-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор пользователя	int	PK	Да	generated always as identity	
2	role_id	Идентификатор роли	int	FK	Да	references roles (id) on delete restrict	
3	surname	Фамилия пользователя	text		Да		
4	name	Имя пользователя	text		Да		
5	patronymic	Отчество пользователя	text		Нет		
6	phone_number	Контактный телефон	text		Да	unique, check (формат +7/8 и 10 цифр)	
7	email	Адрес электронной почты	text		Да	unique, check (формат email)	
8	password_hash	Хеш пароля пользователя	text		Нет		
9	refresh_token	Токен обновления сессии	text		Нет		
10	refresh_token_expired_after	Время истечения токена обновления	timestampz		Нет		

Продолжение таблицы Б.2

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
11	created_at	Дата и время создания записи	timestampz		Да		now()
12	updated_at	Дата и время последнего обновления	timestampz		Да		now()

Таблица Б.3 – Спецификация атрибутов таблицы subscriber_statuses (subscriber-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор статуса абонента	int	PK	Да	generated always as identity	
2	name	Наименование статуса	text		Да		

Таблица Б.4 – Спецификация атрибутов таблицы subscribers (subscriber-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор абонента	int	PK	Да	generated always as identity	
2	account_number	Лицевой счет абонента	text		Да	unique	
3	surname	Фамилия абонента	text		Да		
4	name	Имя абонента	text		Да		
5	patronymic	Отчество абонента	text		Нет		

Продолжение таблицы Б.4

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
6	phone_number	Контактный телефон	text		Да	unique, check (формат +7/8 и 10 цифр)	
7	email	Адрес электронной почты	text		Да	unique, check (формат email)	
8	inn	ИНН	text		Да	check (10-12 цифр)	
9	birth_date	Дата рождения	date		Да	check (>= 1900-01-01, < текущей даты)	
10	status	Статус абонента	int	FK	Да	references subscriber_statuses (id)	1
11	created_at	Дата и время создания записи	timestampz		Да		now()
12	updated_at	Дата и время последнего обновления	timestampz		Да		now()

Таблица Б.5 – Спецификация атрибутов таблицы passports (subscriber-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор паспорта	int	PK	Да	generated always as identity	
2	subscriber_id	Идентификатор абонента	int	FK	Да	references subscribers (id) on delete cascade	

Продолжение таблицы Б.5

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
3	series	Серия паспорта	text		Да	check (4 цифры), unique (series, number)	
4	number	Номер паспорта	text		Да	check (6 цифр), unique (series, number)	
5	issued_by	Кем выдан паспорт	text		Да		
6	issue_date	Дата выдачи паспорта	date		Да		

Таблица Б.6 – Спецификация атрибутов таблицы objects (subscriber-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор объекта учета	int	PK	Да	generated always as identity	
2	address	Адрес объекта	text		Да	unique	
3	have_automation	Наличие коммутационного аппарата	bool		Да		
4	created_at	Дата и время создания записи	timestampz		Да		now()
5	updated_at	Дата и время последнего обновления	timestampz		Да		now()

Таблица Б.7 – Спецификация атрибутов таблицы device_place_types (subscriber-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор типа места установки	int	PK	Да	generated always as identity	
2	name	Наименование типа места установки	text		Да		

Таблица Б.8 – Спецификация атрибутов таблицы devices (subscriber-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор прибора учета	int	PK	Да	generated always as identity	
2	object_id	Идентификтор объекта учета	int	FK	Да	references objects (id) on delete cascade	
3	type	Тип прибора учета	text		Да		
4	number	Заводской номер прибора	text		Да	unique	
5	place_type	Тип места установки прибора	int	FK	Да	references device_place_types (id)	
6	place_description	Описание места установки	text		Да		
7	created_at	Дата и время создания записи	timestampz		Да		now()
8	updated_at	Дата и время последнего обновления	timestampz		Да		now()

Таблица Б.9 – Спецификация атрибутов таблицы seals (subscriber-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор пломбы	int	PK	Да	generated always as identity	
2	device_id	Идентификатор прибора учета	int	FK	Да	references devices (id) on delete cascade	
3	number	Номер пломбы	text		Да	unique	
4	place	Место установки пломбы	text		Да		
5	created_at	Дата и время создания записи	timestampz		Да		now()
6	updated_at	Дата и время последнего обновления	timestampz		Да		now()

Таблица Б.10 – Спецификация атрибутов таблицы contracts (subscriber-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор договора	int	PK	Да	generated always as identity	
2	number	Номер договора	text		Да	unique	
3	subscriber_id	Идентификатор абонента	int	FK	Да	references subscribers (id) on delete restrict	
4	object_id	Идентификатор объекта учета	int	FK	Да	references objects (id) on delete restrict	
5	sign_date	Дата подписания договора	date		Да	check (<= текущей даты)	

Продолжение таблицы Б.10

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
6	created_at	Дата и время создания записи	timestampz		Да		now()
7	updated_at	Дата и время последнего обновления	timestampz		Да		now()

Таблица Б.11 – Спецификация атрибутов таблицы brigade_statuses (brigade-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор статуса бригады	int	PK	Да	generated always as identity	
2	name	Наименование статуса	text		Да		

Таблица Б.12 – Спецификация атрибутов таблицы brigades (brigade-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор бригады	int	PK	Да	generated always as identity	
2	status	Текущий статус бригады	int	FK	Да	references brigade_statuses (id)	
3	created_at	Дата и время создания записи	timestampz		Да		now()
4	updated_at	Дата и время последнего обновления	timestampz		Да		now()

Таблица Б.13 – Спецификация атрибутов таблицы brigade_members (brigade-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	brigade_id	Идентификатор бригады	int	PK, FK	Да	references brigades (id) on delete cascade	
2	inspector_id	Идентификатор инспектора	int	PK	Да		
3	assigned_at	Дата и время назначения в бригаду	timestamp_tz		Да		now()

Таблица Б.14 – Спецификация атрибутов таблицы task_statuses (task-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор статуса задачи	int	PK	Да	generated always as identity	
2	name	Наименование статуса	text		Да		

Таблица Б.15 – Спецификация атрибутов таблицы tasks (task-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор задачи	int	PK	Да	generated always as identity	
2	brigade_id	Идентификатор бригады	int		Нет		
3	object_id	Идентификатор объекта учета	int		Да		

Продолжение таблицы Б.15

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
4	plan_visit_at	Запланированное время визита	timestampz		Нет		
5	status	Текущий статус задачи	int	FK	Да	references task_statuses (id)	
6	comment	Комментарий к задаче	text		Нет		
7	started_at	Время начала выполнения	timestampz		Нет		
8	finished_at	Время завершения выполнения	timestampz		Нет	check (started_at <= finished_at)	
9	created_at	Дата и время создания записи	timestampz		Да		now()
10	updated_at	Дата и время последнего обновления	timestampz		Да		now()

Таблица Б.16 – Спецификация атрибутов таблицы inspection_statuses (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор статуса проверки	int	PK	Да	generated always as identity	
2	name	Наименование статуса	text		Да		

Таблица Б.17 – Спецификация атрибутов таблицы inspection_types (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор типа проверки	int	PK	Да	generated always as identity	
2	name	Наименование типа проверки	text		Да		

Таблица Б.18 – Спецификация атрибутов таблицы inspection_resolutions (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор результата проверки	int	PK	Да	generated always as identity	
2	name	Наименование результата	text		Да		

Таблица Б.19 – Спецификация атрибутов таблицы inspection_methods_by (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор способа введения	int	PK	Да	generated always as identity	
2	name	Наименование способа	text		Да		

Таблица Б.20 – Спецификация атрибутов таблицы inspection_reason_types (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор типа основания	int	PK	Да	generated always as identity	
2	name	Наименование типа основания	text		Да		

Таблица Б.21 – Спецификация атрибутов таблицы inspections (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор проверки	int	PK	Да	generated always as identity	
2	task_id	Идентификатор задачи	int		Да	unique	
3	status	Статус проверки	int	FK	Да	references inspection_statuses (id)	
4	type	Тип проверки	int	FK	Нет	references inspection_types (id)	
5	resolution	Результат проверки	int	FK	Нет	references inspection_resolutions (id)	
6	limit_reason	Основание введения ограничения режима потребления	text		Нет		
7	method	Способ введения ограничения/возобновления	text		Нет		

Продолжение таблицы Б.21

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
8	method_by	Кем было введено ограничение	int	FK	Нет	references inspection_methods_by (id)	
9	reason_type	Тип основания/причины	int	FK	Нет	references inspection_reason_types (id)	
10	reason_description	Описание причины невведения ограничения	text		Нет		
11	is_restrictor_checked	Произведена ли проверка введенного ограничения	bool		Нет		
12	is_violation_detected	Выявлено ли нарушение потребителем	bool		Нет		
13	is_expense_available	Наличие расхода после введенного ограничения	bool		Нет		
14	violation_description	Описание выявленного нарушения	text		Нет		
15	is_unauthorized_consumers	Самовольное подключение энергопринимающих устройств	bool		Нет		
16	unauthorized_description	Описание места и способа самовольного подключения	text		Нет		
17	unauthorized_explanation	Объяснение лица, допустившего самовольное подключение	text		Нет		

Продолжение таблицы Б.21

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
18	inspect_at	Дата и время проведения проверки	timestampz		Нет		
19	energy_action_at	Время действия над подачей электроэнергии	timestampz		Нет		
20	created_at	Дата и время создания записи	timestampz		Да		now()
21	updated_at	Дата и время последнего обновления	timestampz		Да		now()

Таблица Б.22 – Спецификация атрибутов таблицы inspected_devices (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор записи обследования прибора	int	PK	Да	generated always as identity	
2	device_id	Идентификатор прибора учета	int		Да		
3	inspection_id	Идентификатор проверки	int	FK	Да	references inspections (id) on delete cascade	
4	value	Текущее показание прибора	numeric(15, 2)		Да		
5	consumption	Расход электрической энергии, кВт/ч	numeric(15, 2)		Да		
6	created_at	Дата и время создания записи	timestampz		Да		now()

Таблица Б.23 – Спецификация атрибутов таблицы inspected_seals (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор записи осмотра пломбы	int	PK	Да	generated always as identity	
2	seal_id	Идентификатор пломбы	int		Да		
3	inspection_id	Идентификатор проверки	int	FK	Да	references inspections (id) on delete cascade	
4	is_broken	Признак срыва пломбы	bool		Да		
5	created_at	Дата и время создания записи	timestampz		Да		now()

Таблица Б.24 – Спецификация атрибутов таблицы attachment_types (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор типа вложения	int	PK	Да	generated always as identity	
2	name	Наименование типа вложения	text		Да		

Таблица Б.25 – Спецификация атрибутов таблицы attachments (inpection-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор вложения	int	PK	Да	generated always as identity	

Продолжение таблицы Б.25

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательны й	Ограничения	Значение по умолчанию
2	inspection_id	Идентификатор проверки	int	FK	Да	references inspections (id) on delete cascade	
3	type	Тип вложения	int	FK	Да	references attachment_types (id)	
4	file_id	Идентификатор файла	int		Да		
5	created_at	Дата и время создания записи	timestampz		Да		now()

Таблица Б.26 – Спецификация атрибутов таблицы storage_files (file-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательны й	Ограничения	Значение по умолчанию
1	id	Идентификатор файла	int	PK	Да	generated always as identity	
2	file_name	Имя файла	text		Да		
3	file_size	Размер файла в байтах	int		Да		
4	bucket	Наименование бакета в MinIO	text		Да		
5	url	Ссылка на скачивание файла	text		Да		

Таблица Б.27 – Спецификация атрибутов таблицы report_types (analytics-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор типа отчета	int	PK	Да	generated always as identity	
2	name	Наименование типа отчета	text		Да		

Таблица Б.28 – Спецификация атрибутов таблицы reports (analytics-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор отчета	int	PK	Да	generated always as identity	
2	type	Тип отчета	int	FK	Нет	references report_types (id)	
3	period_start	Начало отчетного периода	date		Да		
4	period_end	Окончание отчетного периода	date		Да		
5	created_at	Дата и время создания записи	timestampz		Да		now()

Таблица Б.29 – Спецификация атрибутов таблицы attachments (analytics-service)

№ п/п	Наименование	Описание	Тип данных	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор вложения отчета	int	PK	Да	generated always as identity	
2	report_id	Идентификатор отчета	int	FK	Да	references reports (id) on delete cascade	
3	file_id	Идентификатор файла	int		Да		
4	created_at	Дата и время создания записи	timestampz		Да		now()

Приложение В
(рекомендуемое)
Объекты обращения к базам данных

Листинг В.1 – Объекты аналитики

```

type Report struct {
    ID          int          `db:"id"`
    Type        int          `db:"type"`
    PeriodStart time.Time   `db:"period_start"`
    PeriodEnd   time.Time   `db:"period_end"`
    CreatedAt   time.Time   `db:"created_at"`
}

type Attachment struct {
    ID          int          `db:"id"`
    ReportID    int          `db:"report_id"`
    FileID      int          `db:"file_id"`
    CreatedAt   time.Time   `db:"created_at"`
}

type FinishedTask struct {
    TaskID          int64          `ch:"task_id"`
    Comment         *string        `ch:"comment"`
    PlanVisitAt     *time.Time
    `ch:"plan_visit_at"`
    StartedAt       time.Time
    `ch:"started_at"`
    FinishedAt      time.Time
    `ch:"finished_at"`
    InspectionID    int64
    `ch:"inspection_id"`
    InspectionType  int8
    `ch:"inspection_type"`
    InspectionResolution int8
    `ch:"inspection_resolution"`
    InspectionLimitReason *string
    `ch:"inspection_limit_reason"`
    InspectionMethod string
    `ch:"inspection_method"`
    InspectionMethodBy int8
    `ch:"inspection_method_by"`
    InspectionReasonType int8
    `ch:"inspection_reason_type"`
    InspectionReasonDescription *string
    `ch:"inspection_reason_description"`
    InspectionIsRestrictionChecked bool
    `ch:"inspection_is_restriction_checked"`
    InspectionIsViolationDetected bool
    `ch:"inspection_is_violation_detected"`
    InspectionIsExpenseAvailable bool
    `ch:"inspection_is_expense_available"`
    InspectionViolationDescription *string
    `ch:"inspection_violation_description"`
}

```

```

type Inspector struct {
    ID            int64      `ch:"id"`
    Surname       string     `ch:"surname"`
    Name          string     `ch:"name"`
    Patronymic    string     `ch:"patronymic"`
    PhoneNumber   string     `ch:"phone_number"`
    Email         string     `ch:"email"`
    AssignedAt    time.Time  `ch:"assigned_at"`
}

```

Листинг В.2 – Объекты бригад

```

type Brigade struct {
    ID            int          `db:"id"`
    Status        int          `db:"status"`
    CreatedAt     time.Time    `db:"created_at"`
    UpdatedAt     time.Time    `db:"updated_at"`
}

type Member struct {
    BrigadeID     int          `db:"brigade_id"`
    InspectorID   int          `db:"inspector_id"`
    AssignedAt    time.Time    `db:"assigned_at"`
}

```

Листинг В.3 – Объекты файлов

```

type File struct {
    ID            int          `db:"id"`
    FileName      string       `db:"file_name"`
    FileSize      int64        `db:"file_size"`
    Bucket        string       `db:"bucket"`
    URL           string       `db:"url"`
}

```

Листинг В.4 – Объекты проверок

```

type Inspection struct {
    ID            int          `db:"id"`
    TaskID        int          `db:"task_id"`
    Status        int          `db:"status"`
    Type          *int         `db:"type"`
    Resolution    *int         `db:"resolution"`
    LimitReason   *string      `db:"limit_reason"`
    Method        *string      `db:"method"`
    MethodBy      *int         `db:"method_by"`
    ReasonType    *int         `db:"reason_type"`
    ReasonDescription *string      `db:"reason_description"`
    `db:"unauthorized_explanation"`
    InspectAt     *time.Time   `db:"inspect_at"`
}

```

```

        EnergyActionAt      *time.Time `db:"energy_action_at"`
        CreatedAt            time.Time  `db:"created_at"`
        UpdatedAt            time.Time  `db:"updated_at"`
    }

    type InspectedDevice struct {
        ID                int           `db:"id"`
        DeviceID          int           `db:"device_id"`
        InspectionID      int           `db:"inspection_id"`
        Value              decimal.Decimal `db:"value"`
        Consumption        decimal.Decimal `db:"consumption"`
        CreatedAt          time.Time     `db:"created_at"`
    }

    type InspectedSeal struct {
        ID                int           `db:"id"`
        SealID            int           `db:"seal_id"`
        InspectionID      int           `db:"inspection_id"`
        IsBroken          bool          `db:"is_broken"`
        CreatedAt          time.Time     `db:"created_at"`
    }

    type Attachment struct {
        ID                int           `db:"id"`
        InspectionID      int           `db:"inspection_id"`
        Type              int           `db:"type"`
        FileID            int           `db:"file_id"`
        CreatedAt          time.Time     `db:"created_at"`
    }

```

Листинг В.5 – Объекты абонентов

```

    type Subscriber struct {
        ID                int           `db:"id"`
        AccountNumber     string        `db:"account_number"`
        Surname           string        `db:"surname"`
        Name              string        `db:"name"`
        Patronymic        string        `db:"patronymic"`
        PhoneNumber       string        `db:"phone_number"`
        Email             string        `db:"email"`
        INN               string        `db:"inn"`
        BirthDate         time.Time     `db:"birth_date"`
        Status            int           `db:"status"`
        CreatedAt          time.Time     `db:"created_at"`
        UpdatedAt          time.Time     `db:"updated_at"`
    }

    type Passport struct {
        ID                int           `db:"id"`
        SubscriberID      int           `db:"subscriber_id"`
        Series            string        `db:"series"`
    }

```

```

    Number      string `db:"number"`
    IssuedBy     string `db:"issued_by"`
    IssueDate    string `db:"issue_date"`
}

type Object struct {
    ID           int          `db:"id"`
    Address      string        `db:"address"`
    HaveAutomaton bool         `db:"have_automaton"`
    CreatedAt    time.Time     `db:"created_at"`
    UpdatedAt    time.Time     `db:"updated_at"`
}

type Device struct {
    ID           int          `db:"id"`
    ObjectID     int          `db:"object_id"`
    Type         string       `db:"type"`
    Number       string       `db:"number"`
    PlaceType    int          `db:"place_type"`
    PlaceDescription string     `db:"place_description"`
    CreatedAt    time.Time     `db:"created_at"`
    UpdatedAt    time.Time     `db:"updated_at"`
}

type Seal struct {
    ID           int          `db:"id"`
    DeviceID     int          `db:"device_id"`
    Number       string       `db:"number"`
    Place        string       `db:"place"`
    CreatedAt    time.Time     `db:"created_at"`
    UpdatedAt    time.Time     `db:"updated_at"`
}

type Contract struct {
    ID           int          `db:"id"`
    Number       string       `db:"number"`
    SubscriberID int          `db:"subscriber_id"`
    ObjectID     int          `db:"object_id"`
    SignDate     time.Time     `db:"sign_date"`
    CreatedAt    time.Time     `db:"created_at"`
    UpdatedAt    time.Time     `db:"updated_at"`
}

```

Листинг В.6 – Объекты задач

```

type Status struct {
    ID     int    `db:"id"`
    Name   string `db:"name"`
}

type Task struct {

```

```

ID            int            `db:"id"`
BrigadeID     *int           `db:"brigade_id"`
ObjectID      int            `db:"object_id"`
PlanVisitAt   *time.Time    `db:"plan_visit_at"`
Status        int            `db:"status"`
Comment       *string        `db:"comment"`
StartedAt     *time.Time    `db:"started_at"`
FinishedAt    *time.Time    `db:"finished_at"`
CreatedAt     time.Time      `db:"created_at"`
UpdatedAt     time.Time      `db:"updated_at"`
}

```

Листинг В.7 – Объекты пользователей

```

type Role struct {
    ID int `db:"id"`
}

type User struct {
    ID            int            `db:"id"`
    RoleID        int            `db:"role_id"`
    Surname       string         `db:"surname"`
    Name          string         `db:"name"`
    Patronymic    *string        `db:"patronymic"`
    PhoneNumber   string         `db:"phone_number"`
    Email         string         `db:"email"`
    PasswordHash  *string        `db:"password_hash"`
    RefreshToken  *string        `db:"refresh_token"`
    RefreshTokenExpiredAfter *time.Time    `db:"refresh_token_expired_after"`
    CreatedAt     time.Time      `db:"created_at"`
    UpdatedAt     time.Time      `db:"updated_at"`
}

```

Приложение Г
(рекомендуемое)
SQL-скрипты

Листинг Г.1 – Скрипты создания таблиц

```

create table if not exists reports
(
    id            int primary key generated always as identity,
    type          int references report_types (id) on delete
restrict,
    period_start date          not null,
    period_end   date          not null,
    created_at   timestamptz not null default now()
);

create table if not exists attachments
(
    id            int primary key generated always as identity,
    report_id     int          not null references reports (id) on
delete cascade,
    file_id       int          not null,
    created_at    timestamptz not null default now()
);

create table if not exists brigades
(
    id            int primary key generated always as identity,
    status        int          not null references brigade_statuses
(id) on delete restrict,
    created_at    timestamptz not null default now(),
    updated_at    timestamptz not null default now()
);

-- Участники бригады
create table if not exists brigade_members
(
    brigade_id    int          not null references brigades (id) on
delete cascade,
    inspector_id  int          not null,
    assigned_at   timestamptz not null default now(),
    primary key   (brigade_id, inspector_id)
);

create table if not exists storage_files
(
    id            int primary key generated always as identity,
    file_name     text not null,
    file_size     int  not null,
    bucket        text not null,
    url           text not null -- Ссылка на скачивание
);

create table if not exists inspections
(

```

```

        id                                int primary key generated always as
identity,
        task_id                          int          not null unique,
        status                           int          not null references
inspection_statuses (id) on delete restrict,
        type                             int references inspection_types (id)
on delete restrict,
        resolution                       int references
inspection_resolutions (id) on delete restrict, -- Результаты
проверки
        limit_reason                    text,
-- Основание введения ограничения (приостановления) режима
потребления. Если NULL, то это неполная оплата
        method                          text,
-- Способ введения ограничения, приостановления, возобновления
режима потребления, номера и место установки пломб
        method_by                       int references inspection_methods_by
(id) on delete restrict, -- Кем было введено
        reason_type                     int references
inspection_reason_types (id) on delete restrict,
        reason_description               text,
-- Причина невведения ограничения (только для reason_type = 1)
        is_restriction_checked          bool,
-- Произведена проверка введенного ограничения
        is_violation_detected           bool,
-- Нарушение потребителем введенного ограничения выявлено
        is_expense_available            bool,
-- Наличие расхода после введенного ограничения
        violation_description            text,
-- Иное описание выявленного нарушения/сведения, на основании
которых сделан вывод о нарушении
        is_unauthorized_consumers       bool,
-- Самовольное подключение энергопринимающих устройств Потребителя
к электрическим сетям
        unauthorized_description         text,
-- Описание места и способа самовольного подключения к
электрическим сетям
        unauthorized_explanation          text,
-- Объяснение лица, допустившего самовольное подключение к
электрическим сетям
        inspect_at                      timestampz,
-- Дата проверки
        energy_action_at                timestampz,
-- Время действия над подачей электроэнергии
        created_at                      timestampz not null default now(),
        updated_at                     timestampz not null default now()
);

create table if not exists inspected_devices
(
        id                                int primary key generated always as identity,
        device_id                       int          not null,

```

```

        inspection_id int                not null references inspections
(id) on delete cascade,
        value          numeric(15, 2) not null, -- Текущее показание
        consumption    numeric(15, 2) not null, -- Расход электрической
энергии кВтч
        created_at      timestamptz     not null default now()
);

create table if not exists inspected_seals
(
        id              int primary key generated always as identity,
        seal_id         int              not null,
        inspection_id int                not null references inspections (id)
on delete cascade,
        is_broken       bool             not null, -- Сорвана ли пломба
        created_at      timestamptz not null default now()
);

create table if not exists attachments
(
        id              int primary key generated always as identity,
        inspection_id int                not null references inspections (id)
on delete cascade,
        type            int              not null references attachment_types
(id) on delete restrict,
        file_id         int              not null,
        created_at      timestamptz not null default now()
);

create table if not exists subscribers
(
        id              int primary key generated always as identity,
        account_number text              not null unique,
-- Лицевой счет
        surname         text             not null,
        name            text             not null,
        patronymic      text             not null,
        phone_number    text             not null check ( phone_number ~
'^(\+7|8)\d{10}$' ),
        email           text             not null check ( email ~ '^([A-Za-
z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,})$' ),
        inn             text             not null check ( inn ~
'^\d{10,12}$' ), -- ИНН
        birth_date      date             not null check ( '1900-01-01' <=
birth_date and birth_date < current_date ),
        status          int              not null default 1 references
subscriber_statuses (id) on delete restrict,
        created_at      timestamptz not null default now(),
        updated_at      timestamptz not null default now()
);

create table if not exists passports

```

```

(
    id                int primary key generated always as identity,
    subscriber_id int not null references subscribers (id) on
delete cascade,
    series            text not null check ( series ~ '^\\d{4}$' ), --
Серия
    number            text not null check ( number ~ '^\\d{6}$' ), --
Номер
    issued_by         text not null, --
Кем выдан
    issue_date        date not null, --
Когда выдан
    unique (series, number)
);

create table if not exists objects
(
    id                int primary key generated always as identity,
    address            text not null unique,
    have_automaton bool not null, -- Наличие
коммутационного (вводного) аппарата
    created_at         timestampz not null default now(),
    updated_at         timestampz not null default now()
);

create table if not exists devices
(
    id                int primary key generated always as
identity,
    object_id         int not null references objects (id)
on delete cascade,
    type              text not null,
    number             text not null unique,
    place_type        int not null references
device_place_types (id) on delete restrict,
    place_description text not null, -- Место установки
прибора учета
    created_at         timestampz not null default now(),
    updated_at         timestampz not null default now()
);

create table if not exists seals
(
    id                int primary key generated always as identity,
    device_id         int not null references devices (id) on
delete cascade,
    number            text not null unique,
    place             text not null, -- Место установки
    created_at         timestampz not null default now(),
    updated_at         timestampz not null default now()
);

```

```

create table if not exists contracts
(
    id                int primary key generated always as identity,
    number            text                not null unique,
    subscriber_id     int                not null references subscribers (id)
on delete restrict,
    object_id         int                not null references objects (id) on
delete restrict,
    sign_date         date               not null check ( sign_date <=
current_date ), -- Дата подписания
    created_at        timestamptz not null default now(),
    updated_at        timestamptz not null default now()
);

```

```

create table if not exists tasks
(
    id                int primary key generated always as identity,
    brigade_id        int,
    object_id         int                not null,
    plan_visit_at     timestamptz, -- Запланированное время визита
    status            int                not null references task_statuses
(id) on delete restrict,
    comment           text,
    started_at        timestamptz,
    finished_at       timestamptz,
    created_at        timestamptz not null default now(),
    updated_at        timestamptz not null default now(),
    check ( started_at is null or finished_at is null or
started_at <= finished_at )
);

```

```

create table if not exists users
(
    id                int primary key generated always
as identity,
    role_id           int                not null references
roles (id) on delete restrict,
    surname           text                not null,
    name              text                not null,
    patronymic        text,
    phone_number      text                not null unique check
( phone_number ~ '^(\\+7|8)\\d{10}$' ),
    email             text                not null unique check
( email ~ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\. [A-Za-z]{2,}$' ),
    password_hash     text,
    refresh_token      text,
    refresh_token_expired_after timestamptz,
    created_at        timestamptz not null default
now(),
    updated_at        timestamptz not null default now()
);

```

Листинг Г.2 – Скрипты вставки данных

```

insert into attachments (report_id, file_id)
values (:report_id, :file_id)
returning id, report_id, file_id, created_at;

insert into reports (type, period_start, period_end)
values (:type, :period_start, :period_end)
returning id, type, period_start, period_end, created_at;

insert into brigade_members (brigade_id, inspector_id)
values (:brigade_id, :inspector_id);

insert into storage_files (file_name, file_size, bucket, url)
values ($1, $2, $3, $4)
returning id, file_name, file_size, bucket, url;

insert into attachments (inspection_id, type, file_id)
values ($1, $2, $3)
returning id, inspection_id, type, file_id, created_at;

insert into inspected_devices (device_id, inspection_id, value,
consumption)
values (:device_id, :inspection_id, :value, :consumption);

insert into inspected_seals (seal_id, inspection_id, is_broken)
values (:seal_id, :inspection_id, :is_broken);

insert into inspections (task_id, status)
values ($1, 1)
returning id,
        task_id,
        status,
        type,
        resolution,
        limit_reason,
        method,
        method_by,
        reason_type,
        reason_description,
        is_restriction_checked,
        is_violation_detected,
        is_expense_available,
        violation_description,
        is_unauthorized_consumers,
        unauthorized_description,
        unauthorized_explanation,
        inspect_at,
        energy_action_at,
        created_at,
        updated_at;

```

```

insert into passports (subscriber_id, series, number, issued_by,
issue_date)
values (:subscriber_id, :series, :number, :issued_by, :issue_date)
returning id, subscriber_id, series, number, issued_by,
issue_date;

```

```

insert into subscribers (account_number, surname, name,
patronymic, phone_number, email, inn, birth_date, status)
values (:account_number, :surname, :name, :patronymic,
:phone_number, :email, :inn, :birth_date, :status)
returning id,
    account_number,
    surname,
    name,
    patronymic,
    phone_number,
    email,
    inn,
    birth_date,
    status,
    created_at,
    updated_at;

```

```

insert into devices (object_id, type, number, place_type,
place_description)
values (:object_id, :type, :number, :place_type,
:place_description);

```

```

insert into objects (address, have_automaton)
values (:address, :have_automaton)
returning id, address, have_automaton, created_at, updated_at;

```

```

insert into seals (device_id, number, place)
values (:device_id, :number, :place);

```

```

insert
into devices (object_id, type, number, place_type,
place_description)
select o.id, :type, :number, :place_type, :place_description
from objects o
where address = :object_address
on conflict (number) do update
    set type = excluded.type,
        place_type = excluded.place_type,
        place_description = excluded.place_description;

```

```

insert into objects (address, have_automaton)
values (:address, :have_automaton)
on conflict (address) do update
    set have_automaton = excluded.have_automaton;

```

```

insert

```

```

into seals (device_id, number, place)
select d.id, :number, :place
from devices d
where d.number = :device_number
on conflict (number) do update
    set place = excluded.place;

insert into contracts (number, subscriber_id, object_id,
sign_date)
values (:number, :subscriber_id, :object_id, :sign_date)
returning id, number, subscriber_id, object_id, sign_date,
created_at, updated_at;

insert into tasks (brigade_id, object_id, plan_visit_at, status,
comment)
values (:brigade_id, :object_id, :plan_visit_at, 1, :comment)
returning id,
    brigade_id,
    object_id,
    plan_visit_at,
    status,
    comment,
    started_at,
    finished_at,
    created_at,
    updated_at;

```

Листинг Г.3 – Скрипты получения данных

```

select id, type, period_start, period_end, created_at
from reports;

select id, report_id, file_id, created_at
from attachments
where report_id = any ($1);

select id, status, created_at, updated_at
from brigades;

select brigade_id, inspector_id, assigned_at
from brigade_members;

select id, file_name, file_size, bucket, url
from storage_files
where id = $1;

select id,
    task_id,
    status,
    type,
    resolution,

```

```

        limit_reason,
        method,
        method_by,
        reason_type,
        reason_description,
        is_restriction_checked,
        is_violation_detected,
        is_expense_available,
        violation_description,
        is_unauthorized_consumers,
        unauthorized_description,
        unauthorized_explanation,
        inspect_at,
        energy_action_at,
        created_at,
        updated_at
    from inspections;

```

```

select distinct on (subscriber_id) id, subscriber_id, series,
number, issued_by, issue_date
from passports
order by subscriber_id, issue_date desc;

```

```

select id,
        account_number,
        surname,
        name,
        patronymic,
        phone_number,
        email,
        inn,
        birth_date,
        status,
        created_at,
        updated_at
    from subscribers;

```

```

select id, subscriber_id, series, number, issued_by, issue_date
from passports
where subscriber_id = $1
order by issue_date
limit 1;

```

```

select id,
        object_id,
        type,
        number,
        place_type,
        place_description,
        created_at,
        updated_at
    from devices;

```

```
select id, address, have_automaton, created_at, updated_at
from objects;
```

```
select id, device_id, number, place, created_at, updated_at
from seals;
```

```
select o.id, o.address, o.have_automaton, o.created_at,
o.updated_at
from objects o
      join devices d on o.id = d.object_id
where d.id = $1;
```

```
select o.id, o.address, o.have_automaton, o.created_at,
o.updated_at
from objects o
      join devices d on o.id = d.object_id
      join seals s on d.id = s.device_id
where s.id = $1;
```

```
select s.id, s.device_id, s.number, s.place, s.created_at,
s.updated_at
from seals s
      join devices d on d.id = s.device_id
where d.object_id = $1;
```

```
select id, number, subscriber_id, object_id, sign_date,
created_at, updated_at
from contracts;
```

```
select id, number, subscriber_id, object_id, sign_date,
created_at, updated_at
from contracts
where object_id = $1
order by sign_date
limit 1;
```

```
select id,
       brigade_id,
       object_id,
       plan_visit_at,
       status,
       comment,
       started_at,
       finished_at,
       created_at,
       updated_at
from tasks;
```

Приложение Д
(рекомендуемое)
Структуры слоя представления данных

Листинг Д.1 – Слой представления analytics-service

```
func (s *ServerBuilder) AddReports(service *analytics.Service) {
    r := s.router.SubRouter("/reports")
    r.HandlePost("/basic/{periodStart}/{periodEnd}",
handler.CreateBasicReport(service))
    r.HandleGet("", handler.GetAllReports(service))
}
```

Листинг Д.2 – Слой представления brigade-service

```
func (s *ServerBuilder) AddBrigades(service *brigade.Service) {
    r := s.router.SubRouter("/brigades")
    r.HandlePost("", handler.CreateBrigade(service))
    r.HandleGet("/{id}", handler.GetBrigadeByID(service))
    r.HandleGet("", handler.GetAllBrigades(service))
}
```

Листинг Д.3 – Слой представления file-service

```
func (s *ServerBuilder) AddFiles(service *file.Service) {
    r := s.router.SubRouter("/files")
    r.HandlePost("", handler.UploadFile(service))
    r.HandleGet("/{id}", handler.GetFileByID(service))
    r.HandleGet("", handler.GetFilesByIDs(service))
}
```

Листинг Д.4 – Слой представления inspection-service

```
func (s *ServerBuilder) AddInspections(service
*inspection.Service) {
    r := s.router.SubRouter("/inspections")
    r.HandleGet("", handler.GetAllInspections(service))
    r.HandleGet("/task/{taskId}",
handler.GetInspectionByTaskID(service))
    r.HandlePost("/{id}/photo",
handler.AttachPhotoToInspection(service))
    r.HandlePatch("/{id}/finish",
handler.FinishInspection(service))
}
```

Листинг Д.5 – Слой представления subscriber-service

```
func (s *ServerBuilder) AddSubscribers(service
*subscriber.Service) {
    r := s.router.SubRouter("/subscribers")
    r.HandlePost("", handler.AddSubscriber(service))
    r.HandleGet("/{id}", handler.GetSubscriberByID(service))
    r.HandleGet("", handler.GetAllSubscribers(service))
}
```

```

func (s *ServerBuilder) AddObjects(service *object.Service) {
    r := s.router.SubRouter("/objects")
    r.HandlePost("", handler.AddObject(service))
    r.HandleGet("/{id}", handler.GetObjectByID(service))
    r.HandleGet("/devices/{deviceID}",
handler.GetObjectByDeviceID(service))
    r.HandleGet("/seals/{sealID}",
handler.GetObjectBySealID(service))
    r.HandleGet("", handler.GetAllObjects(service))
}

func (s *ServerBuilder) AddContracts(service *contract.Service) {
    r := s.router.SubRouter("/contracts")
    r.HandlePost("", handler.AddContract(service))
    r.HandleGet("", handler.GetAllContracts(service))
    r.HandleGet("/objects/{objectID}/last",
handler.GetLastContractByObjectID(service))
}

func (s *ServerBuilder) AddRegistry(service *registry.Service) {
    r := s.router.SubRouter("/registry")
    r.HandlePost("/parse", handler.ParseRegistry(service))
}

```

Листинг Д.6 – Слой представления task-service

```

func (s *ServerBuilder) AddTasks(service *task.Service) {
    r := s.router.SubRouter("/tasks")
    r.HandlePost("", handler.AddTask(service))
    r.HandleGet("/{id}", handler.GetTaskByID(service))
    r.HandleGet("/brigade/{brigadeID}",
handler.GetTasksByBrigade(service))
    r.HandleGet("", handler.GetAllTasks(service))
    r.HandlePatch("/{id}/start", handler.StartTask(service))
    r.HandlePatch("/assign",
handler.AssignTaskToBrigade(service))
}

```

Приложение Е
(рекомендуемое)
Структуры слоя бизнес-логики

Листинг Е.1 – Слой бизнес-логики analytics-service

```

type Service struct {
    repository      Repository
    inspectionService InspectionService
    brigadeService  BrigadeService
    subscriberService SubscriberService
    fileService     FileService
    templates       config.Templates
}

func NewService(repository Repository, inspectionService
InspectionService, brigadeService BrigadeService,
    subscriberService SubscriberService, fileService FileService,
templates config.Templates) *Service {
    return &Service{
        repository:      repository,
        inspectionService: inspectionService,
        brigadeService:   brigadeService,
        subscriberService: subscriberService,
        fileService:      fileService,
        templates:       templates,
    }
}

type Service struct {
    scheduler      gocron.Scheduler
    settings       config.Cron
    analyticsService AnalyticsService
    running        *atomic.Bool
}

func NewService(settings config.Cron, analyticsService
AnalyticsService) *Service {
    return &Service{
        settings:      settings,
        analyticsService: analyticsService,
        running:       &atomic.Bool{},
    }
}

```

Листинг Е.2 – Слой бизнес-логики brigade-service

```

type Service struct {
    repository      Repository
    userService     UserService
}

func NewService(repository Repository, userService UserService)
*Service {
    return &Service{

```

```

        repository: repository,
        userService: userService,
    }
}

```

Листинг Е.3 – Слой бизнес-логики file-service

```

type Service struct {
    bucketStorages map[Bucket]Storage
    repository      Repository
}

func NewService(bucketStorages map[Bucket]Storage, repository
Repository) *Service {
    return &Service{
        bucketStorages: bucketStorages,
        repository:      repository,
    }
}

```

Листинг Е.4 – Слой бизнес-логики inspection-service

```

type Service struct {
    repository      Repository
    publisher        *Publisher
    analyzerService AnalyzerService
    subscriberService SubscriberService
    fileService      FileService
    taskService      TaskService
    brigadeService   BrigadeService
    templates        config.Templates
}

func NewService(repository Repository, publisher *Publisher,
analyzerService AnalyzerService, subscriberService
SubscriberService, fileService FileService,
taskService TaskService, brigadeService BrigadeService,
templates config.Templates) *Service {
    return &Service{
        repository:      repository,
        publisher:        publisher,
        analyzerService: analyzerService,
        subscriberService: subscriberService,
        fileService:      fileService,
        taskService:      taskService,
        brigadeService:   brigadeService,
        templates:        templates,
    }
}

```

Листинг Е.5 – Слой бизнес-логики subscriber-service

```

type Service struct {
    repository Repository
}

func NewService(repository Repository) *Service {
    return &Service{
        repository: repository,
    }
}

type Service struct {
    subscriberRepository SubscriberRepository
    objectRepository      ObjectRepository
    contractRepository     ContractRepository
}

func NewService(subscriberRepository SubscriberRepository,
objectRepository ObjectRepository, contractRepository
ContractRepository) *Service {
    return &Service{
        subscriberRepository: subscriberRepository,
        objectRepository:      objectRepository,
        contractRepository:     contractRepository,
    }
}

type Service struct {
    repository Repository
}

func NewService(repository Repository) *Service {
    return &Service{
        repository: repository,
    }
}

type Service struct {
    repository      Repository
    subscriberRepository SubscriberRepository
    taskService     TaskService
}

func NewService(repository Repository, subscriberRepository
SubscriberRepository, taskService TaskService) *Service {
    return &Service{
        repository:      repository,
        subscriberRepository: subscriberRepository,
        taskService:     taskService,
    }
}

```

Листинг Е.6 – Слой бизнес-логики task-service

```
type Service struct {
    repository Repository
    publisher  *Publisher
}

func NewService(repository Repository, publisher *Publisher)
*Service {
    return &Service{
        repository: repository,
        publisher:  publisher,
    }
}
```

Приложение Ж
(рекомендуемое)
Структуры слоя доступа к данным

Листинг Ж.1 – Слой доступа к данным analytics-service

```

type Repository struct {
    postgres *sqlx.DB
    clickhouse driver.Conn
}

func NewRepository(postgres *sqlx.DB, clickhouse driver.Conn)
*Repository {
    return &Repository{
        postgres:  postgres,
        clickhouse: clickhouse,
    }
}

```

Листинг Ж.2 – Слой доступа к данным brigade-service

```

type Repository struct {
    db *sqlx.DB
}

func NewRepository(db *sqlx.DB) *Repository {
    return &Repository{
        db: db,
    }
}

```

Листинг Ж.3 – Слой доступа к данным file-service

```

type Repository struct {
    db *sqlx.DB
}

func NewRepository(db *sqlx.DB) *Repository {
    return &Repository{
        db: db,
    }
}

type Minio struct {
    mc *minio.Client
    bucket, host string
}

func NewMinio(ctx context.Context, mc *minio.Client, bucket, host
string) (*Minio, error) {
    err := db.PrepareBucket(ctx, mc, bucket)
    if err != nil {
        return &Minio{}, fmt.Errorf("could not prepare bucket:
%w", err)
    }
}

```

```

    return &Minio{
        mc:      mc,
        bucket:  bucket,
        host:    host,
    }, nil
}

```

Листинг Ж.4 – Слой доступа к данным inspection-service

```

type Repository struct {
    db *sqlx.DB
}

func NewRepository(db *sqlx.DB) *Repository {
    return &Repository{
        db: db,
    }
}

```

Листинг Ж.5 – Слой доступа к данным subscriber-service

```

type Repository struct {
    db *sqlx.DB
}

func NewRepository(db *sqlx.DB) *Repository {
    return &Repository{
        db: db,
    }
}

type Repository struct {
    db *sqlx.DB
    subscriberRepository SubscriberRepository
    objectRepository      ObjectRepository
}

func NewRepository(db *sqlx.DB, subscriberRepository
SubscriberRepository, objectRepository ObjectRepository)
*Repository {
    return &Repository{
        db: db,
        subscriberRepository: subscriberRepository,
        objectRepository:      objectRepository,
    }
}

```

Листинг Ж.6 – Слой доступа к данным task-service

```
type Repository struct {  
    db *sqlx.DB  
}  
  
func NewRepository(db *sqlx.DB) *Repository {  
    return &Repository{  
        db: db,  
    }  
}
```

Приложение И
(рекомендуемое)
Листинг модульных тестов

```

func TestCreateBrigade_Success(t *testing.T) {
    svc := NewService(
        &mockRepository{brigade: Brigade{ID: 1, Status:
StatusIdle}},
        &mockUserService{users: []user.User{
            {ID: 10, Role: user.RoleInspector, Surname:
"Иванов", Name: "Иван"},
            {ID: 20, Role: user.RoleInspector, Surname:
"Петров", Name: "Пётр"},
        }},
    )

    b, err := svc.CreateBrigade(gotctx.Wrap(context.Background()),
CreateBrigadeRequest{
    InspectorIDs: []int{10, 20},
})
    if err != nil {
        t.Fatalf("unexpected error: %v", err)
    }
    if b.ID != 1 {
        t.Fatalf("expected brigade ID = 1, got %d", b.ID)
    }
    if len(b.Inspectors) != 2 {
        t.Fatalf("expected 2 inspectors, got %d",
len(b.Inspectors))
    }
}

func TestCreateBrigade_UserServiceError(t *testing.T) {
    svc := NewService(
        &mockRepository{},
        &mockUserService{err: errors.New("user service
unavailable")},
    )

    _, err := svc.CreateBrigade(gotctx.Wrap(context.Background()),
CreateBrigadeRequest{
    InspectorIDs: []int{10},
})
    if err == nil {
        t.Fatal("expected error, got nil")
    }
}

func TestCreateBrigade_UserCountMismatch(t *testing.T) {
    svc := NewService(
        &mockRepository{},
        &mockUserService{users: []user.User{
            {ID: 10, Role: user.RoleInspector},
        }},
    )

```

```

_, err := svc.CreateBrigade(gotctx.Wrap(context.Background()),
CreateBrigadeRequest{
    InspectorIDs: []int{10, 20},
})
if err == nil {
    t.Fatal("expected error, got nil")
}
}

func TestCreateBrigade_UserNotInspector(t *testing.T) {
    svc := NewService(
        &mockRepository{},
        &mockUserService{users: []user.User{
            {ID: 10, Role: user.RoleInspector},
            {ID: 20, Role: user.RoleDispatcher},
        }},
    )

    _, err := svc.CreateBrigade(gotctx.Wrap(context.Background()),
CreateBrigadeRequest{
    InspectorIDs: []int{10, 20},
})
if err == nil {
    t.Fatal("expected error, got nil")
}
}

func TestCreateBrigade_RepositoryError(t *testing.T) {
    svc := NewService(
        &mockRepository{err: errors.New("database error")},
        &mockUserService{users: []user.User{
            {ID: 10, Role: user.RoleInspector},
        }},
    )

    _, err := svc.CreateBrigade(gotctx.Wrap(context.Background()),
CreateBrigadeRequest{
    InspectorIDs: []int{10},
})
if err == nil {
    t.Fatal("expected error, got nil")
}
}

```