**3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИС**

**3.1 Выбор программных средств реализации**

Разработка информационной системы требует тщательного подбора технологического стека, обеспечивающего высокую производительность, надежность и масштабируемость решения. Основной упор сделан на современные, проверенные технологии с открытым исходным кодом, обладающие широким сообществом разработчиков и хорошей документацией.

Для реализации серверной части информационной системы был выбран язык программирования Java. Этот выбор обусловлен рядом ключевых преимуществ, делающих Java оптимальным решением для разработки сложных enterprise-приложений. Java представляет собой зрелый, строго типизированный объектно-ориентированный язык с мощной экосистемой и проверенной временем архитектурой.

Одним из главных достоинств Java является ее кроссплатформенность. Благодаря виртуальной машине JVM (Java Virtual Machine), Java-приложения могут выполняться на любом оборудовании и операционной системе, где установлена соответствующая JVM. Это особенно важно для информационных систем, которые могут развертываться в различных инфраструктурных средах.

Безопасность Java обеспечивается несколькими механизмами. Строгая типизация предотвращает множество распространенных ошибок программирования. Управление памятью через сборщик мусора (Garbage Collector) минимизирует риски утечек памяти. Встроенные механизмы безопасности, такие как Security Manager и криптографические библиотеки, позволяют реализовывать надежные системы аутентификации и авторизации.

Для сравнения рассматривались альтернативные технологии. Python, несмотря на простоту синтаксиса, уступает в производительности из-за интерпретируемой природы. C# обладает схожими характеристиками, но сильнее привязан к экосистеме Microsoft. Go демонстрирует отличную производительность, но имеет менее развитую экосистему для enterprise-разработки.

Для разработки клиентской части информационной системы был выбран прогрессивный JavaScript-фреймворк Vue.js. Этот выбор обусловлен оптимальным сочетанием простоты интеграции, гибкости архитектуры и высокой производительности, что делает Vue.js идеальным решением для современных веб-приложений. Vue представляет собой реактивный фреймворк, специально разработанный для поэтапного внедрения и создания сложных пользовательских интерфейсов.

Vue.js отличается продуманной и интуитивно понятной архитектурой, основанной на компонентном подходе. Компоненты Vue инкапсулируют логику, шаблоны и стили, что способствует созданию поддерживаемого и переиспользуемого кода. Реактивная система данных автоматически отслеживает зависимости и эффективно обновляет DOM при изменениях состояния приложения, значительно упрощая разработку динамических интерфейсов.

Для сравнения рассматривались альтернативные фреймворки. React, хотя и обладает большим сообществом, требует дополнительных библиотек для полноценной разработки. Angular, несмотря на комплексность, имеет более высокий порог входа и менее гибкую архитектуру. Vue.js занял золотую середину, предлагая богатые возможности при относительно простом освоении.

Для хранения и обработки данных в информационной системе была выбрана реляционная СУБД PostgreSQL. Этот выбор обусловлен сочетанием надежности, производительности и расширенной функциональности, что делает PostgreSQL оптимальным решением для сложных информационных систем. PostgreSQL представляет собой объектно-реляционную систему управления базами данных с открытым исходным кодом, разрабатываемую более 30 лет.

Для сравнения рассматривались альтернативные СУБД. MySQL, хотя и проще в настройке, уступает в функциональности и надежности для сложных систем. MongoDB, как представитель NoSQL, не обеспечивает необходимого уровня целостности данных для транзакционных систем. Oracle Database обладает схожими возможностями, но является проприетарным и дорогостоящим решением.

Выбор PostgreSQL в качестве СУБД для информационной системы обеспечивает надежное хранение данных, высокую производительность запросов и гибкость при разработке. Расширенные функции, такие как оконные операции и поддержка JSON, позволяют эффективно решать сложные задачи обработки данных. Открытая лицензия и активное сообщество разработчиков гарантируют долгосрочную поддержку и развитие системы.

Выбранный технологический стек обеспечивает модульность архитектуры, что позволяет масштабировать систему и добавлять новый функционал. Сочетание Java и Vue.js создает оптимальный баланс между производительностью серверной части и отзывчивостью клиентского интерфейса. PostgreSQL гарантирует надежное хранение данных и возможность сложного анализа информации. Такой набор технологий полностью соответствует требованиям к современным информационным системам.

**3.2 Структура программного обеспечения**

**3.3 Разработка программного кода**

**3.3.1 Разработка серверной части кода**

Разработка back-end части для информационной системы является ключевым этапом в создании эффективного и надежного программного продукта. Серверная часть обеспечивает обработку данных, взаимодействие с базами данных и реализацию бизнес-логики, что позволяет системе выполнять свои функции на высоком уровне.

Первым шагом создания серверной части, является создание сущностей (моделей). Сущности представляют данные, которые сохраняются в базе данных. Каждая сущность соответствует таблице в реляционной базе данных, а поля сущности соответствуют столбцам таблицы. Сущности аннотируются с помощью @Entity, и для них обычно создаются классы с полями и методами для работы с данными. Они определяют, какие данные должны храниться, какие операции могут быть выполнены с этими данными, и как они соотносятся друг с другом. Они являются частью архитектурного паттерна MVC (Model-View-Controller), где отвечают за работу с информацией: валидацию, хранение, обработку и передачу между слоями приложения. Сущности абстрагируют взаимодействие с базой данных, предоставляя удобный интерфейс для операций.

В Java каждая модель является отдельным классом, в котором описаны поля, которые являются представлением полей в базе данных. Для более удобного создания моделей используется библиотека Lombok. Она позволяет избегать написания большого количества шаблонного кода, автоматически генерируя необходимые методы, такие как геттеры, сеттеры и конструкторы. Например, при создании модели Document, которая представляет собой таможенный документ, можно использовать аннотации @Data, @NoArgsConstructor и @AllArgsConstructor. Аннотация @Data автоматически генерирует методы для доступа к полям класса, а также методы equals(), hashCode() и toString(). Аннотации @NoArgsConstructor и @AllArgsConstructor создают конструкторы без параметров и с параметрами соответственно. Это позволяет легко инициализировать объекты модели.

Исходя из классов моделей, создаются классы DTO. DTO, или Data Transfer Object, представляет собой паттерн проектирования, используемый для передачи данных между сервисами и компонентами. Основная цель DTO заключается в том, чтобы упрощать и оптимизировать процесс передачи данных, минимизируя количество запросов и объем передаваемой информации. Оно позволяет передавать только те данные, которые необходимы, исключая лишнюю информацию. Вся получаемая или отправляемая информация в серверной части будет передаваться в виде DAO классов.

Для конвертации сущностей в DAO классы и обратно используются мапперы. Библиотека MapStruct генерирует реализацию мапперов на этапе компиляции, что обеспечивает высокую производительность и минимизирует вероятность ошибок, связанных с ручным написанием маппинга. Для этого необходимо только создать интерфейс и определить классы для конвертации.

Далее необходимо создать механизм взаимодействия с базой данных с помощью моделей. Для этого используется Java Persistence API (JPA). Это спецификация Java, предназначенная для управления персистентностью данных в Java-приложениях. JPA предоставляет стандартный способ работы с реляционными базами данных, позволяя разработчикам легко сохранять, извлекать, обновлять и удалять данные, представленные в виде объектов. С помощью неё возможно создание репозитория, который автоматически реализует все основные операции с базой данных. Для реализации более сложных запрос используется HQL. Он позволяет писать запросы, которые оперируют объектами и их свойствами, а не таблицами и столбцами реляционной базы данных. Например, вот этот запрос выдает список активных проверок ФЛК у которых совпадает тип документа:

@Query("SELECT f FROM FlkCheck f where f.documentType = :documentType and f.isActive = true")

List<FlkCheck> findByDocumentTypeAndActiveIsTrue(int documentType);