Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Кафедра информатики

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А. С. Кузнецов
подпись
«___» _____ 2023 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.04 – Программная инженерия

Интеграционная межведомственная система учета реабилитационных и абилитационных мероприятий

Руководитель _____ ст. преп. А. К. Погребников
подпись, дата

Выпускник _____ А. Д. Непомнящий
подпись, дата

Нормоконтролер _____ Н. Б. Позолотина
подпись, дата

Красноярск 2023

РЕФЕРАТ

Выпускная квалификационная работа по теме «Интеграционная межведомственная система учета реабилитационных и абилитационных мероприятий» содержит 54 страницы текстового документа, 22 иллюстрации, 1 таблицу, 23 использованных источника.

ИНТЕГРАЦИОННАЯ СИСТЕМА, ИНФОРМАЦИОННАЯ СИСТЕМА, ВЕБ-ПРИЛОЖЕНИЕ, SPRING FRAMEWORK, ANGULAR.

Цель работы: развитие межведомственной информационной системы «Учет реабилитационных и абилитационных мероприятий» и рабочей документации к ней.

Задачи:

- изучение требований к системе;
- обзор схожих программных решений;
- изучение и выбор средств разработки;
- изучение архитектуры системы;
- доработка системы;
- тестирование системы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Гарантирование защиты и помощи лицам с инвалидностью, обеспечение их полноценного развития и интеграции в общество является конституционной обязанностью государства. Важнейшими средствами реализации этой гарантии являются осуществление ранней помощи детям-инвалидам и проведение реабилитационных и абилитационных мероприятий.

Такая поддержка имеет целью обеспечить инвалидов возможностью для развития, восстановления и улучшения их физических, психологических и социальных возможностей. Ранняя помощь (РП) направлена на диагностику и медицинское вмешательство в ранние стадии жизни ребенка с целью предотвращения или смягчения последствий возможных заболеваний или инвалидности, она также подразумевает сотрудничество медицинского персонала с родителями с целью лучшего понимания ими нужд ребенка. Реабилитационные и абилитационные мероприятия, в свою очередь, включают более широкий спектр программ и услуг для непосредственной интеграции в общество, например, профессиональную ориентацию, обучение профессиональным навыкам и иное способствование трудоустройству, психологическую поддержку, обеспечение техническими средствами реабилитации (ТСР).

Осуществление социальной поддержки лиц с инвалидностью, как и обеспечение любых государственных услуг, требует большого объема административной работы. В частности – учет лиц, нуждающихся в услуге, и непосредственных организаций, оказывающих ее, формирование отчетности о результатах такой деятельности, а также обмен соответствующими данными между различными государственными органами. В последнее десятилетие происходит широкое внедрение информационных систем, автоматизирующих такую работу для разных сфер государственных услуг. Это создает благоприятную среду для дальнейшей цифровизации и интеграции этих систем с новыми, и, таким образом, достижения еще большего положительного эффекта

автоматизации.

Красноярскому краю потребовалось создание и внедрение единой информационной межведомственной системы (ЕИМС), организующей межведомственное взаимодействие между органами исполнительной власти и организациями, участвующими в предоставлении помощи инвалидам. Система должна содержать сведения об инвалидах, оказанных им реабилитационных и абилитационных мероприятиях, реабилитационных организациях, формировать сводную отчетность на основании этих сведений [1].

Результатом внедрения такой системы должно стать улучшение работы краевых государственных структур по обеспечению социальной поддержки лиц с инвалидностью. Непосредственно выражено оно будет в увеличении в крае доли получивших раннюю помощь, реабилитацию и абилитацию инвалидов в общем числе инвалидов, имеющих такие рекомендации.

Система была разработана и внедрена, после чего модифицирована по новым требованиям заказчика.

Требования к информационной системе были сформулированы в двух технических заданиях – на разработку и на развитие государственной межведомственной информационной системы Красноярского края «Учет реабилитационных и абилитационных мероприятий» (ГМИС «УРАМ»).

Целью настоящей выпускной квалификационной работы являлось развитие, в соответствии с техническими заданиями, ГМИС «УРАМ» и рабочей документации к ней.

Для достижения поставленной цели были выполнены следующие задачи:

- изучение требований к системе;
- обзор схожих программных решений;
- изучение и выбор средств разработки;
- изучение архитектуры системы;
- доработка системы;
- тестирование системы.

1 Предметная область и требования к интеграционной системе

ГМИС «УРАМ» предназначена для обеспечения электронного межведомственного взаимодействия Министерства социальной политики, Министерства образования, Министерства здравоохранения, Министерства спорта, Агентства труда и занятости населения Красноярского края, организаций, участвующих в предоставлении реабилитационных и абилитационных мероприятий инвалидам. Система должна осуществлять:

- сбор, обработку и анализ информации о данных лицах, содержащей в том числе сведения об оказываемых им реабилитационных и абилитационных мероприятиях;
- формирование сводной отчетности о реализации министерствами и ведомствами индивидуальных программ реабилитации и абилитации (ИПРА) инвалидов, детей-инвалидов.

Заказчик (Министерство социальной политики Красноярского края), предоставил техническое задание, оформленное в соответствии с ГОСТ 34.602–89.

В ходе анализа предметной области и требований к системе было изучено данное ТЗ, а также выполнен обзор существующих схожих программных решений.

1.1 Анализ требований к системе

1.1.1 Характеристика объектов автоматизации

Объектом автоматизации является деятельность Министерства социальной политики Красноярского края, Министерства образования Красноярского края, Министерства здравоохранения Красноярского края, Министерства спорта Красноярского края, Агентства труда и занятости населения Красноярского края.

В Красноярском крае на 01.01.2020 на учете состояло 180,4 тысяч инвалидов, что составляет 6,2 % от численности населения края, из них 12,2 тысяч – дети-инвалиды, 14,8 тысяч – инвалиды молодого возраста от 18 до 35 лет.

Для оценки эффективности оказания помощи лицам с инвалидностью сотрудниками Министерства социальной политики края выполняется расчет ряда показателей на основании данных из отчетов, предоставляемых органами исполнительной власти края, являющихся ответственными исполнителями ИПРА и РП.

Все органы исполнительной власти Красноярского края выгружают выписки из ИПРА федеральной базы данных «Витрина» с отображением только тех мероприятий, исполнителем которых они являются.

В ИПРА могут быть рекомендованы одновременно мероприятия, ответственными за которые являются разные органы исполнительной власти. Таким образом, информация, выгруженная из федеральной базы данных «Витрина», повторяется. В связи с тем, что в крае отсутствует инструмент для формирования сводной отчетности о реализации ИПРА и РП, результат расчета показателей эффективности мероприятий является недостоверным.

Автоматизация сбора данных и формирования отчетности должна решить данную проблему.

1.1.2 Требования к структуре системы

Система должна состоять из следующий подсистем:

а) подсистема авторизации, обеспечивающая авторизацию пользователей;

б) подсистема администрирования, обеспечивающая управление правами пользователей и записывающая протоколы импорта данных в лог;

в) подсистема справочной информации, состоящая из следующих модулей:

1) справочник организаций, обеспечивающий хранение, создание, редактирование и удаление записей об организациях, зарегистрированных в системе;
2) справочник ТСР, обеспечивающий хранение, создание, редактирование и удаление записей о технических средствах реабилитации;
3) справочник мероприятий, обеспечивающий хранение, создание, редактирование и удаление записей о мероприятиях по ИПРА;

4) справочник умерших граждан, обеспечивающая хранение записей об умерших гражданах;

г) подсистема импорта данных, позволяющая загружать данные из выписок ИПРА, о фактах исполнения ИПРА, о ранней помощи;

д) подсистема ввода информации о работе пунктов проката, обеспечивающая хранение, создание, редактирование и удаление записей о предоставлении ТСР инвалидам;

е) подсистема отчетов, обеспечивающая формирование необходимых отчетов с показателями эффективности работы по оказанию помощи инвалидам;

ж) подсистема обмена со сторонними ИС, реализующая точку доступа для приема POST-запросов от сторонних ИС и дальнейшего сохранения полученных данных об ИПРА и умерших гражданах.

1.1.3 Функциональные требования

Каждый зарегистрированный в системе пользователь обладает логином, паролем, фамилией, именем и отчеством, относится к одной организации и относится к одной из ролей.

Система имеет три следующие роли пользователей:

- сотрудник Министерства социальной политики (далее – сотрудник министерства);
- сотрудник иного министерства или ведомства (далее – сотрудник ведомства);
- администратор.

Подсистема авторизации дает возможность аутентификации с использование логина и пароля с последующей выдачей пользователю прав в системе, соответствующих его роли.

Возможные действия пользователей были обозначены на диаграмме прецедентов. Для наглядности на диаграмме пользовательские действия были сгруппированы по подсистемам, их обеспечивающим. Диаграмма разбита на две части и приведена на рисунках 1 и 2.

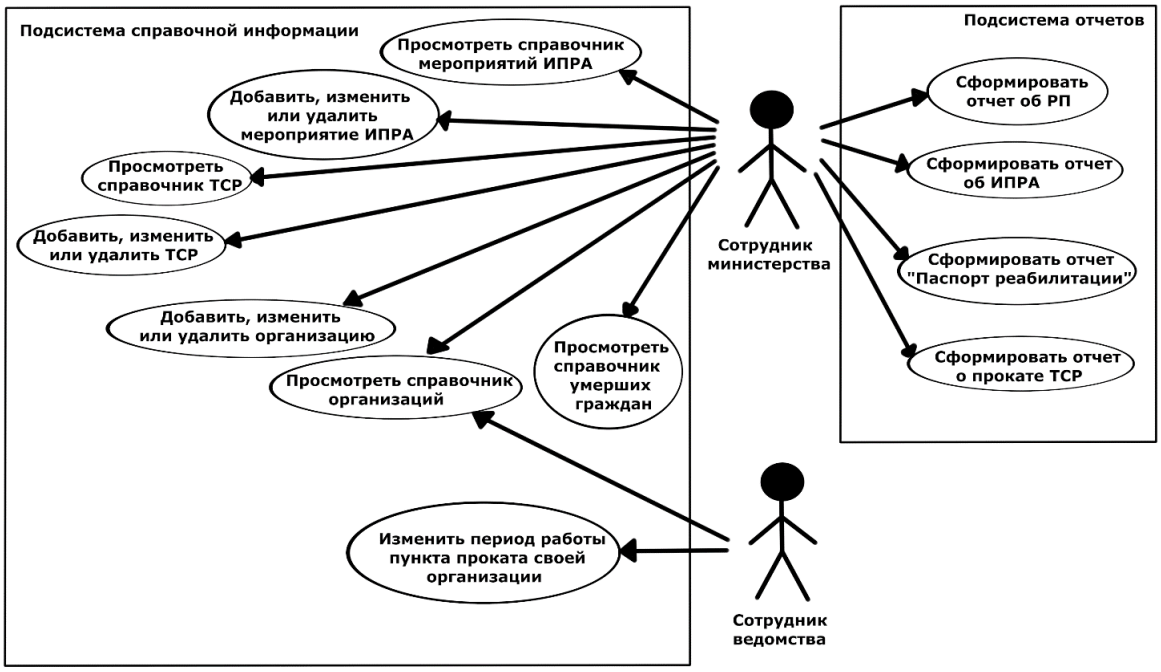


Рисунок 1 – Часть диаграммы прецедентов для подсистем справочной информации и отчетов

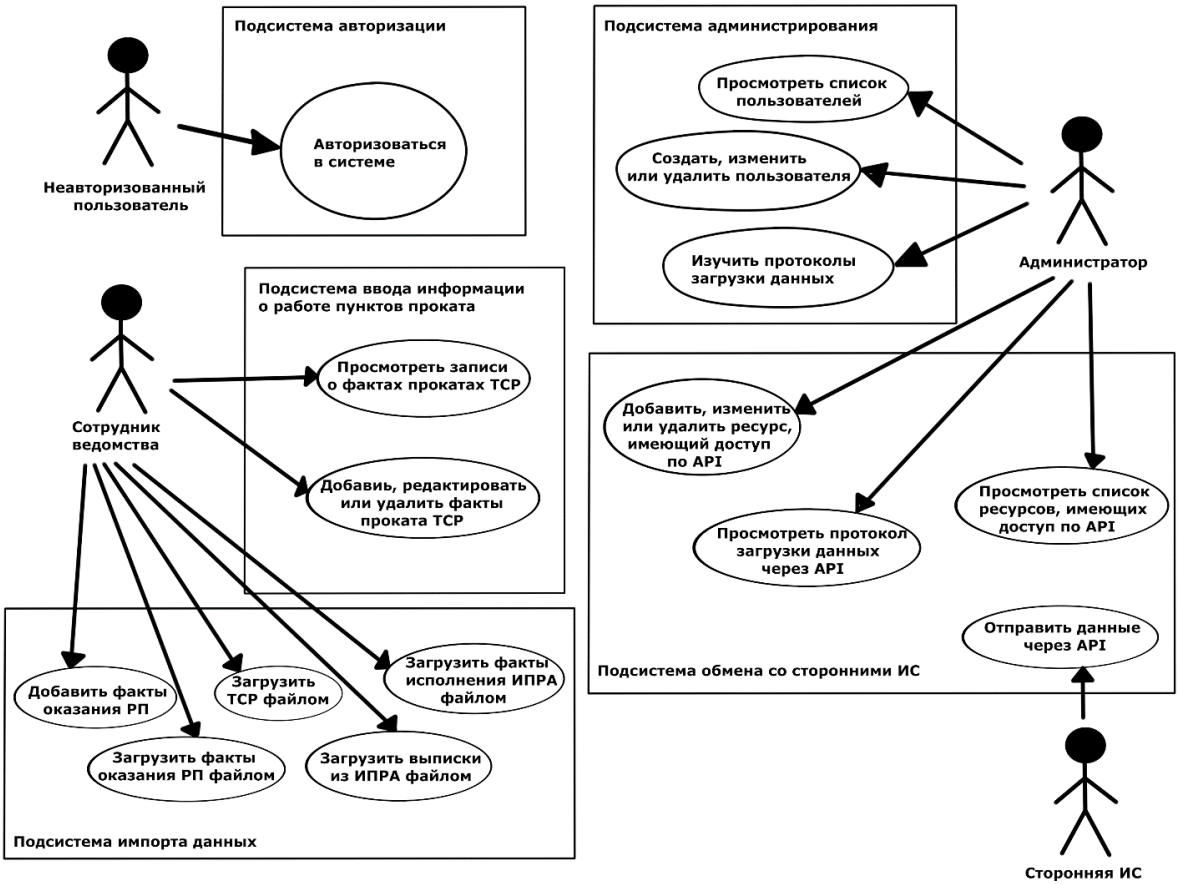


Рисунок 2 – Часть диаграммы прецедентов для остальных подсистем

Далее перечислены ключевые функциональных требований к системе для остальных подсистем.

Ключевые функции подсистемы администрирования:

- просмотр списка пользователей системы администратором;
- добавление пользователя, редактирование данных пользователя, удаление пользователя администратором;
- просмотр протоколов загрузки данных с возможностью скачивания загруженных файлов администратором.

Ключевые функции подсистемы справочной информации:

- просмотр справочников организаций;
- добавление, редактирование, удаление записи об организации сотрудником министерства;
- редактирование периода работы пункта проката только своей организации сотрудником ведомства;
- просмотр справочника ТСР;
- добавление, редактирование, удаление записи о ТСР сотрудником министерства;
- просмотр справочника мероприятий;
- добавление, редактирование, удаление записи о мероприятии по ИПРА сотрудником министерства;
- просмотр справочника умерших граждан.

Ключевые функции подсистемы импорта данных:

- добавление записи о факте оказания РП сотрудником ведомства;
- загрузка сотрудником ведомства фактов оказания РП в формате xlsx, заданном приложением к ТЗ;
- загрузка сотрудником ведомства выписок из ИПРА в формате xml, заданном приложением к ТЗ;
- загрузка сотрудником ведомства фактов исполнения ИПРА в формате csv, заданном приложением к ТЗ.

Ключевые функции подсистемы ввода информации о работе пунктов

проката:

- просмотр записей о фактах проката ТСР;

- добавление, редактирование, удаление записей о фактах проката ТСР сотрудником ведомства.

Ключевые функции подсистемы отчетов:

- формирование сотрудником министерства отчета с показателями «Количество детей, нуждающихся в ранней помощи», «Количество детей, получивших раннюю помощь», «Количество семей, включенных в программу РП», «Количество семей, включенных в программу РП, удовлетворенных качеством услуг ранней помощи» на основании фактов оказания РП, отфильтрованных по любой совокупности атрибутов, вывод таблицы фактов, участвовавших в формировании отчета;

- формирование сотрудником министерства отчета с показателями «Всего инвалидов», «Всего ИПРА» на основании мероприятий ИПРА, отфильтрованных по любой совокупности атрибутов, вывод таблицы мероприятий ИПРА, участвовавших в формировании отчета;

- формирование сотрудником министерства отчета «Паспорт реабилитации» в формате xlsx, заданном приложением к ТЗ, за указанный пользователем период;

- формирование сотрудником министерства отчета с показателями «Количество граждан», «Количество ТСР», «Количество пунктов ТСР», «Количество организаций», на основании фактов проката ТСР, отфильтрованных по любой совокупности атрибутов, вывод таблицы фактов проката ТСР, участвовавших в формировании отчета.

Ключевые функции подсистемы обмена со сторонними ИС:

- автоматический прием POST-запросов от сторонних ИС, содержащих выписки ИПРА, факты исполнения ИПРА в формате, заданном в приложении к ТЗ;

- автоматический прием POST-запросов от сторонних ИС, содержащих данные умерших граждан в формате, заданном в приложении к ТЗ;

- просмотр списка ресурсов, имеющих доступ к системе по интерфейсу прикладного программирования (Application programming interface, API) администратором;

- добавление, изменение, удаление ресурсов, имеющих доступ к системе по API администратором;

- просмотр администратором протоколов загрузки данных через API.

1.1.4 Требования к надежности

Система должна обеспечивать возможность корректного завершения функциональных задач и сохранения данных.

Отказы и сбои в работе технических средств пользователей системы, веб-серверов, серверов баз данных и сетевого оборудования не должны приводить к разрушению данных.

Система должна обеспечивать создание ежесуточных резервных копий базы данных системы.

При возникновении сбоев в аппаратном обеспечении, включая аварийное отключение электропитания, система должна автоматически восстанавливать свою работоспособность после устранения сбоев и корректного перезапуска аппаратного обеспечения (за исключением случаев повреждения рабочих носителей информации с исполняемым программным кодом).

Система не должна требовать регулярного администрирования. Штатные средства системы должны позволять проводить удаленное администрирование базы данных и настройку системы (при наличии технической возможности доступа к серверам системы).

1.1.5 Требования к программному обеспечению

Должны выполняться следующие требования к кодированию и декодированию данных:

- UTF-8 для хранения данных;

- Windows-1251 или UTF-8 для информации, поступающей из

систем-источников.

Система должна функционировать на электронных вычислительных машинах пользователей с любой операционной системой, на которой функционируют браузеры:

- Internet Explorer (версия 11.0 и старше);
- Google Chrome (версия 56.0 и старше);
- Mozilla Firefox (версия 45.0 и старше).

1.1.6 Требования к рабочей документации

Рабочая документация должна быть выполнена на русском языке, за исключением официальных наименований используемого программного обеспечения и технического обеспечения, а также кодов программ.

Разработанная система должна сопровождаться следующей документацией:

- программа и методика опытной эксплуатации – документ, соответствующий ГОСТ 19.301–79, представляющий собой описание процедуры проведения опытной эксплуатации установленного и настроенного программного обеспечения;

- регламент эксплуатации – документ, определяющий порядок установки, настройки и администрирования системы;

- руководство администратора системы – документ, определяющий действия администратора и предоставляющий инструкции для управления системой;

- руководство пользователя системы – документ, определяющий действия пользователя и предоставляющий инструкции для выполнения рабочих задач системы.

1.2 Обзор существующих решений

Сам факт существования спроса на новую систему в высокой степени говорит о том, что полного аналога, удовлетворявшего заказчика, на момент

разработки требований не было. Тем не менее, обзор существующих схожих программных решений является полезным этапом анализа, позволяющим установить причины их несоответствия требованиям заказчика, выявить возможные шаблонные подходы к решению поставленных задач или избежать каких-либо ошибок, допущенных при их разработке.

В понятие информационной системы, как правило, включают программное и аппаратное обеспечение, обеспечивающее хранение и обработку данных, в некоторых источниках – и иные ресурсы, обеспечивающие эту функцию, например ответственный персонал [2, 3]. Это понятие крайне широкое. Потому выбор решений для сравнения осуществлялся по следующим, более узким, критериям:

- полнота доступной информации;
- факт эксплуатации государственными структурами;
- интеграция с другими государственными ИС;
- схожесть схемы данных (организации, личные дела, ресурсы, выделяемые им);
- возможность создания отчетов на основе хранимых данных;
- схожесть подхода к реализации пользовательского интерфейса;
- совпадение предметных областей и объектов автоматизации.

В соответствии с этими критериями для сравнения были выбраны следующие системы:

- программный комплекс «Информационное взаимодействие с ФБ МСЭ», разработанный ООО Социнформтех [4];
- программный комплекс «Ранняя помощь», разработанный ООО Социнформтех [5].

Программный комплекс «Информационное взаимодействие с ФБ МСЭ» позволяет формировать региональную межведомственную базу данных выписок ИПРА и осуществлять информационное взаимодействие между всеми ведомствами, участвующим в процессе реализации мероприятий. Заявленные разработчиком функции включают:

- заполнение формы ИПРА в электронном личном деле гражданина с бумажного документа;

- загрузку файлов с ИПРА, полученных из федеральной базы данных «Витрина», в соответствии с утвержденными форматами;

- формирование отчетов.

Это пересекается с необходимой заказчику функциональностью системы. Информация об ИПРА, хранимая в базе данных в целом соответствует данным, которыми должна оперировать ИС, требуемая заказчику.

Система отчетов выглядит менее гибкой, чем необходимо заказчику, в частности, отсутствует большинство требуемых фильтров, а возможности сформировать отчет «Паспорт реабилитации» нет.

Однако, помимо этих функций, заявлена также и автоматизированная отправка информации об исполнении ИПРА в федеральную базу данных «Витрина». Сам комплекс эксплуатируется в составе более крупной информационной системы «Социальное обслуживание населения». Эта система в свою очередь предназначена для автоматизации гораздо более широкого круга процессов, в частности – для оказания различных социальных услуг гражданам через личный кабинет.

Можно сделать вывод, что программный комплекс «Информационное взаимодействие с ФБ МСЭ» непосредственно участвует в оказании государственных услуг и эксплуатируется другими системами с этой целью, а не является строго внутренним средством сбора данных и аналитики. Он не может быть столь же эффективно использован для целей заказчика, как разработанная система.

Программный комплекс «Ранняя помощь» включает в свою заявленную функциональность следующее:

- автоматизация процессов приёма и учёта граждан в органах социальной защиты;

- организация системы единого учета граждан, обращающихся в органы социальной защиты;

- обеспечение сбора, хранения и обработки информации о детях раннего возраста, нуждающихся в получении услуг ранней помощи;

- интеграция с региональной системой межведомственного электронного взаимодействия;

- загрузка-выгрузка пакетов данных в установленных форматах;

- формирование отчётов, списков, статистических данных, печатных форм по утвержденным формам и в произвольном виде.

Из документа «Информация для эксплуатации экземпляра ПО» на комплекс видно, что хотя сущности и наборы их атрибутов частично совпадают с тем, что требуется заказчику, обрабатываемый объем данных намного больше, чем ему требуется. Многие функции, как например автоматизация приема граждан, также являются излишними для заказчика.

Документ «Описание функциональных характеристик» на комплекс дает некоторое понимание предоставляемого им способа генерации отчетов. Пользователь составляет запрос к БД «путем поиска и выбора параметров БД». Далее из результатов этого запроса строится отчет по установленному пользователем шаблону.

Такой способ, очевидно, будет более требовательным к квалификации работника, чем предоставляемый разработанной системой. И в то же время не исключено, что этот способ окажется недостаточно гибким, чтобы составлять любой необходимый заказчику отчет без привлечения специалиста для модификации кода программного комплекса.

Можно сделать вывод, что программный комплекс «Ранняя помощь» в плане своей функциональности и характера хранимых данных является в лучшем случае излишне сложным для целей заказчика.

Сопоставив требования к разработанной системе с двумя наиболее близкими по предметной области информационными системами, можно выделить ряд принципиальных отличий.

Разработанная система – в первую очередь инструмент для отчетности по оказанию услуг. Остальные функции системы предназначены для корректного

сбора данных и составления отчетов (интеграция с системами, предоставляющими данные, избежание дублирования данных и так далее). В то время, как существующие инструменты предназначены для поддержки самого процесса оказания услуг.

Разработанная система предназначена для формирования определенной группы строго регламентированных и описанных отчетов. Причем один из этих отчетов специфичен для региона заказчика. По этой причине существующие решения априори не могут обеспечить необходимые функции.

Разработанная система интегрируется с другими ИС, но как конечный получатель информации, а не как узел, обеспечивающий промежуточную обработку.

Объединяют же разработанную систему с рассмотренными решениями подход к функционированию системы как к межведомственной ИС, узкая предметная область и вытекающие из этого сходства данных.

В ходе подробного рассмотрения описанных выше решений и поиска иных также было отмечено, что для реализации подобных информационных систем успешно используется классическая трехуровневая архитектура. В качестве системы управления базами данных часто используется PostgreSQL. В подробно рассмотренных программных комплексах также можно отметить неприглядные и сложные пользовательские интерфейсы. В некоторой степени, это можно объяснить сложностью данных, однако современные технологии разработки пользовательского интерфейса, мировой опыт в дизайне и свободные стандарты, например – Material Design [6], позволяют делать визуальную составляющую приятнее и проще.

2 Архитектура интеграционной системы и выбор средств разработки

2.1 Трехуровневая архитектура

ГМИС «УРАМ» построена на базе трехуровневой архитектуры.

Трехуровневая архитектура – широко-используемый подход к построению программных систем [7]. Он подразумевает разделение системы на три следующих логических уровня:

- уровень представления;
- уровень бизнес-логики;
- уровень источника данных.

Уровень представления обеспечивает представление данных пользователю, обработку событий пользовательского интерфейса.

Уровень бизнес-логики обеспечивает функции приложения, предназначенные для выполнения его основных целей.

Уровень источника данных обеспечивает работу с данными на более низком уровне. Он отвечает за долговременное хранение, чтение и запись данных.

Уровень бизнес-логики является центральным и взаимодействует с двумя другими. Уровни представления и источника данных напрямую между собой не взаимодействуют вообще.

Само это разделение функциональности и является основным преимуществом данного подхода. Каждый уровень может быть отделен от других физически и может быть модифицирован и настроен без серьезного влияния на другие. Это также гарантирует возможность одновременной разработки каждого уровня и увеличивает безопасность, изолируя данные от представления.

Взаимодействие между уровнями в разработанной системе базируется на клиент-серверной архитектуре. То есть сервер получает запросы от клиента, обрабатывает их и возвращает необходимые клиенту ответы. В дальнейшем в настоящей работе приложение уровня представления будет называться клиентской частью системы или клиентским приложением, а приложение уровня

бизнес-логики – серверной. Уровень источника данных будет представлен объектно-реляционной СУБД.

Взаимодействие между клиентской и серверной частями осуществляется посредством веб-запросов по протоколу HTTP. СУБД получает от серверной части запросы на языке SQL и возвращает результаты их обработки. Более подробные детали взаимодействия между серверной частью и СУБД взяты на себя соответствующим фреймворком.

Необходимо отметить следующую особенность функционирования клиентской части. Приложение исполняет непосредственно браузер пользователя. Однако делает он это, отображая гипертекст и иные статические ресурсы и исполняя код, которые получает через HTTP-запросы с веб-сервера, хранящего приложение.

Таким образом, архитектура ГМИС «УРАМ» может быть представлена схемой на рисунке 3.

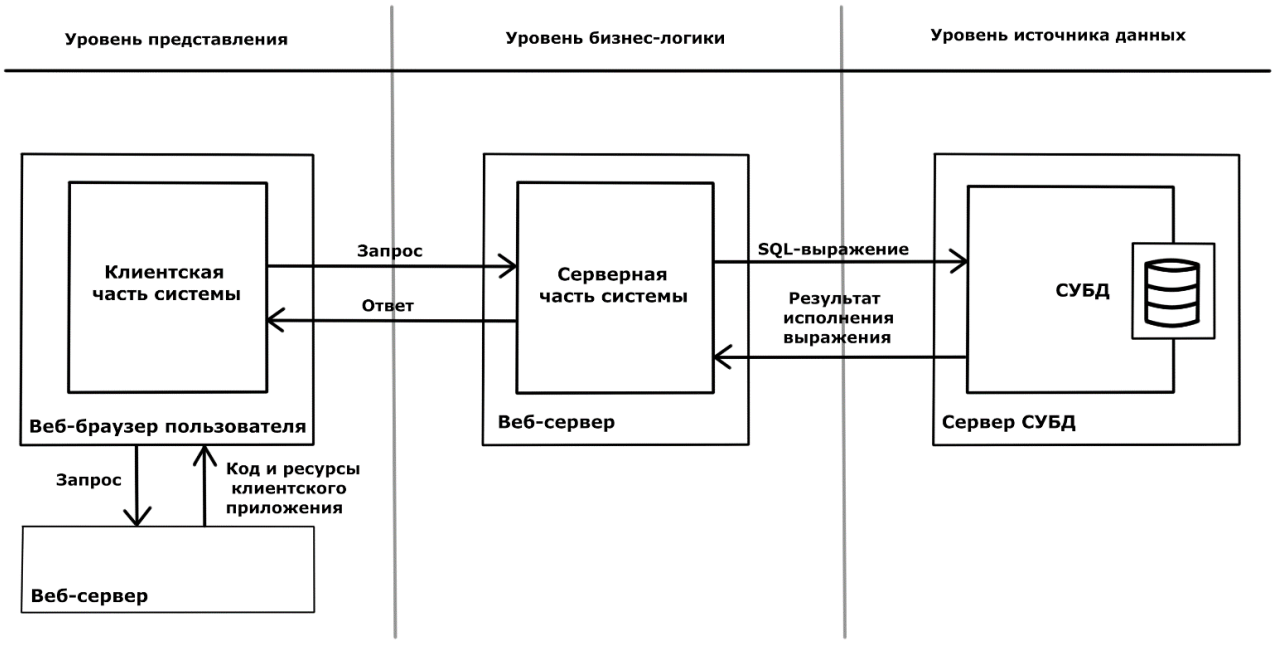


Рисунок 3 – Схема архитектуры системы

2.2 Выбор средств разработки

2.2.1 Выбор средств разработки приложения уровня бизнес-логики

В качестве языка программирования для разработки серверной части был выбран язык Java. Java – язык программирования, компилируемый в байт-код и исполняемый виртуальной машиной. Язык является объектно-ориентированным и поддерживает сильную явную статическую типизацию [8].

Разработанную систему можно охарактеризовать как корпоративное ПО – она разработана для организации и обеспечивает автоматизацию ее бизнес-процессов, подразумевает интеграцию с другими ИС [7]. Java является самым популярным языком в этой сфере [9].

Лицензии на язык Java и на набор инструментов разработчика Java позволяют свободное использование языка в любых, в том числе коммерческих, целях.

Объектно-ориентированная парадигма хорошо сочетается с разработкой информационных систем, потому что позволяет интуитивно отображать сущности бизнес-логики в классы, которые потом можно отображать в таблицы реляционных СУБД.

Для языка Java создан универсальный фреймворк с открытым исходным кодом Spring Framework, который обладает большим количеством встроенных средств для быстрой разработки приложений, подобных требуемому [10]. Среди этих средств можно отметить следующие [11]:

- контейнер инверсии управления, лежащий в самой основе фреймворка;
- MVC-фреймворк, реализующий архитектуру модель-представление-контроллер (Model-view-controller, MVC);
- фреймворк доступа к данным, упрощающий соединение с СУБД и позволяющий использовать фреймворки объектно-реляционного отображения (Object relational mapping, ORM);
- фреймворк аутентификации и авторизации.

Spring Framework является самым популярным в веб-разработке фреймворком для Java [12]. На этом фреймворке построена серверная часть ГМИС «УРАМ», и каждая из перечисленных возможностей была использована в разработке системы.

2.2.2 Выбор средств разработки приложения уровня представления

Клиентское приложение работает в веб-браузере пользователя. Это значит, что выбранный язык программирования должен интерпретироваться указанными заказчиком браузерами. Все эти браузеры снабжены средством исполнения JavaScript.

JavaScript обладает слабой неявной динамической типизацией, что во многом подходит для разработки клиентских приложений, но также может приводит к тяжело уловимым ошибкам, плохо читаемому и поддерживаемому коду. JavaScript поддерживает объектно-ориентированную парадигму [13]. Но объектно-ориентированное программирование в JavaScript отличается от «классических» объектно-ориентированных языков, например, от Java.

Отчасти поэтому был разработан язык программирования TypeScript, транслируемый в JavaScript. Он добавляет опциональную явную статическую типизацию, подход к классам, более похожий на классическое ООП, и ряд других усовершенствований [14]. В результате этих улучшений код на TypeScript легче писать и обслуживать, он также предоставляет больше возможностей для статического анализа, что упрощает написание кода в различных IDE.

По этим причинам в качестве языка программирования для разработки клиентской части был выбран TypeScript.

Для разработки веб-приложений, как правило, используют один из так называемых JavaScript-фреймворков. Это позволяет уменьшить объем повторяемого кода, лучше сочетать разметку и программную составляющую, использовать готовую архитектуру.

Одним из таких фреймворков является Angular. Angular работает с TypeScript, предоставляет широкий набор инструментов и готовых решений для

разработки клиентских приложений [15]. Angular является одним из самых популярных фреймворков для разработки веб-клиентов [12]. У Angular открытый исходный код и лицензия, позволяющая свободное коммерческое использование.

2.2.3 Выбор СУБД

Для хранения данных системой используется реляционная СУБД. ГМИС «УРАМ» должна хранить и обрабатывать структурированные данные о большом количестве сущностей, связанных между собой и обладающих многочисленными атрибутами, и выстраивать эти данные в сложные отчеты. Заказчик также предъявляет высокие требования к надежности хранения данных. Реляционная база данных позволяет выполнять сложные запросы на стороне БД и избавляет разработчика от необходимости заботиться о целостности данных, а приложение от нагрузки их обработки на этом уровне, позволяет обеспечивать необходимую изолированность транзакций [16].

Одной из самых популярных реляционных СУБД является PostgreSQL [12]. PostgreSQL сопровождается выдающейся подробной документацией, а лицензия на нее позволяет полностью свободное использование [17]. Преимущества PostgreSQL перед другими популярными SQL СУБД, такие как, например, полное соответствие требованиям атомарности, согласованности, изоляции и устойчивости и расширяемость, не были критическими при выборе средства хранения данных для системы. Для требуемых задач могли бы подойти и другие реляционные СУБД, но решающую роль сыграли предпочтения заказчика и профессиональный опыт разработчиков.

2.2.4 Используемые средства разработки

Итоговый перечень средств разработки представлен в таблице 1.

Таблица 1 – Используемые средства разработки

Наименование	Применение
Java	язык программирования серверной части
Spring Framework	веб-фреймворк серверной части
Hibernate	ORM-фреймворк серверной части
TypeScript	язык программирования клиентской части
Angular	веб-фреймворк клиентской части
PostgreSQL	СУБД
Git	система контроля версий
GitLab	система управления репозиторием
YouGile	средство управления проектом
IntelliJ IDEA	интегрированная среда разработки
WebStorm	интегрированная среда разработки
DataGrip	интегрированная среда разработки
Postman	инструмент тестирования API сервера

2.3 Архитектура серверной части системы

2.3.1 Модель-представление-контроллер

В контексте как серверной, так и клиентской частей системы можно говорить о применении паттерна модель-представление-контроллер [18]. Это паттерн построения связи бизнес-логики и пользовательского интерфейса. Он состоит в разделении логики приложения на три вынесенные в его название составляющие. Моделью называется часть программы, отвечающая за определение данных, обрабатываемых приложением и бизнес-логику. Контроллер содержит в себе логику управления. Представление – часть программы, отвечающая за отображение данных пользователю и отправку действий пользователя контроллеру.

MVC – абстрактная схема, не определяющая строго свою реализацию,

потому ее воплощения разнятся. В частности, есть разные взгляды на распределение обязанностей между контроллером и моделью, наличие взаимодействия между моделью и представлением и характер этих взаимодействий. Существуют также различные улучшения и расширения MVC, призванные эти неточности конкретизировать.

В случае разработанного серверного приложения, прямого взаимодействия с конечным пользователем системы нет. Есть взаимодействие с клиентской частью, и поэтому представлением следует считать часть программы, обеспечивающую не пользовательский интерфейс, а API.

Как уже было сказано выше, клиент и сервер взаимодействуют друг с другом по протоколу HTTP. Чтобы спроектировать интерфейс прикладного программирования для взаимодействия по HTTP, достаточно простой в реализации и обслуживании на сервере и в использовании на клиенте, следует придерживаться определенной архитектуры. Было принято решение придерживаться основных идей архитектурного подхода REST. При построении API при таком подходе за обмен данными между сервером и клиентом отвечают так называемые точки доступа [19]. По сути, точка доступа – URL, по которому сервер принимает веб-запросы. Веб-запрос получает набор данных от клиента или сторонней системы, вызывает необходимые действия и возвращает данные в ответ на запрос.

Для серверной части именно точки доступа играют роль представления в MVC, потому как они определяют форму отображения данных, передаваемых вовне сервера, и обрабатывают оказываемые извне действия.

Контроллером является часть программы, обрабатывающая приходящие через представление действия. В Spring Framework точки доступа определяются методами так называемых классов-контроллеров. Так как эти классы обрабатывают приходящие запросы, вызывая необходимые методы служб, можно считать, что они играют роль контроллера в MVC для серверной части.

Модель, как было сказано выше – часть программы, ответственная за определение данных и бизнес-логику. Данные в системе хранятся в реляционной

СУБД. Для того чтобы связать таблицы реляционных баз данных и сущности объектно-ориентированного программирования применяют так называемое объектно-реляционное отображение. Оно подразумевает перенос таблиц БД на классы ООП, называемые сущностями, а их строк на объекты этих классов, а также абстракцию от операций по созданию, чтению, обновлению и удалению (Create, Read, Update, Delete, CRUD) записей в сторону операций над классами-сущностями. Для ORM в разработанной системе был применен ORM-фреймворк Hibernate. Для инкапсуляции CRUD-операций над сущностями при работе с ним используется такая абстракция как интерфейсы репозиториев данных [20].

В Spring Framework всю бизнес-логику обычно заключают в так называемые классы служб (или просто службы).

Таким образом, для серверной части роль модели играют сущности и репозитории данных, а также службы.

Соответствие между частями архитектуры модель-представление-контроллер и элементами серверного приложения можно отразить в виде схемы, представленной на рисунке 4.

Рисунок 4 – Схема воплощения архитектуры MVC в серверной части системы

2.3.2 Объектно-реляционное отображение

В корне объектно-реляционного отображения лежат классы-сущности. Класс-сущность описывает структуру сущности, хранящейся в таблице БД, а также ее связи с другими сущностями.

Hibernate предоставляет крайне широкий инструментарий для написания классов-сущностей. В разработанной системе в классах сущностях имеются следующие атрибуты:

- поле первичного ключа;
- поля, представляющие простые поля таблицы БД;
- поля, представляющие связи один-ко-многим и многие-к-одному с другими сущностями;
- правила каскадного удаления связанных сущностей;
- правила автоматической генерации значения поля;
- фильтры со сложным условием;
- фильтры по значению одного поля.

На рисунках 5 и 6 приведены соответственно фрагмент диаграммы классов ORM-сущностей и фрагмент кода класса одной из сущностей. Стрелка между сущностями означает связь многие-к-одному по направлению стрелки. Можно увидеть определение поля первичного ключа с автоматической генерацией, простых полей и связей.

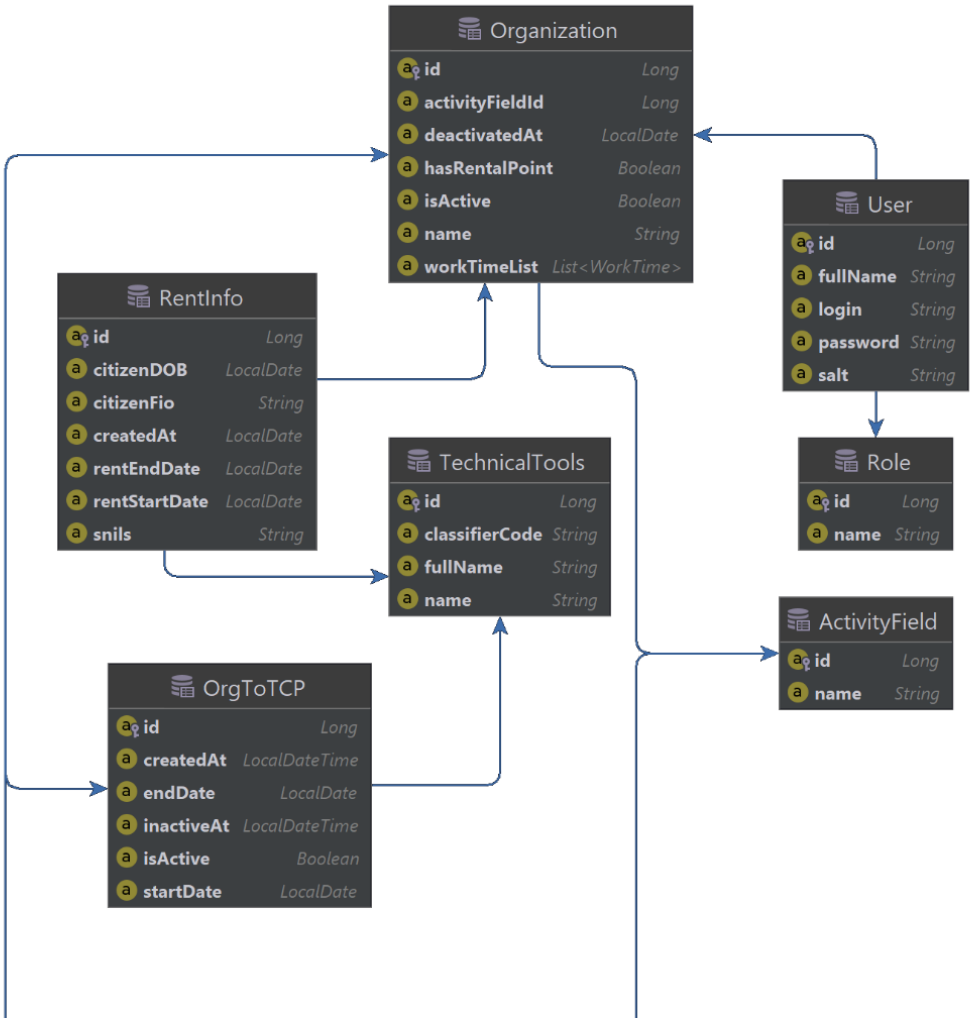


Рисунок 5 – Фрагмент диаграммы классов ORM-сущностей

```java
@Entity
@Table(name = "organization")
public class Organization {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private Long id;

    @Column(name = "name")
    private String name;

    @ManyToOne
    @JoinColumn(name = "activity_field_id")
    private ActivityField activityField;

    @Column(name = "activity_field_id", insertable = false, updatable = false)
    private Long activityFieldId;

    @NotNull
    @Column(name = "has_rental_point", nullable = false, columnDefinition = "boolean default false")
    private Boolean hasRentalPoint = false;

    3 usages
    @OneToMany(mappedBy = "organization")
    private List<OrgToTCP> orgToTCPList = new ArrayList<>();

    @Column(name = "deactivated_at")
    private LocalDate deactivatedAt;

    @NotNull
    @Column(name = "is_active", nullable = false, columnDefinition = "boolean default true")
    private Boolean isActive = true;

    @Type(type = "jsonb")
    @Column(name = "work_times", columnDefinition = "jsonb")
    private List<WorkTime> workTimeList = new ArrayList<>();
```

Рисунок 6 – Фрагмент кода класса-сущности Organization

Hibernate является реализацией стандарта JPA. Для осуществления CRUD-операций при работе с Hibernate в серверном приложении системы используются репозитории данных – соответствующие классам-сущностям наследники интерфейса JpaRepository.

Репозитории позволяют выполнять CRUD-операции без написания кода на SQL. При этом можно осуществлять запросы по значению полей, придерживаясь определенных правил в составлении сигнатур методов репозиториев, а также определять более сложные запросы с на языке HQL, синтаксически очень

схожего с SQL, но интегрированного с ORM сущностями, или нативном SQL. Репозитории поддерживают связи между сущностями и позволяют использовать их в запросах, получая связанные сущности без явного использования оператора соединения или без явного указания условия для соединения. Также репозитории обеспечивают встроенную реализации сортировки и разбиения на странице.

На рисунке 7 приведен фрагмент кода интерфейса репозитория для сущности Organization. Можно увидеть запрос по значению поля и получение связанных сущностей без явного указания условия оператора JOIN.

```
public interface OrganizationRepository extends JpaRepository<Organization, Long> {

    Optional<Organization> findByName(String name);


    @Query(value = "SELECT o FROM Organization o " +
        "LEFT OUTER JOIN o.orgToTCPList " +
        "LEFT OUTER JOIN o.activityField " +
        "WHERE o.id = :id")
    Organization fetchLazyFields(Long id);
```

Рисунок 7 – Фрагмент кода интерфейса репозитория

2.3.3 Объекты передачи данных

Важную роль в коде серверного приложения системы играет такой паттерн проектирования, как объект передачи данных (Data transfer object, DTO) [7]. Он подразумевает создание объектов без поведения, получаемых отображением объектов бизнес-логики. Правило отображения в разработанной системе, как правило, определяется конструктором класса DTO. Объект передачи данных создается для передачи данных при различных вызовах, позволяя сокрыть часть данных объекта бизнес-логики или обеспечить лучшую сериализацию.

Типичными местами применения DTO является возвращение данных в ответ на запрос через точку доступа. Во-первых, может быть необходимость отправить клиенту только часть данных, например – из соображений безопасности. Во-вторых, запрашиваемая сущность может обладать связями с другими. При попытке сериализовать класс-сущность может возникнуть

проблема, связанная с отложенной инициализацией или рекурсивной вложенностью.

Тогда данные скорее всего следует передать в виде лишь одного простого поля или иным образом, не вкладывая все поля связанной сущности. Это можно сделать, создав DTO для этой сущности.

DTO также используются для получения данных из веб-запросов и иных случаев передачи структурированных данных.

На рисунке 8 приведен код одного из DTO классов. Можно увидеть, что в процессе отражения объекта класса-сущности в DTO одна из связанных сущностей тоже преобразуется в DTO во избежание рекурсивной вложенности, а часть данных скрывается.

```
public class OrganizationDTO {
    private String id;
    private String name;
    private String activityField;
    private Boolean hasRentalPoint;
    private List<OrgToTcpDTO> orgToTcpList;
    private List<WorkTime> workTimes;
    private LocalDate deactivatedAt;
    private Boolean isActive;

    4 usages  GenaDyomin +1
    public OrganizationDTO(@NotNull Organization organization) {
        id = organization.getId().toString();
        name = organization.getName();
        activityField = organization.getActivityFieldName();
        hasRentalPoint = organization.getHasRentalPoint();
        deactivatedAt = organization.getDeactivatedAt();
        isActive = organization.getIsActive();
        workTimes = organization.getWorkTimeList();
        orgToTcpList = organization.getOrgToTCPList().stream() Stream<OrgToTCP>
            .filter(orgToTCP → orgToTCP.getIsActive() ≠ null && orgToTCP.getIsActive())
            .map(OrgToTcpDTO::new) Stream<OrgToTcpDTO>
            .collect(Collectors.toList());
    }
}
```

Рисунок 8 – Код DTO класса

2.3.4 Службы

Весь код бизнес-логики, за исключением сущностей, расположен в

службах. Функциональность служб можно условно разбить на следующие группы:

- простые CRUD-операции;
- аутентификация и авторизация;
- импорт данных из файлов;
- сохранение файлов;
- генерация отчета;
- автоматическое выполнение запланированных действий;
- экспорт отчета.

При вызове метода для выполнения простых CRUD-операций службами, как правило, выполняется проверка корректности полученных данных и непротиворечивость данных существующим. В случае положительного результата проверки выполняется операцию и возвращается запрошенный или измененный запросом ресурс или его метаданные. В противном случае вызывается исключение, которое будет далее обработано контроллером.

Аутентификация и авторизация работает на основе токена сессии. При авторизации пользователя создается и возвращается временный токен. Токен передается в качестве заголовка HTTP-запроса и контроллер вызывает соответствующий метод службы авторизации для проверки его валидности. При выходе пользователем из системы контроллер вызывает соответствующий метод службы авторизации для удаления токена. Для аутентификации используются логин и пароль.

Служба авторизации не проверяют роль пользователя для конкретных действий, это возложено на клиентскую часть. Как правило, это не является хорошей практикой, но оправдано в случае с разработанной системы. Веб-серверы клиентской и серверной частей системы работают в закрытом контуре, то есть подключение к ним возможно только для сотрудника организации и после авторизации. Доступ к особенно важным функциям системы – функциям администратора и загрузки данных, возможен только с IP-адресов из белого списка. По этим причинам была сочтена излишней

дополнительная проверка роли пользователя на серверной части.

Импорт данных может быть произведен из файлов трех форматов – csv, xml, xlsx. Структура записей в файлах установлена ТЗ. Для непосредственной работой с файлами службы используют соответствующие классы-инструменты. Это статические классы, который по сути являются пространствами имен для функций, они инкапсулируют специфическую работу с файлами, оставляя службам данные из них. Для работы с форматом xml используется фреймворк Java Architecture for XML Binding стандартной библиотеки Java, для работы с форматом xlsx – класс XSSFWorkbook из библиотеки Apache POI. Далее данные обрабатываются службами так же, как данные для простых CRUD-запросов.

Генерация отчетов отличается сложностью запроса к данным и количеством фильтров. Как можно больший объем работы по подготовке данных переложен на СУБД. По этой причине при генерации отчетов основной объем работы служб – обработка полученного DTO с фильтрами. Служба проверяет значения всех возможных полей DTO и устанавливает соответствующие фильтры. В остальном процесс мало чем отличается от обработки простых запросов на чтение.

Один из отчетов можно получить только в формате xlsx. Класс XSSFWorkbook предоставляет методы для работы с xlsx файлами на уровне абстракций Microsoft Excel, таких, как книги, листы, ячейки. Генерация отчета к экспорту несколько отличается от генерации остальных отчетов для вывода на клиент: ряд фильтров накладывается безусловно, а пользователь может указать только один фильтр. После получения данных из репозитория служба генерирует файл формата xlsx. Форма сгенерированного отчета соответствует установленной ТЗ, жестко определена кодом системы и может быть изменена только разработчиком.

Для автоматического исполнения запланированных действий соответствующая служба использует встроенное решение запланированных задач Spring Framework. Оно позволяет реализовать регулярное выполнение задач, лишь указав необходимый метод и сконфигурировав временной интервал.

В разработанной системе служба автоматического исполнения используется, например, для регулярной проверки наступления дат деактивации сущности. В случае наступления указанной для сущности даты деактивации служба установит у сущности соответствующий флаг и таким образом выполнит ее мягкое удаление.

2.3.5 Контроллеры и точки доступа

В Spring Framework присутствует возможность определять точки доступа аннотируя требуемым образом методы в классах-контроллерах. Совмещая аннотации, сигнатуры методов и типы возвращаемых ими значений можно обеспечить необходимую обработку входящих HTTP-запросов.

Построение API сервера системы в целом соответствует основным принципам REST. Однако сделан ряд отступлений. В частности, в URL запросов, помимо указания на ресурс, присутствуют указания на действие, например update, page, create. Это сделано частично потому, что действия, соответствующие одной комбинации ресурса и метода запроса, могут получать один и тот же результат, но в разном представлении. Так – GET запрос к ресурсу без Id может вернуть простой список всех объектов, а может вернуть одну страницу такого списка заданного размера. Также это помогает соблюдать требования заказчика по интеграции со сторонними системами без создания исключений из правил построения API.

Остальные принципы, такие как архитектура, ориентированная на ресурсы, и отсутствие состояния в работе с API, соблюдены.

Контроллеры сделаны тонкими. При получении запроса, определяется соответствующий ему метод контроллера, откуда вызывается требуемый метод соответствующей службы, проверяется факт вызова исключения, возвращается результат его работы, обернутый в класс HTTP-ответа.

На рисунках 9 и 10 приведены соответственно фрагмент списка точек доступа и спецификация точки доступа для добавления организации.

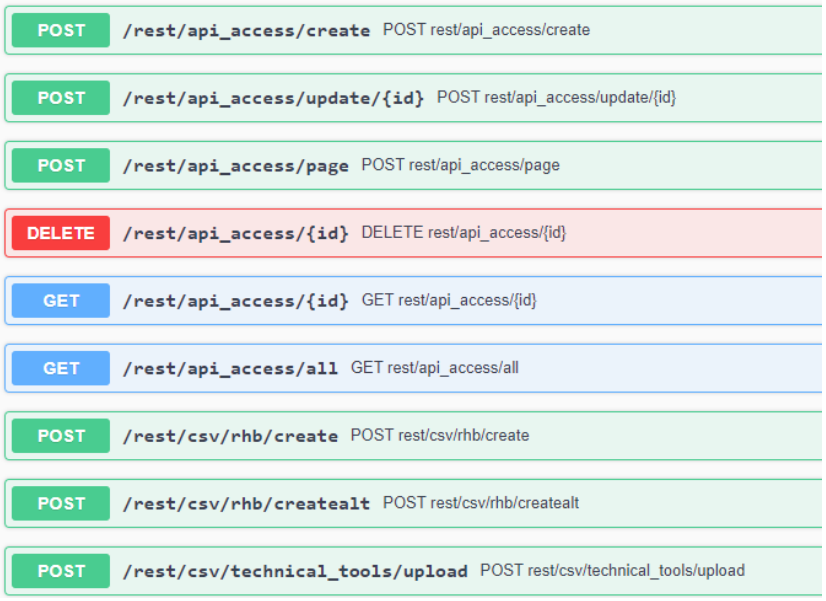


Рисунок 9 – Фрагмент списка точек доступа

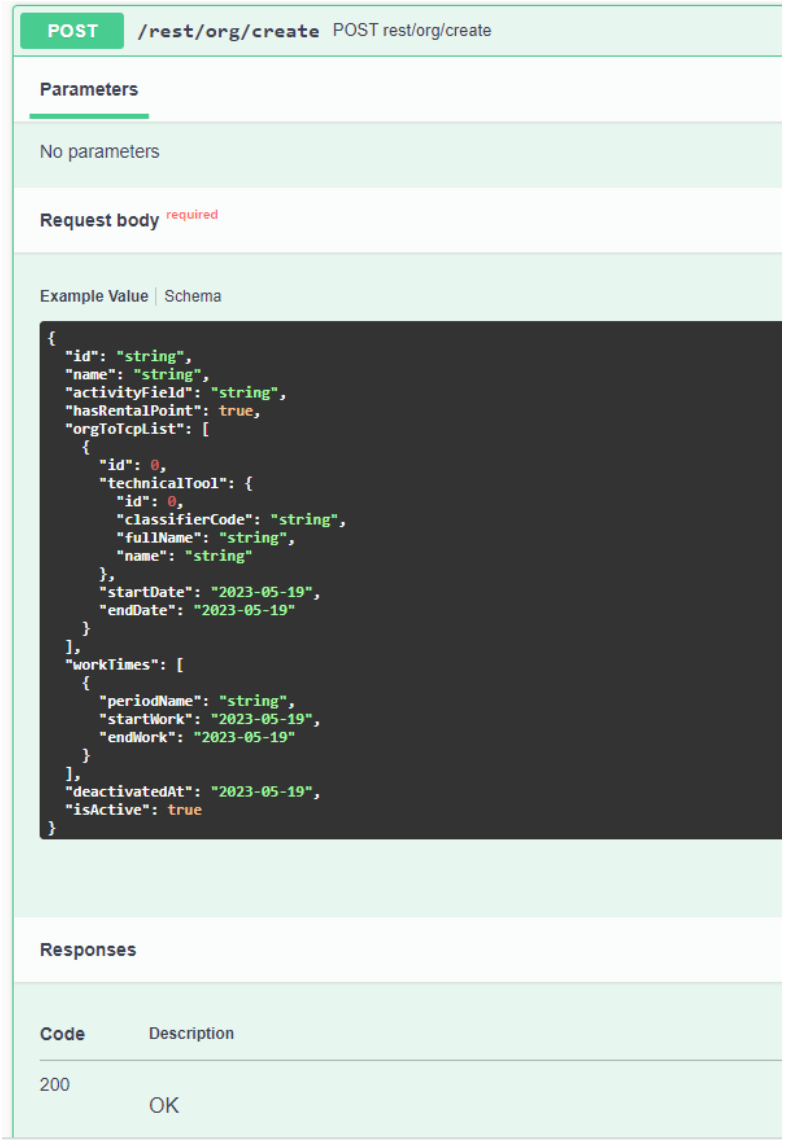


Рисунок 10 – Спецификации точки доступа для добавления организации

2.4 Архитектура клиентской части системы

2.4.1 Структура проекта на Angular

Клиентская часть системы выполнена на фреймворке Angular. Двумя основными конструкциями в Angular являются так называемые модули и компоненты.

Клиент разработанной системы состоит из двух модулей – корневого, являющегося точкой входа, и модуля, ответственного за маршрутизацию. Модули включают компоненты, а также сервисы и другие необходимые компонентам ресурсы.

Каждая страница системы и каждое модальное окно представлены компонентами Angular. Компонент представляет собой класс TypeScript, в котором определяются данные и логика, за которые он ответственен. Компонент также связан с шаблоном, который определяет его внешний вид. Шаблоны написаны с использованием HTML-кода и других конструкций – директив Angular, позволяющих добавить некоторую динамику, например – условное отображение элементов, а также привязок данных.

Общую для нескольких компонентов логику принято выносить в службы. Например, в клиенте разработанной системы в службы вынесены запросы к серверу. Определения данных, с которыми работают компоненты и службы, вынесены в интерфейсы TypeScript.

При воздействии на интерактивный элемент пользовательского интерфейса происходит вызов соответствующей функции-обработчика компонента, которая обрабатывает совершенное пользователем действие и введенные данные, если действие подразумевает заполнение формы. Посредством службы компонент отправляет запрос на сервер, обрабатывает нужным образом полученный ответ и представляет его пользователю.

Таким образом с точки зрения MVC в клиентском приложении шаблоны играют роль представлений, компоненты – роль контроллеров, а моделью являются интерфейсы и получаемы с сервера данные, которые объединяются с

шаблоном для построения готовой страницы.

2.4.2 Одностраничное приложение

Клиентская часть системы выполнена в виде так называемого одностраничного приложения. Вся функциональность приложения доступна из одной веб-страницы. При доступе пользователя к различным функциям браузер обновляет лишь части этой страницы вместо того, чтобы полностью перезагружать страницу. Такой подход обеспечивает более легкое развертывание веб-приложения и снижает нагрузку на сервер, предоставляет более плавный и отзывчивый пользовательский опыт.

Angular спроектирован специально для разработки одностраничных приложений, а потому предоставляет широкий инструментарий для их разработки. Сама описанная компонентно-ориентированная архитектура фреймворка уже служит таким инструментом.

Как было сказано выше, один из модулей приложения предназначен для маршрутизации. Маршрутизация в Angular – это один из элементов воплощения одностраничной архитектуры. Она позволяет определить, какие представления соответствуют текущему URL.

Также важной технологией, используемой Angular для построения одностраничного приложения, является подход AJAX [21]. Он подразумевает асинхронные запросы к серверу без перезагрузки страницы полностью. При этом запрашивается и возвращается не готовая веб-страницы, а данные, необходимые для обновления страницы клиентом. В Angular сейчас, как правило, для этого используется формат JSON. Он же используется клиентской частью ГМИС «УРАМ».

Здесь же можно отметить такой подход к отображению данных, как разбиение на страницы. Он применяется при получении большого списка однотипных объектов, например – при выполнении запроса на получение всех записей таблицы. Отображение такого результата полностью на одной веб-странице неудобно для пользователя, а также может быть затруднительно для

браузера. Поэтому в разработанной системе их вывод осуществляется постранично. Разбиение на страницы делается на стороне сервера встроенным средством Spring Framework, от клиента требуется только указание при запросе размера страницы, поля сортировки и номера необходимой страницы. Важно указать, что разбиение на страницы актуально только для подготовки данных, которые будут отображены пользователю в браузере.

3 Реализация интеграционной системы

3.1 Схема базы данных

Модель данных находится во второй нормальной форме. В ходе доработки системы были созданы новые таблицы БД и соответствующие им сущности ORM: умершие граждане, организации, метаданные загруженных файлов и другие. Некоторые сущности были расширены новыми полями или связями.

На рисунке 11 изображен фрагмент схемы данных. Можно увидеть связи один-ко-многим и связь многие-ко-многим, реализованную через дополнительную таблицу.

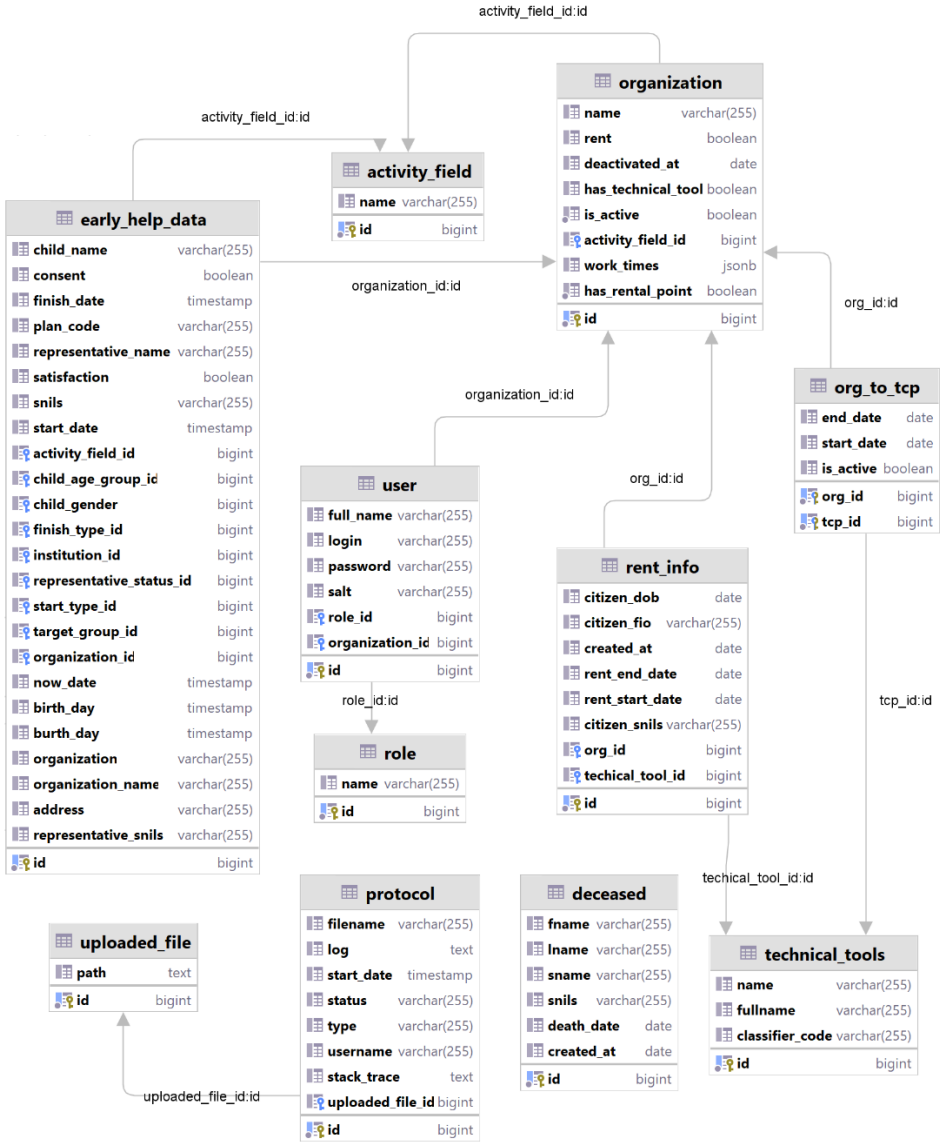


Рисунок 11 – Фрагмент схемы базы данных

3.2 Фильтрация данных

Функции системы подразумевают фильтрацию данных по разнообразным условиям – по простым параметрам одной таблицы (главным образом – для фильтрации справочников) и на основе соединений таблиц (например – для составления отчетов). Вся фильтрация выполняется на уровне СУБД. Для задания фильтров используются фильтры Hibernate на уровне сущности и Criteria API.

В первом случае возможные фильтры определяются в коде сущности и накладываются на сессию перед тем, как ORM-фреймворк осуществит запрос. Этот подход повсеместно использовался в системе отчетов. Это позволило явно определить фильтры отдельно друг от друга (каждый в отдельной аннотации @Filter) и накладывать необходимые фильтры в ходе разбора полученных данных с клиента. При этом сами условия в определении фильтра задаются на языке SQL, что особенно полезно для сложных условий с подзапросами. На рисунке 12 приведен пример определения двух фильтров для отчета. Первый – на простое поле (реализация ИПРА началась раньше конечной даты выборки), а второй – с подзапросом (гражданин на начальную дату выборки не был мертв).

```
@FilterDef(
        name = "prgStartDateFilter",
        parameters = @ParamDef(name = "startDate", type = "java.time.LocalDate")
)
@Filter(
        name = "prgStartDateFilter",
        condition = "end_date > :startDate"
)
@FilterDef(
        name = "prgDeceasedFilter",
        parameters = @ParamDef(name = "startDate", type = "java.time.LocalDate")
)
@Filter(
        name = "prgDeceasedFilter",
        condition = "id not in (SELECT p.id FROM PRG p " +
                "JOIN DECEASED d ON " +
                        "UPPER(p.snils) like '%' || UPPER(d.snils) || '%' AND " +
                        "UPPER(p.lname) like '%' || UPPER(d.lname) || '%' AND " +
                        "UPPER(p.sname) like '%' || UPPER(d.sname) || '%' AND " +
                        "UPPER(p.fname) like '%' || UPPER(d.fname) || '%' " +
                "WHERE d.death_date < :startDate)"
)
@FilterDef(
        name = "prgEndDateFilter",
        parameters = @ParamDef(name = "endDate", type = "java.time.LocalDate")
)
```

Рисунок 12 – Пример определения фильтров на уровне сущности

Хотя Criteria API также позволяет строить сложные запросы, в разработанной системе он использовался в основном для создания запросов с простыми фильтрами. Для этого был написан класс-фабрика, создающий запрос Criteria API для любой сущности по имени параметра и условию (LIKE, EQUAL или IN). Это решение позволило более лаконично создавать запросы, в которых сложная фильтрация была не нужна. На рисунке 13 приведен пример использования класса для простой фильтрации с помощью Criteria API.

```
if (rentInfoPaginationDTO.getFilter().getRentStartDate() != null) {
    filterFields.add(new FilterField( fieldName: "rentStartDate",
            rentInfoPaginationDTO.getFilter().getRentStartDate().plusDays( daysToAdd: 1),
            FilterType.EQUAL));
}

if (rentInfoPaginationDTO.getFilter().getRentEndDate() != null) {
    filterFields.add(new FilterField( fieldName: "rentEndDate",
            rentInfoPaginationDTO.getFilter().getRentEndDate().plusDays( daysToAdd: 1),
            FilterType.EQUAL));
}

return filterFactory.getPage(RentInfo.class, pageSize, page,
                Arrays.asList("organization.name", "rentStartDate"), filterFields)
        .map(rentInfo -> rentInfoRepository.fetchLazyFields(rentInfo.getId()))
        .map(rentInfo -> new RentInfoDTO(rentInfo, rentInfoIsActive(rentInfo)));
```

Рисунок 13 – Пример наложения простых фильтров с помощью класса-фабрики для запросов Criteria API

3.3 Подсистема отчетов

Система позволяет генерировать четыре разных отчета. Три из них оформлены в виде нескольких численных показателей и таблицы (РП, ИПРА, прокат ТСР), один («Паспорт реабилитации») – в виде XLSX документа установленного формата. В ходе развития системы логика формирования отчета о прокате ТСР и паспорта реабилитации была реализована впервые, двух других – доработана с учетом новых сущностей в системе (например, умерших граждан).

На рисунке 14 представлен фрагмент сформированного отчета о прокатах ТСР, на рисунке 15 – фрагмент отчета «Паспорт реабилитации».

Показатели

Количество граждан	Количество ТСР	Количество пунктов ТСР	Количество организаций
4	7	13	6

Данные

№	Дата ввода	Организация	ФИО гражданина	Дата рождения гражданина	СНИЛС гражданина	ТСР	Дата предоставления с	Дата предоставления по
1	2022-11-05	[redacted]	[redacted]	[redacted]	[redacted]		2022-10-31	2022-11-18
2	2022-12-05	[redacted]	[redacted]	[redacted]	[redacted]		2022-12-14	2022-12-16

Рисунок 14 – Фрагмент отчета о прокатах ТСР

Наименование показателей	№ строки	Инвалиды в возрасте 18 лет и старше, имеющие рекомендации в ИПРА/ ИПР, чел.				Дети-инвалиды, имеющие	
		всего	из них, проживающие в стационарных организациях:			всего	из них, прожив
			социальной защиты населения	образования	здравоохранения		социальной защ населения
1	2	3	4	5	6	7	8
Численность инвалидов в возрасте 18 лет и старше и детей-инвалидов, имеющих рекомендации в ИПРА/ ИПР по направлениям реабилитации и абилитации:	01	81170				7378	
по медицинской реабилитации	02	120				12	
по реконструктивной хирургии	03						
по протезированию и ортезированию	04	14				4	
по санаторно-курортному лечению	05	4					
по профессиональной ориентации	06	17					
по общему и профессиональному образованию	07	12	X	X	X	12	
по профессиональному обучению	08						
по содействию в трудоустройстве (в том числе на	09	53					
по производственной адаптации	10						
по социально-средовой реабилитации и абилитации	11	62575				5372	
по психолого-педагогической реабилитации и абилитации	12	X	X	X	X		
по социально-психологической реабилитации и	13	41275				5279	
по социокультурной реабилитации и абилитации	14	43349				4606	
по социально-бытовой адаптации	15	36541				3840	
по физкультурно-оздоровительным мероприятиям, спорту	16	39				5	

Рисунок 15 – Фрагмент отчета «Паспорт реабилитации»

3.4 Импорт файлов и загрузка данных через API

Система позволяет загружать ИПРА, факты исполнения ИПРА и списки умерших граждан сторонними системами через API. В виде файлов пользователь может загрузить ИПРА, факты исполнения ИПРА, факты оказания РП и списки ТСР. Система распознает все принятые различными ведомствами форматы хранения в электронном виде данной информации (5 версий ИПРА бюро медико-социальной экспертизы, ИПРА федерального реестра инвалидов), в том числе устаревшие. Однако по API система принимает лишь актуальные на

момент разработки системы форматы. Используемая версия определяется системой, явного указания версии в файле не требуется.

В работе служб по импорту данных важно отметить одну особенность. При запросе на импорт данных создается ресурс – протокол, который хранит журнал с записями о процессе импорта. Операции по импорту происходят асинхронно, это связано с тем, что они могут занять много времени. Протокол импорта проверяется клиентом, клиент оповещает пользователя о ходе импорта.

В ходе доработки системы была добавлена функция сохранения файлов на долговременный носитель. Все загруженные для импорта файлы сохраняются в файловой системе сервера. При загрузке копия файла сохраняется под гарантированно уникальным именем, при этом метаданные о загруженных файлах вместе с этим именем сохраняются в БД. Запись о метаданных имеет связь с соответствующим протоколом импорта. Администратор видит в пользовательском интерфейсе эти записи и может загрузить копию, если она не была удалена из файловой системы.

На рисунке 16 приведен фрагмент таблицы протоколов импорта. Можно увидеть сообщение об ошибке, возникшей при попытке скачать копию загруженного файла (файл был удален из файловой системы). На рисунке 17 – фрагмент протокола частично успешного (часть записей в файле прошла проверку и была сохранена) импорта. Можно увидеть причины непринятия системой строк файла. На рисунке 18 приведен фрагмент таблицы протоколов загрузки данных по API.

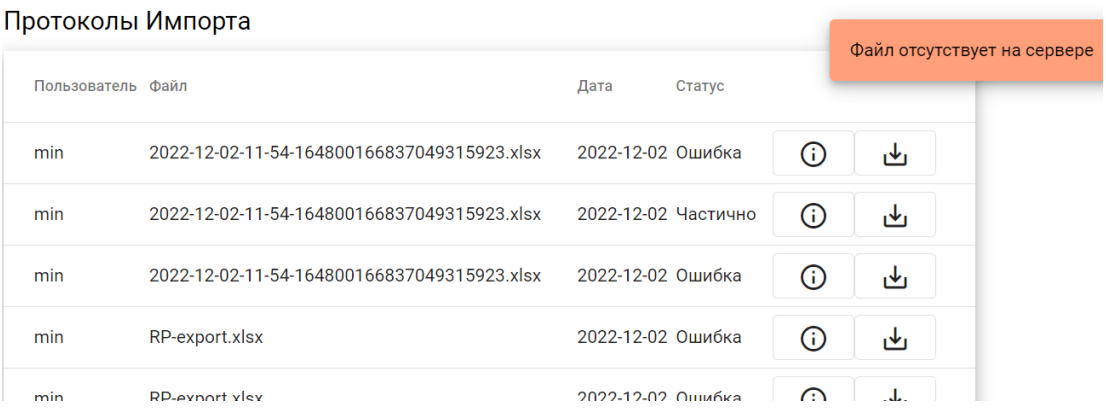


Рисунок 16 – Фрагмент таблицы протоколов импорта

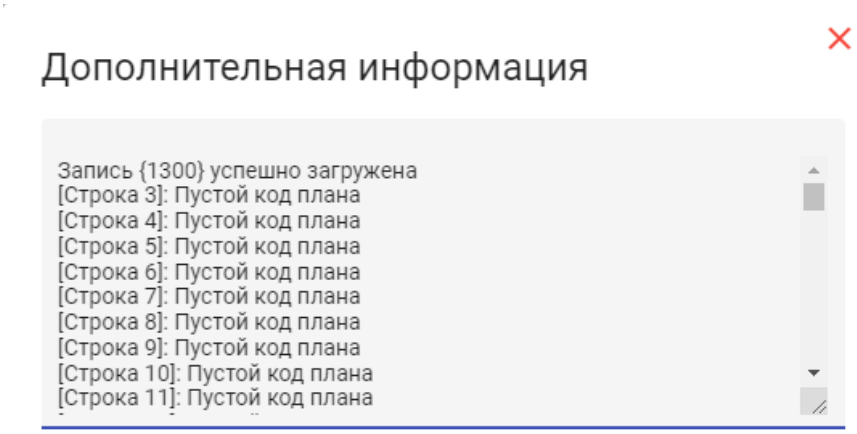


Рисунок 17 – Протокол частично успешного импорта фактов оказания РП

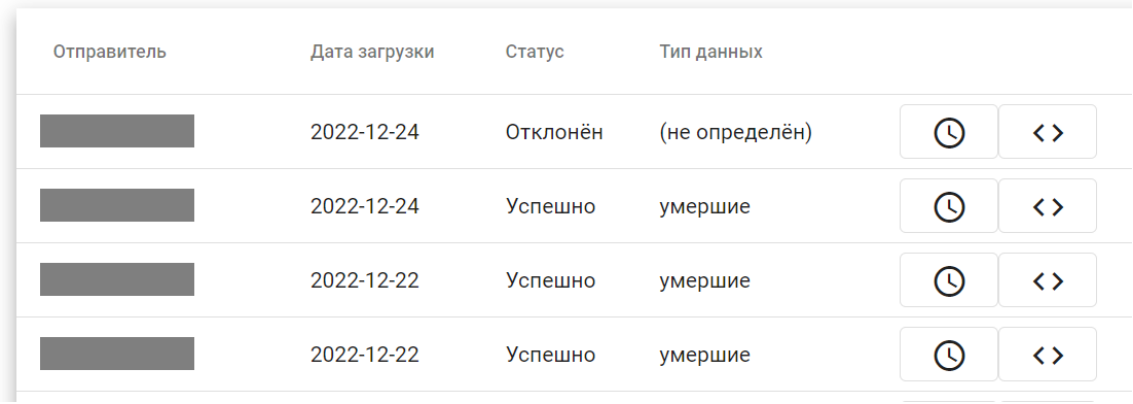


Рисунок 18 – Фрагмент таблицы протоколов загрузки данных по API

3.5 Общий подход к разработке пользовательского интерфейса

Одностраничное приложение Angular было разработано с использованием библиотеки Angular Material, что во многом и определило внешний вид клиента.

Элементы пользовательского интерфейса разбиты на так называемые вкладки, переключение между которыми осуществляется с помощью навигационной панели сверху. Некоторые вкладки также обладают своей навигационной панелью, с помощью которой можно переключаться между частями этих вкладок.

Вкладки и их части сделаны с учетом указанного в ТЗ разбиения системы на подсистемы, но с отступлениями в пользу логичной для пользователя группировки функций.

Данные в системе, как правило, отображаются в виде таблиц. Таблицы,

если система подразумевает такие функции, имеют в шапке поля для установки фильтров, а в крайней правой колонке кнопки для действий, связанных с конкретной записью. Для вызова формы добавления данных (файлом или ручным вводом информации), если это возможно, предусмотрены кнопки, находящиеся над шапкой таблицы.

На рисунке 19 изображен пользовательский интерфейс системы. Открыта вкладка «Справочники», часть этой вкладки «Организации». Можно увидеть таблицу организаций, поля для фильтров в шапке таблицы, кнопки для редактирования записей в крайней правой колонке таблицы, кнопку для добавления организации над таблицей.

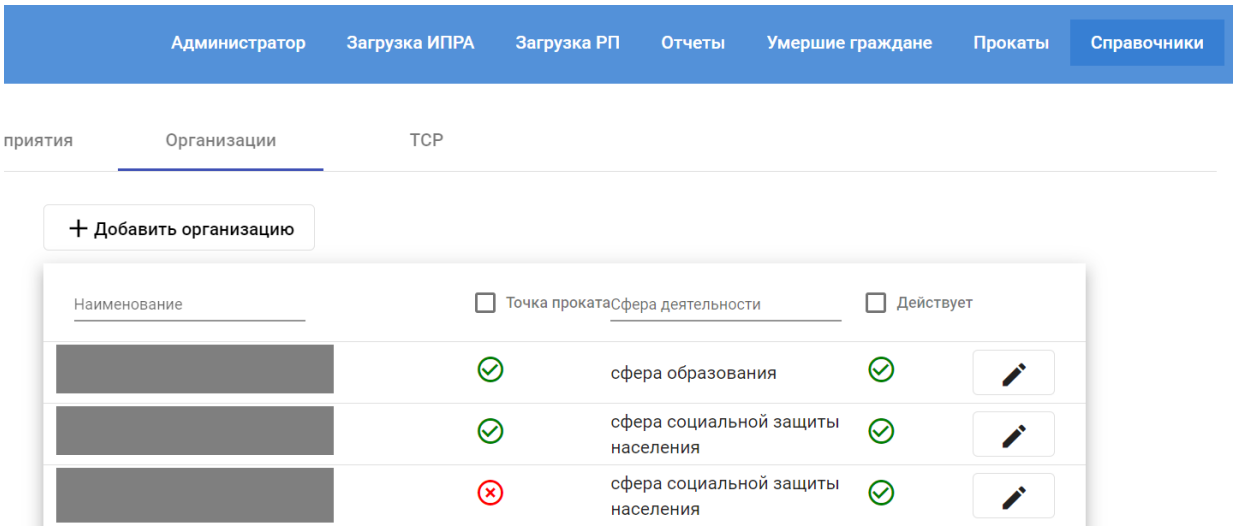


Рисунок 19 – Пользовательский интерфейс справочника организаций

3.6 Написание рабочей документации

Система сопровождена всеми документами, требуемыми заказчиком. По завершении доработки системы документы были дополнены в соответствии с новыми функциями. Во вступительной части каждого документа указано назначение документа и кратко изложены общие сведения о системе.

Программа опытной эксплуатации описывает необходимую последовательность испытаний системы. Испытания сводятся главным образом к исполнению пользовательских сценариев. Пример описания такого сценария приведен на рисунке 20.

Пользователь:
Тип:
- Администратор
Доступные вкладки:
- Администрирование
- Загрузка ИПРА
- Загрузка РП
- Отчеты

Сценарий:
1. Работает в подсистеме администрирования
    1.1. Переходит во вкладку «Администрирование»
    1.2. Создает нового пользователя системы
        1.2.1. Нажимает на кнопку «Добавить пользователя»
        1.2.2. В появившемся модальном окне заполняет все поля
        1.2.3. Выбирает роль «Сотрудник ведомства»
        1.3.3. Сохраняет пользователя нажимая на кнопку «Добавить»
    1.3. Выходит из системы нажимая на кнопку выхода
    1.4. Авторизуется под только что созданным пользователем и демонстрирует доступные ему модули
    1.5. Выходит из системы нажимая на кнопку выхода
    1.6. Авторизуется под изначальным пользователем и демонстрирует доступные ему модули
    1.7. Ищет пользователя по ФИО
    1.8. Меняет пользователю пароль
        1.8.1. Нажимает на кнопку редактирования пользователя
        1.8.2. В появившемся модальном окне нажимает на кнопку «Изменить пароль»
        1.8.3. Меняет пароль и сохраняет, нажав на кнопку «Сохранить»

Рисунок 20 – Описание сценария в регламенте опытной эксплуатации

Ввиду наличия подробных руководств пользователя и администратора регламент эксплуатации главным образом содержит требования к персоналу и описание режима работы системы. Для настоящей работы данный документ не представляет большого интереса.

Руководство пользователя главным образом подробно описывает элементы пользовательского интерфейса и дает инструкции по выполнению пользователем своих задач в системе. Пример такой инструкции приведен на рисунке 21.

4.6.1. Отчет «ИПРА»

Панель ИПРА состоит из перечня базовых фильтров поиска, управляющих кнопок и таблицы результатов (Рисунок 30). Такие фильтры как Сфера деятельности – Тип мероприятия – Мероприятие связаны между собой, т.е. при выборе определенной сферы в справочнике типов мероприятий отображаются только те, что соответствуют данной сфере. При указании периода запроса в выборку попадут записи для всех граждан, кроме тех, которые умерли раньше начала выборки. Для подтверждения примененных фильтров необходимо нажать на кнопку «Найти», а для сброса в значения фильтров по умолчанию – кнопку «Сбросить».

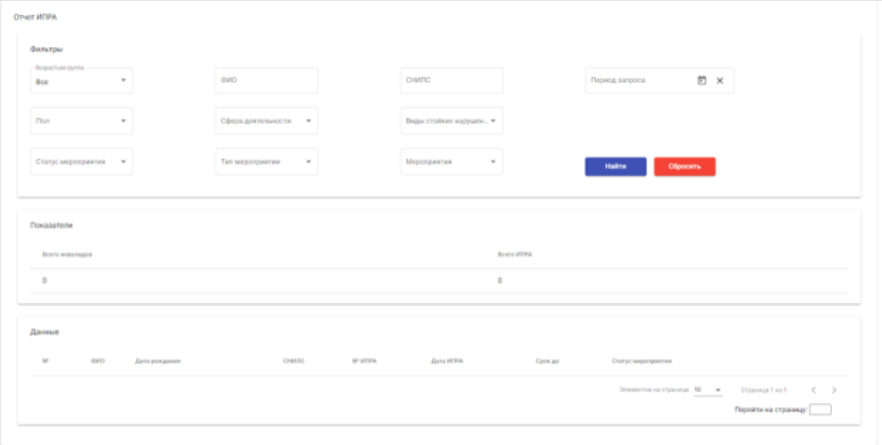
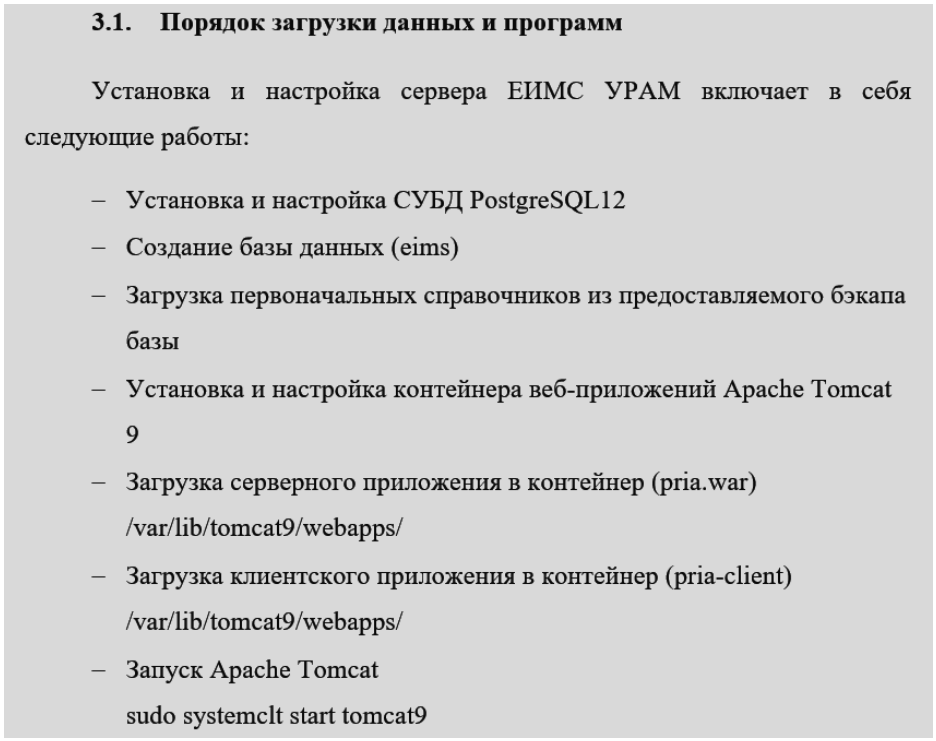


Рисунок 30. Фильтр ИПРА

В результате фильтрации отобразится таблица с данными, подходящими под эти фильтры, а также показатели на основе этих данных (Рисунок 31).

Рисунок 21 – Инструкция в руководстве пользователя

Руководство администратора определяет порядок развертывания, проверки работоспособности и администрирования системы. В остальном оно содержит инструкции формата, аналогичного инструкциям в руководстве пользователя, но для целей администратора системы. Описание развертывания системы из руководства администратора приведено на рисунке 22.



3.1. Порядок загрузки данных и программ

Установка и настройка сервера ЕИМС УРАМ включает в себя следующие работы:

- Установка и настройка СУБД PostgreSQL12
- Создание базы данных (eims)
- Загрузка первоначальных справочников из предоставляемого бэкапа базы
- Установка и настройка контейнера веб-приложений Apache Tomcat 9
- Загрузка серверного приложения в контейнер (pria.war) /var/lib/tomcat9/webapps/
- Загрузка клиентского приложения в контейнер (pria-client) /var/lib/tomcat9/webapps/
- Запуск Apache Tomcat
  sudo systemclt start tomcat9

Рисунок 22 – Порядок развертывания системы в руководстве администратора

3.7 Безопасность

Система рассчитана на функционирование в защищенной сети. Таким образом, любой запрос к серверу или клиенту системы может исходить только от авторизованного в этой сети устройства. Все действия в сети записываются в журнал. Отчасти это снижает требования к безопасности ГМИС «УРАМ», однако ТЗ все равно устанавливает со ссылкой на законы РФ ряд соответствующих требований. Для соответствия им в системе предусмотрены следующие меры:

- использование протокола HTTPS;
- хранение результатов вычисления хеш-функции от паролей, а не паролей в исходном виде;
- проверка доступа пользователей к конкретным функциям системы в зависимости от их роли (только на клиенте);
- белый список IP-адресов с доступом к загрузке данных по API.

3.8 Тестирование

В ходе разработки новых функций системы в первую очередь добавлялись

соответствующие функции серверной части. Реализуемые в порядке от сущностей и репозиториев «вверх» к методам служб и точкам доступа методы вызывались с различными тестовыми данными. Созданные точки доступа проверялись путем отправки запросов с помощью приложения Postman. Далее новые функции реализовывались на клиентской части и тестировались путем выполнения требуемых пользовательских сценариев. Эта часть тестирования выполнялась с точным знанием внесенных в код изменений и со стремлением к наибольшему покрытию кода.

Далее изменения тестировались другим разработчиком, имеющим более глубокое понимание требований к функциональности системы, но не вникающим во внесенные изменения в коде, хотя и имеющим полный доступ к коду системы.

Таким образом можно говорить о применении методов тестирования белого и серого ящика [22, 23].

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы бакалавра было осуществлено развитие ГМИС «УРАМ». Были выполнены все поставленные задачи. В разделе 1 была изучена предметная область и требования к системе, рассмотрены схожие программные решения. В разделе 2 описана архитектура системы и указаны средства разработки. В разделе 3 описан ход реализации системы. В ходе развития системы был затронут каждый аспект существующей на момент принятия технического задания разработки, были внесены значительные изменения в уже имеющийся код сервера и клиента и схему БД.

Модифицированная система была принята заказчиком и введена в эксплуатацию, ожидаемый результат внедрения модифицированной системы был достигнут, что документально подтверждено заказчиком. Таким образом, поставленную цель можно считать выполненной.

В ходе ВКР были применены компетенции, полученные в ходе освоения учебной программы, в особенности – дисциплин «Объектно-ориентированное программирование», «Базы данных», «Серверное программирование», «Разработка Web-приложений», «Разработка корпоративных информационных систем», «Инженерия требований к ПО», «Тестирование ПО».

СПИСОК СОКРАЩЕНИЙ

В настоящей работе применены следующие сокращения:

ГМИС «УРАМ» – государственная межведомственная информационная система «Учет реабилитационных и абилитационных мероприятий»;

ЕИМС – единая межведомственная информационная система;

ИПРА – индивидуальная программа реабилитации и абилитации;

РП – ранняя помощь;

ТСР – техническое средство реабилитации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Правительство Красноярского края. Постановления. Об утверждении государственной программы Красноярского края "Развитие системы социальной поддержки граждан" : Постановление Правительства Красноярского края от 30 сентября 2013 года № 507-п : с изменениями на 18 апреля 2023 // Кодекс : профессиональная справочная система. – URL: https://docs.cntd.ru/document/465805008 (дата обращения: 13.06.2023).

2. Российская Федерация. Законы. Об информации, информационных технологиях и о защите информации. Статья 14 : Федеральный закон от 27.07.2006 № 149-ФЗ : с изменениями на 29 декабря 2022 года : редакция, действующая с 1 марта 2023 // Кодекс : профессиональная справочная система. – URL: https://docs.cntd.ru/document/901990051 (дата обращения: 13.06.2023).

3. ГОСТ 33707-2016 (ISO/IEC 2382:2015). Информационные технологии. Словарь = Information technologies. Vocabulary : Международный стандарт : издание официальное : утвержден и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 22 сентября 2016 г. № 1189-ст : введен впервые : дата введения 2017-09-01 / разработан Совместным техническим комитетом по стандартизации ISO/IEC JTC 1 "Информационные технологии" Международной организации по стандартизации (ISO) и Международной электротехнической комиссии (IEC) // Кодекс : профессиональная справочная система. – URL: https://docs.cntd.ru/document/1200139532 (дата обращения: 13.06.2023).

4. Программный комплекс "Информационное взаимодействие с ФБ МСЭ" // СоцИнформТех : [сайт]. – 2023 – URL: https://www.socit.ru/index.php/component/sppagebuilder/25-programmnyj-kompleks-informatsionnoe-vzaimodejstvie-s-fb-mse (дата обращения: 13.06.2023).

5. Программный комплекс "Ранняя помощь" // СоцИнформТех : [сайт]. – 2023 – URL: https://www.socit.ru/index.php/63-programmnyj-kompleks-

informirovanie-spetsialistov-sluzhby-rannej-pomoshchi-o-neobkhodimosti-otsenki-vyyavlennogo-sluchaya/ (дата обращения: 13.06.2023).

6. Material Design : [сайт]. – 2023. – URL: https://m3.material.io/ (дата обращения: 13.06.2023).

7. Fowler, M. Patterns of Enterprise Application Architecture / M. Fowler. – Boston : Addison-Wesley Professional, 2002. – 560 p. – ISBN 978-0-321-12742-6.

8. Java Documentation // Oracle : [документация]. – 2023. – URL: https://docs.oracle.com/en/java/ (дата обращения: 13.06.2023).

9. These Are the Top Languages for Enterprise Application Development And What That Means for Business // Cloud Foundry Foundation Top Languages for Enterprise Application Development. – 2018. – August – URL: https://www.cloudfoundry.org/wp-content/uploads/Developer-Language-Report_FINAL.pdf (дата обращения: 13.06.2023).

10. Spring Framework Documentation // Spring : [документация]. – 2023. – URL: https://docs.spring.io/spring-framework/reference/ (дата обращения: 13.06.2023).

11. Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools / L. Cosmina, R. Harrop, C. Schaefer, C. Ho. – New York : Apress, 2017 – 878 p. – ISBN 978-1-484-22807-4.

12. Developer Survey // Stackoverflow : [сайт]. – 2021. – URL: https://insights.stackoverflow.com/survey/2021 (дата обращения: 13.06.2023).

13. JavaScript // MDN Web Docs : [документация]. – 2023. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата обращения: 13.06.2023).

14. TypeScript Documentation // TS : [документация]. – 2023. – URL: https://www.typescriptlang.org/docs/ (дата обращения: 13.06.2023).

15. Angular : [документация]. – 2023. – URL: https://angular.io/docs (дата обращения: 13.06.2023).

16. Моргунов, Е. П. PostgreSQL. Основы языка SQL : учебное пособие / Е. П. Моргунов ; под ред. Е. В. Рогова, П. В. Лузанова. – Санкт-Петербург : БХВ-

Петербург, 2018 – 336 с. – URL: https://postgrespro.ru/education/books/sqlprimer (дата обращения: 13.06.2023).

17. PostgreSQL // PostgreSQL : [документация]. – 2023. – 25 May. – URL: https://www.postgresql.org/docs/ (дата обращения: 13.06.2023).

18. MVC // MDN Web Docs : [документация]. – 2023. – URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (дата обращения: 13.06.2023).

19. What is REST // REST API Tutorial. [сайт] – 2022. – 7 April. – URL: https://restfulapi.net/ (дата обращения: 13.06.2023).

20. Hibernate ORM Documentation - 6.2 // Hibernate : [сайт]. – 2023. – URL: https://hibernate.org/orm/documentation/6.2/ (дата обращения: 13.06.2023).

21. Garrett, J. J. Ajax: A New Approach to Web Applications / J. J. Garrett // Adaptive path. – 2005. – 18 February. – URL: https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf (дата обращения: 13.06.2023).

22. Myers, G. J. The Art of Software Testing / G. J. Myers, C. Sandler, T. Badgett. – New York : Wiley, 2011 – 256 p. – ISBN 978-1-118-03196-4.

23. Gray Box Testing // Software Testing Fundamentals : [сайт]. – 2023. – URL: https://softwaretestingfundamentals.com/gray-box-testing/ (дата обращения: 13.06.2023).