Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра экономической информатики

|  |  |
| --- | --- |
|  | *К защите допустить*: |
|  | Руководитель курсового проекта  Ассистент кафедры ЭИ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.О. Кабак |

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

курсового проекта

на тему

**Программное средство для дистанционного управления опрыскивателем растений**

БГУИР КП 6-05-0611-01 25 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | А. В. Еднач |
| Руководитель |  | Н. О. Кабак |

Минск 2025

**Реферат**

БГУИР КР 6-05-0611-01 025 ПЗ

ТЕМА: Программное средство для управления обработкой растений / А.В. Еднач. – Минск : БГУИР, 2025, – п.з. – 92 с., рисунков – 37, источников – 17, приложений – 4.

*Ключевые слова:* автоматизация, управление дроном, обработка растений, клиент-сервер.

*Объект исследования:* управление устройством обработки растений.

*Предмет исследования:* программное средство для управления устройством обработки растений.

*Цель* *курсовой работы*: автоматизация расчёта состава смеси для обработки растений.

*Методология проведения работы*: в процессе разработки системы использованы методы анализа предметной области, классификации данных, функционального моделирования с применением нотации IDEF0, информационного моделирования с использованием IDEF1X, а также объектно-ориентированного моделирования с помощью UML-диаграмм. Реализованы программные модули и графический интерфейс.

*Результаты работы*: выполнена постановка задачи и определены методы ее решения; разработана функциональная модель системы с использованием IDEF0; построена информационная модель базы данных в нотации IDEF1X, соответствующая третьей нормальной форме; созданы UML-диаграммы для моделирования системы; реализованы серверная и клиентская части программного средства на языке C# с использованием технологий .Net, Unity; разработано руководство пользователя; проведено тестирование, подтвердившее соответствие системы функциональным требованиям. Программный продукт обеспечивает автоматизацию учёта препаратов, расчёта зон обработки, соотношение и количество смеси.

*Область применения результатов*: разработанное программное средство может быть применено в парниках, колхозах. Оно способствует упрощению расчёта смеси и снижению вероятности ошибки при расчёте и учёте дозы. Программное средство полностью отвечает функциональным требованиям.

# СОДЕРЖАНИЕ

[\_Toc199794083](#_Toc199794083)[Введение 7](#_Toc199794084)

[1 Описание системы дистанционного управления опрыскивателем растений 9](#_Toc199794085)

[1.1 Понятие дистанционного управления опрыскивателем растений 9](#_Toc199794086)

[1.2 Виды систем дистанционного управления опрыскивателем растений и аналогичные решения 9](#_Toc199794087)

[1.3 Вывод по разделу «Виды систем дистанционного управления опрыскивателем растений» 12](#_Toc199794088)

[2 Постановка задачи для разработки программного средства дистанционного управления опрыскивателем растений 13](#_Toc199794089)

[3 Функционально моделирование программного средства для дистанционного управления опрыскивателем 15](#_Toc199794090)

[4 Информационная модель программного средства для дистанционного управления опрыскивателем растений 21](#_Toc199794091)

[6 Описание алгоритмов, реализующих бизнес-логику серверной части программного средства для управления акциями инвестиционного фонда 24](#_Toc199794092)

[6.1 Схема жизненного цикла клиент-серверного взаимодействия 25](#_Toc199794093)

[6.2 Схема процесса авторизации пользователей 27](#_Toc199794094)

[6.3 Схема процесса расчёта инструкций лечения растения 29](#_Toc199794095)

[7 Моделирование программного средства для дистанционного управления опрыскивателем растений 31](#_Toc199794096)

[7.1 Моделирование функциональности системы с помощью диаграммы вариантов использования 32](#_Toc199794097)

[7.2 Описание структуры системы на основе диаграммы классов 34](#_Toc199794098)

[7.3 Анализ поведения системы с использованием диаграммы последовательностей 41](#_Toc199794099)

[7.4 Моделирование жизненного цикла объектов с помощью диаграммы состояний 42](#_Toc199794100)

[7.5 Моделирование физического размещения системы с помощью диаграммы развертывания 43](#_Toc199794101)

[7.6 Вывод по разделу «Моделирование программного средства для управления акциями инвестиционного фонда с использованием UML» 44](#_Toc199794102)

[8 Руководство пользователя 46](#_Toc199794103)

[8.1 Подготовка к эксплуатации 47](#_Toc199794104)

[8.2 Руководство для администратора 50](#_Toc199794105)

[8.3 Руководство для пользователя 51](#_Toc199794106)

[8.4 Вывод по разделу «Руководство пользователя» 52](#_Toc199794107)

[9 Результаты тестирования разработанной системы 54](#_Toc199794108)

[Заключение 58](#_Toc199794109)

[Список использованных источников 61](#_Toc199794110)

[Приложение А (обязательное) Отчет о проверке на заимствования в системе «Антиплагиат» 62](#_Toc199794111)

[Приложение Б (обязательное) Листинг алгоритмов, реализующих бизнес-логику 63](#_Toc199794112)

[Приложение В (обязательное) Листинг основных элементов программы 71](#_Toc199794113)

[Приложение Г (обязательное) Листинг скрипта генерации базы данных 91](#_Toc199794114)

# ВВЕДЕНИЕ

Современные информационные технологии кардинально изменили подход к агропромышленному производству, включая управление тепличными хозяйствами. Автоматизированные системы позволяют оптимизировать процессы лечения растений, обеспечивая оперативную диагностику заболеваний, точный расчёт дозировок и визуализацию зон обработки. Такие решения интегрируются с базами агрономических знаний, предоставляя операторам и агрономам инструменты для принятия научно обоснованных решений. Внедрение программных средств для подбора терапии растений становится критически важным шагом для повышения урожайности и экологической безопасности тепличных комплексов.

Управление здоровьем растений в условиях закрытого грунта является ключевым элементом деятельности современных агропредприятий. Этот процесс требует высокой точности, учёта множества факторов: видовых особенностей культур, типа патогена, фитосовместимости препаратов и экологических ограничений. В условиях развития точного земледелия автоматизация подбора лечебных схем позволяет сократить потери урожая, повысить эффективность обработок и обеспечить оперативное реагирование на фитосанитарные угрозы.

Актуальность темы курсового проекта обусловлена необходимостью внедрения интеллектуальных систем для лечения растений в теплицах. Традиционные подходы, основанные на ручных расчётах и универсальных рекомендациях, не способны учитывать специфику конкретного тепличного комплекса, динамику развития заболеваний и современные экологические требования. Автоматизированное программное средство обеспечивает:

- Персонализированный подбор схем лечения

- Точный расчёт дозировок с учётом концентрации активных веществ

Объектом исследования является процесс лечения растений в промышленных теплицах, включающий диагностику заболеваний, подбор препаратов, расчёт рабочих растворов и планирование обработок.

Предметом исследования выступает программное средство для автоматизации подбора терапии растений, разработанное с использованием современных информационных технологий.

Целью курсового проекта является создание программного комплекса для оптимизации лечебных мероприятий в тепличных хозяйствах, обеспечивающего повышение эффективности обработок и снижение расхода препаратов.

Для достижения цели поставлены задачи:

1. Провести анализ предметной области и существующих решений в области автоматизации тепличных хозяйств
2. Разработать реляционную модель данных для хранения параметров растений, заболеваний и препаратов
3. Спроектировать архитектуру клиент-серверной системы (Unity/.NET/PostgreSQL)
4. Реализовать алгоритмы:

- Подбора препаратов по двойной совместимости (растение-заболевание-препарат)

- Расчёта зон обработки методом кластеризации точек поражения

- Генерации рецептур рабочих растворов

1. Разработать интуитивный пользовательский интерфейс с пошаговым сценарием работы
2. Обеспечить интеграцию с геоинформационными системами тепличных комплексов
3. Провести тестирование на соответствие агротехнологическим требованиям

Внедрение разработанного программного средства позволит:

- Автоматизировать процесс подбора терапии растений;

- Снизить риски фитотоксичности;

- Повысить точность обработок;

- Упростить документирование лечебных мероприятий;

- Оптимизировать расход препаратов.

Курсовая работа выполнена самостоятельно, проверена в системе «Антиплагиат». Процент оригинальности составляет 89,23%. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников». Скриншот приведен в приложении А (рисунок А.1).

# 1 ОПИСАНИЕ СИСТЕМЫ ДИСТАНЦИОННОГО УПРАВЛЕНИЯ ОПРЫСКИВАТЕЛЕМ РАСТЕНИЙ

# 1.1 Понятие дистанционного управления опрыскивателем растений

Дистанционное управление опрыскивателем растений – процесс отправки инструкций по обработке растения на дрон с опрыскивателем. Для того, чтобы повысить точность операций и предотвратить гибель растения от неправильно подобранного лечения, в программу включен механизм хранения и обработки данных о растениях, заболеваниях и препаратах.

Основными составляющими дистанционного управления опрыскивателем растений являются:

1 Формирование баз данных растений, болезней, препаратов.

2 Формирование баз совместимости растений, болезней, препаратов.

3 Выбор необходимого лечения в соответствии с совместимостью растения с препаратами.

Смесь, которой обрабатывают растение определяется совокупностью нескольких факторов, таких как: вид растения, вид болезни, совместимость растения и лекарства, совместимость болезни и лекарства, ударная доза лекарства растения.

Дистанционное управление опрыскивателем растений играет ключевую роль в автоматизации обработки растений в парнике:

1 Упрощение работы – автоматизация значительно упростит работу работников сельскохозяйственных предприятий.

2 Соответствие нормативам .

3 Оптимизацию инвестиционных процессов – автоматизированные системы ускоряют анализ данных, проведение сделок и формирование отчетности.

В современных организациях для выполнения расчетов используются специализированные программные решения, которые позволяют автоматизировать процесс начислений, уменьшить вероятность ошибок, упростить бухгалтерский учет и обеспечить соответствие законодательным требованиям.

# 1.2 Виды систем дистанционного управления опрыскивателем растений и аналогичные решения

Системы дистанционного управления опрыскивателем растений можно классифицировать по уровню автоматизации, архитектуре управления и способу взаимодействия оператора с дроном. Выбор конкретного подхода зависит от требований к точности обработки, масштаба хозяйства и наличия коммуникационной инфраструктуры.

1. Ручное дистанционное управление. Оператор напрямую задаёт траекторию полёта и параметры распыления (скорость, расход смеси) с пульта или планшета. Такой режим подходит для небольших площадей или для “точечных” обработок, когда требуется мгновенная корректировка в реальном времени.
2. Полуавтономные системы. Полёт по заранее проложенному маршруту (GPS-треке), при этом оператор может вмешиваться и менять параметры на лету. Позволяет сочетать гибкость ручного управления с быстродействием автоматизированного.
3. Полностью автономные решения. Планирование полёта и дозировки смеси выполняется заранее на основании цифровой карты поля и атласов растений (привязка по геокоординатам), а дрон самостоятельно выполняет цикл «взлёт – распыление – посадка». Требует минимального участия оператора и обеспечивает стабильность повторяемости операций.
4. Облачные платформы с клиент–серверной архитектурой. Клиентская часть (мобильное или веб-приложение) служит для подготовки миссии, загрузки карт и данных о растениях; серверная часть хранит базы данных, рассчитывает оптимальные маршруты и обеспечивает логи процедур. Позволяет объединять в единое поле зрения данные с нескольких дронов и операторов.
5. Интегрированные системы “умного” хозяйства. Являются частью более крупной экосистемы: связываются с системами мониторинга почвы, погодными датчиками и системами полива для корреляции данных и адаптивного управления.

Ниже приведены примеры распространённых программно-аппаратных комплексов для дистанционного управления опрыскивателями.



Рисунок 1.2.1 – Логотип системы DroneDeploy

Платформа позволяет загружать спутниковые ортофотокарты, автоматически строить маршруты съёмки и распыления, а также анализировать результаты обработки на облачном сервере.



Рисунок 1.2.2 – Логотип программы DJI Ground Station Pro

Предоставляет инструменты для планирования полётов дронов DJI, включая задания по автоматической залётке, настройке параметров камеры и распылителя, а также передачу телеметрии в реальном времени.



**Рисунок 1.2.3** – Логотип **John Deere Operations Center**

Интегрированная платформа для управления техникой John Deere, в том числе навесными опрыскивателями, с возможностью объединения данных о посевах, урожайности и расходе средств защиты растений.

Основные функции современных систем дистанционного управления опрыскивателем:

1. Управление миссией – создание, редактирование и сохранение маршрутов обработки полей с учётом ширины захвата и зоны перекрытия.
2. Расчёт дозировки смеси – автоматический подбор концентрации препаратов на основании характеристик растения, типа обработки и заданной площади.
3. Мониторинг в реальном времени – отображение местоположения дрона, оставшегося запаса смеси и состояния аккумулятора.
4. Хранение и анализ данных – ведение базы профильных растений, препаратов, результатов обработок и отчётов для последующего аудита.
5. Интеграция с другими сервисами – передача данных в системы учёта и ERP-решения хозяйства.

# 1.3 Вывод по разделу «Виды систем дистанционного управления опрыскивателем растений»

Современные решения для дистанционного управления обработкой растений сочетают несколько уровней автоматизации: от ручного управления до полностью автономных облачных платформ. Благодаря клиент-серверной архитектуре и модульному подходу удаётся:

* Унифицировать работу разных операторов и дронов в едином интерфейсе;
* Повысить точность за счёт расчёта дозировки и оптимизации маршрутов;
* Сократить трудозатраты и риск ошибок при планировании и исполнении миссии;
* Обеспечить прозрачность учёта обработок и требования к нормативам СЗР.

Интеграция с системами “умного” хозяйства и ERP позволяет не только автоматизировать расчёт составов смесей, но и выстраивать комплексное управление агротехнологиями: от мониторинга почвы до оценки экономической эффективности обработок. Такая синергия компонентов обеспечивает повышение урожайности при снижении затрат и рисков.

# 2 Постановка задачи для разработки программного средства дистанционного управления опрыскивателем растений

Цель проекта — создать клиент-серверную систему, позволяющую оператору удалённо планировать и выполнять обработку сельскохозяйственных угодий с помощью опрыскивателя, автоматически рассчитывая рабочие зоны, дозировки препаратов и учитывая специфику растений, болезней и медикаментозных препаратов.

Основные задачи разработки:

1 Многопоточный сервер (C# /.NET), который обрабатывает TCP-запросы от множества Unity-клиентов одновременно, гарантируя низкую задержку и стабильность работы при параллельном взаимодействии нескольких операторов.

2 Unity-клиентское приложение (C# в Unity), обеспечивающее интуитивно понятный интерфейс для:

* аутентификации пользователя (администратора или обычного оператора),
* выбора участка (список растений, их заболеваний и доступных медикаментов),
* формирования маршрута опрыскивателя и задания параметров обработки (радий действия, дозировка),
* визуализации расчётных зон и объёмов жидкости,
* мониторинга состояния опрыскивателя в реальном времени.

3 Настройка взаимодействия с реляционной базой данных для хранения информации.

4 Реализация бизнес-логики на стороне сервера для обработки операций с акциями, управления пользователями, компаниями и генерации файловых отчетов.

5 Надёжное хранение и восстановление данных (PostgreSQL + Npgsql), включая:

* регулярное резервирование базы (например, Docker-контейнер с PostgreSQL),
* «требуемые» внешние ключи для обеспечения целостности (3NF — каждая таблица содержит атомарные поля, нет избыточных данных, все связи реализованы через внешние ключи).

6 Внедрение системы авторизации с разделением прав доступа для администраторов и пользователей.

В ходе разработки программного средства необходимо решить следующие задачи:

1. Реализовать серверную часть на C# с использованием платформы .NET 6 (сборка .NET 6.0), располагающуюся в среде JetBrains Rider. Сервер должен функционировать как многопоточный TCP‑сервер и обеспечивать одновременное обслуживание нескольких клиентских подключений.
2. Разработать клиентскую часть на движке Unity (язык C#), которая взаимодействует с сервером через TCP‑сокеты. Клиентское приложение должно предоставлять пользователю графический интерфейс, созданный средствами Unity UI (графический интерфейс Unity), и обеспечивать отправку команд на сервер и приём ответов.
3. Обеспечить хранение и управление данными в реляционной базе данных PostgreSQL. Подключение к базе данных осуществляется с помощью библиотеки Npgsql. В базе данных хранятся необходимые справочные данные, параметры опрыскивания, учётные записи пользователей и другая информация.
4. Определить протокол обмена командами между клиентом и сервером. Передача команд осуществляется в текстовом формате с использованием разделителей. Сервер и клиент должны обеспечивать ручную обработку (парсинг) таких текстовых сообщений и выполнение соответствующих операций.
5. Внедрить функциональность, предусмотренную предметной областью. К основным функциям системы относятся расчёт зон опрыскивания (например, вычисление площади или схемы обработки в соответствии с входными параметрами), обновление справочников (списков химикатов, параметров опрыскивания и прочих служебных данных) и отображение соответствующей информации в пользовательском интерфейсе клиента. Весь необходимый функционал реализуется в соответствии с имеющимся исходным кодом проекта.
6. Соблюсти остальные требования и замечания, предъявленные руководителем проекта. При разработке программного средства должны быть учтены все ранее указанные особенности используемых технологий и библиотек, а также замечания, представленные в исходном задании и документации (например, использование указанных сред разработки и инструментов, ручная обработка текстовых команд и т.д.).

Таким образом, в рамках данной курсовой работы предстоит разработать полнофункциональный клиент‑серверный программный комплекс, удовлетворяющий указанным требованиям и обеспечивающий надёжное дистанционное управление опрыскивателем растений с соответствующим пользовательским интерфейсом и базой данных.

# 3 Функционально моделирование программного средства для дистанционного управления опрыскивателем

Современные информационные технологии играют ключевую роль в аграрной сфере, проникая во все аспекты управления полевыми работами. В условиях стремительного развития «умного» земледелия фермерские хозяйства и агропредприятия всё чаще сталкиваются с необходимостью автоматизации процессов обработки полей. Однако многие хозяйства, особенно небольшие, продолжают полагаться на устаревшие подходы, такие как ручное планирование маршрутов и использование разрозненных инструментов для учёта операций, что приводит к низкой эффективности, ошибкам и отсутствию оперативного доступа к данным о состоянии полей и технике.

Дистанционное управление опрыскивателем растений — это комплексный процесс, который включает получение от оператора параметров обработки участка, расчёт оптимального маршрута и зон опрыскивания, контроль над выполнением операций, сбор телеметрических данных от техники и предоставление аналитической информации о расходе жидкости и охваченных площадях. Основные задачи системы управления опрыскивателем включают проверку корректности входных команд, расчёт рабочих зон с учётом плотности насаждений и рельефа, построение маршрутов с учётом радиуса обработки и объёма жидкости, а также формирование отчётов для оценки эффективности и экономии ресурсов. Без автоматизации эти процессы становятся трудоёмкими, повышают риск ошибок и затрудняют своевременное получение данных о проведённых работах.

В современных условиях ручной подход к управлению опрыскивателем становится всё менее приемлемым. Развитие технологий позволяет агропредприятиям автоматизировать ключевые этапы обработки полей, что ускоряет операции, повышает точность внесения средств защиты растений и обеспечивает доступность аналитики для операторов и агрономов. Автоматизированные системы дистанционного управления опрыскивателем делают процесс обработки полей более прозрачным, эффективным и удобным для всех участников агротехнологического цикла.

Ниже представлен процесс определения лечения растения IDEF0 [9]. Данная модель описывает процесс до выведения рассчитанных инструкций пользователю.

Процесс составления инструкций к опрыскивателю представлен в виде 4 уровней декомпозиции. Верхний уровень включает один блок: «Выбрать лечение растения».

Второй уровень представляет собой декомпозицию блоков «Подобрать список лекарственных препаратов», «Определить препарат», «Рассчитать область обработки». Декомпозиция блока «Подобрать список лекарственных препаратов» содержит три блока: «Произвести выборку болезней по растениям», «Произвести выборку лекарств по болезням», «Произвести выборку лекарств по растениям». Декомпозиция блока «Определить препарат» содержит два блока: «Выбрать необходимое лекарство», «Выбрать дозировку». Декомпозиция блока «Рассчитать область обработки» содержит три блока: «Перевести координаты поражения», «Рассчитать радиус обработки», «Рассчитать использование препарата.

Третий уровень содержит декомпозицию блока «Рассчитать использование препарата», которая содержит два блока: «Рассчитать площади обработки» , «Рассчитать литраж по ударной дозе».

Четвёртый уровень содержит декомпозицию блока «Рассчитать литраж по ударной дозе», которая содержит два блока: «Сложить все площади» , «Рассчитать литраж по общему объёму площади и ударной дозе».

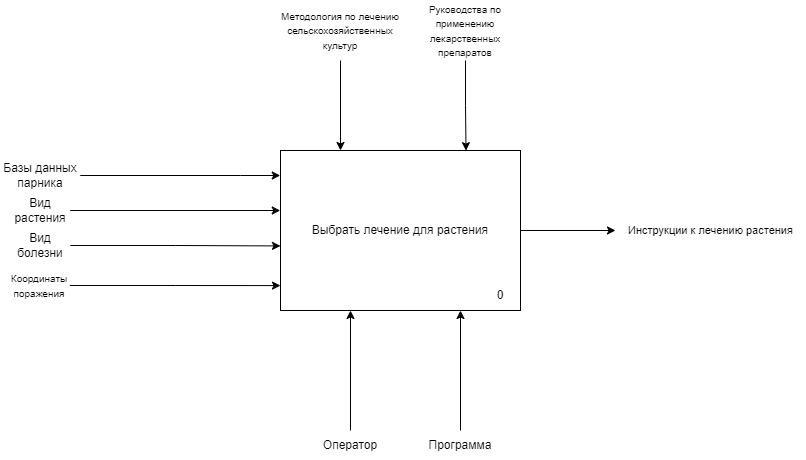


Рисунок 3.1 – Верхний уровень модели

В данной модели процесс составления инструкций к лечению растения. Управляющими факторами выступают методология по лечению сельскохозяйственных культур и руководства по применению лекарственных препаратов. Эти элементы влияют на процесс, но не являются его частью. Входными данными для процесса служат базы данных парника, вид растения, вид болезни его поразившей и координаты поражения. На выходе процесса формируется готовая инструкция по обработке растения, включающая зоны поражения, количество смеси, её концентрацию, данные о препарате, растении и болезни.

К механизмам, обеспечивающим выполнение процесса, относятся клиентское приложение (Unity), серверная часть (.Net), база данных (PostgreSQL). Клиентское приложение предоставляет интерфейс для взаимодействия с пользователем, сервер обрабатывает запросы и выполняет бизнес-логику, база данных хранит информацию о лекарствах, болезнях препаратах и их совместимостях.

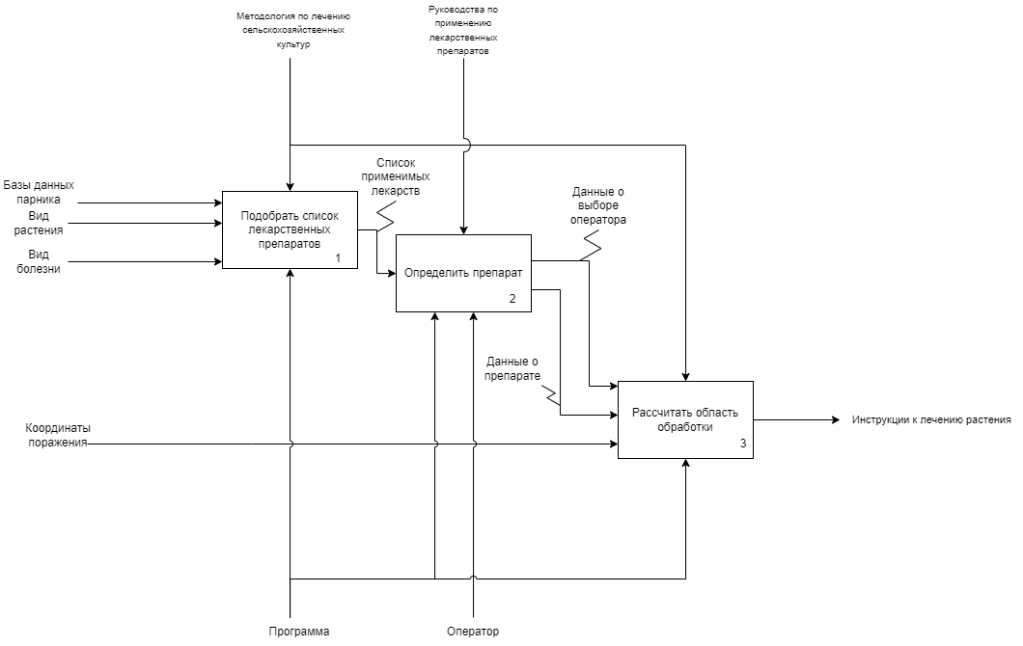


Рисунок 3.2 – Декомпозиция процесса «Выбрать лечение растения»

Процесс «Выбрать лечение для растения» разделен на три ключевых подпроцесса, каждый из которых выполняет свою специфическую задачу в рамках общей операции.

Первый подпроцесс – подобрать список лекарственных препаратов. На этом этапе система сортирует данные о лекарствах в соответствии с растением и болезнью.

Второй подпроцесс – определить препарат. На данном этапе оператор выбирает препарат, который будет использован для лечения растения, а также дозировку.

Третий подпроцесс – рассчитать область обработки. На данном этапе программа рассчитывает область обработки, а также итоговый литраж смеси для обработки.

На выходе процесса формируются инструкции к лечению растения.

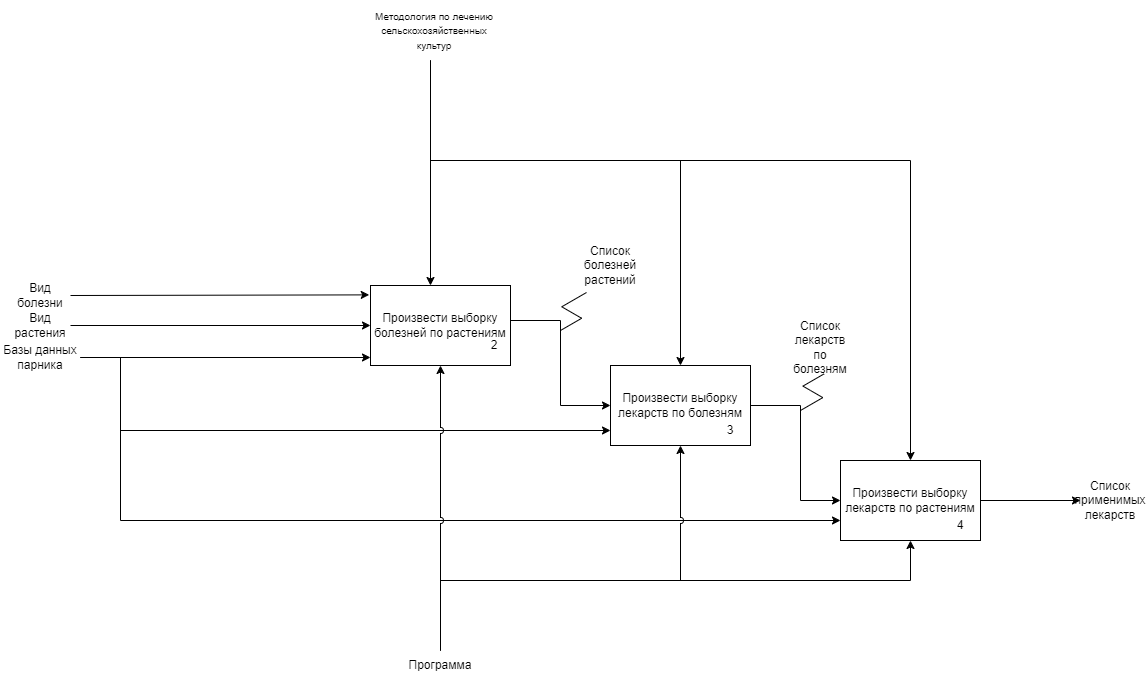


Рисунок 3.3 – Декомпозиция процесса «Подобрать список лекарственных препаратов»

Подпроцесс «Подобрать список лекарственных препаратов» включает три этапа, направленных на выборку подходящих лекарств из базы данных.

Первый этап – произвести выборку болезней по растениям. Возвращает список болезней, подходящих к растению.

Второй этап – произвести выборку лекарств по болезням. Возвращает список лекарств по болезням.

Третий этап – произвести выборку лекарств по растениям. Возвращает список лекарств по растениям.

Итогом выполнения подпроцесса становится список применимых к растению лекарств для лечения данной болезни.

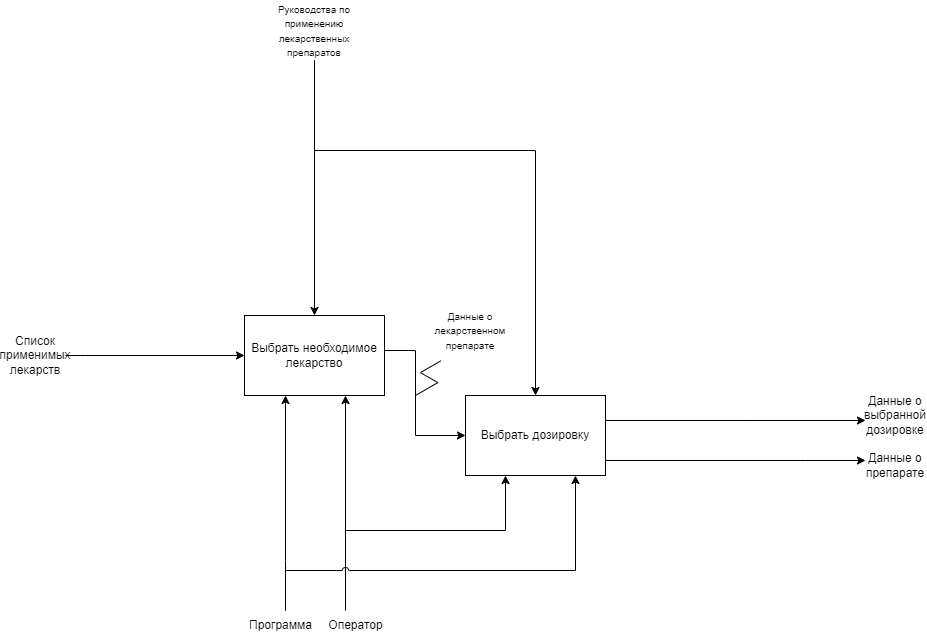


Рисунок 3.4 – Декомпозиция процесса «Определить препарат»

Процесс определения препарата состоит из двух этапов, позволяющих оператору выбрать из списка подходящий препарат для лечения и его дозировку.

Первым этапом является выбор препарата из списка подходящих.

Вторым этапом является выбор дозировки для рассчёта смеси препарата.

Итогом становится данные о препарате и выбранной дозировке.

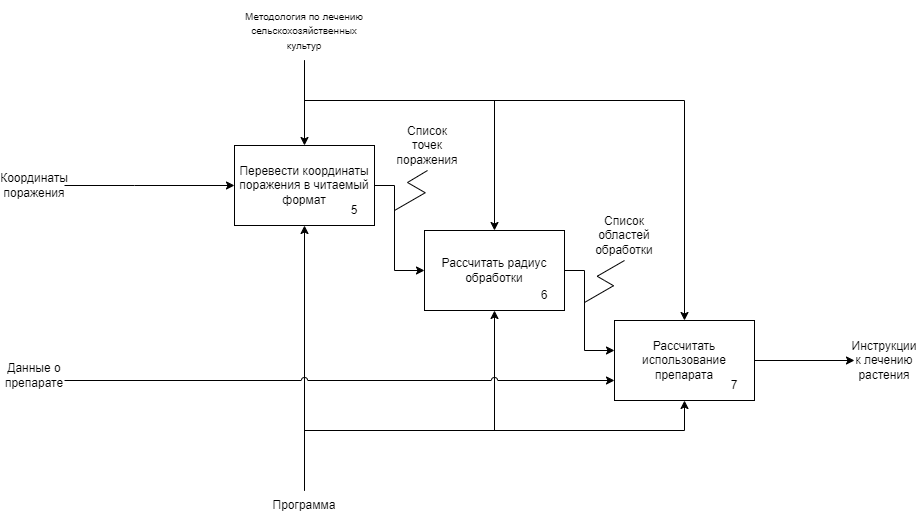


Рисунок 3.5 – Декомпозиция процесса «Рассчитать область обработки»

Процесс рассчёта области обработки рассчитывает зоны обработки и литраж смеси для лечения.

Механизмы поддержки: Операции выполняются программой сервера, обеспечивающей обработку и получение данных из базы данных.

В результате формируются инструкции к лечению растения.

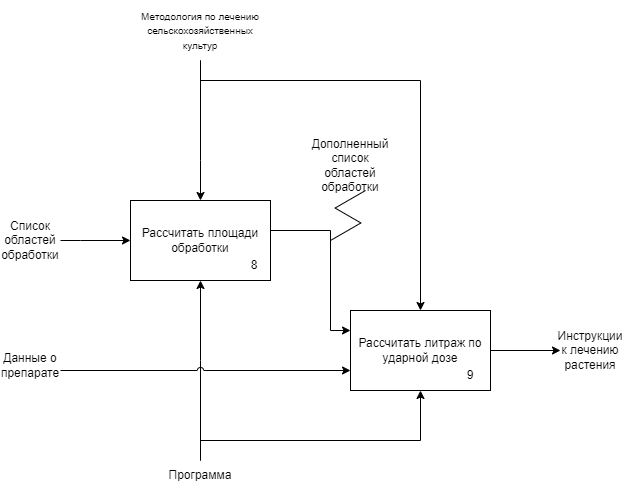


Рисунок 3.6 – Декомпозиция процесса «Рассчитать использование препарата»

Процесс расчёта использования препарата состоит из двух этапов:

1 Рассчитать площади обработки.

2 Рассчитать литраж по ударной дозе.

В результате сформированы инструкции к лечению растения, включающие области обработки и данные о препарате.

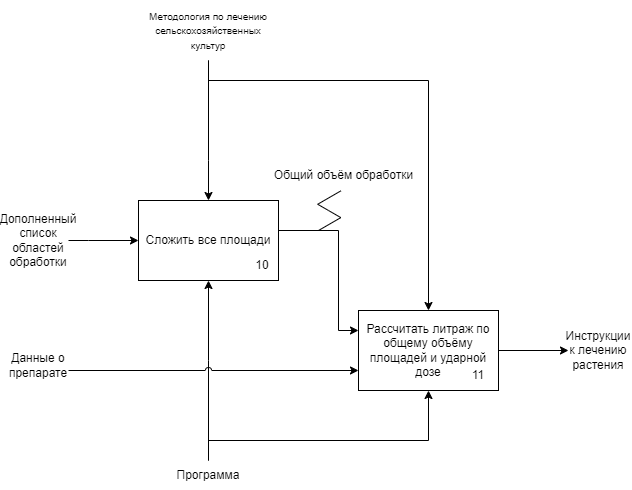


Рисунок 3.7 – Декомпозиция процесса «Рассчитать литраж по ударной дозе»

Процесс расчёта литража препарата состоит из двух этапов:

1 Сложить все площади.

2 Рассчитать литраж по общему объёму площадей и ударной дозе.

В результате сформированы инструкции к лечению растения, включающие области обработки и данные о препарате.

Все описанные процессы, от подбора списка лекарственных препаратов до расчёта областей обработки, образуют последовательную систему этапов, обеспечивающую простоту, надежность, эффективность составления инструкций лечения. Каждый этап выполняет свою специфическую задачу, но их взаимосвязь и строгая последовательность играют решающую роль в определении лечения растения.

# 4 ИНФОРМАЦИОННАЯ МОДЕЛЬ ПРОГРАММНОГО СРЕДСТВА ДЛЯ ДИСТАНЦИОННОГО УПРАВЛЕНИЯ ОПРЫСКИВАТЕЛЕМ РАСТЕНИЙ

Информационная модель IDEF1X, разработанная для программного средства управления опрыскивателем растений, представляет собой реляционную схему базы данных, обеспечивающую структурированное хранение и обработку данных о растениях, болезнях, препаратах, зонах опрыскивания и пользователей [10]. Она состоит из взаимосвязанных сущностей, объединённых первичными и внешними ключами, что позволяет эффективно управлять справочниками и поддерживать целостность операций при расчёте зон и маршрутов опрыскивателя. Ниже приведена информация о ключевых сущностях, их атрибутах и связях, а также подтверждение соответствия модели третьей нормальной форме (3NF), демонстрирующее отсутствие избыточности и обеспечение надёжности данных.

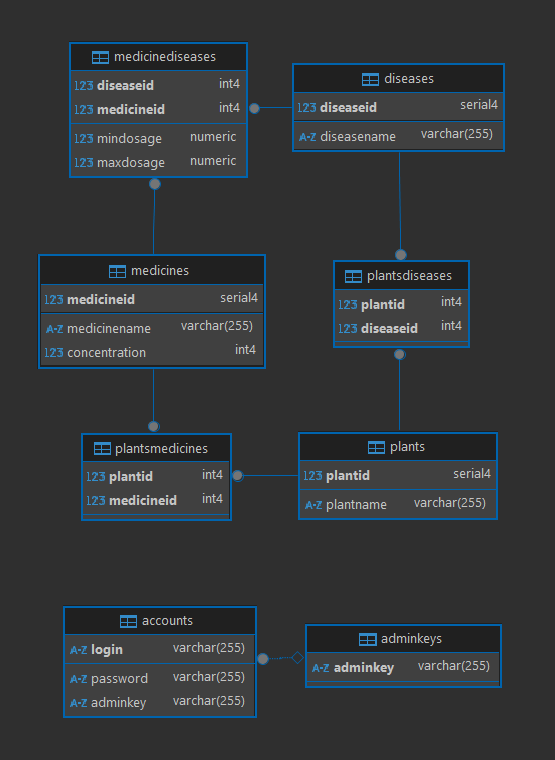


Рисунок 4.1 – Информационная модель IDEF1X

Сущность «AdminKeys» определяет ключи администраторов. Каждый ключ уникален и принадлежит лишь одному администратору. Атрибут adminKey хранит значение ключа, которое позволяет распознать администратора в системе.

Сущность «Accounts» содержит персональные данные людей, зарегистрированных в системе. Атрибуты включают login, password и adminKey (ключ администратора), предоставляя информацию об аккаунтах пользователей

Сущность «Plants» хранит информацию о растениях, на предприятии. Каждое растение идентифицируется первичным ключом plantId. Атрибут plantName содержит название растения.

Сущность «Medicines» хранит информацию о препаратах, на предприятии. Каждое растение идентифицируется первичным ключом medicineId. Атрибут medicineName содержит название растения.

Сущность «Diseases» описывает существующие болезни растений. Каждая болезнь идентифицируется первичным ключом diseaseId. Атрибуты включают disiaseName содержит название болезни.

Сущность «PlantsDiseases» является связующей таблицей, а также хранит информацию о совместимости растений и болезней. Уникальность записи обеспечивается составным первичным ключом (plantId, diseaseId), которые ссылаются на таблицы «Plants» и «Diseasses».

Сущность «PlantsMedicines» является связующей таблицей, а также хранит информацию о совместимости растений и препаратов. Уникальность записи обеспечивается составным первичным ключом (plantId, medicineId), которые ссылаются на таблицы «Plants» и «Medicines».

Сущность «MedicineDisease» является связующей таблицей, описывающей отношение между препаратами и болезнями. Она имеет составной первичный ключ (diseaseId, medicineId), которые ссылаются на таблицы «Diseases» и «Medicines». Аттрибуты у этой сущности определяют пороги минимальной и максимальной дозировок препарата на литр жидкости.

Модель устанавливает логические связи между сущностями, обеспечивая целостность данных и возможность выполнения запросов для управления транзакциями.

Связь между «Plants» и «PlantsDiseases» – один-ко-многим: одно растение может болеть разными болезнями, что реализовано через атрибут plantId в таблице «PlantsDiseases».

Связь между «Diseases» и «PlantsDiseases» – один-ко-многим: одна болезнь может встречаться у разных растений, что реализовано через атрибут diseaseId в таблице «PlantsDiseases».

Связь между «Plants» и «PlantsMedicines» – один-ко-многим: одно растение может обрабатываться разными препаратами, что реализовано через атрибут plantId в таблице «PlantsMedicines».

Связь между «Medicines» и «PlantsMedicines» – один-ко-многим: один препарат может применяться к разным растениям, что реализовано через атрибут medicineId в таблице «PlantsMedicines».

Связь между «Medicines» и «MedicinesDiseases» – один-ко-многим: одно растение может обрабатываться разными препаратами, что реализовано через атрибут medicineId в таблице «MedicinesDiseases».

Связь между «Diseases» и «MedicinesDiseases» – один-ко-многим: один препарат может применяться к разным растениям, что реализовано через атрибут diseaseId в таблице «MedicinesDiseases».

Модель соответствует первой нормальной форме (1NF), так как все атрибуты атомарны.

Для второй нормальной формы (2NF) неключевые атрибуты должны зависеть от всего первичного ключа. В например таблицах с простыми ключами (Plants, Medicines, Diseases), plantName зависит только от plantId. В таблицах с составными ключами (MedicineDiseases) Дозировки (minDosage, maxDosage) зависят от комбинации (medicineId, diseaseId) , что также удовлетворяет 2NF.

Для третьей нормальной формы (3NF) неключевые атрибуты не должны иметь транзитивных зависимостей. В «Medicines» concentration зависит только от medicineId. В «MedicineDiseases» Дозировки зависят только от пары (medicineId, diseaseId). В «Accounts» password и adminKey зависят только от login. Таким образом, модель полностью соответствует 3NF.

Информационная модель IDEF1X для системы дистанционного управления опрыскивателем растений представляет собой логически выстроенную базу данных, соответствующую третьей нормальной форме, что исключает избыточность данных и транзитивные зависимости. Она обеспечивает все необходимые данные и связи для выполнения операций с акциями, гарантируя точность и производительность. Благодаря строгой структуре и соблюдению нормальных форм, данная модель является эффективным решением для автоматизации управления акциями в инвестиционных фондах.

# 5 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОГРАММНОГО СРЕДСТВА ДЛЯ УПРАВЛЕНИЯ АКЦИЯМИ ИНВЕСТИЦИОННОГО ФОНДА

Разработка системы дистанционного управления опрыскивателем растений требует чёткого проектирования и структурированного описания бизнес-логики. В этом процессе схемы алгоритмов служат важнейшим инструментом, позволяя наглядно представить последовательность операций, необходимых для выполнения ключевых функций системы. Они помогают визуализировать процессы обработки данных, выявлять логические зависимости и оптимизировать алгоритмы, что снижает количество ошибок и облегчает дальнейшее сопровождение проекта. Особенно актуально это для серверной части приложения, где сосредоточена основная логика распределения зон обработки, расчёта дозировки и управления коммуникацией с клиентами: прозрачность этих алгоритмов критична для надёжного функционирования системы.

Предметная область дистанционного управления опрыскивателем связана с динамичной обработкой пространственных данных и последовательных команд. Необходимо учитывать координаты точек обработки, радиусы зон для распыления, потребление жидкости и состояние самого оборудования в реальном времени. Любая неточность может привести к перекрытию областей или недостаточной обработке растений, что снижает эффективность работы. Автоматизация этих процессов с помощью программного средства позволяет гарантировать точность расчётов, минимизировать простой техники и обеспечить своевременную реакцию на изменения внешних условий, таких как ветер или состояние баков с рабочим раствором.

Серверная часть системы отвечает за приём TCP-запросов от Unity-клиентов, управление сессиями операторов, выполнение расчётов зон распыления и отправку команд на клиентские устройства. Логика обработки включает проверку корректности входных данных— координат точек, параметров радиуса и дозировки— обновление состояния оборудования, расчёт объединённых зон с учётом возможного перекрытия кругов, а также учёт уровня заполнения бака и оставшегося рабочего ресурса. Эти процессы должны быть надёжными и масштабируемыми, поскольку одновременно несколько операторов могут работать с разными участками. Схемы алгоритмов помогают формализовать каждый этап: от получения набора точек на условной карте до формирования итоговых зон, объединения пересекающихся кругов и вычисления необходимого объёма жидкости, включая обработку исключительных ситуаций, например, когда батарея опрыскивателя разряжена или связь с клиентом прерывается.

Визуализация алгоритмов обеспечивает быструю оценку работы системы без обращения к исходному коду. Это важно при согласовании требований между разработчиками и заказчиком, поскольку наглядные блок-схемы позволяют сразу выявить узкие места— например, неэффективные проверки на стороне сервера или избыточные вычисления при объединении зон. Кроме того, такие схемы становятся актуальной документацией на всём протяжении жизненного цикла системы, облегчая доработку и поддержку. В контексте агротехники, где точность нанесения рабочего раствора и поддержание бесперебойной связи являются критически важными, схемы алгоритмов позволяют зафиксировать каждое действие и обеспечить возможность аудита операций.

В этом разделе представлены схемы алгоритмов, реализующих основные функции серверной части: от авторизации оператора и приёма координатных данных до расчёта зон распыления и генерации команд для визуализации в Unity.

Таким образом, использование схем алгоритмов при разработке системы дистанционного управления опрыскивателем растений является ключевым инструментом для достижения высокого качества решения. Они задают структурированный каркас для реализации бизнес-логики, упрощают взаимодействие между участниками проекта и создают основу для дальнейшего тестирования и оптимизации. Далее приведено детальное описание алгоритмов, лежащих в основе серверной части, подчёркивающее их роль в обеспечении эффективного, точного и безопасного управления опрыскивателем

# 5.1 Схема жизненного цикла клиент-серверного взаимодействия

В разработке программного средства для дистанционного управления опрыскивателем растений ключевое значение имеет четкое определение процессов, обеспечивающих взаимодействие между клиентской и серверной частями системы. Схема жизненного цикла клиент-серверного взаимодействия описывает последовательность операций, начиная от инициализации системы и заканчивая ее завершением, включая этапы авторизации и обработки запросов. Такая схема необходима для структурирования бизнес-логики серверной части, взаимодействие с базой данных и предоставление данных клиентам.



Рисунок 5.1 Схема жизненного цикла клиент-серверного взаимодействия

Схема иллюстрирует жизненный цикл клиент-серверного взаимодействия в системе дистанционного управления опрыскивателем растений. Рассмотрим этапы кратко:

1 Начало. Точка старта процесса.

2 Запуск сервера. Серверное приложение активируется и начинает прослушивать сеть для принятия подключений.

3 Запуск клиента. Клиентское приложение запускается на устройстве пользователя и инициирует подключение к серверу.

4 Авторизация в системе. Клиент отправляет учетные данные (логин, пароль), сервер их проверяет и предоставляет доступ при успехе.

5 Главный цикл, работает пока клиент не отключится.

6 Получение запроса. Сервер принимает запрос от клиента (например, запрос данных или команды).

7 Обработка запроса. Сервер выполняет обработку запроса (обращение к базе данных, вычисления и т.д.).

8 Отправка ответа: Сервер возвращает клиенту результат обработки.

9 Конец при отключении клиента.

# 5.2 Схема процесса авторизации пользователей



Рисунок 5.2 – Схема процесса авторизации пользователей

Схема иллюстрирует процесс авторизации и регистрации пользователя в клиент-серверной системе управления акциями инвестиционного фонда. Рассмотрим этапы кратко:

1. Начало. Точка старта процесса.
2. Вывод панели авторизации. Клиентское приложение отображает интерфейс для ввода учетных данных.
3. Пользователь решил зарегистрироваться. Пользователь выбирает между регистрацией и авторизацией:

3.1 Ввод логина и пароля. Пользователь вводит свои учетные данные.

3.2 Отправка запроса на авторизацию. Клиент отправляет данные на сервер для проверки.

3.3 Пользователь существует. Сервер проверяет наличие пользователя в системе:

3.3.1 Проверка прав пользователя. Сервер определяет права доступа пользователя.

3.3.2 Роль пользователя. Определение роли:

– User. Отображение меню пользователя;

–Admin. Отображение меню администратора.

3.3.3 Проверка прав пользователя. Сервер определяет права доступа пользователя.

3.3.4 Вывод ошибки. Клиент отображает сообщение об ошибке.

3.3.5 Ввод логина, пароля и ключа администратора. Пользователь вводит данные для создания учетной записи.

3.3.6 Отправка запроса на регистрацию. Клиент отправляет данные на сервер.

3.3.7 Данные введены корректно?. Сервер проверяет правильность введенных данных:

3.3.8 Регистрация пользователя. Сервер создает нового пользователя.

* + 1. Вывод ошибки. Клиент отображает сообщение об ошибке.

4 Конец. Завершение процесса.

# 5.3 Схема процесса расчёта инструкций лечения растения

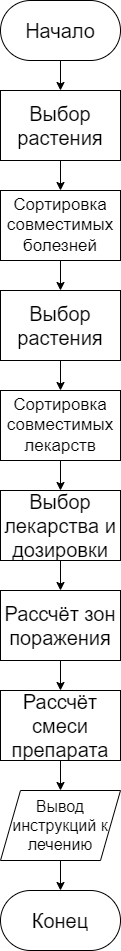


Рисунок 5.3 – Схема расчёта процесса лечения растения

Схема иллюстрирует процесс расчета лечения растения в системе дистанционного управления опрыскивателем растений. Рассмотрим этапы кратко:

1. Начало. Точка старта процесса.
2. Выбор растения. Пользователь выбирает растение для обработки в клиентском интерфейсе.
3. Сортировка совместимых болезней. Система анализирует и отбирает возможные болезни, характерные для выбранного растения.
4. Выбор растения. Подтверждение выбора растения (возможно, уточнение или корректировка).
5. Сортировка совместимых лекарств. Система определяет подходящие препараты для лечения выявленных болезней.
6. Выбор лекарства и дозировки. Пользователь или система выбирает конкретное лекарство и его дозировку.
7. Расчет зон поражения. Система определяет области растения, требующие обработки, на основе анализа.
8. Расчет смеси препарата. Система рассчитывает состав и пропорции смеси для опрыскивания.
9. Вывод инструкций к лечению. Клиенту предоставляются инструкции по применению препарата.
10. Конец. Завершение процесса.

# 6 МОДЕЛИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА ДЛЯ ДИСТАНЦИОННОГО УПРАВЛЕНИЯ ОПРЫСКИВАТЕЛЕМ РАСТЕНИЙ

Современные системы дистанционного управления опрыскивателем растений требуют эффективных и надёжных программных решений для управления справочниками растений, болезней, препаратов, а также для обеспечения взаимодействия между различными участниками процесса опрыскивания. Разработка такого программного средства является сложной задачей, которая требует тщательного проектирования, анализа и моделирования на всех этапах жизненного цикла системы. Для достижения этой цели широко применяются методы объектно-ориентированного анализа и проектирования, в частности, использование унифицированного языка моделирования UML (Unified Modeling Language). UML предоставляет мощный набор инструментов для визуализации, спецификации, конструирования и документирования программных систем, что позволяет разработчикам и аналитикам создавать чёткие и понятные модели, отражающие как функциональные, так и структурные аспекты системы [16].

В данной главе рассматривается процесс моделирования программного средства, предназначенного для дистанционного управления опрыскивателем растений. Основное внимание уделяется применению различных типов диаграмм UML, которые позволяют описать систему с разных точек зрения: функциональной, структурной, поведенческой, архитектурной и физической. Такой комплексный подход обеспечивает глубокое понимание системы, её компонентов, взаимодействия между ними, а также её поведения в различных сценариях использования. Моделирование с использованием UML не только упрощает процесс разработки, но и способствует улучшению качества программного продукта за счёт раннего выявления потенциальных проблем, таких как несогласованность требований, избыточная сложность структуры или неоптимальная архитектура [17].

Целью данной главы является создание всестороннего описания системы дистанционного управления опрыскивателем растений, начиная с её функциональных возможностей и заканчивая физическим размещением компонентов на аппаратных узлах. Каждый из разделов будет сопровождаться подробным описанием и анализом. Такой подход позволяет не только формализовать процесс проектирования, но и обеспечить согласованность между архитектурой системы и её конечной реализацией. Кроме того, использование UML-диаграмм способствует улучшению коммуникации между членами команды разработчиков, аналитиками и другими заинтересованными сторонами, что особенно важно в сложных проектах, таких как разработка программного средства для опрыскивателя растений.

Важно отметить, что система, рассматриваемая в данной главе, ориентирована на клиент-серверную архитектуру, где сервер отвечает за обработку запросов, управление данными и взаимодействие с базой данных, а клиент предоставляет интерфейс для взаимодействия оператора с системой. В рамках моделирования будут учтены такие аспекты, как безопасность (например, авторизация пользователей), производительность (оптимизация взаимодействия с базой данных) и масштабируемость (возможность обработки множества запросов от Unity-клиентов). Особое внимание будет уделено специфике предметной области – управлению справочниками растений, болезней и препаратов, что требует точного учёта данных о растениях, их заболеваниях, доступных химических составах и ролях пользователей (например, администратор, оператор).

В результате работы будет создана комплексная модель программного средства, которая может быть использована как основа для дальнейшей разработки, тестирования и внедрения системы. Модели UML, представленные в этой главе, помогут не только разработчикам, но и аналитикам, тестировщикам и менеджерам проекта, предоставляя единое визуальное представление системы на всех уровнях её проектирования. Далее в главе каждый из перечисленных аспектов будет рассмотрен подробно, начиная с моделирования функциональности системы с помощью диаграммы вариантов использования.

# 6.1 Моделирование функциональности системы с помощью диаграммы вариантов использования

Диаграмма вариантов использования (Use Case) является одним из ключевых инструментов UML, предназначенных для описания функциональности системы с точки зрения взаимодействия пользователей с ней. Она позволяет определить основные роли (актеров), которые взаимодействуют с системой, а также сценарии использования (варианты использования), которые отражают, какие действия могут выполнять пользователи для достижения своих целей. Диаграмма помогает структурировать требования к системе, выявить основные сценарии использования и обеспечить их согласованность с ожиданиями пользователей. В данном разделе будет представлена диаграмма вариантов использования, описывающая функциональные возможности системы, а также проведен ее подробный анализ с акцентом на роли пользователей и их взаимодействие с системой.

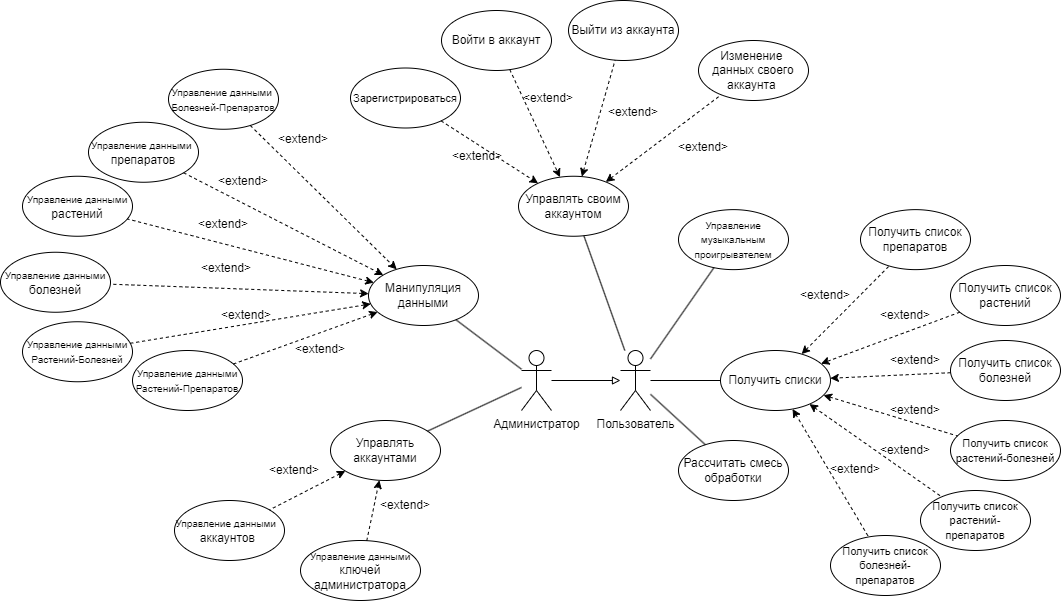


Рисунок 6.1 – Диаграмма вариантов использования системы дистанционного управления опрыскивателем.

Рассмотрим диаграмму более подробно. Начнем с актеров, присутствующих на диаграмме:

1 Администратор. Этот актёр представляет пользователя с расширенными правами, отвечающего за поддержку справочников и пользователей в системе дистанционного управления опрыскивателем растений. Администратор управляет данными об аккаунтах, ключах администратора и всеми справочниками (растений, болезней, препаратов и их связей).

2 Пользователь. Этот актёр представляет оператора, который использует клиентское приложение Unity для выбора участка, формирования маршрута опрыскивателя и расчёта смеси обработки. Пользователь имеет ограниченные права: он может авторизоваться, просматривать справочники, задавать параметры обрабатываемого участка и получать результат расчёта смеси.

Диаграмма содержит два набора вариантов использования: те, что доступны Администратору, и те, что доступны Пользователю и Администратору.

Администратор имеет доступ к следующим вариантам использования:

1 Управлять аккаунтами. Этот сценарий включает действия по созданию, чтению, обновлению и удалению учётных записей пользователей. Администратор может добавить нового оператора, изменить данные существующего (логин, роль) или удалить его из системы.

2 Управление данными ключей администратора. Подсценарий управления аккаунтами: позволяет создавать и удалять дополнительные ключи, дающие доступ к административным функциям.

3 Манипуляция данными (CRUD). Этот общий сценарий включает создание, чтение, обновление и удаление записей во всех справочниках системы. Относится к следующим под-сценариям:

3.1 Управление данными растений;

3.2 Управление данными болезней;

3.3 Управление данными препаратов;

3.4 Управление данными «растений–болезней»;

3.5 Управление данными «растений–препаратов».

4 Получить списки. Этот сценарий позволяет загрузить из базы все справочники в виде одной выборки. Посредством отношения «include» он включает:

4.1 Получить список препаратов;

4.2 Получить список растений;

4.3 Получить список болезней;

4.4 Получить список растений–болезней;

4.5 Получить список растений–препаратов;

4.6 Получить список болезней–препаратов.

5 Рассчитать смесь обработки. Этот сценарий позволяет собрать из справочников необходимые данные (растения, болезни, препараты, их совместимости), задать дозировку, а затем выполнить расчёт зоны распыления и объёма жидкости для каждой зоны.

Пользователь, а также администратор имеет доступ к следующим вариантам использования:

1 Зарегистрироваться. Позволяет создать новую учётную запись оператора;

2 Войти в аккаунт. Сценарий аутентификации: пользователь вводит логин и пароль, клиент отправляет запрос на сервер, сервер проверяет данные;

3 Управлять своим аккаунтом. После входа в систему посетитель может просматривать и изменять свои личные данные (логин, пароль). Этот сценарий включает через отношение «extend»:

3.1 Изменение данных своего аккаунта;

3.2 Выйти из аккаунта.

4 Получить списки. Позволяет получить все справочники для выбора участка и препаратов. Включает:

4.1 Получить список препаратов;

4.2 Получить список растений;

4.3 Получить список болезней;

4.4 Получить список растений–болезней;

4.5 Получить список растений–препаратов;

* 1. Получить список болезней–препаратов.

5 Рассчитать смесь обработки. После выбора болезни и препарата пользователь формирует точки обработки, указывает дозировку и радиус, а система вычисляет зоны опрыскивания и объёмы жидкости.

# 6.2 Описание структуры системы на основе диаграммы классов

Диаграмма классов является ключевым инструментом в объектно-ориентированном проектировании, позволяющим визуализировать структуру системы, её компоненты, их взаимодействие и зависимости. Она предоставляет разработчикам и архитекторам чёткое представление о классах, их атрибутах, методах и связях, что упрощает процесс проектирования, реализации и сопровождения системы. В данном разделе представлены диаграммы классов клиента и сервера, которые описывают архитектуру системы дистанционного управления опрыскивателем растений. Диаграмма клиента фокусируется на пользовательском интерфейсе Unity и логике взаимодействия с сервером, тогда как диаграмма сервера (будет представлена далее) раскроет серверную логику обработки TCP-запросов и управления данными. Эти диаграммы помогают понять, как система разделяет ответственность между клиентом и сервером, обеспечивая модульность и масштабируемость..

На рисунке 6.2 изображена диаграмма классов клиентской части приложения, которая включает в себя классы реализующие процесс взаимодействия пользователя с системой дистанционного управления опрыскивателем. Среди них присутствуют:

– сетевые компоненты (SimpleTCPClient, UIQuerySender) для установления и управления TCP-соединением с сервером;

– контроллеры панелей (AuthorisationPanel, ControlPanel, DosagePanel, AdminPanel и другие), отвечающие за логику отображения и взаимодействие с пользователем через Unity UI;.

– сущности данных (Plant, Disease, Medicine, PlantDisease, PlantMedicine, MedicineDisease, Account и т.д.), которые моделируют информацию о растениях, заболеваниях, препаратах и данных сеансов обработки;

– элементы интерфейса (UIScaling, UISliding, DataStructureElement, Panel и их наследники), обеспечивающие анимацию, плавные переходы и визуализацию данных на экране;

– перечисления (UICommandType), задающие типы команд и действий, передаваемых между клиентом и сервером.

Все эти классы и перечисления связаны ассоциациями и зависимостями, отражая архитектуру клиентской части: одни компоненты формируют и отправляют запросы, другие — обрабатывают ответы и обновляют интерфейс, третьи — управляют локальными коллекциями данных и обеспечивают удобство работы пользователя.

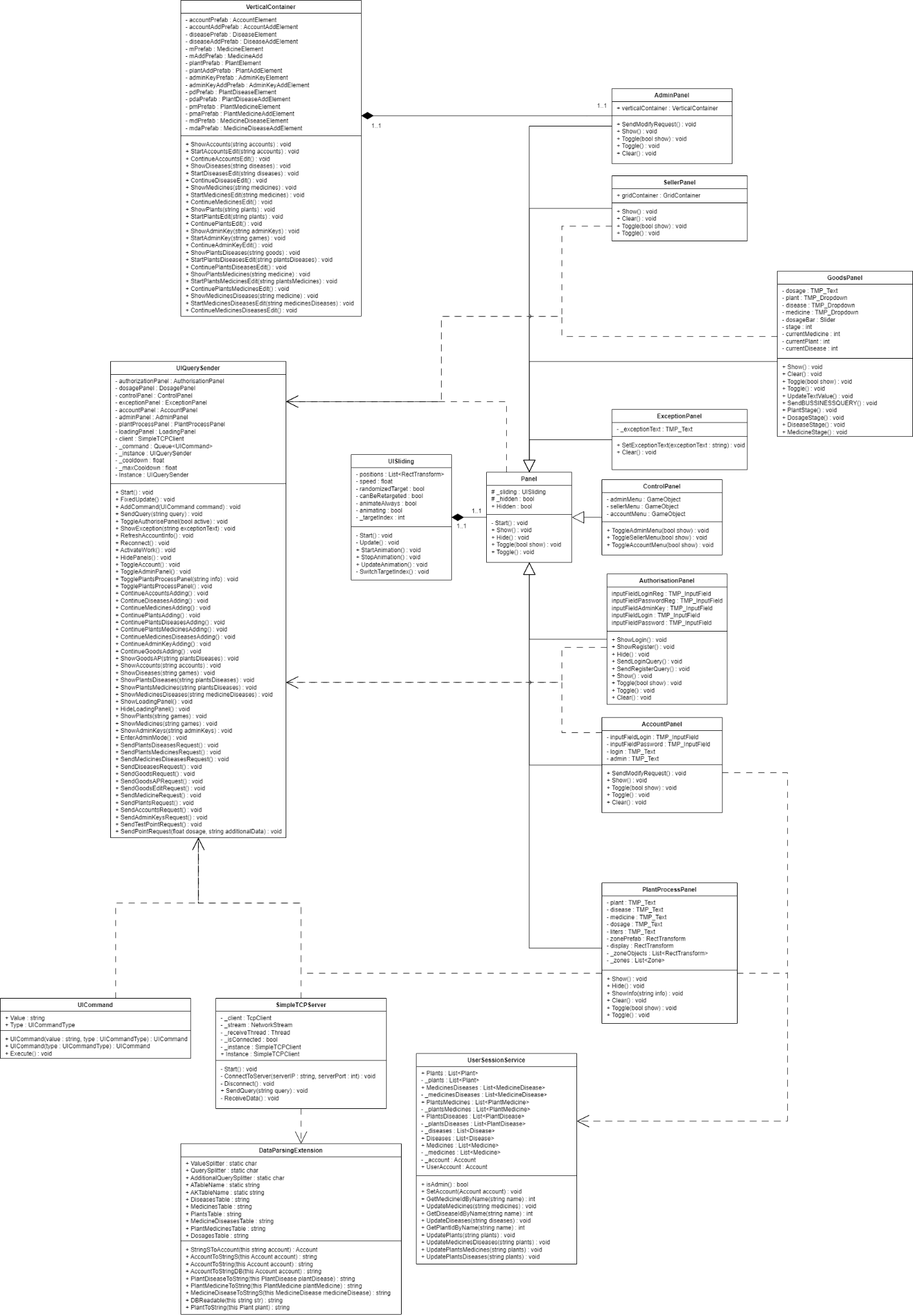


Рисунок 6.2 – Диаграмма классов клиентской части системы дистанционного управления опрыскивателем

На рисунках 6.3.1, 6.3.2 и 6.3.3 изображены диаграммы классов серверной части приложения, которые включают в себя классы и перечисления, реализующие бизнес-логику, взаимодействие с базой данных и механизмы дистанционного управления опрыскивателем. В частности, на этих диаграммах представлены:

– сетевой сервис TCPConnectorService, отвечающий за приём и обработку TCP-запросов от Unity-клиентов;

– сервис доступа к базе данных (DatabaseService), использующий Npgsql для выполнения SQL-запросов и управления транзакциями;

– модели данных (Plant, Disease, Medicine, PlantDisease, PlantMedicine, MedicineDisease, User, Zone) с полями, отражающими структуру таблиц PostgreSQL и связи «многие-ко-многим» или «один-ко-многим» через внешние ключи;

– утилитный класс для разбора входящих сообщений и формирования ответов.

– и другие классы.

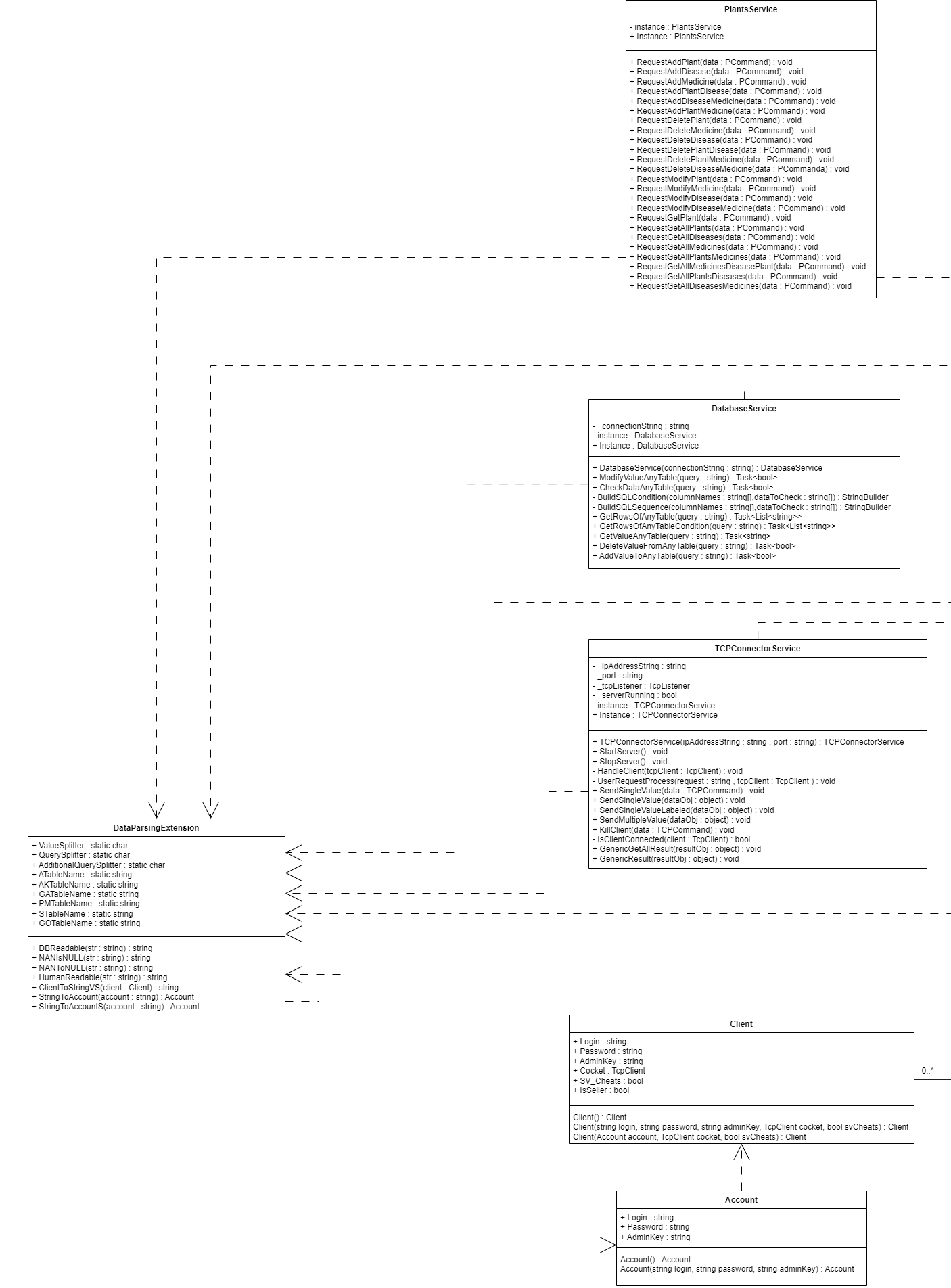


Рисунок 6.3.1 – Диаграмма классов серверной части системы дистанционного управления опрыскивателем, лист 1

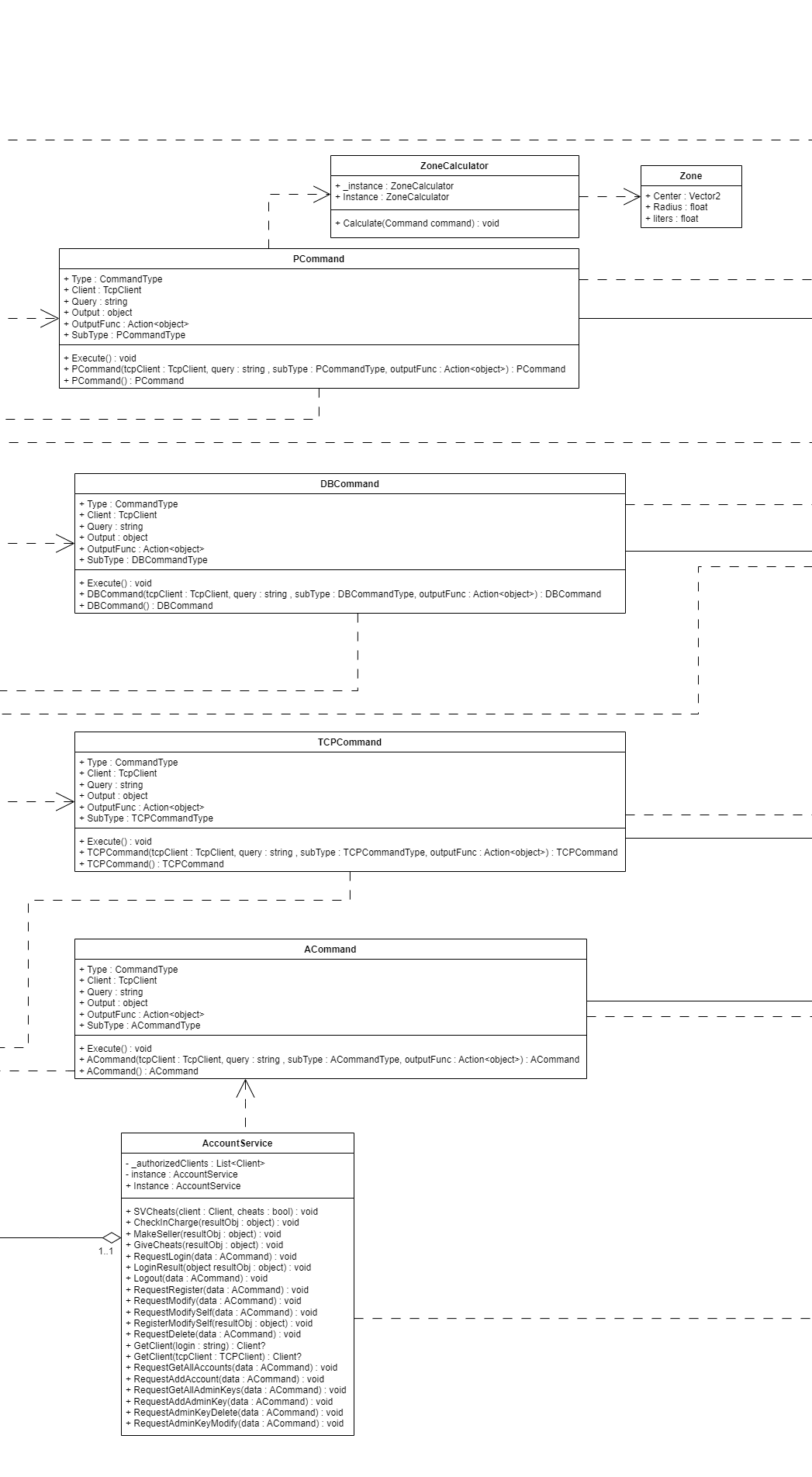


Рисунок 6.3.2 – Диаграмма классов серверной части системы дистанционного управления опрыскивателем, лист 2

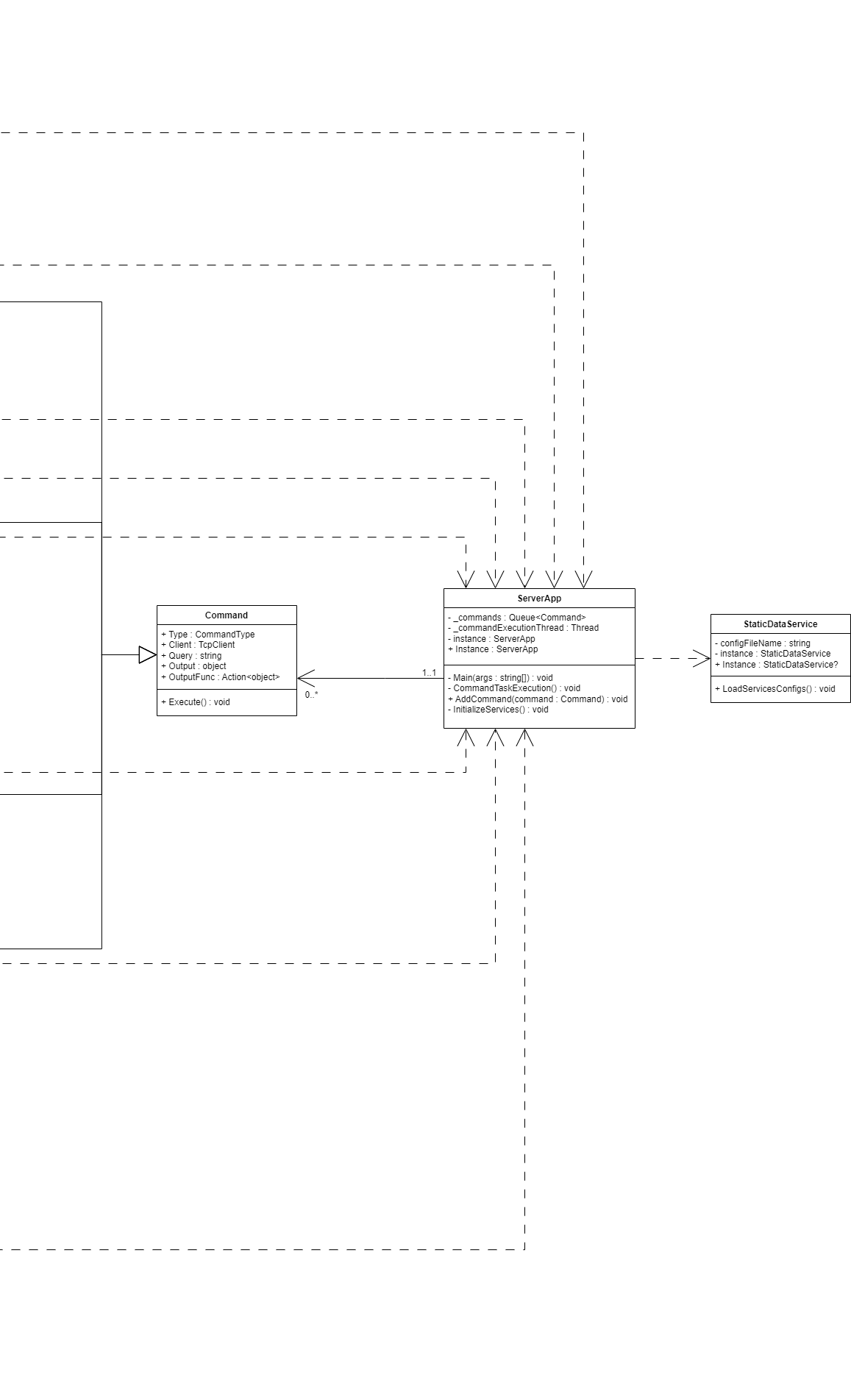


Рисунок 6.3.3 – Диаграмма классов серверной части системы дистанционного управления опрыскивателем, лист 3

# **6.3 Анализ поведения системы с использованием диаграммы последовательностей**

Диаграмма последовательностей является важным инструментом для анализа поведения системы, так как она позволяет визуализировать взаимодействие между объектами и компонентами системы во времени. В контексте системы дистанционного управления опрыскивателем, диаграмма последовательностей описывает процесс аутентификации пользователя, начиная с ввода данных на клиентской стороне и заканчивая успешным входом в систему или обработкой ответа от сервера. Это помогает понять, как данные передаются между клиентом и сервером, как обрабатываются запросы, и как система реагирует на действия пользователя. Диаграмма также выявляет потенциальные узкие места, такие как задержки в сетевом взаимодействии или ошибки в обработке данных, что делает ее полезной для проектирования, реализации и тестирования системы.

Ниже изображена диаграмма последовательностей процесса аутентификации пользователя в системе (рисунок 6.4).

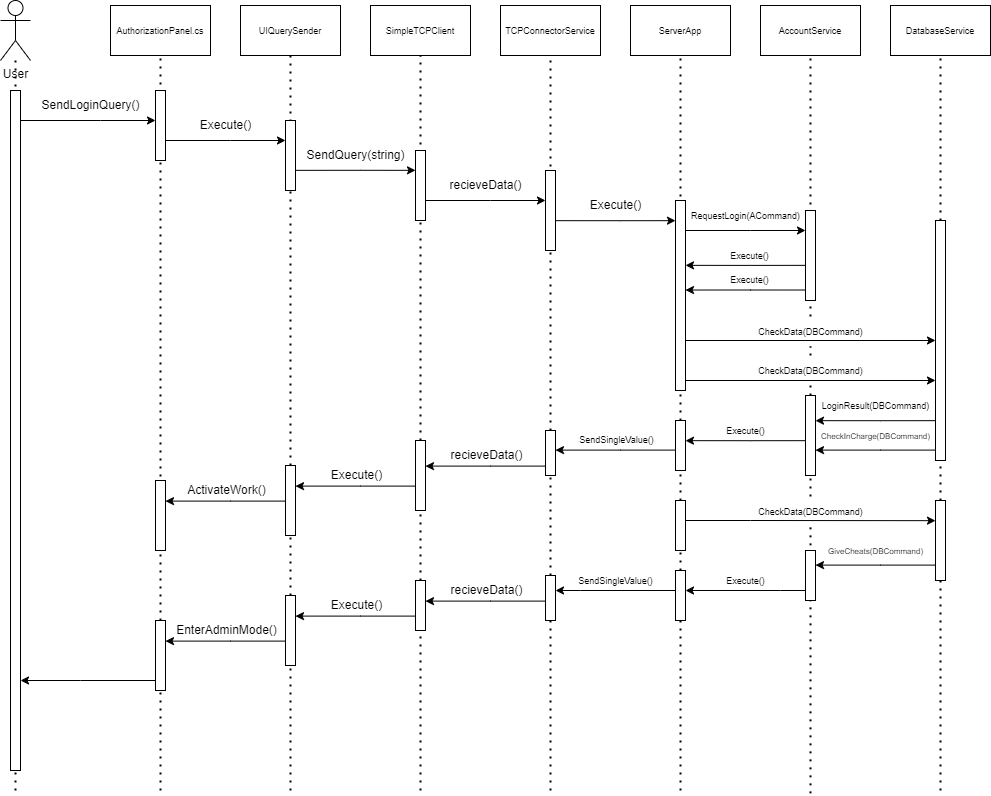


Рисунок 6.4 – Диаграмма последовательностей процесса аутентификации пользователя в системе управления акциями инвестиционного фонда

# 6.4 Моделирование жизненного цикла объектов с помощью диаграммы состояний

Диаграмма состояний (State Machine Diagram) представляет собой мощный инструмент визуального моделирования в UML, который позволяет наглядно отобразить поведение системы в терминах последовательности состояний, через которые проходит объект в ответ на внешние события. Актуальность использования именно этого вида диаграмм обусловлена тем, что процессы дистанционного управления опрыскивателем по своей природе представляют собой конечные автоматы с чётко выраженными состояниями и переходами между ними. В отличие от других UML-диаграмм, диаграмма состояний особенно хорошо подходит для моделирования:

- временных аспектов работы системы;

- реактивного поведения в ответ на внешние события;

- сложной логики принятия решений.

Ниже изображена диаграмма состояний процесса расчёта инструкций к лечению растения (рисунок 6.5).

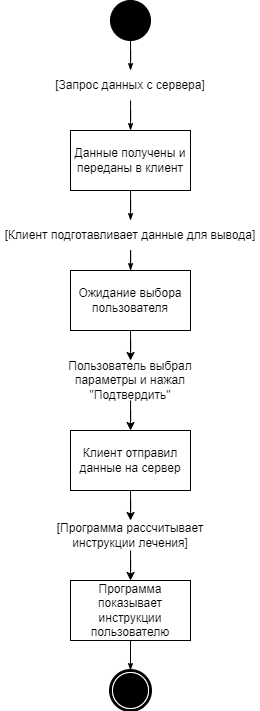


Рисунок 6.5 – Диаграмма состояний процесса покупки акций в системе управления акциями инвестиционного фонда

# 6.5 Моделирование физического размещения системы с помощью диаграммы развертывания

Диаграмма развертывания является важным инструментом в процессе проектирования программного обеспечения, так как она предоставляет наглядное представление физической архитектуры системы. Она показывает, как программные компоненты, называемые артефактами, распределяются по аппаратным устройствам и исполняющим средам, а также описывает их взаимодействие через сетевые соединения. Для программного средства дистанционного управления опрыскивателем диаграмма развертывания необходима для определения инфраструктуры, обеспечивающей функционирование системы. Она помогает разработчикам, системным администраторам и другим заинтересованным сторонам понять, какие аппаратные и программные ресурсы необходимы для работы приложения. Диаграмма позволяет выявить потенциальные узкие места в архитектуре, определить требования к оборудованию и обеспечить масштабируемость и надежность системы. В данном разделе представлена диаграмма развертывания, которая отражает физическое размещение компонентов системы, а также подробное описание ее структуры и процессов.

Диаграмма развертывания программного средства для дистанционного управления опрыскивателем представлена на рисунке 6.6. Она иллюстрирует физическую архитектуру системы, включая аппаратные устройства, исполняющие среды, программные артефакты и их взаимодействие.

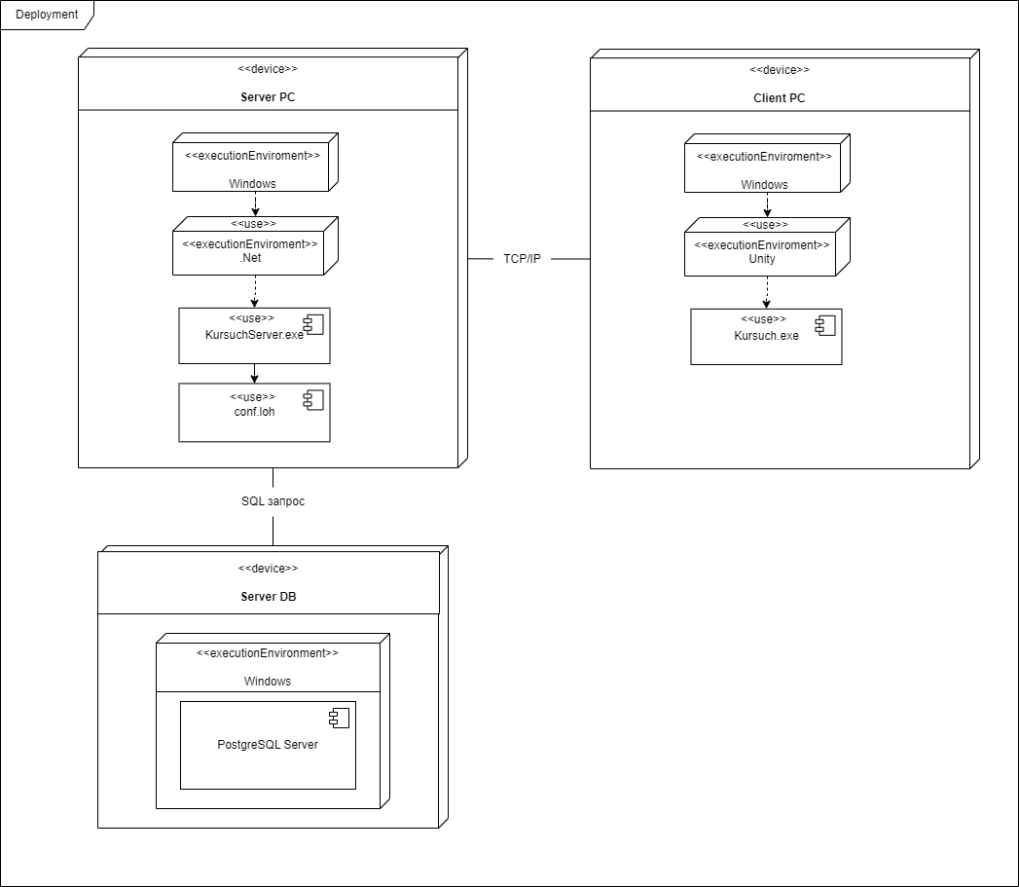


Рисунок 6.6 – Диаграмма развертывания средства для дистанционного управления опрыскивателем

Диаграмма развертывания отражает физическую архитектуру системы и показывает минимальный набор оборудования, необходимый для её функционирования. В состав инфраструктуры входят три основных устройства: машина, на которой развёрнута серверная часть системы; клиентский компьютер с установленным приложением на Unity; а также отдельный сервер баз данных, обеспечивающий хранение и обработку информации. Взаимодействие между клиентом и сервером реализовано с использованием стека протоколов TCP/IP, что гарантирует стабильную передачу данных. Подключение к базе данных осуществляется через библиотеку Npgsql, обеспечивающую эффективное и безопасное взаимодействие с СУБД PostgreSQL.

# 6.6 Вывод по разделу «Моделирование программного средства для управления акциями инвестиционного фонда с использованием UML»

Моделирование программного средства дистанционного управления опрыскивателем растений с использованием языка UML обеспечило комплексное представление о системе, охватывающее её функциональные, структурные, поведенческие и физические аспекты. Применение различных видов UML-диаграмм позволило формализовать архитектуру, упростить этапы проектирования и обеспечить точную координацию между компонентами, что стало надёжной основой для последующей реализации, тестирования и эксплуатации программного продукта.

Диаграмма вариантов использования помогла определить ключевые роли в системе — администратора и оператора — и связать их с основными действиями, доступными через клиентское приложение. Это обеспечило чёткое понимание пользовательских сценариев и позволило выстроить логику интерфейса, соответствующую ожиданиям конечных пользователей. Благодаря учёту взаимосвязей между действиями удалось детализировать поведение системы при различных режимах работы и формализовать требования, предъявляемые к функциям доступа и управления.

Классовые диаграммы клиентской и серверной части отразили структуру системы с акцентом на распределение обязанностей между модулями. Клиентская часть, реализованная в Unity, отвечает за визуализацию и пользовательское взаимодействие, в то время как сервер, реализованный в .NET, обрабатывает сетевые запросы, управляет бизнес-логикой и взаимодействует с базой данных. Отображение классов, их полей, методов и связей обеспечило чёткое соответствие между архитектурой и кодовой реализацией, способствовало модульности, тестируемости и масштабируемости решения.

Диаграмма последовательностей, описывающая процесс авторизации, позволила визуализировать взаимодействие между участниками — от момента ввода данных до обработки на сервере и обратной связи в интерфейсе клиента. Этот сценарий дал возможность оценить динамику сетевых запросов, выявить потенциальные задержки и оптимизировать логику обработки. Благодаря такому анализу обеспечивается устойчивость к сбоям и повышается отзывчивость системы при взаимодействии с пользователем.

Диаграмма состояний, отражающая поведение системы при расчёте зон опрыскивания, показала, как объект (опрыскиватель или сеанс обработки) переходит между состояниями от момента инициализации до завершения обработки. Чёткое определение переходов и условий их выполнения позволило устранить неоднозначности, улучшить предсказуемость поведения и реализовать корректную обработку исключительных ситуаций.

Диаграмма развертывания отразила физическое размещение компонентов: Unity-клиента, серверной части и базы данных. На её основе были определены характеристики взаимодействия между слоями системы, маршруты сетевых соединений, а также требования к аппаратному обеспечению и платформенной независимости. Отдельное внимание было уделено обеспечению надёжного соединения, безопасности передаваемых данных и соответствию системе хранения критически важной информации. Используемые технологии, включая .NET, PostgreSQL, Unity и Npgsql, обеспечили прочную технологическую базу и гибкость масштабирования.

В результате применение UML-диаграмм позволило систематизировать подход к проектированию, устранить дублирование и снизить риски неконсистентности между логикой системы и её реализацией. Визуальное моделирование повысило прозрачность разработки и обеспечило единое понимание структуры всеми участниками проекта. Полученные модели подтвердили, что программное средство для дистанционного управления опрыскивателем обладает высокой степенью надёжности, производительности и адаптивности к дальнейшему расширению.

# 7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программное средство для дистанционного управления опрыскивателем растений представляет собой комплексную систему, созданную для автоматизации процессов планирования и реализации операций по обработке сельскохозяйственных участков, контроля параметров опрыскивателя и обеспечения аналитической поддержки операторов. Для эффективного использования этого решения необходимо наличие ясной и понятной документации, которая позволит всем категориям пользователей быстро освоить функционал и выполнять поставленные задачи без лишних рисков. Настоящий раздел разработан таким образом, чтобы представить подробные инструкции по взаимодействию с приложением, оказывая поддержку как администраторам, так и операторам опрыскивателя. Руководство является важной частью проекта, так как оно не только облегчает работу с системой, но и способствует её надежной и корректной эксплуатации в аграрной сфере.

Основная цель данного раздела заключается в том, чтобы описать ключевые возможности программного обеспечения, развернуть последовательность действий для выполнения типовых операций и разъяснить предназначение ролей внутри системы. Администраторы отвечают за настройку и контроль доступа, управление списками растений, заболеваний и препаратов, а также за обобщение статистики о выполненных операциях и обслуживание базы данных. Операторы, в первую очередь, занимаются выбором участка для опрыскивания, вводом координат, заданием радиуса действия и дозировки, запуском вычислительных модулей и визуализацией расчётных зон. Поскольку процессы обработки растений требуют высокой точности и оперативного реагирования на изменения условий, документация уделяет особое внимание надёжности выполнения операций, правильному использованию интерфейса и последовательным действиям, приводящим к ожидаемому результату. Каждый подраздел сопровождается примерами экранных форм и практическими советами по устранению возможных неточностей, что делает руководство удобным инструментом для пользователей с разным уровнем подготовки.

В условиях агротехнической области, где любая ошибка в параметрах обработки может привести к снижению урожайности или избыточному расходу рабочего раствора, руководство по эксплуатации является ключевым элементом. Оно не только помогает оператору быстро освоить интерфейс и логику работы с программой, но и выстраивает единый стандарт взаимодействия с системой, тем самым сокращая количество ошибок и повышая общую эффективность. Благодаря применению передовых технологий, таких как Unity UI для реализации наглядного и интерактивного пользовательского интерфейса и PostgreSQL в связке с Npgsql для стабильного хранения и обработки данных, система обладает понятным дизайном и высокой эксплуатационной надёжностью, что находит отражение в пошаговых инструкциях данного раздела.

# 7.1 Подготовка к эксплуатации

Для начала работы с программным средством дистанционного управления опрыскивателем растений необходимо выполнить подготовительные шаги для правильной настройки и запуска системы. Данный подраздел описывает процесс создания базы данных, запуска серверного и клиентского приложений, авторизации и настройки интерфейса. Эти действия обеспечивают надёжное и безопасное использование системы, что особенно важно в аграрной сфере:

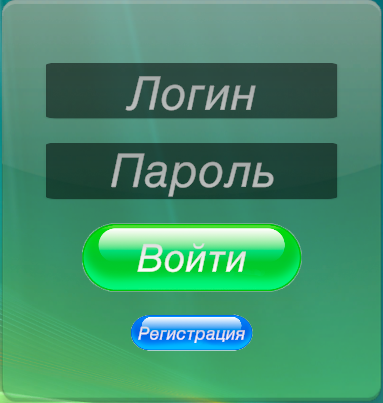
1 Создание базы данных. Программное средство использует PostgreSQL для хранения сведений о растениях, болезнях, препаратах, пользователях и сеансах опрыскивания. Установите PostgreSQL на сервер, создайте новую базу и выполните SQL-скрипт для инициализации таблиц. Это гарантирует сохранение всех необходимых данных.

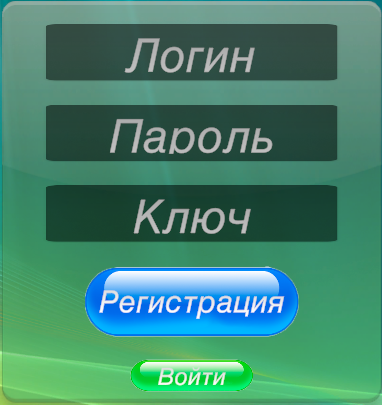
2 Запуск серверного приложения. Консольное C#-приложение запускается на устройстве с .NET 6+. После старта сервер подключается к базе через Npgsql и начинает принимать TCP-запросы от Unity-клиентов. Это обеспечивает работу логики расчёта зон и передачи результатов.

3 Запуск клиентского приложения. Unity-проект на C# (в JetBrains Rider) запускается на Windows или Linux. При старте клиент устанавливает TCP-соединение и отображает интерфейс для авторизации.

4 Авторизация в системе. Для доступа к системе пользователь должен ввести логин и пароль на экране входа. Успешная авторизация открывает интерфейс, соответствующий роли пользователя (администратор или обычный пользователь). Этот шаг гарантирует безопасный доступ и определяет доступные функции. Ниже представлен скриншот, демонстрирующий интерфейс окна авторизации (рисунок 7.1.1).

4.1 В случае, если у пользователя нету учётной записи. Он должен её создать в окне регистрации (Рисунок 7.1.2)





Рисунки 7.1.1 и 7.1.2 – Скриншот окна авторизации и регистрации

5 Панель управления. После авторизации пользователю откроется панель управления, с помощью которой можно перейти в одно из меню программы. (Рисунок 7.2)

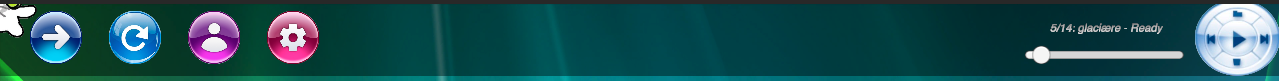


Рисунок 7.2 – Скриншот панели управления

6 Настройка персональных данных. После авторизации пользователь может изменить свои данные, такие как имя логин или пароль, через панель аккаунта в интерфейсе. Ниже представлен скриншот, демонстрирующий интерфейс панели аккаунта (рисунок 7.3).

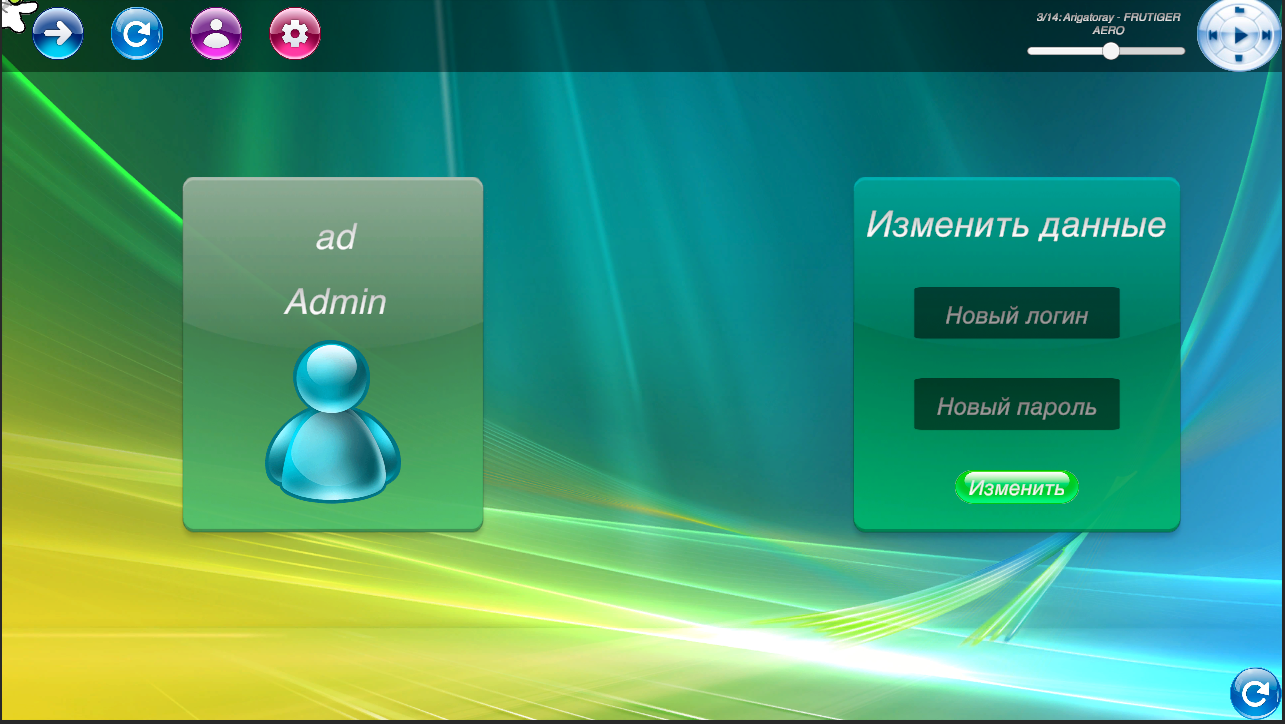


Рисунок 7.3 – Скриншот окна изменения данных

7 Кнопки переподключения находятся в правом нижнем и левом верхнем углах экрана, позволяют пользователю переподключиться к серверу перезапустив клиентское приложение (рисунок 7.4).



Рисунок 7.4 – Скриншот кнопки переподключения

Подготовка к эксплуатации обеспечивает корректную работу программного средства, позволяя пользователям эффективно выполнять задачи, связанные с управлением акциями. Простота настройки и интуитивный интерфейс системы способствуют быстрому освоению, что особенно важно для обеспечения оперативности и точности в финансовых операциях.

# 7.2 Руководство для администратора

Данный подраздел описывает интерфейс и функциональность, доступные пользователю с административными правами. Администратор отвечает за управление учетными записями, акциями, компаниями и настройку системы, что обеспечивает ее стабильную и безопасную работу. В данном разделе представлены основные экраны интерфейса, их элементы и назначение, что позволяет понять, как администратор взаимодействует с системой. Описание ориентировано на предметную область финансов, подчеркивая важность точности и надежности операций.

Панель администратора, представленная на рисунке 7.6.1, она предоставляет доступ к основным функциям управления системой через удобный и структурированный интерфейс. Ниже описаны элементы интерфейса и их назначение:

1 Панель выбора типа записи. В левой части экрана находятся кнопки выбора типа записи, нажав на которые можно добавлять, удалять и изменять соответствующие типы записей базы данных.

2 Записи (в данном случае список болезней). В центральной части экрана расположен список записей с кнопками, позволяющими управлять базой данных.

2.1 Кнопка «Изменить». Позволяет изменить данные конкретной записи в БД, каскадно поменяв её в связанных таблицах (рисунок 7.5.3).

2.2 Кнопка «Удалить». Позволяет удалить данные записи, также каскадно стерев в её в связанных таблицах (рисунок 7.5.3).

2.3 Кнопка «Подтвердить». Позволяет подтвердить добавление новой записи в БД (рисунок 7.5.4).



Рисунок 7.5.1 – Скриншот главного меню администратора



Рисунок 7.5.2 – Скриншот элемента записи

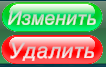


Рисунок 7.5.3 – Скриншот кнопки удалить и изменить на элементе записи



Рисунок 7.5.4 – Скриншот элемента добавления с базу данных

Элементы интерфейса панели администратора организованы в вертикальной компоновке, что обеспечивает удобство навигации. Кнопки имеют четкие подписи, отражающие их назначение. Функциональность, доступная через кнопки, покрывает основные задачи администратора, связанные с управлением системой, что делает меню ключевым элементом административного интерфейса. Добавление и изменение различных типов записей аналогично.

# 7.3 Руководство для пользователя

Данный подраздел описывает интерфейс и функциональность, доступные пользователю с обычными правами. Пользователь может расcчитать лечение растения. В данном разделе представлены основные экраны интерфейса, их элементы и назначение, что помогает понять, как пользователь взаимодействует с системой.

Главное меню, представленное на рисунке 7.6, является основным экраном, отображаемым после успешной авторизации пользователя. Оно служит точкой доступа к функции . Ниже описаны элементы интерфейса и их назначение:

1 Кнопка выбора растения. Позволяет протестировать функционал прототипа, выбрав растение из базы данных.

2 Кнопка выбора болезни. Позволяет протестировать функционал прототипа, выбрав подходящую болезнь из базы данных.

3 Кнопка выбора лекарства. Позволяет протестировать функционал прототипа, выбрав подходящее лекарство из базы данных.

4 Ползунок выбора дозировки лекарства. Позволяет протестировать функционал прототипа, выбрав дозировку лекарства.

5 Кнопка подтверждения выбора. Позволяет начать тестирование прототипа программы запуская процесс расчёта лечения растения.



Рисунок 7.6 – Скриншот главного меню пользователя

# 7.4 Вывод по разделу «Руководство пользователя»

Данный раздел содержит подробное описание процессов и интерфейсов, необходимых для работы с программным средством дистанционного управления опрыскивателем растений. Включённые подразделы, охватывающие подготовку к эксплуатации, работу администратора и оператора, формируют целостное представление о структуре и функциональности системы. Акцент сделан на удобстве использования, надёжности операций и защите данных, что особенно важно в аграрной сфере.

Раздел, посвящённый подготовке к работе, описывает создание базы данных, запуск серверной и клиентской частей, а также начальную авторизацию, что обеспечивает стабильный старт системы. Руководство для администратора охватывает интерфейсы управления справочниками и пользователями, позволяя поддерживать актуальность данных и разграничение прав доступа. Руководство для оператора описывает рабочие панели, используемые для задания параметров обработки и визуализации зон опрыскивания, что упрощает взаимодействие с системой в повседневной работе.

В совокупности содержимое раздела демонстрирует, что руководство пользователя играет важную роль в обеспечении эффективности и безопасности работы программного средства. Чёткое разделение функций, интуитивный интерфейс Unity и корректная обработка пользовательских действий формируют надёжную основу для успешной эксплуатации как администраторами, так и операторами.

# 8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

Тестирование завершает разработку программного средства для дистанционного управления опрыскивателем растений и направлено на подтверждение его функциональности, надёжности и устойчивости при различных условиях эксплуатации. В этом разделе описаны результаты проверок, уделено внимание обработке исключительных ситуаций, некорректных данных и возможных сбоев, возникающих в процессе работы. В агротехнической среде, где точность расчётов, сохранность данных и оперативность реакции имеют решающее значение, тестирование служит ключевым инструментом обеспечения качества и безопасности системы.

Система, реализованная на основе технологий C#, Unity, PostgreSQL, Npgsql, прошла проверку в сценариях как штатного использования, так и экстренных ситуациях. При стандартных операциях проверялось поведение механизма авторизации, приём и корректность команд. В дополнение к этому моделировались ситуации с некорректным вводом—например, передача невалидных данных. В процессе тестирования оценивалось, насколько эффективно система перехватывает исключения, сохраняет непротиворечивость информации в базе данных и сообщает оператору о возникших ошибках.

Одними из ключевых исключительных ситуаций, протестированных в системе, являются различные ошибки ввода данных при авторизации или регистрации, такие как:

1 Ввод неправильного логшина или пароля;

2 Ввод пустых полей;

3 Ввод специальных символов.

Данные ситуации моделируют распространенные ошибки пользователя или потенциальную попытку несанкционированного доступа. При вводе некорректных учетных данных система, возвращает сообщение об ошибке через интерфейс (рисунок 8.1). Это сообщение отображается в диалоговом окне, показывая информацию о внутренней ошибке с сервера, что позволяет персоналу оперативно исправить проблему, а оператору получить точный ответ места проблемы. Система также фиксирует попытку входа в логах обращений к серверу, что позволяет системному администратору сервера отслеживать подозрительную активность.

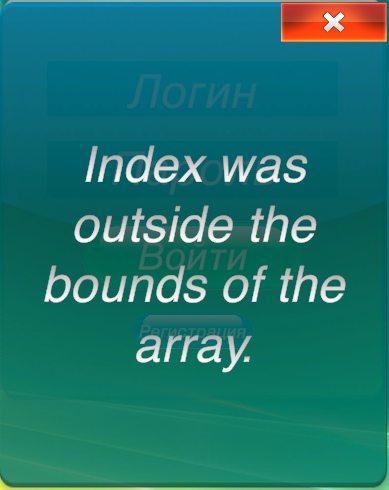


Рисунок 8.1 – Обработка неправильных данных для авторизации

Еще одними из ключевых протестированных ситуаций является попытки регистрации с ошибочно введёнными данными, такие как:

1 Ввод, при регистрации, уже существующих данных;

2 Ввод, при регистрации, пустых полей;

3 Ввод, при регистрации, специальных символов.

При отправке запроса на регистрацию с некорректно введёнными данными (рисунок 8.2), отображает в интерфейсе диалоговое окно с сообщением. Это уведомление информирует пользователя об ошибке добавления записи в базу, позволяя понять причину ошибки.

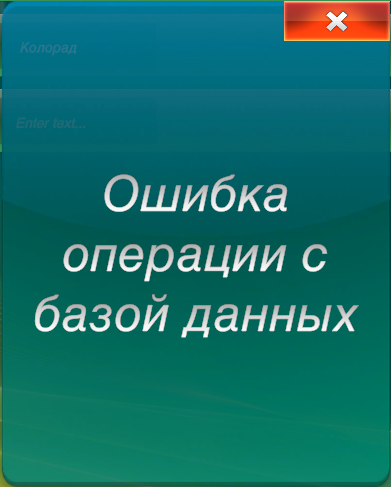


Рисунок 8.2 – Попытка регистрации с некорректными данными

Также протестированы сценарии ввода некорректных данных в элементы панели администратора. В случаях, когда запись уже существует или значения записи введены некорректно, операция на серверной части программы отбракуется и выполнена не будет. Выведется окно об ошибке взаимодействия с базой данных (Рисунок 8.2).

В ходе тестирования была проанализирована работа системы логгирования, которая фиксирует все операции отправленные пользователями. На рисунке 8.3 можно увидеть пример информации, отображаемой в логах. Логи находятся в консоли на сервере. Этот механизм позволяет в реальном времени отслеживать подозрительную активность, диагностировать сбои и проводить аудит, что повышает безопасность и надежность системы. В случае возникновения исключительных ситуаций программа выводит её в логи, позволяя понять ошибку и принять меры.

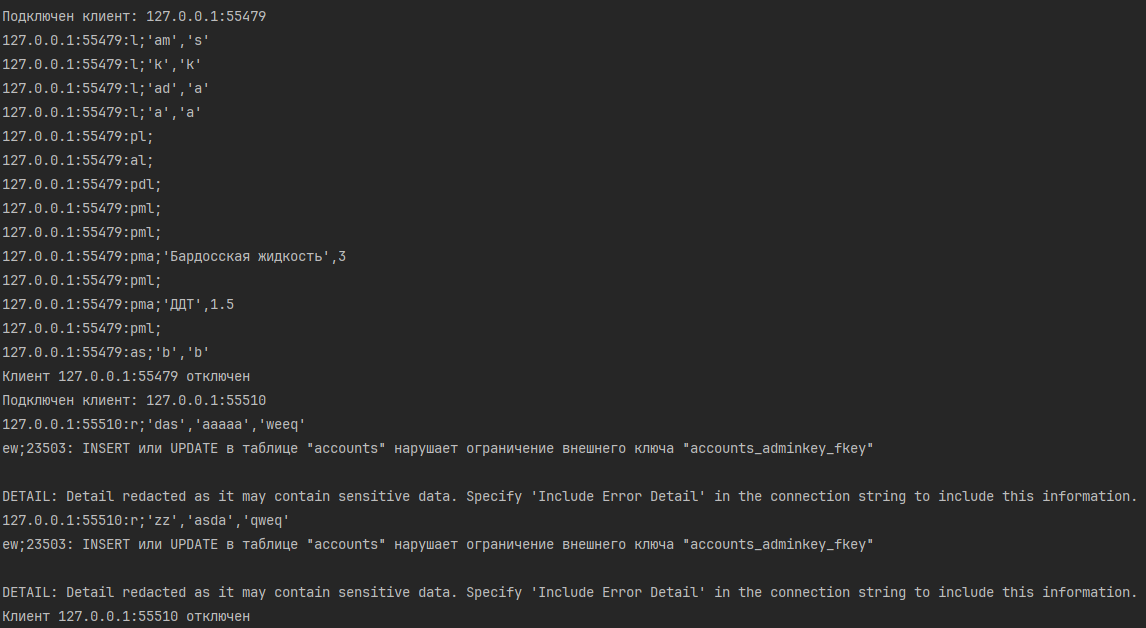


Рисунок 9.3 – Пример логов работы программного средства

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было создано программное средство для дистанционного управления опрыскивателем растений, автоматизирующее процессы мониторинга полевых данных, расчёта зон обработки, отправки команд технике и формирования отчётов. Серверная часть, реализованная на C# в виде консольного приложения, обеспечивает многопоточную обработку запросов, а Unity-клиент, построенный на C# и Unity UI, отвечает за визуализацию интерфейса. Хранение данных в PostgreSQL посредством Npgsql и использование BCrypt.Net для безопасного хэширования паролей гарантируют надёжность и защиту информации. В совокупности получился инструмент, который оптимизирует агротехнологические операции, снижает риски и повышает прозрачность взаимодействий, что критически важно в сфере сельскохозяйственного дистанционного управления.

Первый раздел был посвящён анализу предметной области управления опрыскивателями, в котором рассмотрены ключевые этапы: от выбора культур и учёта заболеваний до подбора препаратов и расчёта маршрутов. Были изучены существующие решения в агроавтоматизации, выявлена необходимость в собственном программном комплексе, способном работать в режиме реального времени с учётом геоданных и параметров обработки. Результатом аналитической части стало обоснование задачи разработки клиент-серверного приложения, способного удовлетворить требования точности, скорости и безопасности.

Во втором разделе сформулированы основные задачи создания многопоточного серверного приложения и Unity-клиента. Описаны требования к функциональности: единый протокол обмена по TCP/IP с использованием текстовых сообщений, эффективные алгоритмы расчёта зон обработки, хранение справочников и сессий в PostgreSQL, а также система авторизации с разграничением прав. Выбран стек технологий – C# и .NET для бизнес-логики, Unity UI для клиентского интерфейса, Npgsql для доступа к базе. Это послужило надежным фундаментом для масштабируемости, производительности и удобства дальнейшей реализации.

Третий раздел посвятили функциональному моделированию с помощью методологии IDEF0. Построена многоуровневая модель процесса расчёта зон: от этапа приёма координатных точек и параметров работы до объединения пересекающихся кругов, вычисления рабочих зон и формирования результатов. В декомпозиции на несколько уровней были явно показаны входящие данные, управляющие сигналы и механизмы (сервер, алгоритм расчёта, отправка ответа), что упростило последующую реализацию и позволило учесть требования по прозрачности и проверке соответствия регламентам агротехнических процедур.

Четвёртый раздел содержит информационную модель IDEF1X, описывающую структуру реляционной базы данных. Нормализация до третьей нормальной формы исключила избыточность и обеспечила целостность данных. Это позволило эффективно хранить и обрабатывать запросы, связанные с проверкой прав доступа, чтением справочников и сохранением результатов расчёта, что является ключевым для устойчивой работы системы.

Пятый раздел посвящён детальному описанию алгоритмов серверной бизнес-логики: процесса авторизации, валидации входных данных, расчёта зон обработки и формирования отчетов. Представлены схемы алгоритмов, наглядно демонстрирующие последовательность шагов: чтение запроса клиента, проверка прав, выбор справочных данных, группировка точек, объединение пересекающихся зон, вычисление объёмов воды и отправка ответа. Такой подход минимизирует вероятность ошибок при реализации и обеспечивает прозрачность всех операций.

Шестой раздел охватывает UML-моделирование системы. На диаграмме вариантов использования показаны роли «Администратор» и «Пользователь», их взаимодействие с системой при настройке справочников и запуске расчётов. Диаграммы классов детализировали структуру клиентской и серверной частей, описали контроллеры, сервисы, модели данных и утилиты. Диаграмма последовательностей раскрыла динамику вызовов при аутентификации и расчёте зон, диаграмма состояний продемонстрировала переходы жизненного цикла сессии расчёта, а диаграмма развёртывания отразила физическую архитектуру: сервер с консольным приложением, клиент-машину с Unity-приложением и сервер базы данных, соединённые через TCP/IP и Npgsql. Все эти модели обеспечили модульность, масштабируемость системы и подтвердили обоснованность архитектурного решения.

Седьмой раздел содержал руководство пользователя, в котором описано начальное развертывание: создание базы данных, запуск серверного и клиентского приложений, регистрация пользователей. Интерфейсы администратора, позволяющие редактировать справочники, управлять учётными записями и настраивать параметры проекта, и интерфейсы оператора, отвечающие за выбор участка, ввод точек и запуск расчёта зон, реализованы с помощью Unity UI и собственных скриптов анимаций UIScaling и UISliding. Это обеспечило интуитивность, безопасность и минимизацию рисков ошибок при эксплуатации.

Восьмой раздел предоставил результаты тестирования, в ходе которого была проверена устойчивость системы к различным исключительным ситуациям: некорректному вводу координат, отсутствию связи с сервером, попыткам неавторизованных действий и обработке ошибок в реальном времени. С помощью Npgsql-транзакций и обработки исключений в C# удалось обеспечить целостность данных и прозрачность операций, что является обязательным в условиях удалённого управления агротехническими задачами.

В результате разработанное программное средство успешно автоматизирует удалённое управление опрыскивателем растений, повышая эффективность расчётов, точность обработки полей и прозрачность взаимодействия между оператором и сервером. Система готова к внедрению в агропромышленных условиях и дальнейшему расширению, полностью соответствуя требованиям производственной сферы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Никитина, Т.В. Основы портфельного инвестирования / Т.В. Никитина. – Москва : Юрайт, 2018. – С. 137–140.

[2] Якушин, Д.И. Управление инвестиционным портфелем / Д.И. Якушин. – Москва : Директ-Медиа, 2022. – 120 с.

[3] A2is. Программы для управления инвестициями [Электронный ресурс]. – Режим доступа: https://a2is.ru/catalog/programmy-dlya-upravleniya-investitsiyami. – Дата доступа: 24.03.2025.

[4] Шилдт, Г. Java. Полное руководство / Г. Шилдт. – Москва : Вильямс, 2018. – 1344 с.

[5] Лузанов, П. PostgreSQL. Профессиональный SQL / П. Лузанов, Е. Рогов, И. Левшин. – Москва : Postgres Professional, 2020. – 320 с.

[6] Бауэр, К. Java Persistence API и Hibernate / К. Бауэр, Г. Кинг, Г. Грегори. – Москва : ДМК-Пресс, 2017. – 652 с.

[7] JavaFX Documentation [Электронный ресурс]. – Режим доступа: https://openjfx.io/. – Дата доступа: 16.04.2025.

[8] Apache Log4j [Электронный ресурс]. – Режим доступа: https://logging.apache.org/log4j/2.x/. – Дата доступа: 17.04.2025.

[9] Черемных, С.В. Моделирование и анализ систем / С.В. Черемных, И.О. Семенов, В.С. Ручкин. – Москва : Финансы и статистика, 2006. – 188 с.

[10] IDEF1X [Электронный ресурс]. – Режим доступа: https://www.cfin.ru/vernikov/idef/idef1x.shtml. – Дата доступа: 22.04.2025.

[11] Project Lombok [Электронный ресурс]. – Режим доступа: https://projectlombok.org/. – Дата доступа: 23.04.2025.

[12] jBCrypt [Электронный ресурс]. – Режим доступа: https://www.mindrot.org/projects/jBCrypt/. – Дата доступа: 23.04.2025.

[13] OpenPDF [Электронный ресурс]. – Режим доступа: https://github.com/LibrePDF/OpenPDF. – Дата доступа: 23.04.2025.

[14] Apache POI [Электронный ресурс]. – Режим доступа: https://poi.apache.org/. – Дата доступа: 23.04.2025.

[15] Гамма, Э. Шаблоны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – Санкт-Петербург : Питер, 2010. – 368 с.

[16] Фаулер, М. UML. Краткое руководство / М. Фаулер. – Москва : ДМК Пресс, 2004. – 192 с.

[17] Ларман, К. Применение UML и шаблонов / К. Ларман. – Москва : Вильямс, 2006. – 624 с.

# ПРИЛОЖЕНИЕ А (обязательное) Отчет о проверке на заимствования в системе «Антиплагиат»

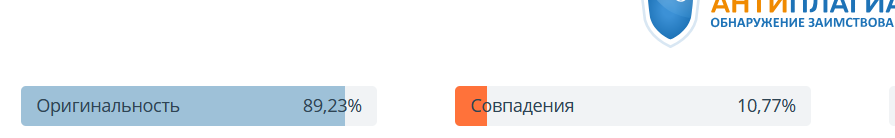


Рисунок А.1 – Проверка на Антиплагиат

# ПРИЛОЖЕНИЕ Б (обязательное) Листинг алгоритмов, реализующих бизнес-логику

Жизненный цикл клиент-серверного взаимодействия (класс ServerApp.cs):

using Npgsql;  
using System;  
using System.Net;  
using System.Net.Sockets;  
using System.Text;  
using System.Threading;  
using KursuchServer.Services;  
  
namespace KursuchServer  
{  
 class ServerApp  
 {  
 private Queue<Command> \_commands = new();  
 private Thread \_commandExecutionThread;  
 private static ServerApp instance = null;  
 public static ServerApp Instance  
 {  
 get  
 {  
 if (instance == null) instance = new ServerApp();  
 return instance;  
 }  
 private set { instance = value; }  
 }  
  
 static void Main(string[] args)  
 {  
 InitializeServices();  
   
 Instance.\_commandExecutionThread = new Thread(Instance.CommandTaskExecution);  
 Instance.\_commandExecutionThread.Start();  
   
 TCPConnectorService.Instance.StartServer();  
 }  
  
 private void CommandTaskExecution()  
 {  
 while (true)  
 {  
 if(\_commands.Count > 0) \_commands.Dequeue().Execute();  
 Thread.Sleep(10);  
 }  
 }  
  
 public void AddCommand(Command command)  
 {  
 \_commands.Enqueue(command);  
 }  
  
 private static void InitializeServices()  
 {  
 StaticDataService.Instance.LoadServicesConfigs();  
 }

Продолжение приложения Б

}  
}

Процесс авторизации, регистрации, обновления и удаления записей аккаунтов (класс AccountService.cs):

using System.Net.Sockets;  
using System.Xml;  
using KursuchServer.DataStructures;  
  
namespace KursuchServer.Services;  
  
public class AccountService  
{  
 private List<Client> \_authorizedClients = new();  
  
 private static AccountService instance;  
  
 public static AccountService Instance  
 {  
 get  
 {  
 if (instance == null) instance = new();  
 return instance;  
 }  
 private set { instance = value; }  
 }  
  
 public void SVCheats(Client client, bool cheats) *//* {  
 client.SV\_Cheats = cheats;  
 }  
  
 public void CheckInCharge(Object resultObj)  
 {  
 var result = (DBCommand)resultObj;  
  
 if (result.Query == "ERR") return;  
  
 var output = (String)result.Output;  
   
 ServerApp.Instance.AddCommand(  
 new DBCommand(result.Client,  
 output.Split(DataParsingExtension.ValueSplitter)[2].DBReadable() +  
 $";adminKey;{DataParsingExtension.AKTableName}",  
 DBCommandType.**CheckData**,  
 GiveCheats));  
 }  
  
 public void GiveCheats(Object resultObj)  
 {  
 var result = (DBCommand)resultObj;  
  
 if (result.Query == "ERR")  
 {  
 GetClient(result.Client).SV\_Cheats = false;  
  
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 $"lov{DataParsingExtension.QuerySplitter}{result.Query}",

Продолжение приложения Б

TCPCommandType.**SendSingleValue**));  
  
 return;  
 }  
 GetClient(result.Client).SV\_Cheats = true;  
   
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 $"loa{DataParsingExtension.QuerySplitter}{result.Query}",  
 TCPCommandType.**SendSingleValue**));  
 }  
  
 public void RequestLogin(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(  
 new DBCommand(data.Client,  
 data.Query.Split(DataParsingExtension.ValueSplitter)[0] + DataParsingExtension.ValueSplitter +  
 data.Query.Split(DataParsingExtension.ValueSplitter)[1] +  
 $";login,password;{DataParsingExtension.ATableName}", DBCommandType.**CheckData**,  
 LoginResult));  
  
 ServerApp.Instance.AddCommand(  
 new DBCommand(data.Client,  
 data.Query.Split(DataParsingExtension.ValueSplitter)[0] +  
 $";login;{DataParsingExtension.ATableName}",  
 DBCommandType.**ValueGet**,  
 CheckInCharge));  
 }  
  
 public void LoginResult(Object resultObj)  
 {  
 var result = (DBCommand)resultObj;  
  
 if (result.Query == "ERR")  
 {  
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 $"ef{DataParsingExtension.QuerySplitter}LE1 Неправильный логин или пароль",  
 TCPCommandType.**SendSingleValue**));  
 return;  
 }  
  
 result.Query = result.Query.HumanReadable();  
  
 \_authorizedClients.Add(new(result.Query.Split(DataParsingExtension.QuerySplitter)[0].StringToAccountS(),  
 result.Client));  
  
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 $"los{DataParsingExtension.QuerySplitter}{result.Query}",  
 TCPCommandType.**SendSingleValue**));  
 }  
  
 public void Logout(ACommand data) *//* {  
 Client client = new(data.Query.StringToAccountS(), data.Client);  
 String login = new String(client.Login.Where(c => c != '\'').ToArray());

Продолжение приложения Б

\_authorizedClients.Remove(\_authorizedClients.First(x => x.Login == login));  
  
 ServerApp.Instance.AddCommand(new TCPCommand(data.Client, $"lo{DataParsingExtension.QuerySplitter}(",  
 TCPCommandType.**DisconnectClient**));  
 }  
  
 public void RequestRegister(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(new DBCommand(data.Client,  
 $"rs;{data.Query};login,password,adminKey;{DataParsingExtension.ATableName}",  
 DBCommandType.**ValueAdd**, TCPConnectorService.Instance.GenericResult));  
 }  
  
 public void RequestModify(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(  
 new DBCommand(data.Client,  
 "am;" + data.Query.Split(DataParsingExtension.AdditionalQuerySplitter)[0] +  
 $";login,password,adminKey;{DataParsingExtension.ATableName};" +  
 data.Query.Split(DataParsingExtension.AdditionalQuerySplitter)[1], DBCommandType.**ValueModify**,  
 TCPConnectorService.Instance.GenericResult));  
 }  
  
 public void RequestModifySelf(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(  
 new DBCommand(data.Client, "as;" + data.Query  
 + $";login,password;{DataParsingExtension.ATableName};" +  
 GetClient(data.Client).ClientToStringVS(),  
 DBCommandType.**ValueModify**, RegisterModifySelf));  
 }  
  
 public void RegisterModifySelf(Object resultObj)  
 {  
 var result = (DBCommand)resultObj;  
  
 if (result.Query == "ERR")  
 {  
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 $"ef{DataParsingExtension.QuerySplitter}am1 Аккаунта не существует",  
 TCPCommandType.**SendSingleValue**));  
 return;  
 }  
  
 Client modifyMe = GetClient(result.Query  
 .Split(DataParsingExtension.QuerySplitter)[4]  
 .Split(DataParsingExtension.ValueSplitter)[0].HumanReadable());  
  
 Account modifiers =

Продолжение приложения Б

result.Query.Split(DataParsingExtension.QuerySplitter)[1].StringToAccountS();  
  
 modifyMe.Login = modifiers.Login.HumanReadable();  
 modifyMe.Password = modifiers.Password.HumanReadable();  
  
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 result.Query.Split(DataParsingExtension.QuerySplitter)[0] + DataParsingExtension.QuerySplitter +  
 modifyMe.Login + DataParsingExtension.ValueSplitter + modifyMe.Password,  
 TCPCommandType.**SendSingleValue**));  
 }  
  
 public void RequestDelete(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(new DBCommand(data.Client,  
 $"ad;{data.Query};login;{DataParsingExtension.ATableName}",  
 DBCommandType.**ValueDelete**,  
 TCPConnectorService.Instance.GenericResult));  
 }  
  
*// TODO Абсолютно небезопасно, ладно.* public Client? GetClient(String login)  
 {  
 return \_authorizedClients.FirstOrDefault(x => x.Login == login) == null  
 ? null  
 : \_authorizedClients.FirstOrDefault(x => x.Login == login);  
 }  
  
 public Client? GetClient(TcpClient tcpClient)  
 {  
 return \_authorizedClients.FirstOrDefault(x => x.Cocket == tcpClient) == null  
 ? null  
 : \_authorizedClients.FirstOrDefault(x => x.Cocket == tcpClient);  
 }  
  
 public void RequestGetAllAccounts(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(  
 new DBCommand(data.Client,  
 $"la{DataParsingExtension.QuerySplitter}\*{DataParsingExtension.QuerySplitter}{DataParsingExtension.ATableName}",  
 DBCommandType.**ValueGetAll**, TCPConnectorService.Instance.GenericGetAllResult));  
 }  
  
 public void RequestAddAccount(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(  
 new DBCommand(data.Client,  
 "aa;" + data.Query + $";login,password,adminKey;{DataParsingExtension.ATableName}",  
 DBCommandType.**ValueAdd**,  
 TCPConnectorService.Instance.GenericResult));  
 }  
  
 public void RequestGetAllAdminKeys(ACommand data) *//* {

Продолжение приложения Б

ServerApp.Instance.AddCommand(  
 new DBCommand(data.Client,  
 $"lk{DataParsingExtension.QuerySplitter}\*{DataParsingExtension.QuerySplitter}{DataParsingExtension.AKTableName}",  
 DBCommandType.**ValueGetAll**, TCPConnectorService.Instance.GenericGetAllResult));  
 }  
  
 public void RequestAddAdminKey(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(  
 new DBCommand(data.Client,  
 "aka;" + data.Query + $";adminKey;{DataParsingExtension.AKTableName}",  
 DBCommandType.**ValueAdd**,  
 TCPConnectorService.Instance.GenericResult));  
 }  
  
 public void RequestAdminKeyDelete(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(new DBCommand(data.Client,  
 $"akd;{data.Query};adminKey;{DataParsingExtension.AKTableName}",  
 DBCommandType.**ValueDelete**,  
 TCPConnectorService.Instance.GenericResult));  
 }  
  
 public void RequestAdminKeyModify(ACommand data) *//* {  
 ServerApp.Instance.AddCommand(  
 new DBCommand(data.Client,  
 "akm;" + data.Query.Split(DataParsingExtension.AdditionalQuerySplitter)[0] +  
 $";adminKey;{DataParsingExtension.AKTableName};" +  
 data.Query.Split(DataParsingExtension.AdditionalQuerySplitter)[1], DBCommandType.**ValueModify**,  
 TCPConnectorService.Instance.GenericResult));  
 }  
}

Процесс расчёта лечения растения (класс ZoneCalculator.cs):

using System.Globalization;  
using System.Numerics;  
using System.Text;  
  
namespace KursuchServer;  
  
class Zone(float radius, Vector2 center)  
{  
 public Vector2 Center = center;  
 public float Radius = radius;  
 public float liters = 0;  
}  
  
public class ZoneCalculator  
{  
 private static ZoneCalculator \_instance = null;  
  
 public static ZoneCalculator Instance  
 {

Продолжение приложения Б

get  
 {  
 if (\_instance == null) \_instance = new();  
 return \_instance;  
 }  
 private set { \_instance = value; }  
 }  
  
 public void Calculate(Command command)  
 {  
 *// Vector2 start = query.Split(DataParsingExtension.QuerySplitter)[0].StringToVector2();  
 // Vector2 end = query.Split(DataParsingExtension.QuerySplitter)[1].StringToVector2();* List<Vector2> points = new List<Vector2>();  
 List<Zone> zones = new List<Zone>();  
 float dosage = float.Parse(command.Query.Split(DataParsingExtension.AdditionalQuerySplitter)[2],  
 CultureInfo.InvariantCulture);  
 float workRadius = float.Parse(command.Query.Split(DataParsingExtension.AdditionalQuerySplitter)[3],  
 CultureInfo.InvariantCulture);  
  
 foreach (var point in command.Query  
 .Split(DataParsingExtension.AdditionalQuerySplitter)[1]  
 .Split(DataParsingExtension.QuerySplitter))  
 {  
 points.Add(point.StringToVector2());  
 zones.Add(new Zone(workRadius, points.Last()));  
 }  
  
 for (int i = 0; i < zones.Count; i++)  
 for (int j = 0; j < zones.Count; j++)  
 if (zones[i].Center == zones[j].Center) continue;  
 else  
 {  
 if (Vector2.Distance(zones[i].Center, zones[j].Center) <= zones[i].Radius + zones[j].Radius)  
 {  
 zones[i].Center.X = (zones[i].Center.X + zones[j].Center.X) / 2;  
 zones[i].Center.Y = (zones[i].Center.Y + zones[j].Center.Y) / 2;  
  
 zones[i].Radius += zones[j].Radius;  
  
 zones.Remove(zones[j]);  
  
 i = 0;  
 j = 0;  
 }  
 }  
  
 StringBuilder output = new();  
 output.Append("zd" + DataParsingExtension.AdditionalQuerySplitter + zones.Count.ToString() + DataParsingExtension.AdditionalQuerySplitter);

Продолжение приложения Б

foreach (var zone in zones)  
 {  
 zone.liters = zone.Radius \* zone.Radius \* (float)Math.**PI** \* dosage;  
 output.Append(zone.Center.X.ToString(CultureInfo.InvariantCulture)  
 + DataParsingExtension.ValueSplitter  
 + zone.Center.Y.ToString(CultureInfo.InvariantCulture)  
 + DataParsingExtension.QuerySplitter  
 + zone.Radius.ToString(CultureInfo.InvariantCulture)  
 + DataParsingExtension.QuerySplitter  
 + zone.liters.ToString(CultureInfo.InvariantCulture)  
 + DataParsingExtension.AdditionalQuerySplitter);  
 }  
 output.Remove(output.Length - 1, 1);  
 output.Append(DataParsingExtension.AdditionalQuerySplitter + dosage.ToString(CultureInfo.InvariantCulture) );  
 output.Append(DataParsingExtension.AdditionalQuerySplitter + workRadius.ToString(CultureInfo.InvariantCulture));  
 output.Append(DataParsingExtension.AdditionalQuerySplitter + command.Query.Split(DataParsingExtension.AdditionalQuerySplitter)[4]);  
 ServerApp.Instance.AddCommand(new TCPCommand(command.Client, output.ToString(),  
 TCPCommandType.**SendSingleValue**));  
 }  
}

# ПРИЛОЖЕНИЕ В (обязательное) Листинг основных элементов программы

Среверный ласс приёма и отправки данных с помощью TCP/IP (класс TCPConnectorService.cs):

using System.Net;  
using System.Net.Sockets;  
using System.Text;  
  
namespace KursuchServer.Services;  
  
public class TCPConnectorService  
{  
 private String \_ipAddressString;  
 private String \_port;  
 private TcpListener \_tcpListener;  
 private bool \_serverRunning = false;  
  
 private static TCPConnectorService instance = null;  
  
 public static TCPConnectorService Instance  
 {  
 get { return instance; }  
 private set { instance = value; }  
 }  
  
 public TCPConnectorService(String ipAddressString, String port)  
 {  
 instance = this;  
  
 \_ipAddressString = ipAddressString;  
 \_port = port;  
 }  
  
 public void StartServer() *//* {  
 \_serverRunning = true;  
 try  
 {  
 *// Запускаем сервер* \_tcpListener = new TcpListener(IPAddress.Any, int.Parse(\_port)); *//Parse(\_ipAddressString) int.Parse(\_port)* \_tcpListener.Start();  
 Console.WriteLine($"Сервер запущен на порту {\_port}. Ожидание подключений...");  
  
 *// Основной цикл обработки подключений* while (\_serverRunning)  
 {  
 if (\_tcpListener.Pending())  
 {  
 TcpClient client = \_tcpListener.AcceptTcpClient();  
Console.WriteLine($"Подключен клиент: {client.Client.RemoteEndPoint}");  
  
 Thread clientThread = new Thread(() => HandleClient(client));  
 clientThread.Start();

Продолжение приложения В

}  
 else  
 {  
 Thread.Sleep(100); *// Небольшая пауза, чтобы снизить нагрузку на CPU* }  
 }  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine($"Ошибка: {ex.Message}");  
 }  
 finally  
 {  
 StopServer();  
 }  
 }  
  
 public void StopServer() *//* {  
 \_serverRunning = false;  
  
 *// Останавливаем прослушивание* \_tcpListener?.Stop();  
  
 *// Закрываем все клиентские подключения  
 // lock (\_connectedClients)  
 // {  
 // foreach (var client in \_connectedClients)  
 // {  
 // client.Close();  
 // }  
 // \_connectedClients.Clear();  
 // }* Console.WriteLine("Сервер остановлен");  
 }  
  
 private void HandleClient(TcpClient tcpClient) *//* {  
 try  
 {  
 NetworkStream stream = tcpClient.GetStream();  
 byte[] buffer = new byte[1024];  
 int bytesRead;  
 bool loginAttempt = false;  
  
 while (IsClientConnected(tcpClient))  
 {  
 if (stream.DataAvailable)  
 {  
 bytesRead = stream.Read(buffer, 0, buffer.Length);  
 if (bytesRead == 0) break;  
  
 string data = Encoding.UTF8.GetString(buffer, 0, bytesRead);  
 Console.WriteLine($"{tcpClient.Client.RemoteEndPoint}:{data}");  
 if (!loginAttempt)  
 {  
 switch (data[0])

Продолжение приложения В

{  
 case 'l':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 data.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AccountLogin**));  
 break;  
 case 'r':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 data.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AccountRegister**));  
 break;  
 }  
 }  
  
 if (AccountService.Instance.GetClient(tcpClient) == null)  
 continue;  
  
 loginAttempt = true;  
  
 UserRequestProcess(data, tcpClient);  
 }  
 else  
 {  
 Thread.Sleep(100);  
 }  
 }  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine($"Ошибка с клиентом {tcpClient.Client.RemoteEndPoint}: {ex.Message}");  
 }  
 finally  
 {if (tcpClient.Connected) Console.WriteLine($"Клиент {tcpClient.Client.RemoteEndPoint} отключен");  
  
 var client = AccountService.Instance.GetClient(tcpClient);  
 if (client != null)  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 client.Login + ',' +  
 client.Password,  
 ACommandType.**AccountLogout**));  
  
 tcpClient.Close();  
 }  
 }  
  
 private void UserRequestProcess(String request, in TcpClient tcpClient)  
 {  
 switch (request[0])  
 {  
 case 'a':  
 switch (request[1])  
 {

Продолжение приложения В

case 'l':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**GetAllAccounts**));  
 break;  
 case 'd':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AccountDelete**));  
 break;  
 case 'o':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AccountLogout**));  
 break;  
 case 's':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AccountModifySelf**));  
 break;  
 case 'm':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AccountModify**));  
 break;  
 case 'a':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AccountAdd**));  
 break;  
 case 'k':  
 switch (request[2])  
 {  
 case 'l':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**GetAllAdminKeys**));  
 break;  
 case 'a':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AdminKeyAdd**));  
 break;  
 case 'd':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AdminKeyDelete**));  
 break;  
 case 'm':  
 ServerApp.Instance.AddCommand(new ACommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 ACommandType.**AdminKeyModify**));  
 break;  
 }  
  
 break;  
 }  
  
 break;  
 case 'p':  
 {  
 switch (request[1])  
 {  
 case 'l':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**LoadAll**));  
 break;  
 case 'z': *// disease* ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request,  
 PCommandType.**CalculateZones**));  
 break;  
 case 'd': *// disease* switch (request[2])  
 {  
 case 'l':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**DiseaseGetAll**));  
 break;  
 case 'a':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**DiseaseAdd**));  
 break;  
 case 'd':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**DiseaseDelete**));  
 break;  
 case 'm':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],

Продолжение приложения В

PCommandType.**DiseaseModify**));  
 break;  
 }  
  
 break;  
 case 'm':  
 switch (request[2])  
 {  
 case 'l':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**MedicineGetAll**));  
 break;  
 case 'a':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**MedicineAdd**));  
 break;  
 case 'm':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**MedicineModify**));  
 break;  
 case 'd':  
 if (request.Length >= 4)  
 switch (request[3])  
 {  
 case 'l':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**MedicineDiseaseGetAll**));  
 break;  
 case 'a':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**MedicineDiseaseAdd**));  
 break;  
 case 'd':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**MedicineDiseaseDelete**));  
 break;  
 }  
 else  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,

Продолжение приложения В

request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**MedicineDelete**));  
  
 break;  
 }  
  
 break;  
 case 'p':  
 switch (request[2])  
 {  
 case 'l':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantGetAll**));  
 break;  
 case 'a':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantAdd**));  
 break;  
 case 'm':  
 if (request.Length >= 4)  
 switch (request[3])  
 {  
 case 'l':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantMedicineGetAll**));  
 break;  
 case 'a':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantMedicineAdd**));  
 break;  
 case 'd':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantMedicineDelete**));  
 break;  
 }  
 else  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantModify**));

Продолжение приложения В

break;  
 case 'd':  
 if (request.Length >= 4)  
 switch (request[3])  
 {  
 case 'l':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantDiseaseGetAll**));  
 break;  
 case 'a':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantDiseaseAdd**));  
 break;  
 case 'd':  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantDiseaseDelete**));  
 break;  
 }  
 else  
 ServerApp.Instance.AddCommand(new PCommand(tcpClient,  
 request.Split(DataParsingExtension.QuerySplitter)[1],  
 PCommandType.**PlantDelete**));  
  
 break;  
 }  
  
 break;  
 }  
 }  
 break;  
 }  
 }  
  
 public void SendSingleValue(TCPCommand data) *//* {  
 NetworkStream stream = data.Client.GetStream();  
  
 *// Отправляем ответ* byte[] responseData = Encoding.UTF8.GetBytes(data.Query + '\n');  
 stream.Write(responseData, 0, responseData.Length);  
 stream.Flush();  
 }  
  
 public void SendMultipleValue(Object dataObj) *//* {  
 Command data = (Command)dataObj;

Продолжение приложения В

NetworkStream stream = data.Client.GetStream();  
  
 StringBuilder builder = new();  
 builder.Append(data.Query.Split(DataParsingExtension.QuerySplitter)[0] + DataParsingExtension.QuerySplitter);  
  
 var values = (List<String>)data.Output;  
 if (values.Count > 0)  
 {  
 foreach (var value in values)  
 builder.Append(value + DataParsingExtension.AdditionalQuerySplitter);  
 builder.Remove(builder.Length - 1, 1);  
 }  
   
 *// Отправляем ответ* byte[] responseData = Encoding.UTF8.GetBytes(builder.ToString());  
 stream.Write(responseData, 0, responseData.Length);  
 stream.Flush();  
 }  
  
 public void KillClient(TCPCommand data) *//* {  
 NetworkStream stream = data.Client.GetStream();  
  
 *// Отправляем ответ* byte[] responseData = Encoding.UTF8.GetBytes(data.Query + " is last words \n");  
 stream.Write(responseData, 0, responseData.Length);  
 Console.WriteLine($"Клиент {data.Client.Client.RemoteEndPoint} отключен");  
  
 data.Client.Close();  
 }  
  
 private bool IsClientConnected(TcpClient client)  
 {  
 if (!client.Connected) return false;  
  
 *// Точная проверка состояния соединения* try  
 {  
 if (client.Client.Poll(0, SelectMode.**SelectRead**))  
 {  
 byte[] dummy = new byte[1];  
 return client.Client.Receive(dummy, SocketFlags.**Peek**) != 0;  
 }  
  
 return true;  
 }  
 catch  
 {  
 return false;  
 }  
 }  
  
 public void GenericGetAllResult(Object resultObj)

Продолжение приложения В

{  
 var result = (DBCommand)resultObj;  
  
 if (result.Output == null) result.Query = "ERR";  
  
 if (result.Query == "ERR")  
 {  
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 $"es{DataParsingExtension.QuerySplitter}Ошибка операции с базой данных",  
 TCPCommandType.**SendSingleValue**));  
 return;  
 }  
  
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 result,  
 TCPCommandType.**SendMultipleValueLabeled**));  
 *//TCPConnectorService.Instance.SendMultipleValue(result);* }  
  
 public void GenericResult(Object resultObj)  
 {  
 var result = (DBCommand)resultObj;  
  
 if (result.Query == "ERR")  
 {  
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 $"es{DataParsingExtension.QuerySplitter}Ошибка операции с базой данных",  
 TCPCommandType.**SendSingleValue**));  
 return;  
 }  
  
 ServerApp.Instance.AddCommand(new TCPCommand(result.Client,  
 result.Query.Split(DataParsingExtension.QuerySplitter)[0] + DataParsingExtension.QuerySplitter,  
 TCPCommandType.**SendSingleValue**));  
 }  
}

Класс комманды на сервера (класс Command.cs):

using System.Net.Sockets;  
using System.Text;  
  
namespace KursuchServer;  
  
public class Command  
{  
 public CommandType Type;  
 public TcpClient Client;  
 public String Query;  
 public Object Output;  
 public Action<Object> OutputFunc;  
   
 public virtual void Execute()  
 { }  
 public virtual void Undo()  
 { }  
}

Продолжение приложения В

Класс расширений для настройки констант и методов парсинга типов данных (класс DataParsingExtension.cs):

using System.Globalization;  
using System.Numerics;  
using System.Text;  
using KursuchServer.DataStructures;  
  
namespace KursuchServer;  
  
public static class DataParsingExtension  
{  
 public static readonly char ValueSplitter = ',';  
 public static readonly char QuerySplitter = ';';  
 public static readonly char AdditionalQuerySplitter = '|';  
  
 public static readonly String ATableName = "Accounts";  
 public static readonly String AKTableName = "AdminKeys";  
   
 public static readonly string DiseasesTable = "Diseases";  
 public static readonly string MedicinesTable = "Medicines";  
 public static readonly string PlantsTable = "Plants";  
 public static readonly string MedicineDiseasesTable = "MedicineDiseases";  
 public static readonly string PlantsMedicinesTable = "PlantsMedicines";  
 public static readonly string PlantsDiseasesTable = "PlantsDiseases";  
  
 public static Vector2 StringToVector2(this string vector)  
 {  
 return new Vector2(float.Parse(vector.Split(ValueSplitter)[0], CultureInfo.InvariantCulture),  
 float.Parse(vector.Split(ValueSplitter)[1], CultureInfo.InvariantCulture));  
 }  
  
 public static String DBReadable(this String str)  
 {  
 return '\'' + str + '\'';  
 }  
  
 public static String NANIsNULL(this String str)  
 {  
 return str == " = 'NAN'" ? " IS NULL" : str;  
 }  
  
 public static String NANToNULL(this String str)  
 {  
 return str == "'NAN'" ? "NULL" : str;  
 }  
  
 public static String HumanReadable(this String str)  
 {  
 return str.Replace("\'", "");  
 }  
  
 public static String ClientToStringVS(this Client client)  
 {  
 return client.Login.DBReadable() + ValueSplitter +

Продолжение приложения В

client.Password.DBReadable();  
 }  
  
 public static Account StringToAccount(this String account)  
 {  
 return new Account(account.Split(ValueSplitter)[0], account.Split(ValueSplitter)[1],  
 account.Split(ValueSplitter)[2]);  
 }  
  
 public static Account StringToAccountS(this String account)  
 {  
 return new Account(account.Split(ValueSplitter)[0], account.Split(ValueSplitter)[1]);  
 }  
}

Сервис ответственный за взаимодействие с базой данных (класс DatabaseService.cs):

using System.Text;  
using Npgsql;  
  
namespace KursuchServer.Services;  
  
public class DatabaseService  
{  
 private String \_connectionString;  
  
 private static DatabaseService instance = null;  
  
 public static DatabaseService Instance  
 {  
 get  
 {  
 if (instance == null) return instance;  
 return instance;  
 }  
 private set { instance = value; }  
 }  
  
 public DatabaseService(String connectionString)  
 {  
 instance = this;  
 \_connectionString = connectionString;  
 }  
  
 *// 3 старые данные, 2 таблица, 1 колонки, 0 новые данные* public async Task<bool> ModifyValueAnyTable(String query)  
 {  
 try  
 {  
 await using var dataSource = NpgsqlDataSource.Create(\_connectionString);  
  
 var condition1 =  
 BuildSQLSequence(  
 query.Split(DataParsingExtension.QuerySplitter)[2].Split(DataParsingExte

Продолжение приложения В

nsion.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[1].Split(DataParsingExtension.ValueSplitter));  
  
 var condition2 =  
 BuildSQLCondition(  
 query.Split(DataParsingExtension.QuerySplitter)[2].Split(DataParsingExtension.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[4].Split(DataParsingExtension.ValueSplitter));  
  
 await using (var cmd = dataSource.CreateCommand(  
 $"UPDATE {query.Split(DataParsingExtension.QuerySplitter)[3]} SET {condition1} WHERE {condition2}"))  
 {  
 await cmd.ExecuteNonQueryAsync();  
 }  
  
 return true;  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine("Error when modifying data " + ex.Message);  
 return false;  
 }  
 }  
  
 public async Task<bool> CheckDataAnyTable(String query)  
 {  
 await using var dataSource = NpgsqlDataSource.Create(\_connectionString);  
  
 var condition =  
 BuildSQLCondition(  
 query.Split(DataParsingExtension.QuerySplitter)[1].Split(DataParsingExtension.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[0].Split(DataParsingExtension.ValueSplitter));  
  
 await using (var cmd = dataSource.CreateCommand(  
 $"SELECT {query.Split(DataParsingExtension.QuerySplitter)[1]} FROM {query.Split(DataParsingExtension.QuerySplitter)[2]} WHERE {condition}"))  
 await using (var reader = await cmd.ExecuteReaderAsync())  
 {  
 return reader.HasRows;  
 }  
 }  
  
 private static StringBuilder BuildSQLCondition(in string[] columnNames, in string[] dataToCheck)  
 {  
 StringBuilder condition = new();

Продолжение приложения В

for (int i = 0; i < dataToCheck.Length; i++)  
 condition.Append(columnNames[i] + (" = " + dataToCheck[i]).NANIsNULL() + " AND ");  
  
 condition.Remove(condition.Length - 5, 5);  
 return condition;  
 }  
  
  
 private static StringBuilder BuildSQLTabledCondition(in string[] columnNames, in string[] dataToCheck, string tableName)  
 {  
 StringBuilder condition = new();  
  
 for (int i = 0; i < dataToCheck.Length; i++)  
 condition.Append(tableName + '.' + columnNames[i] + (" = " + dataToCheck[i]).NANIsNULL() + " AND ");  
  
 condition.Remove(condition.Length - 5, 5);  
 return condition;  
 }  
  
 private static StringBuilder BuildSQLJoinCondition(in string[] columnNames, string table1, string table2)  
 {  
 StringBuilder condition = new();  
  
 for (int i = 0; i < columnNames.Length; i++)  
 condition.Append(table1 + '.' + columnNames[i] + " = " + table2 + '.' + columnNames[i] + " AND ");  
  
 condition.Remove(condition.Length - 5, 5);  
 return condition;  
 }  
  
 private static StringBuilder BuildSQLTabledColumns(in string[] columnNames, string table1)  
 {  
 StringBuilder condition = new();  
  
 for (int i = 0; i < columnNames.Length; i++)  
 condition.Append(table1 + '.' + columnNames[i] + " , ");  
  
 condition.Remove(condition.Length - 3, 3);  
 return condition;  
 }  
  
 private static StringBuilder BuildSQLSequence(in string[] columnNames, in string[] dataToCheck)  
 {  
 string query;  
 StringBuilder condition = new();  
  
 for (int i = 0; i < dataToCheck.Length; i++)  
 condition.Append(columnNames[i] + " = " + dataToCheck[i].NANToNULL() + ",");  
  
 condition.Remove(condition.Length - 1, 1);  
 return condition;  
 }

Продолжение приложения В

public async Task<List<String>> GetRowsOfAnyTable(String query) *//* {  
 List<String> items = new();  
 StringBuilder builder = new();  
 await using var dataSource = NpgsqlDataSource.Create(\_connectionString);  
  
 await using (var cmd = dataSource.CreateCommand(  
 $"SELECT {query.Split(DataParsingExtension.QuerySplitter)[1]} FROM {query.Split(DataParsingExtension.QuerySplitter)[2]}"))  
 {  
 await using (var reader = await cmd.ExecuteReaderAsync())  
 {  
 while (await reader.ReadAsync())  
 {  
 for (int i = 0; i < reader.FieldCount; i++)  
 {  
 if (reader.GetFieldType(i) == typeof(string))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetString(i) + ',');  
 else builder.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(int))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetInt32(i).ToString() + ',');  
 else builder.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(decimal))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetDecimal(i).ToString() + ',');  
 else builder.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(bool))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetBoolean(i).ToString() + ',');  
 else builder.Append("NAN,");  
 }  
 }  
  
 builder.Remove(builder.Length - 1, 1);  
  
 items.Add(builder.ToString());  
 builder.Clear();  
 }  
 }  
 }  
  
 return items;  
 }  
  
 public async Task<List<String>> GetRowsOfAnyTableCondition(String query) *//* {  
 List<String> items = new();

Продолжение приложения В

StringBuilder builder = new();  
 await using var dataSource = NpgsqlDataSource.Create(\_connectionString);  
  
 var condition =  
 BuildSQLCondition(  
 query.Split(DataParsingExtension.QuerySplitter)[4].Split(DataParsingExtension.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[3].Split(DataParsingExtension.ValueSplitter));  
  
 await using (var cmd = dataSource.CreateCommand(  
 $"SELECT {query.Split(DataParsingExtension.QuerySplitter)[1]} FROM {query.Split(DataParsingExtension.QuerySplitter)[2]} WHERE {condition}"))  
 {  
 await using (var reader = await cmd.ExecuteReaderAsync())  
 {  
 while (await reader.ReadAsync())  
 {  
 for (int i = 0; i < reader.FieldCount; i++)  
 {  
 if (reader.GetFieldType(i) == typeof(string))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetString(i) + ',');  
 else builder.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(int))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetInt32(i).ToString() + ',');  
 else builder.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(decimal))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetDecimal(i).ToString() + ',');  
 else builder.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(bool))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetBoolean(i).ToString() + ',');  
 else builder.Append("NAN,");  
 }  
 }  
  
 builder.Remove(builder.Length - 1, 1);  
  
 items.Add(builder.ToString());  
 builder.Clear();  
 }  
 }  
 }  
  
 return items;

Продолжение приложения В

}  
  
 public async Task<List<String>> GetRowsOfAnyTableJoined(String query) *//* {  
 List<String> items = new();  
 StringBuilder builder = new();  
 await using var dataSource = NpgsqlDataSource.Create(\_connectionString);  
  
 var columnsTabled =  
 BuildSQLTabledColumns(  
 query.Split(DataParsingExtension.QuerySplitter)[2].Split(DataParsingExtension.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[4]);  
 var joinCondition =  
 BuildSQLJoinCondition(  
 query.Split(DataParsingExtension.QuerySplitter)[3].Split(DataParsingExtension.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[4],  
 query.Split(DataParsingExtension.QuerySplitter)[5]);  
 var condition =  
 BuildSQLTabledCondition(  
 query.Split(DataParsingExtension.QuerySplitter)[2].Split(DataParsingExtension.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[1].Split(DataParsingExtension.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[4]);  
  
 await using (var cmd = dataSource.CreateCommand(  
 $"SELECT {columnsTabled} FROM {query.Split(DataParsingExtension.QuerySplitter)[4]} INNER JOIN {query.Split(DataParsingExtension.QuerySplitter)[5]} ON {joinCondition} WHERE {condition}"))  
 {  
 await using (var reader = await cmd.ExecuteReaderAsync())  
 {  
 while (await reader.ReadAsync())  
 {  
 for (int i = 0; i < reader.FieldCount; i++)  
 {  
 if (reader.GetFieldType(i) == typeof(string))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetString(i) + ',');  
 else builder.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(int))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetInt32(i).ToString() + ',');  
 else builder.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(decimal))  
 {  
 if (!reader.IsDBNull(i))

Продолжение приложения В

builder.Append(reader.GetDecimal(i).ToString() + ',');  
 else builder.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(bool))  
 {  
 if (!reader.IsDBNull(i)) builder.Append(reader.GetBoolean(i).ToString() + ',');  
 else builder.Append("NAN,");  
 }  
 }  
  
 builder.Remove(builder.Length - 1, 1);  
  
 items.Add(builder.ToString());  
 builder.Clear();  
 }  
 }  
 }  
  
 return items;  
 }  
  
 public async Task<String> GetValueAnyTable(String query) *//* {  
 StringBuilder value = new();  
 var condition =  
 BuildSQLCondition(  
 query.Split(DataParsingExtension.QuerySplitter)[1].Split(DataParsingExtension.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[0].Split(DataParsingExtension.ValueSplitter));  
  
 await using var dataSource = NpgsqlDataSource.Create(\_connectionString);  
  
 await using (var cmd = dataSource.CreateCommand(  
 $"SELECT \* FROM {query.Split(DataParsingExtension.QuerySplitter)[2]} WHERE {condition}"))  
 {  
 await using (var reader = await cmd.ExecuteReaderAsync())  
 {  
 while (await reader.ReadAsync())  
 {  
 for (int i = 0; i < reader.FieldCount; i++)  
 if (reader.GetFieldType(i) == typeof(string))  
 {  
 if (!reader.IsDBNull(i)) value.Append(reader.GetString(i) + ',');  
 else value.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(int))  
 {  
 if (!reader.IsDBNull(i)) value.Append(reader.GetInt32(i).ToString() + ',');  
 else value.Append("NAN,");  
 }  
 else if (reader.GetFieldType(i) == typeof(bool))  
 {

Продолжение приложения В

if (!reader.IsDBNull(i)) value.Append(reader.GetBoolean(i).ToString() + ',');  
 else value.Append("NAN,");  
 }  
  
 if (value.Length == 0) return "ERR";  
  
 value.Remove(value.Length - 1, 1);  
 return value.ToString();  
 }  
 }  
 }  
  
 if (value.Length == 0)  
 {  
 query = "ERR";  
 return query;  
 }  
  
 return value.ToString();  
 }  
  
 public async Task<bool> DeleteValueFromAnyTable(String query) *//* {  
 var condition =  
 BuildSQLCondition(  
 query.Split(DataParsingExtension.QuerySplitter)[2].Split(DataParsingExtension.ValueSplitter),  
 query.Split(DataParsingExtension.QuerySplitter)[1].Split(DataParsingExtension.ValueSplitter));  
  
 await using var dataSource = NpgsqlDataSource.Create(\_connectionString);  
 try  
 {  
 *// Если удалять несуществующую запись, то ошибки не будет* await using (var cmd = dataSource.CreateCommand(  
 $"DELETE FROM {query.Split(DataParsingExtension.QuerySplitter)[3]} WHERE {condition};"))  
 {  
 await cmd.ExecuteNonQueryAsync();  
 }  
  
 return true;  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine($"ew;{ex.Message}");  
 return false;  
 }  
 }  
  
 public async Task<bool> AddValueToAnyTable(String query) *//* {  
 try  
 {  
 await using var dataSource = NpgsqlDataSource.Create(\_connectionString);

Продолжение приложения В

await using (var cmd = dataSource.CreateCommand(  
 $"INSERT INTO {query.Split(DataParsingExtension.QuerySplitter)[3]} ({query.Split(DataParsingExtension.QuerySplitter)[2]}) VALUES ({query.Split(DataParsingExtension.QuerySplitter)[1]});"))  
 {  
 await cmd.ExecuteNonQueryAsync();  
 }  
  
 return true;  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine($"ew;{ex.Message}");  
 return false;  
 }  
 }  
}

}

Интерфейс для проверки корректности объектов (класс Validatable.java):

package by.mrtorex.businessshark.server.interfaces;

/\*\*

\* Интерфейс для проверки корректности объектов.

\*

\* @param <T> тип проверяемого объекта

\*/

public interface Validatable<T> {

/\*\*

\* Проверяет, является ли объект допустимым.

\*

\* @param t объект для проверки

\* @return true, если объект допустим; false в противном случае

\*/

boolean isValid(T t);

}

# ПРИЛОЖЕНИЕ Г (обязательное) Листинг скрипта генерации базы данных

**begin**;

-- Таблица болезней (все атрибуты зависят от первичного ключа)

**CREATE** **TABLE** **IF** **NOT** **EXISTS** Diseases (

diseaseId SERIAL **PRIMARY** **KEY**,

diseaseName **VARCHAR**(255) **NOT** **NULL** **UNIQUE**

);

-- Таблица лекарств (все атрибуты зависят от первичного ключа)

**CREATE** **TABLE** **IF** **NOT** **EXISTS** Medicines (

medicineId SERIAL **PRIMARY** **key** ,

medicineName **VARCHAR**(255) **NOT** **NULL** **unique**,

concentration **int** **not** **null**

);

-- Таблица растений (все атрибуты зависят от первичного ключа)

**CREATE** **TABLE** **IF** **NOT** **EXISTS** Plants (

plantId SERIAL **PRIMARY** **key** ,

plantName **VARCHAR**(255) **NOT** **NULL** **UNIQUE**

);

-- Новая связь M:M растения-болезни

**CREATE** **TABLE** **IF** **NOT** **EXISTS** PlantsDiseases (

plantId **INT** **NOT** **NULL** ,

diseaseId **INT** **NOT** **NULL** ,

**foreign** **key** (diseaseId) **references** Diseases(diseaseId) **ON** **DELETE** **cascade** **on** **update** **cascade**,

**foreign** **key** (plantId) **references** Plants(plantId) **ON** **DELETE** **cascade** **on** **update** **cascade**,

**PRIMARY** **KEY** (plantId, diseaseId)

);

-- Связующая таблица для связи многие-ко-многим между растениями и лекарствами

**CREATE** **TABLE** **IF** **NOT** **EXISTS** PlantsMedicines (

plantId **INT** **NOT** **NULL** ,

medicineId **INT** **NOT** **NULL** ,

**foreign** **key** (medicineId) **references** Medicines(medicineId) **ON** **DELETE** **cascade** **on** **update** **cascade**,

**foreign** **key** (plantId) **references** Plants(plantId) **ON** **DELETE** **cascade** **on** **update** **cascade**,

**PRIMARY** **KEY** (plantId, medicineId)

);

-- Связующая таблица для связи лекарств с болезнями и хранения дозировок

**CREATE** **table** **IF** **NOT** **EXISTS** MedicineDiseases (

diseaseId **INT** **NOT** **NULL** ,

medicineId **INT** **NOT** **NULL**,

minDosage **DECIMAL** **NOT** **NULL** **CHECK** (minDosage > 0),

maxDosage **DECIMAL** **NOT** **NULL** **CHECK** (maxDosage >= minDosage),

**foreign** **key** (medicineId) **references** Medicines(medicineId) **ON** **DELETE** **cascade** **on** **update** **cascade**,

**foreign** **key** (diseaseId) **references** Diseases(diseaseId) **ON** **DELETE** **cascade** **on** **update** **cascade**,

**PRIMARY** **KEY** (medicineId, diseaseId)

);

**CREATE** **TABLE** **IF** **NOT** **EXISTS** AdminKeys(

adminKey **VARCHAR**(255) **primary** **key** **not** **null**

Продолжение приложения Г

);

-- Таблица аккаунтов (отдельно от продавцов)

**CREATE** **TABLE** **IF** **NOT** **EXISTS** Accounts (

login **VARCHAR**(255) **NOT** **NULL** **PRIMARY** **KEY**,

password **VARCHAR**(255) **NOT** **NULL**,

adminKey **VARCHAR**(255),

**foreign** **key** (adminKey) **references** AdminKeys(adminKey) **ON** **DELETE** **cascade** **on** **update** **cascade**

);

-- Заполнение AdminKeys

**INSERT** **INTO** AdminKeys (adminKey) **VALUES**

('admin\_key\_123'),

('key');

-- Добавление аккаунтa администратора

**INSERT** **INTO** Accounts (login, password, adminKey) **VALUES**

('a', 'a', 'admin\_key\_123');

**end**;

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | | Наименование | | | Дополнитель-ные сведения | | | |
|  | | | | | Текстовые документы | | |  | | | |
|  | | | | |  | | |  | | | |
| БГУИР КР 6-05-0611-01 025 ПЗ | | | | | Пояснительная записка | | | 92 с. | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | | Графические документы | | |  | | | |
|  | | | | |  | | |  | | | |
| ГУИР 505403 001 ПД | | | | | Описание процесса выбора | | | Формат А4 | | | |
|  | | | | | лечения растения | | |  | | | |
|  | | | | |  | | |  | | | |
| ГУИР 505403 002 ПД | | | | | Схема алгоритма авторизации | | | Формат А4 | | | |
|  | | | | | пользователей в системе | | |  | | | |
|  | | | | |  | | |  | | | |
|  | | | | |  | | |  | | | |
| ГУИР 505403 003 ПЛ | | | | | Плакат Информационная модель | | | Формат А4 | | | |
|  | | | | | системы | | |  | | | |
|  | | | | |  | | |  | | | |
| ГУИР 505403 004 ПЛ | | | | | Плакат Модели представления | | | Формат А4 | | | |
|  | | | | | системы | | |  | | | |
|  | | | | |  | | |  | | | |
| ГУИР 505403 005 ПЛ | | | | | Плакат Результаты работы | | | Формат А4 | | | |
|  | | | | | приложения | | |  | | | |
|  |  |  |  |  | БГУИР КП 6-05-0611-01 025 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Программное средство для дистанционного управления опрыскивателем растений  Ведомость курсового  проекта | Лит | | | | Лист | Листов |
| Разраб. | | Еднач |  |  |  | Т | |  | 1 | 1 |
| Пров. | | Кабак |  |  | Кафедра ЭИ  гр. 324402 | | | | | |
| Т.контр. | | Кабак |  |  |
| Рец. | |  |  |  |
| Н.контр | |  |  |  |
| Утв. | | Кабак |  |  |
|  | |  |  |  |