РЕФЕРАТ

Тема бакалаврской работы «Разработка веб-сервиса для обработки данных полевых измерений».

Бакалаврская работа содержит --- страницы, --- рисунков, --- таблицы, --- источников.

Ключевые слова: дисперсионный анализ, статистика, анализ данных, веб-технологии, модель-представление-контроллер, MVC, model-view-controller.

Объектом исследования являются данные метеостанции и почвенных датчиков-влагомеров, требующие анализа и визуализации.

Предметом исследования являются алгоритм многофакторного дисперсионного анализа, а также методы предварительной обработки данных для проведения анализа.

Целью выпускной квалификационной работы является создание веб-приложения для обработки и визуализации данных, полученных с автоматической цифровой метеостанции и почвенных датчиков-влагомеров.

Были изучены методы дисперсионного анализа, нормирования данных для обработки ненормально распределённых данных, непараметрические методы, изучены существующие конкурентные решения, реализованы алгоритмы проведения многофакторного дисперсионного анализа, разработана собственная программа отображения результатов многофакторного дисперсионного анализа, а также алгоритм. После разработки было произведено тестирование программы.

В результате был разработана система для анализа данных на основе качественного признака.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 10](#_Toc152880518)

[Цель работы 11](#_Toc152880519)

[Задачи 11](#_Toc152880520)

[1 Использование многофакторного дисперсионного анализа для решения прикладных статистических задач 12](#_Toc152880521)

[1.1 Необходимость разработки программы для проведения статистических исследований 12](#_Toc152880522)

[1.2 Необходимость использования многофакторного дисперсионного анализа 14](#_Toc152880523)

[1.3 Нормирование данных методом box-cox, непараметрические методы 16](#_Toc152880524)

[2 Обзор существующих решений 16](#_Toc152880525)

[2.1 Описание популярных статистических пакетов 16](#_Toc152880526)

[2.2 Основной конкурент – R 23](#_Toc152880527)

[3 Выбор технологий для решения задачи 30](#_Toc152880528)

[3.1 Программа для статистических исследований Schicksal 30](#_Toc152880529)

[3.2 Технология программирования веб сервера и графического пользовательского интерфейса 30](#_Toc152880530)

[4 Программирование веб-сервера, графического пользовательского интерфейса и базы данных 36](#_Toc152880531)

[4.1 Аппаратное обеспечение проекта 36](#_Toc152880532)

[4.2 Программное обеспечение проекта 38](#_Toc152880533)

[5 Реализация алгоритма многофакторного дисперсионного анализа 54](#_Toc152880534)

[5.1 Описание структуры модуля многофакторного дисперсионного анализа 54](#_Toc152880535)

[5.2 Описание структуры модуля обработки данных методом box-cox 54](#_Toc152880536)

[5.3 Описание структуры модуля дисперсионного анализа с использованием непараметрических методов 54](#_Toc152880537)

[5.4 Разработка документации 54](#_Toc152880538)

[6 Тестирование программы 54](#_Toc152880539)

[7 Опытная и промышленная эксплуатация 54](#_Toc152880540)

[ЗАКЛЮЧЕНИЕ 55](#_Toc152880541)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 56](#_Toc152880542)

[ПРИЛОЖЕНИЕ А (листинг файла) 58](#_Toc152880543)

[ПРИЛОЖЕНИЕ Б (листинг файла) 58](#_Toc152880544)

[ПРИЛОЖЕНИЕ В (листинг файла indexcharts.js) 58](#_Toc152880545)

[ПРИЛОЖЕНИЕ Г (листинг файла AccountService.cs) 64](#_Toc152880546)

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

Математическая статистика – наука, разрабатывающая математические методы систематизации и использования статистических данных для научных и практических выводов.

Дисперсионный анализ (ANOVA, ANalysis Of VAriance) – метод в математической статистике, направленный на поиск зависимостей в экспериментальных данных путём исследования значимости различий в средних значениях.

Нормирование – метод, применяемый в многофакторном дисперсионном анализе с целью получения достоверных значений сумм квадратов на ненормально распределённых данных.

Генеральная совокупность – совокупность всех потенциально возможных вариантов, которые можно получить при одинаковых условиях.

Выборка – данные, которые получены при наблюдениях.

Репрезентативность – понятие, которое говорит, насколько показательна выборка, реалистично ли в ней распределены варианты. Выборка считается репрезентативной, если в ней учтено множество параметров и она достоверно отражает генеральную совокупность.

Распределение – показатель, описываемый через математические формулы. Он показывает частоту, с которой в выборке встречаются разные варианты. В результате можно сделать вывод о том, каких вариантов данных больше. Каких меньше – что больше или меньше распространено внутри выборки. Если она репрезентативная, это поможет сделать выводы и о генеральной совокупности.

Статистический пакет – программный продукт, предназначенный для статистической обработки данных. Как правило в статистический пакет входят средства для проведения описательных статистик, дисперсионного и регрессионного анализа, также инструменты для отображения результатов исследования, такие как графики и таблицы.

MVC – «модель-представление-контроллер» (model-view-controller), способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач.

Модель продукционного процесса – программа для ЭВМ, которая описывает поведение агроэкосистемы в зависимости от внешних условий (погоды, технологии возделывания, почвы, культуры).

Ансамблевый расчёт – один и тот же вычислительный эксперимент, осуществляемый на различных моделях продукционного процесса.

Сервер – персональный компьютер, который выполняет сервисные задачи без непосредственного участия человека.

Автоматическая цифровая метеостанция – цифровое устройство с набором датчиков, которые измеряют агрометеорологические характеристики окружающей среды с последующей передачей собранной датчиками информации по беспроводной сети на сервер.

СУБД – Система управления базами данных.

БД – База данных.

# ВВЕДЕНИЕ

Современные технологии в области информатики и компьютерных наук с течением времени активно интегрируются с человеческим бытом, в частности в области прогнозирования и моделирования бизнес и научных процессов. Возможности современных компьютерных систем позволяют собирать, хранить и обрабатывать большие массивы данных и, в конечном итоге, анализировать их и находить закономерности в различных эмпирических наблюдениях.

Также помимо проведения самого статистического анализа необходимо предоставить пользователю наглядные результаты для того, чтобы он непосредственно смог проинтерпретировать результаты и сделать выводы. В качестве инструмента визуализации был выбран веб-сервис на основе технологий ASP.NET C# MVC, с помощью которого можно запрограммировать сервер и разместить на нём веб-сайт, предоставив пользователю удобный графический интерфейс и наглядные динамические графики. Выбор таких технологий обусловлен тем, что C# MVC является популярной платформой для разработки, обладающей большой базой документаций и тематических форумов, вследствие чего проект будет легко масштабироваться и поддерживаться. Также проект является кросс-платформенным, что позволяет разместить сервер на любом аппаратном обеспечении, просто перекомпилировав его под необходимую архитектуру процессора.

Объектом исследования являются данные автоматической цифровой метеостанции и почвенных датчиков-влагомеров, требующие анализа и визуализации.

Предметом исследования являются алгоритм многофакторного дисперсионного анализа, а также методы предварительной обработки данных для проведения анализа.

## Цель работы

Разработка многофакторного дисперсионного анализа, создание алгоритма предварительной обработки данных, разработка веб-сервиса для проведений исследований.

## Задачи

* Определение достоинств и недостатков существующих статистических пакетов;
* Выбор технологий и методов создания приложения для проведения исследований и обработки данных;
* Реализация алгоритмов предварительной обработки данных и алгоритмов проведения многофакторного дисперсионного анализа;
* Создание пользовательского интерфейса приложения для проведения исследования и обработки данных;
* Оценка эффективности работы приложения для проведения исследования и обработки данных.

## 1 Использование многофакторного дисперсионного анализа для решения прикладных статистических задач

## Необходимость разработки программы для проведения статистических исследований

Статистика – наука, которая разрабатывает математические методы систематизации и применения статистических данных для практических и научных выводов. Отрасль тесно связана с математическим аппаратом и с теорией вероятности: часто статистика использует те же формулы и методы, но статистика решает задачи, которые являются противоположными задачам из теории вероятностей. Статистика помогает описывать данные, анализировать их, строить прогнозы. Одним из самых известных и наукоёмких инструментов, применяемых для анализа данных в абсолютно любых предметных областях является информационная система для проведения статистического анализа, так называемый «Статистический пакет».

Смысл этих программ состоит в том, чтобы на основе собранных эмпирических наблюдений в виде таблицы провести статистический анализ соответствующих данных, затем результаты анализа необходимо проинтерпретировать. Статистический анализ может быть реализован с помощью описательных статистик – нахождение среднего, медианы, стандартного отклонения, доверительного интервала и т.д., можно также провести дисперсионный анализ, для исследования наличия или отсутствия существенного влияния какого-либо качественного или количественного признака. Также в классической статистике применяется регрессионный, кластерный и дискриминантный анализ, выбор статистического метода зависит от входных данных и потребностей исследователя, но эти методы не являются объектами текущего исследования.

Сельское хозяйство – это одна из тех предметных областей, где обосновано применение статистических методов из-за того, что в этой науке получаются точные «осязаемые» результаты, в зависимости от исходных условий, при исследовании объекта, такие как: вес растения, количество удобрений и полива, объем корневой системы, тип культуры и т.д.. На сегодняшний день в сельском хозяйстве есть множество статистических и динамических моделей продукционного процесса [Зеленая книга Полуэктов]. Такие программы решают проблемы управления сельским хозяйством, создавая математическую модель развития какой-либо культуры, с момента посадки до момента всходов.

Федеральное государственное бюджетное научное учреждение «Агрофизический научно-исследовательский институт» занимается разработками в сфере сельского хозяйства, и лаборатория математического моделирования отдела 170 «Моделирования адаптивных технологий» поставила задачу – разработка методики интеграции интеллектуальной системы прогноза роста и развития растений с оптимизированной сетевой архитектурой гибридной беспроводной сенсорной сети для решения задач растениеводства на основе системы оперативного мониторинга данных метеостанции.

Цели заказчика:

* Формирование в удобной для проведения ансамблевых расчетов данных автоматической цифровой метеостанции, установленной на сельскохозяйственном поле, на веб-сайте, доступ к которому осуществляется через интернет

Задачи заказчика:

* Разработка информационной системы по сбору, хранению и обработке агрометеорологических данных, поступающих с автоматической цифровой метеостанции, установленной на сельскохозяйственном поле;
* Развертывание информационной системы на сервере;
* Размещение сайта (представления) системы в интернете.

В последующих разделах текущего исследование будет приведено детальное описание технических характеристик инструментов, предоставленных заказчиком, а также разработанного продукта.

## Необходимость использования многофакторного дисперсионного анализа

Дисперсионный анализ (ANOVA – ANalysis Of VAriance) – статистический метод изучения взаимосвязи. Применяется для исследования влияния одной или нескольких качественных переменных на одну зависимую количественную. В основе дисперсионного анализа лежит предположение, что одни переменные могут рассматриваться как причины (независимые переменные), а другие как следствия (зависимые переменные или отклики). Независимые переменные в дисперсионном анализе называют факторами, поскольку в ходе эксперимента исследователь может заменять их значения и анализировать получаемый результат зависимой количественной переменной.

Цель дисперсионного анализа – исследование наличия или отсутствия существенного влияния какого-либо качественного или количественного фактора на изменения исследуемого результативного признака. Для этого фактор, предположительно имеющий или не имеющий существенного влияния, разделяют на классы градации (говоря иначе, группы) и выясняют, одинаково ли влияние фактора путём исследования значимости между средними в наборах данных, соответствующих градациям фактора. Например: исследуется зависимость получаемого процента влажности почвы прибором-влагомером от версии этого прибора (тогда классы градации – версия прибора), или исследуется урожайность сельскохозяйственного поля от количества внесённых удобрений, количества воды при автоорошении и типа почвы (тогда классы градации – количество удобрений, количество воды при автоорошении и тип почвы) Минимальное число классов градации (групп) – два. Классы градации могут быть качественными либо количественными.

Дисперсионный анализ – почти универсальный метод проверки различий в группах, поскольку применяется как в технических науках и сельском хозяйстве, так и в исследованиях поведения человека.

При дисперсионном анализе исследуется отношение двух дисперсий. Дисперсия – характеристика рассеивания данных вокруг среднего значения. Первая – дисперсия, объяснённая влиянием фактора (группы) вокруг средней всех данных. Вторая – необъяснённая дисперсия, которая характеризует рассеивание данных внутри градаций (групп) вокруг средних значений самих групп. Первую дисперсию можно назвать межгрупповой, а вторую – внутригрупповой. Отношение этих дисперсий называется фактическим отношением Фишера и сравнивается с критическим значением отношения Фишера. Если фактическое отношение Фишера больше критического, то средние классов градации отличаются друг от друга и исследуемый фактор существенно влияет на изменение данных. Если меньше, то средние градаций не отличаются друг от друга, и фактор не имеет существенного влияния.

Данный вид статистического анализа можно провести на данных полученных лабораторией «Сектор средств инструментального контроля», где разрабатываются приборы для сбора информации о агроэкосистеме с полей, такие как: датчики-влагомеры, которые заглубляются в почву и получают информацию о влажности всех слоев почвы в реальном времени; цифровая метеостанция, которая может измерять в автоматическом режиме температуру поверхности земли, солнечное излучение, направление и скорость ветра, и т.д..

Данные приборы нуждаются в верификации – проверки гипотезы о том, что те метрики, которые были получены в цифровом виде приборами, соответствуют реальным измерениям, полученными вручную опытным путём. Также необходимо нахождение существенных различий между некими качественными параметрами приборов, например – поколение разработанного прибора. Для решения этих задач метод многофакторного дисперсионного анализа очень хорошо подходит.

## Нормирование данных методом box-cox, непараметрические методы

## 2 Обзор существующих решений

## 2.1 Описание популярных статистических пакетов

Компьютерные программы для статистики разрабатывались еще в прошлом веке, в 1968 году в Чикагском университете была разработана программа SPSS, затем патент на эту программу был приобретён компанией IBM [https://ru.wikipedia.org/wiki/SPSS]. С развитием цифровых технологий развивались и статистические пакеты, эволюционируя из простых консольных приложений с функционалом базовых описательных статистик в программы, которые можно назвать отдельным языком программирования с многочисленным количеством библиотек для всевозможных анализов и экспорта результатов в виде графиков и таблиц, как например язык R и программа RStudio.

Программ статистических пакетов разработано очень много, они различаются по своей форме – могут быть разные графические оболочки для разных операционных систем, или же графической оболочки может не быть вовсе, но в любом случае все статистические пакеты принимают на вход пользовательские данные о наблюдениях и типе анализа, и в виде выходных данных выдают результаты анализа по наблюдениям. Как правило статистический пакет представляет собой клиентское приложение, устанавливаемое на пользовательский компьютер, все расчеты выполняются локально. Схема работы типичного статистического пакета отображено на рисунке 635:

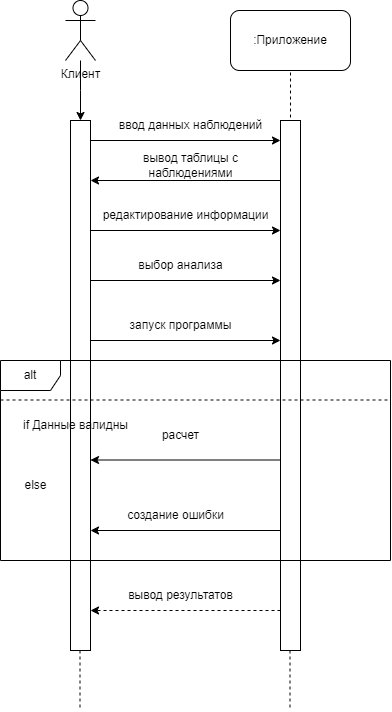


Рисунок 635 – Uml-диаграмма последовательности работы статистического пакета

Безусловно, описанные выше принципы работы статистических пакетов, в их разнообразии приобретают новые черты и особенности с развитием информационных технологий и новых методов измерения. Даже несмотря на то, что почти все статистические методы были выведены математиками задолго до появления первых компьютеров, с течением времени появляются новые методики проведения исследований, а также появляются новые стандарты для программных продуктов.

Важным аспектом отрасли программ для статистики является тот факт, что данные программы рассчитаны на небольшую аудиторию, вследствие этого разработчики редко обновляют, или же вообще не обновляют свое программное обеспечение, и получается так, что они выглядят не только морально устаревши, но и вообще могут не работать на современных операционных системах. С проприетарным программным обеспечением ситуация на сегодняшний день выглядит ещё хуже – данные продукты вообще ушли с российского рынка. Вследствие этих фактов разработка и развитие информационной системы статистического пакета выглядит крайне перспективной.

Одним из наиболее популярных статистических пакетов в мире является R. R – это программа для вычислений, созданная исследователями для обработки и анализа данных, работы с графикой и математического моделирования. Функционал R достаточно велик: его можно использовать как просто калькулятор, можно обрабатывать данные в табличном виде, еще можно проводить статистические исследования, например узнать p – уровень значимости, провести дисперсионный или регрессионный анализ. Также можно проводить более сложные вычисления, строить тепловые карты и графики. Также немаловажным преимуществом этой программы является кроссплатформенность. R – это продукт со свободной лицензией. Также R является самостоятельным языком программирования.

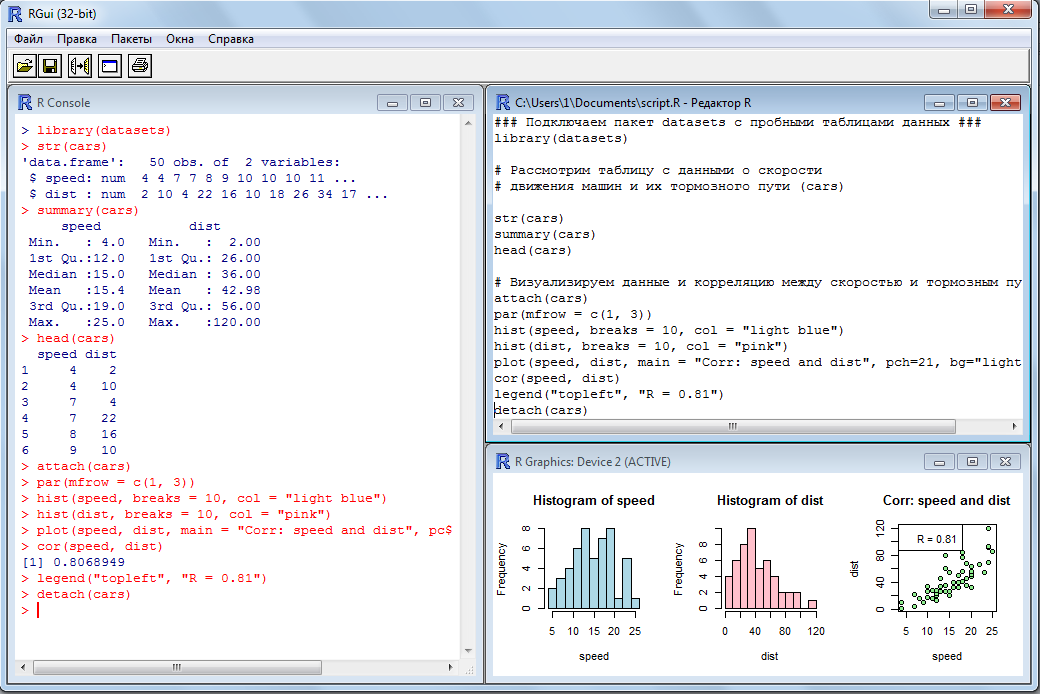


Рисунок 18 – Внешний вид IDE R

На сегодняшний день существует очень много качественных статистических пакетов, среди которых самыми лучшими являются MatLab, SPSS и SAS. Но в 2013 году, несмотря на высочайшую конкуренцию, язык R стал самой используемой программой для статистического исследования в научных публикациях. Достоинство языка R – это его кросплатформенность, бесплатная лицензия, и большой репозиторий пакетов. В Schicksal нет такой поддержи от сообщества, также он работает только в операционной системе Windows, но зато есть инструменты для многофакторного регрессионного анализа, которых нету в R

IBM SPSS Statistics – это ПО для эмпирических исследований, позволяющее проводить статистический анализ деловых данных на продвинутом уровне. Также эта программа может решать задачи построения бизнес – отчётности, планирования и сбора информации.

Программа IBM SPSS Statistics предназначена для статистического анализа данных, позволяя делать выводы по исследуемым данным. IBM SPSS имеет большую популярность среди различных компаний, центрами принятия решений и независимыми исследователями для решения специфических задач.

В числе особенностей программы SPSS Statistics можно выделить:

* Настройка функциональности для любого пользователя;
* SPSS позволяет предварительно обработать данные и вывести результаты в любом удобном виде.
* Удобный пользовательский интерфейс позволяет ориентироваться в озерах данных и работать одновременно сразу с несколькими окнами.
* Есть интеграция с языками R и Python,
* Также система является масштабируемой.

Огромный минус этой программы – это то, что она является платной, также в ней нет инструментов многофакторного дисперсионного анализа.

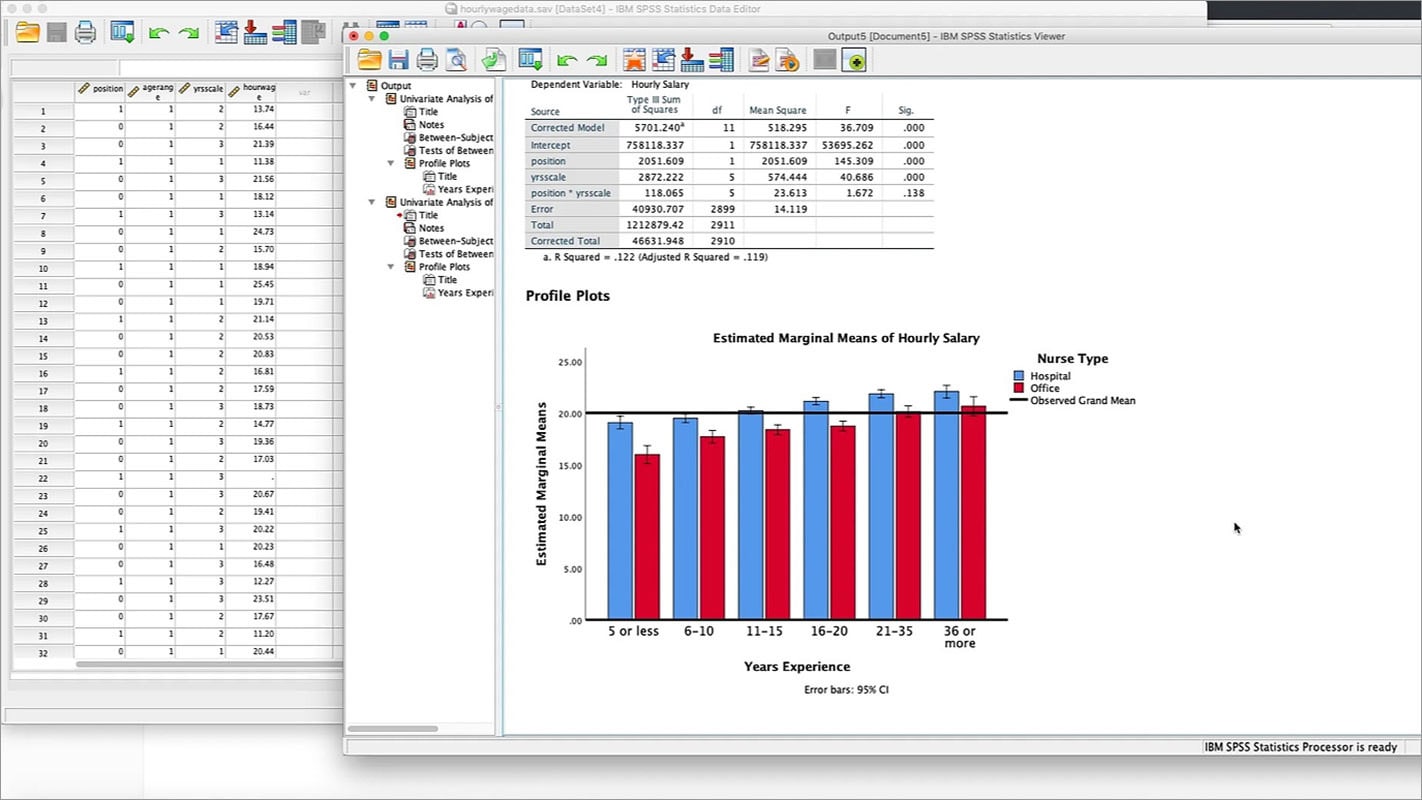


Рисунок 19 – Внешний вид программы SPSS

Пакет STATISTICA разработан корпорацией StatSoft. Изначально пакет входил в качестве плагина в состав других известных в то время электронных таблиц Lotus 1-2-3. В 1991 году программа стала самостоятельным продуктом. Последняя версия приложения была совместима только с Vista, в ней есть GUI и инструмент динамического обмена данными, что позволяет работать пакету с другими Windows – приложениями. В последние версии дистрибутива также включен язык программирования Statistica – Basic. Программа занимает на диске всего 18 Мб и требует всего 4 Мб ОЗУ. Возможности этой программы поистине всеобъемлющие: она может строить уравнение регрессии и делать вычисления по описательным статистикам. Также обладает весьма привлекательным пользовательским интерфейсом в своё время. Также в программе встроен очень удобный модуль подсказок, который позволяет обучаться в процессе исследования

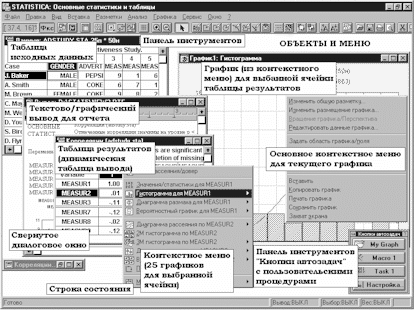


Рисунок 20 – Внешний вид программы STATISTICA

Файлы представляют собой электронные таблицы, похожие на Excel файлы. Файл содержит переменные и наблюдения. Переменные это как поля в базе данных, а наблюдения как записи. Пакет STATISTICA также обладает очень удобным графическим интерфейсом, тоже как в Excel, позволяя делать множественное выделение, перемещение и организацию всплывающего меню по правому клику мыши.

Общее число переменных в файле не может быть больше 4092, а количество наблюдений может быть сколько угодно, также программа позволяет проводить импорт и экспорт данных.

Минусом этого статистического пакета является то, что программа давно не обновляется и является морально и визуально устаревшей на сегодняшний день, но её неоспоримое преимущество – это импорт данных из Excel.

SAS – пакет для статистического анализа, созданный в 1960 г. Этот пакет широко использовался исследователями в американских университетах и компаниях. Продукт является кроссплатформенным, что является очень значительным преимуществом, так как в 60ых годах не было программы для статистического анализа на Macintosh. Этот статистический пакет развивается и по сей день, в него было добавлено множество нововведений такие как: отрисовка диаграмм в реальном времени, возможность работать с потоками данных, параллельная обработка данных и т.д.



Рисунок 21 – Внешний вид программы SAS

Этот статистический пакет был очень распространен в российском бизнесе, но из – за политических санкций этот пакет больше недоступен для скачивания и покупки в России.

## 2.2 Основной конкурент – R

Язык программирования R обладает огромным инструментарием для проведения статистических исследований, собственно, он и позиционируется как язык программирования для статистиков. Графическая оболочка в RStudio разделена на 4 области: область кода, где пользователь пишет инструкции к выполнению и затем последовательно их выполняет, также он может сохранить этот файл с расширением \*.r, чтобы потом заново открыть написанный код

* область кода, где пользователь пишет инструкции к выполнению и затем последовательно их выполняет, также он может сохранить этот файл с расширением \*.r, чтобы потом заново открыть написанный код;
* область консоли и терминала. В консоли можно тоже выполнять инструкции, но они не будут сохранены в файл скрипта R, также в консоли выводится информация о выполненных командах, например когда пользователь устанавливает библиотеку в консоли выводится информация о загрузке, или когда пользователь выполняет инструкцию по выполнению описательных статистик для своих данных, в консоль выводится таблица с описательными статистиками (среднее, медиана, количество наблюдений, первый и третий квартиль и т.д.). В терминале пользователь может открыть командную строку своего компьютера, например git bash для выполнения других функций, связанных с операционной системой или другой программой;
* область окружения. В этой области отображаются переменные, которые объявил пользователь в своей сессии, их можно раскрыть и посмотреть информацию о содержимом списков и таблиц, также в этой области можно посмотреть историю выполнения команд;
* область графиков, пакетов и окно помощи. В этой области отображаются графики такие как гистограммы, ящики с усами и уравнения регрессии, эти графики можно экспортировать в виде изображений на свой жёсткий диск. В окне пакетов отображаются все пакеты и библиотеки, которые можно установить, установленные пакеты помечаются галочками. Пакеты и библиотеки – это наборы дополнительных инструментов для проведений исследований, например библиотека графиков “ggplot2” нужна, если стандартные инструменты визуализации R не подходят. Окно помощи – это инструмент для поиска документации и примеров использования инструкций. В этом окне можно найти описание для всех функций R.

В рамках текущего исследования будет рассмотрен функционал языка R проведения дисперсионного анализа. Далее приведен пример проведения анализа на данных, где есть 3 фактора и одна зависимая переменная-отклик.

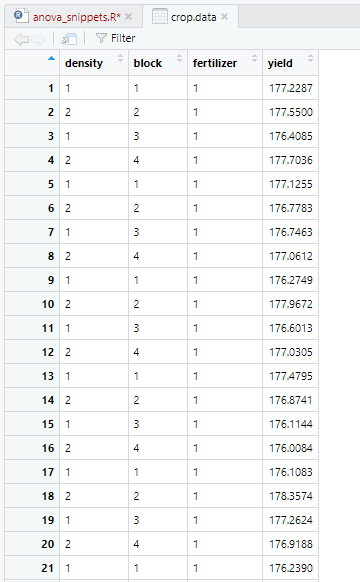


Рисунок XXX – таблица с данными, где density – фактор с 2 уровнями; block – фактор с 4 уровнями; fertilizer – фактор с 3 уровнями, yield – переменная-отклик

Для начала необходимо произвести описательные статистики командой summary(crop.data), для того чтобы понять, что за данные перед нами.

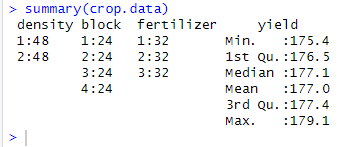


Рисунок XXX – результат проведения описательных статистик

Колонки density, block и fertilizer показывают сумму наблюдений по каждому уровню фактора. Колонка yield отображает базовую статистику: min – минимальное значение, 1st Qu – первый квартиль (25% от всех наблюдений), Median – медиана, Mean – среднее, 3rd Qu – третий квартиль (75% от всех наблюдений), Max – максимальное значение.

Дисперсионный анализ можно проводить только на нормально распределённых данных, если данные имеют другой характер распределения, то p уровень значимости и F-критерий Фишера дадут неверный результат. Для того чтобы узнать имеют ли данные нормальное распределение, необходимо провести тест Шапиро-Уилка с поощью команды shapiro.test(crop.data$yield).

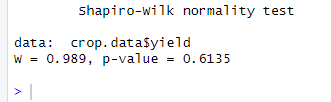


Рисунок XXX – результат проведения тест Шапиро-Уилка

Если p уровень значимости больше критического 0.05, то можно утверждать, что распределение данных существенно не отличается от нормального распределения. Другими словами мы можем предположить нормальность, и продолжить проведение дисперсионного анализа.

Для проведения однофакторного дисперсионного анализа необходимо воспользоваться стандартной функцией R aov(): one.way.fert <- aov(yield ~ fertilizer, data=crop.data), где первый параметр – это результирующая переменная и фактор, который мы хотим исследовать, второй параметр – источника данных (рисунок XXX).

Результаты проведения анализа отображаются инструкцией summary(one.way.fert).

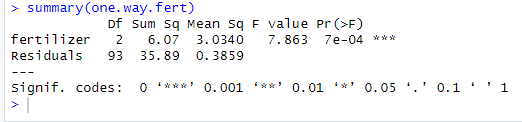


Рисунок XXX – результаты проведения однофакторного дисперсионного анализа

В сводке сначала перечислены независимые переменные, подставленные в модели (в данном случая есть только одна – fertilizer), а также остатки (Residuals) – внутригрупповая дисперсия (её ещё называют шумовой или остаточной дисперсией – в том смысле, что она не может быть объяснена влиянием экспериментального фактора). Остальные значения в выходной таблице описывают независимую переменную и остатки, где:

* Df – количество степеней свободы (в данном случае количество уровней фактора минус один). Df остатков – общее количество наблюдений минус один и минус количество уровней независимых переменных;
* Sum Sq – сумма квадратов отклонений (общее различие между средними значениями группы и общим средним значением);
* Mean Sq – среднее значение суммы квадратов отклонений, рассчитанное путём деления суммы квадратов отклонений на количество степени свободы;
* F value – фактическое значение Фишера (среднее значение суммы квадратов отклонений независимой переменной делённое на среднее значение суммы квадратов отклонений остатков).

p уровень значимости достаточно низкое (p < 0.001), следовательно можно утверждать, что фактор fertilizer оказывает значимое влияние на переменную yield, однофакторный дисперсионный анализ проведён успешно.

Для проведения двухфакторного дисперсионного анализа необходимо использовать ту же функцию aov, но с изменёнными параметрами: two.way <- aov(yield ~ fertilizer \* density, data = crop.data). Знак ”\*” указывает на то, что необходимо узнать влияние каждого фактора по отдельности, а также влияние взаимодействия факторов (fertilizer и density).

Результат анализа вызывается инструкцией summary(two.way):

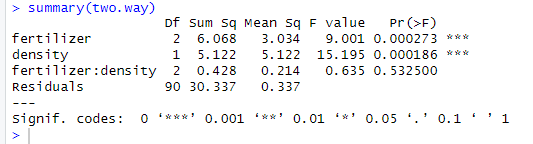


Рисунок XXX – результат выполнения двухфакторного дисперсионного анализа

В выходной таблице переменная “fertilizer:density” имеет низкое значение суммы квадратов отклонений и высокий p уровень значимости, что что означает что взаимодействие факторов fertilizer \* density не оказывает существенного влияния на результирующую переменную yield. Двухфакторный дисперсионный анализ проведён успешно.

Также для наглядности можно визуализировать данные в виде графиков, для наглядности результатов.

Для построения гистограммы частот используется инструкция hist():

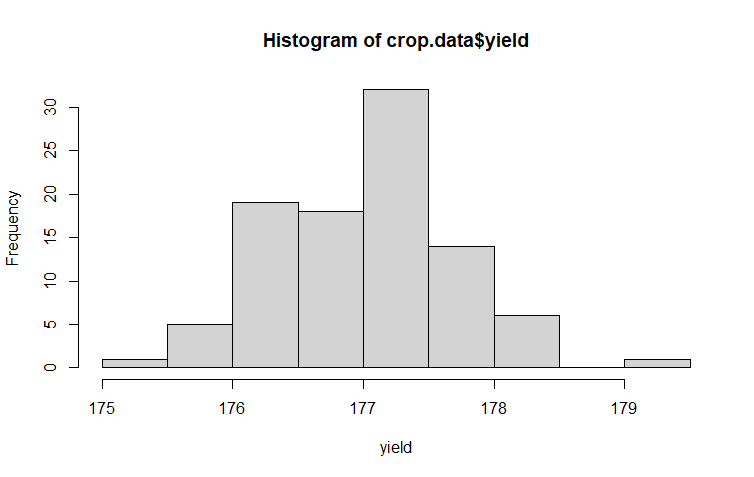


Рисунок XXX – гистограмма частот данных

На данной гистограмме отображена частота значений переменной yield из таблицы данных, визуально данные распределены нормально. При оценки дисперсии данных часто используется график “ящик с усами”, данный график есть в языке R и отображается выполнением инструкции boxplot():

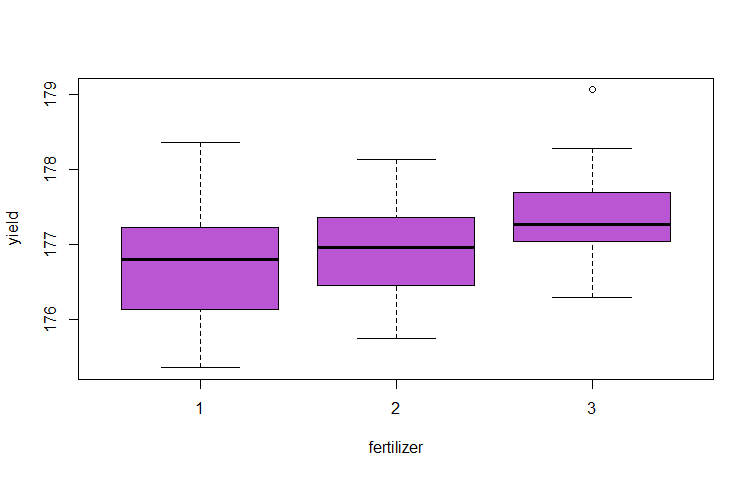


Рисунок XXX – “диаграмма ящик с усами” (boxplot)

Многофакторный дисперсионный анализ и понятие взаимодействия факторов.

## 3 Выбор технологий для решения задачи

## 3.1 Программа для статистических исследований Schicksal

## 3.2 Технология программирования веб сервера и графического пользовательского интерфейса

Далее пойдут ссылки на литературу из отчёта Баденко

В задачу разработки входила необходимость предоставления пользователю доступа к просмотру и использованию указанных выше данных метеостанции. Для этого необходимо создать клиент-серверную архитектуру, где на стороне сервера будут обрабатываться и храниться данные, а на стороне клиента данные будут представляться. Подобную проблему решает архитектурный паттерн MVC [3, 4]. Термин «модель-представление-контроллер» (model-view-controller – MVC) был в употреблении с конца 1970-х годов и происходит из проекта Smalltalk в Xerox PARC [5], где он был задуман как способ организации ряда ранних приложений с графическим пользовательским интерфейсом, и нюансы первоначального паттерна MVC применимы к приложениям – и особенно хорошо к веб-приложениям. Отметим, что в целом MVC хорошо проявила себя при создании веб-приложений для сельского хозяйства [6].

Если оперировать высокоуровневыми понятиями, то паттерн MVC означает, что приложение MVC будет разделено, по крайней мере, на три части:

* Модели, содержащие или представляющие данные, с которыми работают пользователи;
* Представления, используемые для визуализации некоторой части модели в виде пользовательского интерфейса;
* Контроллеры, которые обрабатывают входящие запросы, выполняют операции с моделью и выбирают представления для визуализации пользователю.

Каждая порция архитектуры MVC четко определена и самодостаточна: такое положение вещей называют разделением обязанностей. Логика, которая манипулирует данными в модели, содержится только в модели. Логика, отображающая данные, присутствует только в представлении. Код, который обрабатывает пользовательские запросы и ввод, находится только в контроллере. Благодаря ясному разделению между частями приложение будет легче сопровождать и расширять на протяжении времени его существования вне зависимости от того, насколько большим оно станет. Модели (M в MVC) содержат данные, с которыми работают пользователи. Существуют два обширных типа моделей: модели представлений, которые выражают сами данные, передаваемые из контроллера в представление, и модели предметной области, которые содержат данные в предметной области наряду с операциями, трансформациями и правилами для создания, хранения и манипулирования данными, вместе называемыми логикой модели. Предметная область в текущем проекте – это агрометеорологические данные.

Uml-диаграмма развёртывания информационной системы по сбору, хранению и обработке агрометеорологических данных представлена на рисунке 1.1.

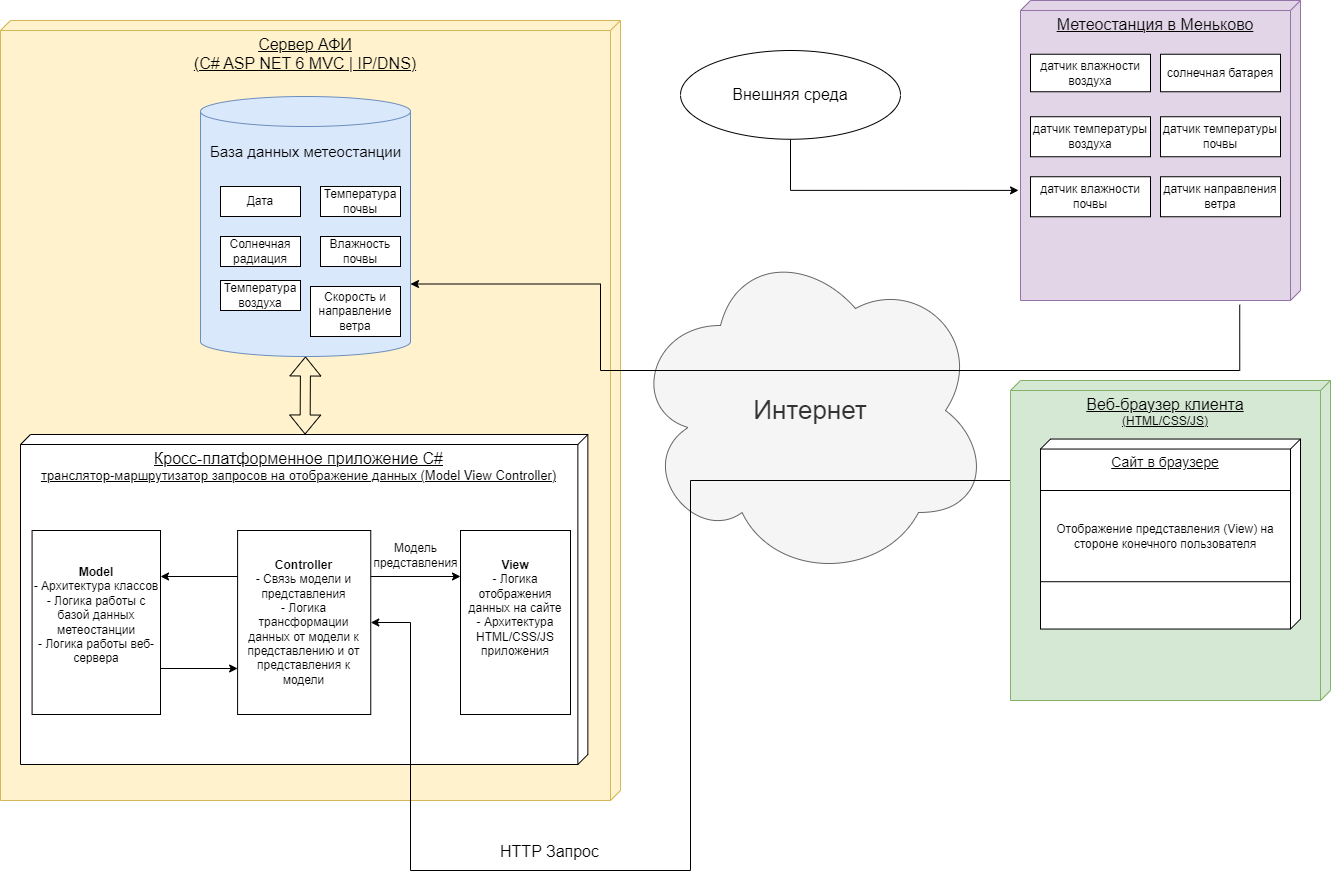


Рисунок 1.1 – Uml-диаграмма развёртывания информационной системы по сбору, хранению и обработке агрометеорологических данных

Контроллеры являются «соединительной тканью» паттерна MVC, исполняя роль каналов между моделью данных и представлениями. Контроллеры определяют действия, предоставляющие бизнес-логику, которая оперирует на модели данных и обеспечивает представления данными, подлежащими отображению для пользователя.

Представления содержат логику, которая требуется для отображения данных пользователю или для сбора данных от пользователя, так что они могут быть обработаны каким-то действием контроллера.

Данный паттерн реализован на платформе ASP.NET 6 [7], то есть он является кроссплатформенным и реализует библиотеку классов, наследуемую от Microsoft.AspNetCore.Mvc [8].

MVC – не единственный архитектурный паттерн программного обеспечения. Есть много других паттернов, и некоторые из них чрезвычайно популярны или, по крайней мере, были таковыми. Преимущества MVC можно обнаружить, сравнивая его с альтернативами.

Один из наиболее распространенных паттернов проектирования известен как «Интеллектуальный пользовательский интерфейс» (Smart UI). Большинству программистов приходилось создавать приложение с интеллектуальным пользовательским интерфейсом на том или ином этапе своей профессиональной деятельности. При построении приложения с интеллектуальным пользовательским интерфейсом разработчики конструируют интерфейс, часто перетаскивая набор компонентов либо элементов управления на поверхность проектирования или холст. Элементы управления сообщают о взаимодействии с пользователем, инициируя события для щелчков кнопками мыши, нажатий клавиш на клавиатуре, перемещений курсора и т.д. Разработчик добавляет код реакции на такие события в набор обработчиков событий, которые являются небольшими блоками кода, вызываемыми при выдаче специфического события в определенном компоненте. В конечном итоге получается монолитное приложение. Код, который поддерживает пользовательский интерфейс и реализует бизнес-логику, перемешан между собой без какого-либо разделения обязанностей.

Код, который определяет приемлемые значения для вводимых данных и запрашивает данные или модифицирует, например, расчетный счет пользователя, оказывается размещенным в небольших фрагментах, связанных друг с другом в предполагаемом порядке поступления событий.

Интеллектуальные пользовательские интерфейсы идеальны для простых проектов, т.к. позволяют добиться неплохих результатов достаточно быстро (по сравнению с разработкой MVC, которая требует определенных начальных затрат, прежде чем будут доставлены результаты). Интеллектуальные пользовательские интерфейсы также подходят для построения прототипов пользовательских интерфейсов. Их инструменты визуального конструирования могут оказаться по-настоящему удобными. Когда необходимо обсуждать с заказчиком требования к внешнему виду и потоку пользовательского интерфейса, тогда инструмент интеллектуального пользовательского интерфейса может быть быстрым и удобным способом для генерации и проверки различных идей.

Самый крупный недостаток интеллектуальных пользовательских интерфейсов связан с тем, что их трудно сопровождать и расширять. Смешивание кода модели предметной области и бизнес-логики с кодом пользовательского интерфейса приводит к дублированию кода, когда один и тот же фрагмент бизнес-логики копируется и вставляется для поддержки вновь добавленного компонента. Нахождение всех дублированных фрагментов и применение к ним исправления может быть непростой задачей. Добавление новой функции может оказаться практически невозможным без нарушения работы каких-то существующих функций. Тестирование приложения с интеллектуальным пользовательским интерфейсом также может быть затруднено. Единственный способ тестирования – эмуляция взаимодействия с пользователем, что является далеким от идеала и трудно реализуемым фундаментом для обеспечения полного покрытия тестами.

В мире MVC интеллектуальный пользовательский интерфейс часто называют антипаттерном, т.е. чем-то таким, чего следует избегать любой ценой. Такая антипатия возникает (по крайней мере частично) оттого, что к инфраструктуре MVC обращаются в поисках альтернативы после множества не слишком успешных попыток разработки и сопровождения приложений с интеллектуальным пользовательским интерфейсом, которые попросту вышли из-под контроля.

Тем не менее, полный отказ от паттерна «Интеллектуальный пользовательский интерфейс» будет заблуждением. Далеко не все в нем настолько плохо, и с данным подходом связаны также положительные аспекты. Приложения с интеллектуальным пользовательским интерфейсом быстро и легко разрабатывать. Производители компонентов и инструментов проектирования приложили немало усилий, чтобы сделать процесс разработки приятным занятием, поэтому даже совершенно неопытный программист за считанные часы может получить профессионально выглядящий и достаточно функциональный результат [9].

Наибольшая слабость приложений с интеллектуальным пользовательским интерфейсом – низкое удобство сопровождения – не проявляется при мелких объемах разработки. Если необходимо создать несложный инструмент для небольшой аудитории, то приложение с интеллектуальным пользовательским интерфейсом может оказаться идеальным решением. Просто отсутствует дополнительная сложность, присущая разработке приложения MVC.

Также существует другая разновидность трёхуровневой архитектуры для клиент-серверных приложений: паттерн «модель-представление-презентатор» (model-view-presenter – MVP). Он разработан для того, чтобы облегчить согласование с поддерживающими состояние платформами графического пользовательского интерфейса, такими как Windows Forms или ASP.NET WebForms. Это достойная попытка извлечь лучшее из паттерна интеллектуального пользовательского интерфейса, избежав проблем, которые он обычно привносит.

В паттерне MVP презентатор имеет те же обязанности, что и контроллер MVC, но он также более непосредственно связан с представлением, сохраняющим информацию о состоянии, напрямую управляя значениями, которые отображаются в компонентах пользовательского интерфейса в соответствии с вводом и действиями пользователя. Существуют две реализации паттерна MVP:

* Реализация пассивного представления, в которой представление не содержит никакой логики. Такое представление служит контейнером для элементов управления пользовательского интерфейса, которым напрямую управляет презентатор.
* Реализация координирующего контроллера, в которой представление может отвечать за определенные элементы логики презентации, такие как привязка данных, и получать ссылку на источник данных от моделей предметной области.

Отличие между указанными двумя реализациями касается уровня интеллектуальности представления. В любом случае презентатор отделен от инфраструктуры графического пользовательского интерфейса, что делает логику презентатора более простой и подходящей для модульного тестирования.

Разработку проекта было решено реализовывать с помощью паттерна MVC.

## 4 Программирование веб-сервера, графического пользовательского интерфейса и базы данных

## 4.1 Аппаратное обеспечение проекта

Основной задачей текущего исследования является визуализация данных, полученных от автоматической цифровой метеостанции и почвенных датчиков-влагомеров. Данные приборы разработаны в лаборатории «Сектор средств инструментального контроля». Автоматическая цифровая метеостанция и сенсорные узлы (почвенные датчики) представлены:

* аппаратным оборудованием;
* локальным хранилищем данных;
* веб-приложением.

Аппаратное оборудование включает в себя базовую станцию и набор датчиков (в составе сенсорных узлов), в том числе удалённых, подключённых к базовой станции с использованием протоколов UART, ZigBee, Wi-Fi. Базовая станция управляется микрокомпьютером, имеющем средства хранения в виде флэш-накопителей, а также связи с сетью интернет посредством 4G модема.

Локальное хранилище данных располагается в базовой станции. Выполнено на СУБД PostgreSQL. Копия данных и сам веб-сервер размещены на сервере Агрофизического Научно-Исследовательского Института. Аппаратная часть веб-сервер представляет собой персональный компьютер со следующими техническими характеристиками: ОС Windows 7; Intel Core i3 1,20 ГГц, 4 ядра, 4 Мб кэш L3; 4 Гб ОЗУ; 2ТБ HDD. Поскольку приложение веб-сервера является кросс-платформенным при необходимости его можно перенести на другие платформы, в том числе и на российские операционные системы.

Параметры работы автоматической цифровой метеостанции:

Метеостанция питается от постоянной сети и отправляет на веб-сервер по сети интернет показатели датчиков каждые 30 минут. Влагомер (сенсорный узел) – это продолговатое устройство, по длине которого расположены сенсоры, измеряющие влажность. Всего сенсоров на влагомере пять, самый верхний измеряет влажность и температуру почвы на первом слое почвы, а самый заглублённый – первый, измеряет влажность на пятом почвенном слое. У каждого сенсора на влагомере есть свой канал, и через этот канал сенсорный узел отправляет данные в базу данных на сервер. В следующей таблице представлены данные, измеряемые метеостанцией и сенсорными узлами:

Таблица 1 – Данные о агроэкосистеме измеряемые метеостанцией

|  |  |  |
| --- | --- | --- |
| Измеряемый параметр | Пример значений | Ед. измерения |
| Температура поверхности земли | 20.4 | 0С |
| Солнечное излучение | 512 | мВ |
| Влажность воздуха | 55.3 | % |
| Температура воздуха | 23.9 | 0С |
| Направление ветра | 33 | градусы |
| Скорость ветра | 44 | м/с |
| Влагомер (5 каналов измерения влажности и температуры почвы на разных глубинах) | Глубина 100 см: 15.5%, 15.44 0С  Глубина 50 см: 8.04%, 16.88 0С  Глубина 30 см: 9.97%, 18.00 0С  Глубина 20 см: 12.64%, 19.38 0С  Глубина 10 см: 14.07%, 22.00 0С | см, %, 0С |

## 4.2 Программное обеспечение проекта

Основной функционал приложения веб-сервера разработан в соответствии с требованиями заказчика, формулирование технических нюансов и организация графического пользовательского интерфейса были делегированы разработчику. Но тем не менее, программа должна реализовывать конкретные функции, описанные ниже. Функции разделены на пользовательские и администраторские, где под пользователем подразумевается любой посетитель веб-сайта, а под администратором – лицо, у которого есть право разработки, поддержки, модерации и масштабирования системы.

Пользовательские функции приложения:

* просмотр информации о погодных и почвенных данных в течении определённой даты;
* возможность выбора даты, за которую необходимо отобразить данные метеостанции и почвенных датчиков;
* возможность выбора нескольких дат, за которые необходимо отобразить данные, должно присутствовать два варианта – показать данные, полученные за один день или показать данные, полученные за три предыдущих дня от выбранной даты;
* просмотр следующих наблюдений:
  1. температура воздуха;
  2. влажность воздуха;
  3. температура почвы;
  4. скорость ветра;
  5. направление ветра;
  6. влажность почвы.
* просмотр информации в виде графика, где по оси абсцисс – время, по оси ординат – соответствующее данному графику наблюдение;
* просмотр информации в табличном виде;
* регистрация, аутентификация и авторизация в приложении;
* возможность авторизованных пользователей сохранения файла с наблюдениями за выбранную дату

Администраторские функции приложения:

* доступ к изменению исходного кода программы и разворачиванию скомпилированной программы на компьютере;
* доступ к системным файлам веб-сервера: файлам журнала, базы данных и т.д.;

В соответствии с функциональными требованиями и особенностями программирования паттерна MVC программная архитектура проекта выглядит следующим образом:

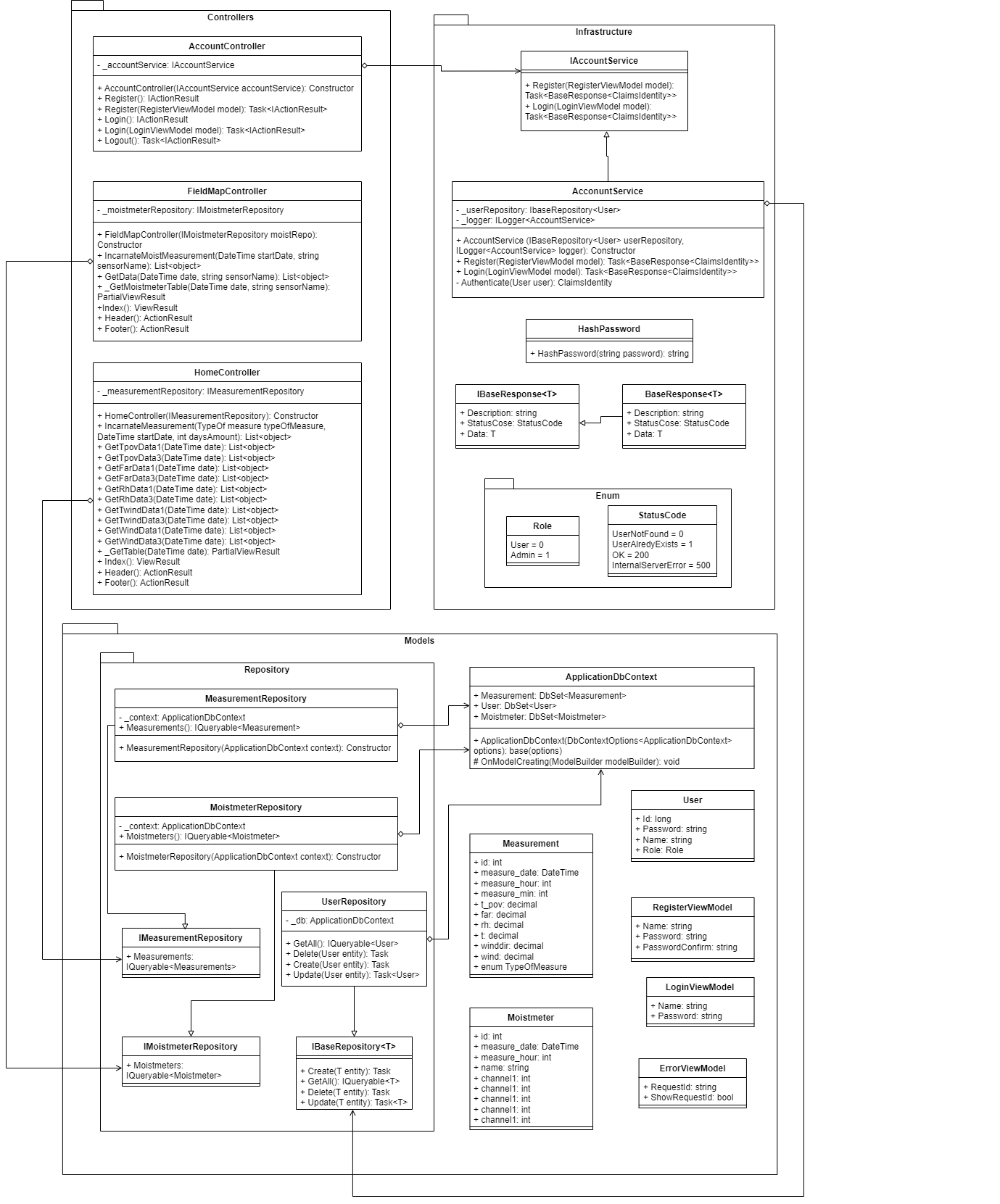


Рисунок 1.1 – Uml-диаграмма классов информационной системы по сбору, хранению и обработке агрометеорологических данных

На диаграмме классов можно увидеть три логических зоны: пространство имён контроллера, инфраструктуры и моделей. В данной схеме не отображены компоненты представлений, поскольку файлы представлений – это не классы, а файлы с HTML разметкой и возможностью вставки C# кода для программирования логики поведения веб страницы, поля для ввода с кнопками, которые есть в файлах представлений – это по сути параметры метода контроллера. Также на схеме не отображена база данных, но её структура отражена в классах моделей. Сперва следует рассмотреть структуру базы данных:

SQL сервер содержит несколько таблиц:

Таблица Measurement содержит данные, собранные метеостанцией, описание данных приведено в таблице 1.

Таблица 1 – Свойства таблицы «Measurement»

|  |  |  |
| --- | --- | --- |
| Название колонки | Тип данных | Описание |
| id | int4 | уникальный идентификатор записи |
| measure\_date | date | дата измерения (ГГГГ-ММ-ДД) |
| measure\_hour | int4 | час измерения (чч) |
| measure\_min | int4 | минута измерения (мм) |
| t\_pow | numeric | температура поверхности почвы (градусы цельсия) |
| far | numeric | датчик освещённости (люкс) |
| rh | numeric | влажность воздуха () |
| t | numeric | температура воздуха (градусы цельсия) |
| winddir | numeric | направление ветра (градусы цельсия) |
| wind | numeric | скорость ветра (м/с) |
| rain | numeric | количество осадков (мм) |

Таблица Moistmeter содержит данные, собранные влагомерами. Описание данных приведено в таблице 1.

Таблица 1 – Свойства таблицы «Moistmeter»

|  |  |  |
| --- | --- | --- |
| Название колонки | Тип данных | Описание |
| id | int4 | уникальный идентификатор записи |
| measure\_date | date | дата измерения (ГГГГ-ММ-ДД) |
| measure\_hour | numeric | час измерения (чч) |
| channel1 | numeric | процент влажности почвы записанный первым каналом на глубине 100 см |
| channel2 | numeric | процент влажности почвы записанный вторым каналом на глубине 50 см |
| channel3 | numeric | процент влажности почвы записанный третьим каналом на глубине 30 см |
| channel4 | numeric | процент влажности почвы записанный четвертым каналом на глубине 20 см |
| channel5 | numeric | процент влажности почвы записанный пятым каналом на глубине 10 см |

Таблица User содержит данные о пользователях системы. В системе предусмотрено 2 типа пользователей: администратор и пользователь. Авторизованный пользователь на сайте может скачать наблюдения за указанный период в формате Excel. Администратор обладает расширенными возможностями. Пароль хранится в зашифрованном виде, алгоритм шифрования – Sha256. Также следует уточнить, что поле Role в таблице может принимать только два числовых значения – 0 для пользователя, 1 для администратора. Описание данных приведено в таблице 1.

Таблица 1 – Свойства таблицы «User»

|  |  |  |
| --- | --- | --- |
| Название колонки | Тип данных | Описание |
| id | int4 | уникальный идентификатор записи |
| Password | text | пароль учётной записи |
| Name | varchar(100) | имя пользователя |
| Role | int4 | роль пользователя, где 0 – это пользователь, а 1 - администратор |

Свойства таблицы «Measurement» из базы данных идентичны с полями класса «Measurement» из пространства имён контроллера. Суть взаимодействия двух конечных точек – базы данных и пользовательского интерфейса реализована с помощью архитектурного принципа «Dependency Injection» или же «Внедрение зависимостей». Данный подход не является обязательным в программировании, но благодаря ему можно писать легкоподдерживаемые и расширяемые приложения. Основная идея внедрения зависимостей – это внедрение интерфейса между взаимодействием класса модели и контроллера обработки данных. Например, чтобы реализовать отображение данных метеостанции в графическом интерфейсе сначала необходимо создать объект «репозиторий» и строку подключения к базе данных, чтобы передавать из базы в репозиторий сами данные.

Строка подключений содержится в файле appsettings.json и выглядит следующим образом:

"ConnectionStrings": {

"DefaultConnection": "Host=localhost;Port=5432;Database=meteostation;Username=postgres;Password=cherry"

}

В данной строке необходимо указать порт, по которому доступна программа SQL сервера, имя базы данных, имя пользователя-хозяина базы данных и пароль.

Ключевым объектом в концепции внедрения зависимостей в данном примере является интерфейс IMeasurementRepository, потому что в контроллере будет создан экземпляр именно интерфейса, а не самого класса MeasurementRepository. Это нужно для того, чтобы функционал манипулирования данных в контроллере строго задавался в интерфейсе какой-либо конкретной модели. Интерфейс находится в файле IMeasurementRepository и выглядит следующим образом:

public interface IMeasurementRepository

{

IQueryable<Measurement> Measurements { get; }

}

Далее необходимо создать реализацию интерфейса, где будет определена сама логика манипулирования данными, реализация интерфейса IMeasurementRepository находится в файле MeasurementRepository.cs и выглядит следующим образом:

public class MeasurementRepository : IMeasurementRepository

{

private ApplicationDbContext \_context;

public MeasurementRepository(ApplicationDbContext context)

{

\_context = context;

}

public IQueryable<Measurement> Measurements => \_context.Measurement;

}

В конструкторе данного класса инициализируется контекст данных, из файла ApplicationDbContext.cs, который будет рассмотрен ниже, а также реализован единственный метод, определённый в интерфейсе, который возвращает данные из БД.

В файле ApplicationDbContext, который наследуется от базового стандартного класса DbContext содержится конструктор и свойство Measurement типа DbSet<Measurement>. С помощью этого свойства можно обращаться к БД за записями из таблицы Measurement. Файл ApplicationDbContext.cs выглядит следующим образом:

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { Database.EnsureCreated(); }

public DbSet<Measurement> Measurement { get; set; }

После создания репозитория и подключения к базе данных необходимо определить контроллер, который будет отвечать за обработку данных из модели. Для этого в контроллере необходимо создать экземпляр класса IMeasurementRepository и проинициализировать его в конструкторе. Содержимое файла HomeController.cs выглядит следующим образом:

private readonly IMeasurementRepository \_measurementRepository;

public HomeController(IMeasurementRepository measRepo)

{

this.\_measurementRepository = measRepo;

}

Затем в контроллере вся логика обработки данных модели будет совершаться только через объект \_measurementRepository. Например передача данных метеостанции за указанную пользователем дату через кнопку календаря в представление определяется следующим методом:

[HttpGet]

public PartialViewResult \_GetTable(DateTime date)

{

var tmp = \_measurementRepository.Measurements.Where(p => p.measure\_date == date.ToUniversalTime());

return PartialView(tmp);

}

Можно увидеть, что посредством Linq запроса к объекту \_measurementRepository будет возвращено представление с отфильтрованными по значению входного параметра метода данными. Атрибут HttpGet и входной параметр метода необходимы для взаимодействия с JavaScript файлами, которые отвечают за создание интерактивных графиков. Сами JavaScript файлы и механизм взаимодействия их с MVC сервером будут описаны ниже в главе.

Далее необходимо перейти к рассмотрению файла представления, который является последним звеном в цепочке программирования паттерна «внедрение зависимостей». Особенность программирования технологии C# MVC заключается в том, что файл представления, в который передаются данные из контроллера должны называться точно также как и метод контроллера, который передаёт эти данные. Из кода выше можно сказать о файле представления следующее: он называется \_GetTable.cshtml (расширение .cshtml говорит само за себя: файл является смесью C# кода и HTML-разметки), и он является частичным представлением, поскольку метод контроллера возвращает объект PartialView. Частичное представление – это тоже файл представления, но он отличается от обычного представления тем, что он встраивается в обычное представление, иными словами частичное представление является частью родительского представления. Код файла \_GetTable.cshtml выглядит следующим образом:

@model IQueryable<Measurement>

@{

if (Model.Count() > 1)

{

<table>

<thead>

<tr><th>date</th><th>t\_pov</th><th>far</th><th>rh</th><th>t</th><th>winddir</th><th>wind</th></tr>

</thead>

<tbody>

@foreach (var measure in Model)

{

DateTime date = new DateTime(measure.measure\_date.Year, measure.measure\_date.Month, measure.measure\_date.Day, measure.measure\_hour, measure.measure\_min, 0, 0);

decimal winddir = @measure.winddir/1200\*360;

<tr>

<td>@date.ToString("yyyy-MM-dd HH:mm")</td>

<td>@measure.t\_pov</td>

<td>@measure.far</td>

<td>@measure.rh</td>

<td>@measure.t</td>

<td>@Math.Round(winddir, 1)</td>

<td>@measure.wind</td>

</tr>

}

</tbody>

</table>

}

else

{

<h1 class="info-header">Данных за выбранную дату нет.</h1>

}

}

В самом начале файла необходимо определить переменную модели, тип которой должен совпадать с типом, который был указан в параметре возвращаемого значения соответствующего контроллера, и далее, обращаясь к переменной модели можно отображать данные, например, как в вышеописанном примере через таблицу, чередуя HTML разметку и C# код. В этом и состоит главное преимущество технологии программирования MVC – разделение модели, контроллеров и представлений.

Данное частичное представление возможно отобразить в главной странице, где находится основная разметка. Таким же способом на основной странице отображаются шапка сайте с панелью навигации, кнопками входа и регистрации и подвал сайта. Благодаря такому подходу получается избежать дублирование кода, если будет необходимость добавлять новые страницы, где будет возможность отображать частичные представления следующим кодом, где асинхронно вызывается метод контроллера с названием данного метода в параметре:

<header class="main-header">

@await Html.PartialAsync("\_Header")

</header>

Финальной частью реализации архитектуры «внедрение зависимостей» является добавление в файл Program.cs конфигурационных правок. Этот файл также не входит в Uml-диаграмму классов, так как он по сути тоже не класс. Этот файл отвечает за запуск и конфигурацию приложения. Для корректной работы веб-приложения необходимо добавить строчку

builder.Services.AddTransient<IMeasurementRepository, MeasurementRepository>();

Тем самым сообщая, что в данном проекте используется внедрение зависимостей, и указывая, какой интерфейс, используемый в контроллере, соответствует репозиторию объектов.

После запуска приложения на экране появится следующая картина:

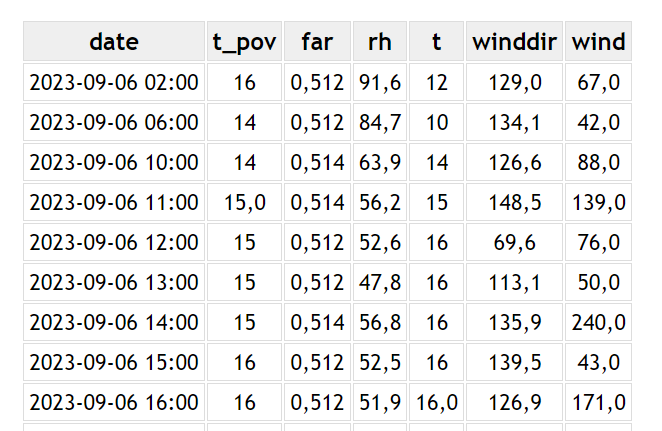


Рисунок ХХХ – отображение данных из БД на сайте в табличном виде

Действительно, таблица из пользовательского графического интерфейса соответствует структуре, и тем данным, что содержатся в PostgreSQL. Операция визуализации данных посредством реализации архитектурного решения «внедрение зависимостей» проведена успешно.

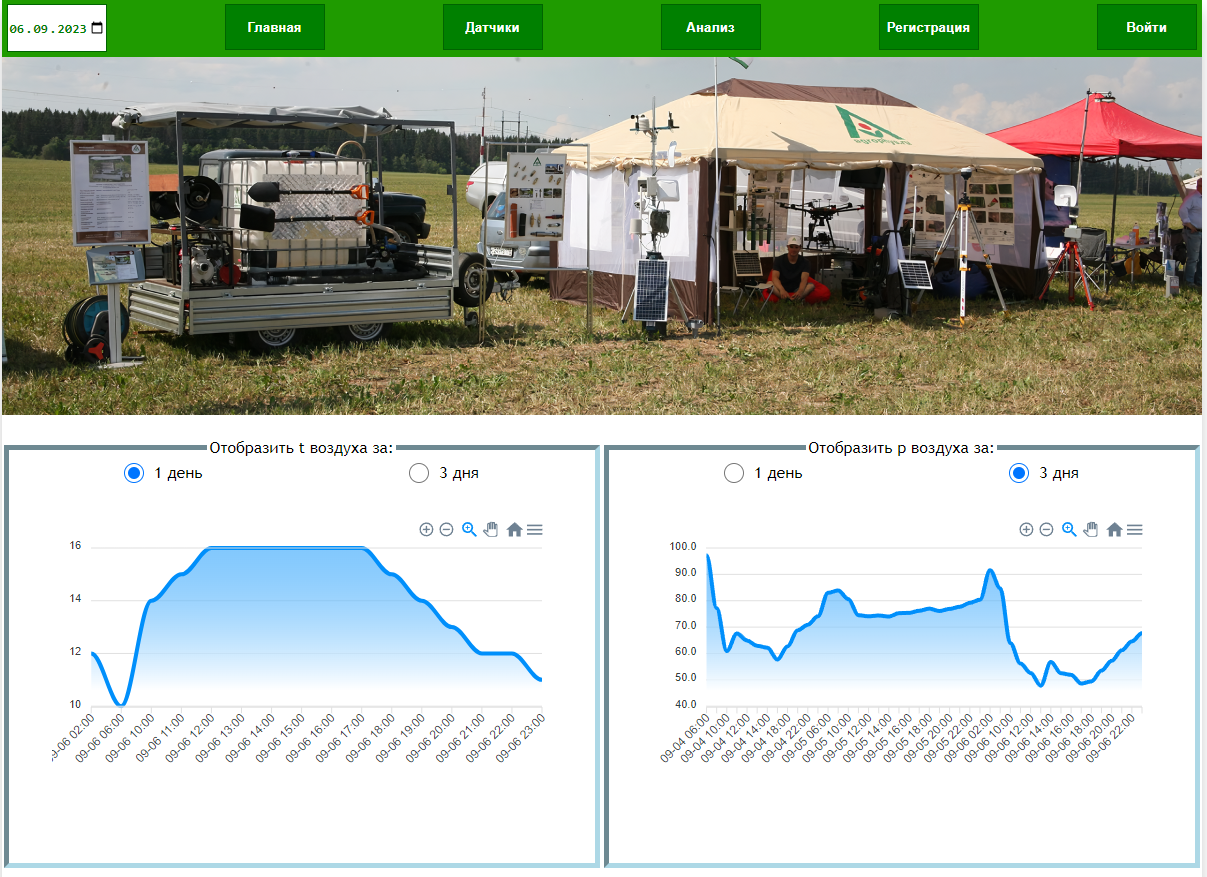


Рисунок ЪЪЪ – Главная страница веб-приложения

Далее необходимо описать, каким образом формируются графики на сайте. Сама реализация графического представления данных не была реализована разработчиком, отображение графиков реализует бесплатная JavaScript библиотека ApexCharts [https://apexcharts.com/].

Логика формирования графиков и взаимодействия кнопок описаны в JavaScript – файлах, код представлен в Приложении В. Далее следует рассмотреть основной принцип визуализации данных.

В JavaScript файле создается обработчик события инициализации страницы для изначальной отрисовки графиков и обработчики событий на кнопки изменения даты, при нажатии которых необходимо обновить информацию в графиках. При срабатывании обработчика события вызывается функция создания графиков или обновления, которые описаны в Приложении В. Наиболее интересной частью кода является создание ajax-get запроса. Ajax-get запрос – это асинхронный способ обращения к методу контроллера. Такой механизм необходим для связи JavaScript файлов и контроллеров приложения C# MVC. В данном запросе необходимо указать url-ссылку с названием самого контроллера и метода действия, в который необходимо передать данные полученные из HTML тега посредством JavaScript инструкций. Запрос передаётся в соответствующий метод, который вызывает другой метод, который возвращает значение из базы данных с отфильтрованными данными, это возвращаемое значение передается обратно в JavaScript файл, в параметры метода, который вызывается при успешном выполнении вышеописанного get-запроса, и в конечном итоге в этом методе обрабатываются данные из параметра для обновления или отрисовки графиков.

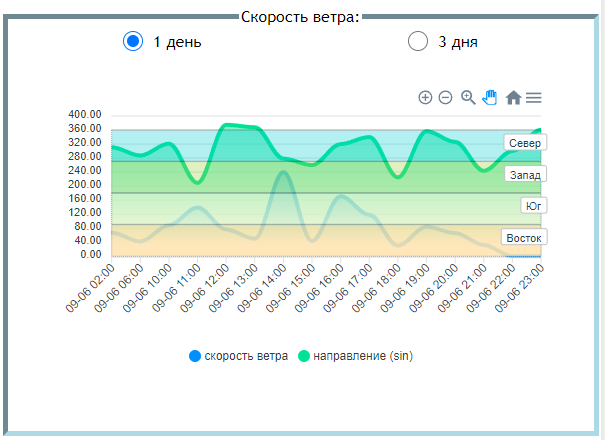


Рисунок ЪЪЪ – график отображения скорости и направления ветра, где по оси абсцисс – время, по оси ординат – фактические значения

Также в приложении присутствует механизм авторизации и аутентификации. Данный функционал мотивирован тем, что только авторизованным пользователям можно выдавать права на скачивание информации метеостанции и влагомеров в табличном виде себе на компьютер.

Аутентификация – это установление личности пользователя, а авторизация – выдача прав. Чтобы выдать права какому – либо пользователю необходимо выяснить, кем он является? Как минимум из-за того, что в приложении нужно разделять роли администраторов и обычных пользователей.

Для реализации алгоритма аутентификации и авторизации необходимо создать интерфейс, в котором будет описаны возможности пользователей. Такой интерфейс описан в файле IAccountService.cs и выглядит следующим образом:

public interface IAccountService

{

Task<BaseResponse<ClaimsIdentity>> Register(RegisterViewModel model);

Task<BaseResponse<ClaimsIdentity>> Login(LoginViewModel model);

}

В приложении можно выполнить всего две функции – зарегистрироваться и войти. Регистрация нужна для создания новых пользователей с обычными правами (администраторы создаются системно), а функционал входа нужен для распознания пользователя и в зависимости от его роли отобразить новый контент на сайте. Реализация методов интерфейса IAccountService находится в файле AccountService.cs и предоставлена в Приложении Г. Также стоит отметить, что при создании и авторизации пользователей используется метод HashPassword, который находится в файле HashPassword.cs. Данный метод шифрует пароли пользователей по алгоритму SHA, в БД хранятся зашифрованные пароли и при входе программа проверяет хэш введённого пользователем пароля с хэшем из базы данных, который хранится в поле «Password». Контроллер AccountController использует функции из AccountService и вызывает представления Login.cshtml и Register.cshtml. Проверка на то, авторизован пользователь или нет определяется кодом

@if (User.Identity.IsAuthenticated)

{

<button class="download-data-button sample-button" href="#">Скачать базу данных</button>

}

Если условие верно (то есть пользователь аутентифицирован), то выполняется код в фигурных скобках (появляется новая кнопка). В противном случае, когда вход в приложение не выполнен, данной кнопки не будет на сайте.

Также можно разграничить права пользователя с помощью кода

@if (User.IsInRole("Admin"))

{

<p>Вы авторизованы как администратор</p>

}

Где в условии проверяется принадлежность пользователя к определённой группе.

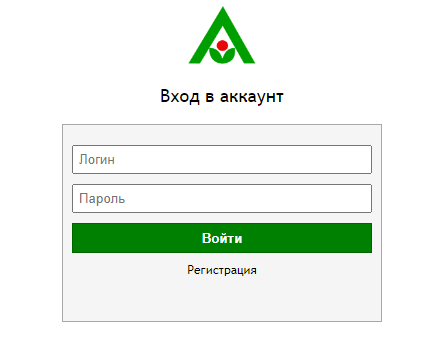


Рисунок ЪЪЪ – окно входа в аккаунт

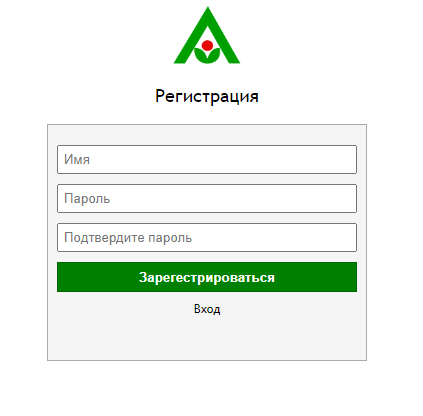


Рисунок ЪЪЪ – окно регистрации



Рисунок ЪЪЪ – изменённая страница, после прохождения процедуры аутентификации и авторизации

Процедура аутентификации и авторизации проведена успешно.

## 5 Реализация алгоритма многофакторного дисперсионного анализа

## 5.1 Описание структуры модуля многофакторного дисперсионного анализа

## 5.2 Описание структуры модуля обработки данных методом box-cox

## 5.3 Описание структуры модуля дисперсионного анализа с использованием непараметрических методов

## 5.4 Разработка документации

## 6 Тестирование программы

## 7 Опытная и промышленная эксплуатация

# ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было проведено исследование существующих приложений для анализа музыки, определены их достоинства и недостатки. Исследован алгоритм их работы распознавания музыкальных композиций и его математические основы. Был изучен и формализован универсальный алгоритм распознавания музыки. Выбраны методы и технологии создания системы распознавания музыки. Разработано приложение для распознавания музыкальных композиций   
на языке C#. Проведена оценка эффективности работы приложения.

По итогам исследования существующих конкурентных системы было определено, что существующие системы не позволяют выбрать алгоритм поиска, как-либо уточнить поиск, а также отсутствуют на платформе Windows и не позволяют выбирать устройство для прослушивания. С учётом этих недостатков были выбраны требования к разрабатываемой системе.

При исследовании алгоритмов распознавания музыкальных композиций был выбран подход, основанный на преобразовании Фурье, поскольку он обеспечивает наибольшую универсальность при наименьших затратах   
на настройку и обучение системы.

Созданная система позволяет успешно анализировать и искать музыкальные композиции по отрывку.

Выбранные средства разработки позволяют в полной мере реализовать функционал системы распознавания музыкальных композиций, и сделать её использование понятным бля большинства пользователей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Станкевич Ф.В. Спицын В.Г. НЕЙРОСЕТЕВОЕ РАСПОЗНАВАНИЕ МУЗЫКАЛЬНЫХ ИНСТРУМЕНТОВ С ИСПОЛЬЗОВАНИЕМ МЕЛ‑ЧАСТОТНЫХ КЕПСТРАЛЬНЫХ КОЭФФИЦИЕНТОВ, 2014.
2. Li-Chun Wang. An Industrial-Strength Audio Search Algorithm, 2003.
3. Веялкин И.А. Шиман Д.В. Обзор структур алгоритмов поиска на основе отпечатков аудиоданных, 2015.
4. Shazam: алгоритмы распознавания музыки, сигнатуры, обработка данных [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/wunderfund/blog/275043/ (дата обращения 19.13.2023).
5. Галанина Н.А. Ефимов А.Д. Моделирование оптимальных алгоритмов быстрого преобразования Фурье и их сравнительный анализ / 2009 год.
6. Рональд Н. Брейсулл. Преобразование Фурье // Scientific American // Издание на русском языке № 8, 1989.
7. Использование NoSQL-хранилища LiteDB в .NET [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/articles/ado.net/1.php / (дата обращения 08.05.2023).
8. Документация Math.Net Fourier and related linear integral transforms [Электронный ресурс]. – Режим доступа: https://numerics.mathdotnet.com/IntegralTransforms / (дата обращения 15.05.2023).
9. Джеффри Рихтер. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд., 2022.
10. В.М. Матюшок. Мировой рынок систем и технологий искусственного интеллекта: становление и тенденция развития // Вестник Российского университета дружбы народов. Серия: Экономика, 2020.
11. Анисимова С.В., Головатый А.И. АЛГОРИТМЫ РАСПОЗНАВАНИЯ МУЗЫКАЛЬНЫХ ОТРЫВКОВ, НОТ И АККОРДОВ, 2021.
12. Анисимова С.В. Алгоритм распознавания музыкальных композиций по вокальному исполнению, 2020.
13. Wang, C., Yang, S. Automatic lyric transcription using speech recognition and the properties of lyrics, 2003.

# ПРИЛОЖЕНИЕ А (листинг файла)

# ПРИЛОЖЕНИЕ Б (листинг файла)

# ПРИЛОЖЕНИЕ В (листинг файла indexcharts.js)

$(function () {

var chartTwind;

var chartPwind;

var chartTpov;

var charWindSpeed;

var calendarButton = document.querySelector(".main-header\_\_calendar");

renderChart = function (type) {

var actionStr = "";

var seriesName = "";

var chartClass;

switch (type) {

case "Twind":

actionStr = "Twind";

seriesName = "t воздуха";

chartClass = document.querySelector(".form\_\_chart--twind");

break;

case "Pwind":

actionStr = "Rh";

seriesName = "p воздуха";

chartClass = document.querySelector(".form\_\_chart--pwind");

break;

case "Tpov":

actionStr = "Tpov";

seriesName = "t поверхности почвы";

chartClass = document.querySelector(".form\_\_chart--tpov");

break;

case "WindSpeed":

actionStr = "Wind";

seriesName = "скорость ветра";

chartClass = document.querySelector(".form\_\_chart--precipitation");

break;

}

$.ajax({

type: "GET",

url: "/Home/Get" + actionStr + "Data1",

data: {

date: calendarButton.value

},

contentType: "application/json; charset=utf-8",

dataType: "json",

success: OnSuccessResult,

error: OnError

});

function OnSuccessResult(temperatureChart) {

var chartLabels = temperatureChart[0];

var chartData = temperatureChart[1];

var chartWindDirData = temperatureChart[2];

var chartMatter;

if (chartWindDirData != null) {

chartMatter = {

chart: {

height: 280,

type: "area"

},

dataLabels: {

enabled: false

},

annotations: {

yaxis: [

{

y: 0,

y2: 90,

borderColor: '#000',

fillColor: '#FEB019',

label: {

offsetY: 20,

text: 'Восток'

}

}, {

y: 91,

y2: 180,

borderColor: '#000',

fillColor: 'Khaki',

label: {

offsetY: 20,

text: 'Юг'

}

},

{

y: 181,

y2: 270,

borderColor: '#000',

fillColor: 'YellowGreen',

label: {

offsetY: 20,

text: 'Запад'

}

},

{

y: 360,

y2: 271,

borderColor: '#000',

fillColor: 'DarkTurquoise',

label: {

offsetY: 20,

text: 'Север'

}

},

]

},

series: [

{

name: seriesName,

data: chartData

},

{

name: "направление (sin)",

data: chartWindDirData

}

],

fill: {

type: "gradient",

gradient: {

shadeIntensity: 1,

opacityFrom: 0.7,

opacityTo: 0.9,

stops: [0, 90, 100]

}

},

xaxis: {

categories: chartLabels,

labels: {

rotate: -45

}

},

yaxis: {

max: 400

}

};

}

else {

chartMatter = {

chart: {

height: 280,

type: "area"

},

dataLabels: {

enabled: false

},

series: [

{

name: seriesName,

data: chartData

}

],

fill: {

type: "gradient",

gradient: {

shadeIntensity: 1,

opacityFrom: 0.7,

opacityTo: 0.9,

stops: [0, 90, 100]

}

},

xaxis: {

categories: chartLabels,

labels: {

rotate: -45

}

}

};

}

switch (type) {

case "Twind":

chartTwind = new ApexCharts(chartClass, chartMatter);

chartTwind.render();

break;

case "Pwind":

chartPwind = new ApexCharts(chartClass, chartMatter);

chartPwind.render();

break;

case "Tpov":

chartTpov = new ApexCharts(chartClass, chartMatter);

chartTpov.render();

break;

case "WindSpeed":

charWindSpeed = new ApexCharts(chartClass, chartMatter);

charWindSpeed.render();

break;

}

}

function OnError(err) {

}

}

updateChart = function (type) {

var chart;

var actionStr = "";

var chartName = "";

switch (type) {

case "Twind":

actionStr = "Twind";

chartName = "TW";

chart = chartTwind;

break;

case "Pwind":

actionStr = "Rh";

chartName = "PW";

chart = chartPwind;

break;

case "Tpov":

actionStr = "Tpov";

chartName = "TS";

chart = chartTpov;

break;

case "WindSpeed":

actionStr = "Wind";

chartName = "PC";

chart = charWindSpeed;

break;

}

var a = "/Home/Get" + actionStr + "Data1";

var tRadio = document.querySelector("input[name=" + chartName + "]:checked");

if (tRadio.value == 1) {

a = "/Home/Get" + actionStr + "Data1";

}

else if (tRadio.value == 3) {

a = "/Home/Get" + actionStr + "Data3";

}

$.ajax({

type: "GET",

url: a,

data: {

date: calendarButton.value

},

contentType: "application/json; charset=utf-8",

dataType: "json",

success: OnSuccessResult,

error: OnError

});

function OnSuccessResult(temperatureChart) {

var chartLabels = temperatureChart[0];

var chartData = temperatureChart[1];

var chartWindDirData = temperatureChart[2];

if (chartWindDirData != null) {

chart.updateOptions({

xaxis: {

categories: chartLabels

},

series: [

{

data: chartData

},

{

name: "направление (град)",

data: chartWindDirData

}

]

})

}

else {

chart.updateOptions({

xaxis: {

categories: chartLabels

},

series: [

{

data: chartData

}

]

})

}

}

function OnError(err) {

}

}

$("input[type=radio][name=TW]").change(() => updateChart("Twind"));

$("input[type=radio][name=PW]").change(() => updateChart("Pwind"));

$("input[type=radio][name=TS]").change(() => updateChart("Tpov"));

$("input[type=radio][name=PC]").change(() => updateChart("WindSpeed"));

$(".main-header\_\_calendar").on("change", function () {

updateChart("Twind");

updateChart("Pwind");

updateChart("Tpov");

updateChart("WindSpeed");

});

$(document).ready(function () {

renderChart("Twind");

renderChart("Pwind");

renderChart("Tpov");

renderChart("WindSpeed");

})

});

# ПРИЛОЖЕНИЕ Г (листинг файла AccountService.cs)

using NuGet.ContentModel;

using ServerMVC.Infrastructure.Enum;

using ServerMVC.Models;

using System.Security.Claims;

using ServerMVC.Infrastructure;

using ServerMVC.Models.Repository;

using Microsoft.EntityFrameworkCore;

namespace ServerMVC.Infrastructure

{

public class AccountService : IAccountService

{

private readonly IBaseRepository<User> \_userRepository;

private readonly ILogger<AccountService> \_logger;

public AccountService(IBaseRepository<User> userRepository,

ILogger<AccountService> logger)

{

\_userRepository = userRepository;

\_logger = logger;

}

public async Task<BaseResponse<ClaimsIdentity>> Register(RegisterViewModel model)

{

try

{

var user = await \_userRepository.GetAll().FirstOrDefaultAsync(x => x.Name == model.Name);

if (user != null)

{

return new BaseResponse<ClaimsIdentity>()

{

Description = "Пользователь с таким логином уже есть",

};

}

user = new User()

{

Name = model.Name,

Role = Role.User,

Password = HashPassword.HashPassowrd(model.Password),

};

await \_userRepository.Create(user);

var result = Authenticate(user);

return new BaseResponse<ClaimsIdentity>()

{

Data = result,

Description = "Объект добавился",

StatusCode = StatusCode.OK

};

}

catch (Exception ex)

{

\_logger.LogError(ex, $"[Register]: {ex.Message}");

return new BaseResponse<ClaimsIdentity>()

{

Description = ex.Message,

StatusCode = StatusCode.InternalServerError

};

}

}

public async Task<BaseResponse<ClaimsIdentity>> Login(LoginViewModel model)

{

try

{

var user = await \_userRepository.GetAll().FirstOrDefaultAsync(x => x.Name == model.Name);

if (user == null)

{

return new BaseResponse<ClaimsIdentity>()

{

Description = "Пользователь не найден"

};

}

if (user.Password != HashPassword.HashPassowrd(model.Password))

{

return new BaseResponse<ClaimsIdentity>()

{

Description = "Неверный пароль или логин"

};

}

var result = Authenticate(user);

return new BaseResponse<ClaimsIdentity>()

{

Data = result,

StatusCode = StatusCode.OK

};

}

catch (Exception ex)

{

\_logger.LogError(ex, $"[Login]: {ex.Message}");

return new BaseResponse<ClaimsIdentity>()

{

Description = ex.Message,

StatusCode = StatusCode.InternalServerError

};

}

}

private ClaimsIdentity Authenticate(User user)

{

var claims = new List<Claim>

{

new Claim(ClaimsIdentity.DefaultNameClaimType, user.Name),

new Claim(ClaimsIdentity.DefaultRoleClaimType, user.Role.ToString())

};

return new ClaimsIdentity(claims, "ApplicationCookie",

ClaimsIdentity.DefaultNameClaimType, ClaimsIdentity.DefaultRoleClaimType);

}

}

}