СОДЕРЖАНИЕ

[Введение 3](#_Toc105102961)

[1 Аналитический раздел 5](#_Toc105102962)

[1.1 Анализ предметной области 5](#_Toc105102963)

[1.2 Обзор аналогов систем контроля экспедиторской деятельности 7](#_Toc105102964)

[1.3 Выбор платформы для реализации и адаптация процессов 11](#_Toc105102965)

[1.4 Разработка требований к информационной системе 16](#_Toc105102966)

[2 Технологический раздел 18](#_Toc105102967)

[2.1 Архитектура системы 18](#_Toc105102968)

[2.2 Выбор средств разработки 20](#_Toc105102969)

[2.2.1 Выбор языка программирования 20](#_Toc105102970)

[2.2.2 Выбор СУБД 23](#_Toc105102971)

[2.3 Проектирование информационной системы 24](#_Toc105102972)

[2.3.1 Проектирование интерфейса бота 24](#_Toc105102973)

[2.3.2 Проектирование базы данных 25](#_Toc105102974)

[2.3.3 Проектирование функциональности 27](#_Toc105102975)

[3. Реализация информационной системы 31](#_Toc105102976)

[3.1 Реализация базы данных 31](#_Toc105102977)

[3.2 Реализация функций системы 31](#_Toc105102978)

[3.3 Развертывание системы 38](#_Toc105102979)

[4. Тестирование информационной системы 44](#_Toc105102980)

[4.1 Юнит тестирование 44](#_Toc105102981)

[4.2 Мануальное тестирование 47](#_Toc105102982)

[Заключение 48](#_Toc105102983)

[Список использованной литературы 49](#_Toc105102984)

[Приложение А 52](#_Toc105102985)

# Введение

В самом начале цифровизации бизнеса, основными средствами автоматизации были базы данных и построенные на их основе приложения, позволяющие автоматизировать расчеты. Далее развитие вычислительных систем позволило автоматизировать более сложные процессы. Основной особенностью данного этапа развития являлось то, что вся основная логика ложилась на систему управления базой данных.

Последующее увеличение объемов данных затрудняло их обработку в самой базе. В таком случае приложения, обеспечивающие взаимодействие с пользователем, стали все больше реализовывать логику обработки данных, забирая часть задач от базы данных.

Следующее этапом развития был переход к распределенным системам. Начиная автоматизацию бизнеса с одного рабочего места, бизнес пришел к большому количеству автоматизированных рабочих мест и распределенным сервисам. Что еще вчера являлось единым приложением сегодня представляет собой совокупность взаимосвязанных микросистем.

Последние пять лет характеризуются интенсивным внедрением мобильных технологий в бизнес-процессы. Мобильные приложения банков, клиентские приложения для сервисов, функции переписки укрепились за мессенджерами. Примерно в это же время мессенджеры из средств коммуникации начали трансформироваться в самостоятельные средства автоматизации деятельности человека, бизнес тоже не стал исключением. Одну из важнейших функций в автоматизации бизнеса приняли на себя чат-боты. С помощью чат-ботов возможно организовывать взаимодействие людей, обмен данными и многое другое. Уже сегодня во всех популярных мессенджерах присутствуют функция создания ботов (например, telegram Instagram, whatsapp, slack и др.).

В виду современных тенденций развития целью дипломной работы является проектирование чат-бота в мессенджере телеграмм. Для автоматизации выбран бизнес-процесс учета и распределения маршрутов доставки аграрной промышленности. Основными задачами дипломной работы являются анализ целевого бизнес-процесса, выбор средств реализации бота, имплементация и внедрение в целевое предприятие – ООО «МирАгро».

# Аналитический раздел

Целью данного раздела является обзор и анализ целевого бизнес-процесса предприятия. Целевым предприятием ВКР является ООО «Мирагро», оно занимается выращиванием сельскохозяйственных культур, таких как пшеница, кукуруза, ячмень и бобовые. Объемы производства составляет 7095 тонн продукции ежегодно, выращиваемой на территории более чем 7500 гектаров плодородной почвы. Продукция размещается на складах, которые позволяют хранить 47000 тонн готовой продукции. Для производства собственная техника, более 30 видов, в том числе специализированная и общего назначения. На сегодняшний день сотрудниками предприятия являются более 70 сотрудников. Среди сотрудников присутствуют такие специальности как логисты, водители, механизаторы, бухгалтера, управленческий персонал и др.

Кроме непосредственно производства предприятие занимается и вопросами доставки собственной продукции как постоянным партнерам, так и в рамках разовых договоров. Ввиду того, что штат сотрудников в сфере доставки продукции расширяется и усложняются процессы целесообразно внедрить систему автоматизации учета и распределения маршрутов между водителей. Проанализируем данный процесс и выделим основные, очевидные потоки данных, а также данные, которые могут сделать процедуру прозрачной и понятной каждому сотруднику.

## 1.1 Анализ предметной области

Все маршруты, производимые сотрудниками компании, разделяются на постоянные и разовые. Постоянные маршруты представляют собой маршруты, осуществляемые в рамках постоянных, долговременных договоров. Разовые маршруты появляются в результате заключения единичных договоров на короткое время.

Распределение между водителями логист производит вручную, среди постоянных заказов и «на лету» при возникновении разовых осуществляется перераспределение. При этом возникают некоторые проблемы в случае болезни или чрезвычайных происшествий с экспедиторами, заключающиеся в перераспределении маршрутов, пересчете объема выполненной работы, оперативности перераспределения и т.д.

Кроме того, существуют особенности оплаты водителям за рейс в рамках временного заказа. А именно оплата труда происходит по факту выполненной работы. Данные сведения должны быть доступны отделу бухгалтерии своевременно и по факту выполнения или по запросу.

Важной составляющей работы управления компанией является планирование и анализ деятельности предприятия. Для обеспечения этих задач необходим инструмент подготовки данных и преобразования их в наглядную форму.

Бизнес-процесс можно представить диаграммой, представленной на рисунке 1.1

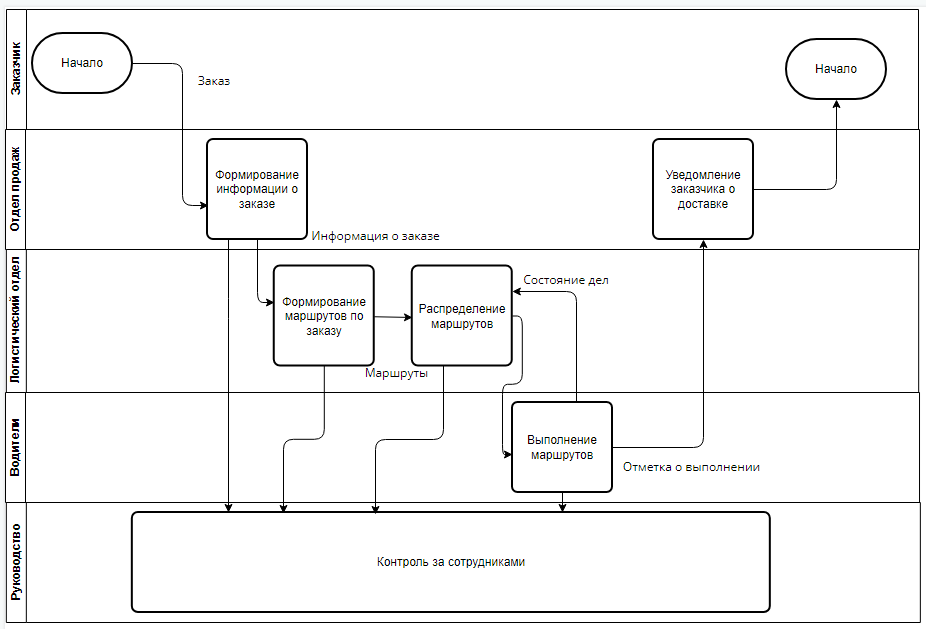


Рисунок 1.1 – Диаграмма выполнения анализируемого бизнес-процесса

Как видно из рисунка 1.1 автоматизации подлежит взаимодействие внутри компании, а именно формирование маршрутов из заявки, их распределение, отметки о выполнении и подготовка отчетности руководству. Выделим входную и выходную информацию для бота, рисунок 1.2.

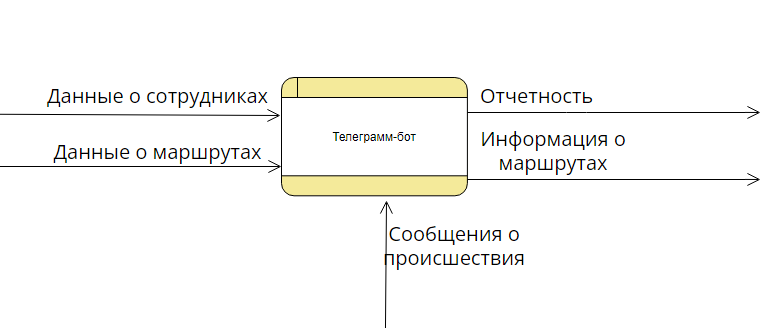


Рисунок 1.2 – Анализ внешних данных в системе

Как видно из рисунка 1.1 входными данными для бота являются данные сотрудниках, для их регистрации и распределения ролей, данные о маршрутах (реализация учета и распределения маршрутов) и сообщения о происшествиях, для взаимодействия водителей и логистов между собой.

Результатными информационными потоками являются отчеты о прохождении маршрутов, справочная информация о маршрутах для сотрудников компании.

Рассмотрим системы представляющие похожий функционал.

## 1.2 Обзор аналогов систем контроля экспедиторской деятельности

Планируемая система не является инновационной, по этой причине проанализируем рынок подобных приложений для определения целесообразности разработки.

**Универсальная Система Учета.** Универсальная система учета представляет собой комплексное решение для автоматизации процессов на предприятии, в том числе и перевозок. Данное программное обеспечение представляет собой набор модулей для автоматизации деятельности перевозчика. Особенностями является необходимость адаптации на предприятии, обучение сотрудников, необходимость устанавливать клиентские приложения сотрудникам. Стоимость на одного сотрудника и одного экземпляра приложения, составляет 57000 рублей.

**Максоптра.** Онлайн-система управления логистикой, позволяющая автоматически распределить задачи между исполнителями и спланировать наиболее быстрые маршруты без лишних затрат. Поддерживает интеграцию с системами ГЛОНАСС и GPS.

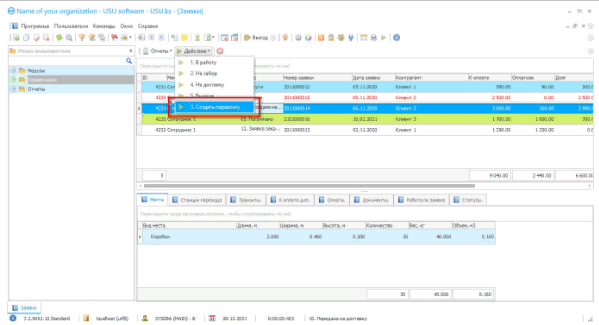


Рисунок 1.3 Внешний вид интерфейса универсальной системы учета

**КиберЛог.** Облачный сервис для управления транспортными перевозками, упрощающий взаимодействие между участниками бизнес процессов. Программа для транспортных компаний и владельцев транспорта. Стоимость от 700 рублей в день, в зависимости от количества сотрудников.

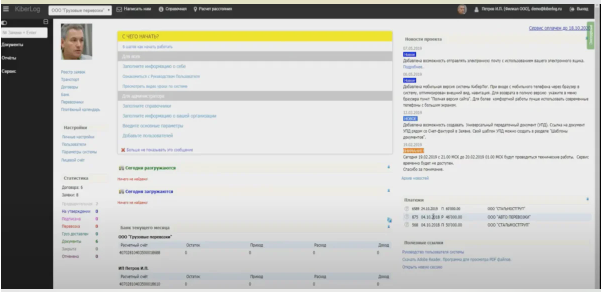


Рисунок 1.4 – Внешний вид программы КиберЛог

**ЯКурьер.** CRM-система для оптимизации транспортного отдела. Мобильное приложение для заказчиков и владельцев транспорта. Стоимость от 1000 рублей в день, в зависимости от количества сотрудников.

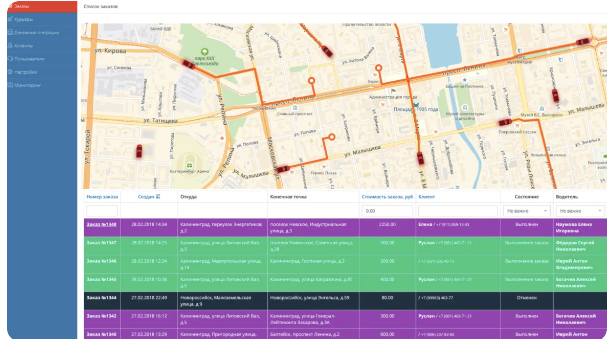


Рисунок 1.5 – Внешний вид программы ЯКурьер

**1С TMS Логистика.** Программа для планирования и учета деятельности транспортных компаний, анализирующая процессы, имеет возможность вести документооборот и организацию многих перевозок. Поддерживает интеграцию с системами ГЛОНАСС и GPS.

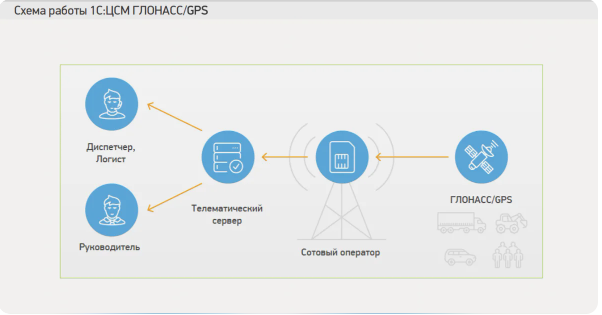


Рисунок 1.6 – Внешний вид программы 1С TMS

Сведенные данные по всем приложениям приведены в таблице 1.1, проанализируем приведенные параметры. Стоимость рассчитано на месяц использования с учетом количества сотрудников.

Таблица 1.1 – Таблица сравнения аналогичных систем учета и распределения маршрутов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Система | Стоимость | Система навигации | Необходимость устанавливать дополнительные системы | Тип приложения |
| 1С TMS Логистика | 35000 | + | + | Серверное |
| КиберЛог | 30000 | + | + | Онлайн сервис |
| ЯКурьер | 45000 | + | + | Серверное и мобильное приложение |
| Максоптра | 31000 | + | + | Серверное |
| УСУ | 57000 | - | - | Приложение на компьютер |

После анализа приведенных приложений и рынка подобных приложений можно отметить следующие плюсы:

Автоматическое распределение маршрутов;

Дополнительный функционал в виде документооборота.

Однако есть и некоторые недостатки:

Часть сервисов представлено в виде облачных сервисов, что не совсем безопасно при наличии чувствительных данных для бизнеса;

Необходимость устанавливать дополнительное приложение для экспедиторов и логистов;

Стоимость внедрения системы превышает выделенные бюджет.

Учитывая достоинства и недостатки целесообразно разработать собственное приложение в виде бота в популярном мессенджере (телеграмм).

## 1.3 Выбор платформы для реализации и адаптация процессов

Как видно из обзора, проведенного в 1.2 существует несколько платформ реализации для автоматизации бизнес процесса:

Веб-сервис;

Мобильное приложение;

Чат-бот.

Рассмотрим применимость перечисленных платформ. Веб сервис – это хорошее функциональное решение для крупных задач в рамках автоматизации процессов, распределенных по местности, не требующих большой мобильности. Так как использовать их в дороге не удобно, занимает большое количество времени и часть информации недоступна.

Главными конкурентами решения такой задачи является мобильное приложение и чат-бот. Однако, важно отметить, что написание приложения для автоматизации отдельного процесса компании нецелесообразно с точки зрения ресурсов (временных, людских и т.д). Кроме того, процесс разработки и развертывания затянуты по времени в отличии с небольшим приложением, коим и является чат-бот. В отличии от мобильного приложения чат-бот нет необходимости устанавливать, как дополнительное приложение. Таким образом, реализация чат-бота для анализируемого процесса рациональнее с точки зрения времени разработки и внедрения в бизнес-процессы.

Разложим процесс, представленный на рисунке 1.7, на подпроцессы с учетом выбранной платформы реализации. Таким образом можно выделить следующие основные подпроцессы:

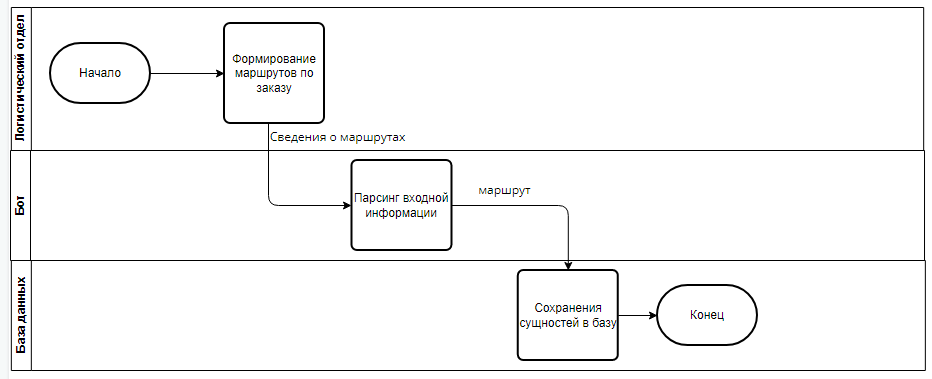


Рисунок 1.7 – Диаграмма процесса сохранения информации о маршрутах

На рисунке 1.7 представлена диаграмма последовательности взаимодействия бота с пользователем с целью добавления новых маршрутов. Полученные сведения о заказе логист преобразует в маршрут и отправляет бот сообщением. Получив сообщение, бот преобразует его при помощи парсинга входного сообщения. Выделив информацию из сообщения бот отправляет его для сохранения в базу данных.

На рисунке 1.8 представлена диаграмма последовательностей предоставления информации о маршрутах.

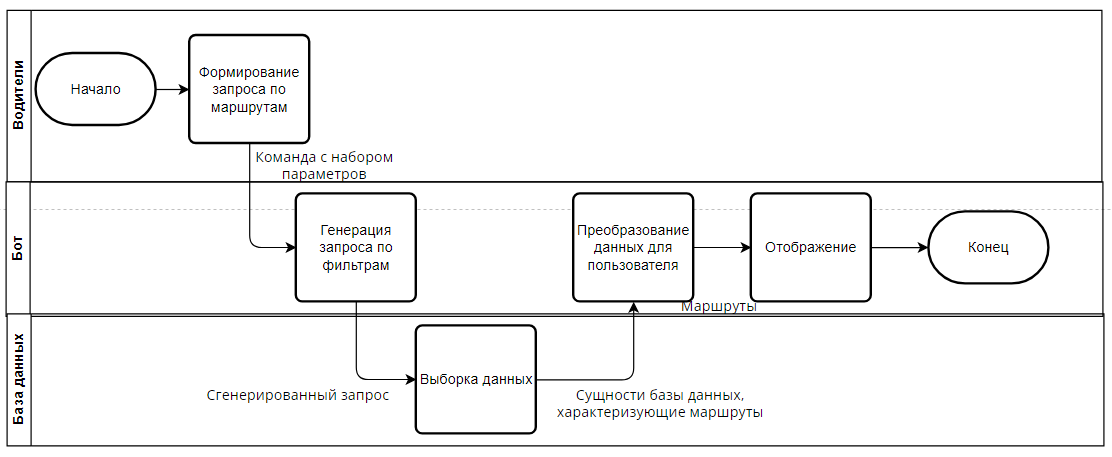


Рисунок 1.8 – Диаграмма последовательности выполнения запроса маршрутов

Как видно из рисунка 1.8 для получения списка маршрутов водителю (процесс аналогичен для всех пользователей) необходимо отправить команду боту. После приема и обработки команды бот производит формирование запроса по фильтрам данных из команды. После чего происходит обращение к базе данных, полученные данные преобразуются к виду понятному для пользователей и отправляются получателю.

На рисунке 1.9 представлена диаграмма последовательности оповещения о чрезвычайной ситуации.

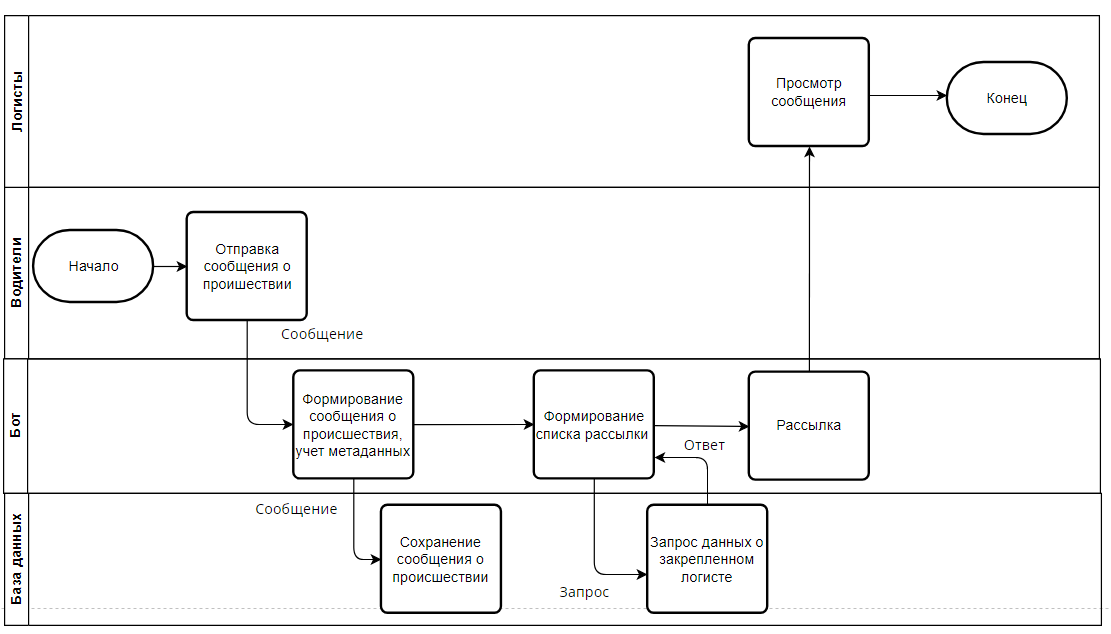


Рисунок 1.9 – Диаграмма последовательности оповещения о чрезвычайной ситуации

В случае чрезвычайного происшествия по маршруту следования водитель отправляет команду в бот, по возможности отправляет текстовое пояснение. При получении команды о ЧП бот запрашивает метаданные пользователе (например, координаты). Сформированное сообщение о ЧП бот сохраняет в базе данных, а затем формирует список рассылки, предварительно получив список закрепленных контактов, и отправляет им сообщение о произошедшем.

На рисунке 1.10 представлена диаграмма выполнения процесса регистрации пользователя. Данный процесс не представлен на рисунке 1.1, но является неотъемлемой частью современных информационных систем для обеспечения безопасности информации.

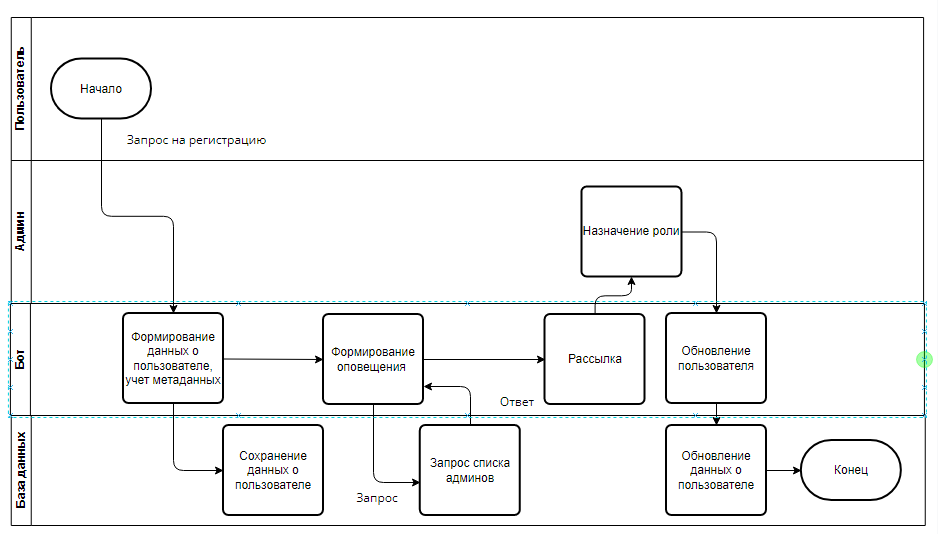


Рисунок 1.10 – Диаграмма регистрации пользователя

При необходимости зарегистрироваться в боте пользователь отправляет команду на регистрацию. Бот, приняв команду, обрабатывает метаданные о пользователе (аккаунте), запрашивая данные о имени, фамилии, идентификаторе и др., сохраняя информацию в базе данных. После сохранения информации о пользователе начинается процесс оповещения админов чата, после получения информации из базы данных формируется сообщение об добавлении нового пользователя, после чего админ вызывает на отображение панель регистрации со списком новых пользователей и выбирает требуемую роль.

На рисунке 1.11 отражена диаграмма процесса получения отчета для руководства компании.

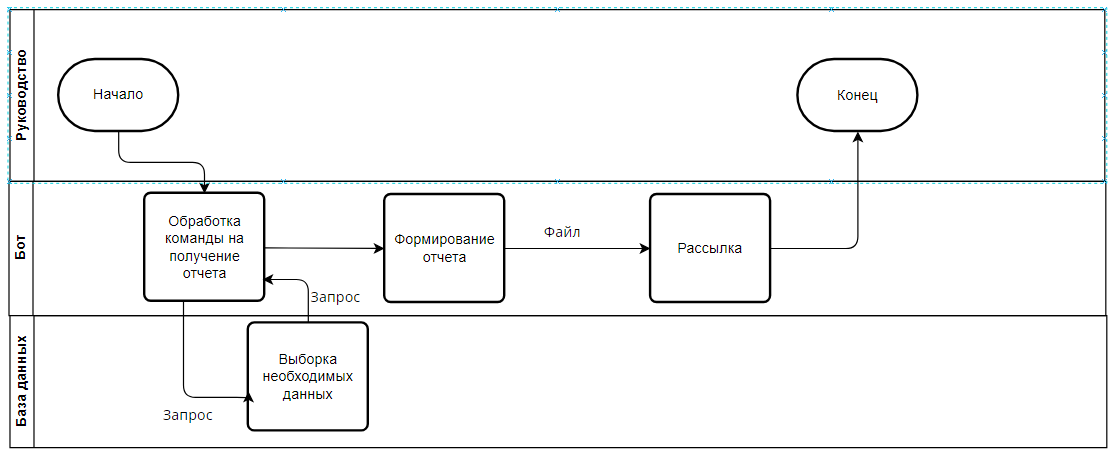


Рисунок 1.11 – Диаграмма процесса получения отчета

Для получения отчета, руководство компании отправляет команду для получения запроса. Далее бот обрабатывает входные данные и генерирует запрос в базу данных. Получив требуемые данные, бот формирует отчет в виде таблицы, после чего отсылает пользователю.

## 1.4 Разработка требований к информационной системе

Для формирования дальнейших требований к системе, сформулируем ее границы.

Система предназначена для автоматизированного учета маршрутов доставки аграрной продукции. Внешними абонентами системы считаются все сотрудники предприятия. Основной целью бота является учет и контроль маршрутов доставки аграрной продукции, а также подготовка отчетов для руководства компании.

*После анализа работы необходимо выделить следующие общие требования к системе:*

программное обеспечение должно соответствовать требованиям импортозамещения Российской Федерации;

стоимость внедрения и эксплуатации не должна превышать десятой части бюджета, выделяемого на обеспечение логистических целей предприятия;

у автоматизированной системы должна быть простая и интуитивно понятная система навигации;

язык реализации интерфейса – русский.

*Исходя из декомпозиции задачи выделим основную функциональность системы:*

регистрация маршрутов доставки продукции;

контроль статуса маршрута по контрольным точкам;

назначение водителя на маршрут;

подготовка отчетов о прохождении маршрутов;

разделение доступа к функциональности.

*Дополнительная функциональность представляет собой следующий перечень возможностей:*

подсистема взаимопощи для экспедиторов;

управление ролями созданной системы;

журналирование действий пользователей системы

управление ролями и пользователями должен осуществлять непосредственно администратор автоматизированной системы.

# Технологический раздел

## 2.1 Архитектура системы

Для построения системы учета заявок целесообразно выбрать клиент-серверную модель построения приложения, а именно веб-приложение. Это обеспечит гибкость использования системы на удаленных точках предприятия, без дополнительных настроек рабочих станций.

Для построения веб систем, как правило, применяют трехзвенную архитектуру. Первый слой является слой взаимодействия с пользователем интерфейс либо командная оболочка. Второй слой является реализацией бизнес-логики, решая задачи преобразования данных, расчетов и др. Третий слой, зачастую представляет собой некое хранилище данных.

Внешним слоем являются наиболее часто меняющиеся составляющие, например, взаимодействие с пользователями (web, cli, api), конкретная реализация паттернов для взаимодействия с внешними источниками. Визуальное представление архитектурного подхода представлено на рисунке 2.1.

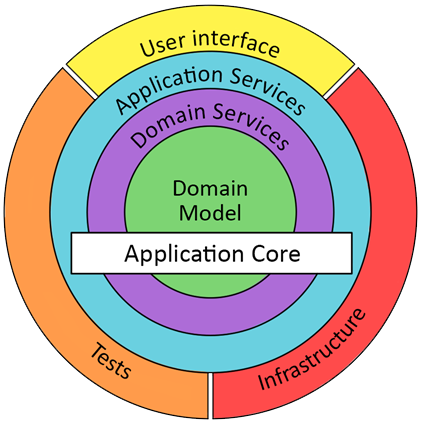


Рисунок 2.1 – Представление луковой архитектуры

Для разрабатываемой автоматизированной системы целесообразно разработать следующие слои:

* модели доменных областей (определение основных сущностей в классах и функциональности в интерфейсах);
* слой доступа к данным (организация работы с базовыми сущностями, работа с базой данных);
* слой приложения – реализация основной логики;
* слой команд – объединения функционала на команды для удобства работы с интерфейсом телеграмма.

Основным паттерном проектирования можно выделить IReposirory паттерн для работы с базой данных, что обеспечивает быстрый переход от одного поставщика данных на другой при необходимости, увеличивает используемость кода, что позволяет следовать принципу разработки DRY (don’t repeat yourself).

Для разработки непосредственно веб части приложения, был использован подход MVC (model view controller) [4]. Визуально, эту архитектуру можно представить, как взаимосвязанный круг преобразования составных частей (рисунок 2.2).

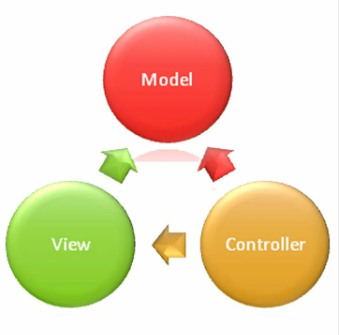


Рисунок 2.2 Внешний вид архитектуры MVC

В схеме MVC модель является независимым компонентом – любые изменения контроллера или представления никак не влияют на модель. Контроллер и представление являются относительно независимыми компонентами. Так, из представления можно обращаться к определенному контроллеру, а из контроллера генерировать представления, но при этом нередко их можно изменять независимо друг от друга.

Такое разграничение компонентов приложения позволяет реализовать концепцию разделение ответственности, при которой каждый компонент отвечает за свою строго очерченную сферу. В связи с чем легче построить работу над отдельными компонентами. И благодаря этому приложение легче разрабатывать, поддерживать и тестировать отдельные компоненты.

## 2.2 Выбор средств разработки

2.2.1 Выбор языка программирования

Основными конкурентами для разработки веб приложений являются Java, JavaScript, python и C#. Рассмотрим подробнее каждый язык программирования и связанные с ними платформы.

Java – строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Разработка ведётся сообществом, организованным через Java Community Process; язык и основные реализующие его технологии распространяются по лицензии GPL. Имеет статическую строгую типизацию и автоматическое управление памятью. Основные области применения: игры, серверные приложения, веб-приложения, программы для сопровождения кода, встраиваемые системы, технологии больших данных и др.

Python – высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Основные направления использования: анализ данных, написание скриптов для автоматического управления инфраструктурой, веб приложения и др.

JavaScript – изначально язык программирования создавался для расширения функционала и возможностей обычных веб-страниц. На современном этапе развития JavaScript, при помощи среды выполнения Node.js, может быть использован и как язык программирования для написания серверной части приложения. Node.js – это кроссплатформенная среда с открытым исходным кодом для разработки серверных и сетевых приложений. Приложения написаны на JavaScript, выполняющиеся в среде исполнения Node.js, работают в таких операционных системах как ОС X, Microsoft Windows и Linux. Node.js использует управляемую событиями, неблокирующую модель ввода-вывода, которая делает его простым и эффективным, идеальным для приложений с интенсивным использованием данных в реальном времени, работающих через распределенные устройства.

С# представляет собой строго типизированный объектно-ориентированный язык программирования с автоматизированным управлением памятью. C# наиболее полно раскрывается при совместном использовании платформы .NET. .NET (ранее известна как .NET Core) это модульная платформа для разработки программного обеспечения с открытым исходным кодом. Совместима с такими операционными системами как Windows, Linux и macOS, то есть, является кроссплатформенной. Спектр использования весьма широк, начиная от десктопных приложений, заканчивая высоконагруженными сервисами. Для написания веб приложений был разработан новый фреймворк от Microsoft. Его преимуществами являются: единое решение для создания пользовательского веб-интерфейса и веб-API, Razor Pages упрощает написание кода для сценариев страниц и повышает его эффективность, Blazor позволяет использовать в браузере язык C# вместе с JavaScript, совместное использование серверной и клиентской логик приложений, написанных с помощью .NET, открытый исходный код и ориентация на сообщество, поддержка размещения служб удаленного вызова процедур (RPC) с помощью gRPC, облачная система конфигурации на основе среды, встроенное введение зависимостей [21].

Существуют еще несколько других популярных решений, таких как PHP или Ruby on Rails. Эти платформы были отклонены по причине невозможности быстро их изучить и реализовать требуемый проект в разумные сроки.

Сравнение платформ приведено в таблице 2.

Таблица 2 – Сравнение платформ разработок

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Параметры | С# | Java | Python | JavaScript |
| Типизация | Строгая | Строгая | нет | нет |
| Эффективность асинхронных операций | высокая | средняя | низкая | низкая |
| Средства разработки | От авторов платформы, бесплатные, удобные | От авторов платформы, бесплатные, неудобные | Есть бесплатные, удобство разработки посредственное | Есть бесплатные, удобство разработки посредственное |
| Скорость работы | Высокая | Высокая | Низкая | Средняя |

Таким образом, для реализации серверной части был выбран C# – объектно-ориентированный язык программирования, основные причины выбора следующие:

строгой типизацией, что упрощает отладку приложения;

хорошей оптимизацией асинхронных операций;

наличие большого количества фреймворков и библиотек, что существенно упрощает разработку;

наличие средств разработки от разработчика платформы, делающие разработку на платформе намного проще;

доступность материалов по изучению и большое сообщество, которое позволяет уменьшить количество нерешаемых проблем.

2.2.2 Выбор СУБД

Основными типами в веб-разработке являются реляционные базы данных и документно-ориентированные. Вторые используются, когда структура данных до конца неизвестна или дублирование данных не критично для рассматриваемой системы. Реляционные позволяют минимизировать объем данных для хранения. Как правило, реляционные базы данных целесообразно выбирать, когда разработчик имеет представление о структуре данных, бизнес-процессы очевидны или устоялись. Немаловажным фактором в выборе СУБД играет роль количество доступных администраторов на рынке труда. Исходя из приведенного описания реляционных баз данных и документно-ориентированных, целесообразно использовать документо-ориентированный тип СУБД для обеспечения гибкости при работе с информацией, также данный тип позволит быстро вводить дополнения и изменения функциональности бота без вмешательства в базу данных.

Самой распространенной СУБД такого типа является MongoDB. Использование MongoDb обусловлено наличием бесплатных онлайн платформ в интернете, что позволит использовать СУБД как сервис совершенно бесплатно. Вторым аргументом большое количество специалистов на рынке труда, что обеспечит поиск сотрудника при масштабировании проекта. Также важным аспектом является скорость чтения из базы данных, она значительно превышает скорость реляционных баз данных, что в данной системе важнее, так как процесс записи будет происходить куда как реже. Также особенность работы базы данных и способ хранения данных обеспечит более удобное расположение данных, что дополнительно увеличит производительность приложения. Сводная таблица анализа серверов базы данных представлена в таблице

Таблица 2.1 – Оценка СУБД

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СУБД | Оценка  производительности | Конкурентный  доступ | Стабильность работы при нагрузке | Поддержка распределенных БД | Работа с ролями пользователей |
| MS  SQL Server | 5 | 6 | 4 | 6 | 8 |
| Oracle | 5 | 9 | 9 | 8 | 9 |
| MS Access | 4 | - | - | - | - |
| MongoDb | 8 | 6 | 9 | 8 | 8 |

Из таблицы 2.1 видно, что документноориентированная база данных (MongoDb) – предпочтительна к использованию в проекте.

## 2.3 Проектирование информационной системы

Информационную систему удобно разбить на три, практически независимые, большие составляющие: пользовательский интерфейс, серверный функционал и база данных. Опишем процесс проектирования каждой составляющей более подробно.

### 2.3.1 Проектирование интерфейса бота

Графически интерфейс будет соответствовать интерфейсу телеграмм ботов и мессенджера вцелом. При проектировании интерфейса целесообразно учесть только навигационные переходы, представленны на рисунке 2.3.

Все представленные блоки представляют собой команды, по которым и происходит навигация по функционалу. Доступность того или иного блока предусматривает наличие соответствующего функционала.



Рисунок 2.3 – Навигация по командам

Все кнопки необходимо реализовать в инлайн режиме, пример оформления представлен на рисунке 2.4.



Рисунок 2.4 – Внешний вид инлайн кнопок телеграмма

### 2.3.2 Проектирование базы данных

При проектировании базы необходимо учесть следующие требования:

1. Разграничение функциональности.
2. Обеспечение сохранения циркулирующих данных.

Для разграничения функциональности введем сущность ApplicationUser. Атрибутами сущности являются: Id – уникальный идентификатор, FirstName – имя пользователя, LastName – фамилия пользователя, ChatId – идентификатор пользователя в телеграмм, IsRegistered – флаг регистрации пользователя, Role – роль пользователя.

Для обеспечения основной функциональности требуется реализовать сущность Route. Атрибуты сущности: Id – уникальный идентификатор, Name – название маршрута, Kilo – вес груза, Goods – название товара, DriverChatId – идентификатор водителя в телеграмм; LogistChatId – идентификатор логиста в телеграмм, AppointDate – дата создания маршрута, FullffilDate – дата выполнения маршрута, Points – набор контрольных точек.

Сущность контрольная точка представляет собой совокупность атрибутов: Id – уникальный идентификатор, Name – название точки, IsFullfil – флаг выполнения точки.

Ввиду того, что выбрана MongoDb явно выраженных связей в базе данных нет, только ссылки на необходимые сущности, ввиду этого концептуальная модель будет содержать в себе только названия таблиц, представленных выше.

Логическая модель базы данных представлена на рисунке 2.3, стрелками указано на что ссылаются поля в базе данных.



Рисунок 2.5 – Логическая модель базы данных

Ввиду того, что выбрана не реляционная, а документно-ориентированная база данных нормализовать базу данных нет необходимости.

### 2.3.3 Проектирование функциональности

Для проектирования дальнейшего функционала следует предусмотреть роли пользователей системы и разграничить между ними возможности использования приложения. Для дальнейшей работы предлагается использовать следующие роли:

User – любой пользователь телеграмм, может нажать кнопку зарегистрироваться, но функционал будет не доступен, пока админ не одобрит заявку.

Admin – роль администратора, доступен функционал управления пользователями, предназначена для системных инженеров и назначенных лиц для администрирования системы.

Driver – водитель, доступен функционал назначения на маршрут, отправки сообщения о помощи или затруднении, отметка о выполнении контрольной точки.

Logic – пользователь системы, способен добавить маршрут и назначить водителя, принимает решения при возникновении затруднений.

Manager – управляющие, доступен функционал отчетности.

Accountant – роль для сотрудников отдела бухгалтерии, доступен функционал получения выполненных маршрутов пользователем.

На основе разработанных ролей в системе определим основные Use-Case элементы.

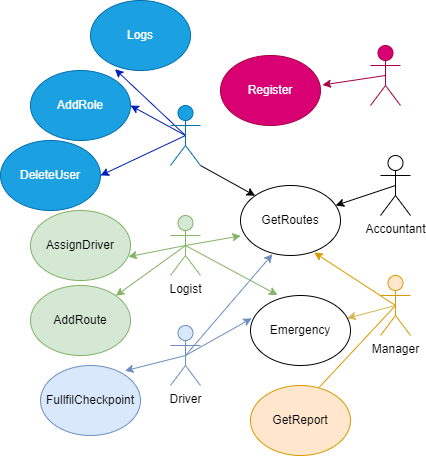


Рисунок 2.6 –Use-Case элементы системы

Далее рассмотрим подробнее use case элементы.

**Register** – use case элемент для регистрации пользователя в систему. Для регистрации в систему пользователь должен зайти в бота и нажать кнопку старт, после чего выбрать кнопку регистрации (единственная кнопка). После чего бот считает его параметры с аккаунта и запишет в базу. Подавшему заявку не предоставляется никакой функциональности. После подачи заявки админу приходит сообщение о том, что кто-то новый зарегистрировался Диаграмма запросов представлена в приложении.

**AddRole** – use case элемент для предоставления прав доступа в систему пользователю, подавшему заявку на участие в боте. Элемент доступен только админу. Для получения доступа необходимо нажать кнопку регистрации в роли админа и на экран выведутся все не проверенные пользователи. Далее, совместно с данными о пользователе размещаются и роли, нажав на которую сформируется команда на изменение роли и, соответственно, предоставление ему прав. Диаграмма запросов представлена в приложении.

**DeleteUser** – use case элемент для удаления пользователя из базы данных. Для того, чтобы удалить пользователя необхоимо нажать кнопку регистрации, потом кнопку «показать всех», после чего высветится список со всеми пользователями. Вместе со списком, напротив каждой фамилии отобразится кнопка «Удалить», нажав на которую, отправиться запрос на удаление пользователя. Элемент доступен только для администратора системы. Диаграмма запросов представлена в приложении.

**Logs** – use case элемент для предоставления логов приложения. В системе необходимо контролировать происходящие процессы и ошибки. По нажатию кнопки старт высветится меню с кнопкой «логи», нажав на которую, пользователю отправятся логи, в зависимости от фильтра. Элемент доступен только администратору. Диаграмма запросов представлена в приложении.

**GetRoutes** – use case элемент для предоставления маршрутов по заданному фильтру, пустой фильтр означает получение всех маршрутов. После входа в бота пользователю предлагается просмотреть информацию о маршрутах. В зависимости от роли пользователя ему будут доступны команды с различными фильтрами. Например, для отдела бухгалтерии доступны только выполненные маршруты. Диаграмма запросов представлена в приложении.

**Emergency** – use case элемент для передачи сообщений о происшествии на маршруте. Доступен только водителю, при нажатии на кнопку происшествие, закрепленному логисту приходит сообщение с описанием происшествия и координатах водителя.

**AddRoute** – use case элемент для добавления маршрута логистом. Для того, чтобы добавить маршрут логист должен либо отослать сообщение в формате json, либо файл с описанием маршрута. Разрешается добавить маршруты списком. Диаграмма запросов представлена в приложении.

**AssignDriver** – use case элемент для назначения водителя на маршрут. Для того, чтобы назначить водителя на маршрут следует нажать на кнопку «маршруты», далее выбрать «не распределенные», после чего высветится список маршрутов и варианты назначения водителей. При просмотре водителями разовых маршрутов доступна кнопка «зарегистрироваться», после ее нажатия маршрут добавиться в список обязательных для водителя. Диаграмма запросов представлена в приложении.

**FullFilCheckPoint** – use case элемент для отметки о выполнении точки в маршруте. Для того, чтобы отметиться на маршруте водитель нажимает кнопку с названием контрольной точки, при отображении информации о маршруте. После нажатия кнопки отсылаются данные о выполнении точки на маршруте. Диаграмма запросов представлена в приложении.

GetReport – use case элемент для предоставления информации в виде отчета (файлом или на экран). Для получения отчета необходимо выбрать в главном меню кнопку отчет, далее выбрать требую информацию (по логистам, водителям или маршрутам), выбрать всех или конкретный объект, файлом или на экран. Процесс выбора предоставляет собой появляющиеся меню на экране. Как только пользователь выбрал все необходимые варианты, отправляется запрос в систему, производит выборку данных, преобразуется к удобному виду и отправляется пользователю. Данный элемент доступен только для руководства компании. Диаграмма запросов представлена в приложении.

# 3. Реализация информационной системы

## 3.1 Реализация базы данных

Спроектированная структура данных была реализована при помощи подхода codeFirst [14]. То есть сначала имплементируются сущности в исходном коде и средствами провайдера данных создаются записи данных в базе.

## 3.2 Реализация функций системы

Планирование работ и их выполнение выполнялось по инкрементальному подходу. Так, для реализации спроектированной системы было запланировано 3 итерации. На первой итерации были реализованы интерфейсы, описывающие функциональность уровня доступа к данным и бизнес логики, классы домена (совпадают с сущностями в базе данных), абстрактные базовые классы для работы с базой данных, сервисов бизнес логики и контроллера. Таким образом на первой итерации были реализованы следующие основные части системы:

IService – интерфейс, описывающий функциональность уровня бизнес-логики. Производными интерфейсами являются IUserService, IRouteService:

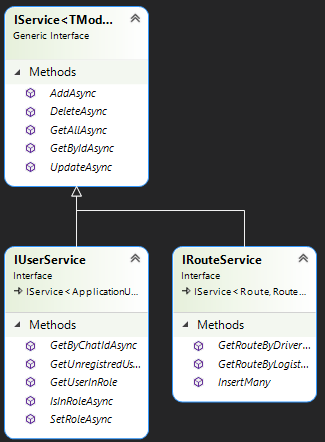


Рисунок 3.0.1 – Диаграмма интерфейсов бизнес-логики

Для бизнес логики работы с маршрутами предусмотрены к реализации следующие функции:

InsertMany – функция предназначена для вставки большого количества маршрутов а раз при заполнении базы из файла;

GetRouteByLogistChatId – запрос маршрутов по номеру логиста;

GetRouteByDriverChatId – запрос маршрутов по номеру водителя.

Для пользователей предусмотрены следующий дополнительный функионал:

GetUserInRole – функция для получения всех пользователей, имеющих заданную роль;

IsInRoleAsync – проверка пользователя на доступ к функционалу, соответствующей заданной роли;

SetRoleAsync – назначить роль пользователю;

GetByChatIdAsync – запрос пользователя по номеру в телеграмм;

GetUnregisteredUsers –запрос на получение всех незарегестрированных пользователей.

IRepository – интерфейс, описывающий функциональность уровня доступа к базе данных. Производными интерфейсами являются IUserRepository, IRouteRepository;

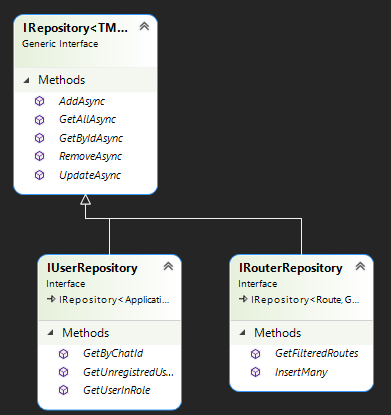


Рисунок 3.0.2 – Диаграмма интерфейсов уровня доступа к базе данных

Интерфейс доступа к базе данных сущности пользователя расширяет подительский интерфейс на 3 метода, а интерфейс сущности маршрутов на 2 метода. Для сущности пользователя это:

GetByChatId – запрос пользователя по номеру в телеграмм;

GetUnregisteredUsers –запрос на получение всех незарегестрированных пользователей.

GetUserInRole – функция для получения всех пользователей, имеющих заданную роль.

А для сущности маршрутов это:

InsertMany – функция предназначена для вставки большого количества маршрутов а раз при заполнении базы из файла;

GetFilteredRoute – запрос на получение списка маршрутов согласно входным фильтрам.

BaseService – базовый абстрактный класс, реализующий общий функционал уровня бизнес-логики.

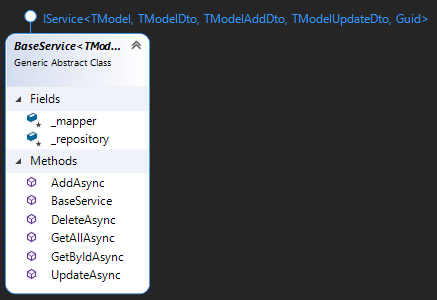


Рисунок 3.0.3 – Диаграмма базового абстрактного класса бизнес логики

Как видно из рисунка 3.3 в классе BaseService были реализованы CRUD операции.

BaseRepository – базовый абстрактный класс, реализующий общий функционал уровня доступа к данным.

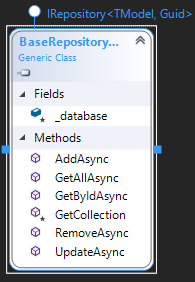


Рисунок 3.0.4 – Диаграмма абстрактного базового класса уровня доступа к данным

Как видно из рисунка 3.3 в классе BaseRepository были реализованы CRUD операции.

На второй итерации были реализованы классы для работы с базой данных каждой сущности и работа бизнес-логики по каждой сущности. А также интерфейс сущности «команда», которая описывает существующие команды в телеграмм.

Как видно из рисунка 3.5 все обязательные функции для классов уровня бизнес-логики, заложенные в первой итерации в интерфейсах, были реализованы. Аналогично реализованы и классы уровня к базе данных, что отражено на рисунке 3.6.

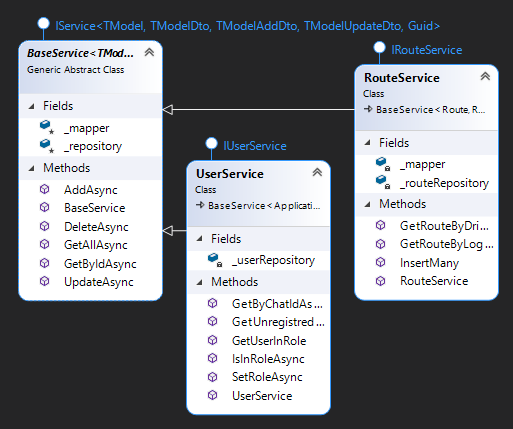


Рисунок 3.0.5 – Диаграмма классов уровня бизнес логики

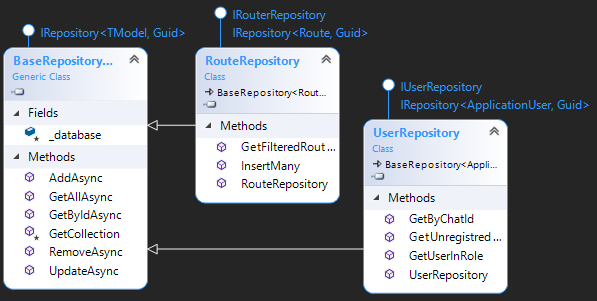


Рисунок 3.0.6 – Диаграмма классов уровня доступа к базе данных

Ввиду того, что взаимоействие пользователя и программы идет через совокупность команд, реализуем интерфейс типовой команды, представлен на рисунке 3.7.

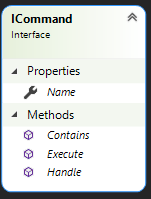


Рисунок 3.0.7 – Диаграмма интерфейса команды телеграмм бота

Как видно из рисунка 3.7 типовая команда состоит из 3 методов:

Contains – проверяет соответствие входящей команды текущей;

Execute – метод выполняется при совпадении входной и текущей команды;

Handle – вызывается для обработки специфичных вызовов, например обратных фызовов (callback).

На третьей итерации были реализованы классы команд, обеспечивающих функциональность бота и контроллер, обеспечивающий взаимодействие с API телеграмма.

Для взаимодействия с API телеграмма необходимо реализовать контроллер с всего лишь одной входной точкой под названием post, принимающей сущность Update. Исходный код контроллера представлен в приложении А.

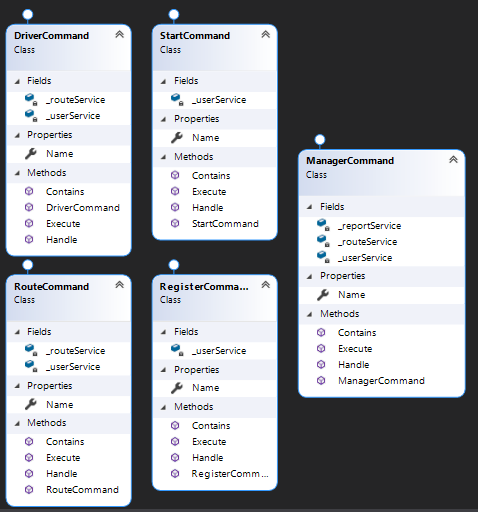


Рисунок 3.0.8 – Диаграмма классов-команд

Типовая команда состоит из названия и дополнительных свойств, предназначенных для навигации (пример реализации команды представлен в приложении).

Для внедрения зависимостей была использована библиотека Microsoft.Extensions.DependencyInjection. В таблице 3.1 представлены метрики кода по итерациям и итоговые.

Как видно из таблицы 3.1, комплексный показатель сильно возрастает на последней итерации. Это обусловлено особенностью написания исходного кода команд, ввиду наличия в них большого количества ветвлений. Глубина наследования достаточна мала, ввиду отсутствия сложных взаимосвязей. Количество строк кода также имеет перекос на третьей итерации, что объясняется написанием функционала команд.

Таблица 3.1 – Таблица метрик качества кода

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метрика | Итерация 1 | Итерация 2 | Итерация 3 | Итого |
| Комплексный показатель качества кода | 116 | 209 | 380 | 380 |
| Глубина наследования | 1 | 2 | 2 | 2 |
| Количество строк кода | 360 | 1062 | 2400 | 2400 |

Таким образом, запланированный функционал был реализован в полном объеме. Написанный код имеет достаточную связность, минимизирующую повторяемость кода, но, в тоже время, не перенасыщая классы реализации. После реализации необходимо приступить к развертыванию системы и ее тестированию.

## 3.3 Развертывание системы

Для доставки программного продукта существуют следующие способы:

1. Копировать сформированный исполняемый файл со всеми необходимыми зависимостями;

2. Распространять подготовленную виртуальную машину с уже установленными зависимостями;

3. Воспользоваться технологией контейнеризации, создать контейнер со всеми необходимыми зависимостями и самим программным обеспечением.

Рассмотрим каждый из представленных пунктов. Первый пункт является самым старым и медленным способом из перечисленных. Для развертывания необходимо установить на сервере или в сети воинской части развернуть систему управления базой данных, прописать настройки подключения к базе данных в файле appsettings.json.

Так как приложения .net core успешно функционируют на операционных системах как Windows, так и на дистрибутивах linux то выбор будет предоставлен инженеру-программисту, назначенному для развертывания. Независимо от операционной системы (далее ОС) необходимо установить платформу .net. Для ОС Windows платформу можно скачать на официальном сайте Microsoft, для дистрибутивов linux можно скачать с официальных репозиториев. Например, для Ubuntu команда будет выглядеть следующим образом:

sudo apt-get update; \

sudo apt-get install -y apt-transport-https && \

sudo apt-get update && \

sudo apt-get install -y aspnetcore-runtime-5.0

После чего будет установлена среда выполнения ASP.NET Core, что позволит запускать приложения, созданные с помощью версии .NET без поддержки среды выполнения.

Если предполагается сборка автоматизированной системы из исходников, при доработке системы, необходимо установить и SDK при помощи команды:

sudo apt-get update; \

sudo apt-get install -y apt-transport-https && \

sudo apt-get update && \

sudo apt-get install -y dotnet-sdk-5.0

Далее выполняется копирование исходного кода из системы контроля версий и сборка проекта. В случае, если развертывается версия по умолчанию этот шаг возможно пропустить.

Далее необходимо использовать один из возможных web-серверов (IIS, Apache, Nginx). Для использования IIS необходимо его настроить согласно инструкции [14], для использования Apache можно воспользоваться литературой [15], а для Nginx [16]. Важно отметить, что приложение, написанное на .net уже содержит свой web-сервер – Kestrel. Kestrel представляет кроссплатформенный веб-сервер, основанный на кросплатформенной библиотеке асинхронного ввода/вывода libuv. Kestrel по умолчанию включается в проект ASP.NET Core.

При инициализации хоста у объекта WebHostBuilder вызывается метод UseKestrel(), который позволяет задействовать Kestrel. Несмотря на то, что по умолчанию исходный код в файле Program.cs не содержит этого вызова, этот метод вызывается автоматически. Однако Microsoft не рекомендует использовать данный web-сервер в продакшене. Основная причина не проработанный до конца механизм защиты от некоторых уязвимостей.

После произведенных настроек автоматизированная система будет доступна для пользователей. Однако, перед ее настройкой администратор должен или использовать интерфейс для создания основных пользователей и присвоения им соответствующих ролей.

Второй вариант доставки кода, использование виртуальных машин заключается в том, что все описанные шаги необходимо выполнить в виртуальной машине и сохранить ее. Далее приходя на место внедрения будет необходимо скопировать виртуальную машину на хост с установленным гипервизором и изменить настройки базы данных в файле appsettings.json. В случае, если сервер базы данных предусмотрен локальный, то и изменять ничего не надо, после развертывания виртуальной машины приложение будет доступно.

Второй подход в развертывании программного обеспечения при помощи виртуальных машин заключается в том, чтобы использовать системы автоматизации их развертывания, например, Packer. Для того, чтобы воспользоваться вторым подходом с использованием виртуальных машин, необходимо описать виртуальную машину на языке hcl, а также написать скрипт развертывания и настройки программного обеспечения, который позволит выполнить все упомянутые в первом способе операции автоматизировано. Преимуществом такого способа является скорость развертывания программного обеспечения, недостатком – объем используемых ресурсов.

Последним упомянутым способом развертывания является использование систем контейнеризации. Этот подход к разработке программного обеспечения, при котором приложение или служба, их зависимости и конфигурация (абстрактные файлы манифеста развертывания) упаковываются вместе в образ контейнера. Контейнерное приложение может тестироваться как единое целое и развертываться как экземпляр образа контейнера в операционной системе (ОС) узла. Для того, чтобы поместить приложение в контейнер необходимо написать специализированный файл. Например, для платформы Docker он называется Dockerfile. В нем описываются все зависимости, которые необходимо предоставить для успешного выполнения приложения в контейнере. Ниже предоставлен листинг Dockerfile для приложений, разработанных на платформе .net core:

FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base

WORKDIR /app

EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build

WORKDIR /src

COPY ["AgroBot /ESC.csproj", " ESC/"]

RUN dotnet restore " AgroBot /ESC.csproj"

COPY . .

WORKDIR "/src/ESC"

RUN dotnet build " AgroBot.csproj" -c Release -o /app/build

FROM build AS publish

RUN dotnet publish " AgroBot.csproj" -c Release -o /app/publish

FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "ESC.dll"]

Из листинга видно, что для создания контейнера используют SDK и runtime которые необходимо устанавливать на хост в первом и втором случае, далее происходит сборка приложения и его запуск. Контейнеры обладают своими уникальными свойствами и характеристиками:

Размер. Контейнеры используют одно общее ядро операционной системы и не используют собственные уникальные операционные системы, поэтому контейнеры являются гораздо меньшими логическими единицами, чем виртуальные машины. Это позволяет компьютеру одновременно размещать гораздо больше контейнеров, чем виртуальная машина.

Скорость. Небольшой размер экземпляров контейнеров позволяет создавать и уничтожать их гораздо быстрее, чем виртуальная машина. Благодаря этому контейнеры хорошо подходят для быстрого масштабирования и краткосрочного использования, что может быть нецелесообразно для виртуальной машины.

Уникальный гипервизор. Для хостинга и управления контейнерами используются специализированные платформы, такие как Docker, rkt и Apache Mesos.

Неизменяемость. В отличие от виртуальной машины, контейнеры не изменяются. Вместо этого контейнерный оркестратор в программном слое контейнера запускает и останавливает контейнеры, когда они необходимы. Аналогичным образом, программное обеспечение, работающее в контейнерах, не обновляется, как традиционное программное обеспечение. Вместо этого обновления включаются в новый образ контейнера, который может быть развернут там, где это необходимо.

После рассмотрения всех способов развертывания приложения следует, что наиболее рациональным выбором развертывания является способ с использованием системы контейнеризации. Однако, приведённый пример написания Dockerfile необходимо дополнить и использовать Docker-compose, для одновременного запуска Apache или Nginx, которые обеспечат безопасность приложения.

Для развертывания был использован следующий docker-compose.yml файл:

version: "3.3"

services:

appserver:

image: lu/app:latest

container\_name: appserver

nginx:

image: lu/final\_nginx:latest

container\_name: proxy

ports:

- "80:80"

links:

- appserver

Таким образом, приложение было развернуто на сервере при помощи средств контейнеризации Docker, в частности при помощи Docker-compose было поднято два контейнера app и final\_nginx. Первый контейнер является приложением, второй прокси сервером, для обеспечения безопасности

# 4. Тестирование информационной системы

## 4.1 Юнит тестирование

Для разработчика важно написание модульных тестов, как правило задача написания модульных тестов и является, зачастую, одной из задач разработчика программного обеспечения. Юнит, модульные тесты – это тесты, которые исследуют небольшие части приложения, которые можно исследовать изолировано. Т.е. можно тестировать методы классов, классы и небольшие библиотеки. Как правило, такие тесты пишут при помощи отдельных фреймворков.

Для написания юнит тестов воспользуемся популярным фреймворком xUnit. Для начала работы с ним необходимо создать новый проект командой:

dotnet new xunit -o AgroBot.Services.Tests

После чего появиться класс с предустановленными зависимостями для проведения тестов (Microsoft.NET.Test.Sdk, xunit, xunit.runner.visualstudio, coverlet.collector). Далее необходимо добавить решение в проект командой:

dotnet sln add / AgroBot.Services.Tests/ AgroBot.Services.Tests.csproj

После чего можно приступать к написанию тестов. Для разрабатываемой автоматизированной системы основным уровнем, подлежащем юнит тестированию подлежит уровень бизнес логики, классы, находящиеся в проекте AgroBot.Services, каждая группа тестов будет соответствовать одному сервису. Следовательно, необходимо создать две группы тестирования.

Для создания одной группы тестирования необходимо создать класс, например, для сервиса UserServices назовем UserServicesTest, для однозначного трактования групп тестирования. Далее в классе необходимо создать экземпляр класса проверяемого класса, мок данные, от которых зависит тестируемый класс, создать тестовые данные, далее создать имитацию зависимостей при помощи пакета из менеджера Mock. При создании инстанса необходимо настроить входящие методы.

В результате создается полноценный инстанс тестируемого класса с иммитацией функциональности его зависимостей.

После создания инстансв тестируемого класса можно приступать к написанию тестов. Тесты начинаются с одного из атрибутов:

Первый атрибут Fact – он обозначает, что далее следует отдельный юнит-тест, не принимающий параметров.

Второй атрибут Theory – это тест, принимающий параметры, при этом может быть несколько сценариев.

И последний, третий, Fixture — класс для настройки и очистки некоторого контекста.

Ввиду того, что тестируемая система в большинстве своем тестирует обычные CRUD операции, то достаточно будет использовать атрибут Fact.

Далее синтаксис совпадает с написанием функции (модификатор доступа, возвращаемый тип, имя, принимаемые параметры, тело функции). Однако есть рекомендации по порядку названия юнит тестов в официальной документации Microsoft, а именно название теста должно содержать название входного типа данных тестируемого приложения, непосредственно само название тестируемой функции, тип возвращаемого значения. Результаты проведения юнит-тестирования представлены на рисунке 4.1.

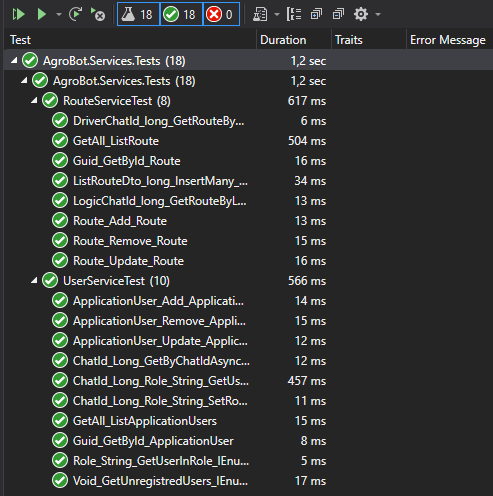


Рисунок 4.1 – Результаты выполнения юнит тестов

Если тесты прошли успешно, значит тестируемый функционал реализован правильно и можно приступать к тестированию более высокого уровня.

Так же можно запустить тесты из командной строки, для этого ввести команду:

dotnet test

Результаты выполенения команды отражен на рисунке 4.2.

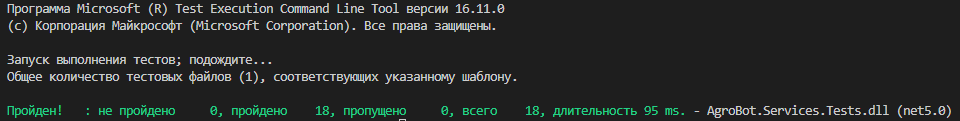


Рисунок 4.2 – Внешний вид выполнения команды dotnet test

Тестирование интеграционное не имеет смысла проводить на проектах такого уровня, ввиду малого объема связей между частями и их очевидности. В таком случае можно переходит к тестированию системы в целом.

## 4.2 Мануальное тестирование

После разработки системы необходимо провести тестирование функционала на предмет адекватности функционирования. Основным видом тестирования по-прежнему является мануальное тестирования. Мануальное тестирование можно разделить на два этапа. Первый этап включает собой проверку функциональности при помощи тестировщика или разработчика после написания системы. Второй этап тестирование ограниченным кругом пользователей на месте функционирования. Ниже представлены результаты проверки функционирования системы.

Работа с простыми сущностями.

Для тестирования простых сущностей необходимо проверить адекватность их создания и удаления. Позитивным случаем будет правильно заполненные поля, негативным – когда одно или более поле пропущено или введены некорректные данные.

Для того что бы проверить функционал добавления необходимо перейти на страницу со списком сущностей и нажать кнопку создать, далее заполнить все поля и нажать кнопку сохранить.

# Заключение

# Список использованной литературы

1. Албахари, Д. C# 6.0. Справочник. Полное описание языка / Д. Албахари, Б. Албахари. – М.: ООО «И.Д. Вильямc», 2016. – 1040 с.
2. Бабаев, З.Д. Анализ финансово-хозяйственной деятельности организации / Бабаев, З.Д., Терехова, В.А., Шеина, Т.Н. – М.: Финансы и статистика, 2016. – 452с.
3. Введение в ASP.NET Core [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1, свободный (Дата обращения: 22.04.2022).
4. Концепция MVC для чайников [Электронный ресурс]. – Режим доступа: https://ruseller.com/lessons.php?, свободный (Дата обращения: 22.04.2022).
5. Куликов, С. Тестирование программного обеспечения. Базовый курс / С. Куликов. – М.: EPAM Systems, 2017. – 296 с.
6. Мюллер, Д. П. C# для чайников / Д. П. Мюллер, Б. Семпф, Ч. Сфер. – М.: Диалектика-Вильямс, 2019. – 608 с.
7. Орлов, С.А. Технологии разработки программного обеспечения / С.А. Орлов, Б.Я. Цилькер. – СПб.: Питер, 2012. – 609 с.
8. Официальная документация MySQL [Электронный ресурс]. – Режим доступа: https://dev.mysql.com/doc/, свободный (Дата обращения: 22.04.2022).
9. Официальная документация SQLite [Электронный ресурс]. – Режим доступа:https://www.sqlite.org/docs.html, свободный (Дата обращения: 22.04.2022).
10. Официальная документация PostgresSQL [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/, свободный (Дата обращения: 22.04.2022).
11. Официальная документация MongoDb [Электронный ресурс]. – Режим доступа: https://www.mongodb.com/docs//, свободный (Дата обращения: 22.04.2022)
12. Официальная документация Visual paradigm [Электронный ресурс]. – <https://www.visual-paradigm.com/support/documents/>, свободный (Дата обращения: 22.04.2022)
13. Паттерны проектирования в C# и .NET [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/patterns/?, свободный (Дата обращения: 22.04.2022).
14. Работа с Entity Framework 6 [Электронный ресурс]. – Режим доступа: https://professorweb.ru/my/entity-framework/6/level1/, (Дата обращения: 22.04.2022).
15. Размещение ASP.NET Core в Windows со службами IIS [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/aspnet/core/host-and-deploy/iis/?view=aspnetcore-5.0 (Дата обращения: 22.04.2022).
16. Размещение ASP.NET Core в операционной системе Linux с Apache [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/ASPNET/core/host-and-deploy/linux-apache?view=aspnetcore-5.0, свободный (Дата обращения: 22.04.2022).
17. Скит, Д. C# для профессионалов. Тонкости программирования / Д. Скит. – М.: ООО «И.Д. Вильямc», 2019. – 608 с.
18. Техническая документация по SQL Server [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver19>, свободный (Дата обращения: 22.04.2022).
19. Уиттакер, Дж. Как тестируют в Google / Дж. Уиттакер, Дж. Арбон, Дж. Каролло. – СПб.: Питер, 2014. – 320 с.
20. Уроки по C# и платформе .NET Framework [Электронный ресурс]. – Режим доступа: https://professorweb.ru/?, свободный (Дата обращения: 22.04.2022).
21. Фриман, А. ASP.NET Core MVC 2 с примерами на C# для профессионалов / А. Фриман. – М.: Диалектика-Вильямс, 2019. – 1008 с.
22. Шапр, Д. Microsoft Visual C# Подробное руководство / Д. Шапр. – Санкт-Петербург: Питер, 2015. – 448 с.
23. Язык программирования C# 7 и платформы .NET и .NET Core [Электронный ресурс]. – Режим доступа: https://www.twirpx.com/file/2901424, свободный (Дата обращения: 22.04.2022).
24. Сьерра К. Head First Java / Кэти Сьерра, Берт Бейтс. – Москва: Эксмо, 2012. – 707 с.
25. Paul Barry. Head First Python: A Brain-Friendly Guide / Paul Barry, – Себастопол: O'Reilly Media, 2016. – 651 с.

# Приложение А

using AgroBot.Models.Interfaces;

using AgroBot.Models.Settings;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Options;

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Telegram.Bot;

using Telegram.Bot.Types;

namespace AgroBot.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class BotController : ControllerBase

{

private TelegramBotClient \_botClient;

private readonly List<ICommand> \_commands;

public BotController(IOptions<BotSettings> options, IBot bot)

{

\_botClient = new TelegramBotClient(options.Value.Token);

\_commands = bot.GetCommands();

}

[HttpPost]

public async Task<OkResult> post([FromBody] Update update)

{

try

{

if (update is null)

return Ok();

Message message;

if (update.Message is null)

{

var callback = update.CallbackQuery;

if (callback is null)

return Ok();

message = callback.Message;

message.Text = callback.Data;

}

else

message = update.Message;

foreach (var command in \_commands)

{

if (command.Contains(message))

{

try

{

await command.Execute(message, update.CallbackQuery, \_botClient);

break;

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

return Ok();

}

catch(Exception ex)

{

return Ok();

}

}

}

}

using AgroBot.Models.Interfaces;

using AgroBot.Models.Interfaces.IService;

using AgroBot.Models.ModelsDB;

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Telegram.Bot;

using Telegram.Bot.Types;

using Telegram.Bot.Types.ReplyMarkups;

namespace AgroBot.Models.Commands

{

public class RegisterCommand : ICommand

{

private readonly IUserService \_userService;

public RegisterCommand(IUserService userService)

{

\_userService = userService;

}

public string Name => @"/register";

public bool Contains(Message message)

{

if (message is null)

return false;

if (message.Type != Telegram.Bot.Types.Enums.MessageType.Text)

return false;

return message.Text.Contains(this.Name);

}

public async Task Execute(Message message, CallbackQuery query, TelegramBotClient client)

{

var chatId = message.Chat.Id;

var exUser = await \_userService.GetByChatIdAsync(chatId);

if (exUser is null)

{

ApplicationUser user = new ApplicationUser

{

ChatId = message.Chat.Id,

FirstName = message.From.FirstName,

LastName = message.From.LastName,

Id = Guid.NewGuid(),

IsRegistred = false

};

var userC = await \_userService.AddAsync(user);

if (userC is null)

await client.SendTextMessageAsync(chatId, "Заявка не принята, обратитесь к админу", Telegram.Bot.Types.Enums.ParseMode.Markdown);

else

{

await client.SendTextMessageAsync(chatId, "Заявка принята", Telegram.Bot.Types.Enums.ParseMode.Markdown);

var admins = await \_userService.GetUserInRole("Admin");

foreach (var item in admins)

{

await client.SendTextMessageAsync(item.ChatId, "новый пользователь", Telegram.Bot.Types.Enums.ParseMode.Markdown);

}

}

}

else

{

if (exUser.Role is null)

await client.SendTextMessageAsync(chatId, "вы уже подали заявку, ожидайте", Telegram.Bot.Types.Enums.ParseMode.Markdown);

else

{

if (await \_userService.IsInRoleAsync(chatId, "Admin"))

{

var cmd = message.Text.Split(" ");

if(cmd.Length == 1)

{

var users = await \_userService.GetUnregistredUsers();

foreach (var item in users)

{

List<List<InlineKeyboardButton>> inlineKeyboardButtons = new List<List<InlineKeyboardButton>>();

InlineKeyboardButton gr442 = new InlineKeyboardButton("Логист");

gr442.CallbackData = @"/register" + " " + item.ChatId + " setrole" + "Logist";

InlineKeyboardButton gr443 = new InlineKeyboardButton("Админ");

gr443.CallbackData = @"/register" + " " + item.ChatId + " setrole" + "Admin";

InlineKeyboardButton gr444 = new InlineKeyboardButton("Водитель");

gr444.CallbackData = @"/register" + " " + item.ChatId + " setrole" + "Driver";

InlineKeyboardButton gr434 = new InlineKeyboardButton("Управление");

gr434.CallbackData = @"/register" + " " + item.ChatId + " setrole" + "Manager";

InlineKeyboardButton gr435g = new InlineKeyboardButton("Бухгалтер");

gr435g.CallbackData = @"/register" + " " + item.ChatId + " setrole " + "Accountant";

var list = new List<InlineKeyboardButton>();

list.Add(gr442);

list.Add(gr443);

list.Add(gr444);

list.Add(gr435g);

list.Add(gr434);

inlineKeyboardButtons.Add(list);

var list2 = new List<InlineKeyboardButton>();

InlineKeyboardButton removeUsers = new InlineKeyboardButton("Удалить пользователя");

removeUsers.CallbackData = @"/register" + " " + item.Id + " " + "Remove";

list2.Add(removeUsers);

inlineKeyboardButtons.Add(list2);

InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup(inlineKeyboardButtons);

await client.SendTextMessageAsync(chatId, item.FirstName + " " + item.LastName, parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown, replyMarkup: inlineKeyboardMarkup);

}

List<List<InlineKeyboardButton>> keybord = new List<List<InlineKeyboardButton>>();

var keybordList = new List<InlineKeyboardButton>();

InlineKeyboardButton allUsers = new InlineKeyboardButton("Показать всех пользователей");

allUsers.CallbackData = @"/register" + " " + "All";

keybordList.Add(allUsers);

keybord.Add(keybordList);

var keybordMain = new List<InlineKeyboardButton>();

InlineKeyboardButton main = new InlineKeyboardButton("Вернуться в главное меню");

main.CallbackData = @"/start";

keybordMain.Add(main);

keybord.Add(keybordMain);

InlineKeyboardMarkup KeyboardMarkup = new InlineKeyboardMarkup(keybord);

await client.SendTextMessageAsync(chatId, "Меню пользователей", parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown, replyMarkup: KeyboardMarkup);

}

if (cmd.Length == 2)

{

var users = await \_userService.GetAllAsync();

foreach (var item in users)

{

InlineKeyboardButton gr434 = new InlineKeyboardButton("Удалить");

gr434.CallbackData = @"/register" + " " + item.Id + " " + "Remove";

List<List<InlineKeyboardButton>> inlineKeyboardButtons = new List<List<InlineKeyboardButton>>();

var list = new List<InlineKeyboardButton>();

list.Add(gr434);

inlineKeyboardButtons.Add(list);

InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup(inlineKeyboardButtons);

await client.SendTextMessageAsync(chatId, item.FirstName + " " + item.LastName, parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown, replyMarkup: inlineKeyboardMarkup);

}

}

if (cmd.Length == 3 && cmd[2] == "Remove")

{

Guid guid = Guid.Empty;

Guid.TryParse(cmd[1], out guid);

await \_userService.DeleteAsync(guid);

}

if (cmd.Length == 4 && cmd[2] == "setrole")

{

await \_userService.SetRoleAsync(long.Parse(cmd[1]), cmd[2]);

await client.SendTextMessageAsync(chatId, "Роль изменена", Telegram.Bot.Types.Enums.ParseMode.Markdown);

}

}

else

{

var ans = string.Format("вы зарегистрированы в качесте");

foreach (var r in exUser.Role)

{

ans += " " + r;

}

await client.SendTextMessageAsync(chatId, ans, Telegram.Bot.Types.Enums.ParseMode.Markdown);

}

}

}

}

public Task Handle(Message message, CallbackQuery query, TelegramBotClient client)

{

throw new NotImplementedException();

}

}

}

using AgroBot.Models.Interfaces.IRepository;

using AgroBot.Models.Interfaces.IService;

using AgroBot.Models.ModelsDB;

using AgroBot.Models.Settings;

using AutoMapper;

using Microsoft.Extensions.DependencyInjection;

using Moq;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using Xunit;

namespace AgroBot.Services.Tests

{

    public class UserServiceTest

    {

        private Mock<IUserRepository> \_userRepository;

        private IUserRepository \_mockUserRepository;

        private IMapper \_mapper;

        private List<ApplicationUser> \_fakeUser;

        private UserService \_userService;

        private static Random \_random;

        public UserServiceTest()

        {

            GenerateData();

        }

        [Fact]

        public async Task<List<ApplicationUser>> GetAll\_ListApplicationUsers()

        {

            GetMockUserService();

            var user = await \_userService.GetAllAsync(default(CancellationToken));

            Assert.Equal(user.Count(), \_fakeUser.Count());

            return (List<ApplicationUser>)user;

        }

        [Fact]

        public async Task<ApplicationUser> Guid\_GetById\_ApplicationUser()

        {

            GetMockUserService();

            var user = await \_userService.GetByIdAsync(\_fakeUser[0].Id, default(CancellationToken));

            Assert.NotNull(user);

            return user;

        }

        [Fact]

        public async Task<ApplicationUser> ApplicationUser\_Remove\_ApplicationUser()

        {

            GetMockUserService();

            var user = await \_userService.DeleteAsync(\_fakeUser[0].Id, default(CancellationToken));

            Assert.NotNull(user);

            return user;

        }

        [Fact]

        public async Task<ApplicationUser> ApplicationUser\_Add\_ApplicationUser()

        {

            GetMockUserService();

            var user = await \_userService.AddAsync(\_fakeUser[0], default(CancellationToken));

            Assert.NotNull(user);

            return user;

        }

        [Fact]

        public async Task<ApplicationUser> ApplicationUser\_Update\_ApplicationUser()

        {

            GetMockUserService();

            var user = await \_userService.UpdateAsync(\_fakeUser[0].Id, \_fakeUser[0], default(CancellationToken));

            Assert.NotNull(user);

            return user;

        }

        [Fact]

        public async Task<ApplicationUser> ChatId\_Long\_GetByChatIdAsync\_ApplicationUser()

        {

            GetMockUserService();

            var user = await \_userService.GetByChatIdAsync(\_fakeUser[0].ChatId, default(CancellationToken));

            Assert.NotNull(user);

            return user;

        }

        [Fact]

        public async Task<IEnumerable<ApplicationUser>> Void\_GetUnregistredUsers\_IEnumerable\_ApplicationUsers()

        {

            GetMockUserService();

            var user = await \_userService.GetUnregistredUsers(default(CancellationToken));

            Assert.Equal(user.Count(), \_fakeUser.Where(p => p.IsRegistred == false).Count());

            return user;

        }

        [Fact]

        public async Task<IEnumerable<ApplicationUser>> Role\_String\_GetUserInRole\_IEnumerable\_ApplicationUsers()

        {

            GetMockUserService();

            var user = await \_userService.GetUserInRole(\_fakeUser[0].Role[0], default(CancellationToken));

            Assert.Equal(user.Count(), \_fakeUser.Count());

            return user;

        }

        [Fact]

        public async Task<bool> ChatId\_Long\_Role\_String\_GetUserInRole\_bool()

        {

            GetMockUserService();

            var user = await \_userService.IsInRoleAsync(\_fakeUser[0].ChatId, \_fakeUser[0].Role[0], default(CancellationToken));

            return user;

        }

        [Fact]

        public async Task<ApplicationUser> ChatId\_Long\_Role\_String\_SetRoleAsync\_ApplicationUsers()

        {

            GetMockUserService();

            var user = await \_userService.SetRoleAsync(\_fakeUser[0].ChatId, \_fakeUser[0].Role[0], default(CancellationToken));

            Assert.NotNull(user);

            return user;

        }

        private void GetMockUserService()

        {

            var services = new ServiceCollection();

            IServiceProvider serviceProvider = services.BuildServiceProvider();

            var myProfile = new MapperProfile();

            var configuration = new MapperConfiguration(cfg =>

            {

                cfg.AddProfile(myProfile);

                cfg.ConstructServicesUsing(serviceProvider.GetService);

            });

            \_mapper = new Mapper(configuration);

            try

            {

                \_userRepository = new Mock<IUserRepository>();

                \_userRepository.Setup(s => s.GetAllAsync(default(CancellationToken))).ReturnsAsync(\_fakeUser);

                \_userRepository.Setup(s => s.GetByIdAsync(It.IsAny<Guid>(), default(CancellationToken))).ReturnsAsync(\_fakeUser[0]);

                \_userRepository.Setup(s => s.GetByChatId(It.IsAny<long>(), default(CancellationToken))).ReturnsAsync(\_fakeUser[0]);

                \_userRepository.Setup(s => s.GetUserInRole(It.IsAny<string>(), default(CancellationToken))).ReturnsAsync(\_fakeUser);

                \_userRepository.Setup(s => s.GetUnregistredUsers(default(CancellationToken))).ReturnsAsync(\_fakeUser.Where(p => p.IsRegistred == false).ToList());

                \_userRepository.Setup(s => s.RemoveAsync(It.IsAny<Guid>(), default(CancellationToken))).ReturnsAsync(\_fakeUser[0]);

                \_userRepository.Setup(s => s.AddAsync(It.IsAny<ApplicationUser>(), default(CancellationToken)));

                \_userRepository.Setup(s => s.UpdateAsync(It.IsAny<Guid>(),  It.IsAny<ApplicationUser>(), default(CancellationToken)));

            }

            catch (Exception ex)

            {

                Console.WriteLine(ex.Message);

            }

            \_mockUserRepository = \_userRepository.Object;

            \_userService = new UserService(\_mockUserRepository, \_mapper);

        }

        private void GenerateData()

        {

            \_random = new Random();

            \_fakeUser = new List<ApplicationUser>();

            for (int i = 0; i < 10; i++)

            {

                \_fakeUser.Add(

                    new ApplicationUser

                    {

                        Id = Guid.NewGuid(),

                        ChatId = \_random.Next(1,100000),

                        FirstName = RandomString(5),

                        LastName = RandomString(8),

                        IsRegistred = false,

                        Role = new List<string>

                        {

                            RandomString(7)

                        }

                    });

            }

        }

        private string RandomString(int length)

        {

            const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

            return new string(Enumerable.Repeat(chars, length)

                .Select(s => s[\_random.Next(s.Length)]).ToArray());

        }

    }

}