СОДЕРЖАНИЕ

[Введение 3](#_Toc102236966)

[1 Предпроектный анализ 4](#_Toc102236967)

[1.1 Характеристика предприятия 4](#_Toc102236968)

[1.2 Описание предметной области 5](#_Toc102236969)

[1.3 Разработка требований к информационной системе 9](#_Toc102236970)

[2 Обзор аналогов и литературы 12](#_Toc102236971)

[2.1 Обзор аналогов систем учета заявок 12](#_Toc102236972)

[2.2 Обзор литературы 17](#_Toc102236973)

[3 Технологический раздел 19](#_Toc102236974)

[3.1 Архитектура системы 19](#_Toc102236975)

[3.2 Выбор средств проектирования 20](#_Toc102236976)

[3.3 Выбор средств разработки 20](#_Toc102236977)

[3.3.1 Выбор языка программирования 20](#_Toc102236978)

[3.3.2 Выбор СУБД 23](#_Toc102236979)

[3.3.3 Архитектура приложения 24](#_Toc102236980)

[4 Проектирование и реализация информационной системы 27](#_Toc102236981)

[4.1 Проектирование информационной системы 27](#_Toc102236982)

[4.1.1 Проектирование интерфейса пользователя 27](#_Toc102236983)

[4.1.2 Проектирование базы данных 28](#_Toc102236984)

[4.1.3 Проектирование функциональности 31](#_Toc102236985)

[4.2 Реализация информационной системы 41](#_Toc102236986)

[4.2.1 Реализация базы данных 41](#_Toc102236987)

[4.2.2 Реализация функций системы 42](#_Toc102236988)

[4.3 Тестирование информационной системы 47](#_Toc102236989)

[4.3.1 Юнит тестирование 47](#_Toc102236990)

[4.3.2. Мануальное тестирование 51](#_Toc102236991)

[4.4 Развертывание системы 51](#_Toc102236992)

[5 Информационная безопасность 57](#_Toc102236993)

[Заключение 58](#_Toc102236994)

[Список использованной литературы 59](#_Toc102236995)

# Введение

С начала разработки первых компьютеров началась информатизация общества, продолжив дело механизации. Одной из главных целей информатизации является разгрузить человека в производственных цепочках и сделать его жизнь и работу проще. В рамках цифровизации общества с одной стороны и решения конкретной задачи реальной жизни с другой стороны, в данной дипломной работе представлена система, автоматизирующая процесс распределения .

# Анализ предметной области

Целью данного раздела является обзор целевого предприятия и предмета автоматизации. Как следует из названия дипломной работы целевой функцией дипломной работы является автоматизация процесса учета и распределения рейсов по доставке аграрной продукции к точкам сбыта.

## 1.1 Описание предметной области

В настоящее время распределение по маршрутам проходит вручную логистом. В случае, когда один из нескольких водителей заболевает не всегда есть возможность своевременно отреагировать и оповестить

## 1.2 Обзор аналогов систем контроля экспедиторской деятельности

**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 30000 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.

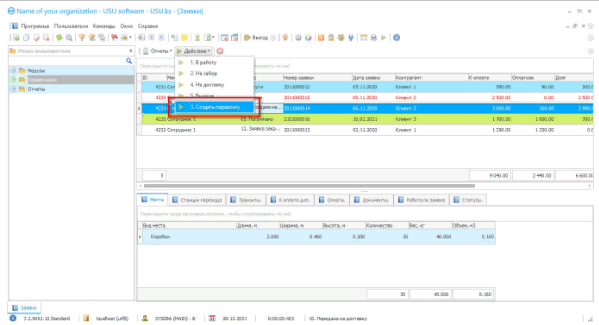


Рисунок . Внешний вид интерфейса универсальной системы учета

**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.

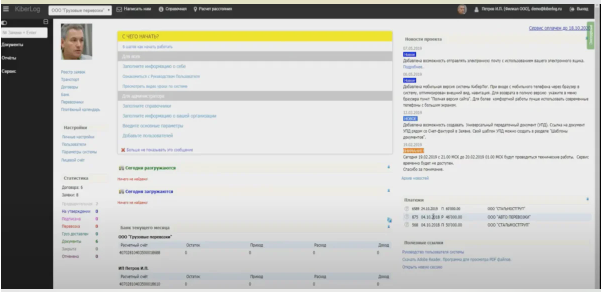


Рисунок . – Внешний вид программы КиберЛог

**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.

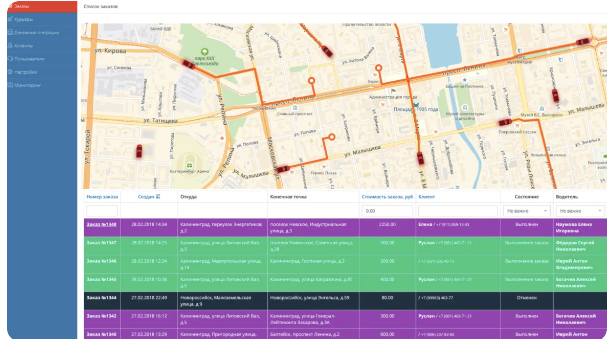


Рисунок . – Внешний вид программы ЯКурьер

**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.

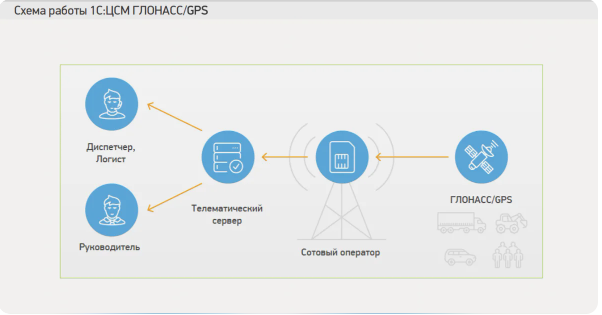


Рисунок . – Внешний вид программы 1С TMS

## 1.3 Обзор литературы

Для разработки системы необходимо обладать знаниями о порядке и проведении планирования работы, знать методику и правила построения программ, уметь выстроить архитектуру разрабатываемой системы, владеть средствами разработки и многое другое. Выделим основные области знания и разобьем из на группы.

Первой группой является группа, обеспечивающая формализацию реальных процессов в понятные разработчику функции, и, последующая декомпозиция функций на конкретные задачи. Этот процесс хорошо описан в литературе приведенной в списке используемой литературы [2, 8,]. Практические действия подробно описаны в [13].

После анализа бизнес процессов и декомпозиции функций на более мелкие начинается процесс синтеза. Первым этапом является выбор архитектуры и принципов разработки, некоторые основы описаны в [14].

Следующим этапом является выбор стека для разработки приложения. В стек для разработки включается выбор языка и платформы программирования, системы управления базой данных, средства развертывания и мониторинга системы. Для выбора из всего многообразия средств разработки всех составляющих системы необходимо произвести обзор популярных платформ разработки. Так, для анализа основных платформ разработки и языков программирования были проанализированы [1, 3, 7, 14, 16, 17, 20, 23, 24, 25, 26, 27, 28]. Для выбора системы управления базами данных были проанализированы основные, представленные на рынке решения [9, 10, 11, 12, 21].

Следующей составляющей при разработке является тестирование приложений, подробный процесс тестирования, а также теоретические основы и практика применения хорошо описаны в [6, 22].

Процесс развертывания систем описан к каждой платформе разработки и СУБД, например [16, 17].

## 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 разработанной системы 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:*

система должна быть построена на основе клиент-серверных технологий, предпочтительно веб-решения;

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.

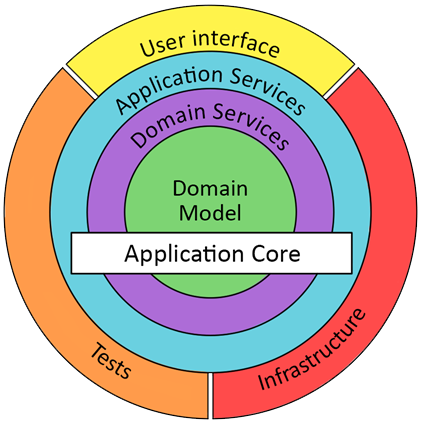


Рисунок . – Представление луковой архитектуры

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.

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) [5]. Визуально, эту архитектуру можно представить, как взаимосвязанный круг преобразования составных частей (рисунок 2.2).

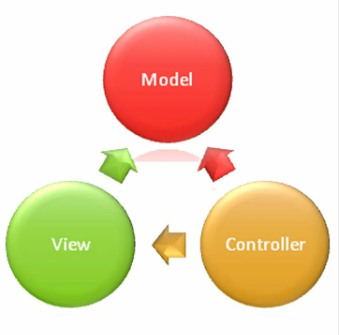


Рисунок . Внешний вид архитектуры MVC

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 [24].

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 реляционный тип СУБД для минимизации объема хранимой информации, исключении дублирования и приемлемой скоростью работы. Среди реляционных баз данных наиболее популярными являются MSSQL, PostgreSQL, MySQL, Oracle.

Для реализации базы данных для разрабатываемой автоматизированной системы была выбрана MSSQL. Выбор был обусловлен популярностью использования (при необходимости увеличения штата легко найти разработчика для администрирования базы данных), хорошими показателями чтения/записи данных.

## 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 базы данных

При проектировании базы необходимо учесть следующие требования:

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

Для обеспечения требование из первого пункта целесообразно использовать фреймворк AspNet.Identity. Данный фреймворк предусматривает создание следующих сущностей в базе данных:

Сущность AspNetRoles хранит информацию о ролях пользователей. Атрибуты сущности: Name – название; Id – уникальный идентификатор роли;

NormalizedName – нормализованное название роли; ConcurrencyStamp – случайное значение, которое должно изменяться при каждом сохранении роли в хранилище.

Сущность AspNetUserClaims хранит информацию о дополнительной информации о пользователях, на основе которой можно принимать решения по авторизации: Id – уникальный идентификатор; RoleId – уникальный идентификатор пользователя; ClaimType – тип claim; ClaimValue – занчения claim.

Сущность AspNetRolesClaims хранит информацию о дополнительной информации о ролях, на основе которой можно принимать решения по авторизации: Id – уникальный идентификатор; UserId – уникальный идентификатор пользователя; ClaimType – тип claim; ClaimValue – занчения claim.

Сущность AspNetUserRoles хранит информацию о связи пользователя и существующих ролей: RoleId – уникальный идентификатор пользователя роли; UserId – уникальный идентификатор пользователя.

Сущность AspNetUsers хранит информацию о пользователях: Id – уникальный идентификатор пользователя; AccessFailedCount – количество неудачных попыток получения доступа; FirstName – имя пользователя; LastName – фамилия пользователя; MiddleName – отчество пользователя; PositionId – идентификационный номер должность пользователя; DepartmentId – идентификационный номер подразделения, к которому относится пользователь; ConcurencyStamp – случайное значение, которое должно изменяться при каждом сохранении роли в хранилище; Email – адрес электронной почты; EmailConfirmed – флаг подтверждения электронной почты; LocoutEnable – флаг разрешения блокировки пользователя; LockoutEnd – флаг блокировки пользователя; NormilezedEmail – нормализованный адрес электронной почты; NormalizedUserName – нормализованное имя пользователя; PasswordHash – хэш пароля; PhoneNumber – номер телефона; PhoneNumberConfirmed – флаг подтверждения номера телефона; SecurityStamp – случайное значение, которое должно изменяться при каждом изменении учетных данных пользователя (изменение пароля, удаленное имя входа); TwoFactorEnabled – флаг о разрешении двухфакторной аутентификации; UserName – имя пользователя.

Сущность AspNetUserLogins хранит информацию об имени входа и связанный с ним поставщик для пользователя. Атрибуты сущности: LoginProvider – поставщик входа для имени входа; ProviderDisplayName – понятное имя, используемое в пользовательском интерфейсе для этого имени входа; ProviderKey – уникальный идентификатор поставщика для данного имени входа; UserId – уникальный идентификатор пользователя.

Сущность AspNetUserTokens хранит маркер проверки подлинности для пользователя. Атрибуты сущности: LoginProvider – поставщик входа для имени входа этого токена; Name – название токена; UserId – уникальный идентификатор пользователя; Value – непосредственно токен.

Все эти сущности обеспечивают полный функционал для обеспечения разраничения доступа в системе согласно требуемых ролей.

Для обеспечения функционирования целевой функциональности необходимо создать следующие сущности:

Department – хранит информацию о существующих подразделениях. Сущность включает в себя следующие атрибуты: Id – уникальный идентификатор; DepartmentName – название отдела; Description – описание отдела; Value – приоритет отдела.

RepairRequest – хранит информацию о заявках и содержит следующие атрибуты должность: Id – уникальный идентификатор; RequestName – название заявки; RequestDescription – подробное описание заявки; ServiceName – название проблемной службы; Stage – состояние заявки; RequesterId – уникальный идентификатор автора заявки; RegisterDate – дата регистрации заявки; RequestTypeId – уникальный идентификатор типа заявки; FixDate – дата выполнения заявки; WorkerId – уникальный идентификатор сотрудника, назначенного для выполнения заявки; IsDeleted – флаг выполнения заявки; WorkerRate – отметка пользователю сотруднику, выполнившему заявку; RequesterComment – комментарии пользователя, отзыв.

Сущность RequestType – сущность для хранения типов заявок, содержит следующщие атрибуты: Id – уникальный идентификатор; RequestTypeName – название типов заявок.

Сущнсоть Position хранит информацию о должностях, имеющихся на предприятии, имеет следующие атрибуты: Id – уникальный идентификатор; PositionName – название должности; Value – поле для определения очередности устранения заявок.

### 2.3.2 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:

Admin – роль администратора, доступен весь функционал в автоматизированной системе, предназначена для системных инженеров и назначенных лиц для администрирования системы.

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

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

Manager – управляющие, доступен функционал отчетности.

На основе разработанных ролей в системе определим основные Use-Case элементы (рисунок 4.2).

# 3. Реализация информационной системы

## 3.1 Реализация базы данных

Спроектированная структура данных была реализована при помощи подхода codeFirst и фреймворка Entity Framework Core [14]. То есть сначала имплементируются сущности в исходном коде и средствами Entity Framework Core (миграции) создается база данных. В таком виде очевидны достоинства при переносе информационной системы на новые сервера и развертывание дополнительных инстансов. Так же для удобства переноса информационной системы и базы данных необходимо только указать сетевой адрес нового сервера и порядок доступа к нему, служебные данные на новом сервере создадутся автоматически (например, роли и учетная запись администратора). Результатом работы является база данных, схема которой представлен на рисунке 4.20.

Рисунок 3.. – Физическая схема базы данных

## 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:

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

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

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

На второй итерации были реализованы классы для работы с базой данных каждой сущности и работа бизнес-логики по каждой сущности.

## 3.3 Тестирование информационной системы

### 4.3.1 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 HelpdeskAirport.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 / HelpdeskAirport.Services.Tests/ HelpdeskAirport.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 HelpdeskAirport.Services, каждая группа тестов будет соответствовать одному сервису. Следовательно, необходимо создать пять групп тестирования.

Для создания одной группы тестирования необходимо создать класс, например, для сервиса ExecuteTypeService назовем ExecuteTypeServiceTest, 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

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

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.

### 4.3.2. 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 тестиривания 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.

## 3.4 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 MsSQL, 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 ["HelpdeskAirport /ESC.csproj", " ESC/"]

RUN dotnet restore " HelpdeskAirport /ESC.csproj"

COPY . .

WORKDIR "/src/ESC"

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

FROM build AS publish

RUN dotnet publish " HelpdeskAirport.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

# 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.