

Содержание

Введение	3
1 Общие сведения и структура предприятия «ИВА-Гомель-Парк».....	4
1.1 История предприятия	4
1.2 Используемое оборудование	6
1.3 Охрана труда на предприятии.....	6
1.4 Должностная инструкция инженера-программиста	9
2 Описание этапов выполнения индивидуального задания.....	11
2.1 Постановка задачи	11
2.2 Архитектура приложения.....	11
2.3 Слой доступа к данным (<i>Data Access Layer</i>).....	13
2.4 Слой бизнес-логики (<i>Business Logic Layer</i>).....	17
2.5 Слой представления (<i>Presentation Layer</i>).....	19
2.6 Верификация программного обеспечения	21
Заключение	29
Список использованных источников	30
Приложение А Текст программы	31

ВВЕДЕНИЕ

Производственная практика является важным этапом в подготовке будущих специалистов и проверки полученных ими знаний на практике. Сложно переоценить её ценность в процессе подготовки специалиста.

В ходе технологической практики необходимо решить следующие задачи:

- закрепить на практике знания о подходах разработки приложений;
- получить опыт работы на предприятии;
- получить навыки работы в команде;
- изучить и проанализировать предметную область, технологии и методы, необходимые для разработки приложения;
- разработаны алгоритмы решения поставленной задачи.

Также в ходе прохождения технологической практики необходимо разработать приложение, которое позволило бы сокращать ссылки большой длины для передачи их в сообщениях с ограничением количества символов и получать статистику в виде таблиц, по имеющимся данным. Также необходимо реализовать систему аутентификации, проверять входные данные пользователя на корректность и подсчитывать количество переходов по сокращённым ссылкам.

1 ОБЩИЕ СВЕДЕНИЯ И СТРУКТУРА ПРЕДПРИЯТИЯ «ИВА-ГОМЕЛЬ-ПАРК»

1.1 История предприятия

IBA Group – это крупная коалиция компаний, которые специализируются в области разработки, производства и поставки решений и услуг в области информационных технологий на территории стран Восточной и Центральной Европы. Штаб-квартира *IBA Group* расположена в г. Праге (Чехия).

Под данным брендом собрано уже более 20 компаний, которые располагаются на территории Республики Беларусь, Чехии, США, Германии, Казахстана, Украины, Словакии, ЮАР, Великобритании, России и на Кипре. После вступления Чехии в Европейский союз в 2005 году местом размещения главного офиса *IBA Group* стала Прага.

За время своего существования *IBA Group* сумели выполнить более 2000 проектов для заказчиков из множества различных секторов экономики, включая такие как транспорт, энергетику, телекоммуникации и информационные технологии.

В 2001 году на территории свободной экономической зоны «Гомель-Ратон» был открыт *IBA Gomel* (Иностранное унитарное предприятие «ИВА-Гомель») – гомельский центр разработок международного холдинга *IBA Group*.

Данное предприятие было создано ради привлечения перспективных IT-специалистов из гомельского региона к разработке современных программных продуктов для различных стран мирового сообщества.

Со дня основания *IBA Group* постоянно работает над созданием, внедрением и совершенствованием систем менеджмента с целью обеспечения стабильного качества выполняемых работ и оказываемых услуг, снижения проектных рисков и обеспечения информационной безопасности. Благодаря этому начиная с 2004 года в компании «ИВА-Гомель» внедрена система менеджмента качества проектирования, разработки и сопровождения ПО. Данная система соответствует международному стандарту менеджмента качества *ISO 9001:2008*, распространяющемуся на проектирование, разработку и сопровождение программного обеспечения, что подтверждается Немецким обществом по аккредитации (*DAkkS TGA*), а так же сертификатом Национальной организации по сертификации. Так же данная система соответствует требованиям СТБ *ISO/IEC 27001* для систем менеджмента информационной безопасности применительно к проектированию, разработке, производству и сопровождению программного обеспечения и автоматизированных информационных систем.

В 2005 году компания «ИВА-Гомель» присоединилась к научно-технологической ассоциации «Инфопарк». С ноября 2008 года в состав гомельского центра разработок было включено дополнительное предприятие ООО «ИВА-Гомель-Парк» (*IBA GomelPark*), призванное предоставлять услуги по проектированию, разработке и поставке ПО для корпоративных заказчиков из Беларуси, стран СНГ, Западной Европы и США. В мае 2009 года «ИВА-Гомель-Парк» стала резидентом Парка высоких технологий (ПВТ).

В 2012 году компания «ИВА-Гомель-Парк» получила награду Национального конкурса «Золотой Байт» в номинации «Лучший региональный центр разработки ПВТ».

«ИВА-Гомель-Парк» объединяет в себе более 350 высококвалифицированных специалистов, обладающих обширным опытом разработки программных решений на самых современных технологиях, платформ и программных продуктов для различных отраслей экономики и производства.

Среди основных направлений деятельности компании можно выделить следующие:

- проектирование, разработка и сопровождение приложений для платформы *Java Enterprise Edition, Microsoft SharePoint/.Net*;
- приложений для извлечения, преобразования, загрузки массивов данных с использованием *ETL*-технологий;
- приложений в среде «облачных» вычислений;
- приложений для работы с *BigData* и *SemanticWeb*;
- приложений для мобильных устройств на различных платформах (*Android, iOS, WindowsPhone/Mobile*);
- приложений с использованием концепции *SPA* и сервис-ориентированной архитектуры (*SOA*);
- приложений на языке *ABAP* для систем *SAP*;
- приложений на базе системы коллективной работы *IBMNotes/Domino*;
- продуктов Интернет и *e-business* (информационных порталов, *Web*-сайтов, систем ведения электронных платежей и др.);
- решений для обмена данными между корпоративными системами, включая унаследованные, *SAP* и *Web*-приложения (на базе *IBM WebsphereMessageBroker*);
- создание и развитие интегрированных систем управления предприятием на базе решений *SAP*;
- миграция программных продуктов между разнообразными платформами и средами;
- проектирование и администрирование баз данных (*DB2, Oracle, MS SQL Server*);
- разработка и внедрение *BI*-решений по отчетности и анализу данных;
- разработка облачных сервисов по управлению *НСИ (MDM, MDG)*;
- анализ качества данных в справочниках (*ISO 8000/22745*);

- построение систем управления НСИ (*SAP MDG, MDM/BPM/Webdynpro*);
- построение онтологий и средств структуризации данных (*RDF/OWL*);
- преобразование данных НСИ к формату *SemanticWeb* и организация доступа (*SPARQL*);
- банковские технологии.

IBA Group имеет партнерские статусы ведущих мировых ИТ-корпораций, таких как *IBM, SAP, Hewlett Packard Enterprise (HPE), OpenText, Oracle, Check Point, Glory Global Solutions, Philips, Diebold Nixdorf, PTC, Splunk Inc, Lenovo* и др.

1.2 Используемое оборудование

Рабочие кабинеты *IBA Gomel* оснащены современным оборудованием, а также всеми сопутствующими необходимыми аксессуарами. Типичное рабочее место инженера-программиста состоит из следующих частей:

- персональный компьютер на базе процессора *Intel Core (i5 или i7)*;
- монитор с диагональю 19-21 дюйм;
- клавиатура, мышь, наушники.

Кроме этого, офис оснащён проводной и беспроводной (*Wi-Fi*) сетью, объединяющей все компьютеры организации. Данные сети, также, предоставляют доступ к сети Интернет. В целом, можно отметить, что материальная база *IBA Gomel* находится на высоком уровне.

1.3 Охрана труда на предприятии

Охрана труда – система обеспечения безопасности жизни и здоровья, работающих в процессе трудовой деятельности, включающая правовые, социально-экономические, организационные, технические, санитарно-гигиенические, психофизиологические, лечебно-профилактические, реабилитационные, и иные мероприятия и средства.

Руководством *IBA Gomel* уделяется большое внимание улучшению эргономики рабочих мест, обеспечению гигиены и совершенствованию организации труда, регламентации режимов труда и отдыха.

Вся деятельность в области охраны труда в *IBA Gomel* регламентирована действующим законодательством Республики Беларусь, санитарными нормами и правилами, гигиеническими нормативами, предписаниями надзорных органов.

К комплексу мероприятий в области охраны труда, осуществляемых в *IBA Gomel* относятся:

- проведение периодических медицинских осмотров персонала;

- проведение производственного лабораторного контроля за условиями труда на рабочих местах;
- разработка инструкций по охране труда и ознакомление с ними персонала;
- обеспечение средствами индивидуальной защиты работников;
- изменение технологических процессов с целью совершенствования их безопасности для здоровья работников;
- модернизация рабочих мест и технологического оборудования;
- создание безопасных условий труда.

Важным фактором создания безопасных условий труда является оптимизация организации рабочих мест. При правильной организации рабочего места производительность труда инженера возрастает с 8 до 20 процентов.

Основным рабочим местом инженера-программиста является стол для выполнения машинописных работ. Основная поза при выполнении работы – вынужденная, сидячая.

Рабочее место для выполнения работ в положении сидя организуется в соответствии с ГОСТ 12.2.032-78 «Система стандартов безопасности труда. Рабочее место при выполнении работ сидя. Общие эргономические требования».

Требования к организации работы при использовании персонального компьютера и организационных средств определяются СанПиН 9-131 РБ 2000 «Гигиенические требования к видеодисплейным терминалам, электронно-вычислительным машинам и организации работы».

Согласно ГОСТ 12.2.032-78 конструкция рабочего места и взаимное расположение всех его элементов должно соответствовать антропометрическим, физическим и психологическим требованиям. Большое значение имеет также характер работы. В частности, при организации рабочего места программиста должны быть соблюдены следующие основные условия:

- оптимальное размещение оборудования, входящего в состав рабочего места;
- достаточное рабочее пространство, позволяющее осуществлять все необходимые движения и перемещения;
- уровень акустического шума не должен превышать допустимого значения.

Рабочая поза сидя вызывает минимальное утомление программиста. Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, расположено в зоне легкой досягаемости рабочего пространства.

Помещения для работы программиста должны иметь естественное и искусственное освещение.

Площадь на одно рабочее место с видео-дисплейным терминалом (ВДТ) и ПЭВМ для взрослых пользователей должна составлять не менее $6,0 \text{ м}^2$, а объем не менее $20,0 \text{ м}^3$.

Искусственное освещение в помещениях эксплуатации ВДТ и ПЭВМ должно осуществляться системой общего равномерного освещения. В административно-общественных помещениях, в случаях преимущественной работы с документами, допускается применение системы комбинированного освещения (к общему освещению дополнительно устанавливаются светильники местного освещения, предназначенные для освещения зоны расположения документов).

Освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300-500 лк. Местное освещение не должно создавать бликов на поверхности экрана и увеличивать освещенность экрана более 300 лк.

В качестве источников света при искусственном освещении необходимо использовать преимущественно люминесцентные лампы.

Конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей (размер ВДТ и ПЭВМ, клавиатуры и др.), характера выполняемой работы. При этом допускается использование рабочих столов различных конструкций, отвечающих современным требованиям эргономики.

Рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также расстоянию спинки от переднего края сиденья, при этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию.

Рабочий стол должен иметь пространство для ног высотой не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм.

Клавиатуру следует располагать на поверхности стола на расстоянии не менее чем 300 мм от края, обращенного к пользователю или на специальной, регулируемой по высоте рабочей поверхности, отделенной от основной столешницы.

Помимо требований к организации рабочего места СанПиН 9-131 РБ 2000 устанавливает требования к микроклимату рабочей зоны: влажности, температуре, скорости потока воздуха и пр.

Техника безопасности – это система организационных мероприятий и технических средств, предотвращающих воздействие на работающих опасных производственных факторов.

Для соблюдения техники безопасности в своей деятельности инженер-программист должен руководствоваться инструкцией по охране труда для программиста при выполнении работ с применением ПЭВМ и ВДТ.

1.4 Должностная инструкция инженера-программиста

Общие положения:

- инженер-программист относится к категории специалистов;
- инженер-программист назначается на должность и освобождается от нее приказом генерального директора по представлению технического директора / начальника структурного подразделения;
- инженер-программист подчиняется непосредственно техническому директору / начальнику структурного подразделения;
- на время отсутствия инженера-программиста его права и обязанности переходят к другому должностному лицу, о чем объявляется в приказе по организации;
- на должность инженера-программиста назначается лицо, отвечающее следующим требованиям: высшее профессиональное (техническое) образование, стаж работы от года.

Инженер-программист должен знать:

- руководящие и нормативные материалы, регламентирующие методы разработки алгоритмов и программ и использования вычислительной техники при обработке информации;
- основные принципы структурного программирования;
- виды программного обеспечения;
- технологию автоматической обработки информации и кодирования информации;
- формализованные языки программирования;
- порядок оформления технической документации.

Инженер-программист руководствуется в своей деятельности:

- законодательными актами РБ;
- уставом организации, Правилами внутреннего трудового распорядка, другими нормативными актами компании;
- приказами и распоряжениями руководства;
- настоящей должностной инструкцией.

Инженер-программист выполняет следующие должностные обязанности:

- на основе анализа математических моделей и алгоритмов решения экономических и других задач разрабатывает программы, обеспечивающие возможность выполнения алгоритма и соответственно поставленной задачи средствами вычислительной техники, проводит их тестирование и отладку;
- разрабатывает технологию решения задачи по всем этапам обработки информации;
- осуществляет выбор языка программирования для описания алгоритмов и структур данных;

- определяет информацию, подлежащую обработке средствами вычислительной техники, ее объемы, структуру, макеты и схемы ввода, обработки, хранения и вывода, методы ее контроля;
- выполняет работу по подготовке программ к отладке и проводит отладку;
- осуществляет запуск отлаженных программ и ввод исходных данных, определяемых условиями поставленных задач;
- проводит корректировку разработанной программы на основе анализа выходных данных;
- разрабатывает инструкции по работе с программами, оформляет необходимую техническую документацию;
- определяет возможность использования готовых программных продуктов;
- осуществляет сопровождение внедрения программ и программных средств;
- разрабатывает и внедряет системы автоматической проверки правильности программ, типовые и стандартные программные средства, составляет технологию обработки информации;
- выполняет работу по унификации и типизации вычислительных процессов.

Права инженера-программиста:

- знакомиться с проектами решений руководства предприятия, касающихся его деятельности;
- вносить на рассмотрение руководства предложения по совершенствованию работы, связанной с предусмотренными настоящей инструкцией обязанностями;
- сообщать своему непосредственному руководителю о всех выявленных в процессе осуществления должностных обязанностей недостатках в деятельности предприятия (его структурных подразделениях) и вносить предложения по их устранению.

Ответственность инженера-программиста:

- за невыполнение и/или несвоевременное, халатное выполнение своих должностных обязанностей;
- за несоблюдение действующих инструкций, приказов и распоряжений по сохранению коммерческой тайны и конфиденциальной информации.

На время отсутствия инженера-программиста (отпуск, болезнь, командировка, пр.) его обязанности исполняет лицо, назначенное в установленном порядке, которое несет ответственность за качественное исполнение возложенных на него обязанностей.

2 ОПИСАНИЕ ЭТАПОВ ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

2.1 Постановка задачи

На период прохождения производственной практики на предприятии «ИВА-Гомель-Парк» была поставлена задача – разработать приложение, которое позволило бы сокращать ссылки большой длины для систем коммуникации с ограниченным числом символов в сообщении. Также требуется предоставить возможность добавлять, просматривать ссылки и группы ссылок и подсчитывать количество переходов по каждой из них. Для регистрации пользователей в приложении необходимо реализовать механизм аутентификации. Таким образом, приложение должно содержать следующие страницы:

- страница регистрации пользователя;
- страница аутентификации пользователя;
- страница тематических групп ссылок;
- страница конкретной группы ссылок;
- страница полного списка имеющихся ссылок;
- страница добавления группы ссылок;
- страница добавления ссылки.

2.2 Архитектура приложения

Проблема масштабируемости, повторного использования кода, модулей, автономности модулей поднимается при решении больших задач и проектировании сложных систем и решений [1]. Одним из распространенных подходов решения большинства из этих задач является разделение на слои. Есть один независимый слой, который находится в центре архитектуры. От этого слоя зависит второй, от второго – третий и т. д., общая схема многоуровневой архитектуры изображена на рисунке 2.1. Количество слоев может отличаться, но в центре всегда находятся классы моделей, которые используются в приложении и объекты которых хранятся в базе данных.

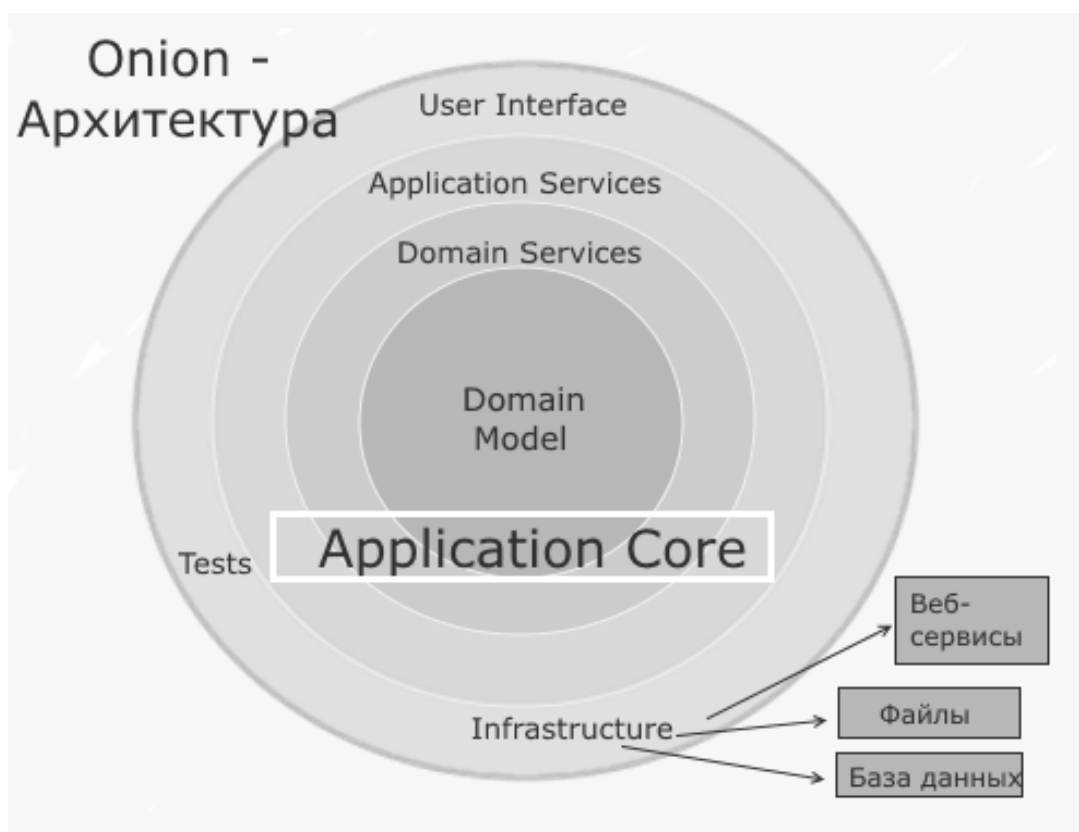


Рисунок 2.1 – Общая схема многоуровневой архитектуры

Слои располагаются друг над другом и содержат логически схожие компоненты с примерно одинаковым уровнем абстракции. Суть в том, что нижележащие слои максимально свободны от вышележащих. Компоненты из одного слоя могут взаимодействовать только с компонентами своего слоя, а также с компонентами более низких слоев.

Разделение на слои делится на гибкое и строгое. Гибкое разделение характеризуется тем, что определенный слой может взаимодействовать со всеми нижележащими слоями. Строгое разделение же подразумевает то, что определенный слой может взаимодействовать только с ближайшим нижележащим слоем. Любое гибкое разделение можно привести к строгому, пробросив взаимодействие через промежуточные слои.

Для решения поставленной задачи было решено использовать классическую трехуровневую архитектуру (рисунок 2.2), подразумевающую разделение приложения на три слоя: слой доступа к данным (*Data Access Layer*), слой бизнес-логики (*Business Logic Layer*), слой представления (*Presentation Layer*).

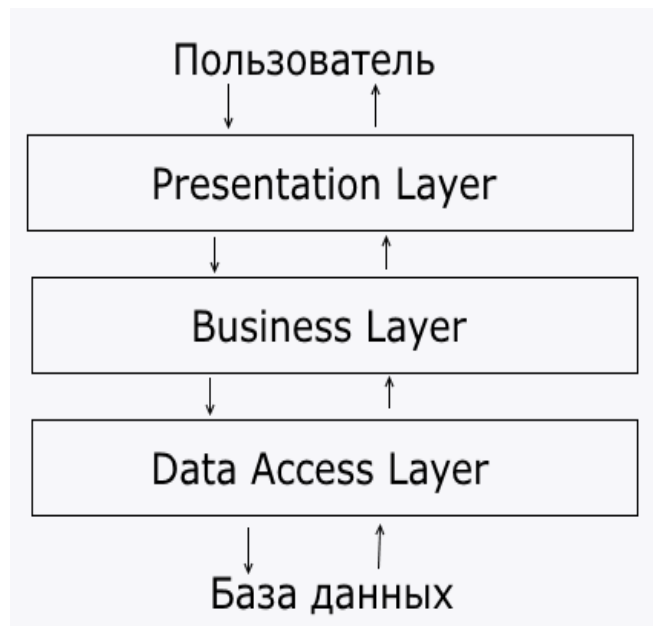


Рисунок 2.2 – Схема классической трехуровневой архитектуры

Слой представления (*Presentation Layer*) – слой, с которым пользователь непосредственно взаимодействует [2]. Этот слой включает в себя механизмы получения команд (ввода) от пользователя, компоненты пользовательского интерфейса (*UI – User Interface*), модели представлений, контроллеры, объекты контекста запроса.

Слой бизнес-логики (*Business Logic Layer*) – слой, отвечающий за обработку данных, полученных от слоя представления и реализующий логику приложения. Он взаимодействует с хранилищем данных через слой доступа к данным и передает слою представления результат обработки.

Слой доступа к данным (*Data Access Layer*) – слой, в котором хранятся модели, описываются сущности, размещаются специфичные классы для работы с разными технологиями доступа к данным (например, класс контекста данных *Entity Framework*).

Подробнее о каждом из слоев.

2.3 Слой доступа к данным (*Data Access Layer*)

Этот слой содержит все модели данных, хранящихся в базе данных (БД), а также классы, через которые идет взаимодействие с БД. В основе концепции лежит принцип управляемого специализируемого хранения данных, который вводит унифицированный и контролируемый способ доступа к различным данным для приложения [3]. Схема концепции изображена на рисунке 2.3.

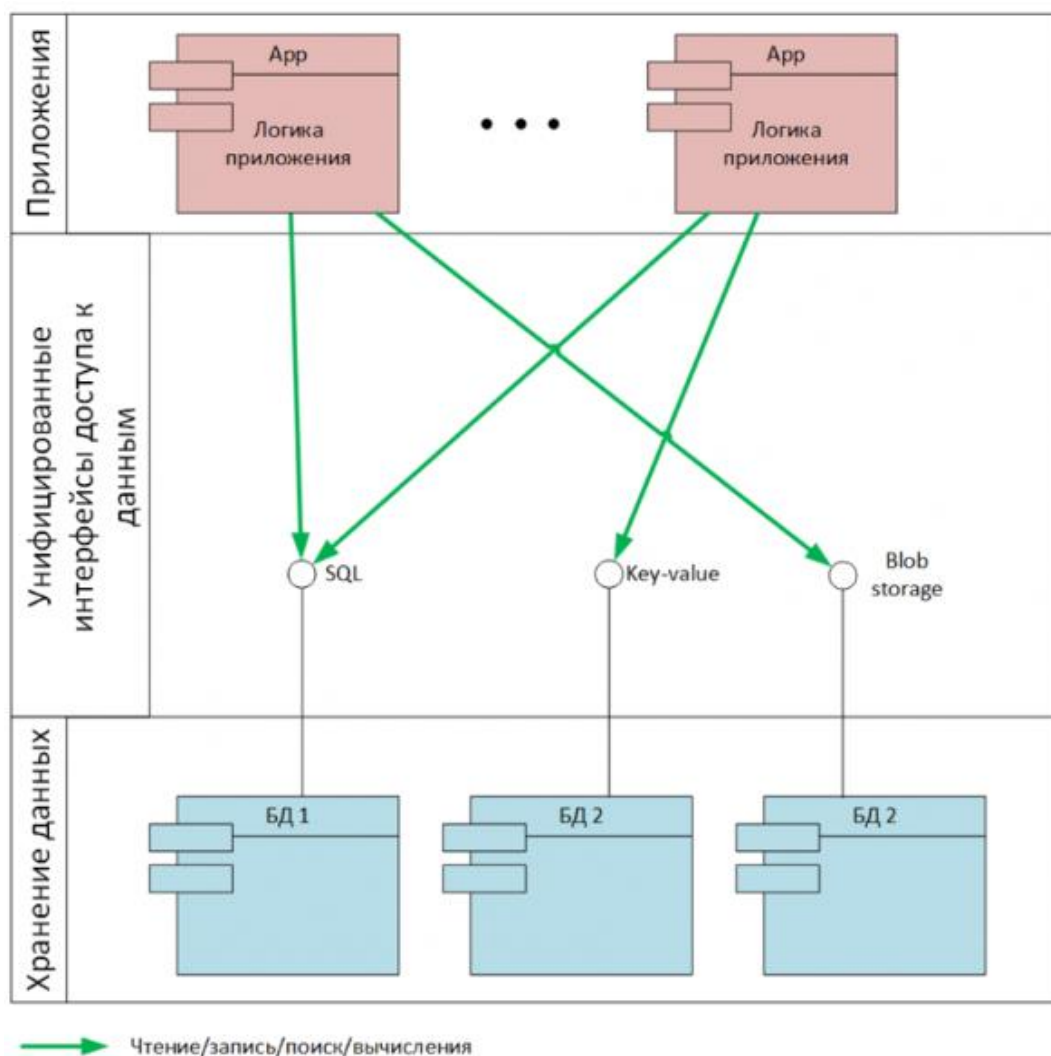


Рисунок 2.3 – Принцип управляемого специализированного хранения данных

В этом слое определяется фиксированный набор классов данных, для которых в инфраструктуре предоставляются некоторые средства хранения. Каждый класс данных при этом предполагает специализированный интерфейс доступа на логическом уровне, оптимальный для данного класса. Например, для класса данных «*key-value*» интерфейс доступа должен предоставлять операции чтения и записи данных по ключу.

Основные паттерны, которые используются в *DAL*-слое:

- «*Repository*»;
- «*Unit of Work*»;
- «*Dispose*».

Для хранения информации о ссылках и группах используются две модели, представленные классами «*Group*», «*Link*». Также специально для репозитория типа *Entity Framework* используется класс контекста данных *Entity Framework*.

Для увеличения гибкости подключения к хранилищам данных используются репозитории, реализующие один интерфейс репозитория *IRepository*. Поскольку будет использоваться несколько репозитория для каждой сущности, то для упрощения работы с сущностями внутри репозитория применён паттерн «*Unit of Work*». Этот паттерн позволяет упростить работу с различными сущностями в пределах одного репозитория и дает уверенность, что в конкретный момент используются репозитории одного конкретного типа. Он предоставляет доступ к репозиториям через отдельные свойства и определяет общий контекст для всех репозитория. Для хранения реализаций данных интерфейсов в слое *DAL* определены классы репозитория «*GroupRepository*», «*LinkRepository*» и класс «*EfUnitOfWork*». «*EF*» в имени репозитория означает репозиторий *Entity Framework*. UML-диаграмма классов репозитория «*LinkRepository*» и «*GroupRepository*» изображена на рисунке 2.4.

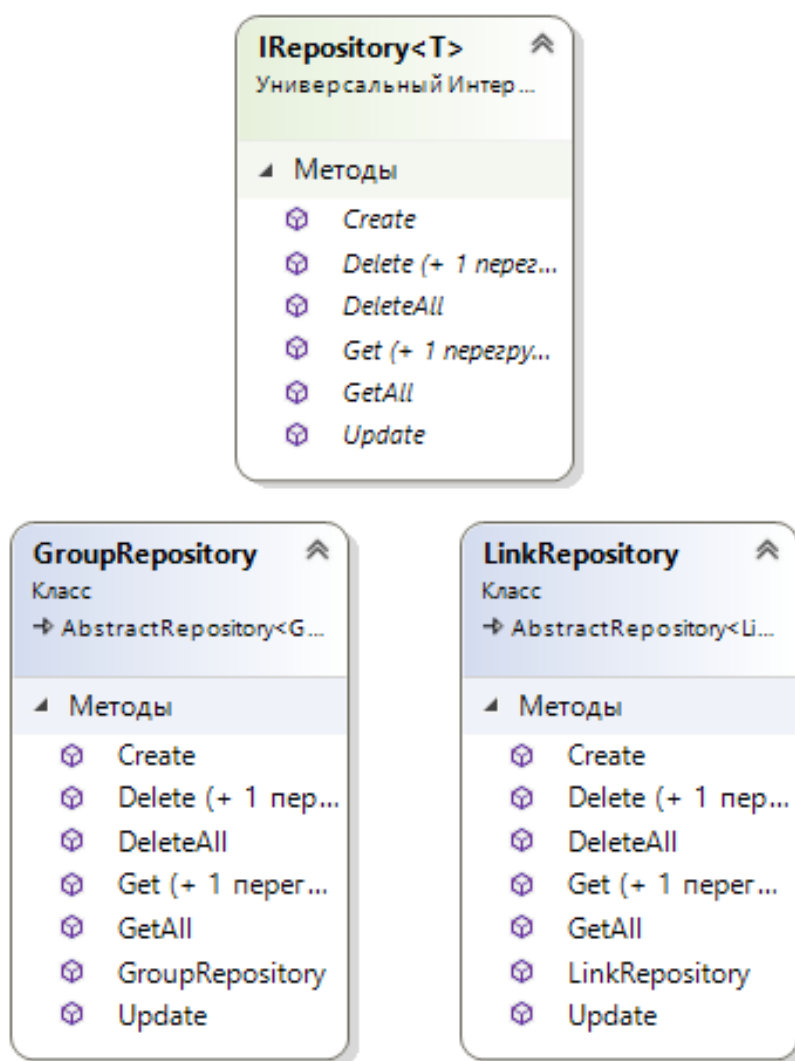


Рисунок 2.4 – UML-диаграмма классов репозитория «*IRepository*», «*GroupRepository*» и «*LinkRepository*»

Класс «*EfUnitOfWork*» в конструкторе принимает строку – название подключения, которая потом будет передаваться в конструктор контекста данных. Собственно, через «*EfUnitOfWork*» и будет происходить взаимодействие с репозиторием типа *EF*. В этом классе также содержатся дополнительные методы *Save* и *Dispose*. Метод *Save* служит для сохранения изменений в репозитории, а метод *Dispose* реализует паттерн «*Dispose*». Идея паттерна «*Dispose*» состоит в следующем: вся логика освобождения ресурсов помещается в отдельный метод, который будет вызываться из метода *Dispose*, и из финализатора, при этом добавляя флаг, который говорит о том, откуда был вызван этот метод. На рисунке 2.5 изображена *UML*-диаграмма классов-реализаций интерфейса *IUnitOfWork*.

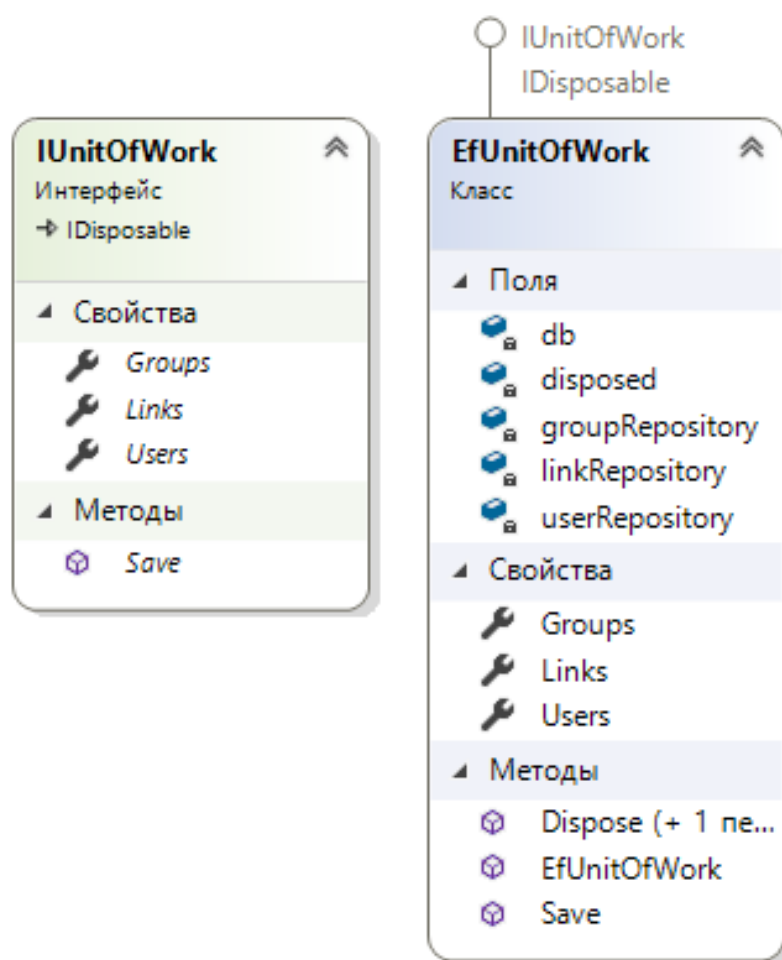


Рисунок 2.5 – *UML*-диаграмма классов-реализации интерфейса *IUnitOfWork*

В итоге приложение не получает прямого доступа к хранилищам данных, а работает через специализированный модуль доступа к данным – *Data Access Layer*. Итоговая структура слоя доступа к данным представлена на рисунке 2.6.

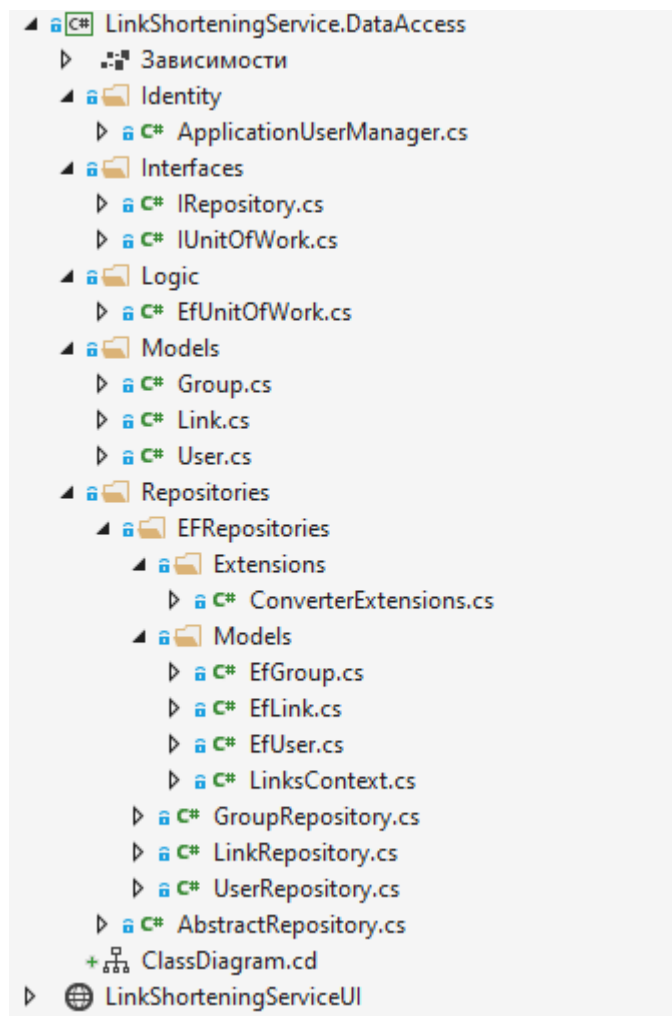


Рисунок 2.6 – Структура *DAL*-слоя

2.4 Слой бизнес-логики (*Business Logic Layer*)

Слой бизнес-логики следует на этапе разработки сразу за слоем доступа к данным. Этот слой инкапсулирует всю логику, все необходимые вычисления, получает объекты из уровня доступа к данным и передает их на уровень представления, либо, наоборот, получает данные с уровня представления и передает их на уровень данных [4]. В нем используются классы из слоя доступа к данным.

Для того, чтобы начать использовать *BLL*-слой в проекте, необходимо добавить ссылку на *DAL*-слой, так как тут будут использоваться классы, определенные в *DAL*-слое. Слой представления не может получать данные из хранилища данных напрямую. Поэтому *BLL* будет выступать посредником между этими двумя слоями. Но слой *BLL* не может также передавать напрямую в контроллеры объекты своих классов, т.к. слой представления не должен иметь доступ к функциональности слоя *DAL*. Для этого в слое *BLL* используются классы «*GroupDTO*» и «*LinkDTO*». Через эти классы передаются объекты

между уровнями. Хотя данные классы во многом похожи по определению на классы «*Group*» и «*Link*», это необязательное условие. Классы *DTO* должны содержать только те данные, которые необходимо передавать на уровень представления или, наоборот, получать с этого уровня. То есть это то, что называется *Data Transfer Object* – специальная модель для передачи данных.

Для упрощения сопоставления классов моделей в проекте используется библиотека *AutoMapper*. *AutoMapper* – шаблон, позволяющий проецировать одну модель на другую. Он проверяет, есть ли соответствующие поля в указанных типах, соответствие проводится как по имени свойства, так и по его типу. Как это работает: вначале создается объект репозитория *IRepository*. Затем в каждом методе контроллера производится конфигурация *Automapper* и сопоставление объектов.

Для настройки конфигурации используется метод *Mapper.Initialize()*. В этот метод передается параметр конфигурации, который с помощью различных методов, в частности, метода *CreateMap*, настраивает сопоставление классов. При конфигурации сопоставляются одноименные свойства классов сущности и модели отображения. При сопоставлении объектов используется метод *Mapper.Map*, причем так же можно создавать коллекции объектов.

Большую роль в приложении играет валидация данных [5]. По большей части за валидацию отвечает именно *BLL*. В контроллере не возникает проблем провалидировать модель через объект *ModelState* и при необходимости возвратить в представление сообщения об ошибках. Но на уровне *BLL* *ModelState* недоступен. Однако валидация, с передачей ошибок в уровень представления, всё же возможна.

Для этого в *BLL* определён класс *ValidationException*. Этот класс наследуется от базового класса исключений *Exception* и определяет свойство *Property*. Это свойство позволяет сохранить название свойства модели, которое некорректно и не проходит валидацию. И также передавая в конструктор базового класса параметр «*message*», определяется сообщение, которое будет выводиться для некорректного свойства в *Property*.

Взаимодействие между остальными уровнями будет происходить через специальный сервис. Для этого сервиса определён интерфейс *ILinkService*. В качестве реализации этого интерфейса определён класс сервиса *LinkService*. *LinkService* в конструкторе принимает объект *IUnitOfWork*, через который идет взаимодействие с уровнем *DAL*.

Основной функционал программы построен таким образом, что все модули *BLL*-слоя напрямую не взаимодействуют с хранилищем данных. Тем самым слои *BLL* и *DAL* наиболее отстранены друг от друга, что приводит к минимальным изменениям внутри компонентов при взаимодействии этих слоев. Итоговая структура слоя бизнес-логики представлена на рисунке 2.7.

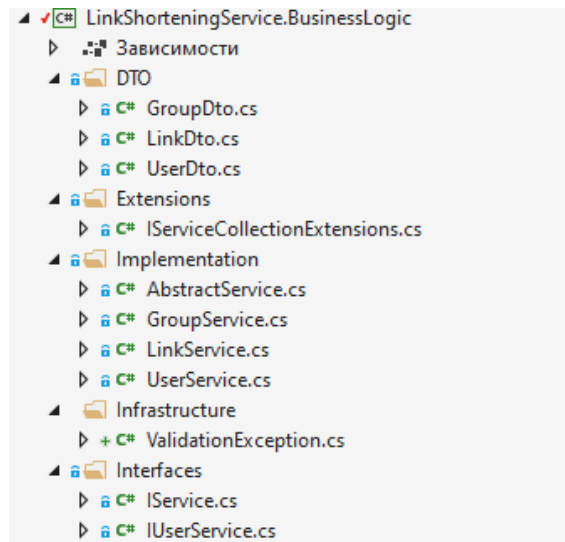


Рисунок 2.7 – Структура *BLL*-слоя

2.5 Слой представления (*Presentation Layer*)

Слой представления, или *Presentation Layer*, – это единственный слой в приложении с трехуровневой архитектурой, непосредственно связанный с пользователем. Он используется для получения данных от пользователя и передачи их *BLL*-слою для дальнейшей обработки, и представляет объект в надлежащей форме, которая понятна пользователю. Слой представления реализован как *web*-приложение *ASP.NET API* и приложение-клиент, написанное при использовании *JavaScript* фреймворка *Angular*.

2.5.1 Аббревиатура *API* расшифровывается как «*Application Programming Interface*» (интерфейс программирования приложений, программный интерфейс приложения) [6]. Большинство крупных компаний на определённом этапе разрабатывают *API* для клиентов или для внутреннего использования. Чтобы понять, как и каким образом *API* применяется в разработке и бизнесе, сначала нужно разобраться, как устроена «всемирная паутина».

«*WWW*» можно представить как огромную сеть связанных серверов, на которых и хранится каждая страница. Обычный ноутбук можно превратить в сервер, способный обслуживать целый сайт в сети, а локальные серверы разработчики используют для создания сайтов перед тем, как открыть их для широкого круга пользователей.

При введении в адресную строку браузера *www.facebook.com* на удалённый сервер *Facebook* отправляется соответствующий запрос. Как только браузер получает ответ, то интерпретирует код и отображает страницу.

Каждый раз, когда пользователь посещает какую-либо страницу в сети, он взаимодействует с *API* удалённого сервера. *API* – это составляющая часть сервера, которая получает запросы и отправляет ответы.

Слово «*application*» (прикладной, приложение) может применяться в разных значениях [7]. В контексте *API* оно подразумевает:

- фрагмент программного обеспечения с определённой функцией;
- сервер целиком, приложение целиком или же просто отдельную часть приложения.

Любой фрагмент ПО, который можно чётко выделить из окружения, может заменять букву «А» в англоязычной аббревиатуре, и тоже может иметь некоторого рода *API*. Например, при внедрении в код разработчиком сторонней библиотеки, она становится частью всего приложения. Будучи самостоятельным фрагментом ПО, библиотека будет иметь некий *API*, который позволит ей взаимодействовать с остальным кодом приложения.

В объектно-ориентированном проектировании код представлен в виде совокупности объектов. В приложении таких объектов, взаимодействующих между собой, могут быть сотни. У каждого из них есть свой *API* набор публичных свойств и методов для взаимодействия с другими объектами в приложении. Объекты могут также иметь частную, внутреннюю логику, которая скрыта от окружения и не является *API*.

Для решения поставленной задачи был разработан *API*, содержащий два контроллера:

- *GroupsController*;
- *LinksController*.

GroupsController отвечает за добавление и удаление групп ссылок. *LinksController* обслуживает запросы по добавлению и удалению ссылок, а также сохраняет укороченный их вариант и количество переходов по каждой ссылке.

Структура получившегося приложения изображена на рисунке 2.8.

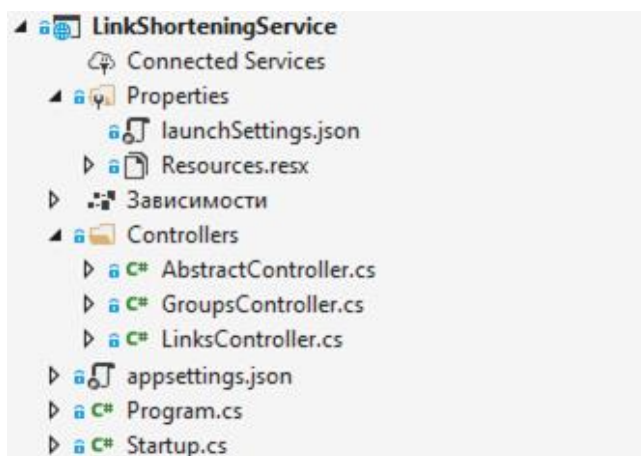


Рисунок 2.8 – Структура *API* приложения

Таким образом был разработан *API*, который полностью реализует функционал, необходимый для решения поставленной задачи.

2.5.2 Angular представляет фреймворк от компании *Google* для создания клиентских приложений. Прежде всего, он нацелен на разработку *Single Page Application*, то есть одностраничных приложений. В этом плане *Angular* является наследником другого фреймворка *AngularJS*. В то же время *Angular* это не новая версия *AngularJS*, а принципиально новый фреймворк.

Angular предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизацию и т.д.

Преимущества *Angular*:

- *Angular* представляет не только инструменты, но и шаблоны дизайна для создания обслуживаемого проекта, при правильном создании *Angular* приложения не будет путаницы классов и методов, которые сложно править и еще сложнее тестировать, код удобно структурирован, можно быстро понять, что к чему;
- *Angular* построен на *TypeScript*, который, в свою очередь, полагается на *ES6*, то есть нет необходимости учить полностью новый язык, и в тоже время предоставляются такие функции как статическая типизация, интерфейсы, классы, пространства имен, декораторы и другие;
- в *Angular* уже есть много инструментов для создания приложения, благодаря директивам;
- *HTML* элементы могут вести себя динамически;
- *FormControl* позволяет применить различные правила валидации;
- есть возможность легко посылать асинхронные *HTTP* запросы различных типов;
- встроенный механизм маршрутизации;
- компоненты разъединены: *Angular* старался убрать жесткую связь между различными компонентами приложения, внедрение проходит подобно *NodeJS*, что позволяет легко заменять компоненты.

Получившееся приложение-клиент на *Angular* имеет семь страниц и реализует весь указанный в задании функционал.

2.6 Верификация программного обеспечения

На рисунке 2.9 приведена диаграмма вариантов использования, которая описывает функциональное назначение разработанного программного средства, взаимоотношение и зависимости между группами вариантов использования и пользователем.

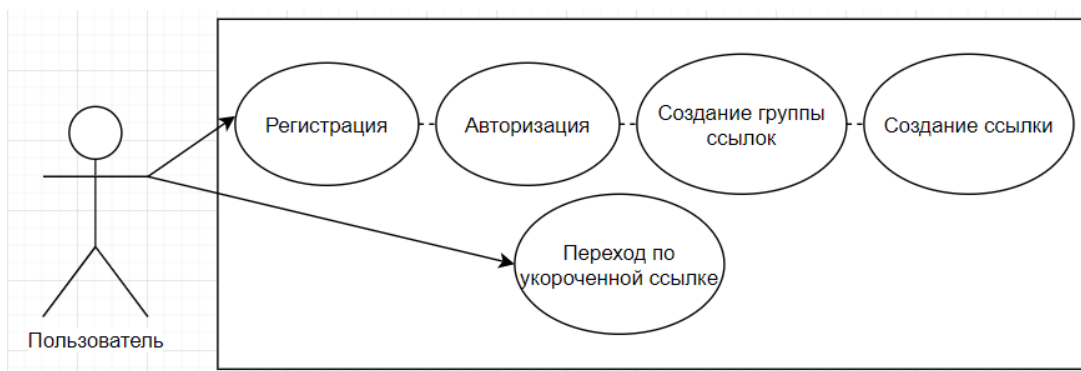


Рисунок 2.9 – Диаграмма вариантов использования приложения

Работа с разработанным программным комплексом состоит из трёх этапов:

- регистрация и авторизация пользователя в сервисе сокращения ссылок для использования реализованного функционала;
- создание групп ссылок и добавление в них требуемых сокращения ссылок на необходимый интернет источник;
- распространение сгенерированной сокращённой ссылки другим пользователям.

На всех этапах работы с программой предусмотрены проверки на корректность, введённых пользователем, данных.

Сразу после запуска программы перед пользователем появляется страница регистрации в веб-приложении (рисунок 2.10). Для корректного завершения регистрации необходимо правильно заполнить обязательные поля формы, иначе авторизоваться в системе будет невозможно.

Registration

Name

Email

Password

Please repeat your password

SIGN UP

Рисунок 2.11 – Регистрационная форма

Если заполнить регистрационную форму некорректными данными или оставить незаполненным обязательное поле ввода, появится блок с описанием ошибки (рисунок 2.12).

The image shows a registration form titled "Registration". It contains four input fields: "Name", "Email", "Password", and "Please repeat your password". The "Email" field contains the text "incorrect.mail" and is highlighted with a red border and a red exclamation mark icon, indicating an error. Below the input fields is a yellow "SIGN UP" button. At the bottom of the form, there is a pink error message box that says "The registration data is incorrect!".

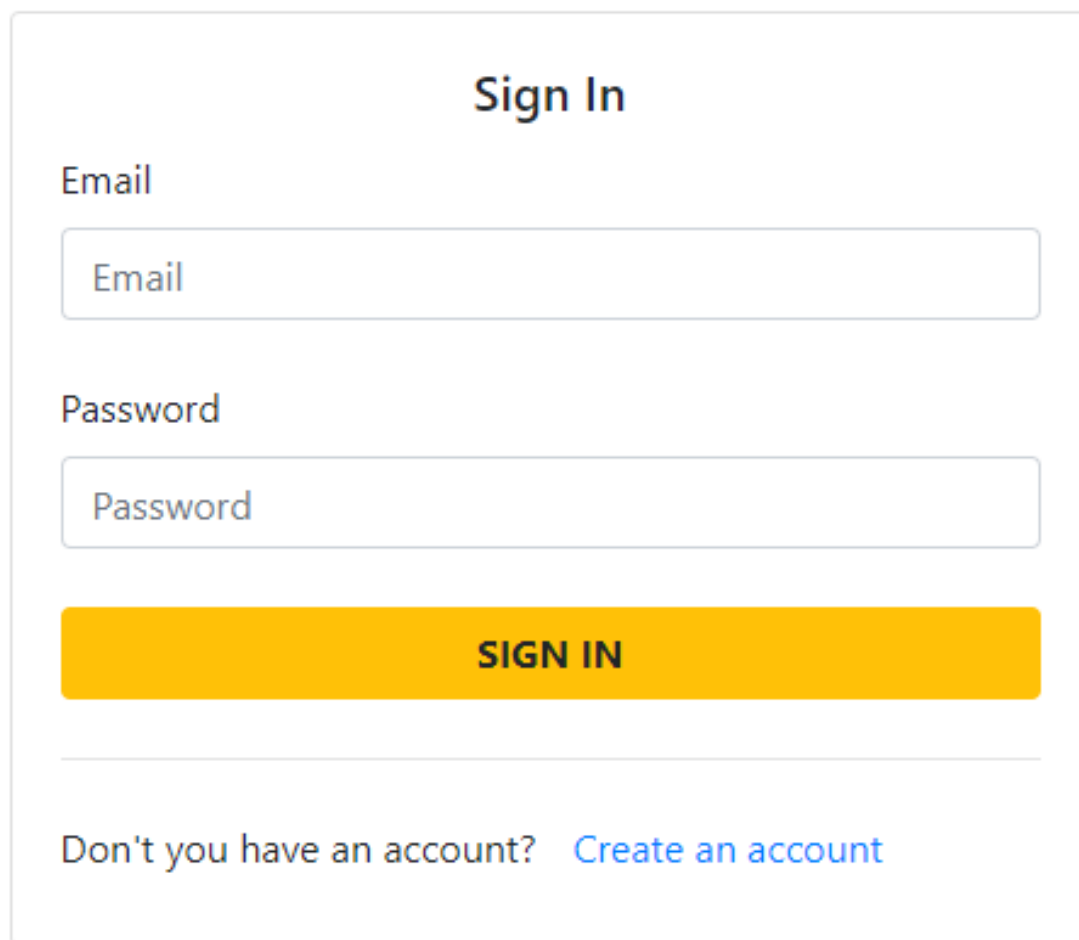
Рисунок 2.12 – Валидация введенных данных

Для успешного завершения регистрации требуется заполнить следующие поля:

- *Name* – имя пользователя;
- *Email* – электронная почта пользователя (необходим для авторизации);
- *Password* – секретный пароль (необходим для авторизации);
- *Repeated password* – повторный секретный пароль. Требуется для того, чтобы сократить шанс ввода неудачной комбинации символов, которую не сможет повторить пользователь при авторизации.

Для успешного входа в систему пользователю следует ввести данные, которые ему приходилось вводить в момент регистрации:

- *Email* – электронная почта пользователя;
- *Password* – секретный пароль пользователя.



The image shows a 'Sign In' form. At the top, the text 'Sign In' is centered in a dark blue font. Below it, there are two input fields. The first is labeled 'Email' and contains the placeholder text 'Email'. The second is labeled 'Password' and contains the placeholder text 'Password'. Below these fields is a large yellow button with the text 'SIGN IN' in black, uppercase letters. At the bottom of the form, there is a horizontal line, and below it, the text 'Don't you have an account?' followed by a blue link 'Create an account'.

Рисунок 2.13 – Авторизационная форма

После корректного заполнения авторизационной формы, появляется окно главной страницы приложения. Оно демонстрирует таблицу тематических групп ссылок и представлено на рисунке 2.14. На ней пользователю предоставляется возможность совершить следующие действия:

- просмотреть все существующие тематические группы ссылок;
- просмотреть конкретную тематическую группу ссылок;
- создать новую тематическую группу ссылок.

В таблице существующих тематических групп ссылок содержатся следующие колонки:

- *#* – номер тематической группы по счёту;
- *Name* – просмотреть таблицу всех существующих ссылок;
- *Number of transitions* – количество переходов по всем ссылкам в группе;

– *Links* – количество ссылок в тематической группе. Строки данной колонки являются ссылками на компонент с информацией о конкретной группе.

Link Shortening Service

Groups

Links

All Groups

Count of links: 121

Create a group

#	Name	Number of transitions	Links
1	Images	18	3 links
2	Music	90	15 links
3	Movies	5	20 links
4	Games	13	1 links
5	Food	66	16 links
6	Girls	1	1 links
7	News	20	4 links
8	Animals	9	2 links
9	Science	12	26 links
10	Sport	14	33 links

© Bootstrap

Рисунок 2.14 – Страница групп ссылок

Перейдя по ссылке из последней колонки таблицы тематических групп можно перейти в конкретную группу ссылок, где будет представлена таблица ссылок конкретной тематической группы со следующими столбцами:

- *#* – номер ссылки по счёту;
- *Name* – наименование ссылки (задаётся самим пользователем);
- *Number of transitions* – количество переходов по ссылке;
- *Url* – ссылка на ресурс пользователя (задаётся самим пользователем);
- *Short url* – сокращённая, разработанным сервисом, ссылка на пользовательский ресурс.

Все ссылки таблицы представленной на рисунке 2.15 являются активными.

Пользователи данного сервиса по сокращению ссылок могут запросто переходить на нужный ресурс по укороченной ссылке.

Link Shortening Service Groups Links				
Images Links <small>Count of links: 4</small>				Create a link
#	Name	Number of transitions	Url	Short url
1	Doggy Icon	0	http://doggy.by/media/categories/1/dog_1.jpg	http://localhost:4200/0l3k6j3t
2	Kitty Icon	0	http://doggy.by/media/categories/2/cat.jpg	http://localhost:4200/h90t9k5u
3	Bambi Icon	0	https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/gettyimages-168504892-1568303467.png?crop=0.754xw:0.911xh;0.191xw,0.0454xh&resize=640:*	http://localhost:4200/jj893opp
4	Bunny Icon	0	https://images-na.ssl-images-amazon.com/images/I/51G4M8IU9FL_AC_SY450.jpg	http://localhost:4200/n98il1ff

© Bootstrap

Рисунок 2.15 – Ссылки конкретной тематической группы

Пользователю также предоставляется возможность создавать необходимые тематические группы ссылок самостоятельно. Для этого имеется возможность воспользоваться кнопкой «*Create a group*» и перейти на форму создания новой группы для её добавления в общую таблицу групп. Для создания новой тематической группы ссылок требуется заполнить обязательное поле «*Name*», которое кратко описывает ссылки, относящиеся к данному разделу. Форма создания новой тематической группы ссылок представлено на рисунке 2.16:

Link Shortening Service

Group creation

Name

Name

Create

© Bootstrap

Рисунок 2.16 – Форма создания новой группы ссылок

На главной странице веб-приложения также существует кнопка перехода на страничку всех созданных в сервисе ссылок. Данная страничка представлена на рисунке 2.17. На ней пользователю выпадает возможность совершить следующие действия:

- просмотреть все существующие полные и сокращённые ссылки;
- перейти по нужному адресу;
- создать новую сокращённую ссылку.

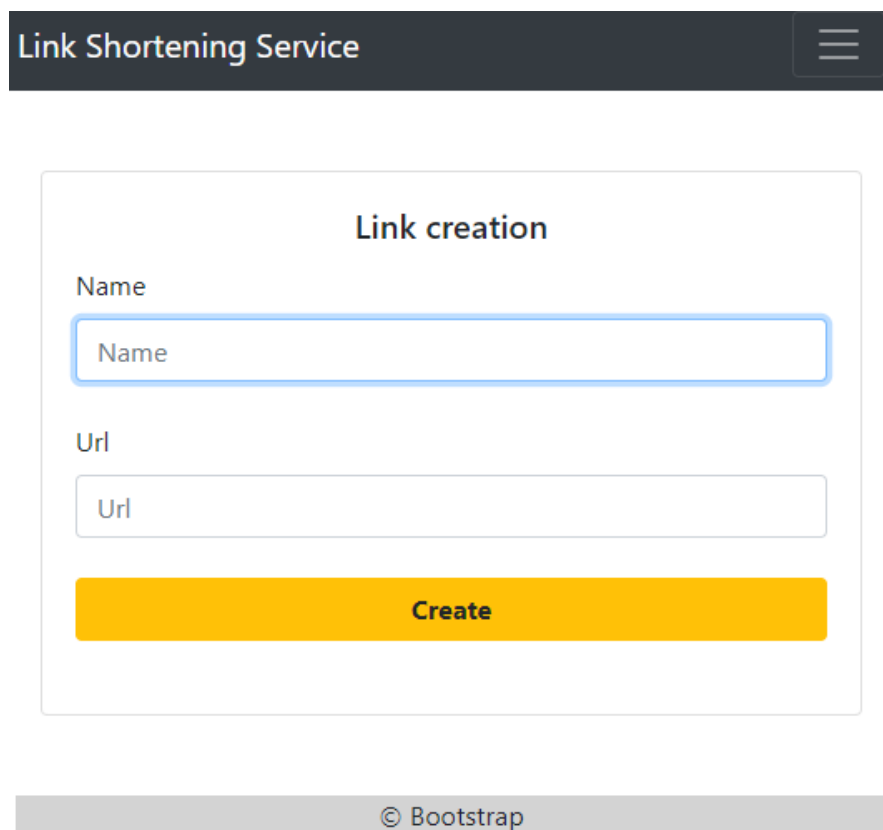
В таблице ссылок содержатся следующие колонки:

- # – номер ссылки по счёту;
- *Name* – наименование ссылки;
- *Number of transitions* – количество переходов по конкретной ссылке;
- *Url* – полный адрес сокращаемой ссылки;
- *Short url* – сокращённая ссылка.

Link Shortening Service Groups Links				
All Links Count of links: 10				Create a link
#	Name	Number of transitions	Url	Short url
1	VK	0	http://vk.com	http://localhost:4200/qwr12e34
2	OK	0	http://ok.ru	http://localhost:4200/qw3er245
3	FB	0	http://facebook.com	http://localhost:4200/qwe5r346
4	TW	0	http://twitter.com	http://localhost:4200/1fn98ilf
5	GOOGLE	0	http://www.google.com	http://localhost:4200/6j3t0l3k
6	YOUTUBE	0	http://www.youtube.com	http://localhost:4200/9k5uh90t
7	Doggy Icon	0	http://doggy.by/media/categories/1/dog_1.jpg	http://localhost:4200/0l3k6j3t
8	Kitty Icon	0	http://doggy.by/media/categories/2/cat.jpg	http://localhost:4200/h90t9k5u
9	Bambi Icon	0	https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/gettyimages-168504892-1568303467.png?crop=0.754xw:0.911xh;0.191xw,0.0454xh&resize=640:*	http://localhost:4200/jj893opp
10	Bunny Icon	0	https://images-na.ssl-images-amazon.com/images/I/51G4M8IU9FL_AC_SY450_.jpg	http://localhost:4200/n98il1ff
© Bootstrap				

Рисунок 2.17 – Страница групп ссылок

Для уменьшения длины нужной ссылки пользователю необходимо создать или выбрать существующую тематическую группу ссылок из главной страницы и воспользоваться кнопкой «*Create a link*», с помощью которой можно добавить укороченную ссылку прямо в саму группу. Форма добавления новой ссылки представлена на рисунке 2.18:



The image shows a web interface for a 'Link Shortening Service'. At the top, there is a dark header bar with the text 'Link Shortening Service' on the left and a hamburger menu icon on the right. Below the header is a white box titled 'Link creation'. Inside this box, there are two input fields: the first is labeled 'Name' and contains the placeholder text 'Name'; the second is labeled 'Url' and contains the placeholder text 'Url'. Below these fields is a large yellow button with the text 'Create'. At the bottom of the page, there is a grey footer bar with the text '© Bootstrap'.

Рисунок 2.18 – Форма создания новой укороченной ссылки

Чтобы создать укороченную ссылку потребуется заполнить такие обязательные поля, как:

- *Name* – наименование ссылки;
- *Url* – полная ссылка к пользовательскому ресурсу.

После заполнения данных вся необходимая информация о ссылке, вместе с её укороченным вариантом, появится в таблице выбранной тематической группы и в полном списке всех созданных ссылок.

По, продемонстрированным выше, рисункам можно сделать вывод, что сервис по сокращению ссылок имеет дружелюбный графический интерфейс, что, в свою очередь, позволяет легко и быстро пользоваться необходимым функционалом.

ЗАКЛЮЧЕНИЕ

Разработанное программное обеспечение позволяет оптимизировать механизм коммуникации, путём замены ссылок большой длины на более мелкие. На этапах разработки была спроектирована архитектура приложения, реализован графический интерфейс и способ аутентификации пользователей, а также настроена маршрутизация при переходе по отдельным страничкам.

В ходе технологической практики были решены следующие задачи:

- приобретены знания о подходах разработки приложений и закреплены на практике;
- получен опыт работы на предприятии;
- получены навыки работы с командой;
- изучены и проанализированы предметная область, технологии и методы, необходимые для решения поставленных задач;
- разработаны алгоритмы решения поставленной задачи;
- на основе разработанных алгоритмов создано и протестировано программное обеспечение, решающее проблему передачи ссылок большой длины в сообщениях с ограниченным набором символов.

Список использованных источников

1. Шаблоны корпоративных приложений / Под ред. М. Фаулер. – Киев: Вильямс, 2016. – 38 с.
2. Комлев, Н. Ю. Объектно Ориентированное Программирование. Хорошая книга для Хороших Людей / Н. Ю. Комлев. – Москва: Дипак, 2014. – 23 с.
3. Фримен, Э. Паттерны проектирования / Э. Фримен, Э. Фримен. – СПб.: Питер, 2011. – 176 с.
4. Фримен, Э. Паттерны проектирования / Э. Фримен, Э. Фримен. – СПб.: Питер, 2011. – 153 с.
5. Фримен, Э. Паттерны проектирования / Э. Фримен, Э. Фримен. – СПб.: Питер, 2011. – 200 с.
6. Маклафлин, Б. Объектно-ориентированный анализ и проектирование / Б. Маклафлин, Г. Поллайс, Д. Уэст. – СПб.: Питер, 2013. – 231 с.
7. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Д. Рихтер. – СПб.Ж Питер, 2017. – 17 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Текст программы

Класс *GroupsController*:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using LinkShorteningService.BusinessLogic.DTO;
using LinkShorteningService.BusinessLogic.Interfaces;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace LinkShorteningService.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class GroupsController : AbstractController<GroupDto, int>
    {
        public GroupsController(IService<GroupDto, int> service) : base(service)
        {
        }

        [HttpPost]
        [Route("CreateGroup")]
        public IActionResult CreateGroup([FromBody] GroupDto group)
        {
            service.Create(group);

            return Ok();
        }

        [HttpGet]
        [Route("GetGroup")]
        public IActionResult GetGroup(int id)
        {
            var group = service.Get(id);
```

```

        return Ok(group);
    }

    [HttpGet]
    [Route("GetGroups")]
    public IActionResult GetGroups()
    {
        var groups = service.GetAll();

        return Ok(groups);
    }

    [HttpPost]
    [Route("UpdateGroup")]
    public IActionResult UpdateGroup([FromBody] GroupDto group)
    {
        service.Update(group);

        return Ok();
    }

    [HttpGet]
    [Route("DeleteGroup")]
    public IActionResult DeleteGroup(int id)
    {
        service.Delete(id);

        return Ok();
    }
}
}

```

Класс *LinksController*:

```

using LinkShorteningService.BusinessLogic.DTO;
using LinkShorteningService.BusinessLogic.Interfaces;
using Microsoft.AspNetCore.Mvc;

namespace LinkShorteningService.Controllers
{

```

```

[Route("api/[controller]")]
[ApiController]
public class LinksController : AbstractController<LinkDto, string>
{
    public LinksController(IService<LinkDto, string> service) : base(service)
    {
    }

    [HttpPost]
    [Route("CreateLink")]
    public IActionResult CreateLink([FromBody] LinkDto link)
    {
        service.Create(link);

        return Ok();
    }

    [HttpGet]
    [Route("GetLink")]
    public IActionResult GetLink(string id)
    {
        var link = service.Get(id);

        return Ok(link);
    }

    [HttpGet]
    [Route("GetLinks")]
    public IActionResult GetLinks()
    {
        var links = service.GetAll();

        return Ok(links);
    }

    [HttpPost]
    [Route("UpdateLink")]
    public IActionResult UpdateLink([FromBody] LinkDto link)
    {
        service.Update(link);
    }
}

```



```

        return Ok();
    }

    [HttpGet]
    [Route("DeleteLink")]
    public IActionResult DeleteLink(string id)
    {
        service.Delete(id);

        return Ok();
    }
}
}

```

Класс *Startup*:

```

using LinkShorteningService.BusinessLogic.DTO;
using LinkShorteningService.BusinessLogic.Extensions;
using LinkShorteningService.BusinessLogic.Implementation;
using LinkShorteningService.BusinessLogic.Interfaces;
using LinkShorteningService.Properties;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

```

```

namespace LinkShorteningService
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }
    }
}

```

// This method gets called by the runtime. Use this method to add services to the container.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    string connStr = Configuration.GetConnectionString(Resources.connectionString);
    services.RegisterEfAsUnitOfWork(connStr);

    //services.AddTransient<IUserService, UserService>();
    services.AddTransient<IService<UserDto, int>, UserService>();
    services.AddTransient<IService<GroupDto, int>, GroupService>();
    services.AddTransient<IService<LinkDto, string>, LinkService>();
}

// This method gets called by the runtime. Use this method to configure the HTTP request
pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
}

```

Класс *Program*:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

```

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace LinkShorteningService
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}

```

Класс *GroupDto*:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace LinkShorteningService.BusinessLogic.DTO
{
    public class GroupDto
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int UserId { get; set; }
    }
}

```

Класс *LinkDto*:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace LinkShorteningService.BusinessLogic.DTO
{
    public class LinkDto
    {
        public string Id { get; set; }
        public string Url { get; set; }
        public int ClickCount { get; set; }
        public int GroupId { get; set; }
    }
}

```

Класс *IServiceCollectionExtensions*:

```

using LinkShorteningService.DataAccess.Interfaces;
using LinkShorteningService.DataAccess.Logic;
using Microsoft.Extensions.DependencyInjection;

namespace LinkShorteningService.BusinessLogic.Extensions
{
    public static class IServiceCollectionExtensions
    {
        public static void RegisterEfAsIUnitOfWork(this IServiceCollection services, string
connectionString)
        {
            services.AddTransient<IUnitOfWork>(service => new EfUnitOfWork(connectionString));
        }
    }
}

```

Класс *GroupService*:

```

using LinkShorteningService.BusinessLogic.DTO;
using LinkShorteningService.DataAccess.Interfaces;
using LinkShorteningService.DataAccess.Models;
using System;
using System.Collections.Generic;

```

```

using System.Linq;

namespace LinkShorteningService.BusinessLogic.Implementation
{
    public class GroupService : AbstractService<GroupDto, int>
    {
        public GroupService(IUnitOfWork unitOfWork) : base(unitOfWork)
        {
        }

        public override void Create(GroupDto item)
        {
            Group group = mapper.Map<Group>(item);
            unit.Groups.Create(group);
            unit.Save();
        }

        public override void Delete(int id)
        {
            unit.Groups.Delete(id);
            unit.Save();
        }

        public override void Dispose()
        {
            unit.Dispose();
        }

        public override GroupDto Get(int id)
        {
            var model = unit.Groups.Get(id);
            var group = mapper.Map<GroupDto>(model);
            return group;
        }

        public override IEnumerable<GroupDto> GetAll()
        {
            var models = unit.Groups.GetAll();
            var groups = models.Select(x => mapper.Map<GroupDto>(x));
            return groups;
        }
    }
}

```

```

    }

    public override void Update(GroupDto group)
    {
        var model = unit.Groups.Get(group.Id);
        model.Name = group.Name;
        model.UserId = group.UserId;
        unit.Groups.Update(model);
        unit.Save();
    }
}
}

```

Класс *LinkService*:

```

using LinkShorteningService.BusinessLogic.DTO;
using LinkShorteningService.DataAccess.Interfaces;
using LinkShorteningService.DataAccess.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LinkShorteningService.BusinessLogic.Implementation
{
    public class LinkService : AbstractService<LinkDto, string>
    {
        public LinkService(IUnitOfWork unitOfWork) : base(unitOfWork)
        {
        }

        public override void Create(LinkDto item)
        {
            item.Id = Guid.NewGuid().ToString().Replace("-", "").Substring(0, 10); //TODO:
            Придумать лучшее место для генерации id и проверки на уникальность
            Link link = mapper.Map<Link>(item);
            unit.Links.Create(link);
            unit.Save();
        }
    }
}

```

```

public override void Delete(string id)
{
    unit.Links.Delete(id);
    unit.Save();
}

public override void Dispose()
{
    unit.Dispose();
}

public override LinkDto Get(string id)
{
    var model = unit.Links.Get(id);
    var link = mapper.Map<LinkDto>(model);
    return link;
}

public override IEnumerable<LinkDto> GetAll()
{
    var models = unit.Links.GetAll();
    var links = models.Select(x => mapper.Map<LinkDto>(x));
    return links;
}

public override void Update(LinkDto link)
{
    var model = unit.Links.Get(link.Id);
    model.Url = link.Url;
    model.ClickCount = link.ClickCount;
    model.GroupId = link.GroupId;
    unit.Links.Update(model);
    unit.Save();
}
}
}

```

Класс *IService*:

```
using System;
```

```

using System.Collections.Generic;
using System.Text;

namespace LinkShorteningService.BusinessLogic.Interfaces
{
    public interface IService<T, K>
    {
        void Create(T item);
        T Get(K id);
        IEnumerable<T> GetAll();
        void Update(T item);
        void Delete(K id);
        void Dispose();
    }
}

```

Класс *ApplicationUserManager*:

```

using LinkShorteningService.DataAccess.Repositories.EFRepositories.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using System;
using System.Collections.Generic;
using System.Text;

namespace LinkShorteningService.DataAccess.Identity
{
    public class ApplicationUserManager : UserManager<EfUser>
    {
        public ApplicationUserManager(IUserStore<EfUser> store) : base(store,)
        {
        }
    }
}

```

Класс *IRepository*:

```

using System;
using System.Collections.Generic;

```



```

using System.Text;

namespace LinkShorteningService.DataAccess.Interfaces
{
    public interface IRepository<T> where T : class
    {
        IEnumerable<T> GetAll();
        T Get(int id);
        T Get(string id);
        void Create(T item);
        void Update(T item);
        void Delete(int id);
        void Delete(string id);
        void DeleteAll();
    }
}

```

Класс *IUnitOfWork*:

```

using LinkShorteningService.DataAccess.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace LinkShorteningService.DataAccess.Interfaces
{
    public interface IUnitOfWork : IDisposable
    {
        IRepository<Link> Links { get; }
        IRepository<Group> Groups { get; }
        IRepository<User> Users { get; }
        void Save();
    }
}

```

Класс *EfUnitOfWork*:

```

using LinkShorteningService.DataAccess.Interfaces;
using LinkShorteningService.DataAccess.Models;
using LinkShorteningService.DataAccess.Repositories;

```

```
using LinkShorteningService.DataAccess.Repositories.EFRepositories.Models;
using Microsoft.EntityFrameworkCore;
using System;
```

```
namespace LinkShorteningService.DataAccess.Logic
{
    public class EfUnitOfWork : IUnitOfWork, IDisposable
    {
        private bool disposed = false;
        private LinksContext db;
        private IRepository<Link> linkRepository;
        private IRepository<Group> groupRepository;
        private IRepository<User> userRepository;

        public EfUnitOfWork(string connectionString)
        {
            var optionsBuilder = new DbContextOptionsBuilder<LinksContext>();
            optionsBuilder.UseSqlServer(connectionString);
            db = new LinksContext(optionsBuilder.Options);
        }

        public IRepository<Link> Links
        {
            get
            {
                if (linkRepository == null)
                {
                    linkRepository = new LinkRepository(db);
                }

                return linkRepository;
            }
        }

        public IRepository<Group> Groups
        {
            get
            {
                if (groupRepository == null)
                {

```

```

        groupRepository = new GroupRepository(db);
    }

    return groupRepository;
}

}

public IRepository<User> Users
{
    get
    {
        if (userRepository == null)
        {
            userRepository = new UserRepository(db);
        }

        return userRepository;
    }
}

public void Save()
{
    db.SaveChanges();
}

public virtual void Dispose(bool disposing)
{
    if (!this.disposed)
    {
        if (disposing)
        {
            db.Dispose();
        }

        this.disposed = true;
    }
}

public void Dispose()
{

```

```

        Dispose(true);
        GC.SuppressFinalize(this);
    }
}

```

Класс *Group*:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace LinkShorteningService.DataAccess.Models
{
    public class Group
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int UserId { get; set; }
    }
}

```

Класс *Link*:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace LinkShorteningService.DataAccess.Models
{
    public class Link
    {
        public string Id { get; set; }
        public string Url { get; set; }
        public int ClickCount { get; set; }
        public int GroupId { get; set; }
    }
}

```

Класс *ConverterExtensions*:

```

using LinkShorteningService.DataAccess.Models;
using LinkShorteningService.DataAccess.Repositories.EFRepositories.Models;

namespace LinkShorteningService.DataAccess.Repositories.EFRepositories.Extensions
{
    public static class ConverterExtensions
    {
        public static Group ToGroup(this EfGroup group)
        {
            return new Group
            {
                Id = group.Id,
                Name = group.Name,
                UserId = group.UserId
            };
        }

        public static EfGroup ToEfGroup(this Group group)
        {
            return new EfGroup
            {
                Id = group.Id,
                Name = group.Name,
                UserId = group.UserId
            };
        }

        public static Link ToLink(this EfLink link)
        {
            return new Link
            {
                ClickCount = link.ClickCount,
                GroupId = link.GroupId,
                Id = link.Id,
                Url = link.Url
            };
        }

        public static EfLink ToEfLink(this Link link)
        {

```

```

        return new EfLink
        {
            ClickCount = link.ClickCount,
            GroupId = link.GroupId,
            Id = link.Id,
            Url = link.Url
        };
    }

    public static User ToUser(this EfUser user)
    {
        return new User
        {
            Id = user.Id,
            Login = user.UserName,
        };
    }
}
}
}

```

Класс *EfGroup*:

```

using LinkShorteningService.DataAccess.Models;
using System.Collections.Generic;

namespace LinkShorteningService.DataAccess.Repositories.EFRepositories.Models
{
    public class EfGroup
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public int UserId { get; set; }
        public EfUser User { get; set; }
        public IEnumerable<EfLink> Links { get; set; }
    }
}

```

Класс *EfLink*:

```

namespace LinkShorteningService.DataAccess.Repositories.EFRepositories.Models
{
    public class EfLink
    {
        public string Id { get; set; }
        public string Url { get; set; }
        public int ClickCount { get; set; }

        public int GroupId { get; set; }
        public EfGroup Group { get; set; }
    }
}

```

Класс *LinksContext*:

```

using LinkShorteningService.DataAccess.Models;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace LinkShorteningService.DataAccess.Repositories.EFRepositories.Models
{
    public class LinksContext : IdentityDbContext<EfUser>
    {
        public LinksContext(DbContextOptions<LinksContext> options) : base(options)
        {
            Database.EnsureCreated();
        }

        public virtual DbSet<EfLink> Links { get; set; }
        public virtual DbSet<EfGroup> Groups { get; set; }
    }
}

```

Класс *GroupRepository*:

```

using LinkShorteningService.DataAccess.Interfaces;
using LinkShorteningService.DataAccess.Models;
using LinkShorteningService.DataAccess.Repositories.EFRepositories.Extensions;
using LinkShorteningService.DataAccess.Repositories.EFRepositories.Models;
using Microsoft.EntityFrameworkCore.Query.SqlExpressions;

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LinkShorteningService.DataAccess.Repositories
{
    public class GroupRepository : AbstractRepository<Group>
    {
        public GroupRepository(LinksContext db) : base(db)
        {
        }

        public override void Create(Group item)
        {
            db.Groups.Add(item.ToEfGroup());
        }

        //public void Delete(int id)
        //{
        //    var group = db.Groups.Find(id);

        //    if (group != null)
        //    {
        //        db.Groups.Remove(group);
        //    }
        //}

        public override void Delete(string strId)
        {
            throw new NotImplementedException();
        }

        public override void Delete(int id)
        {
            var group = db.Groups.Find(id);
            if (group == null)
            {
                return;
            }
        }
    }
}

```



```

        db.Groups.Remove(group);
    }

    public override void DeleteAll()
    {
        db.Groups.RemoveRange(db.Groups);
    }

    //public Group Get(int id)
    //{
    //    return db.Groups.Find(id);
    //}

    public override Group Get(int id)
    {
        return db.Groups.Find(id).ToGroup();
    }

    public override Group Get(string id)
    {
        throw new NotImplementedException();
    }

    public override IEnumerable<Group> GetAll()
    {
        return db.Groups.Select(group => group.ToGroup());
    }

    public override void Update(Group item)
    {
        var group = db.Groups.FirstOrDefault(us => us.Id == item.Id);
        group.Name = item.Name;
        group.UserId = item.UserId;
        db.Groups.Update(group);
    }
}
}

```

Класс *LinkRepository*:

```
using LinkShorteningService.DataAccess.Interfaces;
using LinkShorteningService.DataAccess.Models;
using LinkShorteningService.DataAccess.Repositories.EFRepositories.Extensions;
using LinkShorteningService.DataAccess.Repositories.EFRepositories.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace LinkShorteningService.DataAccess.Repositories
```

```
{
    public class LinkRepository : AbstractRepository<Link>
    {
        public LinkRepository(LinksContext db) : base(db)
        {
        }

        public override void Create(Link item)
        {
            db.Links.Add(item.ToEfLink());
        }

        public override void Delete(string id)
        {
            var link = db.Links.Find(id);

            if (link != null)
            {
                db.Links.Remove(link);
            }
        }

        public override void Delete(int id)
        {
            throw new NotImplementedException();
        }

        public override void DeleteAll()
        {
            //db.Links.RemoveRange(db.Links);
        }
    }
}
```

```

        if(!db.Links.Any())
        {
            return;
        }

        var ids = db.Links.Select(x => x.Id);
        db.Links.RemoveRange(db.Links.Where(x => ids.Contains(x.Id)));
    }

    //public Link Get(int id)
    //{
    //    throw new NotImplementedException();
    //}

    public override Link Get(string id)
    {
        return db.Links.Find(id).ToLink();
    }

    public override Link Get(int id)
    {
        throw new NotImplementedException();
    }

    public override IEnumerable<Link> GetAll()
    {
        return db.Links.Select(link => link.ToLink());
    }

    public override void Update(Link item)
    {
        var link = db.Links.FirstOrDefault(us => us.Id == item.Id);
        link.Url = item.Url;
        link.ClickCount = item.ClickCount;
        link.GroupId = item.GroupId;
        db.Links.Update(link);
    }
}
}

```