

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

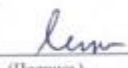
ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Разработка серверного компонента корпоративной доски объявлений»

Автор Мовила Геннадий Юрьевич
(Фамилия, Имя, Отчество) (Подпись)

Направление подготовки 09.03.02 Информационные системы и технологии

Квалификация Бакалавр
(бакалавр, инженер, магистр)

Руководитель Липкин Е.О. 
(Фамилия, И., О., ученое звание, степень) (Подпись)

К защите допустить

Руководитель ОП Парфенов В.Г., проф., д.т.н.
(Фамилия, И., О., ученое звание, степень) (Подпись)

“ ” 2019г.

Санкт-Петербург, 2019 г.

Студент Мовила Г.Ю Группа М3407 Факультет ИТиП

(Фамилия, И. О.)

Направленность (профиль), специализация "Информационные системы и технологии"

Консультант(ы):

а) _____
(Фамилия, И., О., ученое звание, степень) (Подпись)

б) _____
(Фамилия, И., О., ученое звание, степень) (Подпись)

ВКР принята "____" _____ 2019 г.

Оригинальность ВКР _____ %

ВКР выполнена с оценкой _____

Дата защиты "____" _____ 2019 г.

Секретарь ГЭК Маятин Александр Владимирович
(ФИО) _____ (подпись)

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

УТВЕРЖДАЮ

Руководитель ОП

проф. Парфенов В.Г. _____

(ФИО)

(подпись)

«29» «января» 2019 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студенту Мовила Г.Ю. Группа М3407 Факультет ИТиП

Руководитель ВКР Липкин Евгений Олегович, ИТМО, ассистент

(ФИО, ученое звание, степень, место работы, должность)

1 Наименование темы: Разработка серверного компонента корпоративной доски
объявлений

Направление подготовки (специальность) 09.03.02

Направленность (профиль) "Информационные системы и технологии"

Квалификация Бакалавр
(бакалавр, магистр, специалист)

2 Срок сдачи студентом законченной работы «22» «мая» 2019 г.

3 Техническое задание и исходные данные к работе

Проектирование архитектуры ПО, проектирование архитектуры БД, реализация
функциональных требований, реализация на основе проектирования, тестирование

4 Содержание выпускной работы (перечень подлежащих разработке вопросов)

Описание прикладного процесса, описание функциональных и нефункциональных
требований, проектирование системной архитектур, проектирование архитектуры данных,
разработка системы, тестирование полученных результатов

5 Перечень графического материала (с указанием обязательного материала)

Классовые диаграммы всех слоев архитектуры, диаграммы пакетов, диаграммы
последовательности, диаграмма состояний, диаграмма развертывания.

6 Исходные материалы и пособия

CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#, C#
7.0. Справочник. Полное описание языка Альбахари

7. Дата выдачи задания «22» «января» 2019г.

Руководитель _____


(подпись)

Задание принял к исполнению _____


(подпись)

«22» «января» 2019г.

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

АННОТАЦИЯ

ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Студент Мовила Геннадий Юрьевич
(ФИО)

Наименование темы ВКР: Разработка серверного компонента корпоративной доски объявлений

Наименование организации, где выполнена ВКР Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования разработать серверный компонент для корпоративной доски объявлений

2 Задачи, решаемые в ВКР Построение архитектуры ПО и БД, реализация основанная на проектировании

3 Число источников, использованных при составлении обзора 1

4 Полное число источников, использованных в работе 3

5 В том числе источников по годам

Отечественных			Иностранных		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
			X		

6 Использование информационных ресурсов Internet да, 1
(Да, нет, число ссылок в списке литературы)

7 Использование современных пакетов компьютерных программ и технологий (Указать, какие именно, и в каком разделе работы)

Пакеты компьютерных программ и технологий	Параграф работы
Visual Paradigm 13	1,2,3
.NET Core 2.2	3
AutoMapper	3
Entity Framework	3
PostgreSQL	2,3

8 Краткая характеристика полученных результатов были реализованны все функциональные и нефункциональные требования, спроектированная, а затем реализованная системная архитектура, так же проведено тестирование и построена база данных.

9 Полученные гранты, при выполнении работы _____
(Название гранта)

10 Наличие публикаций и выступлений на конференциях по теме выпускной работы _____
(Да, нет)

а) 1 _____
(Библиографическое описание публикаций)
2 _____
3 _____

б) 1 _____
(Библиографическое описание выступлений на конференциях)
2 _____
3 _____

Студент Мовшица Т.Ю. сфн
(ФИО) (подпись)

Руководитель Митин Е.В. Евг
(ФИО) (подпись)

“ 6 ” июня 2019г.

Оглавление

Список терминов и сокращений.....	5
Введение.....	6
Глава 1. Анализ предметной области.....	7
1.1 Описание прикладного процесса	7
1.2. Требования к информационной системе.....	10
1.2.1 Функциональные требования.....	10
1.2.2 Нефункциональные требования.....	13
Требования к технологиям:	13
Глава 2. Проектирование системы	14
2.1 Технологии, используемые в разработке системы.....	14
2.1.1 .NET Core Web API.....	14
2.1.2 Entity Framework	14
2.1.3 LINQ	15
2.1.4 Docker	15
2.1.5 PostgreSQL.....	15
2.3 Проектирование системной архитектуры	17
2.4 Проектирование программной архитектуры.....	18
2.4.1.DAL-уровень.	18
2.4.2 Domain уровень	22
2.4.3 Bll Application-уровень и API уровень	24
2.4.3 Проектирование архитектуры данных	27
Глава 3. Программная реализация	29
3.1 Реализация DAL и Domain уровней	29
3.2 Реализация API и BLL уровней.....	31
3.1 Реализация OAuth авторизации.....	35
Глава 4. Тестирование	37
Заключение.....	40
Список используемых источников	41
Приложение А. Диаграммы Bll и API уровней	42

Список терминов и сокращений

DAL (Data Access Layer) – слой архитектуры в котором описаны модели, а также репозитории моделей.

Модель – это представление таблицы базы данных в коде.

Репозиторий – слой архитектуры, выполняющий CRUD операции.

CRUD (**Create, Read, Update, Delete**) – четыре базовые функции, дословно обозначают: создание, чтение, редактирование, удаление.

Entity Framework (EF) – это инструмент, упрощающий сопоставление объектов в программном обеспечении с таблицами и столбцами реляционной базы данных.

БД (База данных) – это организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации, преимущественно больших объемов.

PG (PostgreSQL) - свободная объектно-реляционная система управления базами данных

JWT (JSON web token) - это открытый стандарт для создания токенов доступа, основанный на формате JSON. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.

Маппинг/Мапить/Мапится – термин обозначающий проекцию одной сущности на другую.

Введение

Последнее время сайты по типу «доски объявлений» набирают всё большую популярность. Всему миру известны такие площадки как: Amazon, eBay, AliExpress, Gear Best, Taobao. Основной функционал данных площадок заключается в размещении объявления продавцом. Пользователи имеют возможность покупать товар, оценивать продавца, оценивать товар, на этих площадках находятся не только лица по типу индивидуальных предпринимателей или какие-либо частные лица, но и официальные компании, такие как Microsoft, Samsung, LG и так далее.

Так же, появилась тенденция по созданию собственных досок объявлений для отдельной компании. Например, очень известная в Японии компания Rakuten имеет подобный сайт, доступ к которому есть только у сотрудников, объясняется это тем, что людям гораздо проще продавать или же обменивать товары друг с другом, если за ними стоит такой грант доверенности как репутация компании, а так же, людям гораздо проще взаимодействовать друг с другом, зная, что они находятся в одной сфере деятельности. Например, в IT компании.

На размещении, оценке, комментировании и покупке - функционал данных площадок не ограничивается, так, на eBay могут быть аукционы, на Mass Drop существует функция «записаться на дроп», грубо говоря, как только набирается определённое количество человек, которые хотят этот товар, начинается оптовая закупка со стороны сервиса, в данном случае Mass Drop-a.

Целью работы является разработка одностраничного веб приложения, которое позволит автоматизировать процесс подачи и рассмотрения объявлений для сотрудников компании ООО «ЕПАМ СИСТЭМЗ».

Для осуществления данной цели необходимо последовательно решить несколько следующих задач:

- Описать требования к разрабатываемой системе;
- Выбрать стек технологий для разработки, исходя из сформулированных требований
- Описать функциональную, информационную, системную, программную архитектуры и архитектуру данных системы;
- Выполнить разработку системы, согласно требованиям;
- Провести тестирование системы (Unit тестирование, Интеграционное тестирование)

Важно отметить то, что на сайте не производится оплата товаров. Это противоречит политике компании, вся оплата происходит путем личной договорённости между сотрудниками.

Глава 1. Анализ предметной области

Началом анализа послужит изучение потребности создания приложения.

Основные потребности:

- Торговля между работниками фирмы в более комфортных условиях. Были проведены опросы внутри фирмы, результаты которых доказали предположение о том, что людям гораздо комфортнее торговаться между собой, если они являются сотрудниками одной фирмы, т.к. появляется некий фактор доверия, обусловленный репутацией фирмы. Именно это и стало ключевым фактором в создании данной площадки, а не использование каких-либо сторонних площадок, плюсом так же является то, что ЕРАМ большая компания от этого и вопрос просматриваемости объявления не стоит.
- Регистрация на внутренние мероприятия таких как: настольные игры, общие лекции, митапы, конференции.

Таким образом, содержание глав настоящей работы будет построено следующим образом:

Первая глава содержит информацию, связанную с описанием прикладного процесса, требованиями к системе, а также набором технологий, на которых будет базироваться система

Вторая глава включает в себя результаты проектирования функциональной, информационной, системной, программной архитектур и архитектуры данных системы соответственно. Кроме того, содержит описания взаимодействия компонентов системы.

В третьей главе описываются итоги разработки и тестирования системы.

1.1 Описание прикладного процесса

Прикладной процесс может рассматриваться с двух сторон:

- С точки зрения пользователя
- С точки зрения владельца поста

Отличия имеются лишь в том, что у создателя поста добавляется новый рабочий поток, который олицетворяет работу непосредственно с заполнением обязательных полей при размещении объявлений.

Описание процесса с точки зрения пользователя можно наблюдать на Рис.1

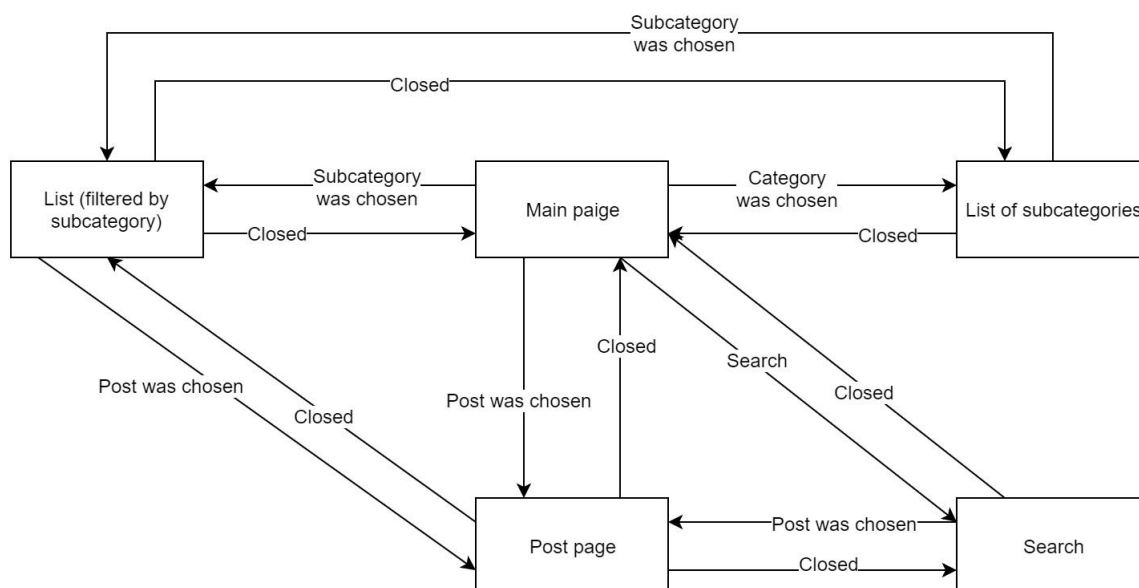


Рисунок 1 – Диаграмма состояний с точки зрения роли User (Customer)

Всё начинается с Main Page компонента, затем система реагирует в зависимости от действий пользователя.

Как пример, если пользователь выбирает категорию постов, на которые он хочет взглянуть, то после её выбора, его запрос отправится в компонент List (filtered by subcategory), после чего поток обращается к компоненту Post Page. Таким образом пользователь увидит пост, который удовлетворяет условиям поиска.

В системе существует определённое количество ролей, взаимодействия которых наглядно показаны на Рис.2.

Content administrator – роль, которая отвечает за модерацию объявлений/комментариев. Имеет такой функционал как:

- Заблокировать пост
- Корректировать пост
- Пометить комментарий некорректным по отношению к пользователям

User (Customer) – роль, которая присваивается всем, кто не является Post's owner-ом. Но при этом любой User в любой момент может стать Post's owner-ом.

Имеет такой функционал как:

- Подписываться на тот или иной фильтр
- Фильтровать посты
- Комментировать объявления
- Отправлять фидбек
- Создавать объявления
- Создавать неуместные объявления

Post's Owner – роль, которая присваивается всем, кто создал объявление. Имеет такой функционал как:

- Заполнить детали поста
- Просрочить пост
- Комментировать
- Удалить пост вручную

Системные роли:

EPAM SSO – роль, которая имеет смысл только на системном уровне. Позволяет авторизовывать пользователей в систему. Имеет всего одну функцию – это отправка токена, в ответ на запрос системы.

EPAM Sharepoint – роль, которая позволяет интегрировать те или иные события внутри продуктов Microsoft. Например, когда вы забронировали вещь, она автоматически добавится в ваш Outlook календарь.

EPAM Bulletin Board – главная системная роль, которая взаимодействует со всеми модулями и компоует их вместе.

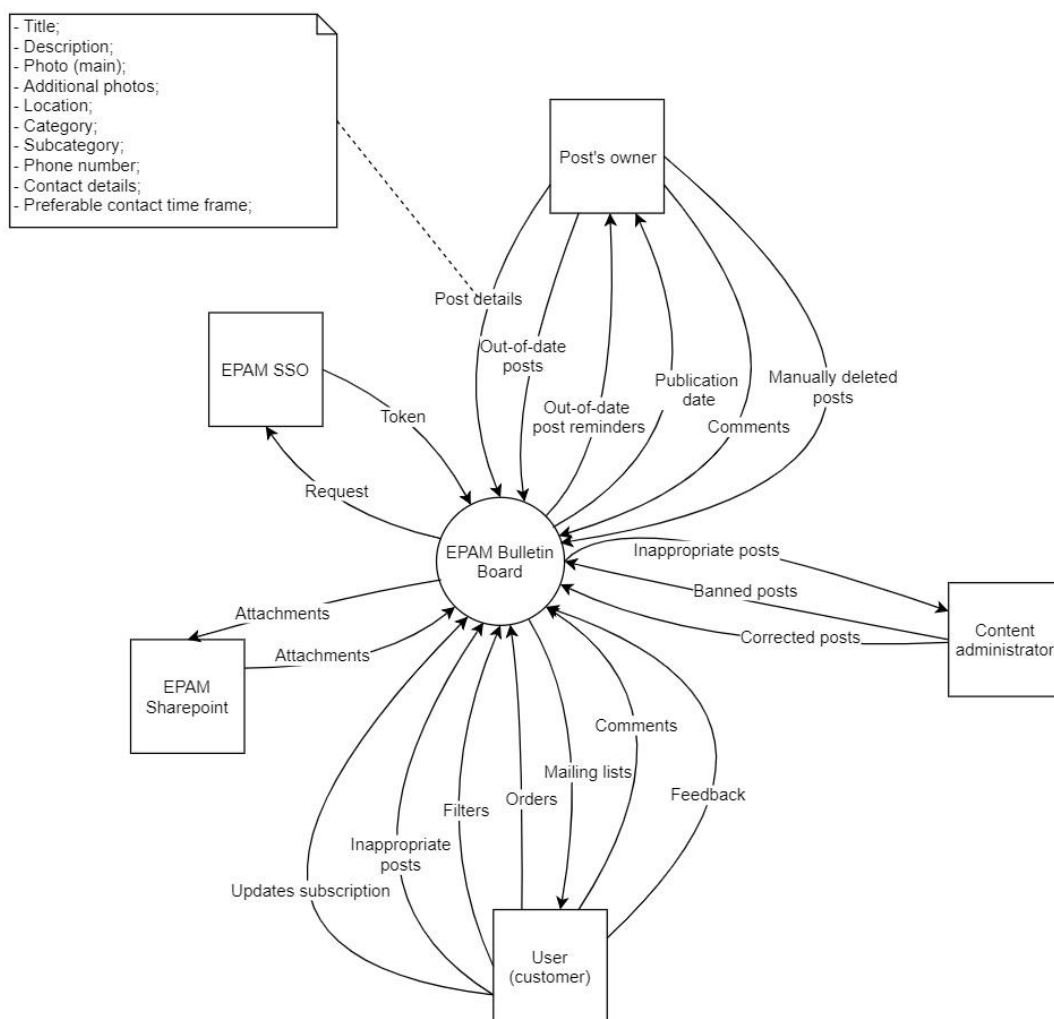


Рисунок 2. Роли системы, а также их функционал

Для того что бы добиться высокой производительности с точки зрения разработки, систему пришлось разделить на три части:

- Часть, связанная с авторизацией пользователя в системе;
- Часть, отвечающая за выдачу ролей пользователю;
- Часть, содержащая в себе основной функционал системы: создание поста, комментирование, поиск, подписка, фильтрация и т.д.

1.2. Требования к информационной системе

В подпунктах ниже описаны требования, которые были сформулированы в результате взаимодействия с заказчиком (компанией ЕРАМ). Поскольку объем функционала, который необходимо реализовать в системе достаточно велик, то в следующих подпунктах настоящей работы приведена только часть технического задания, которое необходимо выполнить в целом.

1.2.1 Функциональные требования

Функциональные требования к приложению:

1. Создавать объявления, которые содержат:
 - a. Название,
 - b. Описание
 - c. Главную Фотографию
 - d. Местонахождение товара/проведения события
 - e. Контакты владельца поста
 - f. Дополнительные фотографии
 - g. Категорию
 - h. Подкатеорию
 - i. Номер мобильного телефона
2. Комментировать то или иное объявление
3. Разграничение постов на категории и подкатегории
4. Механизм подписки на
 - a. Категорию
 - b. Имя
 - c. Минимальную цену
 - d. Максимальную цену
 - e. Подкатеорию
5. Фильтрация поиска постов по
 - a. Категориям
 - b. Имени
 - c. Минимальной цене

- d. Максимальной цене
- e. Подкатегориям
- 6. Поиск по ключевым словам
- 7. Выставления рейтинга продавцам
- 8. Возможность общения пользователя с владельцами сайта, в данном случае компанией ЕРАМ. (Фидбек)
- 9. Редактирование профиля, а именно редактирование
 - a. Фотографии
 - b. Офиса
 - c. Контактных данных
- 10. Модерирование комментариев и объявлений, а именно:
 - a. Указание на то, что комментарий/объявление является неприемлемым и противоречат уставу компании.

Рис.3-5 наглядно демонстрирует все функциональные требования.

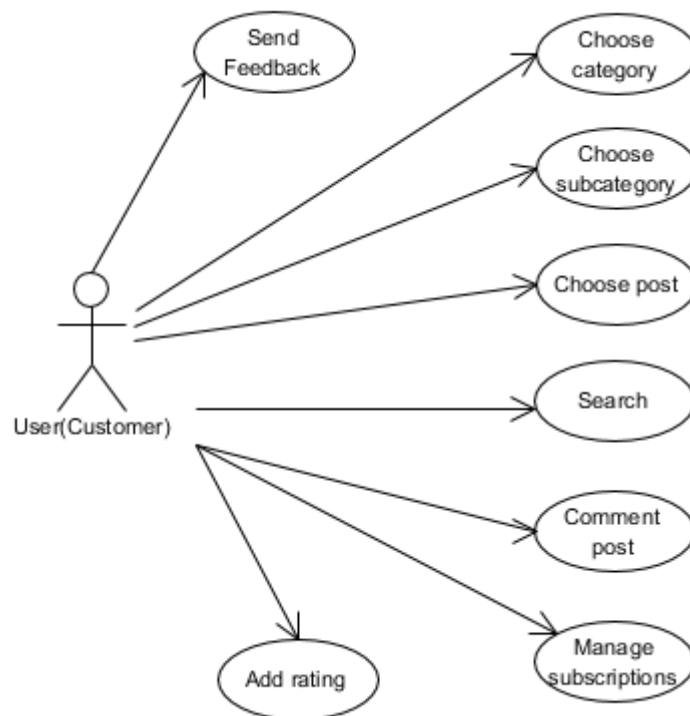


Рисунок 3. Диаграмма вариантов использования пользователя

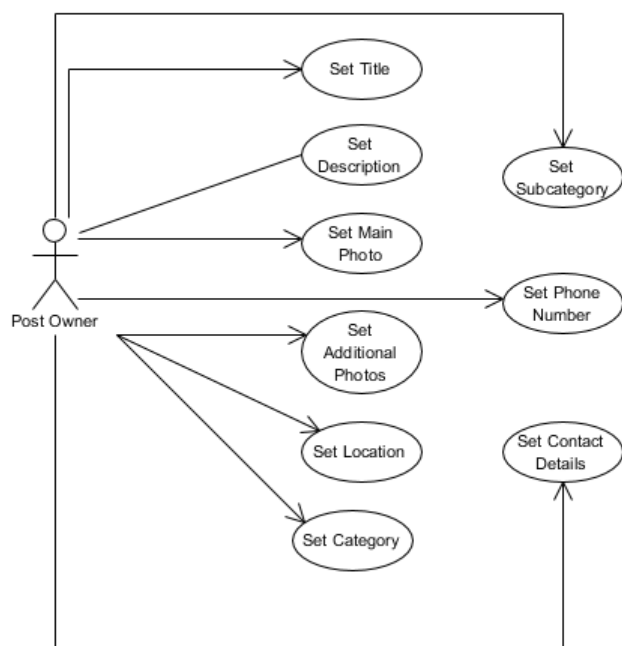


Рисунок 4. Диаграмма вариантов использования владельца поста (Создание объявления)

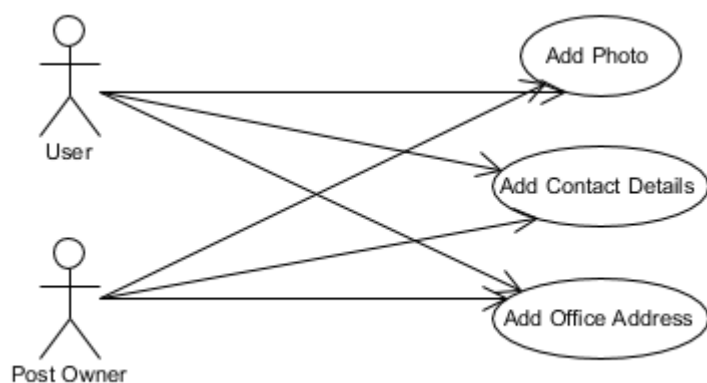


Рисунок 5. Диаграмма вариантов использования редактирования профиля

На Рис.6 представлено дерево категории, существующие в приложении. С деревом работают такие механизмы как: подписка, фильтрация, поиск.

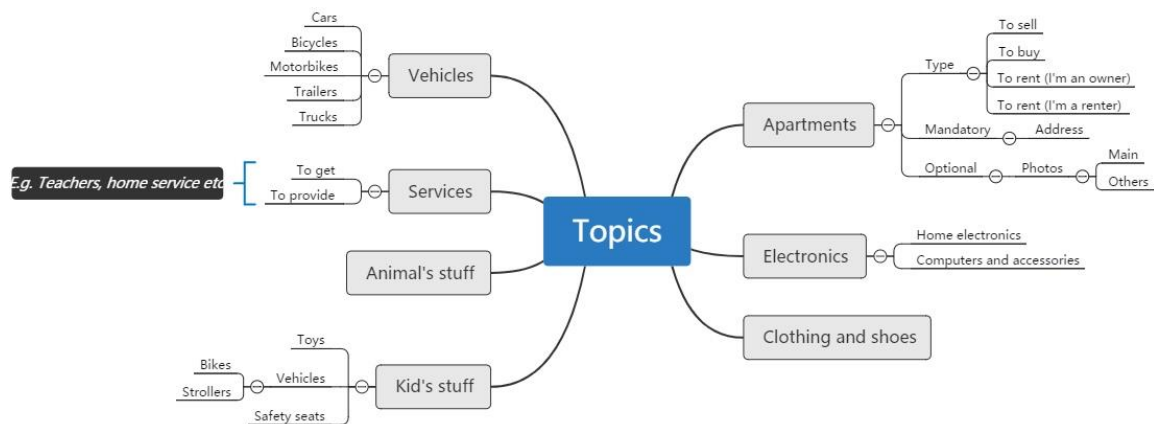


Рисунок 6. Древо категорий

1.2.2 Нефункциональные требования

Требования к технологиям:

1. .Net Core (последняя версия)
2. AutoMapper
3. Entity Framework
4. PostgreSQL
5. NUnit
6. Moq
7. Docker

Требования к безопасности:

1. При аутентификации пользователя должен использоваться Google сервис.
2. Система должна предусматривать неотложные ситуации путем создания бэкапов, а также иметь резервный сервер.
3. Система должна распределять нагрузку. Если система не может обрабатывать запросы всех пользователей, то должно увеличиться время запроса для половины пользователей, в то время как другая половина обрабатывается в штатном режиме.
4. Система должна предусматривать разграничение пользователей по ролям.

Требования к тестированию:

1. Система должна пройти нагрузочное тестирование, а именно: максимальное количество пользователей, максимально обрабатываемый поток информации, время отклика при максимальной загрузке.
2. Должны использовать как интеграционные, так и юнит тесты.
3. Должны быть написаны автоматические UI тесты

Глава 2. Проектирование системы

2.1 Технологии, используемые в разработке системы

2.1.1 .NET Core Web API

.NET Core Web API – один из самых быстро развивающихся Фреймворков семейства .NET. Включает в себя весь функционал .NET Framework и так же использует все преимущества .NET среды. Отличительной особенностью является кроссплатформенность, а также свежие взгляды на старые механизмы. Большое количество «синтаксического сахара» позволяет существенно сократить время на разработку, повысить уровень читаемости кода и сократить время на тестирование.

Так же Core включает в себя такие механизмы как IoC и DI контейнеры. Позволяющие регистрировать сервисы, а затем «доставать» их в нужный момент. .NET CORE обратно совместим, поэтому обновление на более новые версии Фреймворка не составит труда. К тому же он может легко разворачиваться в Docker контейнерах, речь о которых пойдет ниже

2.1.2 Entity Framework

Entity Framework – Object relation mapping (ORM) инструмент позволяющий легко взаимодействовать с базой данных путем манипуляций из кода. Существует два актуальных подхода, позволяющие работать с EF. В данной работе рассматривается лишь Code First подход, т.к. он и был задействован в проекте. Code First - достаточно элементарный подход, который гласит о том, что сначала описываются все модели внутри приложения, а затем с помощью EF и консольных команд стандартного пакета .NET CORE «накатывается» миграция на базу. Миграция – это автоматически сгенерированный SQL запрос. Суть миграций в том, что существует возможность безболезненно для базы, через код, менять любые связи между таблицами, а также добавлять новые поля, новые модели и всё это будет применено на базу данных. Так же есть возможность легко «откатить» последнюю миграцию, что существенно упрощает разработку.

Одной из главных функций EF является то, что он способен сам распутывать сложные зависимости без ручной конфигурации. Например – используя тип свойства/поля одного класса внутри другого, EF автоматически способен распутывать зависимости создавая JOIN запросы к БД при обращении к этому полю.

2.1.3 LINQ

LINQ – язык запросов. Позволяет легко манипулировать коллекциями, потоками ввода/вывода, но самое главное – что позволяет легко работать с Entity Framework-ом. С помощью данного инструмента можно легко выполнять сложные запросы путем «текущего синтаксиса». От самых простых селектов, до сложных вложенных джоиннов. Правда есть одно «но», за такую легкость в разработке приходится платить производительностью. Дело в том, что EF далеко не всегда строит оптимальные запросы к БД, а порой и совсем сильно усложняет простые запросы, которые должны выполняться мгновенно.

2.1.4 Docker

Docker – инструмент позволяющий автоматизировать развертывание и управление приложениями в средах с поддержкой контейнеризации. В данном случае докер разворачивается в Windows 10 ОС. Но на самом деле прослойкой между Windows и докером всё равно служит Linux ядро.

2.1.5 PostgreSQL

PostgreSQL – свободная объектно-реляционная система управления базами данных. Выбор СУБД был обусловлен большим количеством связей.

2.1.6 AutoMapper

AutoMapper – библиотека позволяющая проецировать одну сущность на другую путем конфигурационных файлов, либо же автоматически, если сущности совпадают по своему описанию.

2.2 Проектирование информационной архитектуры

Информационная архитектура организована в виде дерева следующим образом:

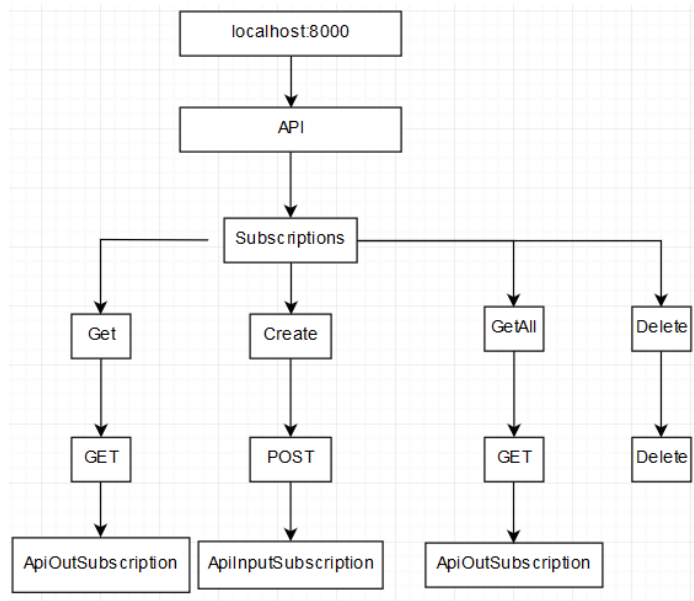


Рисунок 7 – Информационная архитектура на примере методов модуля подписки (Subscription).

Рис.7 представляет из себя граф роутинга, что означает представление пути от самого начала (сервера) до конца (выходной модели). За образец был взят обширный модуль Subscription, т.к. он охватывает все запросы, которые могут быть использованы в системе.

- Корнем является название сервера, на котором было размещено приложение (в данный момент: localhost: номер_порта)
- На втором уровне указывается прослойка, через которую идёт обращение к основным модулям
- На третьем уровне указывается название модуля (в данном случае Subscriptions)
- На четвертом уровне указывается название контроллера (Get, Create, GetAll, Delete)
- На пятом уровне указывается тип запроса
- На шестом уровне указывается бизнес-сущность, в которую будут складываться результаты запроса

Таким образом, в соответствии с вышеизложенной логикой запросы к системе, описанные на Рисунке 7, можно представить следующим образом

- a) localhost:8000/api/subscriptions/get/{id}
- b) localhost:8000/api/subscriptions/create
- c) localhost:8000/api/subscriptions/getall
- d) localhost:8000/api/subscriptions/delete/{id}

2.3 Проектирование системной архитектуры

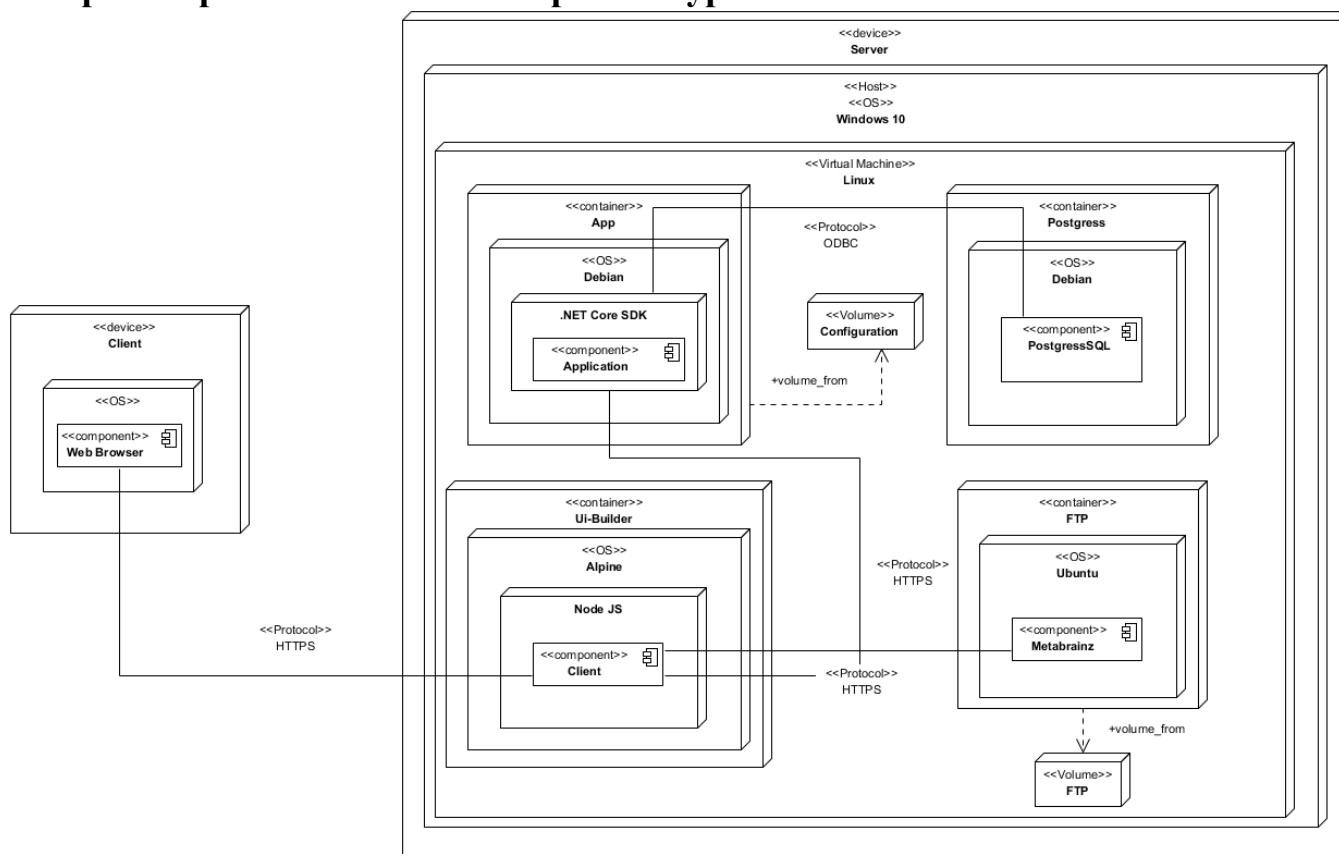


Рисунок 8 – Диаграмма развертывания

На Рис.8 показана общая архитектура приложения.

Пользователь системы работает через устройство, на котором установлен браузер. Браузер обращается по HTTPS протоколу к UI-Builder контейнеру, который в свою очередь разворачивает внутри себя Client компонент. Client компонент представляет из себя сборку UI части проекта, которая работает с помощью Webpack сборщика, который в свою очередь собирает React Framework.

App контейнер – это контейнер в котором разворачивается SDK, а так же вся backend часть приложения, Postgress контейнер представляет из себя экземпляр базы данных, FTP контейнер является обычным FTP сервером, для того что бы хранить там статичные элементы, такие как: картинки, логотипы, элементы UI и так далее.

Каждый контейнер разворачивает внутри себя свою гостевую ОС. А уже затем всю среду для запуска того или иного элемента.

2.4 Проектирование программной архитектуры

Приложение рассчитано на пользователей, которые используют персональные компьютеры/планшеты для работы. В качестве системной архитектуры был выбрана архитектура клиент-сервер.

Клиентом является пользовательское устройство, а сервером – удалённая машина. Т.к. приложение состоит из нескольких модулей, а именно: база данных, FTP сервер для хранения статичной информации, а также backend часть – то необходимо иметь несколько серверов, либо один, но достаточно мощный.

Данный между клиентом и сервером передаются по HTTPS запросам. Такие запросы backend от клиента после чего отправляет их в базу и в ответ получает заранее подготовленные backend-ом данные. Но возможна так же и ситуация, что не придётся отправлять запрос в базу и вся обработка произойдёт на сервере.

Архитектура backend части поделена на несколько слоев:

- Самый нижний слой архитектуры (DAL-уровень)
- Средний слой архитектуры (DOMAIN – уровень)
- Высокий слой архитектуры (Bll Application – уровень)
- Высший уровень архитектуры (API уровень)

2.4.1.DAL-уровень.

Содержит в себе описание всех моделей, а также напрямую связан с таблицами в БД с помощью ORM Entity Framework-а.

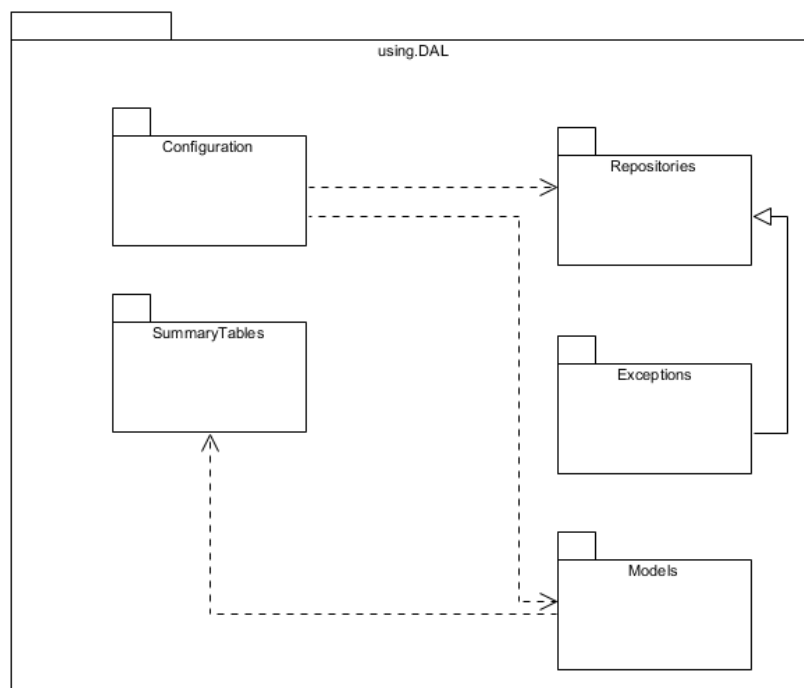


Рисунок 9. DAL-уровень архитектуры

Configuration – папка, которая содержит в себе настройки DbContext-а и DalMapperConfigurat-ора, первый отвечает за то, конфигурацию доступа к БД, второй же отвечает за конфигурацию «маппинга» моделей на БД, в нем обрабатываются исключительные ситуации которые не может обработаться EF автоматически.

Repositories – папка, которая содержит в себе классы, которые позволяют осуществлять самый низкий уровень CRUD операций. Сами CRUD операции, на самом деле описаны на самом верхнем уровне (API), здесь же как раз-таки и происходит «маппинг» таблиц, на модели, которые приходят из вне. Например, когда пользователь хочет получить лист пользователей, или же постов, то он выполняет Read запрос, который в свою очередь тянет целую цепочку классов, которая будет рассмотрена позже. На данном этапе рассмотрения будет достаточно того факта, что на уровне DAL мы производим маппинг наших таблиц, в наши сущности, что описаны на уровнях выше.

Exceptions – папка содержащая в себе классы, описывающие «кастомные» исключения.

Models – модели, которые и описывают таблицы БД.

SummaryTables – папка, которая содержит в себе классы сложных моделей.

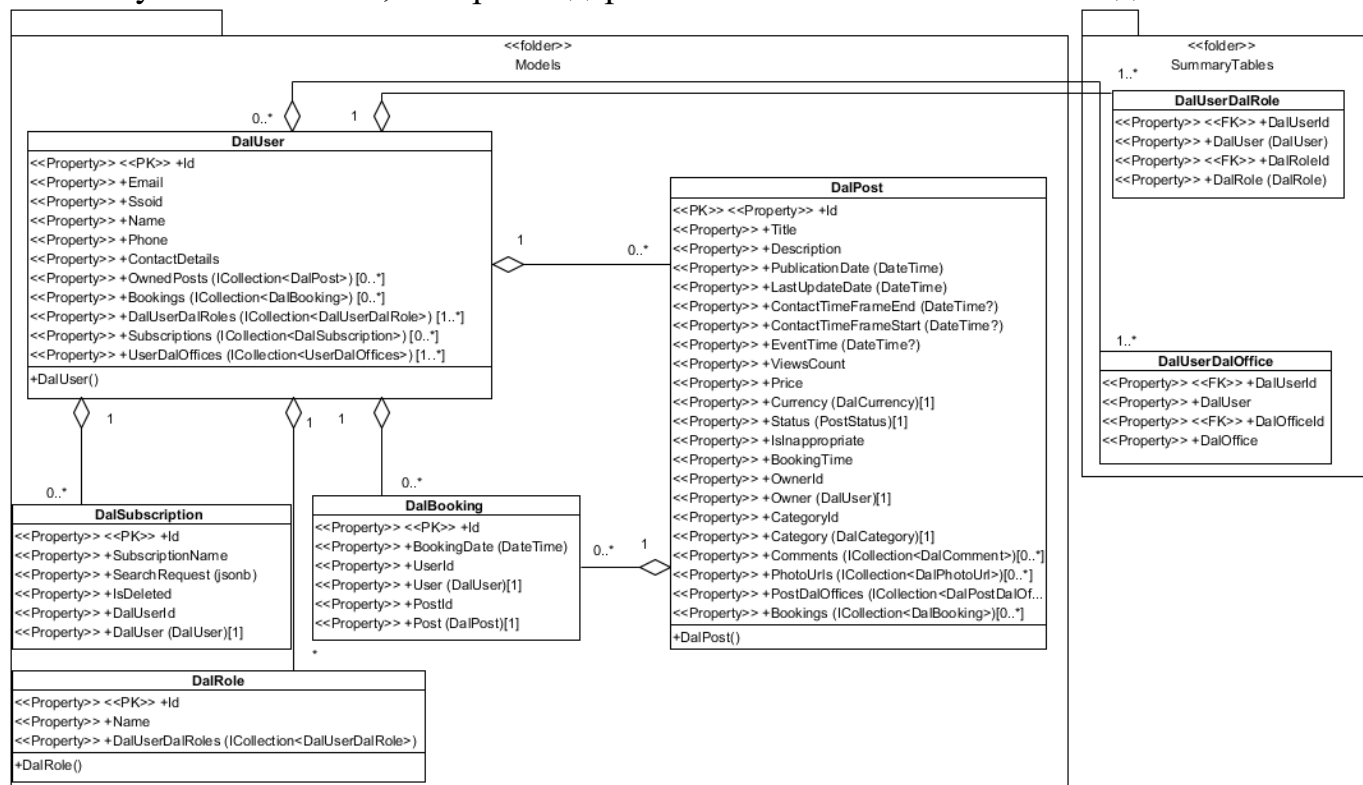


Рисунок 10. Описание моделей (DalUser, DalSubscription, DalRole, DalBooking, DalPost, DalUserDalRole, DalUserDalOffice)

Конструкторы - являются конструкторами с параметрами, но параметры не были представлены в диаграмме в угоду упрощения и читаемости. Каждая модель, представленная на Рис.10, имеет конструктор, который инициализирует экземпляры

сложных(составных) объектов. Таких как OwnedPosts, Bookings, DalUserDalRoles, Subscriptions, UserDalOffices.

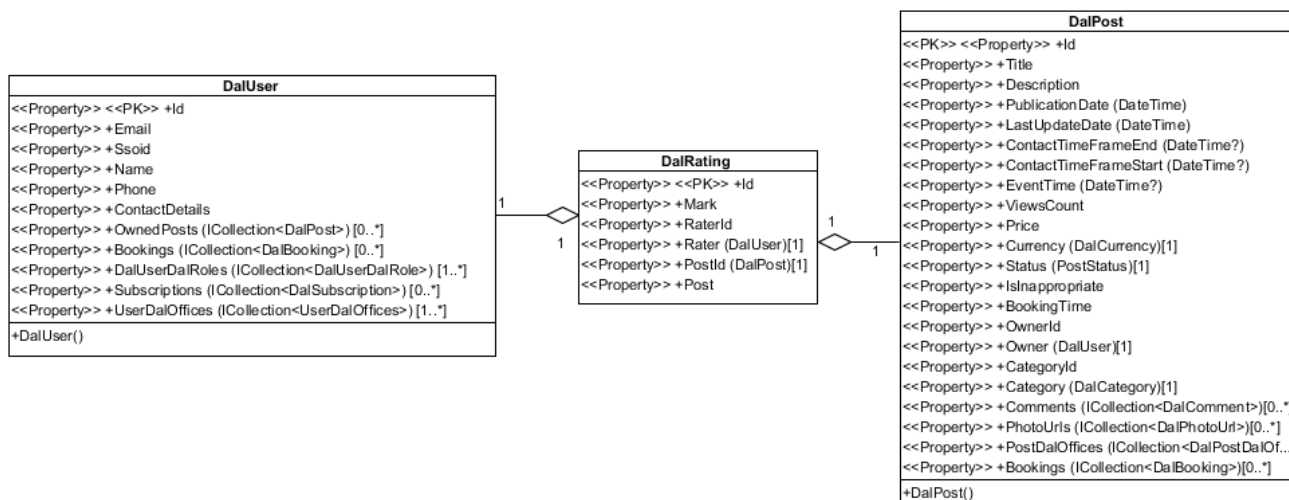


Рисунок 11. Описание модели (DalRating)

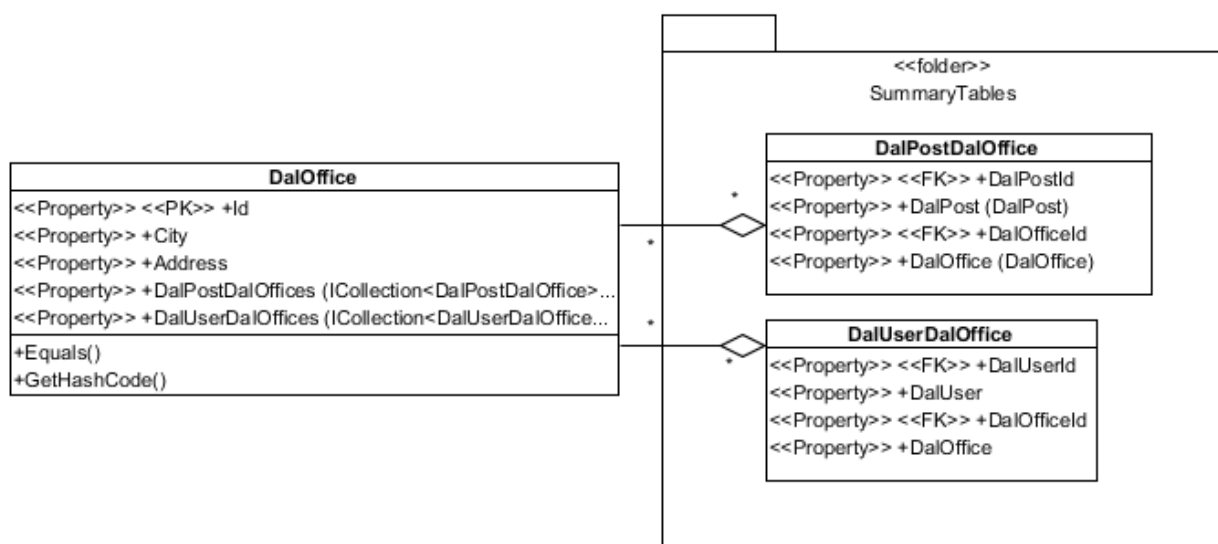


Рисунок 12. Описание модели DalOffice, а также двух составных моделей DalPostDalOffice, DalUserDalOffice.

Все модели, описанные в данном слое архитектуры, имеют прямое отношение к таблицам БД, т.к. применяется Code First подход, при генерации миграции все таблицы в БД имеют архитектуру из Рис.10,11,12,13 за исключением сложных составных полей, которые представлены только для EF для составления сложных запросов и обращений к другим таблицам баз данных с помощью LINQ to Entity запросов.

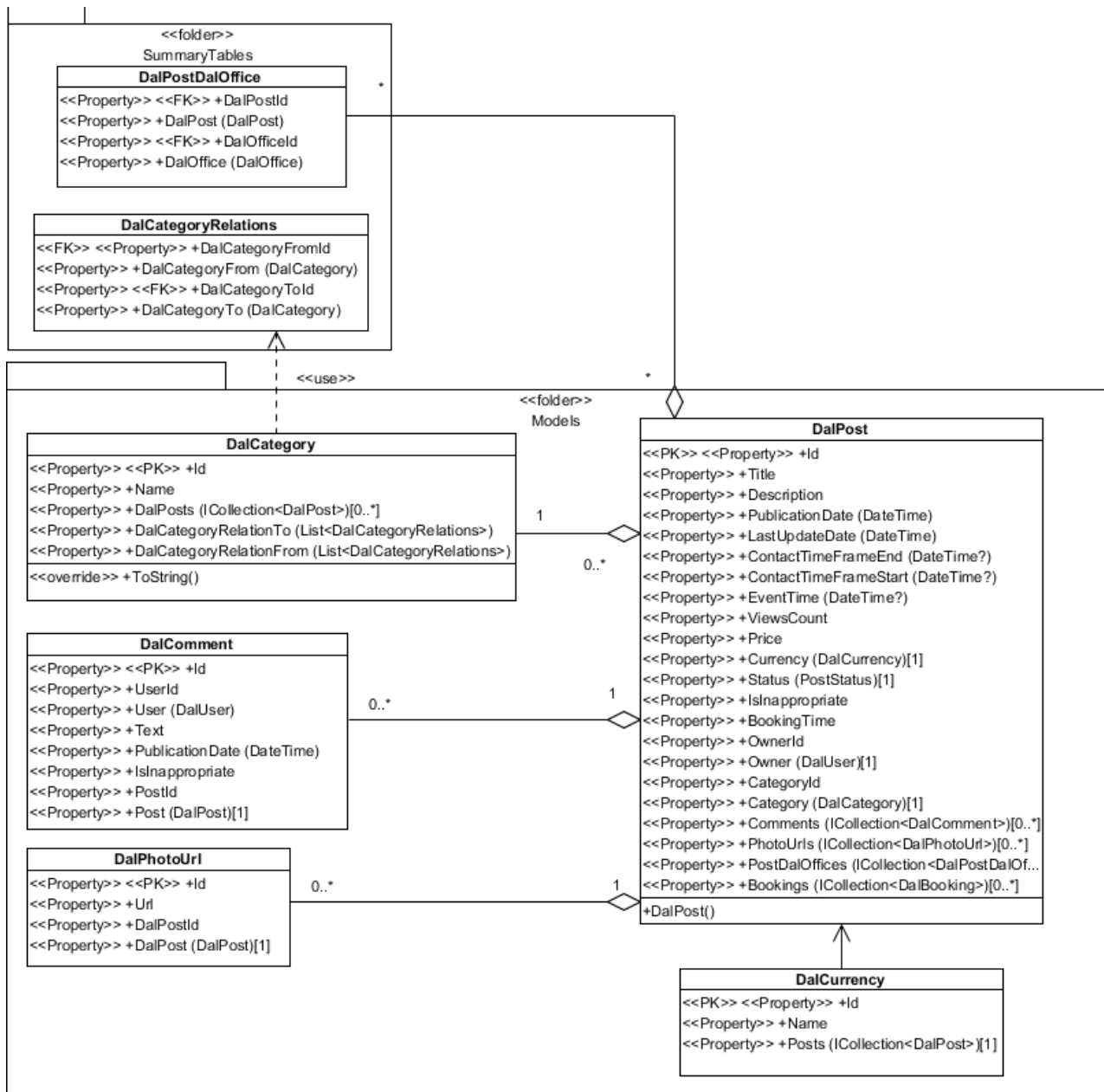


Рисунок 13. Описание моделей (DalPost, DalCurrency, DalComment, DalPhotoUrl, DalComment, DalCategory), а также двух составных моделей (DalCategoryRelations, DalPostDalOffices)

2.4.2 Domain уровень

Это уровень, в котором описаны все интерфейсы моделей, репозитория, сервисов, а также наши сущности.

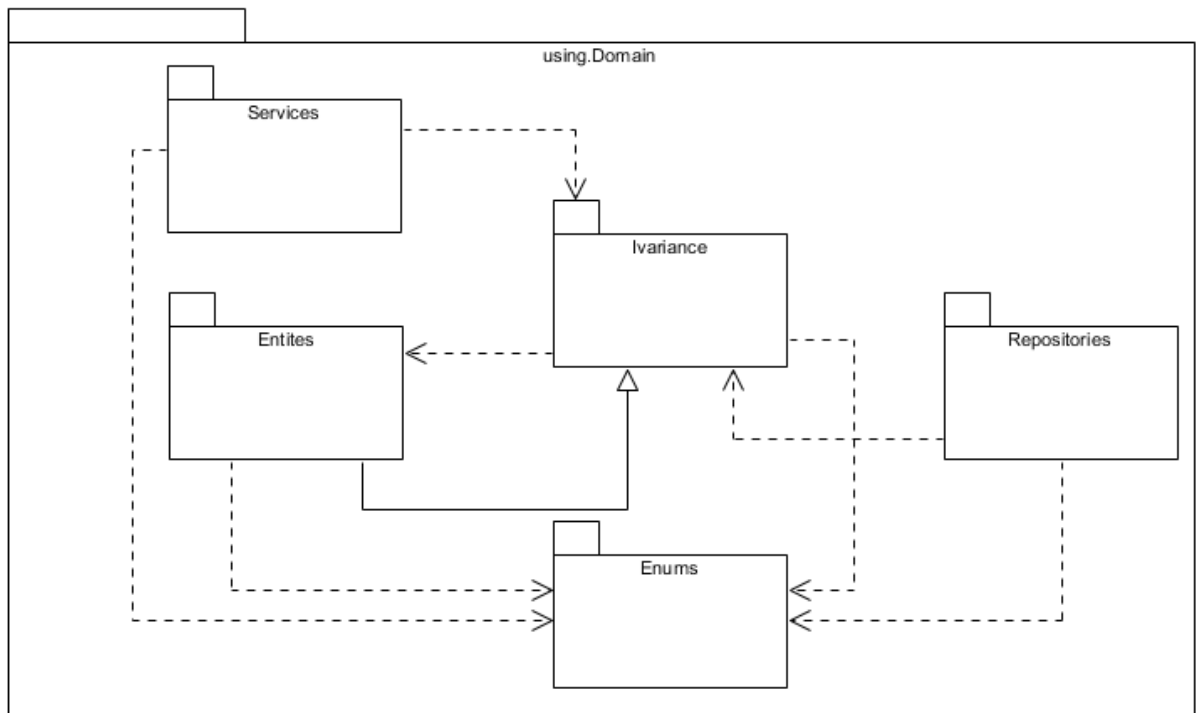


Рисунок 14. Domain-уровень архитектуры

Services – папка содержащая в себе классы-интерфейсы всех сервисов

Entities – папка содержащая в себе классы-интерфейсы всех сущностей

Invariance – папка содержащая в себе классы-интерфейсы всех моделей, который используются в API уровне.

Repositories – папка содержащая в себе классы-интерфейсы всех репозитория.

Enums – папка которая содержит в себе все перечисления.

Исходя из Рис.14 можно заметить, что все пакеты зависят от Invariance и Enums, обусловлено это тем, что в Invariance описаны все интерфейсы Entities, что, к слову, так же заметно из диаграммы т.к. Entities реализуют интерфейсы из Invariance.

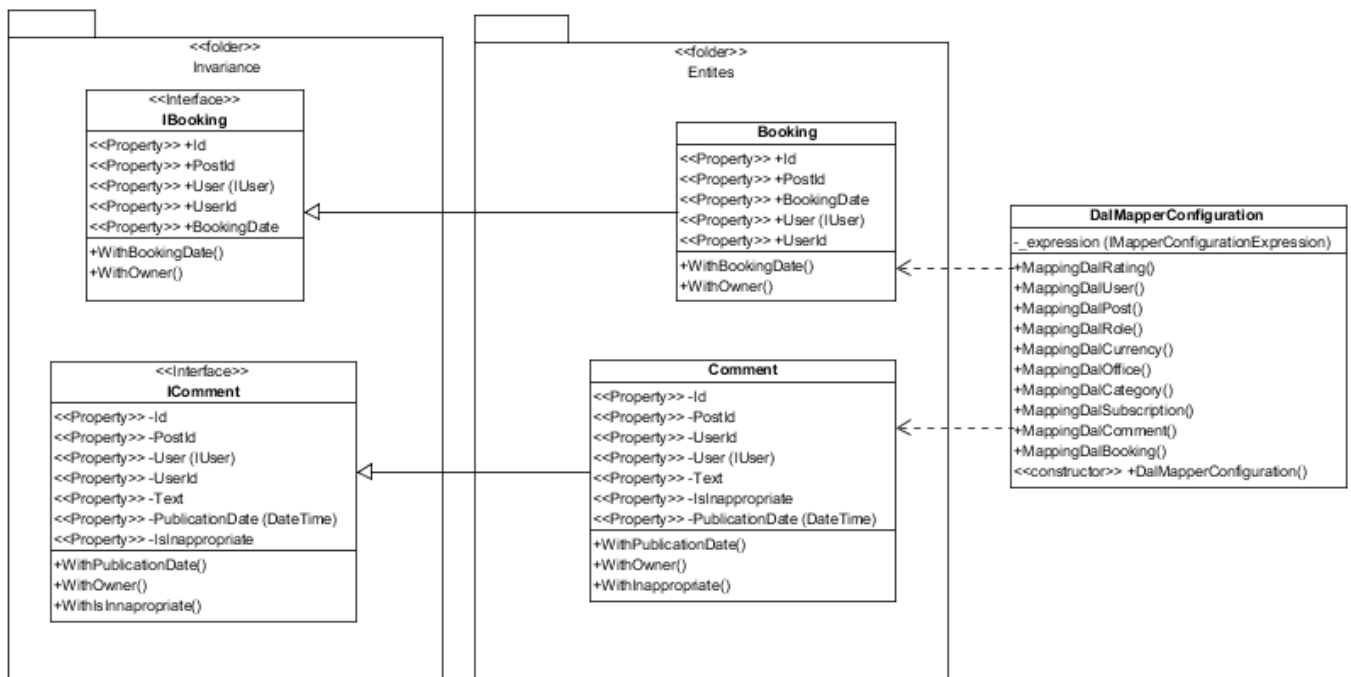


Рисунок 15. Классовая диаграмма Domain уровня архитектуры

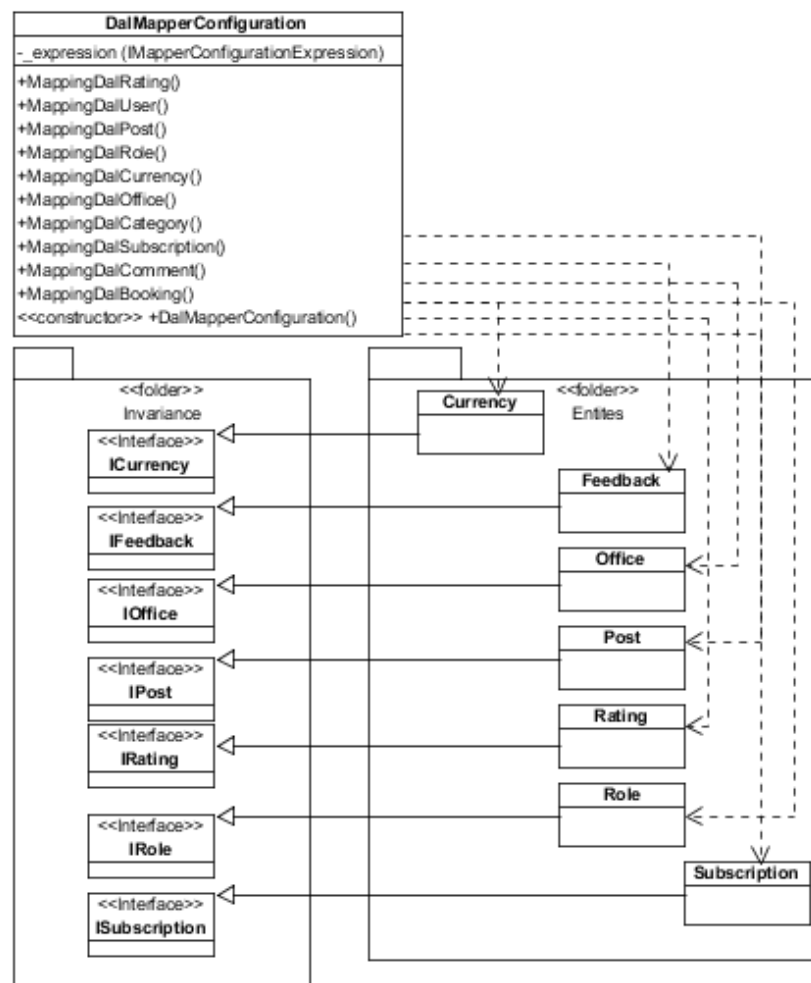


Рисунок 16. Классовая диаграмма Domain уровня архитектуры (2 часть)

На примере двух интерфейсов и двух моделей было рассмотрена логика построения классов на данном уровне.

Абсолютно все модели на этом уровне «мапятся» по своим полям с моделями DAL уровня, поэтому их описание можно найти на Рис.8-11. Единственное отличие – это наличие методов написанные по паттерну With, который позволяет создавать экземпляр того или иного класса, внутри другого класса без применения наследования, это нужно для того, чтобы максимально разграничивать каждый слой. Говоря более простым языком, этот паттерн позволяет нам на уровне бизнес логики, не создавать отдельные экземпляры классов, а просто обращаться к методу той или иной модели. Работает он на основе метода MemberwiseClone – позволяющий копировать объект.

Но также стоит отметить и минусы этого паттерна, а именно - он работает с помощью механизма Reflection. Который достаточно сильно замедляет быстроедействие системы.

На Рис.15,16 так же указано взаимодействие с Автомаппером, он напрямую зависит от классов Entities, именно поэтому и показан с помощью связи Dependency. Изменив какой-либо класс, изменится работа маппера, он выдаст другую сущность, которая пробросится дальше.

2.4.3 Bll Application-уровень и API уровень

Bll Application-уровень - представляет из себя описание всех сервисов– бизнес логики. На этом уровне происходят все вычисления, сортировки, обращения к репозиториям.

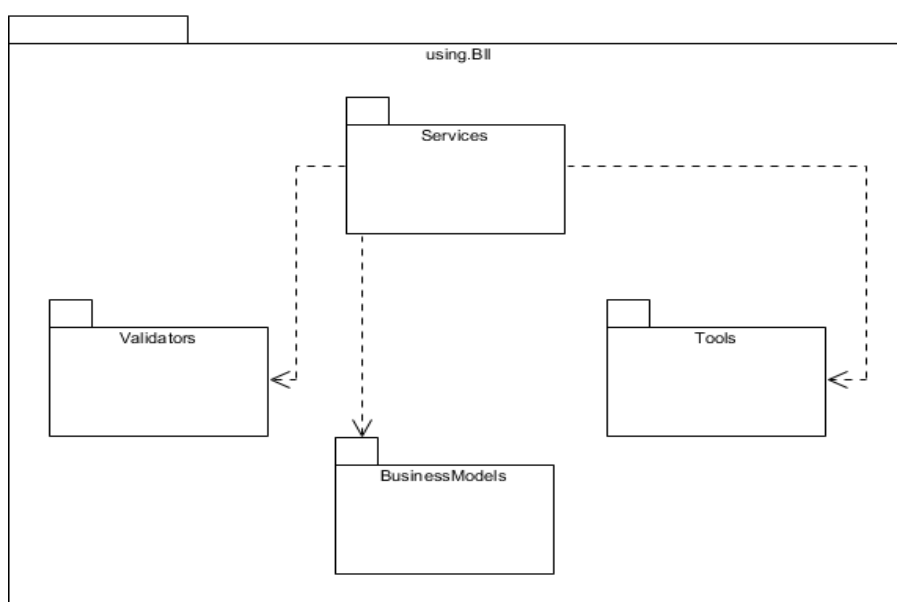


Рисунок 17. Bll Application-уровень архитектуры

Tools и Validators выполняют тривиальную работу, простейшие валидации данных либо, классы расширения для перевода в строковое представление или же формирования даты.

Services – папка в которой описаны классы по работе с бизнес логикой, а так же репозиториями. Они задействуют интерфейсы из Domain уровня.

Business Models – содержит в себе дополнительные модели, которые не выходят за уровень Bll.

Application – папка – песочница, хранит в себе классы, которые нужны в основном для тестов.

API уровень – содержит в себе все контроллеры, а так же конфигурацию AutoMapper-a, инициализацию IoC контейнера, файлы отвечающие за авторизацию, описание обработчика ошибок.

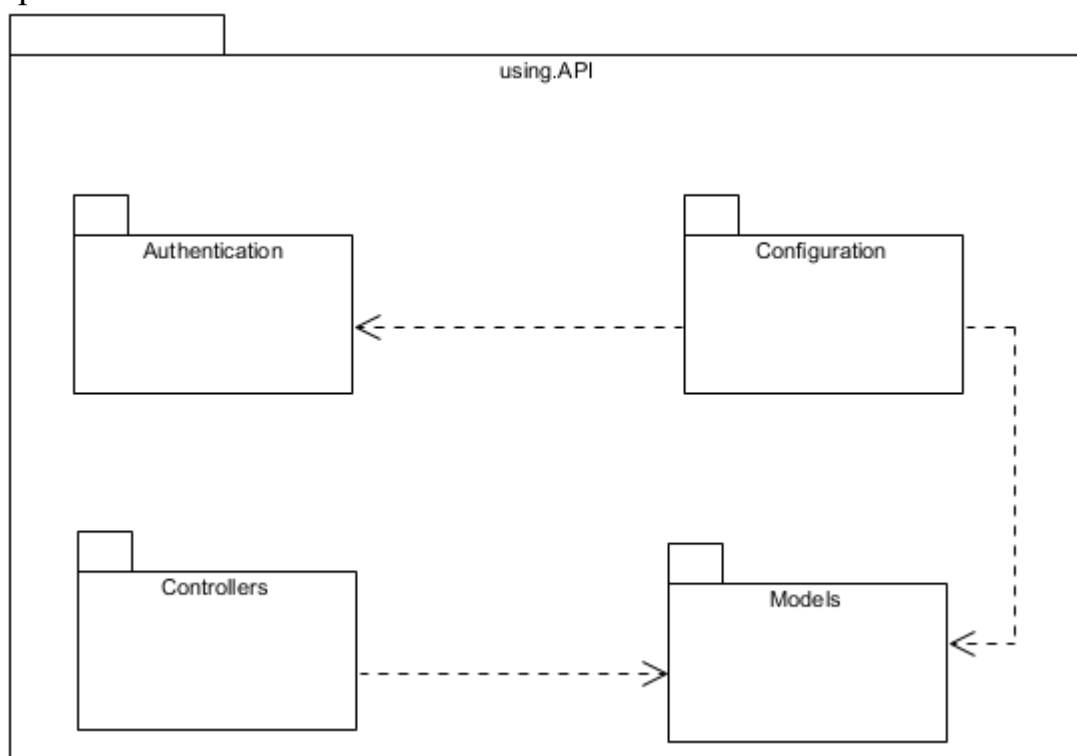


Рисунок 18. Api-уровень архитектуры

Это самый высокий уровень архитектуры, здесь описаны контроллеры, которые и являются начальным звеном в цепочке вызовов, а также модели, которые уже отличаются от тех, что были представлены на уровне Domain тем, что это модели, которые, получает клиент по Get запросам, либо же отправляет их по POST запросам. Данные модели разграничены на Input и Output модели. Они используют такой же MapperConfigurator, который использовался на Domain уровне, только он уже занимается пробросом Input/Output модели, в модель низшего уровня, которые описаны в папке Entities.

Папка Authentication содержит в себе классы по работе с Google Authentication. В данном пункте было охвачено сразу два уровня т.к. гораздо нагляднее прослеживается цепочка работы.

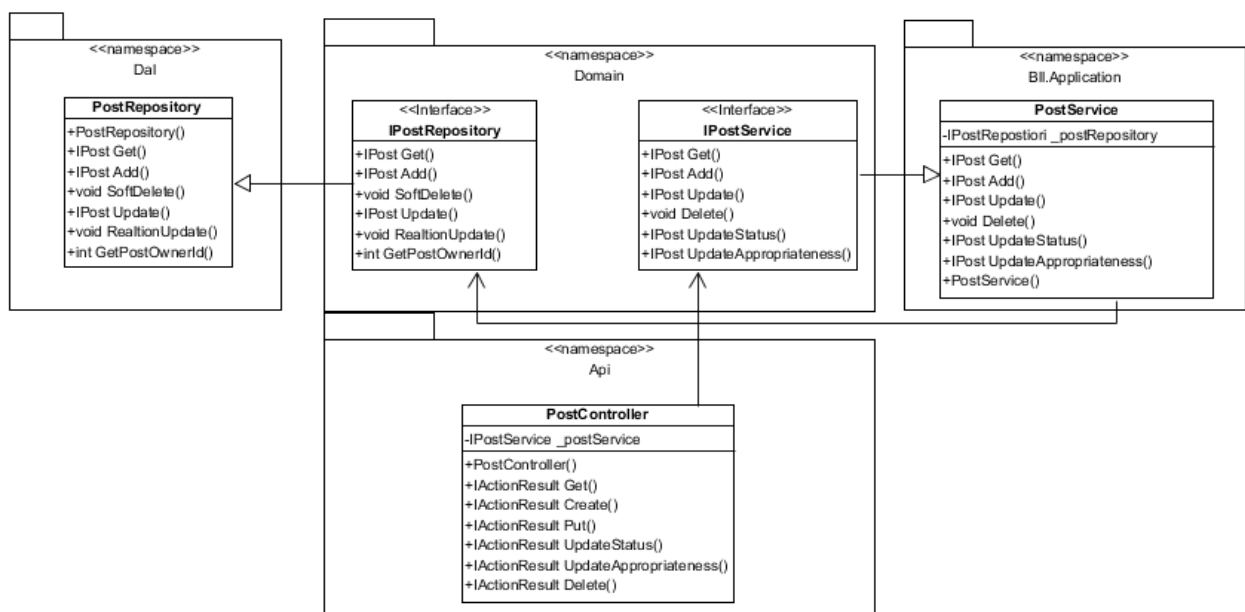


Рисунок 19. Описание взаимоотношений между контроллером, репозиторием, сервисом - Post модели

На Рис.19 изображена классовая диаграмма, которая наглядно показывает, как связаны различные уровни, а также какой класс вызывает другой.

Все начинается с PostController-а, когда вызывается какой-то метод, мы обращаемся в PostService через интерфейс IPostService. Такой подход называется Dependency Injection. Мы используем методы того или иного класса, через интерфейс, который реализует этот же класс. Таким образом соблюдается правило инкапсуляции.

PostService в свою же очередь использует PostRepository тем же путем, что мы использовали и в Service.

Таким образом, была показана цепочка наследования и реализации классов друг за другом.

2.4.3 Проектирование архитектуры данных

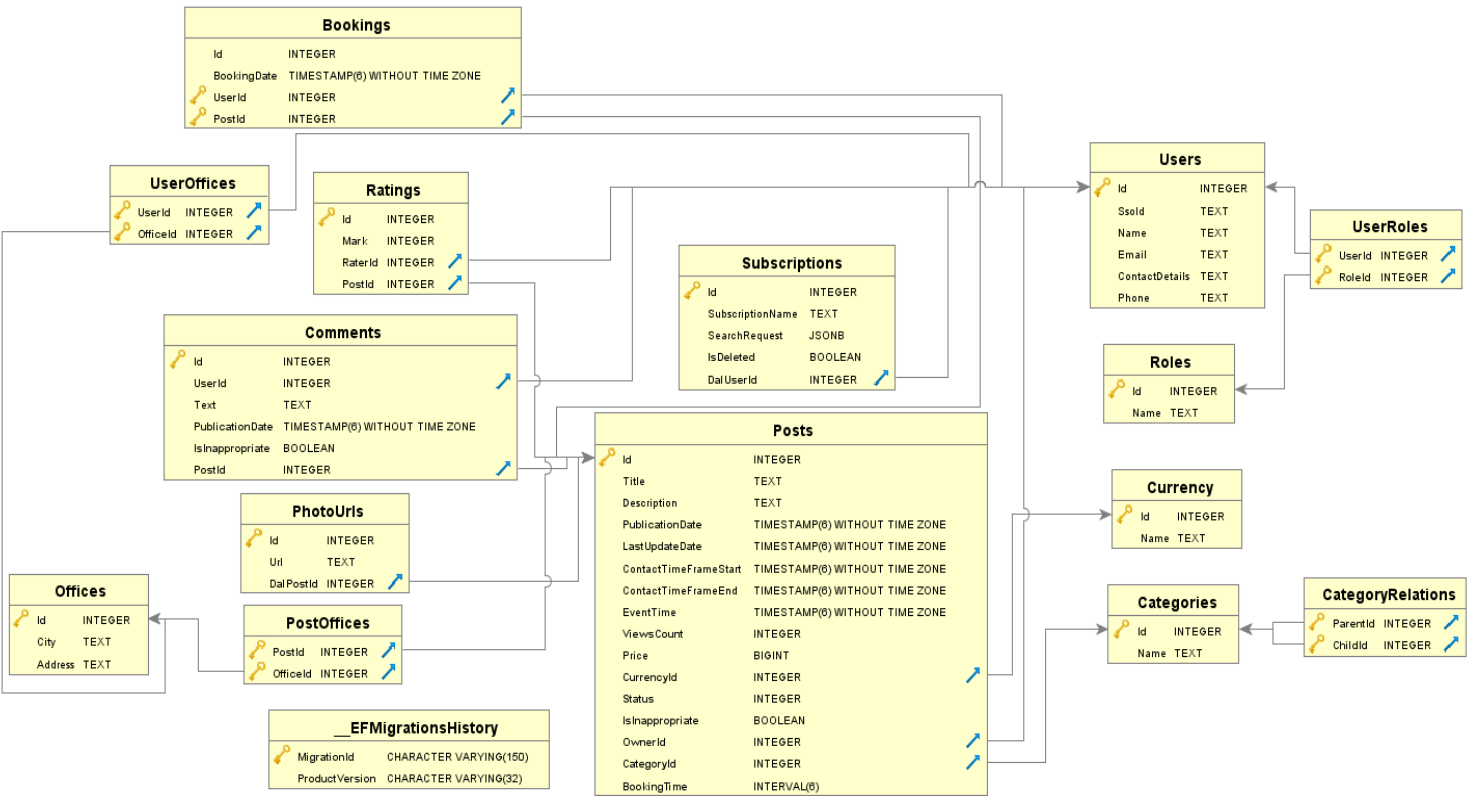


Рисунок 20. ER диаграмма архитектуры данных

Для данного приложения была выбрана реляционная модель базы данных. Такой выбор обусловлен большим количеством зависимостей между таблицами. Исходя из Рис.20 можно заметить, что одной из самых «завязанных» таблиц является Posts таблица.

Схема базы данных, как и сказано было выше, генерируется с помощью ORM, отсюда следует то, что она полностью совпадает с описание классов из главы о проектировании программной архитектуры.

Имя таблицы	Описание	Зависимость	Тип отношения
Bookings	Содержит информацию о дате бронирования	Posts Users	Один к одному Один ко многим
UserOffices	Составная таблица для таблиц User и Offices	Users Offices	Один ко многим
Ratings	Содержит информацию о оценке для постов/пользователей	Posts Users	Один к одному

Offices	Содержит информацию о адресе и городе, в котором находится офис	UserOffices	Один ко многим
Comments	Содержит текст комментария, дату публикации и фильтр модерирования	Users Posts	Один ко многим Многие ко многим
PhotoUrls	Содержит информацию о Url фотографии поста	Posts	Многие ко многим
PostOffices	Содержит информацию офисе прикрепленному к посту	Offices Post	Один ко многим
Subscriptions	Содержит информацию о имени подписке, применённом фильтре, удалении.	Users	Один ко многим
Users	Содержит информацию о авторизации, имени, е-мейле, контактных данных, телефоне		
UserRoles	Составная таблица для таблиц User и Roles	Users Roles	Один ко многим
Roles	Содержит информацию о роли пользователей		
Currency	Содержит информацию о виде валюты		
Categories	Содержит информацию о имени категории		
CategoryRelations	Составная таблица для категорий	Categories	Многие ко многим
__EFMigrations	Содержит информацию для EF		
Posts	Содержит информацию о названии поста, описании, даты публикации, даты последнего обновления, диапазон удобного времени, количество просмотров, цену, статус, фильтр модерирования, время бронирования	Categories Users Currency	Один ко многим Многие ко многим Один к одному

Таблица 1. Описание архитектуры данных

Глава 3. Программная реализация

3.1 Реализация DAL и Domain уровней

DAL слой разрабатывался в соответствии с проектированием в главе 2. Т.к. он ничто иное как проекция на базу данных, то и реализация БД сущностей выглядит соответствующим образом. На Рис.21 можно увидеть реализацию одной из моделей.

```
8 references
public class DalSubscription
{
    [Key]
    1 reference
    public int Id { get; set; }

    0 references
    public string SubscriptionName { get; set; }

    [Column(TypeName = "jsonb")]
    2 references
    public string SearchRequest { get; set; }

    1 reference
    public bool IsDeleted { get; set; }

    //ONE DalUser has MANY DalSubscriptions
    2 references
    public int DalUserId { get; set; }
    1 reference
    public DalUser DalUser { get; set; }
}
```

Рисунок 21. DalSubscription модель

С помощью атрибутов задается первичный ключ, и исключением является поле JSONb т.к. EF не может самостоятельно спроецировать это поле на базу данных, но с помощью данного ключа обычная строка будет передаваться с базу с пометкой, что это JSONb. В остальном всю работу на себя берёт Entity Framework. Он сам может создать миграцию на основе программных типов, проецируя их на типы в PostgreSQL. Саму миграцию можно увидеть на Рис.22.

```

migrationBuilder.CreateTable(
    name: "Subscriptions",
    columns: table => new
    {
        Id = table.Column<int>(nullable: false)
            .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.SerialColumn),
        SubscriptionName = table.Column<string>(nullable: true),
        SearchRequest = table.Column<string>(type: "jsonb", nullable: true),
        IsDeleted = table.Column<bool>(nullable: false),
        DalUserId = table.Column<int>(nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Subscriptions", x => x.Id);
        table.ForeignKey(
            name: "FK_Subscriptions_Users_DalUserId",
            column: x => x.DalUserId,
            principalTable: "Users",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

```

Рисунок 22. Миграция основанная на DalSubscription

Так же в DAL уровне находятся Репозитории, классы, которые реализуют CRUD операции с помощью LINQ to EF запросов. Пример можно увидеть на Рис. 23. На нем продемонстрированы только две операции, Create и Delete. Все довольно-таки тривиально, метод Map вызывает автомашпер, а в качестве аргумента передается результат запроса из контекста.

```

1 subscription
public ISubscription Create(ISubscription subscription)
{
    subscription = subscription ?? throw new ArgumentNullException(nameof(subscription));
    var retVal = _context.Subscriptions.Add(_mapper.Map<DalSubscription>(subscription));
    _context.SaveChanges();

    return _mapper.Map<Subscription>(retVal.Entity);
}

1 reference
public bool Delete(int id)
{
    var subscription = new DalSubscription { Id = id };
    var entity = _context.Subscriptions.Remove(subscription);
    if (entity == null)
    {
        return false;
    }
    _context.SaveChanges();

    return true;
}

```

Рисунок 23. Subscription репозиторий

Domain уровень представляет из себя описание всех интерфейсов, которые задействованы в системе, а так же бизнес сущности. Это один из самых простых слоев архитектуры, и он так же, как и все остальные, был реализован в соответствии с проектированием. На Рис. 24 показан пример бизнес сущности подписчика.

Как можно заметить, поля совпадают с тем, что было описано на Dal уровне, отличием являются лишь сложные поля, например, Categories, которые достаются по итогу с помощью метода, позволяющего клонировать объект.

```

10 references
public class Subscription : ISubscription
{
    2 references
    public int Id { get; set; }

    3 references
    public int OwnerId { get; set; }

    0 references
    public string SubscriptionName { get; set; }

    4 references
    public IReadOnlyList<SubscriptionCategory> Categories { get; set; }

    4 references
    public ISubscriptionRequest SearchRequest { get; set; }

    1 reference
    public ISubscription WithOwner(IUser user)
    {
        user = user ?? throw new ArgumentNullException(nameof(user), "subscription owner can't be null!");
        var clone = (Subscription)MemberwiseClone();
        clone.OwnerId = user.Id;
        return clone;
    }

    2 references
    public ISubscription WithCategory(IReadOnlyList<SubscriptionCategory> category)
    {
        category = category ?? throw new ArgumentNullException(nameof(category), "category can't be null!");
        var clone = (Subscription) MemberwiseClone();
        clone.Categories = category;
        return clone;
    }
}

```

Рисунок 24. Subscription бизнес сущность

3.2 Реализация API и BLL уровней

На Рис.25 показана основная последовательность любого компонента из главы 2. Любой запрос GET/POST/DELETE/PUT будет выглядеть с точки зрения диаграммы последовательности именно так, за исключением разве что, названий. Диаграмма не учитывает исключительные ситуации, т.к. это бы существенно её усложнило. Большую часть валидаций покрывает .NET Core Framework, т.к. в новой версии

можно использовать [ApiController] атрибут для того, чтобы возложить проверку на NULL, CLR среде.

Начинается всё с того, что клиентская часть приложения генерирует запрос на тот или иной Controller (1), тот же в свою очередь перенаправляет запрос в сервис, где происходит вся бизнес логика (2). В данном случае, запрос отправится дальше в репозиторий, т.к. нужно получить данные из базы (3). После пункта 3 сформируется DbEntity которая будет содержать в себе нужные данные (4), затем она отправится в сервис для дальнейшего маппинга в Entity или так называемую бизнес-сущность. Она же в свою очередь отправится в контроллер, где сматится в OutEntity (выходная сущность) её и получает Client часть приложения, а также сообщение о том, что всё прошло успешно.

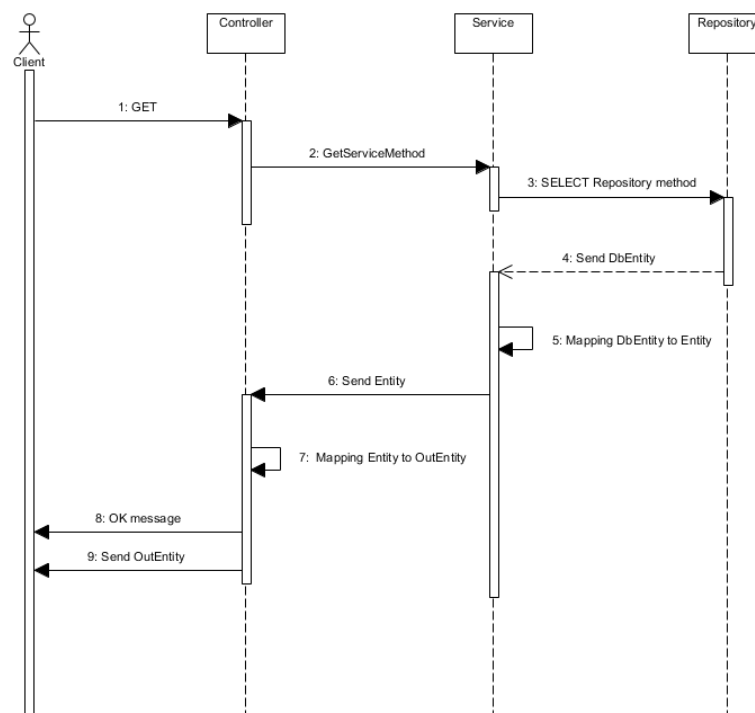


Рисунок 25. Диаграмма последовательности основных компонентов

Главной частью приложения является уровни Bll и API. Их реализация показана на Рис. 26 на примере модуля Subscription. На нем можно увидеть атрибут, который обозначает непосредственный путь до этого контроллера, так же атрибут, который проверяет наличие авторизации и ApiController о котором говорилось ранее.

Контроллеры были написаны в точности с планом разработки, описанным в главе 2. Можно заметить ApiInputSubscription модель, которая используется для проекции в бизнес сущность, а затем уже в сущность базы данных.

Сами по себе контроллеры используют те или иные сервисы, доступ к которым происходит через объявленный интерфейс, который, в свою очередь регистрируется в IoC контейнере как показано на Рис. 27

```

[Route("api/Subscriptions")]
[Authorize]
[ApiController]
1 reference
public class SubscriptionsController : Controller
{
    5 references
    private readonly ISubscriptionService _subscriptionService;
    5 references
    private readonly IMapper _mapper;
    1 reference
    private readonly ILogger _logger;

    0 references
    public SubscriptionsController(ISubscriptionService subscriptionService, ILogger<SubscriptionsController> logger, IMapper mapper)
    {
        _subscriptionService = subscriptionService;
        _logger = logger;
        _mapper = mapper;
    }

    [HttpPost]
    0 references
    public IActionResult Create([FromBody]ApiInputSubscription subscription)
    {
        var retVal = _subscriptionService.Add(_mapper.Map<Subscription>(subscription));
        if(retVal == null)
        {
            return StatusCode(500);
        }
        return Created($"api/Subscriptions/{retVal.Id}", _mapper.Map<ApiOutSubscription>(retVal));
    }
}

```

Рисунок 26. Часть реализации механизма подписки на API уровне

После описания работы контроллеров, а также их доступа к сервисам, можно перейти к описанию реализации сервисов, находящихся в Vll уровне архитектуры.

Как и было заявлено в проектировании, сервисы используют репозитории для общения и взаимодействия с базой. Частой практикой является использование паттерна With, который позволяет создавать экземпляр того или иного класса, внутри другого класса без применения наследования или же, говоря проще, паттерн позволяет не создавать экземпляр класса на уровне бизнес логики – а вместо этого, просто обращаться к методу той или иной модели.

На Рис.28 можно увидеть реализацию двух методов сервиса Subscriptions.


```

0 references
public static class BllServicesRegistrar
{
    1 reference
    public static void AddBll(this IServiceCollection services)
    {
        services.AddTransient<IBinaryDataService, BinaryDataService>();
        services.AddTransient<IBookingService, BookingService>();
        services.AddTransient<ICategoryService, CategoryService>();
        services.AddTransient<ICommentService, CommentService>();
        services.AddTransient<ICurrencyService, CurrencyService>();
        services.AddTransient<IFeedbackService, FeedbackService>();
        services.AddTransient<IOfficeService, OfficeService>();
        services.AddTransient<IPermissionChecker, PermissionChecker>();
        services.AddTransient<IPostService, PostService>();
        services.AddTransient<IRoleService, RoleService>();
        services.AddTransient<ISearchService, SearchService>();
        services.AddTransient<ISubscriptionService, SubscriptionService>();
        services.AddTransient<IPermissionChecker, PermissionChecker>();
        services.AddTransient<IBookingService, BookingService>();
        services.AddTransient<IRatingService, RatingService>();
        services.AddTransient<IUserService, UserService>();

        services.AddTransient<FilterMaker>();
        services.AddTransient<PostValidator>();
        services.AddTransient<UserValidator>();

        services.AddSingleton<IDateTimeProvider, DateTimeProvider>();
    }
}

```

Рисунок 27. Регистрация сервисов в ЮС контейнер

```

2 references
class SubscriptionService : ISubscriptionService
{
    1 reference
    private readonly ILogger _logger;
    5 references
    private readonly ISubscriptionRepository _subscriptionRepository;
    2 references
    private readonly IUserService _userService;
    3 references
    private readonly ICategoryService _categoryService;

    0 references
    public SubscriptionService(ILogger<SubscriptionService> logger, ISubscriptionRepository repository,
        IUserService userService, ICategoryService categoryService)
    {
        _logger = logger;
        _subscriptionRepository = repository;
        _userService = userService;
        _categoryService = categoryService;
    }

    1 reference
    public ISubscription Add(ISubscription subscription)
    {
        if (subscription.IsNullOrEmpty())
        {
            return null;
        }

        var retSubs = subscription.WithOwner(_userService.GetCurrentUser());
        return _subscriptionRepository.Create(retSubs);
    }

    1 reference
    public ISubscription Get(int id)
    {
        var getSubscription = _subscriptionRepository.Get(id);
        var retSubs = getSubscription.WithCategory(_categoryService.GetCategoryNameAndIdsPair(id));
        return retSubs;
    }
}

```

Рисунок 28. Subscription сервис

3.1 Реализация OAuth авторизации

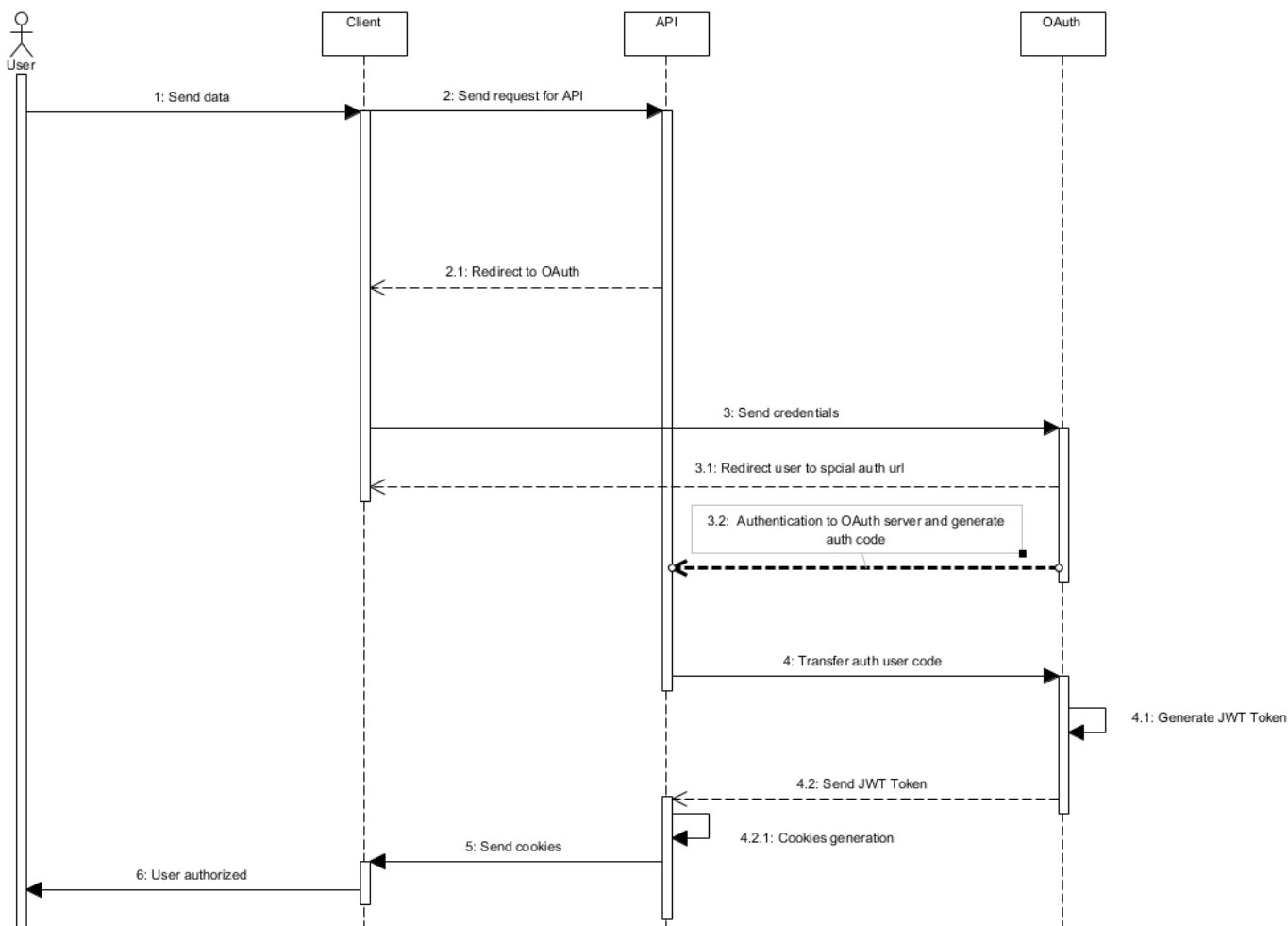


Рисунок 29. Диаграмма последовательности OAuth авторизации

На Рис.29 показан последовательность исключительно в случае успешного выполнения, провал на любом этапе ведёт к единственному результату – пользователь не авторизован.

Рис.29 не показывает подробного механизма генерации JWT токена, а также валидаций.

Обусловимся, что API – это сервер нашего Web-приложения, OAuth – сервер авторизации.

Client – браузер User и всё, что он видит на своем экране.

1. Пользователь нажимает кнопку на клиенте (1). Клиент формирует URL вида: `<сервис_авторизации>?response_type=code&scope=openid&clientId=<ид клиента>&redirectUrl=<redirectUrl>`

2. После чего клиент посылает HTTP заголовок Location с этим URL, о котором говорилось выше (2).

3. Браузер пользователя производит редирект на OAuth сервер (2.1).

4. Сервис авторизации запрашивает логин и пароль пользователя, а также разрешения, готов ли он дать доступ приложения клиента на свои данные (3).

5. Сервис редиректит браузер клиента на
<redirectUrl>?code=<AuthorizationCode> (3.1)

6. OAuth отдает авторизационный код backend части приложения (3.2).

7. Backend часть приложения читает параметр code полученный из запроса (полученном из предыдущего пункта) и после чего делает запрос на token_endpoint.
<token_endpoint>?code=<AuthorizationCode>&clientId=<clientId>&clientSecret=<ClientSecret>

8. Получает в ответ JSON подобного вида (4.1):

```
{  
  access_token: xxcvasdfasfdasdf,  
  id_token: avasdfasfdasdfa,  
  ...  
}
```

id_token – является JWT.

JWT состоит из трёх частей разделённых точкой
base64url(header).base64url(payload).base64url(sign)

9. Из 8 этапа берётся payload. Который в свою очередь выглядит следующим образом:

```
{  
  sub: asdfasdfasdf  
  aud: addfasdfasdf  
  iss: asdfasdf  
}
```

Здесь мы проверяем наш aud с полученным client_id, а также что iss – это адрес авторизационного сервера. (4.2)

10. Теперь мы готовы отправить сгенерированные с JWT Cookies на сторону клиента

11. Пользователь видит у себя успешно выполненную авторизацию (5,6)

В промежуток между 9 и 10 этапом, есть ещё одна вещь, о которой нужно сказать. Это проверка подлинности подписи(sign) в JWT токене.

И так, механизм проверки подписи заключается в том, что сначала нужно раскодировать первую часть JWT токена, а именно base64url(header), тогда мы получим JSON вида

```
{  
  typ: "JWT"  
  alg: "RS256"  
}
```

В данном случае алгоритмом шифрования подписи является RS256, а это значит, что подпись формировалась следующим образом: base64url(header) + "." +

base64url(payload), от этого получается хэш SHA256, потом хэш шифруется приватным ключом, а после ещё раз шифруется полученное в base64url.

Теперь, после того как был описан механизм шифрования, можно рассмотреть проверку подписи.

1. Получаем id_token, который был описан в 8 пункте. Из него выборочно забираем всё что идёт до второй точки, а затем используем алгоритм SHA256 для получения хэша. То есть

SHA256(base64url(header).base64url(payload)

2. После чего берётся base64url(sign), раскодируется с помощью base64urlDecode, применяется публичный ключ и наконец, мы можем получить SHA256 хэш, который нам нужно сравнить с хэшем из 1 пункта. Если они совпадают – значит всё верно, иначе, пользователь не авторизован.

Публичный ключ в случае с Google аутентификацией лежит в Document Discovery, по адресу:

accounts.google.com/.well-known/openid-configuration.

Глава 4. Тестирование

Существует несколько подходов к тестированию системы, в основном они делятся на ручное и автоматическое.

К ручному типу относятся такие тесты как, Manual Tests – самое дорогое тестирование с точки зрения бизнеса, т.к. выполняется людьми в ручном режиме. Тестировщики вручную проверяют корректность работы бизнес-кейсов, а также создают записи о багах для разработчиков. В данный момент времени многие компании стараются переходить на автоматическое тестирование, т.к. это позволяет избежать больших затрат на проверку одних и тех же мест системы, т.к. автоматические тесты могут повторяться раз за разом после каждого изменения приложения.

Автоматическое тестирование делится на несколько типов:

1. UI Tests – автоматическое тестирование UI компонентов. По сути, эмуляция Manual Tests. Одни из самых «тяжеловесных» тестовых этапов.

2. API Tests – отличительной особенностью этого вида тестирования является то, что тестировщик должен симулировать среду при котором тот или иной API работает, после чего тестируется основная бизнес-логика. Отличается от прошлого тем, что тесты не работают непосредственно с UI компонентами.

3. Integration Tests – тесты, которые пишут непосредственно разработчики, они проверяют любую сложную логику происходящую на стороне backend части

приложения. Например, проверяют корректность тех или иных транзакций, или же правильную обработку того или иного компонента

4. Unit Tests – тесты, которые пишут разработчики для проверки почти любой логики происходящей в приложении, например: маппинг сущностей, калькуляция того или иного свойства/поля, правильность работы сервисов.

Unit Tests и Integration Tests очень похожи между собой по смыслу выполнения, в основном они отличаются лишь тем, что Unit Tests работают с эмулированной средой и они, чаще всего, изолированы и не зависят никак от окружения. Integrations Tests же абсолютно наоборот, зависят от окружения и чаще всего работают с тестовой базой данных. Они напрямую обращаются к тем или иным таблицам, поэтому выполняются гораздо дольше чем Unit.

В данной работе будут рассматриваться только Unit Tests. Интеграционное тестирование не было осуществлено в данном приложении, т.к. было решено не использовать тестовую базу данных.

Для тестирования приложения был создан отдельный проект внутри архитектуры, в котором и были написаны все нужные тесты.

Так как в приложении достаточно малое количество какой-либо бизнес логики, которая требует калькуляции, вычислений, взаимосвязи с другими сервисами, всего было написано 20 юнит тестов.

На Рис.30 показаны 4 секции юнит тестов:

1. AutoMapperUnitTests – секция в которой проверяется правильный маппинг всех сущностей что есть в проекте. Так как используется AutoMapper Framework для этих целей, сделать подобные тесты не составило труда и не пришлось поднимать окружение, либо же эмулировать его для корректной проверки маппинга сущностей.

2. BinaryDataRepositoryUnitTests – секция которая отвечает за то, что бы корректно проверить работу с бинари файлами.

3. CategoryServiceUnitTests – секция которая отвечает за тестирование работы категорий. Так как категории представляют из себя сложную древовидную структуру, были написаны тесты, которые позволяют тестировать правильность получения той или иной категории.

4. FilterMakerUnitTests – секция которая отвечает за тестирование работы фильтра. Проверяются общие кейсы использования фильтров на уровне backend части приложения.

Исходя из всего вышеперечисленного можно прийти к выводу, что основные компоненты системы с точки зрения разработчика были протестированы.

▲	✓	Ebb.Core.Tests (20)	717 ms
▲	✓	Ebb.Core.Tests (20)	717 ms
▲	✓	AutoMapperUnitTests (2)	462 ms
	✓	ApiMapperConfigurationTest	361 ms
	✓	DalMapperConfigurationTest	101 ms
▲	✓	BinaryDataRepositoryUnitTests (4)	52 ms
	✓	BinaryDataRepositoryDeleteInputFilenameNullShouldFail	46 ms
	✓	BinaryDataRepositoryDownloadInputFilenameNullShoul...	3 ms
	✓	BinaryDataRepositoryUploadNoDataShouldFail	2 ms
	✓	BinaryDataRepositoryUploadNoFiletypeShouldFail	1 ms
▲	✓	CategoryServiceUnitTests (5)	156 ms
	✓	GetCategoryNameAndIdsPair_FourInARow_UnitTest	156 ms
	✓	GetCategoryNameAndIdsPair_RootChild_UnitTest	< 1 ms
	✓	GetCategoryNameAndIdsPair_RootObject_UnitTest	< 1 ms
	✓	GetCategoryNameAndIdsPair_ThreeInARow_UnitTest	< 1 ms
	✓	GetCategoryNameAndIdsPair_TwoInARow_UnitTest	< 1 ms
▲	✓	FilterMakerUnitTests (9)	47 ms
	✓	CreateBookingSpecification_Custom_UnitTest	33 ms
	✓	CreatePostSpecification_CategoryId_UnitTest	8 ms
	✓	CreatePostSpecification_Custom_UnitTest	1 ms
	✓	CreatePostSpecification_EventRange_UnitTest	1 ms
	✓	CreatePostSpecification_MitPrice_UnitTest	1 ms
	✓	CreatePostSpecification_NullQuery_UnitTest	< 1 ms
	✓	CreatePostSpecification_Photo_UnitTest	1 ms
	✓	CreatePostSpecification_PriceRange_UnitTest	1 ms
	✓	CreatePostSpecification_Text_UnitTest	1 ms

Рисунок 30. Список Unit тестов

Заключение

В рамках данной работы был проанализирован процесс выставления объявлений. Было найдено решение по автоматизации данного процесса.

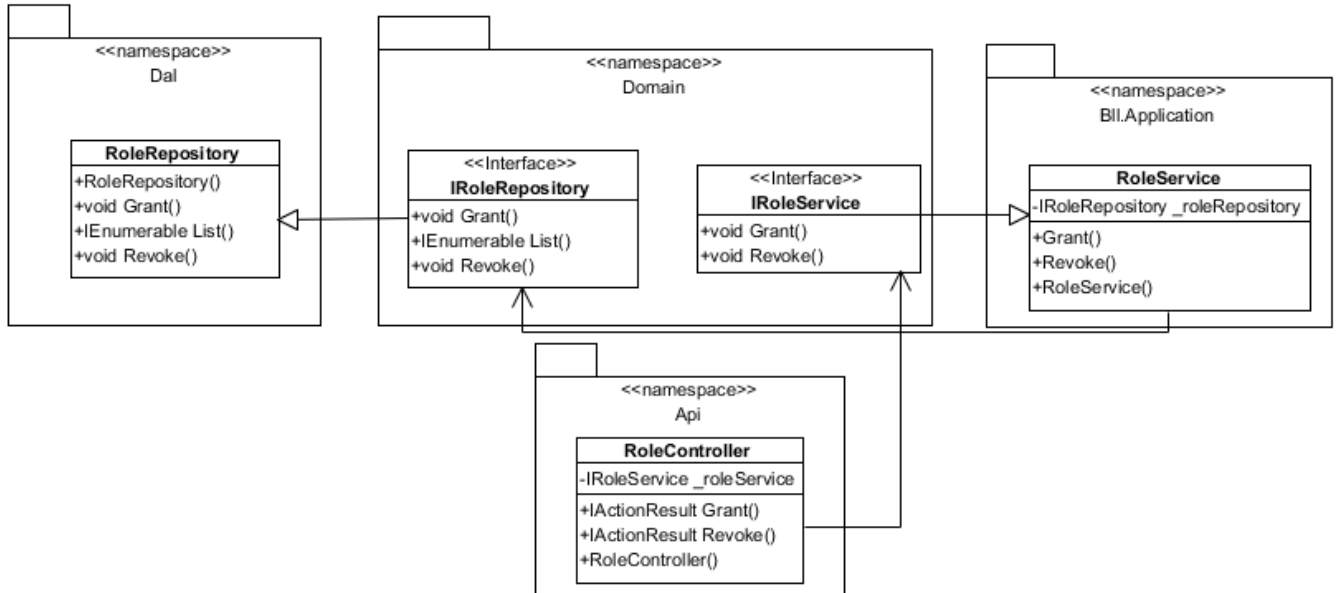
Все функциональные требования были выполнены.

Результатом данной работы является одностраничное веб-приложение, позволяющее автоматизировать процесс подачи объявлений, а также их комментирования, сортировки, фильтрации, поиска. Приложение упрощает подачу объявлений и взаимодействие пользователей друг с другом.

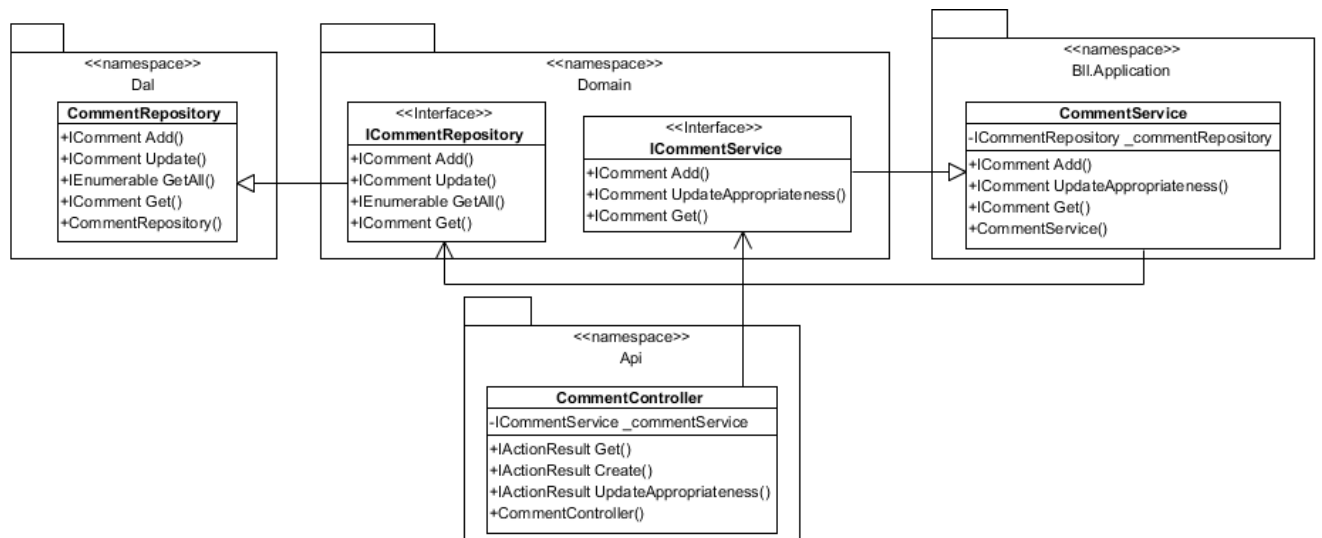
Список используемых источников

1. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C# [Текст]
2. C# 7.0. Справочник. Полное описание языка [Текст]
3. DotNet documentation [Электронный ресурс]. URL:
<https://docs.microsoft.com/en-us/dotnet/>

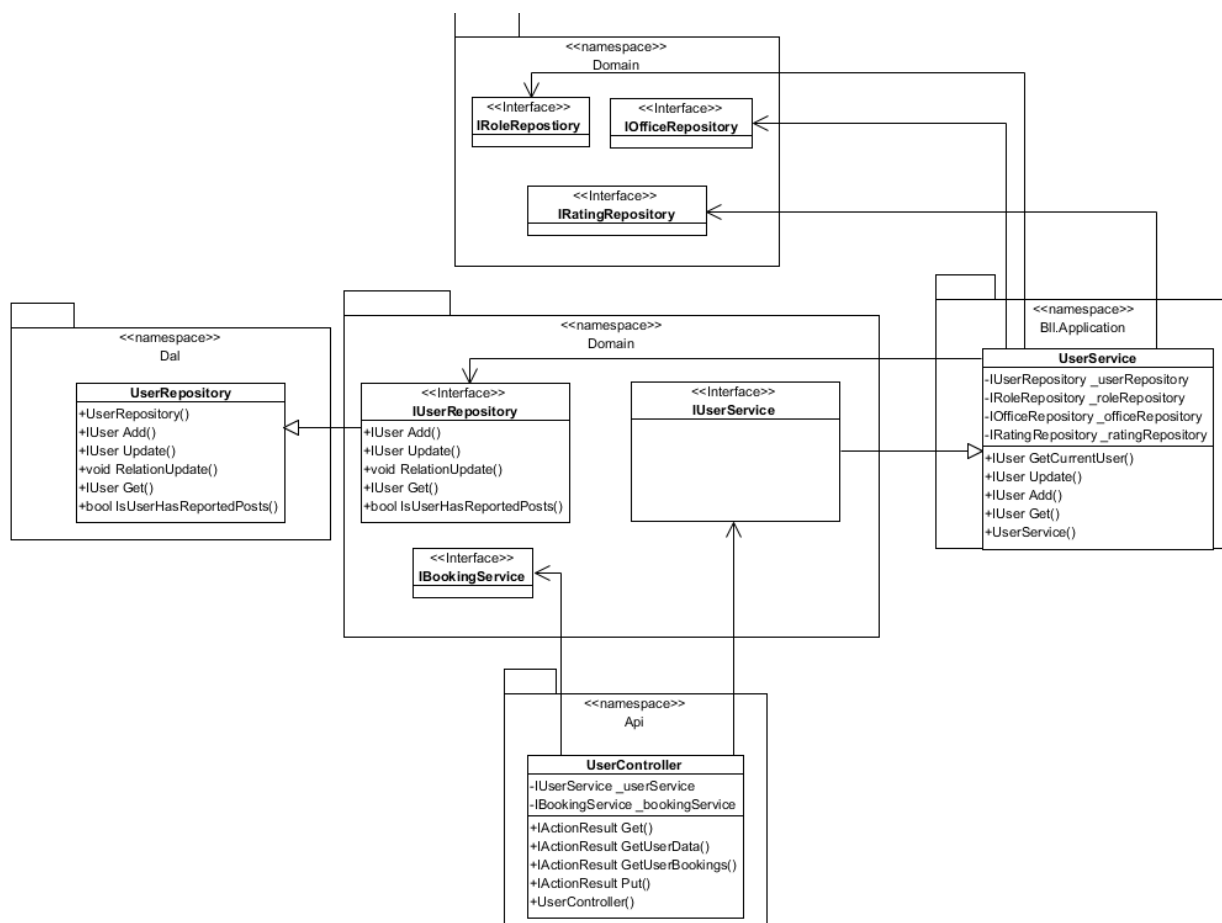
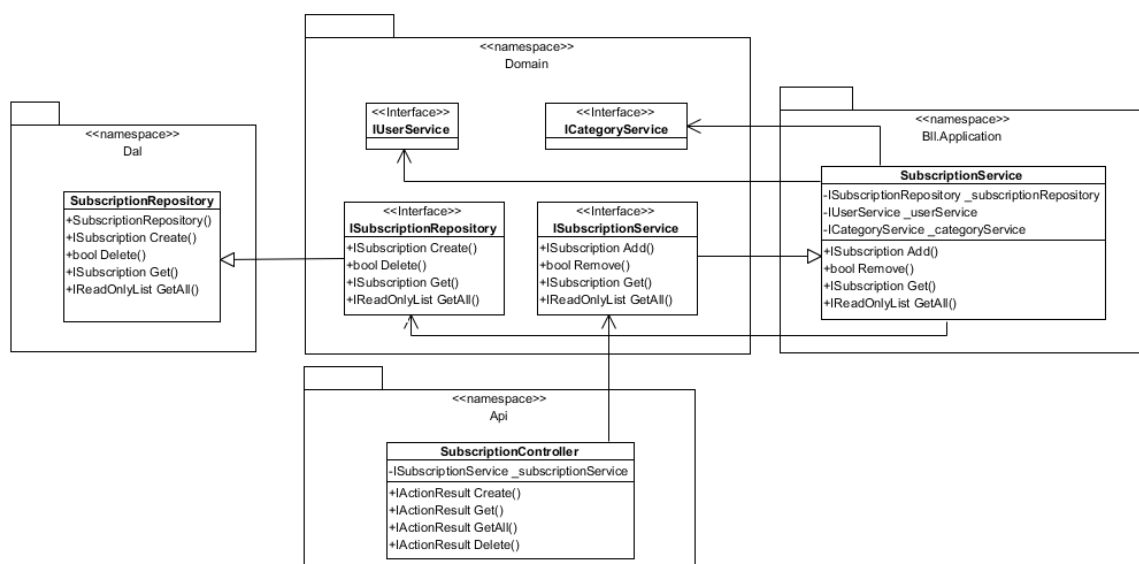
Приложение А. Диаграммы Bll и API уровней

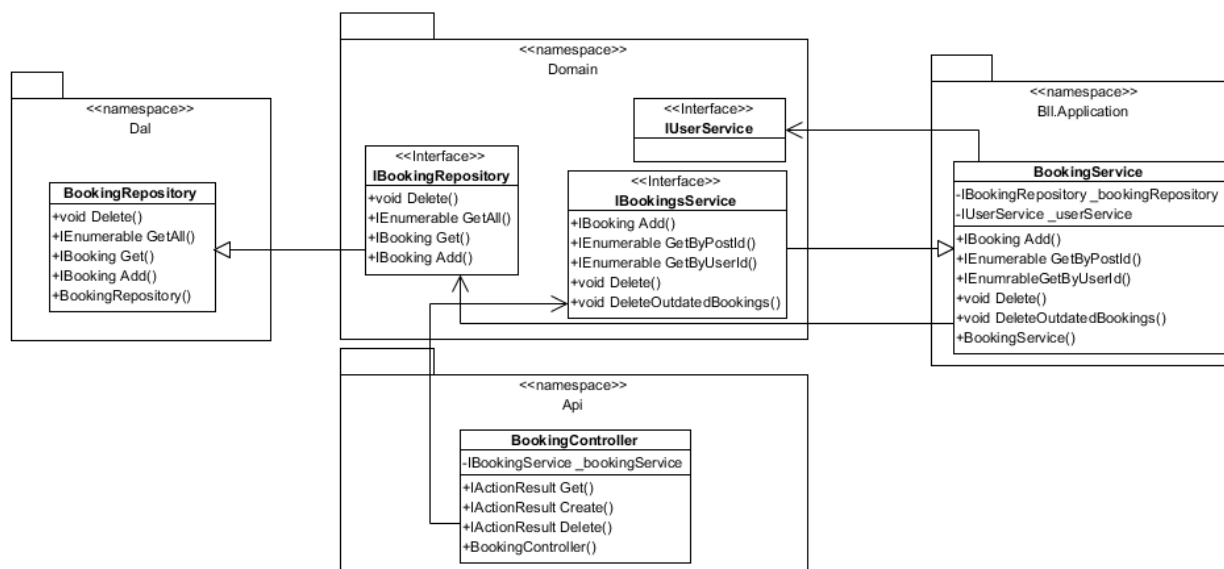


Приложение 1. Описание взаимоотношений между контроллером, репозиторием, сервисом Role модели

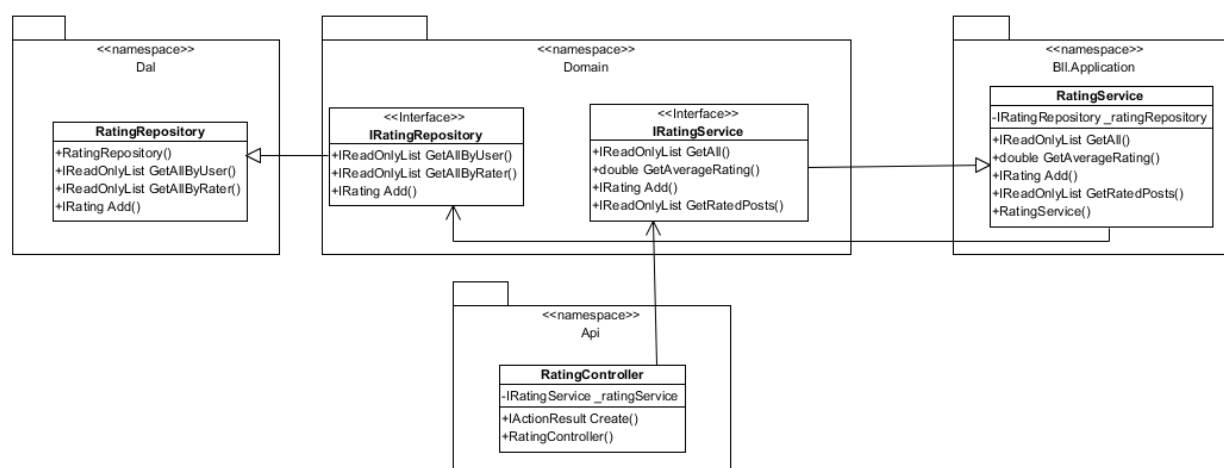


Приложение 2. Описание взаимоотношений между контроллером, репозиторием, сервисом Comment модели

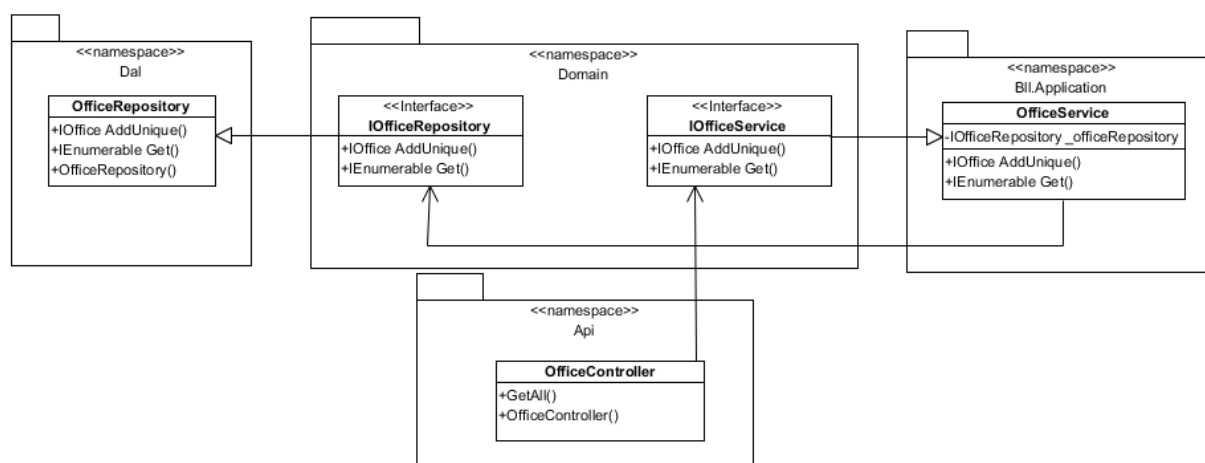




Приложение 5. Описание взаимоотношений между контроллером, репозиторием, сервисом Booking модели



Приложение 6. Описание взаимоотношений между контроллером, репозиторием, сервисом Rating модели



Приложение 7. Описание взаимоотношений между контроллером, репозиторием, сервисом Office модели