Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

Рязанский государственный радиотехнический университет  
имени В.Ф. Уткина

Кафедра Автоматизированных Систем Управления

К защите

Руководитель работы:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ**

по дисциплине

**«Технологии программирования»**

Тема:

«Разработка информационно-поисковой системы для предметной области:

«Прием в поликлинике»

Выполнил студент группы 135

Сидорин Д.Г.

дата сдачи на проверку, подпись

Руководитель работы

Преподаватель каф. АСУ

Аникеев Д.В.

оценка дата защиты, подпись

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

Рязанский государственный радиотехнический университет

им.В.Ф.Уткина

Кафедра Автоматизированных Систем Управления

З А Д А Н И Е

на курсовую работу по дисциплине

"Технологии программирования"

Студент — Сидорин Даниил Геннадьевич группа – 135

1. Тема: Разработка информационно-поисковой системы «Прием в поликлинике»

2. Срок представления работы к защите - « » 2023г.

3. Исходные данные для проектирования:

* 1. Язык программирования – C#.
  2. СУБД — MS SQL Compact Edition, MS SQL Express Edition
  3. Технология объектного связывания данных – Entity Framework.
  4. Общее количество таблиц - не менее 5.
  5. Обязательные операции с данными — просмотр записей таблиц в виде списка, сортировка, поиск, добавление, изменение, удаление записей.
  6. Все страницы сайт должны быть наполнены осмысленной информацией.

4. Содержание курсовой работы

4.1. Задание.

4.2. Пояснительная записка, оформленная по ГОСТ 19.404-79, содержащая:

4.2.1. Описание предметной области.

4.2.2. Структурную схему модели данных.

4.2.3. Структурную схему программы.

Руководитель работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Аникеев С.В. /

(подпись)

Задание принял к исполнению\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ « » 2023г.

(подпись)

Содержание

[Введение 4](#_Toc135855697)

[1 Описание предметной области 5](#_Toc135855698)

[1.2 Сравнительный анализ 6](#_Toc135855699)

[2 Разработка системы 7](#_Toc135855700)

[Описание BPMN - модели 7](#_Toc135855701)

[BPMN модель (Рисунок 1) 8](#_Toc135855702)

[Диаграмма сущностей (Рисунок 2) 9](#_Toc135855703)

[Use Case модель (Рисунок 3) 9](#_Toc135855704)

[Архитектура информационной системы 11](#_Toc135855705)

[База данных — определение 15](#_Toc135855706)

[3 Тестирование системы 19](#_Toc135855707)

[4 Список используемых источников 19](#_Toc135855708)

[5 Листинг программы 21](#_Toc135855709)

# **Введение**

# **Описание предметной области**

В связи с тем, что темпы современной жизни, как и население планеты, возрастают с каждым годом, повышается объем работ с информацией для персонала различных государственных учреждений. При этом имеется необходимость в следующих аспектах:⠀

• быстрое и качественное обслуживание;⠀

• быстрая работа с данными;⠀

• увеличение персонала и числа учреждений, соответствующее росту населения;⠀

• доступность данных.

Следовательно, будет целесообразно создать систему, которая позволит персоналу какого-либо заведения упростить работу. Наличие баз данных в городской больнице позволит решить многие вопросы, такие как:⠀

• скорость обслуживания – отсутствие бумажной волокиты и наличие электронного устройства с доступом в Интернет позволит увеличить скорость работы с данными и избавиться от очередей;⠀

• улучшение качества работы врачей – в связи с тем, что отпадет необходимость в заполнении лишних бумаг у врачей будет больше возможностей уделять пациентам;⠀

• доступность данных – при походе в другое учреждение пациенту не нужно будет забирать свою медицинскую карту из больницы, достаточно просто показать ее на сайте; врачи и медсестры смогут работать и следить за состоянием пациентов дистанционно;⠀

• удобство работы с данными – наличие связанных между собой баз данных позволит легко их дополнять или редактировать;⠀

• сохранность данных – срок хранения существенно возрастает по сравнению с бумажными носителями информации. Целью данной работы будет создание системы для автоматизации ключевых бизнес-процессов городской больницы города. Преимущества данной системы заключаются в следующем:⠀

• простота освоения и использования технологии (для этого достаточно иметь минимальные навыки пользования ПК или мобильным устройством и наличие сети Internet);⠀

• доступность (просмотр или редактура доступны из любого места с любого устройства, при наличии интернета);⠀

• удобность администрирования (сайт и базу данных можно хранить на локальном сервере, расположенном в больнице);⠀

• наглядность;⠀

• обновление в реальном времени (если в карту внесены изменения, они тут же отобразятся у пациента, нет необходимости в постоянном ее получении);⠀

• отсутствие очередей в регистратуру.

# **1.2 Сравнительный анализ**

# **2 Разработка системы**

### Описание BPMN – модели

BPMN диаграмма показывает в какой последовательности совершаются рабочие действия и перемещаются потоки информации (см. рисунок 1)

1. Первым этапом получаем информацию о пациенте.⠀

1.1. Если нет мед карты, то оформляем карту и заносим в бд пациента.

2. Запись⠀⠀

2.1. Если на сегодняшнее число, то проверяем свободен ли врач принять пациента сегодня.⠀⠀

2.2. Если нет, то согласуем определенную дату с пациентом.

3. Уведомление пациента о приёме смс сообщением.

4. Прием.⠀⠀

4.1. Если пациент не пришел, то врач передает информацию в регистратуру.⠀⠀

4.2. Пациент пришел, врач получает описание симптомов и делает первичный осмотр.⠀⠀

4.3. Проверка наличия анализов.⠀ ⠀

4.4. Если результатов нет, то выписывается направление на сдачу анализов.⠀ ⠀

4.5. Если есть, то ставится диагноз и выписывается рецепт.⠀ ⠀

4.6. Информация о текущем состоянии пациента. Если состояние нестабильно, то происходит госпитализация. Если стабильно, то согласуется дата повторного приема

5. Повторный прием.⠀⠀

5.1. Если симптомов больше нет, то завершается прием.⠀ ⠀

5.2. Если симптомы остались, то выписываются другие лекарства или направление к другому врачу.

### BPMN модель (Рисунок 1)

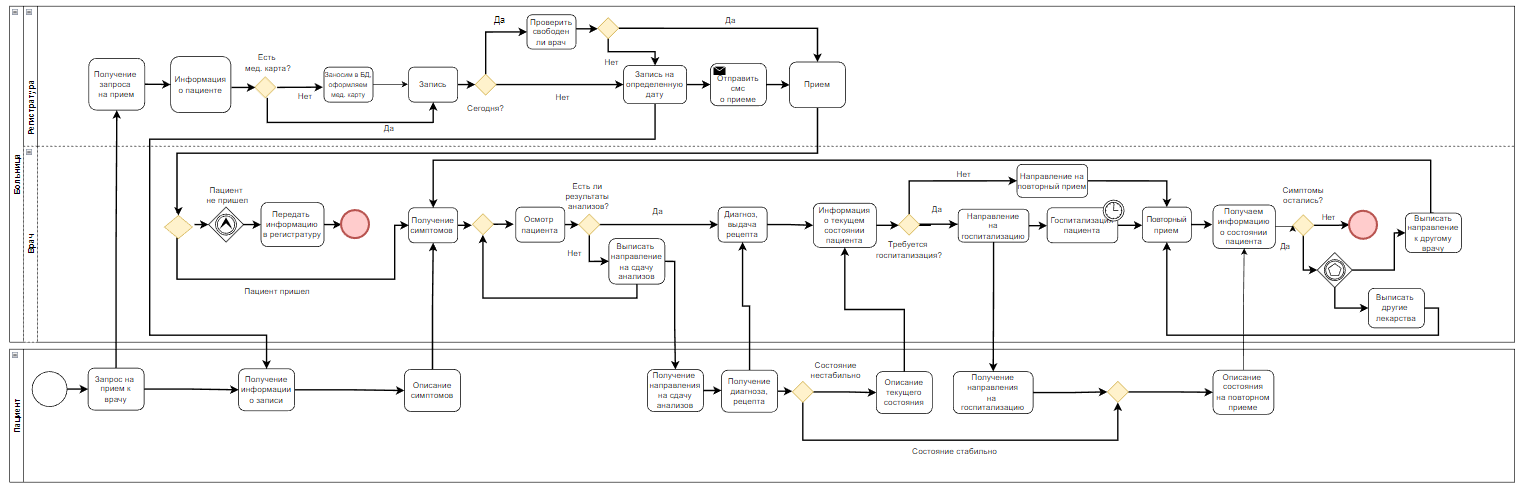


Рисунок 1 - BPMN модель

### Диаграмма сущностей (Рисунок 2)

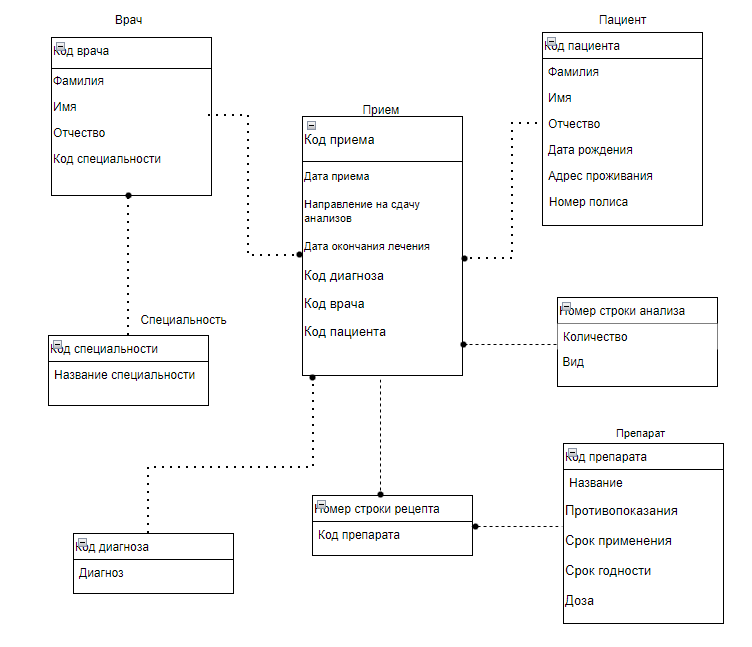


Рисунок 2 - Диаграмма сущностей

### Use Case модель (Рисунок 3)

Назначение данной диаграммы состоит в следующем: проектируемая информационная система представляется в форме так называемых вариантов использования, с которыми взаимодействуют внешние сущности или акторы. При этом актором или действующим лицом называется любой объект, субъект или система, взаимодействующая с моделируемой бизнес-системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая служит источником воздействия на моделируемую систему так, как определит разработчик. Вариант использования служит для описания сервисов, которые система предоставляет актору. Другими словами, каждый вариант использования определяет набор действий, совершаемый системой при диалоге с актором. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие акторов с системой и собственно выполнение вариантов использования.

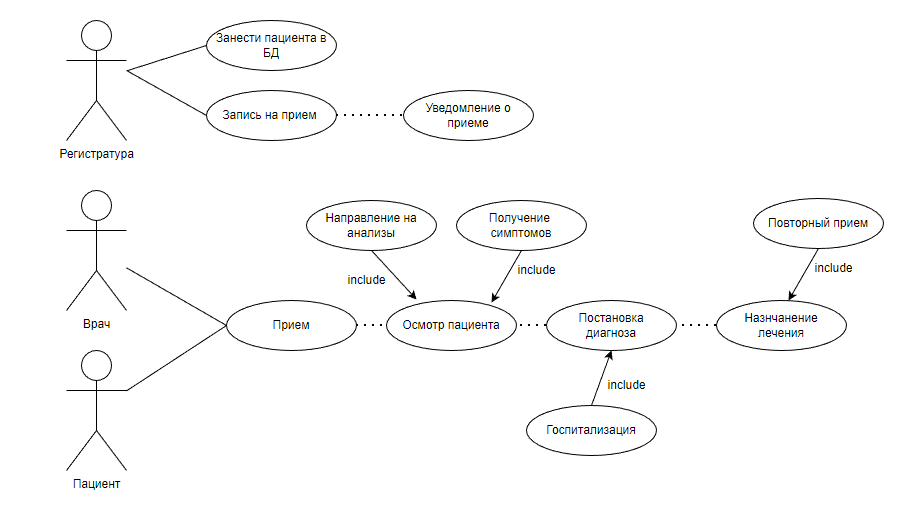


Рисунок 3 - Use Case модель

### Архитектура информационной системы

В соответствии с ранее упомянутыми схемами и диаграммами можно начать разработку самой системы. В данной работе система будет реализована через трёхуровневая архитектуру приложения ASP.NET MVC [7], [8] на языке C# [9].

С# – это объектно-ориентированный язык программирования. Он был создан в период с 1998 по 2002 год командой инженеров Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота.

Язык входит в семью С-подобных языков. Синтаксис приближен к Java и C++. Его особенности:

* статистическая типизация,
* поддерживается полиморфизм,
* поддерживается перегрузка операторов,
* доступна делегация, атрибуты, события, обобщенные типы и анонимные функции.

С# популярен за счет своей «простоты». Простоты для современных программистов и больших команд разработчиков, чтобы те могли в сжатые сроки создавать функциональные и производительные приложения. Этому способствуют нетипичные конструкции языка и специфичный синтаксис, помогающий максимально органично реализовать намеченные функции.

Язык C# практически универсален. Можно использовать его для создания любого ПО: продвинутых бизнес-приложений, видеоигр, функциональных веб-приложений, приложений для Windows, macOS, мобильных программ для iOS и Android.

ASP.NET — это бесплатная веб-платформа для создания веб-сайтов и веб-приложений с помощью HTML, CSS и JavaScript. Возможно также создавать веб-API и использовать технологии в режиме реального времени, такие как веб-сокеты. ASP.NET предлагает три платформы для создания веб-приложений: веб-формы, ASP.NET MVC и веб-страницы ASP.NET. Все три платформы являются стабильными и зрелыми, и реально создавать хорошие веб-приложения с любым из них. Независимо от выбранной платформы преимущества и функции ASP.NET прослеживаются везде.

Если оперировать высокоуровневыми понятиями, то архитектурный шаблон MVC означает, что приложение MVC будет разделено, по крайней мере, на три части, которые описаны ниже:

* Модели, содержащие или представляющие данные, с которыми работают пользователи. Они могут быть простыми моделями представлений, которые только представляют данные, передаваемые между представлениями и контроллерами, или же они могут быть моделями предметной области, которые содержат бизнес-данные, а также операции, преобразования и правила для манипулирования этими данными.
* Представления, применяемые для визуализации некоторой части модели в виде пользовательского интерфейса.
* Контроллеры, которые обрабатывают поступающие запросы, выполняют операции с моделью и выбирают представления для визуализации пользователю.

Модели — это определение "вселенной", в которой функционирует ваше приложение. Например, в банковском приложении модель представляет все аспекты банковской деятельности, поддерживаемые приложением, такие как расчетные счета, главная бухгалтерская книга и кредитные лимиты для клиентов, равно как и операции, которые могут использоваться для манипулирования данными в модели, такие как внесение денежных средств и списание со счетов. Модель отвечает также за сохранение общего состояния и целостности данных - например, удостоверяясь, что все транзакции внесены в главную книгу, а клиент не снимает со счета больше денежных средств, чем имеет на то право, или больше, чем имеется в распоряжении самого банка.

Модели определяются также и тем, за что они не отвечают: модели не имеют дела с визуализацией пользовательских интерфейсов или обработкой запросов, т.к. это ответственность представлений и контроллеров.

Представления содержат логику, необходимую для отображения элементов модели пользователю - и ничего более. Они не имеют никаких прямых сведений о модели и не обмениваются данными непосредственно с ней. Контроллеры являются шлюзом между представлениями и моделью - запросы поступают от клиента и обслуживаются контроллером, который выбирает подходящее представление для отображения пользователю и, если требуется, соответствующую операцию, которая должна быть выполнена над моделью.

Каждая часть архитектуры MVC является четко определенной и самодостаточной — это то, что называется разделением ответственности. Логика, которая манипулирует данными в модели, содержится только в модели, логика, отображающая данные - только в представлении, а код, который обрабатывает запросы и ввод пользователей - только в контроллере. При четком разграничении всех частей приложение будет легче сопровождать и расширять в течение его срока существования, независимо от того, насколько большим оно станет.

Модель предметной области

Наиболее важной частью приложения MVC является модель предметной области. Эта модель создается за счет определения реальных объектов, операций и правил, существующих в данной отрасли или роде деятельности, которые должно поддерживать приложение; такая совокупность и называется предметной областью.

После этого создается программное представление предметной области - модель предметной области. Для целей ASP.NET MVC Framework модель предметной области - это набор типов C# (классов, структур и т.д.), которые все вместе называются типами предметной области. Операции из предметной области представляются методами, определенными в типах предметной области, а правила предметной области выражаются логикой, заключенной внутри этих методов или путем применения атрибутов C#.

Создавая экземпляр типа предметной области для представления конкретной порции данных, мы тем самым создаем объект предметной области. Обычно модели предметной области устойчивы и долговечны. Существует множество различных способов достижения этого, но наиболее распространенным вариантом остаются реляционные базы данных.

Короче говоря, модель предметной области — это единственное официальное определение бизнес-данных и процессов внутри приложения. Устойчивая модель предметной области является также официальным определением состояния представления предметной области. Подход с применением модели предметной области решает многие проблемы, возникающие при сопровождении приложения. Если необходимо манипулировать данными в модели либо добавить новый процесс или правило, то модель предметной области будет единственной частью приложения, которую придется изменить.

Распространенный способ отделения модели предметной области от остальных частей приложения ASP.NET MVC предусматривает помещение модели в отдельную сборку C#. Такой подход позволяет создавать ссылки на модель предметной области из других частей приложения, но не будет гарантировать отсутствие каких-либо ссылок в обратном направлении. Это особенно полезно в крупномасштабных проектах.

Реализация MVC в ASP.NET

В инфраструктуре MVC контроллеры — это классы C#, обычно производные от класса System.Web.Mvc.Controller. Каждый открытый метод в классе, производном от Controller, является методом действия, который посредством системы маршрутизации ASP.NET ассоциируется с конфигурируемым URL. Когда запрос отправляется по URL, связанному с методом действия, операторы в классе контроллера выполняются, чтобы провести некоторую операцию над моделью предметной области и затем выбрать представление для отображения клиенту.

Взаимодействия между контроллером, моделью и представлением показаны на рисунке ниже (Рисунок 5):

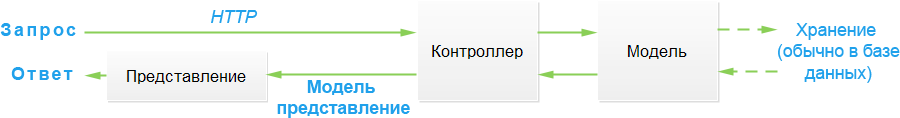


Рисунок 5- Взаимодействия между контроллером, моделью и представлением

Взаимодействия в приложении MVC В инфраструктуре ASP.NET MVC Framework используется механизм визуализации - компонент, отвечающий за обработку представления с целью генерации ответа для браузера. В ранних версиях MVC применялся стандартный механизм визуализации ASP.NET, который обрабатывал ASPX-страницы с использованием модернизированной версии синтаксиса разметки Web Forms. В MVC 3 появился механизм визуализации Razor, усовершенствованный в версии MVC 4 (и оставшийся неизменным в версии MVC 5), в котором применяется совершенно другой синтаксис.

Среда Visual Studio обеспечивает поддержку средства IntelliSense для механизма Razor, упрощая внедрение и ответ на данные представления, передаваемые контроллером.

Инфраструктура ASP.NET MVC не налагает никаких ограничений на реализацию модели предметной области. Модель можно создать с помощью обычных объектов C# и реализовать постоянство с применением любой базы данных, инфраструктуры объектно-реляционного отображения или другого инструмента работы с данными, поддерживаемого .NET.

После описания базовых принципов технологии становится возможным перейти к первому этапу непосредственного создания системы – проектированию и оформлению Базы данных [10].

## База данных — определение

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Данные вместе с СУБД, а также приложения, которые с ними связаны, называются системой баз данных, или, для краткости, просто базой данных.

Данные в наиболее распространенных типах современных баз данных обычно хранятся в виде строк и столбцов, формирующих таблицу. Этими данными можно легко управлять, изменять, обновлять, контролировать и упорядочивать. В большинстве баз данных для записи и запросов данных используется язык структурированных запросов (SQL).

Для базы данных обычно требуется комплексное программное обеспечение, которое называется системой управления базами данных (СУБД). СУБД служит интерфейсом между базой данных и пользователями или программами, предоставляя пользователям возможность получать и обновлять информацию, а также управлять ее упорядочением и оптимизацией. СУБД обеспечивает контроль и управление данными, позволяя выполнять различные административные операции, такие как мониторинг производительности, настройка, а также резервное копирование и восстановление.

Для создания и управление базой данных в работе использовалась СУБД SQL Server Management Studio (SSMS) от Microsoft. Таблицы в базе данных разрабатывались на основании диаграммы сущностей (Рисунок 2) и в соответствии с принципом нормализации базы данных, с упором на 3 нормальную форму [11], [12].

Нормализация баз данных

В реляционных базах данных есть такое понятия, как «Нормализация». Нормализация – это процесс удаления избыточных данных. Также нормализацию можно рассматривать и с позиции проектирования базы данных, в таком случае мы можем сформулировать определение нормализации следующим образом.

Нормализация – это метод проектирования базы данных, который позволяет привести базу данных к минимальной избыточности. Избыточность устраняется, как правило, за счёт декомпозиции отношений (таблиц), т.е. разбиения одной таблицы на несколько.

Избыточность данных создает предпосылки для появления различных аномалий, снижает производительность, и делает управление данными не гибким и не очень удобным.

Отсюда можно сделать вывод, что нормализация нужна для:

* Устранения аномалий
* Повышения производительности
* Повышения удобства управления данными

Избыточность данных – это когда одни и те же данные хранятся в базе в нескольких местах, именно это и приводит к аномалиям. Так как в этом случае необходимо добавлять, изменять или удалять одни и те же данные в нескольких местах. Например, если не выполнить операцию в каком-нибудь одном месте, то возникает ситуация, когда одни данные не соответствуют вроде как точно таким же данным в другом месте.

Следуя определённым правилам и соблюдая определенные требования, мы приводим базу данных к определенной нормальной форме.

Нормальная форма базы данных – это набор правил и критериев, которым должна отвечать база данных. Каждая следующая нормальная форма содержит более строгие правила и критерии, тем самым приводя базу данных к определённой нормальной форме мы устраняем определённый набор аномалий. Отсюда можно сделать вывод, что чем выше нормальная форма, тем меньше аномалий в базе будет.

Иными словами, процесс перехода от одной нормальной формы к следующей – это усовершенствование базы данных.

Так как если база данных находится в какой-то определённой нормальной форме – это означает, что в базе данных отсутствует определенный вид аномалий. Существует 5 основных нормальных форм базы данных:

* Первая нормальная форма (1NF)
* Вторая нормальная форма (2NF)
* Третья нормальная форма (3NF)
* Четвертая нормальная форма (4NF)
* Пятая нормальная форма (5NF)

Однако выделяют еще дополнительные нормальные формы:

* Ненормализованная форма или нулевая нормальная форма (UNF)
* Нормальная форма Бойса-Кодда (BCNF)
* Доменно-ключевая нормальная форма (DKNF)
* Шестая нормальная форма (6NF)

База данных считается нормализованной, если она находится как минимум в третьей нормальной форме (3NF). В реальном мире нормализация до третьей нормальной формы (3NF) является обычной, стандартной практикой, так как 3NF устраняет достаточное количество аномалий, при этом производительность базы данных, а также удобство ее использования не снижается, что нельзя сказать о всех последующих формах.

Требование третьей нормальной формы (3NF) заключается в том, чтобы в таблицах отсутствовала транзитивная зависимость.

Транзитивная зависимость – это когда не ключевые столбцы зависят от значений других не ключевых столбцов. Если в первой нормальной форме основное требование заключалось в соблюдении реляционных принципов, во второй нормальной форме в центре внимания был первичный ключ, то в третьей нормальной форме стоить следить за столбцами, которые не являются первичным ключом, т.е. не ключевыми столбцам. Чтобы нормализовать базу данных до третьей нормальной формы, необходимо сделать так, чтобы в таблицах отсутствовали не ключевые столбцы, которые зависят от других не ключевых столбцов.

Иными словами, не ключевые столбцы не должны пытаться играть роль ключа в таблице, т.е. они действительно должны быть не ключевыми столбцами, такие столбцы не дают возможности получить данные из других столбцов, они позволяют просмотреть информацию, которая в них содержится, так как в этом их назначение.

Главное правило третьей нормальной форме (3NF) звучит следующим образом: Таблица должна содержать правильные не ключевые столбцы.

Таким образом, получается следующая база данных (Рисунки 6-9):

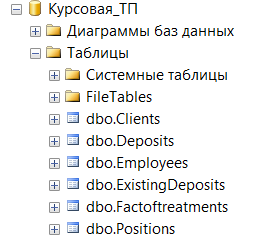


Рисунок 6 – Общий вид таблиц в базе данных

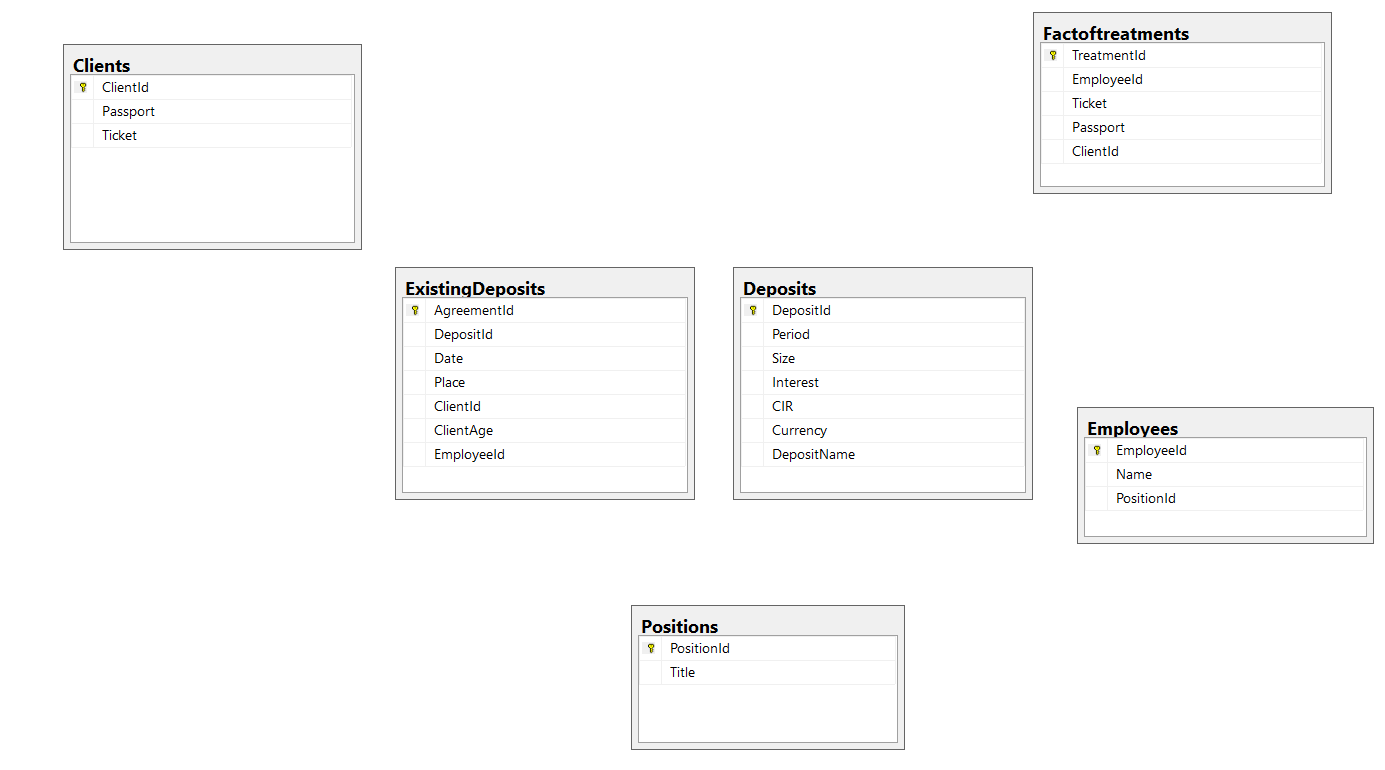


Рисунок 7 – Общая диаграмма таблиц с полями

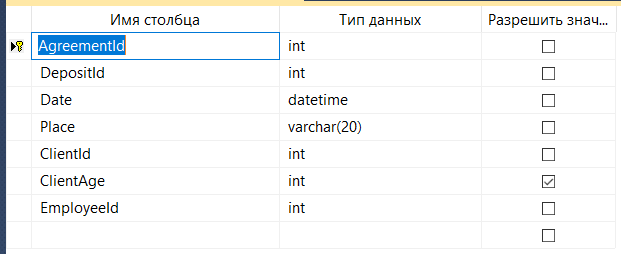


Рисунок 8 – Устройство таблицы в базе данных на примере таблицы оформленных вкладов

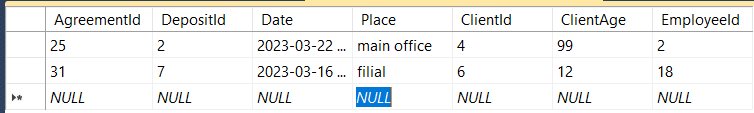


Рисунок 9 – Пример записей в таблице оформленных вкладов

Каждая таблица в базе отражает ранее описанную сущность, а каждое поле характеризует её параметр.

### Концепция Data Access Layer

Следующим шагом в разработке системы является создание «ДАЛа» - слоя доступа к данным [13].

Уровень доступа к данным (DAL) — это компонент архитектуры программного обеспечения, отвечающий за управление хранением и извлечением данных приложения. Он находится между уровнем бизнес-логики и системой хранения данных и обеспечивает уровень абстракции, который позволяет уровню бизнес-логики взаимодействовать с системой хранения данных, не зная о ее конкретной реализации (Рисунок 10).

DAL отвечает за выполнение таких задач, как:

* Подключение к системе хранения данных и управление подключением.
* Генерация и выполнение SQL-запросов или других команд доступа к данным для извлечения и хранения данных.
* Отображение данных из системы хранения данных в объекты данных приложения и наоборот.
* Обработка ошибок и исключений, связанных с доступом к данным.
* Обеспечение поддержки транзакций и других функций доступа к данным.

DAL предназначен для многократного использования и не зависит от бизнес-логики и реализации хранилища данных. Это позволяет легко модифицировать или расширять приложение, не затрагивая базовый код доступа к данным.

DAL может быть реализован с использованием различных технологий доступа к данным, таких как библиотеки ADO.NET, JDBC или ORM (объектно-реляционное сопоставление). Выбор технологии будет зависеть от конкретных требований приложения и используемой системы хранения данных.

Таким образом, уровень доступа к данным (DAL) — это компонент архитектуры программного обеспечения, который отвечает за управление хранением и извлечением данных приложения. Он находится между уровнем бизнес-логики и системой хранения данных и обеспечивает уровень абстракции, который позволяет уровню бизнес-логики взаимодействовать с системой хранения данных, не зная о ее конкретной реализации. DAL предназначен для многократного использования и не зависит от бизнес-логики и реализации хранилища данных. Он может быть реализован с использованием различных технологий доступа к данным и обеспечивает поддержку транзакций и других функций доступа к данным.

Проясним некоторые используемые термины:

1. Объект доступа к данным (DAO): это объекты, которые в совокупности известны как уровень доступа к данным (DAL). DAL может быть одним классом или может состоять из нескольких объектов доступа к данным (DAO).

2. Уровень бизнес-логики. Уровень бизнес-логики состоит из всех объектов и функций, которые обрабатывают бизнес-логику приложения. Именно здесь происходит основная реализация приложения, и он уникален для каждого приложения.

3. Уровень хранения: фактическая база данных, в которой хранятся данные, с которой могут выполняться операции CRUD [14]. Это может быть какое-то удаленное или облачное хранилище.

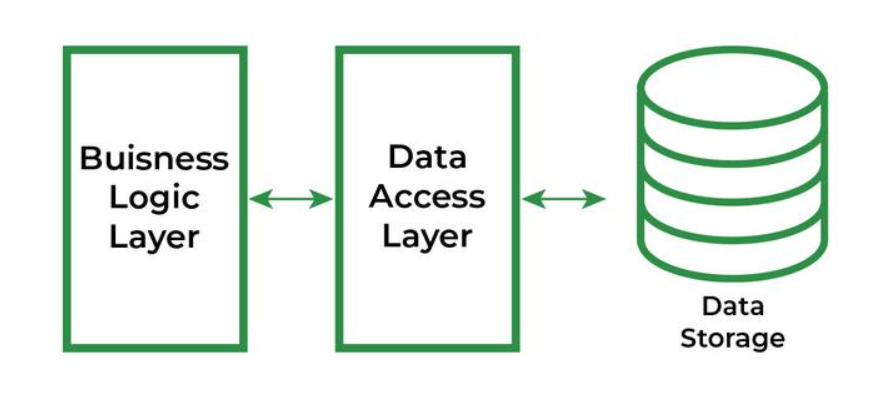


Рисунок 10 - Data Access Layer в общей архитектуре

Всякий раз, когда приложению (BLL) требуется конкретный запрос данных или обновление, запрос отправляется на уровень доступа к данным, который затем предпринимает соответствующие действия, чтобы обеспечить выполнение запрошенных данных или обновления, что приводит к уровню абстракции, при котором бизнес-уровень будет знать только, какую функцию или методы вызывать, и ему не нужно знать фактическую архитектуру или детали всей базы данных.

Любые организации, отслеживающие данные (например, учетные записи, платежную информацию или другие записи), нуждаются в системах, обеспечивающих постоянное хранение, которое обычно организовано в базу данных. Реляционная база данных состоит из данных, организованных по строкам и столбцам. Их можно подключить к другим таблицам с помощью первичных и внешних ключей.

CRUD (создание, чтение, обновление, удаление) — это аббревиатура, обозначающая четыре функции, которые мы используем для реализации приложений постоянного хранения и приложений реляционных баз данных, включая Oracle Database, Microsoft SQL Server и MySQL.

В таблице ниже показано, что означает каждая операция CRUD (Таблица 1).

|  |  |  |
| --- | --- | --- |
| **Письмо** | **Операция** | **Функция** |
| C | Создавать | Вставлять |
| р | Читать | Выбирать |
| U | Обновлять | Редактировать |
| D | Удалить | Удалить |

Таблица 1 – Значение каждой из CRUD операций

Для SQL карты CRUD для вставки, выбора, обновления и удаления соответственно. Такие операции, как управление безопасностью, управление транзакциями, доступ и разрешение, а также оптимизация производительности, основаны на CRUD.

Почему CRUD так важен?

CRUD постоянно используется для всего, что связано с базами данных и проектированием баз данных. Разработчики программного обеспечения ничего не могут сделать без операций CRUD. Например, при разработке веб-сайтов используется REST (передача репрезентативного состояния), который является надмножеством CRUD, используемого для ресурсов HTTP.

С другой стороны, CRUD не менее важен для конечных пользователей. Без него такие вещи, как регистрация на веб-сайтах, создание блогов или закладок, были бы невозможны. Большинство приложений, которые мы используем, позволяют нам добавлять или создавать новые записи, искать существующие, вносить в них изменения или удалять их.

**CRUD предлагает множество преимуществ, в том числе:**

* Это облегчает контроль безопасности, удовлетворяя различные требования доступа.
* Он упрощает и упрощает разработку приложений, делая их более масштабируемыми.
* Он имеет лучшую производительность по сравнению со специальными операторами SQL.

CREATE

Create позволяет добавлять новые строки в таблицу.

READ

Функция чтения похожа на функцию поиска, поскольку позволяет извлекать определенные записи и считывать их значения. Она не внесет никаких изменений в таблицу, а просто отобразит все записи в ней.

UPDATE

Обновление — это то, как можно изменять существующую запись в таблице. Мы можем использовать это для изменения существующих записей в базе данных. При выполнении UPDATE необходимо определить целевую таблицу и столбцы, которые необходимо обновить, могут также понадобиться связанные значения, а иногда и строки.

DELETE

Удалить используется для удаления записи из таблицы. SQL имеет встроенную функцию удаления для одновременного удаления одной или нескольких записей из базы данных. Некоторые приложения реляционных баз данных могут разрешать жесткое удаление (безвозвратное удаление) или мягкое удаление (обновление статуса строки).

Обозначив основные термины, можно приступить к реализации компонента – в общем смысле, ДАЛ составлен из моделей, контроллеров, Application context [15] и связи с БД (Рисунок 11).

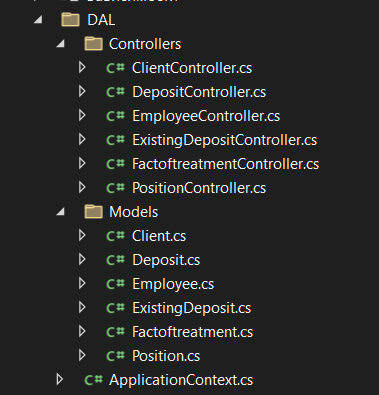


Рисунок 11 – Общий вид слоя доступа к данным

Модели – это сущности, реализованные ранее в Базе данных, а контроллеры, прописанные под них, позволяют производить с сущностями CRUD операции.

Однако моделей и контроллеров для работы с базой данных недостаточно, чтобы реализовать полноценное взаимодействие можно использовать технологию Entity Framework Core [15] через Application context и Connection String.

Что такое Entity Framework Core

Entity Framework представляет ORM-технологию (object-relational mapping - отображения данных на реальные объекты) от компании Microsoft для доступа к данным. Entity Framework Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными как с объектами классом независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Как технология доступа к данным Entity Framework Core работает поверх платформы .NET и поэтому может использоваться на различных платформах стека .NET.

Также стоит отметить, что EF Core предоставляет универсальный API [16] для работы с данными. И если, к примеру, мы решим сменить целевую СУБД, то основные изменения в проекте будут касаться прежде всего конфигурации и настройки подключения к соответствующим провайдерам. А код, который непосредственно работает с данными, получает данные, добавляет их в БД и т.д., останется прежним.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность определяет набор данных, которые связаны с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их коллекциями.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст. Свойства необязательно представляют простые данные типа int или string, но могут также представлять и более комплексные типы данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Для дальнейшего описания стоит дать и раскрыть определение понятия API и REST API, поскольку последнее использовалось при построении системы:

Что значит API?

API – Application Programming Interface, что значит программный интерфейс приложения. В контексте API слово «приложение» относится к любому ПО с определенной функцией. Интерфейс можно рассматривать как сервисный контракт между двумя приложениями. Этот контракт определяет, как они взаимодействуют друг с другом, используя запросы и ответы. Документация API содержит информацию о том, как разработчики должны структурировать эти запросы и ответы.

Как работают API?

Архитектура API обычно объясняется с точки зрения клиента и сервера. Приложение, отправляющее запрос, называется клиентом, а приложение, отправляющее ответ, называется сервером. Итак, в примере с погодой база данных службы – это сервер, а мобильное приложение – это клиент.

Существует четыре различных способа работы API в зависимости от того, когда и почему они были созданы.

SOAP API

SOAP – Simple Object Access Protocol, т. е. простой протокол доступа к объектам. Клиент и сервер обмениваются сообщениями посредством XML. Это менее гибкий API, который был более популярен в прошлом.

RPC API

Такие API называются системой удаленного вызова процедур. Клиент выполняет функцию (или процедуру) на сервере, и сервер отправляет результат обратно клиенту.

Websocket API

Websocket API – это еще одна современная разработка web API, которая использует объекты JSON для передачи данных. WebSocket API поддерживает двустороннюю связь между клиентскими приложениями и сервером. Сервер может отправлять сообщения обратного вызова подключенным клиентам, что делает его более эффективным, чем REST API.

REST API

На сегодняшний день это самые популярные и гибкие API-интерфейсы в Интернете. Клиент отправляет запросы на сервер в виде данных. Сервер использует этот клиентский ввод для запуска внутренних функций и возвращает выходные данные обратно клиенту. Давайте рассмотрим API REST более подробно ниже.

Что такое REST API?

REST – это Representational State Transfer, т. е. передача репрезентативного состояния. REST определяет набор функций, таких как GET, PUT, DELETE и т. д., которые клиенты могут использовать для доступа к данным сервера. Клиенты и серверы обмениваются данными по протоколу HTTP.

Главной особенностью REST API является то, что такая передача выполняется без сохранения состояния. Без сохранения состояния означает, что серверы не сохраняют клиентские данные между запросами. Клиентские запросы к серверу аналогичны URL-адресам, которые вы вводите в браузере для посещения веб-сайта. Ответ от сервера представляет собой простые данные без типичного графического отображения веб-страницы.

Что такое web API?

Web API или Web Service API –это интерфейс обработки приложений между веб-сервером и веб-браузером. Все веб-сервисы являются API, но не все API являются веб-сервисами. REST API – это особый тип Web API, в котором используется стандартный архитектурный стиль, описанный выше.

Различные термины, которые относятся к API, такие как Java API или сервисные API, существуют потому, что исторически API были созданы до всемирной паутины. Современные web API – это REST API, и эти термины могут использоваться взаимозаменяемо.

Что такое интеграции API?

Интеграции API – это программные компоненты, которые автоматически обновляют данные между клиентами и серверами. Некоторыми примерами интеграции API являются автоматическая синхронизация данных в облаке из галереи изображений телефона или автоматическая синхронизация времени и даты ноутбуке при смене часового пояса. Организации также могут использовать их для эффективной автоматизации многих системных функций (Рисунок 12).

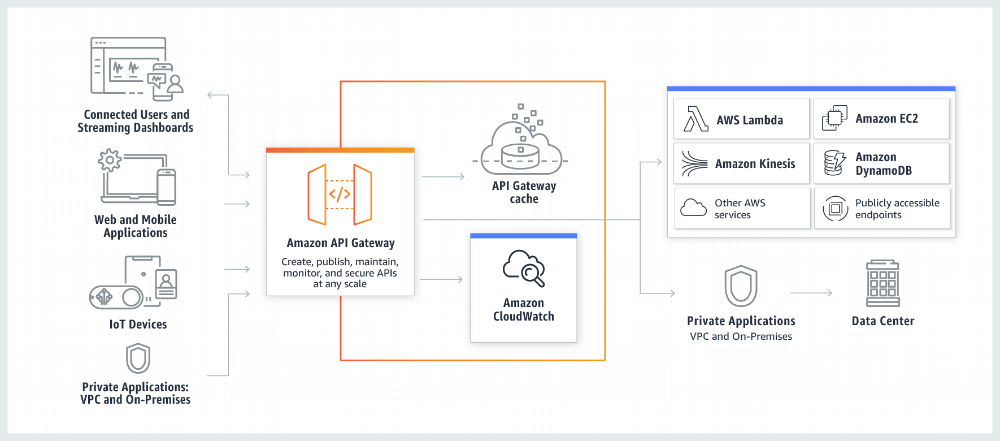


Рисунок 12 – Интеграция API на примере Амазон

Также стоит упомянуть и ASP.NET Core [17]:

ASP.NET Core представляет технологию для создания веб-приложений на платформе .NET, развиваемую компанией Microsoft. В качестве языков программирования для разработки приложений на ASP.NET Core используются C# и F#.

Архитектура и модели разработки

Текущую архитектуру платформы ASP.NET Core можно выразить следующим образом (Рисунок 13):

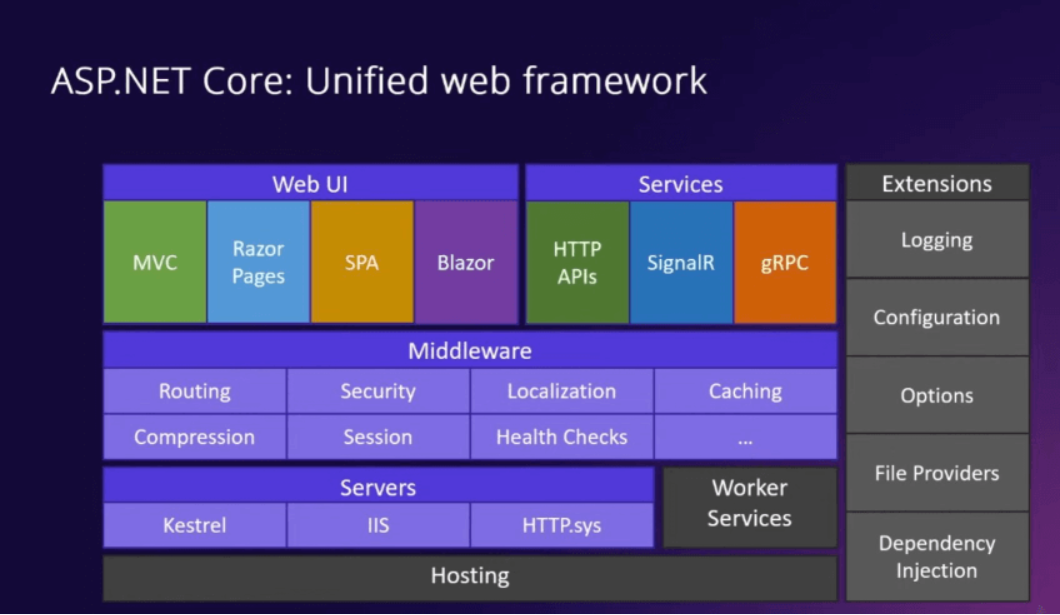


Рисунок 13 – Архитектура ASP.NET Core

на самом верхнем уровне располагаются различные модели взаимодействия с пользователем. Это технологии построения пользовательского интерфейса и обработки ввода пользователя, как MVC, Razor Pages, SPA (Single Page Application - одностраничные приложения с использованием Angular, React, Vue) и Balzor. Кроме того, это сервисы в виде встроенных HTTP API, библиотеки SignalR или сервисов GRPC.

Все эти технологии базируются и/или взаимодействуют с чистым ASP.NET Core, который представлен прежде всего различными встроенными компонентами middleware - компонентами, которые применяются для обработки запроса. Кроме того, технологии высшего уровня также взаимодействуют с различными расширениями, которые не являются непосредственной частью ASP.NET Core, как расширения для логирования, конфигурации и т.д.

И на самом нижнем уровне приложение ASP.NET Core работает в рамках некоторого веб-сервера, например, Kestrel, IIS, библиотеки HTTP.sys.

Это вкратце архитектура платформы, теперь рассмотрим моделей разработки приложения ASP.NET Core:

Прежде всего это базовый ASP.NET Core, который поддерживает все основные моменты, необходимые для работы современного веб-приложения: маршрутизация, конфигурация, логирования, возможность работы с различными системами баз данных и т.д. В ASP.NET Core 6 в фреймворк был добавлен так называемый Minimal API - минимизированная упрощенная модель, который еще упростила процесс разработки и написания кода приложения. Все остальные модели разработки работаю поверх базового функционала ASP.NET Core

ASP.NET Core Web API представляет реализацию паттерна REST, при котором для каждого типа http-запроса (GET, POST, PUT, DELETE) предназначен отдельный ресурс. Подобные ресурсы определяются в виде методов контроллера Web API. Данная модель особенно подходит для одностраничных приложений, но не только.

Последние два упомянутые компонента – класс Application context и ConnectionString необходимы для взаимодействия с БД.

Application context взаимодействует с БД, в том числе, посредством классов DbSet [18] и DbContext [19].

DbSet Класс

Неуниверсийная версия, которую можно использовать, если тип сущности DbSet <TEntity> неизвестен во время сборки.

DbContext Класс

Экземпляр DbContext представляет сеанс с базой данных и может использоваться для запроса и сохранения экземпляров сущностей. DbContext — это сочетание шаблонов "Единица работы" и "Репозиторий".

В ходе своей работы Application context использует вышеупомянутые классы и с помощью файла appsettings.json и строки подключения Connection String устанавливает соединение с базой данных по указанному названию БД и сервера.

### Слой бизнес-логики [20]

Для пояснения необходимости слоя бизнес-логики воспользуемся изображением (Рисунок 14).

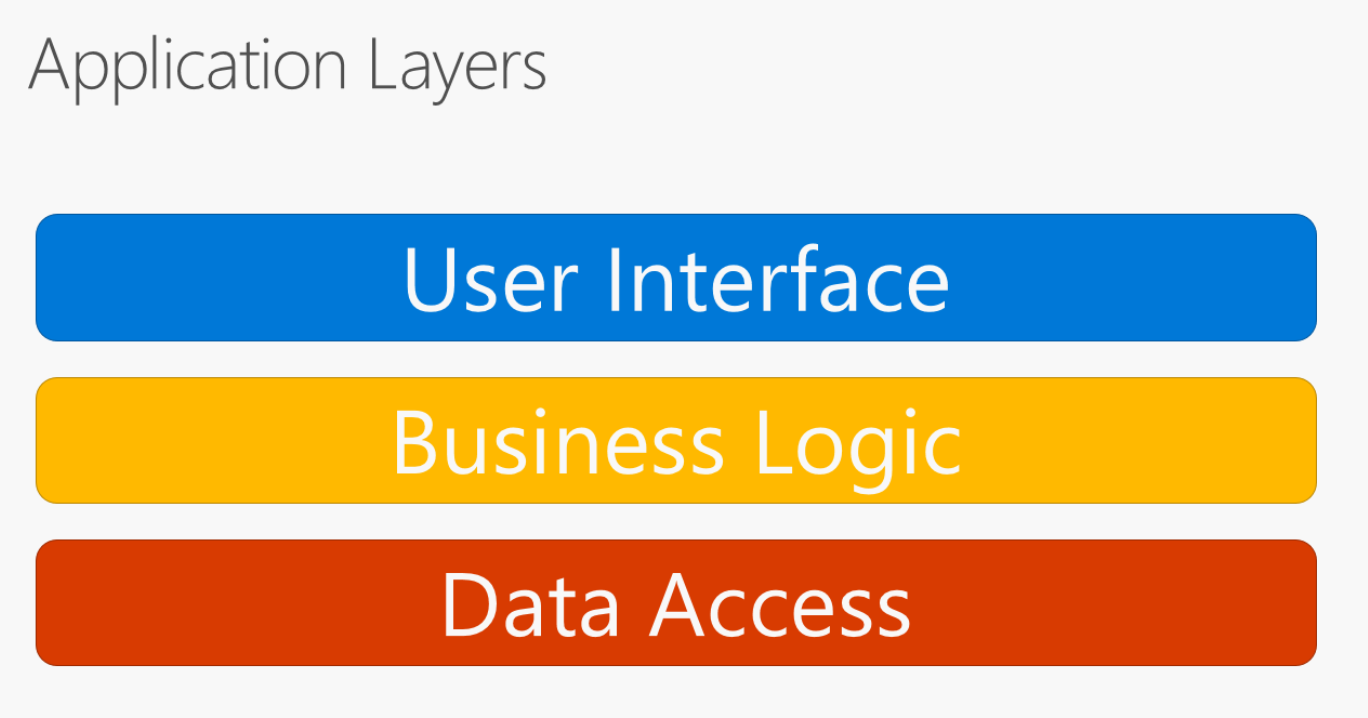


Рисунок 14 – Слои веб-приложения

На рисунке 14 вновь представлена ранее упомянутая трёхслойная структура приложения, только если раньше она упоминалась с позиции слоя данных, теперь уже с точки зрения слоя бизнес-логики.

Что представляют собой слои?

По мере увеличения сложности приложения для эффективного управления им может применяться разбиение по обязанностям и задачам. Такой подход соответствует принципу разделения задач и помогает сохранить организацию расширяющейся базы кода, благодаря чему разработчики могут быстро определять, где именно реализованы определенные функции. Многослойная архитектура имеет также целый ряд других преимуществ.

Благодаря упорядочению кода с помощью слоев общие низкоуровневые функции могут многократно использоваться по всему приложению. Это крайне важно, поскольку такой подход требует меньшего объема кода и, за счет стандартизации приложения на уровне одной реализации, соответствует принципу "Не повторяйся".

В приложениях с многослойной архитектурой могут устанавливаться ограничения на взаимодействие между слоями. Такая архитектура помогает реализовать инкапсуляцию. При изменении или замене слоя будут затронуты только те слои, которые работают непосредственно с ним. Ограничивая зависимости слоев друг от друга, можно уменьшить последствия внесения изменений, в результате чего единичное изменение не будет влиять на все приложение.

Применение слоев (и инкапсуляция) позволяет заметно упростить замену функциональных возможностей в рамках приложения. Например, приложение может изначально использовать собственную базу данных SQL Server для сохраняемости, а впоследствии перейти на стратегию сохранения состояния на основе облака или веб-API. Если в приложении надлежащим образом инкапсулирована реализация сохраняемости на логическом слое, этот слой SQL Server может быть заменен новым, где будет реализовываться тот же открытый интерфейс.

Помимо возможности замены реализаций в связи с последующими изменениями, применение слоев в приложении также позволяет менять реализации в целях тестирования. Вместо написания тестов, которые применяются к слоям реальных данных или пользовательского интерфейса приложения, во время тестирования они заменяются фиктивными реализациями, которые демонстрируют известную реакцию на запросы. Как правило, это значительно упрощает написание тестов и ускоряет их выполнение по сравнению с тестированием в реальной инфраструктуре приложения.

Слои обеспечивают логический уровень разделения в приложении. Если логика приложения физически распределена между несколькими серверами или процессами, такие раздельные физические целевые объекты развертывания называются уровнями. Таким образом, не только возможно, но и широко распространено развертывание N-слойных приложений на одном уровне.

Как правило, в приложении определяются слои пользовательского интерфейса, бизнес-логики и доступа к данным. В рамках такой архитектуры пользователи выполняют запросы через слой пользовательского интерфейса, который взаимодействует только со слоем бизнес-логики. Слой бизнес-логики, в свою очередь, может вызывать слой доступа к данным для обработки запросов. Слой пользовательского интерфейса не должен выполнять запросы напрямую к слою доступа к данным и какими-либо другими способами напрямую взаимодействовать с функциями сохраняемости. Аналогичным образом, слой бизнес-логики должен взаимодействовать с функциями сохраняемости только через слой доступа к данным. Таким образом, для каждого слоя четко определена своя обязанность.

После общего введения в вопрос необходимости реализации слоя бизнес-логики стоит отдельно выделить термин НТТР [21] и связанные с ним понятия [22], [23], поскольку именно благодаря HTTP запросам происходит взаимодействие внутри системы.

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту. После этого клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом.  
  
Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером. На данный момент именно благодаря протоколу HTTP обеспечивается работа Всемирной паутины.  
  
Также HTTP часто используется как протокол передачи информации для других протоколов прикладного уровня, таких как SOAP, XML-RPC и WebDAV. В таком случае говорят, что протокол HTTP используется как «транспорт».  
  
API многих программных продуктов также подразумевает использование HTTP для передачи данных — сами данные при этом могут иметь любой формат, например, XML или JSON.  
  
Как правило, передача данных по протоколу HTTP осуществляется через TCP/IP-соединения. Серверное программное обеспечение при этом обычно использует TCP-порт 80 (и, если порт не указан явно, то обычно клиентское программное обеспечение по умолчанию использует именно 80-й порт для открываемых HTTP-соединений), хотя может использовать и любой другой.

Методы HTTP запроса

HTTP определяет множество методов запроса, которые указывают, какое желаемое действие выполнится для данного ресурса. Несмотря на то, что их названия могут быть существительными, эти методы запроса иногда называются HTTP глаголами. Каждый реализует свою семантику, но каждая группа команд разделяет общие свойства: так, методы могут быть безопасными, идемпотентными или кешируемыми.

GET Метод

GET запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные.

POST Метод

POST используется для отправки сущностей к определённому ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере.

Для отправки HTTP-запросов в .NET применяется класс HttpClient из пространства имен System.Net.Http.

При создании объекта HttpClient следует учитывать, что он нацелен на многоразовое использование в течение всей жизни приложения. Создание отдельного объекта для каждого запроса может привести к исчерпанию количества доступных сокетов и приведет к ошибкам SocketException.

### Фронт-энд, слой пользовательского интерфейса

После разработки предыдущих двух слоёв и базы данных, можно приступить к созданию самого сайта.

Основой нашего веб-сайта являются HTML, CSS, JavaScript.

Что такое HTML

Это один из основных строительных блоков сети. HTML, или «язык гипертекстовой разметки», существует в той или иной форме примерно с 1993 года, когда он был создан физиком Тимом Бернерсом-Ли. Сейчас это пятое поколение HTML5.

HTML — это не язык программирования, а «язык разметки». Например, он использует синтаксис тегов для изменения способа отображения текста. Он также может определять, где изображения размещаются на странице.

Что такое CSS

CSS или «каскадные таблицы стилей» работают с HTML для создания формата веб-страниц. Он работает поверх основы страницы, созданной с помощью HTML. Это также помогает адаптировать страницы к различным форматам, оптимизированным для настольных компьютеров или мобильных устройств. Последним стандартом является CSS3, то есть они относятся к третьему поколению.

CSS — это то, что придает веб-сайтам безупречный и профессиональный вид. Это также позволяет добавлять интерактивные элементы, такие как цвет фона, заголовки, формы, графика и многое другое, что делает веб-сайты и веб-приложения гораздо более привлекательными для зрителя.

Что такое JavaScript

JavaScript, который часто сокращают до JS, — это еще один интерфейсный язык программирования, который является одной из основных технологий Всемирной паутины, наряду с HTML и CSS. По состоянию на 2022 год 98% веб-сайтов используют JavaScript на стороне клиента для поведения веб-страниц, часто включая сторонние библиотеки, такие как JQuery. В то время как HTML и CSS используются для управления представлением, форматированием и макетом, JavaScript используется для управления поведением различных веб-элементов.

Несмотря на то, что эти технологии легли в основу нашего сайта, реализован он был в основном через библиотеку вышеупомянутого JavaScript – React [25].

React**— это JavaScript-библиотека для создания пользовательских интерфейсов.** Именно библиотека, а не фреймворк. React часто называют фреймворком, но это ошибка. Во-первых, его использование ни к чему вас не обязывает, не формирует «фрейм» проекта. Во-вторых, React выполняет единственную задачу: показывает на странице компонент интерфейса, синхронизируя его с данными приложения, и только этой библиотеки в общем случае недостаточно для того, чтобы полностью реализовать проект.

Вскоре после появления React и подобные ему решения (Vue.js, Svelte) практически захватили мир фронтенда: потому что они помогают решать проблемы, основываясь на идее декларативного программирования, а не на императивном подходе.

— **Декларативный подход** состоит в описании конечного результата (что мы хотим получить).

— При **императивном подходе** описываются конкретные шаги для достижения конечного результата (как мы хотим что-то получить).

Оказалось, что декларативный подход отлично подходит для создания интерфейсов, и он прижился в сообществе. Этот подход работает не только в вебе: сравнительно недавно компания Apple представила фреймворк SwiftUI, основанный на тех же принципах.

При использовании React можно выделить такие особенности:

* Разметка и относящаяся к ней логика находятся рядом и связаны друг с другом. Это упрощает дальнейшую работу с кодом.
* Выделен счётчик с кнопкой в компонент. Это значит, что мы можем очень легко его пере использовать.
* Больше не нужно использовать идентификаторы для обращения к DOM-элементам, что также делает код более легко поддерживаемым.
* Состояние компонента изолировано: нет никакой возможности модифицировать значение извне, если мы явно такое не задумывали. Благодаря этому поток данных в приложении становится более предсказуемым, что упрощает разработку и отладку.

Также стоит отметить, что в React-приложениях мы не работаем напрямую с DOM-деревом. Вместо этого мы описываем разметку с помощью JSX [26], а React уже сам решает, как превратить её в реальные DOM-элементы.

JSX — расширение синтаксиса JavaScript. Этот синтаксис выглядит как язык шаблонов, но наделён всеми языковыми возможностями JavaScript. В результате компиляции JSX и вызова React.createElement() возникают простые объекты — «React-элементы».

Следующим шагов в разработке веб-интерфейса стало внедрение REST API через Fetch API и Axios.

В React доступны различные способы использования REST API в приложениях, в том числе использование встроенного JavaScript- метода fetch() и Axios, который является HTTP-клиентом на основе промисов для браузера.

API fetch() является встроенным методом JavaScript, предназначенным для получения ресурсов от сервера или конечной точки API. Он похож на [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest" \t "_blank), но [fetch API](https://developer.mozilla.org/ru/docs/Web/API/Fetch_API" \t "_blank) предоставляет более мощный и гибкий набор функций.

Он определяет такие понятия, как [CORS](https://developer.mozilla.org/ru/docs/Web/HTTP/CORS) и семантика [заголовка HTTP Origin](https://developer.mozilla.org/ru/docs/Web/HTTP/%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8), перенося их отдельные определения в другие места.

Метод fetch() API всегда принимает обязательный аргумент, представляющий собой путь или URL-адрес ресурса, который вы хотите получить. Он возвращает промис, который указывает на ответ от запроса, независимо от того, был ли запрос успешным или нет. При желании вы также можете передать объект параметров инициализации в качестве второго аргумента.

После получения ответа доступно несколько встроенных методов, позволяющих определить, каково содержимое тела и как его следует обрабатывать.

Axios — это простой в использовании HTTP-клиент для браузера и node.js на основе промисов. Поскольку Axios основан на промисах, мы можем воспользоваться асинхронностью и ожидать более читабельного и асинхронного кода. С помощью Axios мы получаем возможность перехватывать и отменять запросы, а также у нас появится встроенная функция, которая обеспечивает защиту от подделки межсайтовых запросов на стороне клиента.

Функции Axios

* Перехват запросов и ответов.
* Упрощенная обработка ошибок.
* Защита от
* Поддержка процесса загрузки.
* Задержка ответа.
* Возможность отмены запросов.
* Поддержка устаревших браузеров.
* Автоматическое преобразование данных JSON.

Отдельное внимание стоит уделить хукам в React [27].

Хуки — механизм в React, который позволяет работать полностью без классов. Он не приносит ничего нового, но облегчает повторное использование кода для решения общих задач. Сейчас это основной способ написания React-приложений. Но хуки не заменяют собой классы целиком.

В моей работе использовался useState – встроенный в React хук.

На уровне фронт-энд реализовывалась также и фильтрация объектов по параметру, она производилась посредством вышеописанных технологий.

Поле выполнения всех ранее названных действий, информационно-справочная система является готовой.

# **3 Тестирование системы**

# **4 Заключение**

Создание Информационно-справочной системы представляет собой сложный процесс проектирования, разработки и тестирования. Целью проектирования являются обозначение контуров и деталей будущей системы и создание соответствующей документации. В процессе разработки выстраивается сам объект, устанавливается его архитектура, связи между компонентами, характерные особенности. Конечным этапом является тестирование системы, в ходе которого выявляются все возможные недостатки и ошибки, с последующим их устранением.

В данной курсовой работе был разработан проект информационной системы «Вклад в банке»

В ходе курсовой работы были решены следующие поставленные задачи:

- выполнен анализ предметной области;

- смоделирована предметная область;

- разработаны все слои системы.

-протестирован функционал конечного продукта

# **5 Список используемых источников**

1) Банки инвестируют в информационные технологии / fintechru.org [сайт]. – 2022. – URL: https://www.fintechru.org/publications/banki-investiruyut-v-informatsionnye-tekhnologii/ (дата обращения 21.05.2023).

2) Цифровая трансформация российского финансового рынка как ключевая стратегия в пост ковидный период / cyberleninka.ru [сайт]. – 2021. – URL: https://cyberleninka.ru/article/n/tsifrovaya-transformatsiya-rossiyskogo-finansovogo-rynka-kak-klyuchevaya-strategiya-v-postkovidnyy-period (дата обращения 21.05.2023).

3) Современные информационные системы в банковской деятельности / elib.bsu.by [сайт]. – 2021. – URL:<https://elib.bsu.by/bitstream/123456789/216655/1/165-168.pdf> (дата обращения 21.05.2023).

4) Информационные технологии в банковской сфере: ТОП-5 IT трендов / doczilla.pro [сайт]. – 2021. – URL: https://doczilla.pro/ru/blog/informacionnye-tekhnologii-v-bankovskoj-sfere-top-5-it-trendov/ (дата обращения 22.05.2023).

5) Приложение в браузере: чем заменить привычные мобильные сервисы/ trends.rbc.ru [сайт]. – 2022. – URL: https://trends.rbc.ru/trends/industry/625e988e9a794721e46f03df/ (дата обращения 22.05.2023).

6) Mobile banking and fintech apps: industry trends for UI design and technologies / advapay.eu [сайт]. – 2023. – URL: <https://advapay.eu/mobile-banking-and-fintech-apps-industry-trends-for-ui-design-and-technologies/>(дата обращения 22.05.2023).

7) Обзор ASP.NET / learn.microsoft.com [сайт]. – 2022. – URL: https://learn.microsoft.com/ru-ru/aspnet/overviewcsharp (дата обращения 23.05.2023).

8) Архитектура ASP.NET MVC 5 / professorweb.ru [сайт]. – 2022. – URL: https://learn.microsoft.com/ru-ru/aspnet/overviewcsharp (дата обращения 23.05.2023).

9) Язык программирования C#: краткая история, возможности и перспективы / timeweb.com [сайт]. – 2021. – URL: https://timeweb.com/ru/community/articles/chto-takoe-csharp (дата обращения 23.05.2023).

10) Что такое база данных? / oracle.com [сайт]. – 2022. – URL: https://www.oracle.com/cis/database/what-is-database/ (дата обращения 23.05.2023).

11) Нормализация баз данных простыми словами / info-comp.ru [сайт]. – 2020. – URL: https://info-comp.ru/database-normalization (дата обращения 24.05.2023).

12) Третья нормальная форма (3NF) базы данных / info-comp.ru [сайт]. – 2020. – URL: https://info-comp.ru/third-normal-form (дата обращения 24.05.2023).

13) Data-Access Layer / geeksforgeeks.org [сайт]. – 2023. – URL: https://www.geeksforgeeks.org/data-access-layer/ (дата обращения 24.05.2023).

14) Что такое CRUD-операции/ bestprogrammer.ru [сайт]. – 2021. – URL: https://bestprogrammer.ru/izuchenie/chto-takoe-crud-operatsii (дата обращения 24.05.2023).

15) Руководство по Entity Framework Core 7 / metanit.com [сайт]. – 2022. – URL: https://metanit.com/sharp/efcore/ (дата обращения 24.05.2023).

16) Что такое API? / aws.amazon.com [сайт]. – 2022. – URL: https://aws.amazon.com/ru/what-is/api/ (дата обращения 24.05.2023).

17) Введение в ASP.NET Core/ metanit.com [сайт]. – 2022. – URL: https://metanit.com/sharp/aspnet6/1.1.php (дата обращения 24.05.2023).

18) DbSet Класс/ learn.microsoft.com [сайт]. – 2022. – URL: https://learn.microsoft.com/ru-ru/dotnet/api/system.data.entity.dbset?view=entity-framework-6.2.0 (дата обращения 24.05.2023).

19) DbContext Класс/ learn.microsoft.com [сайт]. – 2022. – URL: https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.entityframeworkcore.dbcontext?view=efcore-7.0 (дата обращения 24.05.2023).

20) Microsoft Developer Division, .NET, and Visual Studio product teams// Architecting-Modern-Web-Applications-with-ASP.NET-Core-and-Azure / [книга]. Microsoft Corporation 2022. 22 с. – URL: <https://dotnet.microsoft.com/en-us/download/e-book/aspnet/pdf> (дата обращения 24.05.2023).

21) Простым языком об HTTP/ habr.com [сайт]. – 2014. – URL: https://habr.com/ru/articles/215117/ (дата обращения 24.05.2023).

22) Методы HTTP запроса/ developer.mozilla.org [сайт]. – 2023. – URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Methods (дата обращения 24.05.2023).

23) Создание HttpClient/ metanit.com [сайт]. – 2022. – URL: https://metanit.com/sharp/net/2.1.php (дата обращения 24.05.2023).

24) What are HTML, CSS, & Javascript? / ciat.edu [сайт]. – 2022. – URL: https://www.ciat.edu/blog/html-css-javascript/#:~:text=HTML%20defines%20the%20structure%20of,learn%20them%20in%20that%20order. (дата обращения 24.05.2023).

25) Что такое React и как его освоить? / academy.yandex.ru [сайт]. – 2021. – URL: https://academy.yandex.ru/journal/chto-takoe-react-i-kak-ego-osvoit (дата обращения 24.05.2023).

26) Словарь терминов React / ru.legacy.reactjs.org [сайт]. – 2022. – URL: https://ru.legacy.reactjs.org/docs/glossary.html#:~:text=JSX%20%E2%80%94%20%D1%80%D0%B0%D1%81%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%B8%D0%B5%20%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81%D0%B0%20JavaScript.,%E2%80%94%20%C2%ABReact%2D%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%8B%C2%BB. (дата обращения 24.05.2023).

# **6 Листинг программы**