**2 АРХИТЕКТУРА СИСТЕМЫ И ИСПОЛЬЗОВАННЫЕ ТЕХНОЛОГИИ**

**2.1 Архитектура системы**

Так как хотелось, чтобы разработанной системой можно было пользоваться на максимальном количестве устройств, несмотря на операционную систему и тип (персональный компьютер, планшет, мобильный телефон), было выбрано реализовывать систему как Web-приложение. Для реализации web-приложения был выбран язык программирования C#. В качестве web-фреймворка был выбран .NET 4.5 – один из наиболее популярных web-фреймворков, который можно использовать с C#. Для работы с данными используется фреймворк Entity Framework, который позволяет абстрагироваться от конкретного типа базы данных в приложении. В качестве базы данных используется Oracle. В подпунктах главы 2 подробнее будет рассмотрена каждая из выбранных технологий и будет дано обоснование ее выбора.

Для реализации Web-приложения в качестве архитектуры было решено использовать приложение одной страницы (SPA). SPA – это Web-приложение, размещенное на одной странице, которое для обеспечения работы загружает все javascript-файлы (модули, виджеты, контролы и т.д.), а также файлы CSS вместе с загрузкой самой страницы. Основные плюсы использования SPA принципа архитектуры приложения:

1. Работа на большом количестве устройств, а значит, создав одно приложение, вы получаете гораздо большую аудиторию пользователей нежели при использовании стандартного подхода.
2. Богатый пользовательский интерфейс. Так как web-страница одна, построить богатый, насыщенный пользовательский интерфейс гораздо проще. Проще хранить информацию о сеансе, управлять состояниями представлений и управлять анимацией в некоторых случаях.
3. Существенно сокращается так называемые “хождения по кругу”, то есть загрузку одного и того же контента снова и снова. Если ваше приложение использует шаблон, то вместе с основным содержанием какой-либо страницы пользователь обязательно загружает разметку шаблона. Действительно, на данном этапе развития браузеров кэширование данных достигло высочайших результатов, но с другой стороны, если нечего кэшировать, то время и ресурсы на это не тратятся.

Для реализации SPA было выбрано использовать на клиентской стороне приложения MVW-фреймворк AngularJS, а на серверной части приложения ASP .NET MVC 5.

Ниже приведен полный стек использованных технологий:

* платформа .NET 4.5;
* язык программирования С#;
* тип приложения ASP .NET Web API + ASP .NET MVC 5;
* Entity Framework;
* база данных Oracle;
* AngularJS;
* KendoUI;
* Twitter Bootstrap;
* Html, CSS, LESS;
* JS Plugins (Latex, ...);
* Grunt, Gulp.

**2.3.1 Язык программирования C#**

В качестве языка программирования был выбран C#. На сегодняшний момент язык программирования C# - один из самых мощных, быстро развивающихся и востребованных языков в IT-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программ до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

C# является объектно-ориентированным языком с Си-подобным синтаксисом. Он поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, с каждой новой версией появляется все больше полезных функциональностей, как, например, лямбды, динамическое связывание, асинхронные методы и т.д.

Одним из основных преимуществ С# является то, что он работает на CLR, что упрощает интеграцию с компонентами, написанными на других языках (в частности, CLR-совместимыми языками) и многими проприетарными технологиями Microsoft. Стоит уточнить, что CLR — общеязыковая исполняющая среда — исполняющая среда для байт-кода CIL (MSIL), в который компилируются программы, написанные на .NET-совместимых языках программирования (C#, Managed C++, Visual Basic .NET, F# и прочие). Также важно, что большая часть .NET была стандартизирована, что означает, что она может работать на других платформах.

Также в C# есть автоматический сборщик мусора – очень эффективная система для освобождения памяти от неиспользуемых объектов при выполнении программы, которая действует незаметно и без всякого вмешательства со стороны программиста, при этом она не зависает во время выполнения. Вызывается сборщик мусора вышеупомянутой общеязыковой средой CLR. Кроме этого в C# практически нет проблем с утечкой памяти, поскольку в C# есть сильная резервная память.

Написание кода на C # также дает единый доступ ко всем библиотекам классов .NET фрэймворка, которые достаточно обширны; их использование значительно упрощает реализацию многого функционала. Помимо этого, язык C # богат языковыми конструкциями, полезными при написании кода, такими как делегаты (указатели на функции), перегрузка операторов, универсальные шаблоны (generics), перечисления (enums), методы расширения (extension methods), лямбда выражения (lamda expressions), выражения запросов (query expressions) и так далее.

Кроме того, дизайнеры .NET фрэймворка и C # хорошо подумали о потребностях разработчика. Инструментальная поддержка .NET очень хороша, дизайнеры библиотек очень постарались, чтобы сделать их удобными и простыми для использования с точки зрения разработчика.

Прочитать более подробную информацию о возможностях и синтаксисе языка С# можно в источнике [8] и руководстве по программированию на C# на официальном сайте, к которому я не раз обращалась для реализации приложения.

https://docs.microsoft.com/ru-ru/dotnet/articles/csharp/programming-guide/index

* + 1. **Платформа .NET 4.5**

Фреймворк .NET представляет мощную платформу для создания приложений. Одной их основных черт фреймворка является поддержка нескольких языков. Так как основой платформы является общеязыковая среда исполнения (CLR), .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) -своего рода ассемблер платформы .NET. Благодаря этому есть возможность сделать отдельные модули одного приложения на отдельных языках.

Фреймворк .NET является кроссплатформенным, так версия .NET 4.5.2, используемая в приложении, поддерживается на большинстве современных операционных систем Windows (Windows 8.1/8/7/Vista). А благодаря проекту Mono можно создавать приложения, которые будут работать и на других операционных системах семейства Linux, в том числе на мобильных платформах Android и iOS.

.NET представляет мощную единую для всех поддерживаемых языков библиотеку классов. Эту библиотеку так или иначе задействуют все разработчики, какое бы приложение они ни писали на C#. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF. Для создания веб-сайтов - ASP.NET и т.д.

Часто приложение, созданное на C#, называют управляемым кодом (managed code). Это значит, что данное приложение создано на основе платформы .NET и поэтому управляется общеязыковой средой CLR, которая загружает приложение и при необходимости очищает память. Но есть также приложения, например, созданные на языке С++, которые компилируются не в общий язык CIL, как C# или VB.NET, а в обычный машинный код. В этом случае .NET не управляет приложением. В то же время платформа .NET предоставляет возможности для взаимодействия с неуправляемым кодом. Есть возможность наряду со стандартными классами библиотеки .NET также использовать сборки COM.

Код на C# компилируется в приложения или сборки с расширениями exe или dll на языке CIL. Далее при запуске на стадии выполнения подобного приложения происходит JIT-компиляция (Just-In-Time) в машинный код, который затем выполняется. При этом, поскольку приложение может быть большим и содержать кучу инструкций, в текущий момент времени будет компилироваться лишь та часть приложения, к которой непосредственно идет обращение. Если мы обратимся к другой части кода, то она будет скомпилирована из CIL в машинный код. Притом уже скомпилированная часть приложения сохраняется до завершения работы программы, и в итоге повышается производительность.[5]

**2.3.3 ASP .NET MVC5**

Платформа ASP.NET MVC представляет собой фреймворк для создания сайтов и веб-приложений с помощью реализации паттерна модель-вид-представление MVC.

Концепция паттерна (шаблона) MVC (model - view - controller) предполагает разделение приложения на три компонента:

1. Контроллер (controller) представляет класс, обеспечивающий взаимодействие с пользователем, работу с моделью, а также выбор представления, отображающего пользовательский интерфейс. Он получает вводимые пользователем данные и обрабатывает их, затем в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления.
2. Представления (view**)** являются визуальной частью или пользовательским интерфейсом приложения. Примером может служить html-страница, которую пользователь видит, зайдя на сайт.
3. Модель (model) представляет класс, описывающий логику используемых данных. Объекты моделей часто получают и сохраняют состояние модели в базе данных. Например, объект модели может получать информацию из базы данных, работать с ней, а затем записывать обновленные данные в соответствующую таблицу базы данных.

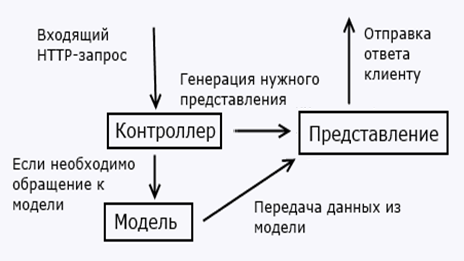
 Общую схему взаимодействия этих компонентов можно представить рисунком 2.

Рисунок 2 - Взаимодействие Модели, Контроллера и Представления

Mодель является независимым компонентом в этой схеме, так как любые изменения контроллера или представления ее не затрагивают. Контроллер и представление же являются относительно независимыми компонентами, и часто их можно изменять независимо друг от друга.

Благодаря своей гибкости и простоте паттерн модель-вид-контроллер стал очень популярным в последнее время, особенно в сфере веб-разработки, хотя следует отметить, что его конкретные реализации и определения могут отличаться. Свою реализацию паттерна представляет платформа ASP.NET MVC. Последняя версия - MVC 5 вышла в 2013 году вместе с релизом Visual Studio 2013, которая предоставляет инструментарий для работы с этим фрэймворком.

Естественное разделение различных ответственностей приложения по независимым друг от друга частям программного обеспечения, которое поддерживается архитектурой MVC, позволяет изначально строить легко сопровождаемые и тестируемые приложения. Однако проектировщики ASP.NET MVC на этом не остановились. Для каждого фрагмента компонентно-ориентированного проекта инфраструктуры они обеспечили структурированность, необходимую для удовлетворения требований модульного тестирования и средств имитации.

В среду Visual Studio добавлен набор мастеров для автоматизированного создания проектов модульного тестирования, которые могут быть интегрированы с такими инструментами модульного тестирования с открытым кодом, как NUnit и xUnit. Следует упомянуть, что можно тестировать представление независимо от контроллера, если, допустим, важна именно визуальная часть или фронтэнд. Либо можно сосредоточиться на бэкэнде и тестировать контроллер.

Основная и главная причина использования ASP.NET MVC на проекте – это его абсолютная расширяемость. В этом плане ASP.NET MVC – это действительно каркас, который может быть заполнен так, как нужно разработчику. Можно сравнить его с наброском или эскизом, который зарисовать и доделать предоставляют самому разработчику. В ASP.NET MVC есть возможность переопределить действие механизма на любом этапе от обработки запроса, до отправки результата. Таким образом, сила этого фрэймворка в его возможностях и потенциале к расширяемости.

Более подробно о платформе ASP.NET MVC можно прочитать в официальной документации https://msdn.microsoft.com/ru-ru/library/dn448362(v=vs.118).aspx и в источнике [8].

**2.3.4 Entity Framework**

Для работы с данными в приложении используется фреймворк Entity Framework – официально рекомендуемая и поддерживаемая Microsoft технология доступа к базам данных. Преимущество этого фреймворка состоит в том, что он позволяет абстрагироваться от структуры конкретной базы данных, от написания sql-запросов, и вести все операции с данными через модель. Это позволяет полностью сосредоточиться на логике приложения.

Entity Framework (EF) — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Аббревиатура ORM расшифровывается как Object–Relational Mapping, что в переводе на русский язык значит Объектно-Реляционное Отображение. Как следует из названия, основной задачей ORM является установка соответствия между объектами, используемыми в приложении, и таблицами, хранящимися в реляционных базах данных. При этом решается также ряд таких задач, как: создание SQL запросов, несоответствие типов, различные типы связей между таблицами, отсутствие классов, отсутствие наследования, сокрытие идентификатора id, абстракция используемой базы данных.

В процессе работы с ORM библиотекой приложение оперирует привычными ему объектами. При этом для хранения информации могут быть использованы различные реляционные системы управления базами данных: SQL Server, SQL Server Express, SQL Server Compact, mySQL и т.д. Это вносит дополнительный уровень гибкости в архитектуру приложения.Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Для облегчения построения web-решений используется как ADO.NET Data Services (Astoria), так и связка из Windows Communication Foundation и Windows Presentation Foundation, позволяющая строить многоуровневые приложения, реализуя один из шаблонов проектирования MVC, MVP или MVVM.

Entity Framework поддерживает подход "Code first", который предполагает сохранение или извлечение информации из БД на SQL Server без создания схемы базы данных или использования дизайнера в Visual Studo. Наоборот, мы создаем обычные классы, а Entity Framework уже сам определяет, как и где сохранять объекты этих классов. Для подключения к базе данных через Entity Framework, нам нужен посредник **- контекст данных**. Контекст данных представляет собой класс, производный от класса DbContext. Контекст данных содержит одно или несколько свойств типа DbSet<T>, где T представляет тип объекта, хранящегося в базе данных.

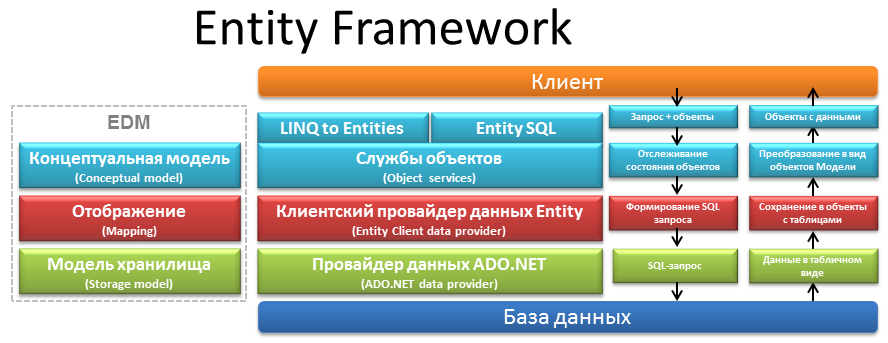
 Схема архитектуры библиотеки представлена на рисунке 4.

Рисунок 4 – Архитектура Entity Framework

Ниже приведу краткое описание назначения основных составляющих архитектуры.

Модель данных Entity (EDM или Entity Data Model) это модель, описывающая отношение клиентских объектов и таблиц, расположенных в базе данных. Можно выделить следующие её составляющие:

1. Концептуальная модель (Conceptual model) – содержит описание классов клиентской модели и взаимоотношения между ними.
2. Модель хранилища (Storage model) – аналогична концептуальной модели, но описывает таблицы, расположенные в реляционной базе данных.
3. Отображение (Mapping) – содержит схему соответствия между указанными выше моделями.

На слое службы объектов (Object Services) клиент может воспользоваться любым из двух поддерживаемых языков для получения необходимой информации: Entity SQL и LINQ to Entities.

Запросы передаются в слой службы объектов, который отвечает за взаимодействие с объектами клиентской части. Здесь они преобразуются в деревья команд (command tree). Кроме того, осуществляется контроль текущего состояния объектов. Это необходимо для сохранения сделанных в них изменений. Кроме того, данный слой отвечает за преобразование данных, передаваемых клиенту от базы данных. При этом объекты с табличной структурой преобразуются в экземпляры классов концептуальной модели.

Слой клиентского провайдера данных используется для взаимодействия с базой данных. Для упрощения архитектуры, он не обращается к ней напрямую, а использует провайдера данных ADO.NET. При получении от службы объектов дерева команд, данный слой создает SQL запрос. Для этого используется все составляющие, входящие в модель данных Entity. После чего результат передается на выполнение в базу данных, используя ADO.NET. При получении результата, слой клиентского провайдера данных преобразует его из простой табличной формы в специальные объекты и передает далее в службы объектов для окончательной обработки.

Слой провайдера данных ADO.NET (ADO.NET data provider) используется для непосредственного обращения к реляционной системе управления базами данных.

Подробнее об Entity Framework можно прочитать в официальной документации https://msdn.microsoft.com/en-us/library/bb399572(v=vs.110).aspx и источнике [8].

**2.3.5 Oracle Database**

Oracle Database или Oracle RDBMS — объектно-реляционная система управления базами данных компании Oracle.

Oracle Database — одна из самых популярных БД, используемых в настоящее время для разработки ПО, так как она функционирует на большинстве платформ. В том числе на больших ЭВМ, UNIX-серверах, персональных компьютерах и т. д. Другой важной характеристикой является поддержка Oracle всех возможных вариантов архитектур, в том числе симметричных многопроцессорных систем, кластеров, систем с массовым параллелизмом и т. д. Очевидна значимость этих характеристик для крупномасштабных организаций, где эксплуатируется множество компьютеров различных моделей и производителей. В таких условиях фактором успеха является максимально возможная типизация предлагаемых решений, ставящая своей целью существенное снижение стоимости владения программным обеспечением. Унификация систем управления базами данных —один из наиболее значимых шагов на пути достижения этой цели.

Поддержка Oracle большинства популярных компьютерных платформ и архитектур достигается за счет жесткой технологической схемы разработки кода СУБД. Разработку серверных продуктов выполняет единое подразделение корпорации Oracle, изменения вносятся централизованно. После этого все версии подвергаются тщательному тестированию в базовом варианте, а затем переносятся на все платформы, где также детально проверяются. Возможность переноса Oracle обеспечивается специфической структурой исходного программного кода сервера баз данных. Приблизительно 80% программного кода Oracle — это программы на языке программирования C, является платформо-независимым. Примерно 20% кода, представляющее собой ядро сервера, реализовано на машинно-зависимых языках; и эта часть кода, разумеется, переписывается для различных платформ.

Однa из отличительных особенностей сервера Oracle — возможность хранения и обработки различных типов данных. Данная функциональность интегрирована в ядро СУБД и поддерживается модулем interMedia в составе Oracle Database. Он обеспечивает работу с текстовыми документами, включая различные виды поиска, в том числе контекстного; работу с графическими образами более 20–ти форматов; работу с аудио и видеоинформацией.

СУБД Oracle не только предоставляет расширенный набор встроенных типов данных, но и позволяет за счет использования Object Option конструировать новые типы данных со спецификацией методов доступа к ним. Это означает фактически, что разработчики получают в руки инструмент, позволяющий строить структурированные типы данных, непосредственно отображающие объекты предметной области.

Oracle включает в себя очень много различных компонентов и модулей, в том числе используемые в разработанной системе расширения Holt-Winters Additive и Holt-Winters Multiplicative для Oracle Hyperion Planning Predictive Planning Smart View приложения для сравнения полученных в разработанной системе прогнозов с прогнозами, построенными приложениями Oracle, в рамках тестирования реализованного метода прогнозирования продаж.[8]

**2.3.6 AngularJS**

AngularJS — JavaScript-фреймворк с открытым исходным кодом. Предназначен для разработки одностраничных приложений (SPA). Его цель — расширение браузерных приложений на основе MVC-шаблона, а также упрощение тестирования и разработки.

Фреймворк работает с HTML, содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON-данных.

AngularJS спроектирован с убеждением, что декларативное программирование лучше всего подходит для построения пользовательских интерфейсов и описания программных компонентов, в то время как императивное программирование отлично подходит для описания бизнес-логики. Фреймворк адаптирует и расширяет традиционный HTML, чтобы обеспечить двустороннюю привязку данных для динамического контента, что позволяет автоматически синхронизировать модель и представление. В результате AngularJS уменьшает роль DOM-манипуляций и улучшает тестируемость.  
 Двустороннее связывание данных в AngularJS является наиболее примечательной особенностью и уменьшает количество кода, освобождая сервер от работы с шаблонами. Вместо этого, шаблоны отображаются как обычный HTML, наполненный данными, содержащимися в области видимости, определенной в модели. Сервис $scope в Angular следит за изменениями в модели и изменяет раздел HTML-выражения в представлении через контроллер. Кроме того, любые изменения в представлении отражаются в модели. Это позволяет обойти необходимость манипулирования DOM и облегчает инициализацию и прототипирование веб-приложений.

Причины использования AngularJS:

1. Отделение DOM-манипуляции от логики приложения, что улучшает тестируемость кода.
2. Разделение клиентской и серверной стороны, что позволяет вести разработку параллельно, получать структурированный код, который уменьшит сложность тестирования.
3. Проведение разработчика через весь путь создания приложения: от проектирования пользовательского интерфейса, через написание бизнес-логики, к тестированию.

AngularJS придерживается MVC-шаблона проектирования и поощряет слабую связь между представлением, данными и логикой компонентов. Используя внедрение зависимости, Angular переносит на клиентскую сторону такие классические серверные службы, как видозависимые контроллеры. Следовательно, уменьшается нагрузка на сервер и веб-приложение становится легче.[8]