**1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

* 1. **Особенности «Скандинавских» торгов**

Шаг ставки на скандинавском аукционе является фиксированным и составляет очень незначительную величину в сравнении с вероятной стоимостью товара. Скажем, на лот, стоимость которого на рынке составляет 300-400 рублей, стартовая цена может быть установлена всего в 10 рублей, при этом участники торгов имеют возможность повышать ставки всего на 25 копеек.

Следует отметить, что наиболее важное обстоятельство скандинавского аукциона заключается в том, что, делая ставку в 25 копеек, участник должен её оплатить, и плата за ставку составляет 1 рубль. Таким образом, ведя азартную борьбу за лот с другими участниками, сделав, скажем, 20 ставок по 25 копеек, на самом деле участник, даже не выиграв лот, в итоге тратит 20\*1=20 рублей только за возможность делать эти ставки. Прибыль скандинавского аукциона составляет не процент с выигрышной суммы, а именно эта плата за ставку.

Другое важное отличие скандинавского аукциона от обычного состоит в том, что в первом не существует фиксированного периода торгов. По истечении определённого времени каждая сделанная пользователем ставка немного продлевает срок торгов, на период от нескольких секунд до нескольких минут. И, если за это время никто другой не сделает новой ставки, тот, кто сделал последнюю, считается победителем.

**1.2 Сравнение аналогов**

Перед тем, как приступить к реализации проекта, следует проанализировать существующие на данный момент системы подобного характера и тематики. С помощью этого мы можем сделать верные выводы о том, как необходимо создавать собственную систему на основе достоинств и недостатков рассмотренных ресурсов.

Наиболее популярные веб-ориентированные приложения скандинавских аукционов, которые действуют на территории СНГ:

* Gagen.ru – первый российский аукцион, работающий по системе скандинавских аукционов. Есть несколько непродуманных моментов, дисбаланс из-за автоставок и бесплатной раздаче администрацией ставок своим партнерам. Но играть и выигрывать можно. Есть форум, для решения любых вопросов.
* Getbuy.ru – Красивый сайт, много аукционов, но имеется наличие ботов или подставных лиц.
* Vauctione.ru – Молодой и красивый сайт. Недостатки сайта: нет помощи, форума, официального договора публичной оферты.
* Internetlot.ru – Достаточно сырой и не продуманный сайт, яркий пример одной из многих копий Gagen.ru.
* minilot.ru – Копия Gagen.ru, но там организаторы пытаются предоставить честные условия.
* tvoypriz.ru – Неплохой аукцион. Но, полностью отсутствует прозрачность торгов, нет чата или форума. Небольшие технические возможности.
* Goodwin.by – Сайт от белорусских разработчиков. Красивый и понятный дизайн, в сочетании с большими возможностями, тщательной продуманностью торгов, отсутствием ботов и наличии средств коммуникации(форум) делают его лучшим не только в Беларуси, но пожалуй и на всей территории СНГ.

## 1.3 Архитектура клиент-сервер

Важным вопросом в разработке веб-приложений является их архитектура. Наиболее эффективную работу приложений обеспечивает архитектура «клиент-сервер».

Особенность данной архитектуры заключается в том, что само веб-приложение находится и выполняется на сервере, клиент при этом получает только результаты работы. Работа приложения основывается на получении запросов от пользователя (клиента), их обработке и выдачи результата. Передача запросов и результатов их обработки происходит через Интернет.



Рисунок 1.1 Архитектура клиент-сервер

Отображением результатов запросов, а также приемом данных от клиента и их передачей на сервер обычно занимается специальное приложение – браузер (InternetExplorer, Mozilla, Opera и т. д.). Как известно, одной из функций браузера является отображение данных, полученных из Интернета, в виде страницы, описанной на языке HTML, следовательно, результат, передаваемый сервером клиенту, должен быть представлен на этом языке.

На стороне сервера веб-приложение выполняется специальным программным обеспечением (веб-сервером), который и принимает запросы клиентов, обрабатывает их, формирует ответ в виде страницы, описанной на языке HTML, и передает его клиенту.

В процессе обработки запроса пользователя веб-приложение компонует ответ на основе исполнения программного кода, работающего на стороне сервера, веб-формы, страницы HTML, другого содержимого, включая графические файлы. В результате, как уже было сказано, формируется HTML-страница, которая и отправляется клиенту. Получается, что результат работы веб-приложения идентичен результату запроса к традиционному веб-сайту, однако, в отличие от него, веб-приложение генерирует HTML-код в зависимости от запроса пользователя, а не просто передает его клиенту в том виде, в котором этот код хранится в файле на стороне сервера. То есть веб-приложение динамически формирует ответ с помощью исполняемого кода – так называемой исполняемой части.

За счет наличия исполняемой части, веб-приложения способны выполнять практически те же операции, что и обычные Windows-приложения, с тем лишь ограничением, что код исполняется на сервере, в качестве интерфейса системы выступает браузер, а в качестве среды, посредством которой происходит обмен данными, – Интернет. К наиболее типичным операциям, выполняемым веб-приложениями, относятся:

* прием данных от пользователя и сохранение их на сервере;
* выполнение различных действий по запросу пользователя: извлечение данных из базы данных (БД), добавление, удаление, изменение данных в БД, проведение сложных вычислений;
* аутентификация пользователя и отображение интерфейса системы, соответствующего данному пользователю;
* отображение постоянно изменяющейся оперативной информации.

Основными достоинствами архитектуры «клиент-сервер» являются:

* возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети, что позволяет упростить обслуживание вычислительной системы;
* все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов;
* на сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;
* позволяет объединить различные клиенты;
* использовать ресурсы одного сервера часто могут клиенты с разными аппаратными платформами, операционными системами и т.п.

Среди недостатков можно выделить необходимость квалифицированного профессионала для администрирования данной системы. В случае использования централизованной системы, неработоспособность основного сервера может сделать неработоспособным всё приложение. Также, немаловажным фактором является высокая стоимость оборудования.

## 1.4 REST-сервис

REST – это набор архитектурных принципов и стиль проектирования приложений, ориентированный на создание сетевых систем, в основе которых лежат механизмы для описания и обращения к ресурсам.



Рисунок 1.2 REST-сервис

Примером такой системы может служить WorldWideWeb. В REST определяется строгое разделение ответственности между компонентами клиент-серверной системы, облегчающее реализацию необходимых актеров. Другой целью REST является упрощение семантики взаимодействия компонентов сетевых систем, что позволяет улучшить масштабируемость и повысить производительность. В основу REST заложен принцип автономности запросов, означающий, что запросы, обрабатываемые клиентом или сервером, должны включать всю контекстную информацию, необходимую для их понимания.

При работе REST-систем для обмена данными стандартных медиа-типов используется минимальное количество запросов. REST-системы используют URI (универсальные идентификаторы ресурсов) для поиска и получения доступа к представлениям необходимых ресурсов. В течение последних нескольких лет разработчики создавали REST- сервисы для своих ASP.NET-приложений, используя самые разнообразные технологии. Архитектура REST отличается своей простотой, требуя от приложений обеспечить только возможность приема сообщений с HTTP- заголовками. Эта функция легко реализуется простыми контроллерами в ASP.NET Web Api.

**1.5 Шаблон проектирования MVC**

MVC (Model-View-Controller) – схема использования шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.

Впервые паттерн MVC появился в языке SmallTalk. Разработчики должны были придумать архитектурное решение, которое позволяло бы отделить графический интерфейс от бизнес логики, а бизнес логику от данных.

Популярность данной структуры в веб приложениях сложилась благодаря её включению в две среды разработки, которые стали очень популярными: Struts и RubyonRails. Эти две среды разработки наметили пути развития для сотен рабочих сред, созданных позже.

Идея, которая лежит в основе конструкционного шаблона MVC, очень проста: нужно чётко разделять ответственность за различное функционирование в наших приложениях. Приложение разделяется на три основных компонента, каждый из которых отвечает за различные задачи (принцип единой ответственности).

Контроллер управляет запросами пользователя (получаемые в виде запросов HTTP GET или POST, когда пользователь нажимает на элементы интерфейса для выполнения различных действий). Его основная функция – вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.

Модель – это данные и правила, которые используются для работы с данными, которые представляют концепцию управления приложением. В любом приложении вся структура моделируется как данные, которые обрабатываются определённым образом. Что такое пользователь для приложения – сообщение или книга? Только данные, которые должны быть обработаны в соответствии с правилами (дата не может указывать в будущее, email должен быть в определённом формате, имя не может быть длиннее Х символов, и так далее).



Рисунок 1.3 - Шаблон проектирования MVC

Модель даёт контроллеру представление данных, которые запросил пользователь (сообщение, страницу книги, фотоальбом, и тому подобное). Модель данных будет одинаковой, вне зависимости от того, как мы хотим представлять их пользователю. Поэтому мы выбираем любой доступный вид для отображения данных.

Модель содержит наиболее важную часть логики нашего приложения, логики, которая решает задачу, с которой мы имеем дело (форум, магазин, банк, и тому подобное). Контроллер содержит в основном организационную логику для самого приложения (очень похоже на ведение домашнего хозяйства).

Стоит отметить, что в данном случае описан подход с «толстой» моделью и «тонким» контроллером. Очень часто практикуется подход наоборот – «тонкая» модель и «толстый» контроллер – когда бизнес-логика заключена в контроллере, а модель является лишь данными.

Вид обеспечивает различные способы представления данных, которые получены из модели. Он может быть шаблоном, который заполняется данными. Может быть несколько различных видов, и контроллер выбирает, какой подходит наилучшим образом для текущей ситуации.

Веб-приложение обычно состоит из набора контроллеров, моделей и видов. Контроллер может быть устроен как основной, который получает все запросы и вызывает другие контроллеры для выполнения действий в зависимости от ситуации.

Самое очевидное преимущество, которое мы получаем от использования концепции MVC – это чёткое разделение логики представления (интерфейса пользователя) и логики приложения.

Поддержка различных типов пользователей, которые используют различные типы устройств является общей проблемой наших дней. Предоставляемый интерфейс должен различаться, если запрос приходит с персонального компьютера или с мобильного телефона. Модель возвращает одинаковые данные, единственное различие заключается в том, что контроллер выбирает различные виды для вывода данных.

Помимо изолирования представления от логики приложения, концепция MVC существенно уменьшает сложность больших приложений. Код получается гораздо более структурированным, и, тем самым, облегчается поддержка, тестирование и повторное использование решений.

## 1.6 Выбор модели жизненного цикла

Так в проекте требования будут изменяться по ходу разработки, то подходящие модели жизненного цикла приведены ниже.

Таблица 1.1 – Модели ЖЦ

|  |  |
| --- | --- |
| Вид модели | Качество данной модели |
| Инкрементная | + Добавление функции с каждым инкрементом.  - Возможны ситуации, требующие добавления сразу нескольких взаимосвязанных функций. |
| На основе ранее созданных компонентов | + Снижает время разработки.  - Искажаются требования. |
| Спиральная | +Быстрое получение результата.  +Изменяющиеся требования — не проблема.  - Усложнённая структура разработки. |

Из таблицы видно, что наиболее подходящей моделью жизненного цикла является спиральная модель, так как в ходе проекта ожидается возможное изменение требований.

Отличительной особенностью этой модели является специальное внимание к рискам, влияющим на организацию жизненного цикла. Каждый виток спирали соответствует созданию фрагмента или версии программного обеспечения, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали.

**1.7** **Выбор средств разработки**

Для хранения информации наиболее часто используются реляционные системы управления базами данными. Информация в них представлена в виде таблиц. Они содержат простые типы данных и, при необходимости, могут быть взаимосвязаны между собой.

Приложения, как правило, оперируют экземплярами классов (объектами), которые являются абстракциями объектов реального мира. Такой подход более удобен с точки зрения бизнес-логики. Кроме того, это дает такие преимущества при разработке как проверки типов, скорость работы кода и пр.

Таким образом, для использования реляционной базы данных необходимо использовать конвертер, который будет преобразовывать объекты в табличный вид и наоборот. При этом он должен учитывать различия в организации информации в обоих форматах. Например, в таблицах все строки отличаются друг от друга.

Для использования в разработке базы данных была выбрана база данных – MSSQLServer. MSSQLServer имеет прекрасные характеристики по производительности на большом объеме данных, к тому же она имеет большое количество настроек для оптимизации.

В качестве языка программирования баз данных был выбран язык запросов TSQL, т.к. этот язык используется совместно с MSSQLServer.

В качестве языка программирования серверной части был выбран C# и платформа ASP.NET Core которая представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

ASP.NET Core является полностью opensource-фреймворком. Все исходные файлы фреймворка доступны на [GitHub](https://github.com/aspnet/).

ASP.NET Core построен на основе кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS X, Linux. То есть веб-приложения могут быть запущены не только на ОС Windows, но и на Linux и Mac OS. А для развертывания веб-приложения можно использовать традиционный IIS, либо кросс-платформенный веб-сервер Kestrel.

Хотя ASP.NET Core преимущественно нацелено на использование .NET Core, но фреймворк также может работать и с полной версией фреймворка .NET.

Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер Nuget. Кроме того, в отличие от предыдущих версий платформы нет необходимости использовать библиотеку System.Web.dll.

ASP.NET Core включает в себя фреймворк MVC, который объединяет функциональность MVC, Web API и Web Pages. В предыдущих версии платформы данные технологии реализовались отдельно и поэтому содержали много дублирующей функциональности. Сейчас же они объединены в одну программную модель ASP.NET Core MVC. А Web Forms полностью ушли в прошлое.

Кроме объединения вышеупомянутых технологий в одну модель в MVC был добавлен ряд дополнительных функций.

Одной из таких функций являются тэг-хелперы (tag helper), которые позволяют более органично соединять синтаксис html с кодом С#.

ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. Может использоваться встроенная реализация этих компонентов, либо можно расширить их с помощью механизма наследования, либо вовсе создать и применять собственные компоненты с собственным функционалом.

Также было упрощено управление зависимостями и конфигурирование проекта. Фреймворк теперь имеет свой легковесный контейнер для внедрения зависимостей, и больше нет необходимости применять сторонние контейнеры, такие как Autofac, Ninject.

В качестве инструментария разработки могут использоваться выпуски Visual Studio, начиная с версии Visual Studio 2015. Кроме того, приложения могут создаваться в среде Visual Studio Code, которая является кросс-платформенной и может работать как на Windows, так и на Mac OS X и Linux.

При разработке в Visual Studio 2015/2017 проекты приложений имеют встроенную поддержку с такими популярными инструментами, как Bower, Grunt, Gulp, который позволяют управлять скриптами JavaScript и стилями CSS, автоматизировать и оптимизировать процесс веб-разработки.

Для разработки клиентской части был выбран фреймворк Angular v4.0.

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

Одной из ключевых особенностей Angular является то, что он использует в качестве языка программирования TypeScript.

Angular 2 предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация и так далее.

**1.8 Выбор платёжной системы**

Для участия в аукционе пользователь должен иметь биды. Их он может получить купив с помощью оплаты товара в сети интернет. Для начала рассмотрим существующие системы оплаты в сети Интернет.  
Традиционные методы оплаты, включая наличные деньги, банковские переводы, чеки, пластиковые карточки, изобретены задолго до возникновения электронной коммерции. Поэтому нет ничего удивительного в том, что они не полностью соответствуют ее потребностям. Безусловно, при покупке обычного физического товара деньги можно взять с покупателя при доставке. Но если приобретается цифровой товар или информация, обязательно должен быть способ оплатить покупку прямо на месте, то есть на сайте продавца. Вот почему в электронной коммерции чрезвычайно остро стоит вопрос разработки универсального способа оплаты покупок в Интернет, который бы позволил делать дешевые и безопасные платежи в режиме реального времени. К способам оплаты покупок в Интернет, проводимым электронным путём, можно отнести следующие:

- Оплата со счета мобильного телефона (m-commerce);

- Оплата по картам предоплаты (скретч-картам);

- Оплата электронными деньгами;

- Оплата платежными картами.

- Оплата со счета мобильного телефона (m-commerce).

Мобильная коммерция это использование мобильных портативных устройств для общения, развлечения, получения и передачи информации, совершения транзакций через общественные и частные сети.

Сегодня на рынке появилась прекрасная возможность оплачивать услуги или товары различных компаний через мобильный телефон. Вы можете делать покупки в Интернет - магазинах, оплачивать коммунальные платежи и даже переводить деньги по банковским реквизитам. Вне зависимости оттого, что и как Вы оплачиваете, процесс оплаты занимает всего пару минут, а сам платеж происходит практически мгновенно.

Оплата услуги или товара происходит с помощью посылки секретного сообщения (SMS), включающего в себя цепочку информации типа номер счёта владельца телефона/сумма к оплате или кодовый номер товара.

В случае если размер совершаемого платежа превышает остаток средств на текущей карте, необходимо активировать новую карту. При этом неиспользованные остатки денежных средств с предыдущих карт прибавляются к номиналу активированной карты.

Важно отметить, что для того, чтобы Вы могли использовать эту функцию ваш телефон должен быть оснащён функцией WAP или некоторым собственным микробраузером. Мобильная коммерция делает пользователя еще более независимым, не привязанным к стационарным устройствам, предоставляя все вышеперечисленные возможности при наличии одного только мобильного телефона или карманного компьютера.

Цифровые деньги являются основой платежной системы Интернет. Они могут быть классифицированы по видам организации их функционирования, по уровню безопасности, а также по способу расчета.

Цифровую наличность можно представить себе как файлы-жетоны, заменяющие наличные деньги. Продавцы и покупатели могут свободно обмениваться этими "монетами" по сети, оплачивая ими товары и услуги. Для указанной цели участники системы устанавливают у себя на компьютерах особую программу - "электронный кошелек", который обеспечивает учет и передачу жетонов, а также проверку их подлинности. Цифровые деньги могут неограниченно долго обращаться в сети, но также могут в любое время быть обменены на настоящие деньги у организаторов системы или в банках, участвующих в ней.

В данном дипломном проекте будет использоваться платёжная система WebMoney.