Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

|  |
| --- |
| *К защите допустить*: |
| Заведующая кафедрой ПОИТ |
| \_\_\_\_\_\_\_\_\_\_\_\_Н. В. Лапицкая |

# Пояснительная записка

к дипломному проекту

на тему

**ВЕБ-ОРИЕНТИРОВАННОЕ ПРОГРАММНОЕ СРЕДСТВО «СКАНДИНАВСКИЙ АУКЦИОН»**

БГУИР ДП 1-40 01 01 03 087 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | А. Л. Чечётко |
| Руководитель |  | А. И. Парамонов |
| Консультанты: |  |  |
| *от кафедры ПОИТ* |  | А. А. Ктотов |
| *по экономической части* |  | Г. А. Харитонов |
|  |  |  |
| Нормоконтролер |  | Г. В. Данилова |
|  |  |  |
| Рецензент |  | И. П. Иванов |

Минск 2017

**СОДЕРЖАНИЕ**

**ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ**

Жизненный цикл программного средства - период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Программное средство - компьютерная программа или логически связанная совокупность программ, предназначенная для автоматизации в определённой области профессиональной деятельности.

Фреймворк - программное обеспечение, облегчающее разработку и объединение различных компонентов большого программного проекта.

URL - Uniform Resource Locator (уникальный адрес, определяющий однозначно конкретную страницу в сети).

SERP - Search Engine Results Page (веб-страница, генерируемая поисковой системой в ответ на поисковый запрос пользователя).

DAL - Data Access layer (уровень доступа к данным, хранит модели, описывающие используемые сущности).

Layer - логически обособленный слой в архитектуре приложения.

Тег - идентификатор для категоризации, описания, поиска данных и задания внутренней структуры.

Хештег - тег, которому предшествует символ #.

CLI - Common Language Infrastructure (спецификация общеязыковой инфраструктуры).

CLR - Common Language Runtime (исполняющая среда для байт-кода CIL (MSIL), в который компилируются программы, написанные на .NET).

ORM - Object-Relational Mapping (технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования).

БД - база данных.

**ВВЕДЕНИЕ**

Стремление современного человека к материальному благополучию порой приобретает несколько вычурные формы. Желание экономить на покупке дорогих вещей, конечно, вполне оправдано, если только такая экономия не превращается в навязчивую идею и не заставляет человека совершать поступки, о которых, возможно, он впоследствии пожалеет.

Попытки сэкономить там, где другой платит полную цену, активно используются ловкими предпринимателями, исповедующими принцип «цель оправдывает средства». Балансируя на грани дозволенного, подступая к краю законодательства, они предлагают доверчивым обывателям различные способы, соблазняющие возможностью быстро поправить свои финансовые дела.

Последнее время сеть заполонили так называемые скандинавские аукционы, где можно за бесценок приобрести достаточно дорогие вещи. «Скандинавским» этот вид аукционов называется лишь формально - на деле такая система торгов не имеет к Скандинавии никакого отношения. В западных странах этот вид аукционов имеет определённое распространение, но английское название гораздо более точно отражает суть процесса торгов – «Penny bid auction».

Обычный аукцион, как правило, не ограничивает пользователя в размере совершаемой ставки. Скандинавский аукцион похож на обычный лишь терминологией. Основные принципы проведения торгов в этих системах торгов радикально отличаются. Как иногда случается на обычном аукционе, в скандинавском аукционе начальная ставка на тот или иной товар также устанавливается значительно более низкой, чем его рыночная стоимость, но в проведении торгов заложены существенные отличия.

Целью дипломного проекта является создание веб-ориентированного приложения, в котором пользователи смогут разыгрывать товары различного назначения по системе скандинавского аукциона.

**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:

1. 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.
2. Getbuy.ru – Красивый сайт, много аукционов, но имеется наличие ботов или подставных лиц.
3. Vauctione.ru – Молодой и красивый сайт. Недостатки сайта: нет помощи, форума, официального договора публичной оферты.
4. Dominanta.ru – Достаточно сырой и не продуманный сайт (рисунок 1.1), яркий пример одной из многих копий Gagen.ru.

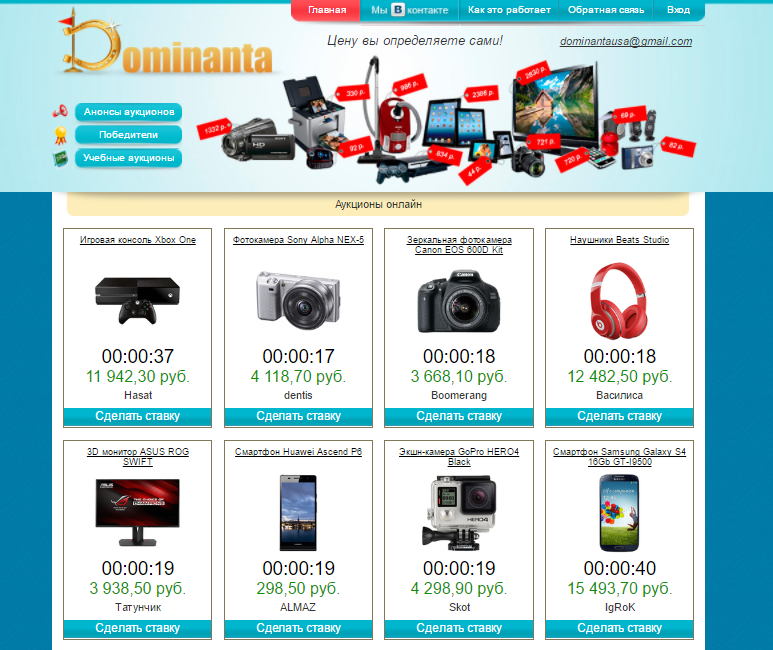


Рисунок 1.1 - Веб-сайт «Dominanta»

1. minilot.ru – Копия Gagen.ru, но там организаторы пытаются предоставить честные условия.
2. tvoypriz.ru – Неплохой аукцион. Но, полностью отсутствует прозрачность торгов, нет чата или форума. Небольшие технические возможности.
3. 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 и пожалуй 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 Model View Controller**

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 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.

**2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

**2.1 Описание функциональности ПС**

Для описания системы на концептуальном уровне была выбрана диаграмма вариантов использования. Она позволяет наглядно отобразить отношения между актёрами и прецедентами. На рисунке 2.1 представлена диаграмма вариантов использования для разрабатываемого программного средства.

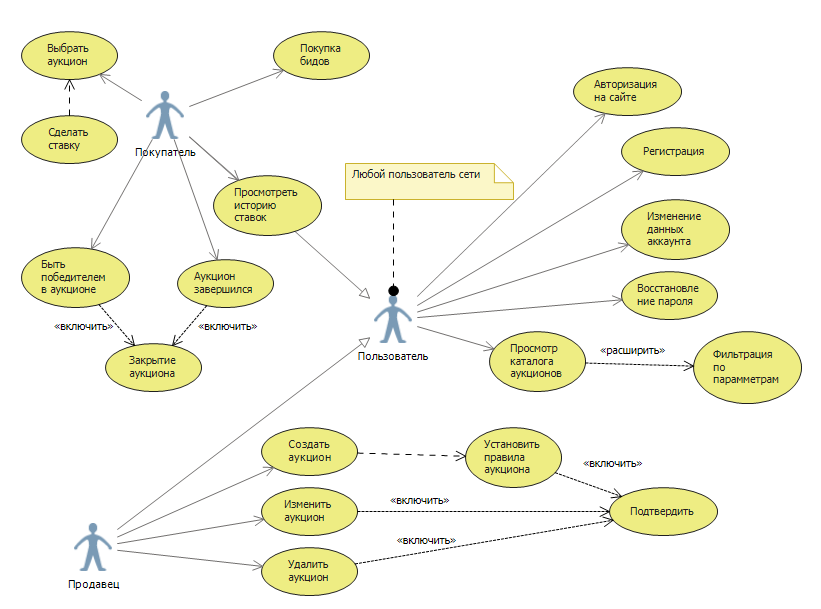


Рисунок 2.1 – UML Use-case -диаграмма взаимодействия пользователя

На диаграмме можно выделить два основных составляющих элемента – актёр и прецедент.

Актёр – это человек, имеющий некоторую роль в системе. Каждой роли соответствует свой набор функций.

Прецедент – это часть функциональности системы, которая определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой.

Как видно из данной диаграммы, в системе будет несколько актёров

Для работы в системе пользователь должен быть зарегистрирован и авторизирован на сайте для этого есть соответствующие действия в системе. Каждый пользователь может просматривать выставленные товары для аукциона и отфильтровать их отображение по заданным параметрам.

Условно можно сделать разделение пользователя на аукционера и аукциониста. Аукционер – участник аукциона. Чтобы сделать ставку аукционист должен заранее приобрести биды. Аукционист создаёт аукцион, а также он может изменять условия его или отменить проведение аукциона.

Любой зарегистрировавшийся пользователь, имеет определённые права, и доступ к тем или иным функциям веб-портала. Ниже представлено подробное описание прецедентов пользователя:

1. Регистрация. Для получения пользователем полномочия на вход в

систему под своим логином и паролем, ему необходимо создать аккаунт в веб портале. Как видно из диаграммы все прецеденты, направленные на создание/редактирование, находятся в связи “зависит” с прецедентом авторизация, который в свою очередь зависит от регистрации.

1. Авторизация. Для получения пользователем прав на участие или добавление аукциона, системе необходимо определить пользователя по введённым логину и паролю. После подтверждения авторизации пользователь получает права аукционера или аукциониста.
2. Управление аукционом. Если пользователь имеет роль аукционист, то он может создавать аукционы. Также аукционист может изменить или удалить созданный им аукцион.
3. Покупка бидов. Пользователь с ролью аукционер может покупать биды с помощью платёжной системы WebMoney. Бид используются в качестве ставки в аукционе.
4. Участие в аукционе. Пользователь с ролью аукционер может делать ставки при наличии бидов в аукционе.
5. Просмотр истории торгов. Пользователь может просматривать историю уже окончившихся аукционов.
6. Просмотр каталога аукционов. Пользователь, может не заходя в систему просматривать доступные аукционы.

## **2.2 Спецификация функциональных требований.**

На основании анализа исходных данных для проектируемого программного средства можем выделить, что основной целью является создание качественного программного продукта, позволяющего проводить аукционы в режиме реального времени через интернет.

В ходе разработки будут реализованы следующие возможности:

1. Аутентификация. Пользователю будут доступны следующие функции:
   1. регистрация;
   2. авторизация;
   3. анонимный доступ.
2. Управление аукционом. Пользователю будут доступны следующие функции:
   1. создание аукциона;
   2. редактирование аукциона;
   3. удаление аукциона;
3. Участие в аукционе. Пользователю будут доступны следующие функции:
   1. просмотр аукционов;
   2. покупка бидов;
   3. сделать ставку;
   4. просмотр истории проведения аукциона;
4. Навигация по сайту. Пользователю будут доступны следующие функции:
   1. пагинация;
   2. читабельная строка запроса;
   3. фильтрация аукционов.
5. Управление личным аккаунтом. Пользователю будут доступны следующие функции:
   1. изменение данных аккаунта;
   2. сброс пароля;

**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**3.1 Архитектура программного модуля**

Архитектура программного обеспечения (software architecture) – это представление, которое даёт информацию о компонентах ПО, обязанностях отдельных компонентов и правилах организации связей между компонентами. Необходимость архитектуры обоснована сложностью программного обеспечения. Продуманная архитектура облегчает разработку и дальнейшее развитие ПО. Она служит базисом, каркасом создаваемой системы, интегрируя отдельные компоненты и создавая высокоуровневую модель системы.

Набор принципов, используемых в архитектуре, формирует архитектурный стиль (software architecture style). Применение архитектурного стиля сродни употреблению шаблона проектирования, но не на уровне компонента (модуля или класса), а на уровне всей создаваемой системы ПО. Как и шаблоны проектирования, архитектурные стили упрощают коммуникацию разработчиков и предлагают готовые решения целого класса абстрактных проблем. В таблице 3.1 представлено короткое описание основных архитектурных стилей.

Многоуровневая архитектура (multilayered architecture) сосредоточена на иерархическом распределении отдельных частей системы при помощи эффективного разделения отношений.

Таблица 3.1 – Основные архитектурные стили

|  |  |
| --- | --- |
| Архитектурный стиль | Описание |
| Клиент-серверная модель | Разделение системы на два приложения – клиент и сервер. При работе клиент посылает запросы на обслуживание серверу |
| Компонентная архитектура | Деление системы на компоненты, которые могут быть повторно использованы и не зависят друг от друга. Каждый компонент снабжается известным интерфейсом для  коммуникаций |
| Многоуровневая архитектура | Разделение функций приложения на группы (уровни), которые организованы в виде стекового набора |
| Шина сообщений | Система, которая может посылать и передавать информационные сообщения в определённом формате по общему  коммуникационному каналу. Благодаря этому организуется взаимодействие систем без указаний конкретных получателей сообщений |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Многозвенная архитектура | Разделение функций подобно многоуровневой архитектуре, но группировка происходит не только на логическом, а и на физическом уровне – отдельным группам соответствует отдельный компьютер (сервер, кластер) |
| Объектно-ориентированная  архитектура | Представление системы в виде набора взаимодействующих объектов |
| Выделенное представление | Выделение в системе отдельных групп функций для взаимодействия с пользователями и обработки данных |
| Архитектура, ориентированная на сервисы | Каждый компонент системы представлен в виде независимого сервиса, предоставляющего свои функции по  стандартному протоколу |

Каждая часть соотносится с определённым уровнем (layer), для каждого уровня заданы выполняемые им функции, уровни выстроены в стековую структуру (то есть находятся один поверх другого). Например, типичная многоуровневая архитектура веб-приложения включает уровень представления (компоненты пользовательского интерфейса), уровень бизнес-логики (обработка данных) и уровень доступа к данным [4]. При этом уровень представления считается высшим, за ним идёт уровень бизнес-логики, а за уровнем бизнес-логики – уровень доступа к данным.

Основные принципы многоуровневой архитектуры:

* проектирование чётко устанавливает разграничение функций между уровнями;
* нижние уровни независимы от верхних уровней;
* верхние уровни вызывают функции нижних уровней, но при этом взаимодействуют только соседние уровни иерархии.

Использование многоуровневой архитектуры обеспечивает следующие преимущества:

1. Изоляция. Разработка и обновление ПО могут быть изолированы рамками одного уровня.
2. Производительность. Распределение уровней на отдельные физические компьютеры повышает производительность и отказоустойчивость.
3. Тестируемость. Уровни допускают независимое тестирование.

Многоуровневая архитектура активно применяется при создании бизнес-приложений и сайтов, особенно приложений масштаба предприятия. При этом обычно используется следующий набор уровней (см. рисунок 3.1):

1. Уровень представления (presentation layer) ответственен за взаимодействие с пользователем, ввод и вывод информации.
2. Бизнес-уровень или уровень бизнес-логики (business logic layer) обрабатывает информацию, реализуя конкретные бизнес-правила.
3. Уровень доступа к данным (data access layer) обеспечивает загрузку и сохранение информации, используя источник данных (файл, база данных) или внешний сервис.

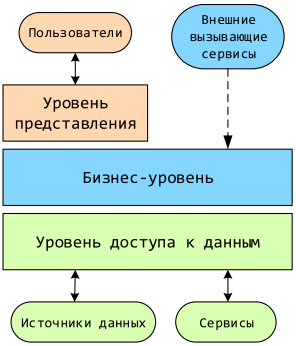


Рисунок 3.1 – Типичные уровни бизнес-приложения

**3.2 Структура программного модуля**

В процессе проектирования при решении всех требуемых задач проект был разделен на относительно большие части, взаимодействующие между собой и выполняющие свои особые функции (см. рисунок 3.2).

Рассмотрим структурные модули:

1. Модуль организации пользовательского интерфейса. При помощи этого модуля пользователь может осуществлять базовую навигацию на сайте: выбирать интересующие аукционы, знакомиться с правилами его проведения, просматривать данные своего аккаунта. Область ответственности этого модуля - обработка ввода информации от пользователя, а также формирование нужных запросов сетевым модулям.



Рисунок 3.2 – Модули приложения

Обобщенная структурная схема программного средства изображена на рисунке 3.3.



Рисунок 3.3 – Обобщенная структурная схема

1. Препроцессор Sass используется для компиляции css-кода ответственного за отображение элементов пользовательского интерфейса (кнопки, списки, текст и т.д.).
2. Модуль сервисных интерфейсов является прослойкой между внутренней логикой приложения и внешней стороной, которая видна пользователю. Обеспечивает возможность гибкой перестройки внутренностей модуля, не затрагивая пользовательский интерфейс.
3. Модели уровня бизнес-логики являются абстракцией первого уровня над моделями базы данных. Они содержат в себе первичные правила и способы хранения и обработки записей более низкого уровня абстракции. Служат вспомогательным инструментом, не допускающим рост сложности работы с БД при развитии приложения.
4. Сервисы уровня бизнес-логики отвечают за передачу данных от БД к пользовательскому интерфейсу и обратно. В дополнении происходит проверка последних. При добавлении функциональных возможностей архитектура легко расширяется с учетом новых требований.
5. Уровень доступа к базе данных. Отвечает за хранение, создание, редактирование информации. Реализован с помощью MSSQL Server с применением *объектно-реляционного отображателя* (object relational mapper – ORM) Entity Framework. Адаптеры являются посредниками между уровнями области определения и распределения данных (domain and data mapping layers). Объекты-клиенты создают описание запроса декларативно и направляют их к объекту-репозиторию (Repository) для обработки. Объекты могут быть добавлены или удалены из репозитория, как будто они формируют простую коллекцию объектов. А код распределения данных, скрытый в объекте Repository, позаботится о соответствующих операциях незаметно для разработчика. В двух словах, паттерн Repository инкапсулирует объекты, представленные в хранилище данных и операции, производимые над ними, предоставляя более объектно-ориентированное представление реальных данных. Repository также преследует цель достижения полного разделения и односторонней зависимости между уровнями области определения и распределения данных.

**3.3 Алгоритмы, положенные в основу программного средства**

**3.3.1** Алгоритм процесса проведения аукциона

Процесс управления аукционами делится на 3 параллельных задачи:

а) Отслеживание аукционов, которые должны начаться в будущем

private void SetWaitingAuctions()

{

var waitedAuctions = \_auctionService.GetAuctions().Where(x => x.TradingStart > DateTime.Now);

foreach (var auction in waitedAuctions)

{

if (!\_waitedAuctions.ContainsKey(auction.Id))

{

var waitedAuction = new WaitedAuction

{

AuctionId = auction.Id,

LastUser = auction.User.Email,

StartPrice = auction.StartPrice,

TradingStart = auction.TradingStart

};

\_waitedAuctions.Add(auction.Id, waitedAuction);

}

}

}

б) Управление начавшимися аукционами – алгоритм добавляет в хранилище данных аукционы, которые должны начаться

private void SetLiveAuctions()

{

var liveAuctions = \_waitedAuctions.Where(x => x.Value.TradingStart <= DateTime.Now);

foreach (var auction in liveAuctions)

{

if (!\_liveAuctions.ContainsKey(auction.Key))

{

var liveAuction = new LiveAuction

{

AuctionId = auction.Key,

CurrentPrice = auction.Value.StartPrice,

LastUser = auction.Value.LastUser,

Ticks = 5

};

\_liveAuctions.Add(auction.Key, liveAuction);

\_waitedAuctions.Remove(auction.Key);

}

}

}

в) Тайминг аукциона. Так как скандинавский аукцион заканчивается когда время таймера истекает, необходимо отслеживать эту ситуацию. Алгоритм запускается каждую секунду и уменьшает время таймера у активного аукциона. Когда время истекает, из хранилища активных аукционов удаляется завершившийся аукцион и запускается событие определения победителя аукциона.

private void TimerHandler(object obj)

{

ParallelLoopResult result = Parallel.ForEach(\_liveAuctions,

(x) =>

{

x.Value.Ticks--;

if (x.Value.Ticks >= 0)

{

SendState?.Invoke(x.Value.AuctionId, x.Value.Ticks, x.Value.CurrentPrice, x.Value.LastUser);

}

else

{

SetWinner?.Invoke(x.Value.AuctionId, x.Value.Ticks, x.Value.CurrentPrice, x.Value.LastUser);

\_waitedAuctions.Remove(x.Value.AuctionId);

}

});

}

**3.3.2** Алгоритм регистрации пользователя

После ввода в адресной строке URL-адреса веб-сайта пользователю предоставляется ограниченная функциональность. Для получения расширенных возможностей необходимо создать учётную запись.

Учётная запись — хранимая в [компьютерной системе](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) совокупность [данных](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5) о пользователе, необходимая для его опознавания ([аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F)) и [предоставления доступа](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B0%D0%B2%D0%B0_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%D0%B0) к его личным данным и настройкам. Для использования учётной записи (другими словами, для входа в систему под чьим-то именем) обычно требуется ввод имени и [пароля](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%BE%D0%BB%D1%8C). Также может требоваться другая дополнительная информация. Пользователи [портала](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) могут воспринимать учётную запись как личную страничку, профиль, кабинет, место хранения личных и других сведений на интернет-ресурсе.

Для создания учётной записи пользователю необходимо зарегистрироваться. Регистрация производится путём корректного ввода: имени, почты и пароля.

Выполнение алгоритма начинается с перехода пользователя на страницу регистрации и ввода данных в доступные поля. Алгоритм начинается с получения пользовательских данных, таких как электронная почта, имя, пароль. После получения данных их нужно провалидировать. Если в пользовательском браузере присутствует JavaScript, то появляется возможность валидации на стороне клиента, это означает, что проверка правильности ввода будет осуществляться непосредственно при вводе значений, без перезагрузки страницы. Данные прошедшие валидацию на стороне клиента отправляются на сервер, там они должны пройти серверную валидацию, которая по логике не должна отличаться от клиентской.

После прохода блока валидации данные должны быть приведены к пользовательской сущности и записаны в базу данных. На этапе приведения происходит шифрования пароля, запоминание текущей даты для пользовательского поля «Дата регистрации», создаётся авторизованная пользовательская сущность, которая определяет права и уникальность пользователя в системе.

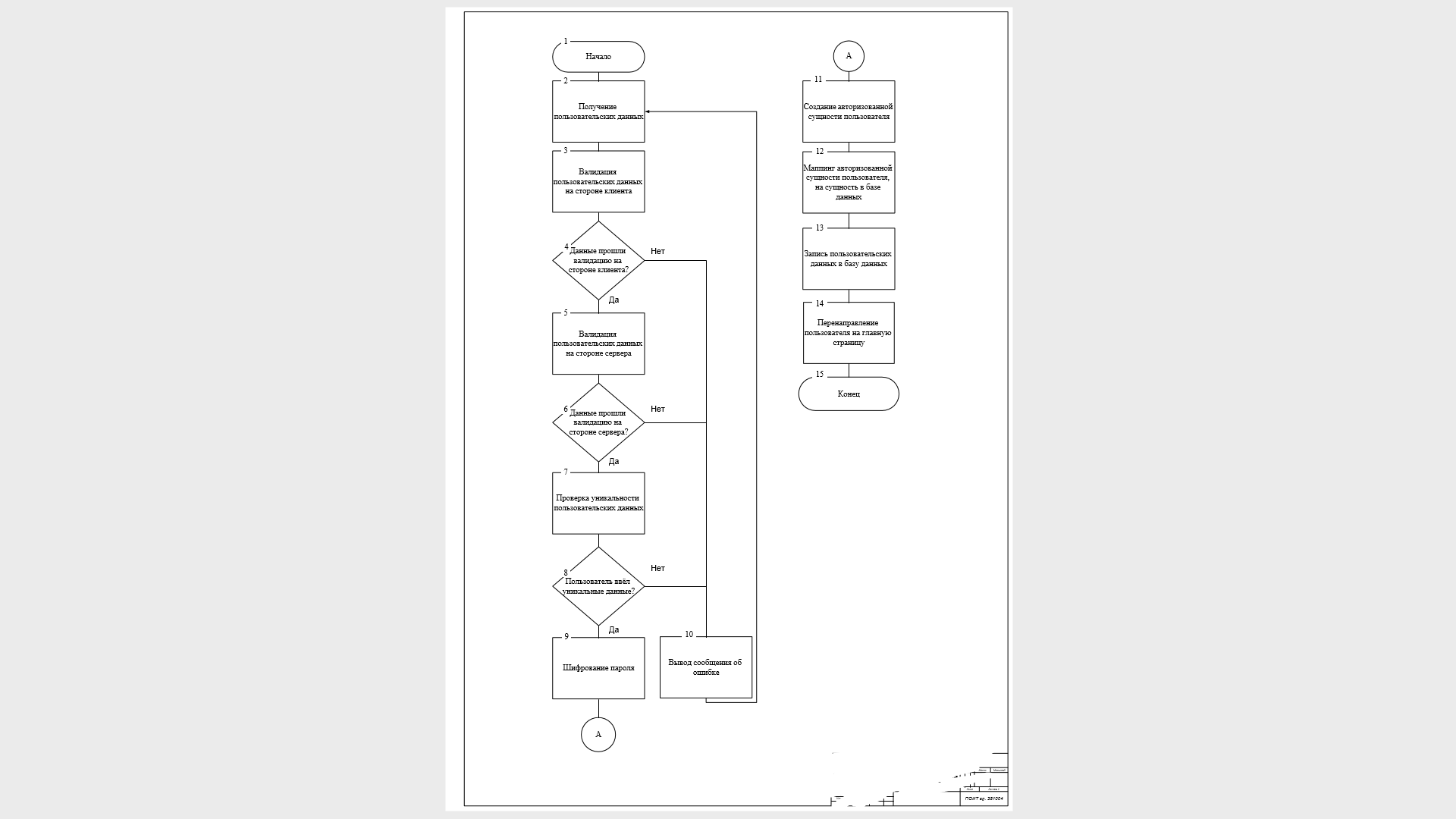


Рисунок 3.4 - Схема алгоритма регистрации пользователя

Для избежание пробелов в безопасности пользовательские данные подвергаются валидации. В разрабатываемом программном средстве используется двухэтапная валидация. Такая валидация подразумевает валидацию при непосредственном вводе информации в поля, а также валидацию подтверждении регистрации. В таблице 3.5 предоставлены требования к регистрационным полям.

Таблица 3.5 – Валидационные требования к регистрационным полям

|  |  |  |
| --- | --- | --- |
| Название | Описание | Требование |
| Email | Электронная почта пользователя | * уникальное; * длина от 5 до 30 символов; * более одного символа до знака @; * знак @; * доменное имя. |
| Name | Имя пользователя. Это имя будет видно другим пользователям | * не уникальное; * длина от 3 до 15 символов; * только буквенные символы; |
| Password | Пользовательский пароль | * длина от 6 до 15 символов; * наличие хотя бы одного цифрового символа. * наличие хотя бы одного символа в верхнем регистре * только латинские символы |
| Confirm password | Подтверждения пользовательского пароля | * полное совпадение с полем Password. |

**3.3.3** Алгоритм сохранения аукционов

Данный алгоритм начинает выполняться при создании аукциона пользователем. После получения сервером объекта, содержащим информацию об аукционе, необходимо произвести проверку полученных данных:

а) Полученный файл должен быть изображением, а значит содержать разрешение графических файлов.

б) Время начала аукциона не должно быть меньше текущего времени.



Рисунок 3.3 – Схема алгоритма сохранения аукциона

**4 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Для успешного создания программных средств необходимо обратить внимание на выбор языков программирования, сред разработки и технологий. Удачный выбор инфраструктуры для разработки программных средств позволяет снизить расходы на создание программного средства, а также увеличить качество и производительность труда.

Ниже представлены некоторые факторы, которые повлияли на выбор технологий:

* удобство разработки, позволяющее быстро и качественно разрабатывать программные средства;
* переносимость программного средства между различными персональными компьютерами;
* доступная документация для обращения при возникших затруднениях во время разработки;
* распространённость ОС, для которой разрабатывается программное средство.

Приняв во внимание вышеперечисленные факторы, а также рассмотрев особенности различных технологий, были выбраны следующие технологии:

* операционная система Windows 10;
* язык программирования C# 6.0;
* платформа .Net Framework 4.5.2;
* среда разработки Visual Studio 2015;
* фреймворк ASP.NET MVC.

Далее представлено подробное описание используемых языков программирования и технологий.

**4.1 Используемые технологии**

**4.1.1** Язык программирования C#

C# – элегантный, типобезопасный объектно-ориентированный язык, предназначенный для разработки разнообразных безопасных и мощных приложений, выполняемых в среде .NET Framework. С помощью языка C# можно создавать обычные приложения Windows, XML-веб-службы, распределённые компоненты, приложения «клиент-сервер», приложения баз данных и т. д. Visual C# предоставляет развитый редактор кода, конструкторы с удобным пользовательским интерфейсом, встроенный отладчик и множество других средств, упрощающих разработку приложений на базе языка C# и .NET Framework [10].

Синтаксис C# очень выразителен, но прост в изучении. Все, кто знаком с языками C, C++ или Java с лёгкостью узнают синтаксис с фигурными скобками, характерный для языка C#. Разработчики, знающие любой из этих языков, как правило, смогут добиться эффективной работы с языком C# за очень короткое время. Синтаксис C# делает проще то, что было сложно в C++, и обеспечивает мощные возможности, такие как типы значений Nullable, перечисления, делегаты, лямбда-выражения и прямой доступ к памяти, чего нет в Java. C# поддерживает универсальные методы и типы, обеспечивая более высокий уровень безопасности и производительности, а также итераторы, позволяющие при реализации коллекций классов определять собственное поведение итерации, которое может легко использоваться в клиентском коде. Выражения LINQ делают строго типизированный запрос очень удобной языковой конструкцией.

Как объектно-ориентированный язык, C# поддерживает понятия инкапсуляции, наследования и полиморфизма. Все переменные и методы, включая метод Main – точку входа приложения – инкапсулируются в определения классов. Класс может наследовать непосредственно из одного родительного класса, но может реализовывать любое число интерфейсов. Для методов, которые переопределяют виртуальные методы в родительском классе, необходимо ключевое слово override, чтобы исключить случайное повторное определение. В языке C# структура похожа на облегченный класс: это тип, распределяемый в стеке, реализующий интерфейсы, но не поддерживающий наследование.

В дополнение к основным описанным объектно-ориентированным принципам, язык C# упрощает разработку компонентов программного обеспечения благодаря нескольким инновационным конструкциям языка, в число которых входят следующие:

* инкапсулированные сигнатуры методов, называемые делегатами, которые поддерживают типобезопасные уведомления о событиях;
* свойства, выступающие в роли методов доступа для закрытых переменных-членов;
* атрибуты с декларативными метаданными о типах во время выполнения;
* встроенные комментарии XML-документации;
* LINQ, предлагающий встроенные возможности запросов в различных источниках данных.

Если потребуется обеспечить взаимодействие с другим программным обеспечением Windows, таким как объекты COM или собственные библиотеки DLL Win32, в языке C# можно использовать процесс, который называется Interop. Процесс Interop позволяет программам на C# выполнять практически любые действия, которые может выполнять исходное приложение на C++. Язык C# поддерживает даже указатели и понятие «небезопасного» кода для тех случаев, когда прямой доступ к памяти имеет крайне важное значение.

Процесс построения C# по сравнению с C и C++ прост и является более гибким, чем в Java. Нет отдельных файлов заголовка, а методы и типы не требуется объявлять в определенном порядке. В исходном файле C# может быть определено любое число классов, структур, интерфейсов и событий.

**4.1.2** Платформа .Net Framework

Программа на языке C# выполняется в среде .NET Framework – интегрированном компоненте Windows, содержащем виртуальную систему выполнения (среда CLR) и унифицированный набор библиотек классов. Среда CLR представляет собой коммерческую реализацию Майкрософт инфраструктуры CLI, международного стандарта, основы сред выполнения и разработки с тесным взаимодействием языков и библиотек [11].

Исходный код, написанный на языке C#, компилируется в промежуточный язык (IL) в соответствии со спецификацией CLI. Код IL и ресурсы, такие как растровые изображения и строки, хранятся на диске в исполняемом файле, называемом сборкой, с расширением EXE или DLL в большинстве случаев. Сборка содержит манифест со сведениями о типах сборки, версии, языке и региональных параметрах, и требованиях безопасности.

При выполнении программы на C# сборка загружается в среду CLR в зависимости от сведений в манифесте. Далее, если требования безопасности соблюдены, среда CLR выполняет JIT-компиляцию для преобразования кода IL в инструкции машинного кода. Среда CLR также предоставляет другие службы, относящиеся к автоматическому сбору мусора, обработке исключений и управлению ресурсами. Код, выполняемый средой CLR, иногда называют «управляемым кодом» в противопоставление «неуправляемому коду», который компилируется в машинный код, предназначенный для определенной системы.

Взаимодействие между языками является ключевой особенностью .NET Framework. Поскольку код на промежуточном языке (IL), создаваемый компилятором C#, соответствует спецификации CTS, код IL на основе C# может взаимодействовать с кодом, создаваемым версиями языков Visual Basic, Visual C++, Visual J# платформы .NET Framework и еще более чем 20 CTS-совместимых языков. В одной сборке может быть несколько модулей, написанных на разных языках платформы .NET Framework, и типы могут ссылаться друг на друга, как если бы они были написаны на одном языке.

Помимо служб времени выполнения, в .NET Framework также имеется обширная библиотека, состоящая из более чем 4000 классов, организованных по пространствам имен, которые обеспечивают разнообразные полезные функции для любых действий, начиная от ввода и вывода файлов для управления строками для разбивки XML, и заканчивая элементами управления Windows Forms. В обычном приложении на языке C# библиотека классов .NET Framework интенсивно используется для «устройства» кода.

**4.1.3** Фреймворк ASP.NET MVC

ASP.NET MVC Framework — фреймворк для создания веб-приложений, который реализует шаблон Model-view-controller.

Платформа ASP.NET MVC базируется на взаимодействии трех компонентов: контроллера, модели и представления. Контроллер принимает запросы, обрабатывает пользовательский ввод, взаимодействует с моделью и представлением и возвращает пользователю результат обработки запроса.

Модель представляет слой, описывающий логику организации данных в приложении. Представление получает данные из контроллера и генерирует элементы пользовательского интерфейса для отображения информации [12].

Для управления разметкой и вставками кода в представлении используется движок представлений. До версии MVC 5 использовались два движка: Web Forms и Razor. Начиная с MVC 5 единственным движком, встроенным по умолчанию, является Razor. Движок WebForms использует файлы .aspx, а Razor — файлы .cshtml и .vbhtml для хранения кода представлений. Основой синтаксиса Razor является знак @, после которого осуществляется переход к коду на языках C#. Также возможно и использование сторонних движков. Файлы представлений не являются стандартными статическими страницами с кодом html, а в процессе генерации контроллером ответа с использованием представлений компилируются в классы, из которых затем генерируется страница html.

При обработке запросов фреймворк ASP.NET MVC опирается на систему маршрутизации, которая сопоставляет все входящие запросы с определенными в системе маршрутами, которые указывают, какой контроллер и метод должен обработать данный запрос. Встроенный маршрут по умолчанию предполагает трехзвенную структуру: контроллер/действие/параметр.

ASP.NET MVC и ASP.NET Web Forms являются двумя родственными технологиями, в основании которых лежит одна платформа ASP.NET. И все же ASP.NET MVC имеет ряд преимуществ перед ASP.NET Web Forms:

* разделение ответственности (отдельная разработка разных компонентов - контроллера, моделей, представлений);
* улучшенная тестируемость;
* соответствие протоколу HTTP (ASP.NET MVC не поддерживает объект ViewState);
* повышенная гибкость и настраиваемость под собственные нужды.

**4.1.4** Библиотека jQuery

jQuery — [библиотека JavaScript](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_JavaScript), фокусирующаяся на взаимодействии [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) и [HTML](https://ru.wikipedia.org/wiki/HTML). Библиотека jQuery помогает легко получать доступ к любому элементу [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model), обращаться к атрибутам и содержимому элементов [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model), манипулировать ими. Также библиотека jQuery предоставляет удобный [API](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) для работы с [AJAX](https://ru.wikipedia.org/wiki/AJAX).

Преимущества jQuery:

* движок кросс-браузерных CSS-селекторов [Sizzle](https://ru.wikipedia.org/w/index.php?title=Sizzle&action=edit&redlink=1), выделившийся в отдельный проект;
* переход по дереву DOM, включая поддержку XPath как плагина;
* события;
* визуальные эффекты;
* AJAX-дополнения;
* JavaScript-плагины.

Точно так же, как [CSS](https://ru.wikipedia.org/wiki/CSS) отделяет визуализацию от структуры HTML, JQuery отделяет поведение от структуры HTML. Например, вместо прямого указания на обработчик события нажатия кнопки, управление передаётся JQuery, которая идентифицирует кнопки и затем преобразует его в обработчик события клика. Такое разделение поведения и структуры также называется принципом [ненавязчивого JavaScript](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BD%D0%B0%D0%B2%D1%8F%D0%B7%D1%87%D0%B8%D0%B2%D1%8B%D0%B9_JavaScript).

Библиотека jQuery содержит функциональность, полезную для максимально широкого круга задач. Тем не менее, разработчиками библиотеки не ставилась задача совмещения в jQuery функций, которые подошли бы всюду, поскольку это привело бы к большому коду, бо́льшая часть которого не востребована. Поэтому была реализована архитектура компактного универсального ядра библиотеки и плагинов. Это позволяет собрать для ресурса именно ту JavaScript-функциональность, которая на нём была бы востребована.

**4.1.5** Внедрение зависимостей, контейнер зависимостей Autofac

Внедрение зависимости ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Dependency injection, DI) — процесс предоставления внешней зависимости программному компоненту. Является специфичной формой «[инверсии управления](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Inversion of control, IoC), когда она применяется к управлению зависимостями. В полном соответствии с [принципом единственной ответственности](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%B5%D0%B4%D0%B8%D0%BD%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B9_%D0%BE%D0%B1%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8) объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму.

Работа фреймворка, обеспечивающая внедрение зависимости, описывается следующим образом. Приложение, независимо от оформления, исполняется внутри контейнера [IoC](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), предоставляемого фреймворком. Часть объектов в программе по-прежнему создается обычным способом языка программирования, часть создается контейнером на основе предоставленной ему конфигурации.

Условно, если [объекту](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) нужно получить доступ к определенному [сервису](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B5%D1%80%D0%B2%D0%B8%D1%81_(%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC)&action=edit&redlink=1), объект берет на себя ответственность за доступ к этому сервису: он или получает прямую [ссылку](https://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) на местонахождение сервиса, или обращается к известному «сервис-локатору» и запрашивает ссылку на реализацию определенного типа сервиса. Используя же внедрение зависимости, объект просто предоставляет свойство, которое в состоянии хранить ссылку на нужный тип сервиса; и когда объект создается, ссылка на реализацию нужного типа сервиса автоматически вставляется в это свойство, используя средства среды.

Внедрение зависимости более гибко, потому что становится легче создавать альтернативные реализации данного типа сервиса, а потом указывать, какая именно реализация должна быть использована в, например, [конфигурационном файле](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB_%D0%BA%D0%BE%D0%BD%D1%84%D0%B8%D0%B3%D1%83%D1%80%D0%B0%D1%86%D0%B8%D0%B8), без изменений в объектах, которые этот сервис используют. Это особенно полезно в [юнит-тестировании](https://ru.wikipedia.org/wiki/%D0%AE%D0%BD%D0%B8%D1%82-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), потому что вставить реализацию «[заглушки](https://ru.wikipedia.org/wiki/Mock-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82)» сервиса в тестируемый объект очень просто.

С другой стороны, излишнее использование внедрения зависимостей может сделать приложения более сложными и трудными в сопровождении: так как для понимания поведения программы программисту необходимо смотреть не только в исходный код, а еще и в конфигурацию, а конфигурация, как правило, невидима для [IDE](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), которые поддерживают анализ ссылок и [рефакторинг](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3), если явно не указана поддержка [фреймворков](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с внедрениями зависимостей.

Autofac - это легкая платформа, упрощающая внедрения зависимостей для приложений .NET. Позволяющая разделить приложение на множество слабосвязанных фрагментов, а затем внедрить их в гибкой манере. Использование Autofac для поддержки архитектуры программного обеспечения позволит повторно использовать модули, упростить тестирование и модифицировать.

Основные преимущества Autofac:

* Сосредоточенность на выполняемых функциях. Слишком много существующих контейнеров зависимостей приносят в жертву удобство использования для функций, которые не часто необходимы. Каждый раз, когда функция добавляется в Autofac, ее необходимость сравнивается с той сложностью, которую она добавляет к ежедневному использованию. У Autofac есть много продвинутых функций, но их понимание не требуется для использования основных функций.
* Быстрота. Вместо того, чтобы полагаться на механизм отражение для внедрения зависимостей, Autofac использует генерацию кода в CLR. Это может привести к резкому (8-50 раз) улучшению производительности во многих ситуациях.
* Точность. Вместо того, чтобы полагаться на файлы сопоставления XML и идентификаторы строк для подключения компонентов, Autofac обеспечивает надежный предметно-ориентированный язык. Это означает, что Autofac использует возможность языка C# такую как безопасность типов и возможности IDE такие как IntelliSense и автоматическое дописывание кода.
* Скрытность. Autofac не вторгается в код. Можно легко изолировать зависимости в одной сборки в проекте.

**4.1.6** Entity Framework

Entity Framework представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными 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 работает с объектами [13].

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами. Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Свойства необязательно представляют простые данные типа int, могут представлять более комплексные структуры 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. Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из БД. С помощью LINQ можно не только извлекать определенные строки, хранящие объекты, из базы данных, но и получать объекты, связанные различными ассоциативными связями.

Другим ключевым понятием является Entity Data Model. Эта модель сопоставляет классы сущностей с реальными таблицами в БД. Entity Data Model состоит из трех уровней: концептуального, уровень хранилища и уровень сопоставления (маппинга).

На концептуальном уровне происходит определение классов сущностей, используемых в приложении.

Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется используемая база данных.

Уровень сопоставлен служит посредником между предыдущими двумя, определяя сопоставление между свойствами класса сущности и столбцами таблиц. Таким образом, появляется возможность через классы, определенные в приложении, взаимодействовать с таблицами из базы данных.

Способы взаимодействия с БД Entity Framework предполагает три возможных способа взаимодействия:

* Database first. Entity Framework создает набор классов, которые отражают модель конкретной базы данных;
* Model first. Разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере;
* Code first. Разработчик создает класс модели данных, которые будут храниться в базе данных, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы.

**4.1.7** SignalR

Для упрощения работы с коммуникациями реального времени была создана специальная библиотека под названием SignalR. При этом SignalR не является библиотекой, предназначенной только для использования в ASP.NET MVC. Ее также можно использовать и в веб-формах, а также в консольных приложениях, десктопных приложения и в WPF.

SignalR предоставляет простой API для создания функционала, который позволяет вызывать функции JavaScript на стороне клиента из серверного кода, написанного с помощью языков платформы .NET. SignalR значительно упрощает работу с коммуникациями реального времени. Библиотека обрабатывает все подключения и автоматически рассылает сообщения всем подключенным клиентам, либо каким-нибудь специфическим клиентам.

Фактически библиотека SignalR состоит из API серверной стороны, который применяется в коде на C#, и из клиентских библиотек JavaScript.

SignalR предоставляет разработчикам две модели: постоянные подключения (Persistent Connection) и хабы (Hubs).

Постоянные подключения (Persistent Connection API) представляют разработчикам прямой доступ к низкоуровневому протоколу коммуникации. Подключения в этой модели представляют конечную точку, к которой подключаются клиенты, наподобие модели подключений в WCF.

Хабы же предоставляют протокол взаимодействия более высокого уровня. Они представляют верхний слой над Persistent Connection API и позволяют клиенту и серверу напрямую вызывать методы друг друга.

SignalR поддерживается наиболее распространенными серверными платформами на базе ОС Windows:

а) Windows Server 2012

б) Windows Server 2008 r2

с) Windows 8

в) Windows 7

г) Windows Azure

Для работы на стороне сервера также необходима версия фреймворка .net от 4.0 и выше.

Для обмена данными между клиентом и сервером SignalR использует тот способ передачи или тот транспорт, который наиболее подходит к данной ситуации. Однако разработчики могут переопределить способ передачи. SignalR предоставляет следующие типы технологий для взаимодействия сервера и клиента:

а) WebSockets

б) Server-sent events

в) Forever Frames

г) Long polling

При создании подключения SignalR выбирает, если доступно, технологию WebSocket, так как это наиболее оптимальная технология для SignalR, наиболее эффективно использующая память сервера. В то же время WebSocket может использоваться только на серверах под управлением Windows Server 2012 или Windows 8 и при наличии установленного .NET Framework 4.5. При этом также технология WebSocket должна поддерживаться и браузером клиента. И если технология WebSocket недоступна на сервере или клиенте, то выбирается другой транспорт.

**4.1.9** SASS

SASS это язык похожий на [HAML](http://haml-lang.com/), но предназначенный для упрощения создания CSS-кода. Проще говоря, SASS это такой язык, код которого специальной ruby-программой транслируется в обычный CSS код. Синтаксис этого языка очень гибок, он учитывает множество мелочей, которые так желанны в CSS. Более того, в нём есть даже логика (@if, [each](https://habrahabr.ru/users/each/)), математика (можно складывать как числа, строки, так и цвета).

**4.1.8** Среда разработки Visual Studio

Visual Studio – интегрированная среда разработки ПО от компании Microsoft. Visual Studio используется для разработки консольных приложений, приложений с графическим интерфейсом, веб-сайтов, веб-приложений, программ с поддержкой Windows Forms, а также для работы с системой построения клиентских приложений – Windows Presentation Foundation (WPF). Можно разрабатывать веб-сервисы и в родном, и в управляемом кодах для всех платформ, поддерживаемых .NET Framework, Microsoft Windows, Windows Mobile, Microsoft Silverlight и .NET Compact Framework [14].

Visual Studio состоит из редактора кода, поддерживающего IntelliSense и рефакторинг кода. Встроенный отладчик можно использовать как для отладки уровня исходного кода, так и в качестве отладчика машинного уровня. Другие встроенные инструменты включают:

– редактор форм для разработки приложений GUI;

– веб-редактор;

– редактор классов;

– редактор схемы баз данных.

Visual Studio даёт возможность устанавливать плагины для расширения функциональных возможностей почти на любом уровне, в том числе для поддержки систем контроля исходного кода и новых инструментов для редактирования на предметно-ориентированных языках программирования, а также инструментов для других аспектов жизненного цикла разработки программного обеспечения.

Visual Studio предоставляет языковые сервисы, которые позволяют редактору кода и отладчику поддерживать практически любой язык программирования. Встроенные языки программирования включают:

– C/C++;

– VB.NET;

– C#;

– F#.

Также Visual Studio поддерживает XML/XSLT, HTML/XHTML, JavaScript и CSS.

* 1. **Описание проектной структуры**

**4.2.1** Взаимодействие проектов

В разделе 3.1 рассматривалась архитектура программного средства, однако эта архитектура абстрагирована от конкретного фреймворка на котором и производится разработка.

При использовании ASP.NET MVC как правило, приложение в целом будет представлять решение (англ. Solution) в Visual Studio, а отдельные уровни - проекты. На рисунке 4.1 представлена структура программного средства в рамках решения в Visual Studio. Из него видно, что структура разделена на три слоя:

* слой представления – проект Application;
* слой бизнес-логики – проект BLL;
* слой доступа к данным – проекты DAL и Domain.



Рисунок 4.1 - Структура решения в Visual Studio

Каждый слой связан со следующим с помощью интерфейсов, это сделано для того, чтобы в случае необходимости изменить логику слоя, для этого необходимо заменить классы, которые содержат логику (BLL, DAL), на новые реализующие интерфейсы этого слоя [12].

Так же слой доступа к данным содержит проект под названием Domain, который выполняет роль связующего звена с базой данных. Для доступа к разным базам данных используются разные библиотеки, написанные разными фирмами, поэтому вынесение логики, связанной с доступом к специфической базе может быть полезно при последующей смене базы данных.

**4.2.2** Структура проекта Application

На рисунке 4.2 представлена файловая структура проекта Application который входит в слой представления и по сути является пользовательским интерфейсом [7]. Проект состоит из следующих частей:

* App\_Data. В данной папке хранятся все необходимые файлы и ресурсы, такие как, базы данных, используемые приложением. После развертывания приложения только непосредственно приложение может работать с этой папкой, доступ же простых пользователей в эту папку запрещен.
* Файл Global.asax и папка App\_Start. Папка App\_Start включает весь функционал конфигурации приложения, который в предыдущих версиях содержался в файле Global.asax, а теперь перенесен в набор статичных классов, вызываемых в Global.asax. Эти статичные классы содержат некоторую логику инициализации приложения, выполняющуюся при запуске.
* Файл Web.config. Файл конфигурации приложения, который находится в корневой папке приложения.
* Content. В данной папке хранятся вспомогательные файлы, которые не включают код на c# или javascript, и которые развертываются вместе с приложением. В частности, здесь могут размещаться файлы стилей css. В папке scss находятся стили приложения, в папке images изображения, а в папке fonts шрифты.
* Controllers. Содержит контроллеры - классы, отвечающие за работу приложения.
* Scripts. В данной папке содержатся скрипты на JavaScript, используемые в приложении. По умолчанию эта папка уже содержат файлы библиотеки jQuery.
* Models. Содержит модели, используемые слоем представления.
* Util. Содержит конфигурацию внедрения зависимостей.
* Hubs. Здесь размещается класс используемый для организации двустороннего соединения.
* Views. Здесь размещаются представления. Представления группированы по папкам, каждая из которых соответствует одному контроллеру. После получения и обработки запроса контроллер, отправляет одно из этих представлений, заполненных некоторыми данными, клиенту. Кроме того, имеется папка общих для контроллеров представлений - папка Shared.

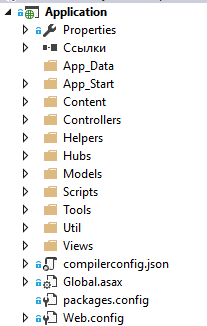


Рисунок 4.2 - Файловая структура проекта Application

**4.3 Уровень доступа к базе данных**

В данном подразделе приводится описание шаблонов доступа к данным и их реализации. Уровень доступа к базе данных отвечает за чтение, создание, редактирование, удаление информации. Реализован с помощью MSSQL Server с применением объектно-реляционного отображателя Entity Framework.

**4.3.1 Традиционные объекты CLR**

*Традиционный объект CLR* (Plain Old CLR Object – POCO) – это класс .NET, который используется для представления класса сущности (модели). Этот класс фокусируется на ключевых атрибутах (свойства) и поведении (методы) сущности и связанных с ней сущностей, не требуя никакого специфичного для базы данных кода [5].

Основная цель использования классов POCO заключается в проектировании бизнес-модели проекта с *игнорированием постоянства* (persistence ignorance – PI). Такой подход к проектированию позволяет бизнес-модели проекта развиваться независимо от модели доступа к данным. Поскольку бизнес-модель не содержит кода доступа к данным, облегчается тестирование модели в изоляции, и лежащие в основе данные могут легко быть заменены в случае изменения бизнес-требований.

**4.3.2 Использование шаблона Repository**

*Шаблон Repository (Репозиторий)* – это шаблон доступа к данным, который способствует слабо связанному подходу в отношении доступа к данным. Вместо помещения логики доступа к данным на уровень выше, ответственность за постоянство бизнес-модели проекта возлагается на отдельный класс или набор классов, называемых *репозиторием*.

За счет применения этого шаблона изолируется уровень доступа к данным от остальной части проекта.

Для проектирования репозитория существуют разные подходы, которые кратко описаны ниже:

1. Один на бизнес-модель. Наиболее прямолинейный способ заключается в создании репозитория для каждого класса бизнес-модели. Хотя этот подход и прост, он может приводить к проблемам, таким как дублирование кода или сложность, когда множеству репозиториев понадобится взаимодействовать друг с другом.
2. Использование агрегированного корня. *Агрегированный корень* – это класс, который может существовать сам по себе и являться ответственным за управление ассоциациями с другими связанными классами. Например, в проекте электронной коммерции может существовать класс OrderRepository, который будет обрабатывать создание заказа и связанных с ним элементов в заказе.
3. Обобщенный репозиторий. Вместо создания специфических классов репозитория можно воспользоваться обобщениями .NET для построения общего репозитория, который будет применяться множеством приложений.

Ниже представлена базовая структура обобщенного класса репозитория:

public interface IGenericRepository<T>

where T : class

{

IQueryable<T> GetAll();

T Get(int id);

void Create(T item);

void Update(T item);

void Delete(int id);

}

Класс бизнес-логики взаимодействуют классом UnitOfWork, а UnitOfWork взаимодействует с репозиториями для загрузки и сохранения бизнес-модели проекта. За счет использования технологии внедрения зависимостей (DI) репозитории могут внедряться через конструктор сервиса. На рисунке 4.4 изображено отношение между классом UnitOfWork, репозиторием и контекстом данных EntityFramework, при котором сервисы взаимодействуют с UnitOfWork , а не напрямую с EntityFramework.



Рисунок 4.4 – Организация взаимодействия при наличии репозитория

**3.1.6 Использование шаблона Unit Of Work**

*Шаблон Unit Of Work (Единица работы)* – это шаблон проектирования, который обслуживает набор объектов, изменяемых в бизнес-транзакции (бизнес-действии) и управляет записью изменений и разрешением проблем конкуренции данных.

Когда необходимо писать и читать из базы данных, важно следить за тем, что было изменено, и если не изменено, то не записывать данные в базу данных. Также необходимо вставлять данные о новых объектах и удалять данные о старых.

Можно записывать в базу данных каждое изменение объекта, но это приведёт к большому количеству мелких запросов к базе данных, что закончится замедлением работы проекта. Более того, это требует держать открытую транзакцию все время работы приложения, что непрактично, если проект обрабатывает несколько запросов одновременно. Ситуация еще хуже, если необходимо следить за чтением из базы данных, чтобы избежать неконсистентного чтения.

Реализация паттерна Unit Of Work следит за всеми действиями проекта, которые могут изменить базу данных в рамках одного бизнес-действия. Когда бизнес-действие завершается, Unit Of Work выявляет все изменения и вносит их в базу данных (см. чертеж ГУИР.400201.035 РР.1).

Ниже представлена структура класса, реализующего шаблон Unit Of Work:

public interface IUnitOfWork : IDisposable

{

IGenericRepository<EndedAuction> EndedAuctions { get; }

IGenericRepository<Auction> Auctions { get; }

Task SaveAsync();

void Dispose(bool disposing);

}

Шаблон UnitOfWork призван отслеживать все изменения данных, которые производятся над доменной моделью в рамках бизнес-транзакции. После того, как бизнес-транзакция закрывается, все изменения попадают в БД в виде единой транзакции.

**3.2 Описание сервисов уровня бизнес-логики**

Сервисы на данном уровне проекта являются промежуточным звеном в процессах обработки данных, которые перемещаются между уровнями проекта. Также сервисы играют роль поставщиков данных, когда уровень представления запрашивает их с помощью интерфейса, предоставляемого сервисами. Сервисы используют экземпляр класса, реализующего паттерн Unit Of Work, как средство связи с уровнем доступа к данным. С помощью Unit Of Work сервисы составляют различные запросы к данным:

* сохранение данных;
* извлечение данных по различным критериям;
* удаление данных;
* модификация данных.

**3.2.1 Класс AuctionService**

Класс AuctionService является сервисом, который наследуется от интерфейса IAuctionService и предоставляет перечень операций совершаемых над сущностью Auction.

Интерфейс класса AuctionService:

public interface IAuctionService

{

IEnumerable<Auction> GetAuctions();

Auction GetAuctionById(int id);

Task CreateAuction(Auction auction);

Task UpdateAuction(Auction auction);

Task DeleteAuction(Auction auction);

}

**3.2.1 Класс UserService**

Класс UserService является сервисом, который предоставляет перечень операций, совершаемых над сущностью ApplicationUser, реализует интерфейс IUserService.

Основные методы класса UserManagerService:

* Authenticate(UserDto userDto) – асинхронно совершает поиск пользователя по данным авторизации;
* Register(UserDto userDto) – асинхронно создает пользователя с указанным паролем;
* IsEmailConfirmAsync(string id) – асинхронно подтверждает адрес электронной почты пользователя;
* FindByNameAsync(string email) – асинхронно совершает поиск пользователя по имени;
* ResetPassword(string email, string code, string password) – асинхронно производит сброс пароля;
* FindByIdAsync(string id) – асинхронно совершает поиск пользователя по идентификатору;
* GetUsers() – извлекает список пользователей;

**3.1.3 Выбор подхода доступа к данным**

В Microsoft признают, что единый подход для доступа к данным не работает. Существует подход, ориентированный на данные, сосредоточенный на проектировании базы данных первой с последующей генерацией для проекта бизнес-модели, во многом управляемой структурой базы данных. В другом подходе используются классы POCO для определения структуры бизнес-модели и либо генерации базы данных из классов POCO, либо привязки модели к существующей базе данных.

В результате ADO.NET Entity Framework позволяет осуществлять выбор между тремя стилями:

1. Database First (Сначала база данных). Для тех, кто предпочитает применять более ориентированный на данные подход или начинать с существующей базы данных, Entity Framework предлагает возможность генерировать бизнес-модель на основе таблиц и их строк в реляционной базе данных. Инфраструктура Entity Framework использует специальный конфигурационный файл (с расширением .edmx) для хранения информации о схеме базы данных, концептуальной модели данных и сведений об отображении между ними.
2. Model First (Сначала модель). Если существующей базы данных нет, Entity Framework предлагает визуальный конструктор, который может применяться для создания концептуальной модели данных. Как и в случае подхода Database First, инфраструктура Entity Framework использует файл схемы для хранения информации, связанной с отображением модели на схему базы данных. После того как модель создана, визуальный конструктор EF может сгенерировать схему базы данных, посредством которой можно создать саму базы данных.
3. Code First (Сначала код). Разработчики, желающие применять подход с большим игнорированием постоянства, могут создавать бизнес-модель прямо в коде. Инфраструктура Entity Framework предоставляет специальный API-интерфейс отображения и поддерживает набор соглашений, обеспечивающих работу этого подхода. При подходе Code First инфраструктура Entity Framework не использует внешний конфигурационный файл для хранения схемы базы данных, поскольку API-интерфейс отображения применяет эти соглашения для генерации схемы базы данных динамически во время выполнения.

**3.1.4 Использование подхода Entity Framework Code First**

Движущей силой подхода Code First является возможность применения классов POCO. В рамках подходов Database First и Model First классы модели, сгенерированные EF, унаследованы от базового класса EntityObject, который предоставляет необходимое оснащение для отображения класса на лежащую в основе схему базы данных. Поскольку подходы Database First и Model First требуют наследования классов хранения от класса EntityObject, они не поддерживают игнорирование постоянства.

Вместо использования для отображения базового сущностного класса в подходе Code First установлен набор соглашений для отображения классов POCO:

* имена таблиц определяются с применением формы множественного числа имени сущности класса;
* имена столбцов выводятся из имен свойств;
* первичные ключи основываются на свойствах с именами ID и имяКлассаID.
* стандартная строка соединения соответствует имени класса DataContext.

Инфраструктура Entity Framework включает множество атрибутов аннотирования данных, которые можно использовать для управления обработкой отображаемых сущностей (см. таблицу 3.1).

Таблица 3.1 – Аннотации данных Code First

|  |  |
| --- | --- |
| Аннотация | Описание |
| Column | Имя столбца базы данных, порядковый номер и тип данных для отображения свойств |
| ComplexType | Используется на классах, которые не содержат ключей и не могут управляться инфраструктурой Entity Framework. Обычно применяется для управления скалярными свойствами в связанной сущности |
| DatabaseGenerated | Используется для пометки свойства, которое должно быть сгенерировано базой данных |
| ForeignKey | Используется для идентификации связанной сущности; представляет ограничение внешнего ключа, устанавливаемое между столбцами |
| InverseProperty | Используется для идентификации свойства, которое представляет другой конец отношения |
| Key | Одно или более свойств, применяемых для уникальной идентификации сущности |
| MaxLength | Максимальная длина свойства (столбца) |
| MinLength | Минимальная длина свойства (столбца) |
| NotMapped | Помечает свойство, которое не будет отображаться инфраструктурой Entity Framework |
| Required | Помечает свойство как являющееся обязательным (не допускающим null) |
| StringLength | Определяет минимальную и максимальную длину поля |
| Table | Используется для определения имени таблицы, применяемого для сущности |

Хотя соглашения направлены на улучшение продуктивности разработки, в Entity Framework допускается наличие ситуаций, когда необходимо нарушить одно и более применяемых соглашений, и для этого предлагается API-интерфейс, который позволяет разработчикам обходить существующие соглашения.

Ниже приведен пример сущностного класса, сконфигурированного с использованием атрибута Key, то есть аннотации данных, которая переопределяет стандартное отображение первичного ключа:

public class Entity

{

[Key]

public long Id { get; set; }

}

Подход доступа к данным ADO.NET Entity Framework Code First предполагает создание класса контекста доступа к данным, который унаследован от DbContext. Этот класс должен содержать свойства для каждой сущности в модели предметной области. Специальный класс контекста доступа может переопределять методы базового класса контекста для поддержки любой специализированной логики запросов и сохранения данных, а также реализовывать любую необходимую логику для отображения сущностей.

Ниже приведен контекст данных Entity Framework Code First, который содержит сущности, соответствующие таблицы базы данных. После определения класса контекста данных с помощью запроса LINQ можно совершать различные операции над любым списком сущностей:

public class ApplicationDbContext : DbContext

{

public DbSet<Cinema> Cinemas { get; set; }

public DbSet<Film> Films { get; set; }

public DbSet<Review> Reviews { get; set; }

public DbSet<Seance> Seances { get; set; }

public DbSet<Ticket> Tickets { get; set; }

public DbSet<PaymentMethod> PaymentMethods { get;set; }

public DbSet<Seat> Seats { get; set; }

public DbSet<ReservedSeat> ReservedSeats { get; set; }

public DbSet<Hall> Halls { get; set; }

public ApplicationDbContext() {}

public static ApplicationDbContext Create()

{

return new ApplicationDbContext();

}

}

Вместо работы напрямую с контекстом данных Entity Framework используется шаблон Repository. Это устанавливает уровень абстракции между сервисами уровня бизнес-логики проекта и инфраструктурой Entity Framework. Обеспечение реализации классом репозитория интерфейса IRepository позволяет применять контейнер IoC для внедрения репозитория в клиентский код (например, сервис).

Ниже приведен пример репозитория, который настроен для абстрагирования от контекста данных Entity Framework:

public class BaseRepository<TEntity> : IRepository<TEntity> where TEntity : Entity

{

private readonly ApplicationDbContext \_context;

public BaseRepository(ApplicationDbContext context)

{

\_context = context;

}

}

**3.1.5 Описание моделей уровня доступа к данным**

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами. Эти объекты являются проекциями таблиц базы данных (см. чертеж ГУИР.400201.035 РР.2).

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает кинотеатр, то можно выделить такие свойства, как название, адрес, номер телефона, путь к файлу с изображением кинотеатра и другие. Свойства необязательно представляют простые данные типа int, но и могут представлять более комплексные структуры данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность.

**3.1.5.1 Класс Entity**

Класс Entity является базовым классом для моделей уровня доступа к данным.

Свойства класса Entity:

* public long Id – первичный ключ для уникальной идентификации сущностей, которые наследуются от данного класса.

**3.1.5.2 Класс ApplicationUser**

Класс ApplicationUser является сущностью, которая соответствует таблице ApplicationUser базы данных, и реализует интерфейс IdentityUser.

Свойства класса ApplicationUser:

* public string FirstName – хранит имя пользователя, помечено атрибутами Required и StringLength;
* public string LastName – хранит фамилию пользователя, помечено атрибутами Required и StringLength;
* public bool IsBanned – определяет заблокированный статус пользователя;
* public virtual ICollection<Ticket> Tickets – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Ticket;
* public virtual ICollection<Review> Reviews – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Review;
* public virtual ICollection<ReservedSeat> ReservedSeats – навигационное свойство, обозначающее отношение между данной сущностью и сущностью ReservedSeats.

Методы класса ApplicationUser:

* public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager) – асинхронный метод, создающий объект ClaimsIdentity.

**3.1.5.3 Класс Cinema**

Класс Cinema является сущностью, которая соответствует таблице Cinema базы данных, и наследует свойства и методы базового класса Entity.

Свойства класса Cinema:

* public string Name – хранит название кинотеатра, помечено атрибутами Required и StringLength;
* public string ImagePath – хранит относительный путь к файлу изображения кинотеатра, помечено атрибутами Required и StringLength;
* public string Address – хранит адрес кинотеатра, помечено атрибутами Required и StringLength;
* public string PhoneNumber – хранит номер телефона кинотеатра, помечено атрибутами Required и StringLength;
* public virtual ICollection<Seance> Seances – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Seance;
* public virtual ICollection<Hall> Halls – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Hall.

**3.1.5.4 Класс Film**

Класс Film является сущностью, которая соответствует таблице Film базы данных, и наследует свойства и методы базового класса Entity.

Свойства класса Film:

* public string Title – хранит название фильма, помечено атрибутами Required и StringLength;
* public string Description – хранит описание фильма, помечено атрибутами Required и StringLength;
* public string ImagePath – хранит относительный путь к файлу изображения фильма;
* public int Duration – хранит длительность фильма, помечено атрибутами Required и Range;
* public virtual ICollection<Review> Reviews – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Review;
* public virtual ICollection<Seance> Seances – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Seance.

**3.1.5.5 Класс Hall**

Класс Hall является сущностью, которая соответствует таблице Hall базы данных, и наследует свойства и методы базового класса Entity.

Свойства класса Hall:

* public string Name – хранит название зала кинотеатра, помечено атрибутами Required и StringLength;
* public int RowAmount – хранит количество рядов в зале, помечено атрибутом Range;
* public int ColumnAmount – хранит количество мест в ряду зала, помечено атрибутом Range;
* public virtual Cinema Cinema – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Cinema;
* public long CinemaId – внешний ключ, связывающий данную сущность и сущность Cinema;
* public virtual ICollection<Seat> Seats – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Seat;
* public virtual ICollection<Seance> Seances – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Seance.

**3.1.5.6 Класс PaymentMethod**

Класс PaymentMethod является сущностью, которая соответствует таблице PaymentMethod базы данных, и наследует свойства и методы базового класса Entity.

Свойства класса PaymentMethod:

* public string Name – хранит название способа оплаты, помечено атрибутами Required и StringLength;
* public virtual ICollection<Ticket> Tickets – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Ticket.

**3.1.5.7 Класс Seat**

Класс Seat является сущностью, которая соответствует таблице Seat базы данных, и наследует свойства и методы базового класса Entity.

Свойства класса Seat:

* public int RowNumber – хранит номер ряда, в котором находится конкретное место в зале;
* public int ColumnNumber – хранит номер места в ряду зала;
* public virtual Hall Hall – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Hall;
* public long HallId – внешний ключ, связывающий данную сущность и сущность Hall.

**3.1.5.8 Класс Review**

Класс Review является сущностью, которая соответствует таблице Review базы данных, и наследует свойства и методы базового класса Entity.

Свойства класса Review:

* public int Rate – хранит значение рейтинга данного конкретного отзыва о фильме, помечено атрибутом Range;
* public string Description – хранит описание фильма, помечено атрибутами Required и StringLength;
* public DateTime ReviewDate – хранит дату создания отзыва о фильме;
* public virtual ApplicationUser ApplicationUser – навигационное свойство, обозначающее отношение между данной сущностью и сущностью ApplicationUser;
* public string ApplicationUserId – внешний ключ, связывающий данную сущность и сущность ApplicationUser;
* public virtual Film Film – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Film;
* public long FilmId – внешний ключ, связывающий данную сущность и сущность Film.

**3.1.5.9 Класс ReservedSeat**

Класс ReservedSeat является сущностью, которая соответствует таблице ReservedSeat базы данных, и наследует свойства и методы базового класса Entity.

Свойства класса ReservedSeat:

* public int RowNumber – хранит номер ряда, в котором находится заказанное место в зале;
* public int ColumnNumber – хранит номер места в ряду зала;
* public DateTime ReservationTime – хранит дату заказа места в зале;
* public bool IsSold – определяет заказанный статус места в зале;
* public virtual Seance Seance – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Seance;
* public long SeanceId – внешний ключ, связывающий данную сущность и сущность Seance;
* public string ApplicationUserId – внешний ключ, связывающий данную сущность и сущность ApplicationUser.

**3.1.5.10 Класс Seance**

Класс Seance является сущностью, которая соответствует таблице Seance базы данных, и наследует свойства и методы базового класса Entity.

Свойства класса Seance:

* public DateTime Date – хранит дату проведения сеанса;
* public DateTime Time – хранит время проведения сеанса;
* public decimal Price – хранит цену сеанса;
* public virtual Film Film – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Film;
* public long FilmId – внешний ключ, связывающий данную сущность и сущность Film;
* public virtual Hall Hall – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Hall;
* public long HallId – внешний ключ, связывающий данную сущность и сущность Hall;
* public virtual ICollection<Ticket> Tickets – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Ticket;
* public virtual ICollection<ReservedSeat> ReservedSeats – навигационное свойство, обозначающее отношение между данной сущностью и сущностью ReservedSeat.

**3.1.5.11 Класс Ticket**

Класс Ticket является сущностью, которая соответствует таблице Ticket базы данных, и наследует свойства и методы базового класса Entity.

Свойства класса Ticket:

* public DateTime PaymentDate – хранит дату оплаты билета;
* public virtual ApplicationUser ApplicationUser – навигационное свойство, обозначающее отношение между данной сущностью и сущностью ApplicationUser;
* public string ApplicationUserId – внешний ключ, связывающий данную сущность и сущность ApplicationUser;
* public virtual Seance Seance – навигационное свойство, обозначающее отношение между данной сущностью и сущностью Seance;
* public long SeanceId – внешний ключ, связывающий данную сущность и сущность Seance;
* public virtual PaymentMethod PaymentMethod – навигационное свойство, обозначающее отношение между данной сущностью и сущностью PaymentMethod;
* public long PaymentMethodId – внешний ключ, связывающий данную сущность и сущность PaymentMethod;
* public long SeatId – внешний ключ, связывающий данную сущность и сущность Seat.

**3**

**3.2 Уровень бизнес**-**логики**

**3.2.1 Описание сервисов уровня бизнес-логики**

Сервисы на данном уровне проекта являются промежуточным звеном в процессах обработки данных, которые перемещаются между уровнями проекта. Также сервисы играют роль поставщиков данных, когда уровень представления запрашивает их с помощью интерфейса, предоставляемого сервисами. Сервисы используют экземпляр класса, реализующего паттерн Unit Of Work, как средство связи с уровнем доступа к данным. С помощью Unit Of Work сервисы составляют различные запросы к данным:

* сохранение данных;
* извлечение данных по различным критериям;
* удаление данных;
* модификация данных.

**3.2.1.1 Класс BaseService**

Класс BaseService является базовым классом для сервисов уровня доступа к данным, реализует интерфейс IService.

Поля класса BaseService:

* readonly IUnitOfWork Uow – объект, реализующий интерфейс IUnitOfWork.

Конструктор класса BaseService:

* BaseService(IUnitOfWork unitOfWork) – конструктор, инициализирующий поля данного класса.

**3.2.1.2 Класс UserManagerService**

Класс UserManagerService является сервисом, который предоставляет перечень операций, совершаемых над сущностью ApplicationUserDomainModel, наследуется от класса BaseService, реализует интерфейс IUserManagerService.

Основные методы класса UserManagerService:

* FindAsync(string email, string password) – асинхронно совершает поиск пользователя по данным авторизации;
* CreateAsync(ApplicationUserDomainModel userDomainModel, string password) – асинхронно создает пользователя с указанным паролем;
* ConfirmEmailAsync(string userId, string code) – асинхронно подтверждает адрес электронной почты пользователя;
* FindByNameAsync(string email) – асинхронно совершает поиск пользователя по имени;
* ResetPasswordAsync(string userId, string code, string password) – асинхронно производит сброс пароля;
* FindByIdAsync(string userId) – асинхронно совершает поиск пользователя по идентификатору;
* GetUsers() – извлекает список пользователей;
* IsInRole(string userId, string role) – проверяет, принадлежит ли пользователь указанной роли;
* RemoveFromRole(string userId, string role) – удаляет у пользователя указанную роль;
* AddToRole(string userId, string role) – добавляет пользователю указанную роль;
* GetRoles(string userId) – извлекает список ролей пользователя;
* BanUser(string userId) – блокирует пользователя;
* UnbanUser(string userId) – снимает блокировку с пользователя.

**3.2.1.3 Класс CinemaService**

Класс CinemaService является сервисом, который предоставляет перечень операций, совершаемых над сущностью CinemaDomainModel, наследуется от класса BaseService, реализует интерфейс ICinemaService.

Основные методы класса CinemaService:

* Add(CinemaDomainModel entity) – создает новый кинотеатр;
* Read() – извлекает список кинотеатров;
* ReadById(object id) – извлекает кинотеатр по идентификатору;
* Update(CinemaDomainModel entity) – модифицирует данные кинотеатра;
* Delete(CinemaDomainModel entity) – удаляет кинотеатр.

**3.2.1.4 Класс FilmService**

Класс FilmService является сервисом, который предоставляет перечень операций, совершаемых над сущностью FilmDomainModel, наследуется от класса BaseService, реализует интерфейс IFilmService.

Основные методы класса FilmService:

* Add(FilmDomainModelentity entity) – создает новый фильм;
* Read() – извлекает список фильмов;
* ReadById(object id) – извлекает фильм по идентификатору;
* Update(FilmDomainModelentity entity) – модифицирует данные фильма;
* Delete(FilmDomainModelentity entity) – удаляет фильм.

**3.2.1.5 Класс HallService**

Класс HallService является сервисом, который предоставляет перечень операций, совершаемых над сущностью HallDomainModel, наследуется от класса BaseService, реализует интерфейс IHallService.

Основные методы класса HallService:

* Add(HallDomainModel entity) – создает новый зал;
* Read() – извлекает список залов;
* ReadById(object id) – извлекает зал по идентификатору;
* Update(HallDomainModel entity) – модифицирует данные зала;
* Delete(HallDomainModel entity) – удаляет зал.

**3.2.1.6 Класс PaymentMethodService**

Класс PaymentMethodService является сервисом, который предоставляет перечень операций, совершаемых над сущностью PaymentMethodDomainModel, наследуется от класса BaseService, реализует интерфейс IPaymentMethodService.

Основные методы класса PaymentMethodService:

* Add(PaymentMethodDomainModel entity) – создает новый способ оплаты;
* Read() – извлекает список способов оплаты;
* ReadById(object id) – извлекает способ оплаты по идентификатору;
* Update(PaymentMethodDomainModel entity) – модифицирует данные способа оплаты;
* Delete(PaymentMethodDomainModel entity) – удаляет способ оплаты.

**3.2.1.7 Класс SeatService**

Класс SeatService является сервисом, который предоставляет перечень операций, совершаемых над сущностью SeatDomainModel, наследуется от класса BaseService, реализует интерфейс ISeatService.

Основные методы класса SeatService:

* Add(SeatDomainModel entity) – создает новое место;
* Read() – извлекает список мест;
* ReadById(object id) – извлекает место по идентификатору;
* Update(SeatDomainModel entity) – модифицирует данные места;
* Delete(SeatDomainModel entity) – удаляет место.

**3.2.1.8 Класс ReviewService**

Класс ReviewService является сервисом, который предоставляет перечень операций, совершаемых над сущностью ReviewDomainModel, наследуется от класса BaseService, реализует интерфейс IReviewService.

Основные методы класса ReviewService:

* Add(ReviewDomainModel entity) – создает новый отзыв;
* Read() – извлекает список мест;
* ReadById(object id) – извлекает отзыв по идентификатору;
* Update(ReviewDomainModel entity) – модифицирует данные отзыва;
* Delete(ReviewDomainModel entity) – удаляет отзыв.

**3.2.1.9 Класс ReservedSeatService**

Класс ReservedSeatService является сервисом, который предоставляет перечень операций, совершаемых над сущностью ReservedSeatDomainModel, наследуется от класса BaseService, реализует интерфейс IReservedSeatService.

Основные методы класса ReservedSeatService:

* Add(ReservedSeatDomainModel entity) – создает новый отзыв;
* Read() – извлекает список мест;
* ReadById(object id) – извлекает отзыв по идентификатору;
* Update(ReservedSeatDomainModel entity) – модифицирует данные отзыва;
* Delete(ReservedSeatDomainModel entity) – удаляет отзыв;
* ExemptExpiredSeats(long currentSeanceId) – снимает статус брони устаревших мест указанного сеанса.

**3.2.1.10 Класс SeanceService**

Класс SeanceService является сервисом, который предоставляет перечень операций, совершаемых над сущностью SeanceDomainModel, наследуется от класса BaseService, реализует интерфейс ISeanceService.

Основные методы класса SeanceService:

* Add(SeanceDomainModel entity) – создает новый сеанс;
* Read() – извлекает список мест;
* ReadById(object id) – извлекает сеанс по идентификатору;
* Update(SeanceDomainModel entity) – модифицирует данные сеанса;
* Delete(SeanceDomainModel entity) – удаляет сеанс.

**3.2.1.11 Класс Ticket**

Класс TicketService является сервисом, который предоставляет перечень операций, совершаемых над сущностью TicketDomainModel, наследуется от класса BaseService, реализует интерфейс ITicketService.

Основные методы класса TicketService:

* Add(TicketDomainModel entity) – создает новый билет;
* Read() – извлекает список билетов;
* ReadById(object id) – извлекает билет по идентификатору;
* Update(TicketDomainModel entity) – модифицирует данные билета;
* Delete(TicketDomainModel entity) – удаляет билет.

**3.3 Уровень представления**

На данном уровне проекта происходит формирование интерфейса, с которым в конечном итоге взаимодействует пользователь, а также обработка его действий, совершаемых в ходе работы пользователя с разрабатываемым приложением.

Уровень представления разделен на несколько компонентов, взаимодействующих друг с другом. В этом подразделе приведено описание работы этих компонентов, примененных шаблонов проектирования и связь уровня представления с уровнем бизнес-логики.

**3.3.1 Понимание MVC паттерна**

По большому счету MVC паттерн обозначает, что MVC приложение будет разделено как минимум на три части:

1. *Модели*, которые содержат или представляют данные, с которыми работают пользователи. Это могут быть простые *модели представления*, которые только представляют данные, передаваемые от контроллера представлению, или они могут быть *доменными моделями*, которые содержат данные домена, а также операции, преобразования и правила работы с этими данными.
2. *Представления*, которые используются для того, чтобы обработать некоторые части модели в качестве пользовательского интерфейса.
3. *Контроллеры*, которые обрабатывает входящие запросы, выполняют операции для модели и выбирают представления для показа пользователю.

В MVC контроллеры являются C# классами, как правило, производными от класса System.Web.Mvc.Controller. Каждый открытый (public) метод в классе, производный от Controller, называется методом действия, который связан с настраиваемым URL через систему маршрутизации (роутингом) ASP.NET. При отправке запроса на URL, связанный с методом действия, исполняются выражения в классе контроллера для того, чтобы выполнить некоторые операции над доменной моделью, а затем выбрать представление для отображения клиенту. На рисунке 3.2 показано взаимодействие между контроллером, моделью и представлением.



Рисунок 3.2 – Взаимодействие основных компонентов в MVC приложении

MVC – это не единственный архитектурный паттерн программного обеспечения, конечно, есть много других, и некоторые из них (или по крайней мере были) чрезвычайно популярны. Далее будут кратко описаны различные подходы к структурированию приложения и сопоставлены с MVC. Некоторые из паттернов являются близкими вариациями на тему MVC, в то время как другие совершенно отличаются.

**3.3.1.1 Паттерн Smart UI**

Один из наиболее распространенных паттернов известен как smart UI. Чтобы создать Smart UI приложение, разработчики выстраивают пользовательский интерфейс, обычно путем перетаскивания набора компонентов или элементов управления на дизайнерскую поверхность. Элементы управления сообщают о взаимодействии с пользователем, представляя события для нажатия кнопок, нажатия клавиш, движения мыши и так далее. Разработчик добавляет код в ответ на эти события в серии обработчиков событий – небольших блоков кода, которые вызываются, когда происходит определенное событие для определенного компонента. И тут заканчивается монолитное приложение, как показано на рисунке 3.3. Код, который обрабатывает пользовательский интерфейс и логика смешиваются вместе, понятия вообще не разделены. Код, который определяет допустимые значения для вводимых данных, запросов данных или изменяет учетную запись пользователя, делится на маленькие кусочки, соединенные вместе по порядку, в котором ожидаются события.



Рисунок 3.3 – Smart UI паттерн

Самым большим недостатком этой конструкции является то, что ее трудно поддерживать и расширять: смешивание доменной модели и кода бизнес логики с кодом пользовательского интерфейса приводит к дублированию, где тот же фрагмент бизнес логики копируется и вставляется для поддержки вновь добавленного компонента. Поиск всех дублирующих частей и внесение правок может быть трудным, а в сложном Smart UI приложении иногда почти невозможно добавить новую функцию, не ломая существующую. Тестирование Smart UI приложений также может быть трудным, единственный способ – это имитации взаимодействий с пользователем, которая далека от идеала и не обеспечивает основу для полного тестирования.

В мире MVC Smart UI часто называют анти-паттерном, которого следует избегать любой ценой. Эта антипатия возникла, по крайней мере, частично потому, что люди пришли к MVC в поисках альтернативы, проведя часть своей карьеры, пытаясь развивать и поддерживать Smart UI приложения. Это, безусловно, относится и к нам, и мы оба несем в себе следы этих долгих лет, но мы не отвергаем Smart UI паттерн полностью. Не все гладко в Smart UI паттерне, но есть и положительные аспекты в таком подходе. Smart UI приложения быстро и легко разрабатывать: создатели инструментария приложили много усилий для того, чтобы разработка была приятной, и даже самый неопытный программист может создать нечто профессионально выглядящее и достаточно функциональное в течение нескольких часов.

**3.3.1.2 Паттерн Model-View**

Область, в которой, как правило, возникают проблемы с поддержкой в Smart UI приложениях – это бизнес-логика, и внесение изменений может стать очень неприятным процессом. Улучшения в этой области предлагает архитектура model-view (модель-представление), которая вытаскивает бизнес-логику в отдельную доменную модель. При этом данные, процессы и правила все сосредоточены в одной части приложения, как показано на рисунке 3.4.



Рисунок 3.4 – Model-View паттерн

Архитектура model-view является значительным улучшением по сравнению со Smart UI – например, ее гораздо легче поддерживать. Тем не менее, возникают две проблемы. Первое, поскольку пользовательский интерфейс и доменная модель настолько тесно интегрированы, затрудняется выполнение юнит тестирования. Вторая проблема вытекает из практики, а не из определения паттерна. Эта модель обычно содержит массу кода доступа к данным, а это значит, что модель данных содержит не только бизнес данные, операции и правила.

**3.3.1.3 Паттерн Model-View-Presenter**

MVP является вариацией MVC, который разработан, чтобы соответствовать GUI платформам, сохраняющим состояние (stateful), таким как Windows Forms или ASP.NET Web Forms, и это стоящая попытка представить лучшие аспекты Smart UI без тех проблем, которые он обычно приносит.

В этом паттерне *презентер* (presenter) имеет те же обязанности, что и MVC контроллер, но он также имеет более непосредственное отношение к представлениям, напрямую управляя значениями, отображаемыми в UI компонентах, в зависимости от действий и данных, введенных пользователем. Существуют две реализации этого паттерна:

1. Реализация *пассивного представления*, в котором представление не содержит логики – это контейнер для элементов управления UI, которые напрямую управляются презентером.
2. Реализация *надзирающего контроллера*, в котором представление может быть ответственным за некоторые элементы логики представления, такие как связанные данные, и к источнику данных можно ссылаться из доменной модели.

Разница между этими двумя подходами касается того, насколько умным является представление. В любом случае, презентер отделяется от GUI фреймворка, что делает логику презентера простой и подходит для модульного тестирования.

**3.3.1.4 Паттерн Model-View-View Model**

MVVM паттерн является самой последней вариацией MVC. Он возник в 2005 году в команде Microsoft, которая разрабатывала технологию, ставшую Windows Presentation Foundation (WPF) и Silverlight.

В MVVM паттерне модели и представления играют те же роли, что и в MVC. Разница состоит в MVVM концепции модели представления, которая является абстрактным представлением пользовательского интерфейса. Как правило, это C# класс, который раскрывает оба свойства для данных, которые будут отображаться в интерфейсе, и операции по данным, которые могут быть вызваны из пользовательского интерфейса. В отличие от MVC контроллера, модель представления MVVM не имеет понятия, что существует представление (или любая конкретная UI технология). MVVM представление использует связывающую функцию WPF/Silverlight для двунаправленного связывания свойств, предоставляемых элементами управления в представлении (пункты в выпадающем меню или результат нажатия кнопки), со свойствами, предлагаемыми моделью представления.

MVVM тесно сотрудничает со связывающими функциями WPF и поэтому это не тот паттерн, который легко применить и к другим платформам.

**3.3.2 Описание контроллеров**

**3.3.2.1 Класс UserManipulationController**

Класс UserManipulationController управляет представлениями, которые позволяют администраторам и модераторам приложения следить и управлять ролями, правами доступа других пользователей.

Основные методы класса UserManipulationController:

* UserManagement() – возвращает представление для управления пользователями;
* SetToRole(string name) – задает роль пользователя;
* ChangeUserAccess(string name) – изменяет права доступа пользователя.

**3.3.2.2 Класс HomeController**

Класс HomeControllerr управляет представлениями, которые позволяют пользователям взаимодействовать с домашней страницей сайта и совершать базовые действия.

Основные методы класса HomeController:

* Index() – возвращает представление домашней страницы сайта;
* FilmList() – возвращает представление списка доступных фильмов;
* Details(long? filmId) – возвращает представление с подробным описанием конкретного фильма;
* Browse() – возвращает представление дат проведения киносеансов;
* CinemaRowList(string date) – возвращает представление списка кинотеатров по конкретной дате.

**3.3.2.3 Класс AccountController**

Класс AccountController управляет представлениями, которые позволяют пользователям аутентифицироваться, регистрироваться в системе, сбрасывать пароль, управлять своими учетными данными.

Основные методы класса AccountController:

* Summary() – возвращает представление с общей информацией о пользователе;
* Login(string returnUrl) – возвращает представление аутентификации пользователя в системе;
* Login(LoginViewModel model, string returnUrl) – аутентифицирует пользователя в системе;
* ExternalLoginCallback(string returnUrl) – аутентификация пользователя с помощью внешней учетной записи;
* Register() – возвращает представление регистрации пользователя в системе;
* Register(RegisterViewModel model) – аутентифицирует пользователя в системе;
* ConfirmEmail(string userId, string code) – подтверждает адрес электронной почты пользователя;
* ResetPassword(string code) – возвращает представление по сбросу пароля;
* ResetPassword(ResetPasswordViewModel model) – сбрасывает пароль пользователя;
* Manage(ManageMessageId? message) – возвращает представление по управлению учетными данными пользователя;
* Manage(ManageUserViewModel model) – сохраняет изменения учетных данных пользователя;
* LogOff()– выход пользователя из системы.

**3.3.2.4 Класс CinemaController**

Класс CinemaController дает возможность администраторам и модераторам управлять данными кинотеатров.

Основные методы класса CinemaController:

* Index() – возвращает представление со списком кинотеатров;
* Details(long? id) – возвращает представление с подробным описанием конкретного кинотеатра;
* Create() – возвращает представление с формой для создания нового кинотеатра;
* Create(CinemaViewModel cinemaViewModel) – создает новый кинотеатр;
* Edit(long? id) – возвращает представление с формой для редактирования данных кинотеатра;
* Edit(CinemaViewModel cinemaViewModel) – сохранение изменений в данных о кинотеатре;
* Delete(long? id) – возвращает представление с подтверждением удаления кинотеатра;
* DeleteConfirmed(long id) – удаляет кинотеатр.

**3.3.2.5 Класс FilmController**

Класс FilmController дает возможность администраторам и модераторам управлять данными кинотеатров.

Основные методы класса FilmController:

* Index() – возвращает представление со списком фильмов;

**5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Для обеспечения надёжности программного средства был разработан набор тестовых случаев для тестирования функциональности ПС.

Тестирование программного средства производилось на портативном персональном компьютере со следующими системными характеристиками:

– Операционная система: Windows 10 64 Bit.

– Видеокарта: NVIDIA GeForce 1070 8GB

– Процессор: Intel Core i7-6700k 4.2GHz.

– Оперативная память (ОЗУ): 16GB.

В таблице 5.1 представлены некоторые тестовые случаи, входящие в набор.

Таблица 5.1 – Тестовые случаи для разрабатываемого программного средства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
| 1 | Регистрация | 1. Выбрать пункт «Регистрация» в навигационном меню.  2. Заполнить регистрационные поля в соответствии с валидационными требованиям (см. таблица 3.5).  3. Подтвердить регистрацию нажав кнопку «Зарегистрироваться» расположенной под формой. | 1. Сайт должен открыться по адресу <имя сайта>/Account/Register, вместо формы регистрации должна появиться надпись «Регистрация произведена успешно». | 1. Сайт открылся по адресу <имя сайта>/Account/Register, вместо формы регистрации появилась надпись «Регистрация произведена успешно». |
| 2 | Вход в систему | 1. Выбрать пункт «Войти» в навигационном меню.  2. Заполнить | 1. Сайт должен открыться по адресу <имя сайта>/articles.  2. Вместо кнопок «Register» и «Login» должна появиться надпись «Login as <электронная | 1. Сайт открылся по адресу <имя сайта>/articles  2. В правом углу меню навигации появилась надпись «Login as <электронная почта пользователя>» и |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
|  |  | поля входа в соответствии с валидационными требованиям (см. таблица 3.5).  3. Подтвердить вход нажав кнопку «Login» расположенной под формой | почта пользователя>» и кнопка «Logoff». | кнопка «Logoff». |
| 3 | Выход из системы | 1. Выполнить вход в систему.  2. Нажать кнопку «Logoff» пункта навигационного меню. | 1. В правом углу меню навигации должна исчезнуть надпись «Login as <электронная почта пользователя>» и кнопка «Logoff».  2. В правом углу меню навигации должны появиться кнопки «Register» и «Login».  3. Доступ к аккаунту, а также к изменению какого-либо контента должен быть закрыт вызывать перенаправление на страницу логина. | 1. В правом углу меню навигации исчезла надпись «Login as <электронная почта пользователя>» и кнопка «Logoff».  2. В правом углу меню навигации появились кнопки «Register» и «Login».  3. Доступ к аккаунту, а также к изменению какого-либо контента закрыт и вызывает перенаправление на страницу логина. |
| 4 | Создание статьи обсуждения | 1. Выполнить вход в систему.  2. Выбрать подпункт «Create article» навигационного меню.  3. Заполнить поля «Title», «Header», «Content», а | 1. Сайт должен открыться по адресу <имя сайта>/articles.  2. В разделе «Latest blog posts» на первом месте должно быть превью («Title», «Header», картинка) созданной статьи. | 1. Сайт открылся по адресу <имя сайта>/articles  2. В разделе «Latest blog posts» на первом месте находится превью («Title», «Header», картинка) вновь созданной статьи. |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
|  |  | также добавить картинку через кнопку «Choose a header picture».  4. Нажать кнопку «Create». |  |  |
| 5 | Просмотр статьи | 1. Перейти на главную страницу портала.  2. Нажать на кнопку «Continue» статьи которую нужно просмотреть. | 1. Сайт должен открыться по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи>.  2. Дополнительно к превью статьи появится основной текст и комментарии. | 1. Сайт открылся по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи>.  2. Дополнительно к превью статьи появился основной текст и секция комментариев. |
| 6 | Изменение статьи | 1. Выполнить вход в систему под пользователем с правами модератора или пользователем, который является автором статей.  2. Нажать на кнопку «Edit post» статьи которую нужно отредактировать и к которой есть доступ у текущего пользователя.  3. Выполнить изменения статьи и подтвердить, нажав кнопку «Edit». | 1. Сайт должен открыться по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи>/edit.  2. На странице должны быть текстовые поля с уже имеющимся текстом статьи.  3. Подтвердив изменение сайт должен открыться по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи> с изменённой статьёй. | 1. Сайт открылся по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи>/edit.  2. На странице присутствуют текстовые поля, заполненные текстом статьи.  2. При нажатии кнопки «Edit» сайт открылся по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи> с изменённой статьёй. |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
| 7 | Удаление статьи | 1. Выполнить вход в систему под пользователем с правами модератора или пользователем, который является автором статей.  2. Нажать на кнопку «Delete post» статьи которую нужно удалить и к которой есть доступ у текущего пользователя.  3. Подтвердить удаление нажав кнопку «Delete». | 1. Сайт должен открыться по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи>/delete.  2. На странице должна быть кнопка подтверждения удаления.  3. Подтвердив удаление сайт должен открыться по адресу <имя сайта>/articles. В ленте статей не должно быть удалённой статьи. | 1. Сайт открылся по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи>/delete.  2. На странице присутствует кнопка подтверждения удаления.  3. При нажатии кнопки «Delete» сайт открылся по адресу <имя сайта>/articles. В ленте статей отсутствует удалённая статья. |
| 8 | Комментирование статьи | 1. Выполнить вход в систему.  2. Зайти на страницу статьи.  3. Написать комментарии в секции под основным текстом статьи. | 1. После заполнения текстового поля для комментария и нажатия кнопки «Comment», новый комментарий должен вставиться в начало секции комментариев без перезагрузки страницы.  2. Возле созданного комментария должны появиться иконки, означающие изменение и удаление. | 1. После заполнения текстового поля для комментария и нажатия кнопки «Comment», новый комментарий вставился в начало секции комментариев без перезагрузки страницы.  2. Возле созданного комментария появились иконки, означающие изменение и удаление. |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
| 9 | Поиск статей | 1. Перейти на главную страницу портала.  2. В текстовом поле поиска расположенном в правом верхнем углу ввести выражение, которое необходимо найти. | 1. Сайт должен открыться по адресу <имя сайта>/articles.  2. При вводе выражение в текстовое поле поиска, лента статей должна замениться на результаты поиска. | 1. Сайт должен открыться по адресу <имя сайта>/articles.  2. После ввода выражения в текстовое поле поиска, лента статей заменилась на статьи, подходящие по критерию поиска. |
| 10 | Создание пользователя | 1. Выполнить вход в систему под пользователем с правами администратора.  2. Перейти на страницу со списком пользователей.  3. Перейти на страницу создания пользователя путём нажатия на кнопку «Create».  4. Заполнить поля почты, имени, пароля, роли. | 1. После создания пользователя сайт должен открыться по адресу <имя сайта>/users.  2. В списке пользователей должен находиться созданный пользователь. | 1. После создания пользователя сайт открывается по адресу <имя сайта>/users.  2. В списке пользователей должен созданный пользователь. |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
| 11 | Изменение пользователя | 1. Выполнить вход в систему под пользователем с правами администратора.  2. Зайти на пользовательский профиль.  2. Нажать на кнопку «Edit».  3. Выполнить изменения статьи и подтвердить, нажав кнопку «Edit». | 1. Сайт должен открыться по адресу <имя сайта>/users/<номер пользователя>/<заголовок статьи>/edit.  2. На странице должны быть текстовые поля с уже имеющимся текстом статьи.  3. Подтвердив изменение сайт должен открыться по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи> с изменённой статьёй. | 1. Сайт открылся по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи>/edit.  2. На странице присутствуют текстовые поля, заполненные текстом статьи.  2. При нажатии кнопки «Edit» сайт открылся по адресу <имя сайта>/articles/<номер статьи>/<заголовок статьи> с изменённой статьёй. |
| 12 | Удаление пользователя | 1. Выполнить вход в систему под пользователем с правами администратора.  2. Перейти на страницу пользователя.  3. Нажать на кнопку «Delete user».  4. Подтвердить удаление нажав кнопку «Delete». | 1. Подтвердив удаление сайт должен открыться по адресу <имя сайта>/users. В списке пользователей не должно быть удалённого пользователя. | 1. Подтвердив удаление сайт открылся по адресу <имя сайта>/users. В списке пользователей нет удалённого пользователя. |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
| 13 | Изменение профильной картинки | 1. Выполнить вход в систему.  2. Выбрать подпункт «Account» навигационного меню.  3. Нажать кнопку «Edit account».  4. Загрузить новую профильную картинку и нажать «Update picture», | 1. Сайт должен открыться по адресу <имя сайта>/account/edit.  2. При загрузке картинки и нажатии кнопки «Edit account» должно быть перенаправление на страницу аккаунта.  3. Профильная картинка должна быть обновлена. | 1. Сайт открылся по адресу <имя сайта>/account/edit.  2. При загрузке картинки и нажатии кнопки «Edit account» произошло перенаправление на страницу аккаунта.  3. Профильная картинка обновилась. |
| 14 | Выбор ответа на вопрос | 1. Выполнить вход в систему.  2. Создать или перейти на страницу вопроса, автором которого является авторизованный пользователь.  3. Выбрать ответ на вопрос нажав кнопку «Mark as answer». | 1. Сайт должен открыться по адресу <имя сайта>/question/  <номер вопроса>/  <заголовок вопроса>.  2. Возле ответов должен быть кнопки «Mark as answer».  3. При нажатии кнопки выбранный ответ визуально изменяется от остальных ответов. | 1. Сайт открылся по адресу <имя сайта>/question/  <номер вопроса>/  <заголовок вопроса>.  2. Возле ответов присутствуют кнопки «Mark as answer».  3. При нажатии кнопки выбранный ответ визуально изменился от остальных ответов. |

1. **МЕТОДИКА ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО СРЕДСТВА**

Для запуска программы необходимо ввести «http://localhost:{port}/» или доменное имя сайта, если такое есть, в адресной строке браузера. После ввода адреса, откроется главная страница приложения (см. рисунок 6.1).

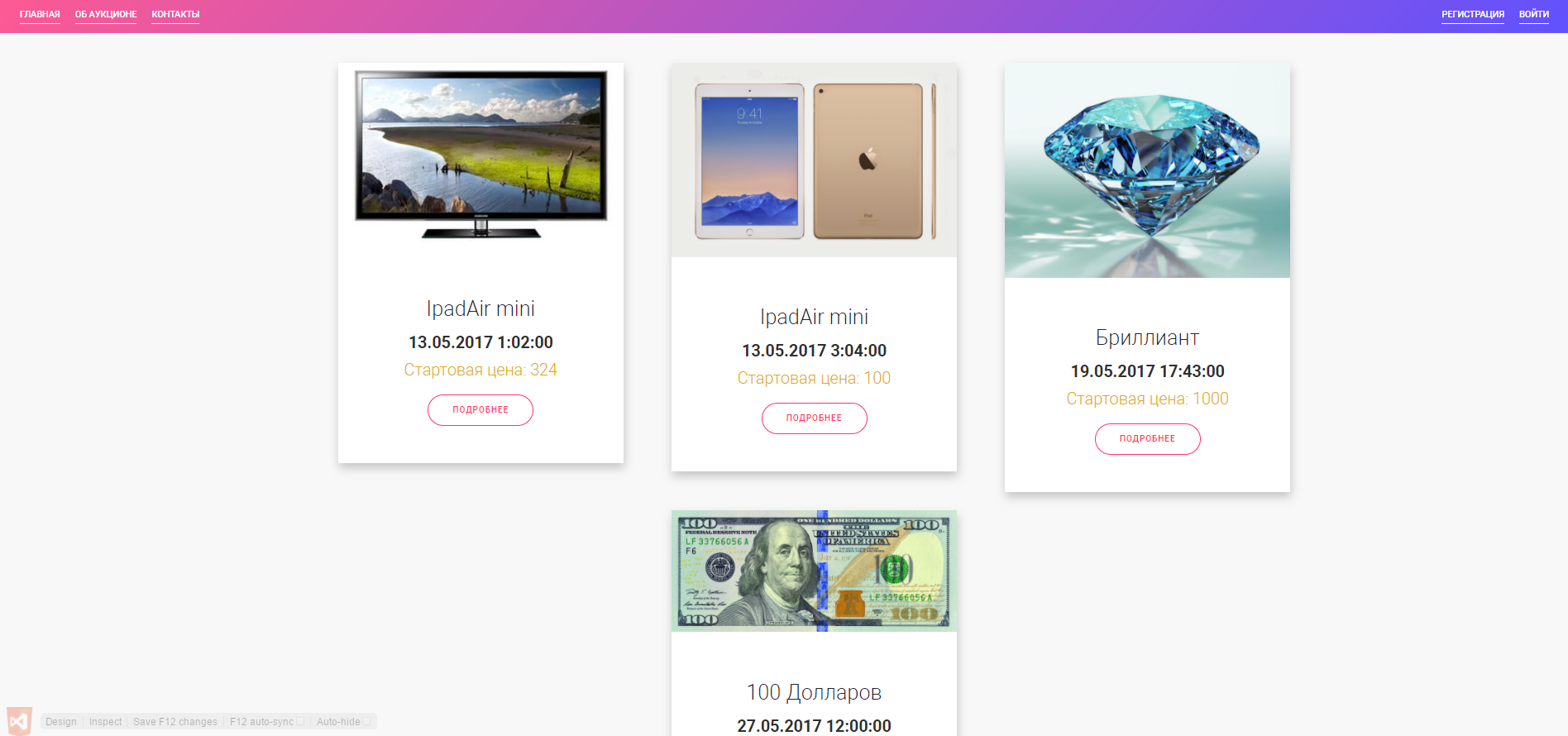


Рисунок 6.1 - Главная страница веб-сайта

Главная страница веб-сайта состоит из нескольких частей: навигационного меню и списка доступных аукционов.

**6.1 Навигационное меню**

Навигационное меню также состоит из нескольких частей. Основная часть – кнопки навигации такие как «Главная», «Об аукционе», «Контакты», «Регистрация», «Войти». В навигационном меню также имеется дополнительная функциональность доступная только авторизированным пользователям. Дополнительное поле состоит из таких кнопок как «Аукционы», «Добавить аукцион», «Завершившиеся аукционы», «Купить биды» и последняя часть содержит текущее количество бидов у пользователя.

На рисунке 6.2 показано навигационное меню которое отображается для анонимных пользователей. На рисунке 6.3 показано навигационное меню которое отображается для авторизированных пользователей.



Рисунок 6.2 - Навигационное меню для анонимного пользователя

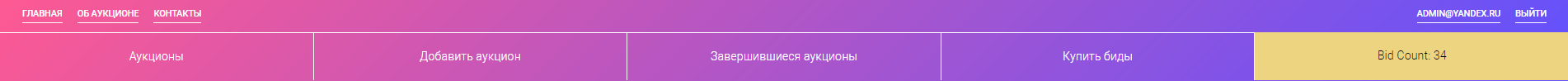


Рисунок 6.3 - Навигационное меню для авторизированного пользователя

**6.3 Процесс аукциона**

Для того чтобы участвовать в аукционе авторизованный пользователь должен перейти на страницу информации об аукционе (рисунок 6.4). Эта страница состоит из трёх частей:

а) Левая часть содержит краткую информацию о выставляемом лоте;

б) В центральной части пользователь может увидеть таймер окончания аукциона, текущую стоимость лота, имя пользователя, который сделал последнюю ставку и кнопку с помощью которой можно сделать ставку;

в) В правой части находится список пользователей делавших ставки. В ячейке выделенной зелёным фоном находится имя пользователя, который сделал последнюю ставку.

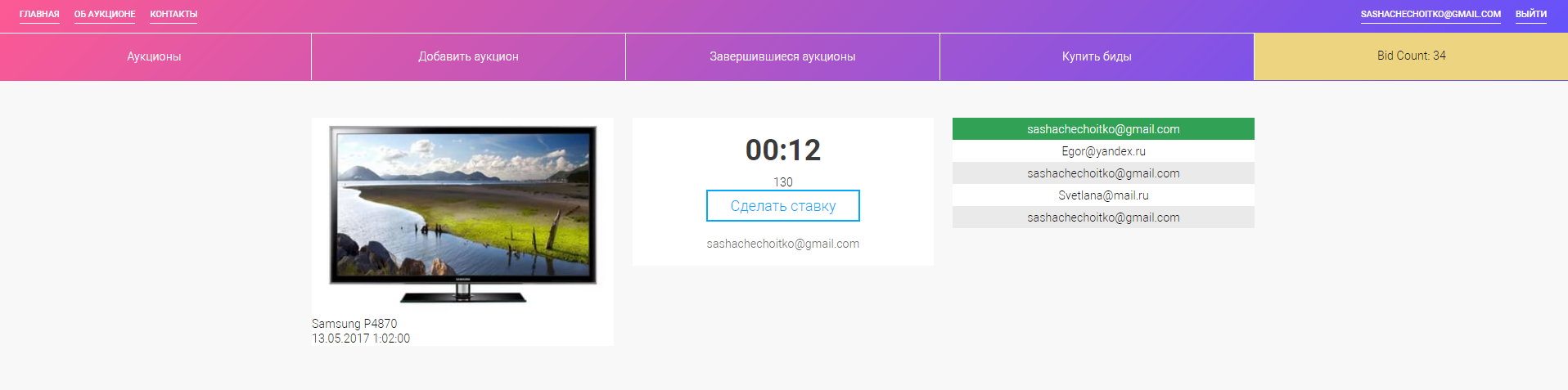


Рисунок 6.7 – Процесс аукциона

* 1. **Создание аукциона**

Для того чтобы создать аукцион, авторизованный пользователь должен перейти на страницу создания аукциона. В предложенные поля пользователь должен ввести данные для создания аукциона. На рисунке 6.3 показана форма для ввода данных.

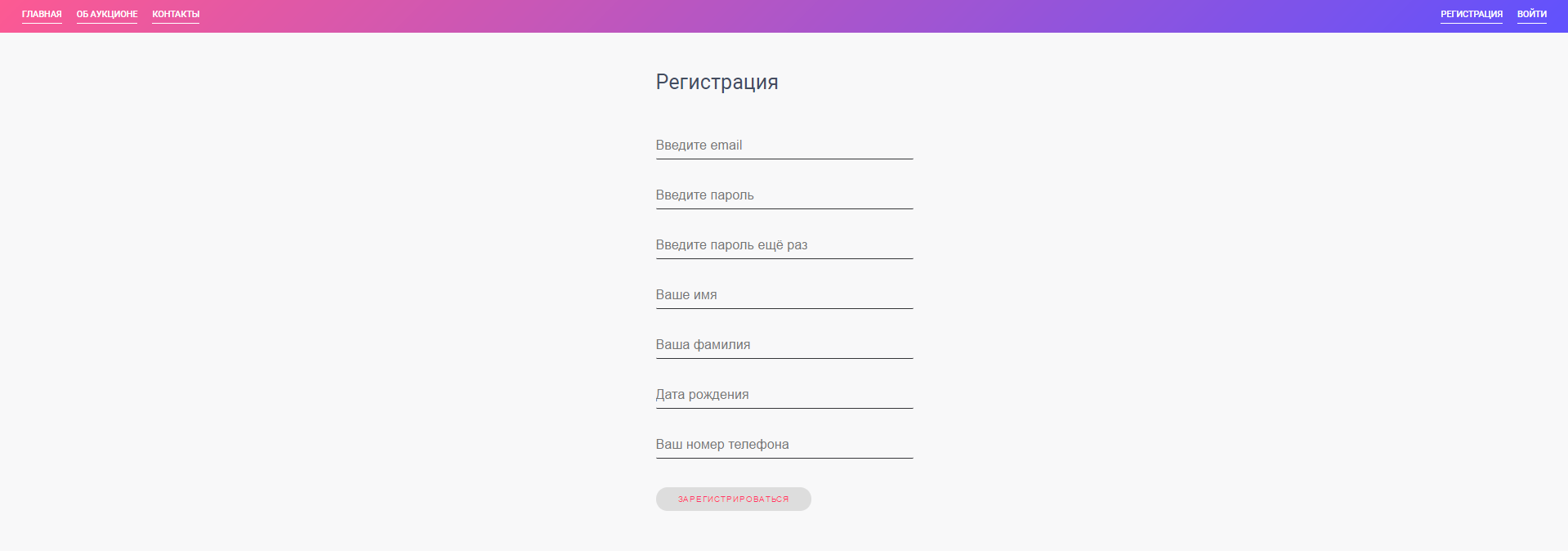


Рисунок 6.4 – Создание аукциона

**ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА**

**7.1 Описание проекта**

Целью настоящего проекта явилось создание интерактивной системы для организации процесса проведения скандинавского аукциона и управления сопутствующими процессами. По задумке проект включает наиболее успешные практики работы с пользователями с использованием информационных технологий.

Основная цель данной системы – автоматизация процесса проведения скандинавского аукциона.

Клиенты могут удаленно посредством разрабатываемого модуля участвовать в аукционе и делать ставки на выставленный лот. А также выставлять свои товары на аукцион. Это позволяет клиентам зарабатывать деньги, не выходя из дома.

Таким образом, положительный экономический эффект у заказчика может быть получен вследствие:

* уменьшения бумажного документооборота;
* автоматизации процесса проведения аукциона;
* прибыль от проведённого аукциона;
* повышения качества обслуживания клиентов.

Целью технико-экономического обоснования программного приложения является определение экономической выгодности создания и реализации данного продукта. В данном разделе дипломного проекта определяется его экономическая эффективность, для этого рассчитываются затраты на производство и реализацию программы, цена и прибыль от продаж.

**7.2 Расчёт сметы затрат на проект**

На основании сметы затрат и анализа рынка ПС определяется плановая отпускаемая цена.

Базой для расчета плановой сметы затрат на разработку ПО является объем ПО. Общий объем программного продукта Vо, LOC, определяется по формуле:



где Vi – объем отдельной функции ПО, LOC;

n – общее число функций.

На основании информации о функциях разрабатываемого ПО по каталогу функций определяется объем функций и общий объем ПО, который уточняется с учетом условий разработки ПО в организации. Рассчитывается уточненный объем ПО Vу, LOC, по формуле:



Все функции, реализованные в проекте «Скандинавский аукцион», представлены в таблице 7.1. Язык программирования для разработки – C#.

Таблица 7.1 – Функции программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| N функци | Наименование функции | Объем функции (LОС) | |
| по каталогу Vi | уточненный Vyi |
| 101 | Организация ввода информации | 510 | 405 |
| 102 | Контроль, предварительная обработка и ввод информации | 570 | 485 |
| 201 | Генерация структуры базы дан- ных | 2000 | 1310 |
| 208 | Организация поиска и поиск в базе данных | 1720 | 1560 |
| 506 | Обработка ошибочных и сбой- ных ситуаций | 1040 | 850 |
| 506 | Обеспечение интерфейса между  компонентами | 1680 | 950 |
| 703 | Расчет показателей | 375 | 280 |
| 707 | Графический вывод результатов | 420 | 395 |
| 709 | Измерение состояния ресурсов в  интерактивной системе | 570 | 475 |
| 811 | Администрирование и обновление сайта | 240 | 145 |
| Итог | | 9125 | 6855 |

**7.3 Расчёт нормативной трудоемкости**

По уточненному объему ПО и нормативам затрат труда в расчете на единицу объема определяются нормативная и общая трудоемкость разработки ПО.

На основании принятого к расчету объема (Vу) и категории сложности определяется нормативная трудоемкость ПО (Тн) по формуле:

Tо = Tн⋅Kс⋅Kт⋅Kн ,

где Kc − коэффициент, учитывающий сложность ПО;

Kт − поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей;

Kн − коэффициент, учитывающий степень новизны ПО.

Коэффициент сложности (Кс) учитывает дополнительные затраты труда, связанные со сложность разрабатываемого программного продукта и вычисляется по формуле:



где Кi – коэффициент повышения сложности ПО;

n – количество учитываемых характеристик.

Данное ПО отнесено ко 3-ей группе сложности, поэтому коэффициент сложности Кс = 1 + 0,12 = 1,12.

Коэффициент, учитывающий степень использования при разработке ПО стандартных модулей (Кт) показывает удельный вес стандартных модулей в разрабатываемом ПО в общем объеме проектируемого продукта. При разработке данного ПО доля используемых стандартных модулей превысила 50%, а Кт = 0,5.

Коэффициент новизны разрабатываемого ПО (Кн) позволяет сравнить характеристики разрабатываемого ПО с имеющимися аналогами и определить экспертным путем степени его новизны. В данном случае коэффициент новизны равен 0,8.

Для ПС 3-ой группы сложности, к которой относится разрабатываемый программный продукт, нормативная трудоемкость составит 278 человеко-дней.

Общая трудоёмкость To равна:

To =

Эффективный фонд времени работы одного работника (Фэф), определятся по формуле:

Фэф = Дг – Дп – Дв – До,

где Дг – количество дней в году, дн.;

Дп – количество праздничных дней в году, дн.;

Дв – количество выходных дней в году, дн.;

До – количество дней отпуска, дн.

Согласно данным, приведенным в производственном календаре для пятидневной рабочей недели в 2017 году для Беларуси, фонд рабочего времени составит:

Фэф = 365 – 7 – 103 – 24 = 231 дн.

Численность исполнителей проекта (Чр), чел, рассчитывается по формуле:



где Тр  – срок разработки проекта, лет.

Рассчитаем численность исполнителей проекта, учитывая, что срок разработки проекта не более четырёх месяцев:

Это означает, что один человек будет занят на проекте на протяжении четырёх месяцев, т.е. инженер-программист будет работать полный рабочий день в течение четырёх месяцев на данном проекте.

**7.4 Расчёт основной заработной платы исполнителей**

Основная заработная плата исполнителей, занятых разработкой программного продукта, (Зо) рассчитывается по формуле:

**

где n – количество исполнителей, занятых разработкой программного продукта;

Tчі – часовая тарифная ставка i-го исполнителя (тыс. руб.);

Фэi – эффективный фонд рабочего времени i-го исполнителя (дней);

Тч – количество часов работы в день (ч);

К – коэффициент премирования.

Разработкой программного продукта «Скандинавский аукцион» занимался один исполнитель: инженер-программист без категории (3,63), эффективный фонд рабочего времени 4 месяца или 88 рабочих дней. Количество часов работы в день — 8 часов. Тарифная ставка 1-ого разряда – 298 тыс. руб. Коэффициент премирования — 1,3.

Исходя из полученных данных, рассчитаем основную заработную плату исполнителя. Полученные данные приведены в таблице 7.2.

Таблица 7.2 — Разряды, ставки и тарифные коэффициенты работников

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование должности | Тарифный коэффициент | Разряд | Месячная тарифная ставка, тыс. руб. | Часовая тарифная ставка, тыс. руб. |
| Инженер-программист без категории | 3, 63 | 10 | 1 082 | 6,1 |

Рассчитаем основную заработную плату:

Дополнительная заработная плата исполнителей, занятых разработкой программного продукта, (Зд) рассчитывается по формуле:

,

где Нд – норматив дополнительной заработной платы.

Рассчитаем дополнительную заработную плату (норматив дополнительной заработной платы равен 20%):

Отчисления в фонд социальной защиты населения (Зсз) определяются в соответствии с действующими законодательными актами по формуле:

,

где Зд – дополнительная заработная плата (руб.);

Нсз – норматив отчислений в фонд социальной защиты населения.

Рассчитаем отчисления в фонд социальной защиты (норматив отчислений в фонд социальной защиты населения равен 34%):

Отчисления по обязательному страхованию определяются по нормативу:



Рассчитаем отчисления по обязательному страхованию (норматив отчислений по обязательному страхованию равен 0,6%):

Расходы по статье «Материалы» определяются по формуле:



Норму расхода материалов в суммарном выражении (НМ) возьмем в процентах от основной заработной платы разработчиков (3%).

Расходы по статье «Машинное время» (Рм) включают оплату машинного времени, необходимого для разработки и отладки программного продукта определяются по формуле:



где ti – время работы над программным продуктом i-ого разработчика;

Цм – цена одного машино-часа;

li –количество разработчиков.

Рассчитаем расходы по статье «Машинное время» (рыночная цена машино-часа равна 5 тыс. руб.):

Рм =

Расходы на научные командировки работников ПО в рамках данного проекта не предусмотрены.

Затраты по статье «Накладные расходы» (Рн) рассчитываются по формуле:

,

где Нрн – норматив накладных.

Рассчитаем расходы по статье «Накладные расходы» (норматив накладных расходов в организации равен 5%):

Общая сумма расходов по всем статьям сметы (Ср) на программный продукт рассчитывается по формуле:



Рассчитаем сумму расходов по всем статьям сметы:

Ср =

Затраты на сопровождение и адаптацию программного продукта, которые определяются по нормативу (НРСА):



где Нрас – норматив расходов на сопровождение и адаптацию.

Рассчитаем затраты на сопровождение и адаптацию (норматив расходов на сопровождение и адаптацию равен 10%):

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость программного продукта (СП) определяется по формуле:



Общая сумма расходов равна:

Прибыль от программного продукта рассчитывается по формуле:



где Ппсi – прибыль от реализации кроссплатформенного интерфейса заказчику (руб.);

УРпi – уровень рентабельности программного продукта;

СПi – себестоимость программного продукта (руб.).

Прибыль от программного продукта (уровень рентабельности программного продукта примем 30%):

Прогнозируемая отпускная цена без налогов рассчитывается по формуле:



Рассчитаем налог на добавочную стоимость (НДС) по формуле:

Согласно действующему законодательству, норматив налога на добавленную стоимость равен 20%:

Прогнозируемая отпускная цена рассчитывается по формуле:

**7.5 Расчёт капитальных вложений заказчика**

Общие капитальные вложения (Ко) заказчика (потребителя), связанные с приобретением, внедрением и использованием программного продукта, рассчитываются по формуле:

Ко = Кпр + Кос + Ктс + Коб

где Кпр – затраты пользователя на приобретение системы по отпускной цене разработчика с учетом стоимости услуг по эксплуатации и сопровождению ();

Кос – затраты пользователя на освоение программного продукта (руб.).

Затраты на освоение программного продукта рассчитываются по формуле:

Кос = Зсм Кнз Чпо Тос + Цм Тмос

где Зсм – среднемесячная заработная плата работающего оператора, использующего систему (тыс. руб.); Кнз – коэффициент начислений на зарплату (ед.); Чпо – численность операторов, занятых освоением программного продукта (чел.); Тос – продолжительность освоения (мес.); Цм – цена одного машино-часа работы ЭВМ (тыс. руб.); Тмос – расход машинного времени на освоение программного продукта (машино-часов).

Общие капитальные вложения заказчика равны:

**7.6 Расчёт экономии ресурсов за счёт внедрения программного продукта в производство**

В качестве примера предположим, что программный продукт будет использоваться внутри предприятия-заказчика на 3 автоматизированных рабочих местах и каждый оператор будет использовать в среднем 4 раза день.

Плановый объем выполненных работ за год:

A2 = 3231= 2 772 (задач).

Таблица 5.3 – Исходные данные

|  |  |
| --- | --- |
| Название | Значение |
| Количество операторов | 3 |
| Месячная заработная плата оператора (тыс. руб.) | 3 400 |
| Процент дополнительной заработной платы | 30,00% |
| Количество задач на одного сотрудника за месяц | 78 |
| Количество задач на одного сотрудника за год | 936 |

Экономия затрат на заработную плату в расчете на 1 задачу (Cзe):

,

где Зcм− среднемесячная заработная плата одного маркетолога (тыс. руб.);

Tc1, Tc2− снижение трудоемкости работ в расчете на 1 задачу (человеко-часов);

Tч− количество часов работы в день (ч);

Дp− среднемесячное количество рабочих дней.

Экономия заработной платы при использовании нового программного обеспечения (тыс. руб.):

Сз=СзеА2,

где Сз− экономия заработной платы;

А2− количество типовых задач, решаемых за год (задач).

Cз= = 53499 тыс. руб.

Экономия с учетом дополнительной заработной платы (Сн):

Cн= = 69549 тыс. руб.

Экономия за счет сокращения простоев сервиса (Сс) рассчитывается по формуле:

где Дрг – количество рабочих дней в году;

Сп – стоимость 1 часа простоя сервиса, тыс.руб.

Общая годовая экономия текущих затрат:

Co= Cн +Сс= 69549 + 1 540 = 71089 тыс. руб.

**7.7 Расчёт экономического эффекта**

Внедрение нового программного продукта позволит пользователю сэкономить на текущих затратах, т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль − дополнительная прибыль, остающаяся в его распоряжении (ΔПЧ), которая определяется по формуле:

,

где Hп− ставка налога на прибыль равная 18%.

В процессе использования нового программного продукта чистая прибыль в конечном итоге возмещает капитальные затраты. Однако, полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2017 год) путем умножения результатов и затрат за каждый год на коэффициент приведения (), который рассчитывается по формуле:



где Ен – норматив приведения разновременных затрат и результатов;

tp – расчетный год, tp = 1;

t – номер года, результаты и затраты которого приводятся к расчетному.

Норматив приведения разновременных затрат и результатов (Енвт) для программных средств ВТ в существующей практике принимается равным 0,2. Следовательно, при решении данной задачи коэффициентам приведения () по годам будут соответствовать следующие значения:

- расчетный год

- 2018 год

- 2019 год

- 2020 год

Результаты расчета показателей эффективности приведены в таблице 7.3

Рассчитаем рентабельность инвестиций в приобретение программного продукта (РИ) по формуле:



где ПЧСР - среднегодовая величина чистой прибыли за расчетный период, тыс. руб., которая определяется по формуле:

,

где Пчt - чистая прибыль, полученная в году t, тыс. руб.

Таблица 7.3 - Расчет экономического эффекта от программного продукта

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Единица  измерения | Годы | | | |
| 2017 | 2018 | 2019 | 2020 |
| Результаты: |  |  |  |  |  |
| Прирост прибыли за счет экономии затрат (Пч) | тыс. руб. |  |  |  |  |
| То же с учетом фактора времени | тыс. руб. |  | 49 011,5 | 40 744,5 | 34 249 |
| Затраты: |  |  |  |  |  |
| Приобретение ПО (Kпp) | тыс. руб. |  |  |  |  |
| Освоение ПО (Koc) | тыс. руб. | 465 |  |  |  |
| Всего затрат: | тыс. руб. |  |  |  |  |
| Экономический эффект: |  |  |  |  |  |
| Превышение результата над затратами | тыс. руб. | - | 49 011,5 | 40 744,5 | 34 249 |
| То же с нарастающим итогом | тыс. руб. | - |  | 67015,4 | 101264,4 |
| Коэффициент приведения | ед. | 1 | 0,83 | 0,69 | 0,58 |

В результате технико-экономического обоснования применения программного продукта были получены следующие значения показателей их эффективности:

1. Среднегодовая величина чистой прибыли:
2. Чистый дисконтированный доход за четыре года производства продукции составит 101264,4тыс. руб.
3. Все инвестиции окупаются на второй год использования программного продукта.
4. Рентабельность инвестиций в приобретение программного продукта:

**7.8 Вывод по технико-экономическому обоснованию**

Таким образом было произведено технико-экономическое обоснование разрабатываемого проекта, составлена смета затрат и рассчитана прогнозируемая прибыль, и показана экономическая целесообразность разработки.

Полученные результаты свидетельствуют об эффективности разработки и внедрения в эксплуатацию веб-ориентированного программного средства «Скандинавский аукцион».

**ЗАКЛЮЧЕНИЕ**

В ходе дипломного проектирования была разработана интерактивная система для организации процесса проведения скандинавского аукциона и управления сопутствующими процессами. Были применены нестандартные подходы к автоматизации процесса бронирования билетов и контроля над инфраструктурой системы.

Для хранения информации была построена база данных на сервере Microsoft SQL Server 2016. Интеграция приложения с внешним хранилищем данных реализована с использованием объектно-реляционного отображателя Entity Framework 6.

Основа приложения построена на базе платформы ASP.NET MVC. Проект технически выполнен в виде сайта и может быть развернут на сервере, управляемом операционной системой Microsoft Windows Server 2008.

Была достигнута главная цель разработки – проведение аукциона в режиме реального времени.

В данном проекте акцент был сделан на создание мощной и легко расширяемой архитектуры. В качестве усовершенствования приложения в дальнейшем планируется:

* расширение системы для достижения более высоких показателей быстродействия и устойчивости к нагрузкам;
* организация обратной связи между пользователем и администрацией с целью обмена дополнительной информацией и корректировки обнаруженных изъянов;
* реализация системы оповещений пользователя о скором начале аукциона.
* возможность использования других интернет-кошельков.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Система вопросов и ответов [Электронный ресурс]. – Электронные данные. – Режим доступа https://en.wikipedia.org/wiki/Система\_вопросов\_и\_ответов\_(веб-сервис) – Дата доступа: 15.03.2017

[2] Хабрахабр [Электронный ресурс]. – Электронные данные. – Режим доступа https://en.wikipedia.org/wiki/Хабрахабр – Дата доступа: 15.03.2017

[3] Stack Overflow [Электронный ресурс]. – Электронные данные. – Режим доступа https://en.wikipedia.org/wiki/Client%E2%80%93server\_model – Дата доступа: 15.03.2017

[4] Client-Server [Электронный ресурс]. – Электронные данные. – Режим доступа https://ru.wikipedia.org/wiki/Stack\_Overflow – Дата доступа: 15.03.2017

[5] Reddit [Электронный ресурс]. – Электронные данные. – Режим доступа https://ru.wikipedia.org/wiki/Reddit – Дата доступа: 15.03.2017

[6] REST [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/REST – Дата доступа: 15.03.2017

[7] MVC [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Model-View-Controller – Дата доступа: 15.03.2017

[8] Спиральная модель [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Спиральная\_модель – Дата доступа: 15.03.2017

[9] Трех уровневая архитектура [Электронный ресурс]. – Электронные данные. – Режим доступа: https://en.wikipedia.org/wiki/Multitier\_architecture – Дата доступа: 15.03.2017

[10] Введение в язык C# и .NET Framework [Электронный ресурс]. – Электронные данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/z1zx9t92.aspx – Дата доступа: 15.03.2017

[11] Д. Рихтер. Программирование на платформе Microsoft .NET Framework [Текст]: учебное пособие; Изд-во: Русская Редакция, 2002 г., 486с.

[12] А. Фримен. ASP.NET MVC 4 с примерами на C# 5.0 для профессионалов [Текст]: учебное пособие; Изд-во: Вильямс, 2090 г., 274с

[13] Д. Лерман. Enity Framework [Текст]: учебное пособие; Изд-во: Вильямс, 2013 г., 688с

[14] Visual Studio [Электронный ресурс]. – Электронные данные. – Режим доступа: http://devopswiki.net/index.php/Visual\_Studio – Дата доступа: 15.03.2016

[15] Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. [Приемы объектно-ориентированного проектирования. Паттерны проектирования](https://ru.wikipedia.org/wiki/Design_Patterns). — [Addison-Wesley](https://ru.wikipedia.org/wiki/Addison-Wesley), [«Питер»](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%82%D0%B5%D1%80_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)), 1994. —  395с.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Текст программного модуля контроллера статей**