**СОДЕРЖАНИЕ**

[Введение 8](#_Toc482845033)

[1 Анализ предметной области 9](#_Toc482845034)

[1.1 Особенности «Скандинавских» торгов 9](#_Toc482845035)

[1.2 Сравнение аналогов 9](#_Toc482845036)

[1.3 Архитектура клиент-сервер 10](#_Toc482845037)

[1.4 REST-сервис 12](#_Toc482845038)

[1.5 Шаблон проектирования Model View Controller 13](#_Toc482845039)

[1.6 Выбор модели жизненного цикла 16](#_Toc482845040)

[1.7 Выбор платёжной системы 16](#_Toc482845041)

[2 Fнализ предметной области и разработка модели](#_Toc482845042) [программного средства 19](#_Toc482845043)

[2.1 Описание функциональности программного средства 19](#_Toc482845044)

[2.2 Спецификация функциональных требований. 20](#_Toc482845045)

[3 Проектирование программного средства 22](#_Toc482845046)

[3.1 Архитектура программного модуля 22](#_Toc482845047)

[3.2 Структура программного модуля 24](#_Toc482845048)

[3.3 Алгоритмы, положенные в основу программного средства 26](#_Toc482845049)

[4 Техническое проектирование программного](#_Toc482845050) [средства 32](#_Toc482845051)

[4.1 Используемые технологии 32](#_Toc482845052)

[4.2 Описание проектной структуры 41](#_Toc482845053)

[4.3 Уровень доступа к базе данных 43](#_Toc482845054)

[4.4 Описание сервисов уровня бизнес-логики 46](#_Toc482845055)

[4.5 Выбор подхода доступа к данным 47](#_Toc482845056)

[4.6 Описание моделей уровня доступа к данным 50](#_Toc482845057)

[5 Тестирование программного средства 53](#_Toc482845058)

[6 Методика использования программного](#_Toc482845059) [средства 56](#_Toc482845060)

[6.1 Навигационное меню 56](#_Toc482845061)

[6.2 Процесс аукциона 57](#_Toc482845062)

[6.3 Создание аукциона 57](#_Toc482845063)

[7 Технико-экономическое обоснование проекта 58](#_Toc482845064)

[7.1 Описание проекта 58](#_Toc482845065)

[7.2 Расчёт сметы затрат на проект 58](#_Toc482845066)

[7.3 Расчёт нормативной трудоемкости 59](#_Toc482845067)

[7.4 Расчёт основной заработной платы исполнителей 61](#_Toc482845068)

[7.5 Расчёт капитальных вложений заказчика 65](#_Toc482845069)

[7.6 Расчёт экономии ресурсов за счёт внедрения программного продукта в производство 66](#_Toc482845070)

[7.7 Расчёт экономического эффекта 67](#_Toc482845071)

[7.8 Вывод по технико-экономическому обоснованию 70](#_Toc482845072)

[Заключение 71](#_Toc482845073)

[Список использованных источников 72](#_Toc482845074)

[Приложение А. Исходный код программы 73](#_Toc482845075)

**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

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

Бид (англ. Bid; предлагаемая цена) – цена спроса, наивысшая цена покупателя, по которой он согласен купить валюту, ценные бумаги и т. п. активы.

# **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**

**1.1** 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:

а) 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.

б) Getbuy.ru – красивый сайт, много аукционов, но имеется наличие ботов или подставных лиц.

в) Vauctione.ru – молодой и красивый сайт. Недостатки сайта: нет помощи, форума, официального договора публичной оферты.

г) Dominanta.ru – достаточно сырой и не продуманный сайт (см. рисунок 1.1), яркий пример одной из многих копий gagen.ru.

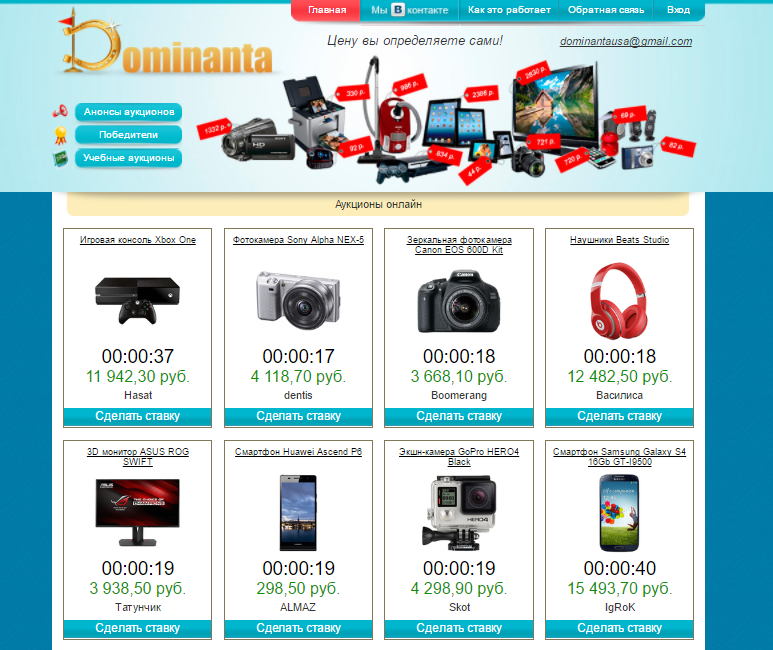


Рисунок 1.1 – Веб-сайт «Dominanta»

д) Minilot.r  – копия gagen.ru, но там организаторы пытаются предоставить честные условия.

е) Tvoypriz.ru – неплохой аукцион. Но, полностью отсутствует прозрачность торгов, нет чата или форума. Небольшие технические возможности.

ж) 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 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» изображённая на рисунке 1.2.

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 1.2 – Архитектура клиент-сервер

Отображением результатов запросов, а также приемом данных от клиента и их передачей на сервер обычно занимается 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:

а) приём данных от пользователя и сохранение их на сервере;

б) выполнение различных действий по запросу пользователя: извлечение данных из базы данных (БД), добавление, удаление, изменение данных в БД, проведение сложных вычислений;

в) аутентификация пользователя и отображение интерфейса системы, соответствующего данному пользователю;

г) отображение постоянно изменяющейся оперативной информации.

Основными достоинствами архитектуры «клиент-сервер» являются:

д) возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети, что позволяет упростить обслуживание вычислительной системы;

е) все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов;

ж) на сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа;

з) позволяет объединить различные клиенты;

и) 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 (см. рисунок 1.3).



Рисунок 1.3 – 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).

На рисунке 1.4 изображена модель шаблона проектирования MVC.



Рисунок 1.4 – 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 на таблице 1.1.

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:

а) оплата со счета мобильного телефона (m-commerce);

б) оплата по картам предоплаты (скретч-картам);

в) оплата электронными деньгами;

г) оплата платежными картами;

д) 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** **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА МОДЕЛИ**

# **ПРОГРАММНОГО СРЕДСТВА**

## **2.1** Описание функциональности программного средства

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.

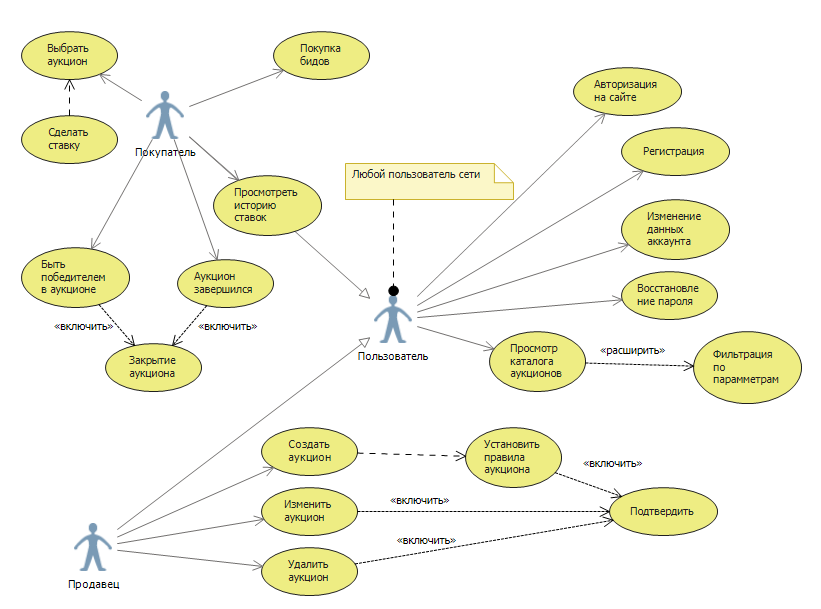


Рисунок 2.1 – UML Use-case диаграмма взаимодействия пользователя

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:

а) 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.

б) 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.

в) Управление аукционом. Если пользователь имеет роль аукционист, то он может создавать аукционы. Также аукционист может изменить или удалить созданный им аукцион.

г) Покупка бидов. Пользователь с ролью аукционер может покупать биды с помощью платёжной системы WebMoney. Бид используются в качестве ставки в аукционе.

д) Участие в аукционе. Пользователь с ролью аукционер может делать ставки при наличии бидов в аукционе.

е) Просмотр истории торгов. Пользователь может просматривать историю уже окончившихся аукционов.

ж) Просмотр каталога аукционов. Пользователь, может не заходя в систему просматривать доступные аукционы.

## **2.2** Спецификация функциональных требований.

В ходе разработки будут реализованы следующие возможности:

а) аутентификация. Пользователю будут доступны следующие функции:

1) регистрация;

2) авторизация;

3) анонимный доступ;

б) управление аукционом. Пользователю будут доступны следующие функции:

1) создание аукциона;

2) редактирование аукциона;

3) удаление аукциона;

в) участие в аукционе. Пользователю будут доступны следующие функции:

1) просмотр аукционов;

2) покупка бидов;

3) сделать ставку;

4) просмотр истории проведения аукциона;

г) навигация по сайту. Пользователю будут доступны следующие функции:

1) пагинация;

2) читабельная строка запроса;

3) фильтрация аукционов;

д) управление личным аккаунтом. Пользователю будут доступны следующие функции:

1) изменение данных аккаунта;

2) сброс пароля.

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.

# **3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## **3.1** Архитектура программного модуля

Архитектура программного обеспечения (software architecture) – это представление, которое даёт информацию о компонентах программного обеспечения, обязанностях отдельных компонентов и правилах организации связей между компонентами. Необходимость архитектуры обоснована сложностью программного обеспечения. Продуманная архитектура облегчает разработку и дальнейшее развитие программного обеспечения. 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 программного обеспечения. 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. |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Архитектурный стиль | Описание |
| 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:

а) Проектирование чётко устанавливает разграничение функций между уровнями.

б) Нижние уровни независимы от верхних уровней.

в) Верхние уровни вызывают функции нижних уровней, но при этом взаимодействуют только соседние уровни иерархии.

Использование многоуровневой архитектуры обеспечивает следующие преимущества:

а) Изоляция. Разработка и обновление программного обеспечения могут быть изолированы рамками одного уровня.

б) Производительность. Распределение уровней на отдельные физические компьютеры повышает производительность и отказоустойчивость.

в) 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):

а) Уровень представления (presentation layer) ответственен за взаимодействие с пользователем, ввод и вывод информации.

б) Бизнес-уровень или уровень бизнес-логики (business logic layer) обрабатывает информацию, реализуя конкретные бизнес-правила.

в) Уровень доступа к данным (data access layer) обеспечивает загрузку и сохранение информации, используя источник данных (файл, база данных) или внешний сервис.

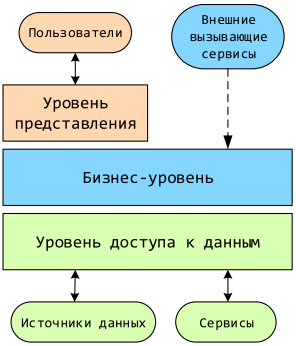


Рисунок 3.1 – Типичные уровни бизнес-приложения

## **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:

а) 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

б) Препроцессор Sass используется для компиляции css-кода ответственного за отображение элементов пользовательского интерфейса (кнопки, списки, текст и т.д.).

в) 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.

г) 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.

д) 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.

е) 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.

На рисунке 3.4 изображена схема алгоритма регистрации пользователя.

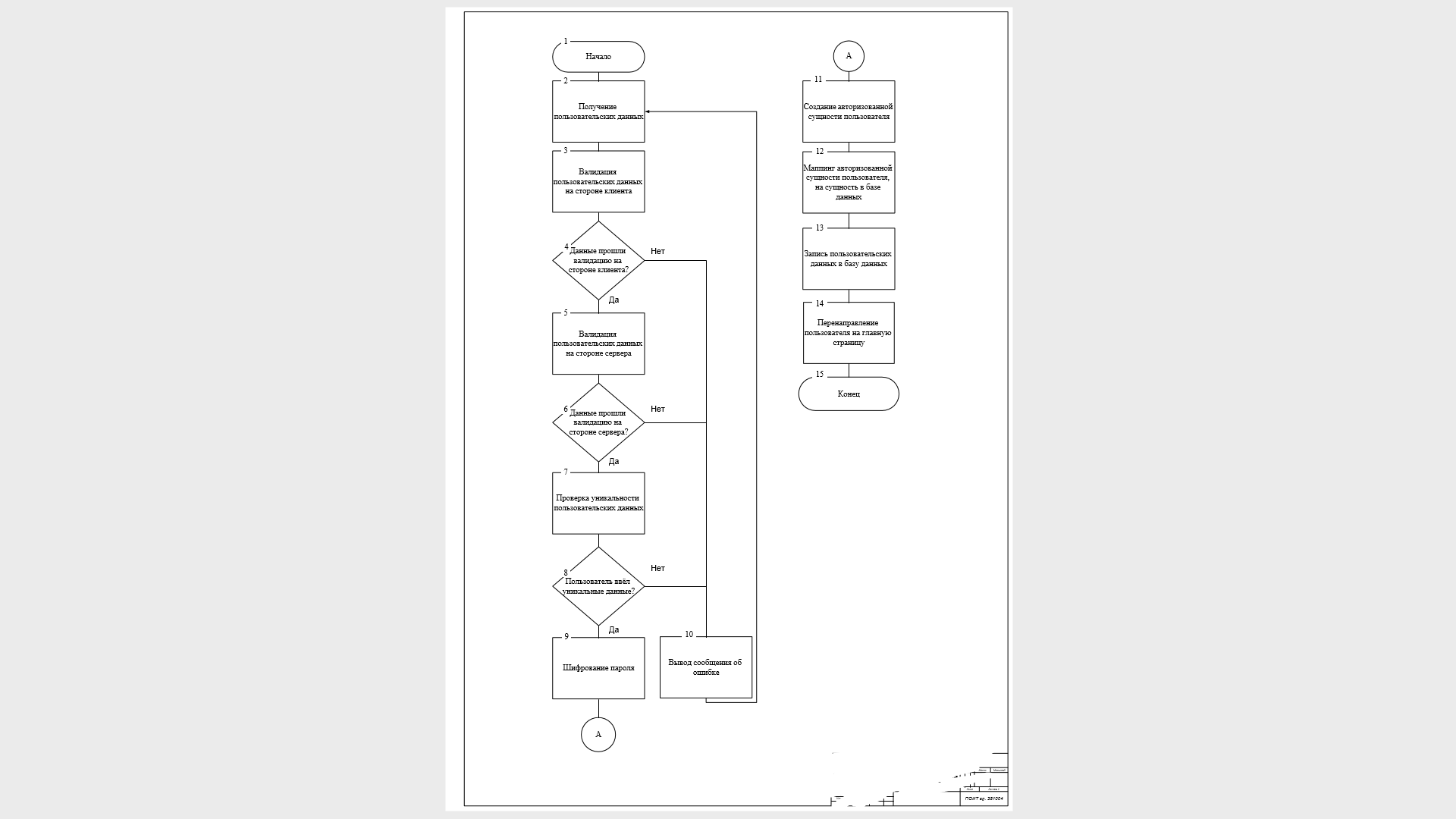


Рисунок 3.4 – Схема алгоритма регистрации пользователя

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 3.2 предоставлены требования к регистрационным полям.

Таблица 3.2 – Валидационные требования к регистрационным полям

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

На рисунке 3.5 изображена блок-схема алгоритма сохранения аукциона.



Рисунок 3.5 – 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:

а) удобство разработки, позволяющее быстро и качественно разрабатывать программные средства;

б) переносимость программного средства между различными персональными компьютерами;

в) доступная документация для обращения при возникших затруднениях во время разработки;

г) распространённость ОС, для которой разрабатывается программное средство.

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

а) операционная система Windows 10;

б) язык программирования C# 6.0;

в) платформа .Net Framework 4.5.2;

г) среда разработки Visual Studio 2015;

д) 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:

а) инкапсулированные сигнатуры методов, называемые делегатами, которые поддерживают типобезопасные уведомления о событиях;

б) свойства, выступающие в роли методов доступа для закрытых переменных-членов;

в) атрибуты с декларативными метаданными о типах во время выполнения;

г) встроенные комментарии XML-документации;

д) 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:

а) разделение ответственности (отдельная разработка разных компонентов - контроллера, моделей, представлений);

б) улучшенная тестируемость;

в) соответствие протоколу HTTP (ASP.NET MVC не поддерживает объект ViewState);

г) 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:

а) движок кросс-браузерных CSS-селекторов [Sizzle](https://ru.wikipedia.org/w/index.php?title=Sizzle&action=edit&redlink=1), выделившийся в отдельный проект;

б) переход по дереву DOM, включая поддержку XPath как плагина;

в) события;

г) визуальные эффекты;

д) AJAX-дополнения;

е) 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:

а) 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.

б) 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.

в) 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.

г) Скрытность. Autofac не вторгается в код. Можно легко изолировать зависимости в одной сборки в проекте.

**4.1.6** БЭМ

БЭМ (аббревиатура от слов — Блок, Элемент и Модификатор) — это методология разработки программ и интерфейсов, способ описания сущностей, не привязанный к конкретным технологиям реализации.

а) Блок — это отдельный компонент приложения. Он независим от других блоков и может содержать в себе другие блоки и элементы.

б) Элемент — это часть блока, отвечающая за отдельную функцию. Он не имеет смысла в отрыве от блока.

в) Модификатор — это свойство блока или элемента, отвечающее за его внешний вид или поведение. Модификаторы описывают состояние блока или элемента.

БЭМ предоставляет абстракцию над DOM-деревом. Блоки независимы друг от друга и инкапсулируют в себе всю функциональность и элементы. Не важно, какими HTML-тегами будет реализован блок - div или form, всегда можно изменить или добавить дополнительные обёртки. Любые изменения не должны оказывать влияние на остальные блоки. Приложение описывается компонентами интерфейса, а не HTML-тегами.

**4.1.7** 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:

а) Database first. Entity Framework создает набор классов, которые отражают модель конкретной базы данных.

б) Model first. Разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.

в) 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.

**4.1.8** 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).

**4.1.10** Среда разработки Visual Studio

Visual Studio – интегрированная среда разработки программного обеспечения 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:

а) редактор форм для разработки приложений GUI;

б) веб-редактор;

в) редактор классов;

г) 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:

а) C/C++;

б) VB.NET;

в) C#;

г) F#.

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

**4.2** 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:

а) слой представления – проект Application;

б) слой бизнес-логики – проект BLL;

в) 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:

а) 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.

б) 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.

в) Файл Web.config. Файл конфигурации приложения, который находится в корневой папке приложения.

г) 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.

д) Controllers. Содержит контроллеры - классы, отвечающие за работу приложения.

е) Scripts. В данной папке содержатся скрипты на JavaScript, используемые в приложении. По умолчанию эта папка уже содержат файлы библиотеки jQuery.

ж) Models. Содержит модели, используемые слоем представления.

з) Util. Содержит конфигурацию внедрения зависимостей.

и) Hubs. Здесь размещается класс используемый для организации двустороннего соединения.

к) 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.

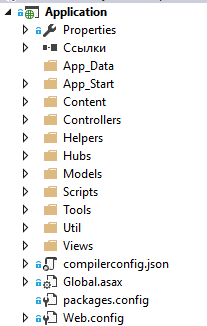


Рисунок 4.2 – Файловая структура проекта Application

## **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:

а) 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.

б) 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.

в) 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 4.3 изображено отношение между классом UnitOfWork, репозиторием и контекстом данных EntityFramework, при котором сервисы взаимодействуют с UnitOfWork , а не напрямую с EntityFramework.



Рисунок 4.3 – Организация взаимодействия при наличии репозитория

**4.3.3** 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 интерфейса, 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.

## **4.4** 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:

а) сохранение данных;

б) извлечение данных по различным критериям;

в) удаление данных;

г) модификация данных.

**4.4.1** 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);

}

**4.4.2** Класс UserService

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

Основные методы класса UserManagerService:

а) Authenticate(UserDto userDto) – асинхронно совершает поиск пользователя по данным авторизации;

б) Register(UserDto userDto) – асинхронно создает пользователя с указанным паролем;

в) IsEmailConfirmAsync(string id) – асинхронно подтверждает адрес электронной почты пользователя;

г) FindByNameAsync(string email) – асинхронно совершает поиск пользователя по имени;

д) ResetPassword(string email, string code, string password) – асинхронно производит сброс пароля;

е) FindByIdAsync(string id) – асинхронно совершает поиск пользователя по идентификатору;

ж) GetUsers() – извлекает список пользователей;

## **4.5** 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:

а) 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.

б) 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.

в) 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.

**4.5.1** Использование подхода Entity Framework Code First

Движущей силой подхода Code First является возможность применения классов POCO. Вместо использования для отображения базового сущностного класса в подходе Code First установлен набор соглашений для отображения классов POCO:

а) имена таблиц определяются с применением формы множественного числа имени сущности класса;

б) имена столбцов выводятся из имен свойств;

в) первичные ключи основываются на свойствах с именами ID и имяКлассаID.

г) 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 4.1).

Таблица 4.1 – 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) |

Продолжение таблицы 4.1

|  |  |
| --- | --- |
| Аннотация | Описание |
| 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 Auction

{

[Key]

public int Id { get; set; }

public string ImagePath { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public LotPhoto LotPhoto { get; set; }

public double StartPrice { get; set; }

public DateTime TradingStart { get; set; }

public string UserId { get; set; }

public virtual ApplicationUser User { 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 ApplicationContext : IdentityDbContext<ApplicationUser>

{

public ApplicationContext() : base("DefaultConnection")

{

Database.SetInitializer(new ApplicationInitializer());

}

public DbSet<Auction> Auctions { get; set; }

public DbSet<EndedAuction> EndedAuctions { get; set; }

}

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).

## **4.6** 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 аукцион, то можно выделить такие свойства, как название, время начала , начальная цена лота, путь к файлу с изображением лота. 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.

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

Содержит следующие поля:

1) идентификационный номер пользователя ;

2) логин - email-адрес ;

3) фамилию пользователя;

4) имя пользователя;

5) отчество пользователя;

6) пароль в зашифрованном виде;

7) количество доступных бидов.

б) Roles - сущность Roles является информацией о роли пользователя для разграничения доступа к элементам системы. и содержит идентификационный номер роли и её название.

Содержит следующие поля:

1) идентификационный номер роли;

2) название роли .

в) Auction – сущность Auction содержит информацию о выставляемом лоте и правила проведения которые задаёт аукционист.

Содержит следующие поля:

1) идентификационный номер аукциона;

2) название лота;

3) время создания аукциона;

4) время начала аукциона;

5) время окончания аукциона;

6) изображение лота;

7) начальная цена лота.

г) EndedAuction – сущность AuctionHistory содержит информацию о проведённых аукционах.

Содержит следующие поля:

1) идентификационный номер аукциона;

2) идентификационный номер победителя в аукционе;

3) идентификационный номер аукциониста выставляемого лота;

4) цена лота после завершения аукциона.

**4.7** Описание контроллеров

**4.7.1** Класс ProfileController

Класс ProfileController управляет представлениями, которые позволяют зарегистрированным пользователям управлять своим аккаунтом

Основные методы класса ProfileController:

а) UserManagement() – возвращает представление для управления пользователями;

б) SetToRole(string name) – задает роль пользователя;

в) ChangeUserAccess(string name) – изменяет права доступа пользователя.

**4.7.2** Класс HomeController

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

Основные методы класса HomeController:

а) Index() – возвращает представление страницы настройки профиля;

б) EditProfile(ProfileViewModel model) – сохраняет отредактированные данные;

в) ChangePassword() – возвращает представление с формой для изменения пароля.

**4.7.3** Класс AccountController

Класс AccountController управляет представлениями, которые позволяют пользователям аутентифицироваться, регистрироваться в системе.

Основные методы класса AccountController:

а) Login(string returnUrl) – возвращает представление аутентификации пользователя в системе;

б) Login(LoginViewModel model, string returnUrl) – аутентифицирует пользователя в системе;

в) Register() – возвращает представление регистрации пользователя в системе;

г) Register(RegisterViewModel model) – аутентифицирует пользователя в системе;

д) ConfirmEmail(string userId, string code) – подтверждает адрес электронной почты пользователя;

е) LogOff()– выход пользователя из системы.

**4.7.4** Класс BidsController

Класс BidsController дает возможность зарегестрированным пользователям купить биды для ставок.

Основные методы класса BidsController:

а) Index() – возвращает представление страницы покупки бидов;

б) BuyBids(string id) – производит покупку бидов через интернет кошелёк.

**4.7.5** Класс AuctionController

Класс AuctionController дает возможность пользователям управлять данными аукционов.

Основные методы класса AuctionController:

а) Index() – возвращает представление со списком аукционов;

б) CreateAuction() – создаёт аукцион;

в) DeleteAuction(int id) – удаляет аукцион;

г) Open(int id) – возвращает представление страницы подробного описания аукциона.

# **5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Тестирование программного обеспечения является важной частью жизненного цикла программных продуктов. Задачами современного тестирования является не только обнаружение ошибок в программах, но и выявление причин их возникновения.

Целью тестирования являлось подтверждение качества программного средства и соответствие заявленным требованиям.

Тестирование программного средства производилось по методу чёрного ящика на портативном персональном компьютере со следующими системными характеристиками:

а) операционная система: Windows 10 64 Bit;

б) видеокарта: NVIDIA GeForce 1070 8GB;

в) процессор: Intel Core i7-6700k 4.2GHz;

г) оперативная память (ОЗУ): 16GB.

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

Таблица 5.1 – Основные тестовые случаи для разрабатываемого программного средства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
| 11 | 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». |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
| 12 | Вход в систему | 1. Выбрать пункт «Войти» в навигационном меню.  2. Заполнить поля входа в соответствии с валидационными требованиям (см. таблица 3.5).  3. Подтвердить вход нажав кнопку «Войти» расположенной под формой | 1. Сайт должен открыться по адресу <имя сайта>/.  2. Вместо кнопок «Регистрация» и «Войти» должна появиться надпись «<электронная почта пользователя>» и кнопка «Выйти».  3. Должно появиться дополнительное меню | 1. Сайт открылся по адресу <имя сайта>/  2. В правом углу меню навигации появилась надпись «<электронная почта пользователя>» и кнопка «Выйти».  3. Появляется дополнительное меню |
| 23 | Выход из системы | 1. Выполнить вход в систему.  2. Нажать кнопку «Выйти» пункта навигационного меню. | 1. В правом углу меню навигации должна исчезнуть надпись «<электронная почта пользователя>» и кнопка «Выйти».  2. В правом углу меню навигации должны появиться кнопки «Регистрация» и «Войти».  3. Доступ к аккаунту, а также к изменению какого-либо контента должен быть закрыт.  4. Должно исчезнуть дополнительное меню | 1. В правом углу меню навигации исчезла надпись « <электронная почта пользователя>» и кнопка «Выйти».  2. В правом углу меню навигации появились кнопки «Регистрация» и «Войти».  3. Доступ к аккаунту, а также к изменению какого-либо контента.  4. Исчезает дополнительное меню |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Название | Описание | Ожидаемый результат | Полученный результат |
| 44 | Создание аукциона | 1. Выполнить вход в систему.  2. Выбрать подпункт «Создать аукцион» навигационного меню.  3. Заполнить поля.  4. Нажать кнопку добавить | 1. Сайт должен открыться по адресу <имя сайта>/  2. В списке аукционов должен появиться созданный аукцион. | 1. Сайт открылся по адресу <имя сайта>/  2. В списке аукционов появился созданный аукцион |
| 55 | Сделать ставку | 1. Перейти на страницу описания аукциона.  2. Если аукцион в активном состоянии нажать кнопку сделать ставку. | 1. Время таймера должно увеличится на 5 секунд.  2. Цена лота должна увеличится на 5.  3. В списке ставок на первом месте должно записаться имя пользователя. | 1. Время таймера увеличивается на 5 секунд.  2. Цена лота увеличивается на 5.  3. В списке ставок на первом месте записывается имя пользователя. |

# **6 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).

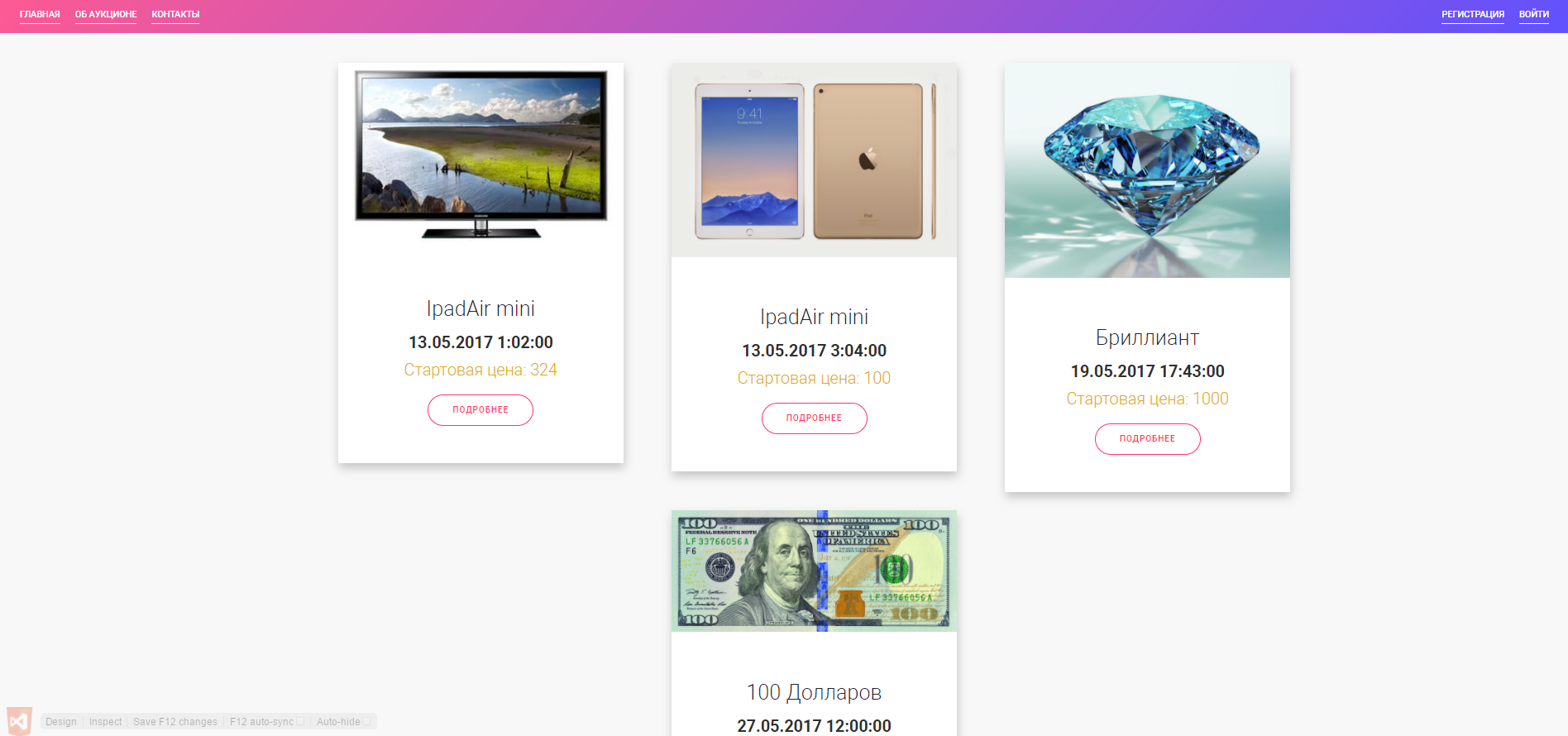


Рисунок 6.1 - Главная страница веб-сайта

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

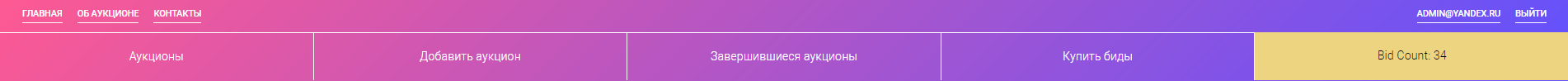


Рисунок 6.3 - Навигационное меню для авторизированного пользователя

## **6.2** 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.

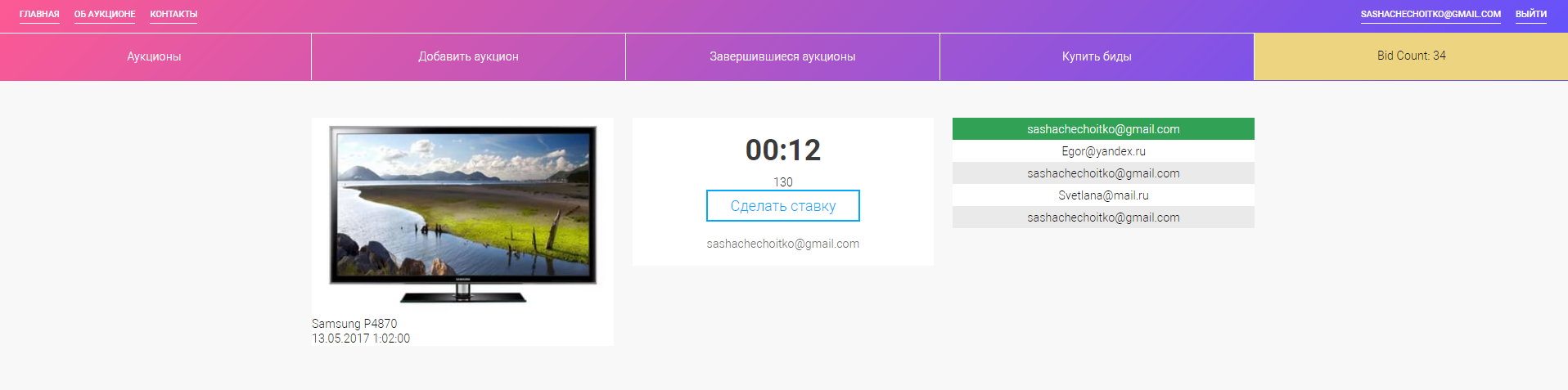


Рисунок 6.7 – Процесс аукциона

* 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.

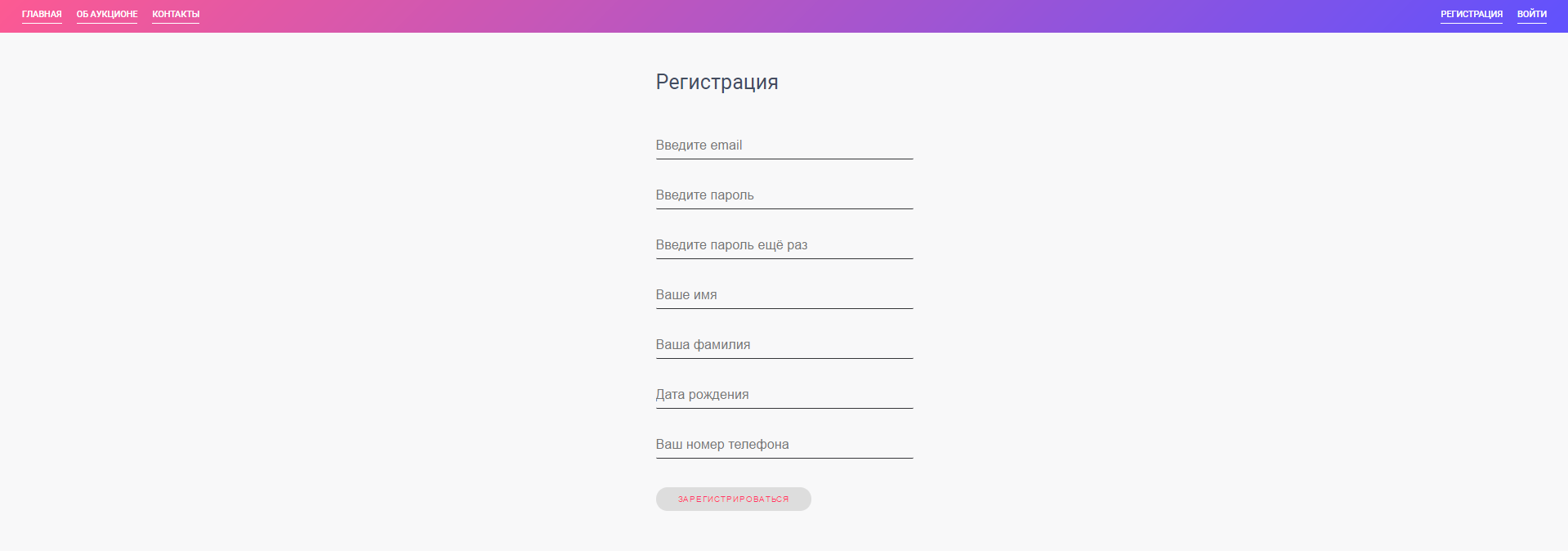


Рисунок 6.4 – Создание аукциона

# **7 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:

а) уменьшения бумажного документооборота;

б) автоматизации процесса проведения аукциона;

в) прибыль от проведённого аукциона;

г) 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 программных средств определяется плановая отпускаемая цена.

Базой для расчета плановой сметы затрат на разработку программного обеспечения является объем программного обеспечения. Общий объем программного продукта Vо, LOC, определяется по формуле 7.1:

 (7.1)

где Vi – объем отдельной функции программного обеспечения, LOC;

n – общее число функций.

На основании информации о функциях разрабатываемого программного обеспечения по каталогу функций определяется объем функций и общий объем программного обеспечения, который уточняется с учетом условий разработки программного обеспечения в организации. Рассчитывается уточненный объем программного обеспечения Vу, LOC, по формуле 7.2:

 (7.2)

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 программного обеспечения и нормативам затрат труда в расчете на единицу объема определяются нормативная и общая трудоемкость разработки программного обеспечения.

На основании принятого к расчету объема (Vу) и категории сложности определяется нормативная трудоемкость программного обеспечения (Тн) по формуле 7.3:

Tо = Tн⋅Kс⋅Kт⋅Kн , (7.3)

где Kc − коэффициент, учитывающий сложность программного обеспечения;

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

Kн − коэффициент, учитывающий степень новизны программного обеспечения.

Коэффициент сложности (Кс) учитывает дополнительные затраты труда, связанные со сложность разрабатываемого программного продукта и вычисляется по формуле 7.4:

 (7.4)

где Кi – коэффициент повышения сложности программного обеспечения;

n – количество учитываемых характеристик.

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

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

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

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

Общая трудоёмкость To равна:

To = 1.12 \* 0.5 \* 0.8 = 90 чел/дн.

Эффективный фонд времени работы одного работника (Фэф), определятся по формуле 7.5:

Фэф = Дг – Дп – Дв – До, (7.5)

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 7.6:

 (7.6)

где Тр  – срок разработки проекта, лет.

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

Чр = 90 / 0.3 / 231 = 1 чел.

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 7.7:

** (7.7)

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:

Зо = 6,1 \* 8 \* 88 \* 1,3 = 5582 тыс. руб.

Дополнительная заработная плата исполнителей, занятых разработкой программного продукта, (Зд) рассчитывается по формуле 7.8:

о, (7.8)

где Нд – норматив дополнительной заработной платы.

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

Зд = 5582 \* 20 / 100 = 1116 тыс. руб.

Отчисления в фонд социальной защиты населения (Зсз) определяются в соответствии с действующими законодательными актами по формуле 7.9:

Зсз = (Зо + Зд) \* нс з/ 100, (7.9)

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%):

Зсз = (5582 + 1116) \* 34 / 100 = 2277 тыс. руб.

Отчисления по обязательному страхованию определяются по нормативу 7.10:

Зoci = (Зоi + Зdi) \* нoi/ 100, (7.10)

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

Зoci =(5582 + 1116) \* 0,6 / 100 = 40,1 тыс.руб.

Расходы по статье «Материалы» определяются по формуле 7.11:

 (7.11)

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

Мi = 5582 \* 3 / 100 = 167 тыс. руб.

Расходы по статье «Машинное время» (Рм) включают оплату машинного времени, необходимого для разработки и отладки программного продукта определяются по формуле 7.12:

 (7.12)

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 = 88 \* 8 \* 5 \* 1 = 3520 тыс.руб.

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

Затраты по статье «Накладные расходы» (Рн) рассчитываются по формуле 7.13:

, (7.13)

где нрн – норматив накладных.

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

РН = 5582 \* 5 / 100 = 179,1 тыс. руб.

Общая сумма расходов по всем статьям сметы (Ср) на программный продукт рассчитывается по формуле 7.14:

 (7.14)

Рассчитаем сумму расходов по всем статьям сметы:

Ср = 5582 + 1116 + 2277 + 40,1 + 167 + 3520 + 0 + 279, 1 = 12981,2 тыс. руб.

Затраты на сопровождение и адаптацию программного продукта, которые определяются по нормативу (НРСА) (см. Формулу 7.15):

 (7.15)

где Нрас – норматив расходов на сопровождение и адаптацию.

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

Рcai = 12981,2 \* 10 / 100 = 1298,1 тыс. руб.

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость программного продукта (СП) определяется по формуле 7.16:

 (7.16)

Общая сумма расходов равна:

Cni = 12981,2 +1298,1 = 14279,3 тыс. руб.

Прибыль от программного продукта рассчитывается по формуле 7.17:

 (7.17)

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%):

Ппсi = 14279,3 \* 30 / 100 = 4283 тыс. руб.

Прогнозируемая отпускная цена без налогов рассчитывается по формуле 7.18:

 (7.18)

Цпi = 14279,3 + 4283 = 18563 тыс. руб.

Рассчитаем налог на добавочную стоимость (НДС) по формуле 7.19:

(7.19)

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

НДСi = 18563 \* 20 / 100 = 3712,6 тыс. руб.

Прогнозируемая отпускная цена рассчитывается по формуле 7.20:

(7.20)

Цо = 18563 + 3712,6 = 22275,6 тыс. руб.

## **7.5** Расчёт капитальных вложений заказчика

Общие капитальные вложения (Ко) заказчика (потребителя), связанные с приобретением, внедрением и использованием программного продукта, рассчитываются по формуле 7.21:

Ко = Кпр + Кос + Ктс + Коб (7.21)

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 7.22:

Кос = Зсм Кнз Чпо Тос + Цм Тмос (7.22)

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).

Кос = 3400 \* 0,5 \* 1 \* 0,25 + 5 \* 8 = 465 тыс. руб.

Общие капитальные вложения заказчика равны:

Ко = 22275,6 + 465 = 22740,6 тыс. руб.

## **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:

А2 = 3 \* 231 \* 4 = 2772 (задач)

Таблица 7.3 – 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) (см. формулу 7.23):

, (7.23)

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.

Сзе = 3400 \* (2 - 1) / (8 \* 22) = 19,3 тыс. руб.

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).

Сз = 19,3 \* 2772 = 53499 тыс. руб.

Экономия с учетом дополнительной заработной платы (Сн):

Сн = 53499 \* 1,3 = 69549 тыс. руб.

Экономия за счет сокращения простоев сервиса (Сс) рассчитывается по формуле 7.24:

(7.24)

где Дрг – количество рабочих дней в году;

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

СС = 231\* (20 - 10) \* 40 / 60 = 1540 тыс. руб.

Общая годовая экономия текущих затрат:

Со = Сн + Сс = 69549 + 1540 = 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 7.25:

, (7.25)

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 7.26:

 (7.26)

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:

а1 = (1 + 0,2)1-1 = 1 - расчетный год

а1 = (1 + 0,2)1-2 = 0,83 - 2018 год

а1 = (1 + 0,2)1-3 = 0,69 - 2019 год

а1 = (1 + 0,2)1-4 = 0,58 - 2020 год

Результаты расчета показателей эффективности приведены в таблице 7.4

Рассчитаем рентабельность инвестиций в приобретение программного продукта (РИ) по формуле 7.27:

 (7.27)

где ПЧСР - среднегодовая величина чистой прибыли за расчетный период, тыс. руб., которая определяется по формуле 7.28:

, (7.28)

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

Таблица 7.4 - 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 программного обеспечения (Kпp) | тыс. руб. |  |  |  |  |
| Освоение программного обеспечения (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:

а) Среднегодовая величина чистой прибыли:

Пчср = (0 + 58292 + 58292 + 58292) / 4 = 43719 тыс. руб.

б) Чистый дисконтированный доход за четыре года производства продукции составит 101264,4тыс. руб.

в) Все инвестиции окупаются на второй год использования программного продукта.

г) Рентабельность инвестиций в приобретение программного продукта:

Ри = 43719 / 22740,6 = 192,2%

## **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:

а) расширение системы для достижения более высоких показателей быстродействия и устойчивости к нагрузкам;

б) организация обратной связи между пользователем и администрацией с целью обмена дополнительной информацией и корректировки обнаруженных изъянов;

в) реализация системы оповещений пользователя о скором начале аукциона.

г) возможность использования других интернет-кошельков.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Р. Мартин. Чистый код. Создание, анализ и рефакторинг [Текст]: учебное пособие; [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)), 2012г., 256с.

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

[3] 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.

[4] Design Patterns in c# [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.dofactory.com/net/design-patterns

[5] Работа с сущностями POCO [Электронный ресурс]. – Электронные данные. – Режим доступа https://msdn.microsoft.com/ru-ru/library/dd456853(v=vs.100).aspx – 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] Основы CSS [Электронный ресурс] – Электронные данные. – Режим доступа: <http://www.w3schools.com/css/>

[9] Онлайн-книга Изучаем ASP.NET MVC 4

[Электронный ресурс]– Режим доступа: <http://www.metanit.com/sharp/mvc/>

[10] Статья о метапрограммировании [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/137446/

[11] Bootstrap свободный набор инструментов для создания сайтов и веб-приложений. [Электронный ресурс] – Электронные данные. – Режим доступа: <http://www.getbootstrap.com/>

[12] Yukihiro Matsumoto, Head First Design Patterns [Текст]: учебное пособие. -  М.:O’REILLY, 2014. - 696с.

[13] Using the CodeDOM [Электронный ресурс]. – Электронные данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/y2k85ax6(v=vs.110).aspx.

[14] Синтаксис движка Razor [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.w3schools.com/aspnet/razor\_syntax.asp

[15] ASP.NET | Файл web.config [Электронный ресурс]. –Электронные данные. – Режим доступа: http://professorweb.ru/my/ASP\_NET/base/level4/

[16] Нейгел, К. C# 5.0 и платформа .NET 4.5 для профессионалов[Текст] / К. Нейгел, Б. Ивьен, Д. Глинн, К. Уотсон, М. Скиннер. – М.: Диалектика, 2014.

[17] Объект ViewData [Электронный ресурс]. –Электронные данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/ss7fbaez%28v=vs.110%29.aspx

# **ПРИЛОЖЕНИЕ А. Исходный код программы**

public class AccountController : BaseController

{

private IUserService \_userService;

public AccountController(IUserService userSevice)

{

\_userService = userSevice;

}

[AllowAnonymous]

public ActionResult Login()

{

return View();

}

[AllowAnonymous]

[HttpPost]

public async Task<ActionResult> Login(LoginViewModel model)

{

if (ModelState.IsValid)

{

UserDTO userDto = new UserDTO { Email = model.Email, Password = model.Password };

ClaimsIdentity claim = await \_userService.Authenticate(userDto);

if (claim == null)

{

ModelState.AddModelError("", "Неверный логин или пароль.");

}

else if (await \_userService.IsLockout(model.Email))

{

return View("Lockout");

}

else

{

AuthenticationManager.SignOut();

AuthenticationManager.SignIn(new AuthenticationProperties

{

IsPersistent = true

}, claim);

return RedirectToAction("Index", "Home");

}

}

return View();

}

public ActionResult Logout()

{

AuthenticationManager.SignOut();

return RedirectToAction("Index", "Home");

}

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

[AllowAnonymous]

[HttpPost]

public async Task<ActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

UserDTO userDto = new UserDTO

{

Email = model.Email,

Password = model.Password,

FirstName = model.FirstName,

LastName = model.LastName,

DateOfBirth = model.DateOfBirth,

PhoneNumber = model.PhoneNumber,

Bids = 0,

Role = "user"

};

OperationDetails operationDetails = await \_userService.Register(userDto);

if (operationDetails.Succedeed)

return View("SuccessRegister");

else

ModelState.AddModelError(operationDetails.Property, operationDetails.Message);

}

return View(model);

}

[AllowAnonymous]

public ActionResult ForgotPassword()

{

return View();

}

[AllowAnonymous]

[HttpPost]

public async Task<ActionResult> ForgotPassword(ForgotPasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await \_userService.FindByNameAsync(model.Email);

if (user == null)

{

return View("ForgotPasswordConfirmation");

}

string code = await \_userService.GeneratePasswordResetTokenAsync(user.Id);

var callbackUrl = Url.Action("ResetPassword", "Account", new { code = code }, protocol: Request.Url.Scheme);

\_userService.SendEmail(user.Id, "Сброс пароля", "Для сброса пароля, перейдите по ссылке <a href=\"" + callbackUrl + "\">сбросить</a>");

return View("ForgotPasswordConfirmation");

}

return View(model);

}

[AllowAnonymous]

public ActionResult ResetPassword(string code)

{

ResetPasswordViewModel model = new ResetPasswordViewModel();

if (code != null)

{

model.Code = code;

return View("ResetPassword", model);

}

return View("error");

}

[AllowAnonymous]

[HttpPost]

public async Task<ActionResult> ResetPassword(ResetPasswordViewModel model)

{

if (model.NewPassword == model.ConfirmPassword)

{

OperationDetails details = await \_userService.ResetPassword(model.Email, model.Code, model.NewPassword);

if (details.Succedeed)

{

return RedirectToAction("Login");

}

ModelState.AddModelError("Password", details.Message);

}

return View(model);

}

}

public class AuctionController : BaseController

{

private List<string> \_allowedMimetypes = new List<string>

{

"image/jpeg",

"image/pjpeg",

"image/png",

"image/svg+xml"

};

private IUserService \_userService;

private IAuctionService \_auctionService;

public AuctionController(IUserService userSevice, IAuctionService auctionService)

{

\_userService = userSevice;

\_auctionService = auctionService;

}

public ActionResult Index()

{

return View();

}

[Authorize(Roles = "admin, user")]

public async Task<ActionResult> CreateAuction()

{

return View();

}

[HttpPost]

[Authorize(Roles = "admin, user")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> CreateAuction(AuctionViewModel auctionVm)

{

if (ModelState.IsValid)

{

if (auctionVm.LotInformation.UploadedImage != null && auctionVm.LotInformation.UploadedImage.ContentLength > 0)

{

if (\_allowedMimetypes.Contains(auctionVm.LotInformation.UploadedImage.ContentType))

{

string userId = AuthenticationManager.User.Claims.ElementAt(0).Value;

Auction auction = new Auction

{

LotPhoto = new LotPhoto

{

FileName = Path.GetFileName(auctionVm.LotInformation.UploadedImage.FileName),

ContentType = auctionVm.LotInformation.UploadedImage.ContentType

},

Title = auctionVm.LotInformation.Title,

StartPrice = auctionVm.LotInformation.StartPrice,

TradingStart = auctionVm.TradingStart,

UserId = userId,

};

using (var reader = new BinaryReader(auctionVm.LotInformation.UploadedImage.InputStream))

{

auction.LotPhoto.Content = reader.ReadBytes(auctionVm.LotInformation.UploadedImage.ContentLength);

}

await \_auctionService.CreateAuction(auction);

return RedirectToAction("Index", "Home");

}

else

{

ModelState.AddModelError("LotPhoto", "Некорректный формат изображения");

}

}

else

{

ModelState.AddModelError("LotPhoto", "Изображение не было получено");

}

}

return View(auctionVm);

}

[AllowAnonymous]

public async Task<ActionResult> GetImage(int id)

{

var auction = \_auctionService.GetAuctionById(id);

var fileToRetrieve = auction.LotPhoto;

return File(fileToRetrieve.Content, fileToRetrieve.ContentType);

}

[AllowAnonymous]

public async Task<ActionResult> Open(int id)

{

Auction auction = \_auctionService.GetAuctionById(id);

if (auction == null)

{

ModelState.AddModelError("", "Лот не найден");

return View("error");

}

string userName = AuthenticationManager.User.Identity.Name;

AuctionDescriptionViewModel model = AuctionDescriptionViewModel.BindModel(auction, userName);

return View("Auction", model);

}

}

public class ProfileController : BaseController

{

private IProfileService \_profileService;

private IUserService \_userService;

public ProfileController(IProfileService profileService, IUserService userService)

{

\_profileService = profileService;

\_userService = userService;

}

public async Task<ActionResult> Index()

{

var userDTO = await \_profileService.GetUserProfile(AuthenticationManager.User.Identity.Name);

ProfileViewModel model = new ProfileViewModel

{

FirstName = userDTO.FirstName,

LastName = userDTO.LastName,

DateOfBirth = userDTO.DateOfBirth,

PhoneNumber = userDTO.PhoneNumber

};

return View(model);

}

public PartialViewResult ChangePassword()

{

return PartialView("\_ChangePasswordPartial");

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ChangePassword(ChangePasswordViewModel model)

{

string oldPass = model.OldPassword;

string newPass = model.NewPassword;

string confirmPass = model.ConfirmPassword;

if (newPass == confirmPass)

{

OperationDetails details = await \_profileService.ChangePassword(AuthenticationManager.User.Identity.Name, oldPass, newPass);

if (details.Succedeed)

{

return RedirectToAction("Index");

}

}

return View(model);

}

public async Task<PartialViewResult> EditProfile()

{

var userDTO = await \_profileService.GetUserProfile(AuthenticationManager.User.Identity.Name);

ProfileViewModel model = new ProfileViewModel

{

FirstName = userDTO.FirstName,

LastName = userDTO.LastName,

DateOfBirth = userDTO.DateOfBirth,

PhoneNumber = userDTO.PhoneNumber

};

return PartialView("\_EditProfilePartial", model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> EditProfile(ProfileViewModel model)

{

UserDTO userDTO = new UserDTO

{

FirstName = model.FirstName,

LastName = model.LastName,

DateOfBirth=model.DateOfBirth,

PhoneNumber = model.PhoneNumber

};

OperationDetails result = await \_profileService.EditProfile(userDTO,

AuthenticationManager.User.Identity.Name);

if (result.Succedeed)

{

return View("Index", model);

}

return View("Error");

}

[Authorize(Roles = "admin")]

public PartialViewResult ManageUsers()

{

ManageUsersViewModel model = new ManageUsersViewModel();

model.Users = \_profileService.GetUserList();

return PartialView("\_ManageUsersPartial", model);

}

[Authorize(Roles = "admin")]

public async Task<ActionResult> LockoutUser(string email)

{

var result = await \_profileService.LockoutUser(email);

if (result.Succedeed)

{

return View("Index");

}

ModelState.AddModelError(result.Property, result.Message);

return View("Index");

}

[Authorize(Roles = "admin")]

public async Task<ActionResult> ResetUserPassword(string email)

{

if (ModelState.IsValid)

{

var result = await \_profileService.ResetUserPassword(email);

if (!result.Succedeed)

{

ModelState.AddModelError(result.Property, result.Message);

return RedirectToAction("Index");

}

var user = await \_userService.FindByNameAsync(email);

if (user == null)

{

return View("Index");

}

string code = await \_userService.GeneratePasswordResetTokenAsync(user.Id);

var callbackUrl = Url.Action("ResetPassword", "Account", new { code = code }, protocol: Request.Url.Scheme);

\_userService.SendEmail(user.Id, "Сброс пароля", "Для сброса пароля, перейдите по ссылке <a href=\"" + callbackUrl + "\">сбросить</a>");

}

return RedirectToAction("Index");

}

}

public class EmailService : IIdentityMessageService

{

public Task SendAsync(IdentityMessage message)

{

var from = "sasha.chechyotko@yandex.ru";

var pass = "4qz1001hp";

SmtpClient client = new SmtpClient("smtp.yandex.ru", 25);

client.DeliveryMethod = SmtpDeliveryMethod.Network;

client.UseDefaultCredentials = false;

client.Credentials = new System.Net.NetworkCredential(from, pass);

client.EnableSsl = true;

var mail = new MailMessage(from, message.Destination);

mail.Subject = message.Subject;

mail.Body = message.Body;

mail.IsBodyHtml = true;

return client.SendMailAsync(mail);

}

}

public class ProfileService : IProfileService

{

private const string ProfileSuccessEdit = "Регистрация успешно пройдена";

private const string SomethingWrong = "Что-то пошло не так";

private const string UserNotFound = "Пользователь не найден";

private const string SuccessPasswordChanged = "Пароль успешно изменён";

private const string ErrorResetPassword = "Не удалось сбросить пароль";

private const string SuccessResetPassword = "Пароль был сброшен";

private const string SuccessLockoutUser = "Пароль был сброшен";

private const string PasswordsDoNotMatch = "Пароли не совпадают";

private ApplicationRoleManager \_roleManager;

private ApplicationUserManager \_userManager;

public ProfileService(

ApplicationRoleManager roleManager,

ApplicationUserManager userManager)

{

\_roleManager = roleManager;

\_userManager = userManager;

}

public async Task<OperationDetails> EditProfile(UserDTO userDTO, string name)

{

ApplicationUser user = await \_userManager.FindByEmailAsync(name);

if (user != null)

{

user.FirstName = userDTO.FirstName;

user.LastName = userDTO.LastName;

user.DateOfBirth = userDTO.DateOfBirth;

user.PhoneNumber = userDTO.PhoneNumber;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

return new OperationDetails(true, ProfileSuccessEdit, "");

}

}

return new OperationDetails(false, SomethingWrong, "");

}

public async Task<UserDTO> GetUserProfile(string name)

{

ApplicationUser user = await \_userManager.FindByEmailAsync(name);

UserDTO userDTO = new UserDTO();

if (user != null)

{

userDTO.Email = user.Email;

userDTO.FirstName = user.FirstName;

userDTO.LastName = user.LastName;

userDTO.DateOfBirth = user.DateOfBirth;

userDTO.PhoneNumber = user.PhoneNumber;

}

return userDTO;

}

public IList<ApplicationUser> GetUserList()

{

var users = \_userManager.Users.ToList();

return users;

}

public async Task<OperationDetails> ChangePassword(string id, string oldPassword, string newPassword)

{

ApplicationUser user = await \_userManager.FindByNameAsync(id);

if (user == null)

{

return new OperationDetails(false, UserNotFound, "password");

}

var result = await \_userManager.RemovePasswordAsync(user.Id);

if (result.Succeeded)

{

result = await \_userManager.AddPasswordAsync(user.Id, newPassword);

if (result.Succeeded)

{

return new OperationDetails(true, SuccessPasswordChanged, "password");

}

}

return new OperationDetails(false, PasswordsDoNotMatch, "password");

}

public async Task<OperationDetails> LockoutUser(string email)

{

ApplicationUser user = await \_userManager.FindByNameAsync(email);

if (user != null)

{

user.LockoutEnabled = true;

user.LockoutEndDateUtc = DateTime.UtcNow.AddMinutes(42);

await \_userManager.UpdateAsync(user);

return new OperationDetails(true, SuccessLockoutUser, "");

}

return new OperationDetails(false, UserNotFound, "");

}

public async Task<OperationDetails> ResetUserPassword(string email)

{

ApplicationUser user = await \_userManager.FindByNameAsync(email);

if (user == null)

{

return new OperationDetails(false, UserNotFound, "password");

}

var result = await \_userManager.RemovePasswordAsync(user.Id);

if (!result.Succeeded)

{

return new OperationDetails(false, ErrorResetPassword, "");

}

return new OperationDetails(true, SuccessResetPassword, "");

}

}

public class UserService : IUserService

{

private const string RegisterSuccessMessage = "Регистрация успешно пройдена";

private const string UserWithThisLoginAlreadyExistMessage = "Пользователь с таким логином уже существует";

private const string UserNotFound = "Пользователь не найден";

private const string PasswordSuccessChanged = "Пароль успешно изменён";

private const string IncorrectCode = "Неправильный код";

private ApplicationRoleManager \_roleManager;

private ApplicationUserManager \_userManager;

public UserService(

ApplicationRoleManager roleManager,

ApplicationUserManager userManager)

{

\_roleManager = roleManager;

\_userManager = userManager;

}

public async Task<OperationDetails> Register(UserDTO userDTO)

{

ApplicationUser user = await \_userManager.FindByEmailAsync(userDTO.Email);

if (user == null)

{

user = new ApplicationUser {

Email = userDTO.Email,

UserName = userDTO.Email,

FirstName = userDTO.FirstName,

LastName = userDTO.LastName,

DateOfBirth = userDTO.DateOfBirth,

PhoneNumber = userDTO.PhoneNumber,

Bids = userDTO.Bids

};

var result = await \_userManager.CreateAsync(user, userDTO.Password);

if (!result.Succeeded)

{

return new OperationDetails(false, result.Errors.FirstOrDefault(), "");

}

await \_userManager.AddToRoleAsync(user.Id, userDTO.Role);

return new OperationDetails(true, RegisterSuccessMessage, "");

}

return new OperationDetails(false, UserWithThisLoginAlreadyExistMessage, "Email");

}

public async Task<ClaimsIdentity> Authenticate(UserDTO userDto)

{

ClaimsIdentity claim = null;

ApplicationUser user = await \_userManager.FindAsync(userDto.Email, userDto.Password);

if (user != null)

{

claim = await \_userManager.CreateIdentityAsync(user, DefaultAuthenticationTypes.ApplicationCookie);

}

return claim;

}

public void Dispose()

{

\_userManager.Dispose();

}

public void SendEmail(string id, string title, string message)

{

\_userManager.SendEmail(id, title, message);

}

public async Task<ApplicationUser> FindByNameAsync(string name)

{

return await \_userManager.FindByNameAsync(name);

}

public async Task<bool> IsEmailConfirmedAsync(string id)

{

return await \_userManager.IsEmailConfirmedAsync(id);

}

public async Task<string> GeneratePasswordResetTokenAsync(string id)

{

return await \_userManager.GeneratePasswordResetTokenAsync(id);

}

public async Task<OperationDetails> ResetPassword(string email, string code, string password)

{

ApplicationUser user = await \_userManager.FindByNameAsync(email);

if (user == null)

{

return new OperationDetails(false, UserNotFound, "password");

}

var result = await \_userManager.ResetPasswordAsync(user.Id, code, password);

if (result.Succeeded)

{

return new OperationDetails(true, PasswordSuccessChanged, "password");

}

return new OperationDetails(false, IncorrectCode, "password");

}

public async Task<bool> IsLockout(string name)

{

ApplicationUser user = await \_userManager.FindByNameAsync(name);

return await \_userManager.IsLockedOutAsync(user.Id);

}

public async Task<ApplicationUser> FindByIdAsync(string id)

{

return await \_userManager.FindByIdAsync(id);

}

}

public class ApplicationContext : IdentityDbContext<ApplicationUser>

{

public ApplicationContext() : base("DefaultConnection")

{

Database.SetInitializer(new ApplicationInitializer());

}

public DbSet<Post> Posts { get; set; }

public DbSet<Category> Categories { get; set; }

public DbSet<Comment> Comments { get; set; }

public DbSet<Tag> Tags { get; set; }

public DbSet<Auction> Auctions { get; set; }

public DbSet<EndedAuction> EndedAuctions { get; set; }

}

public static class AutofacConfig

{

private static IContainer \_container;

public static void ConfigureContainer()

{

var builder = new ContainerBuilder();

builder.RegisterControllers(typeof(MvcApplication).Assembly);

RegisterDbContext(builder);

RegisterRepositories(builder);

RegisterUnitOfWork(builder);

RegisterServices(builder);

RegisterManagers(builder);

\_container = builder.Build();

DependencyResolver.SetResolver(new AutofacDependencyResolver(\_container));

}

public static IUserService GetUserContext()

{

return \_container.Resolve<IUserService>();

}

public static IAuctionService GetAuctionContext()

{

return \_container.Resolve<IAuctionService>();

}

private static void RegisterDbContext(ContainerBuilder builder)

{

builder.RegisterType<ApplicationContext>().As<DbContext>().SingleInstance();

}

private static void RegisterRepositories(ContainerBuilder builder)

{

builder.RegisterGeneric(typeof(GenericRepository<>)).As(typeof(IGenericRepository<>)).SingleInstance();

}

private static void RegisterUnitOfWork(ContainerBuilder builder)

{

builder.RegisterType<UnitOfWork>().As<IUnitOfWork>().SingleInstance();

}

private static void RegisterServices(ContainerBuilder builder)

{

builder.RegisterType<UserService>().As<IUserService>().SingleInstance();

builder.RegisterType<ProfileService>().As<IProfileService>().SingleInstance();

builder.RegisterType<PostService>().As<IPostService>().SingleInstance();

builder.RegisterType<CommentService>().As<ICommentService>().SingleInstance();

builder.RegisterType<CategoryService>().As<ICategoryService>().SingleInstance();

builder.RegisterType<TagService>().As<ITagService>().SingleInstance();

builder.RegisterType<AuctionService>().As<IAuctionService>().SingleInstance();

builder.RegisterType<EndedAuctionService>().As<IEndedAuctionService>().SingleInstance();

}

private static void RegisterManagers(ContainerBuilder builder)

{

builder.RegisterType<UserStore<ApplicationUser>>().As<IUserStore<ApplicationUser>>().SingleInstance();

builder.RegisterType<RoleStore<ApplicationRole>>().AsSelf().SingleInstance();

builder.RegisterType<ApplicationUserManager>().AsSelf().SingleInstance();

builder.RegisterType<ApplicationRoleManager>().AsSelf().SingleInstance();

}

}

public class AuctionProcess

{

private IAuctionService \_auctionService;

private Dictionary<int, LiveAuction> \_liveAuctions = new Dictionary<int, LiveAuction>();

private Dictionary<int, WaitedAuction> \_waitedAuctions = new Dictionary<int, WaitedAuction>();

private object \_locker = new object();

public delegate void SendAuctionStateHandler(int auctionId, int ticks, double currentPrice, string lastUser);

public event SendAuctionStateHandler SendState;

public event SendAuctionStateHandler SetWinner;

public AuctionProcess(IAuctionService auctionService)

{

\_auctionService = auctionService;

}

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);

}

}

}

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);

}

}

}

public void PlaceBet(int auctionId, string userName)

{

}

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);

}

});

}

public void Start()

{

TimerCallback timerCallback = new TimerCallback(TimerHandler);

Timer timer = new Timer(timerCallback, null, 100, 1000);

}

}

public class UnitOfWork : IUnitOfWork

{

private readonly DbContext \_dbContext;

private IGenericRepository<Category> \_categoryRepository;

private IGenericRepository<Post> \_postRepository;

private IGenericRepository<Tag> \_tagRepository;

private IGenericRepository<Comment> \_commentRepository;

private IGenericRepository<Auction> \_auctionRepository;

public IGenericRepository<Auction> Auctions

{

get

{

if (\_auctionRepository == null)

\_auctionRepository = new GenericRepository<Auction>(\_dbContext);

return \_auctionRepository;

}

}

public IGenericRepository<Category> Categories

{

get

{

if (\_categoryRepository == null)

\_categoryRepository = new GenericRepository<Category>(\_dbContext);

return \_categoryRepository;

}

}

public IGenericRepository<Post> Posts

{

get

{

if (\_postRepository == null)

\_postRepository = new GenericRepository<Post>(\_dbContext);

return \_postRepository;

}

}

public IGenericRepository<Tag> Tags

{

get

{

if (\_tagRepository == null)

\_tagRepository = new GenericRepository<Tag>(\_dbContext);

return \_tagRepository;

}

}

public IGenericRepository<Comment> Comments

{

get

{

if (\_commentRepository == null)

\_commentRepository = new GenericRepository<Comment>(\_dbContext);

return \_commentRepository;

}

}

public UnitOfWork(DbContext dbContext)

{

\_dbContext = dbContext;

}

public async Task SaveAsync()

{

List<string> errors = new List<string>();

try

{

await \_dbContext.SaveChangesAsync();

}

catch (DbEntityValidationException ex)

{

foreach (DbEntityValidationResult validationError in ex.EntityValidationErrors)

{

errors.Add("Object: " + validationError.Entry.Entity.ToString());

foreach (DbValidationError err in validationError.ValidationErrors)

{

errors.Add(err.ErrorMessage + "");

}

}

}

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

private bool disposed = false;

public virtual void Dispose(bool disposing)

{

if (!this.disposed)

{

if (disposing)

{

}

this.disposed = true;

}

}

}

public class GenericRepository<T> : IGenericRepository<T>

where T : class

{

private readonly DbContext \_database;

public GenericRepository(DbContext dbContext)

{

\_database = dbContext;

}

public IQueryable<T> GetAll()

{

return \_database.Set<T>();

}

public T Get(int id)

{

return \_database.Set<T>().Find(id);

}

public void Create(T item)

{

var result = \_database.Set<T>().Add(item);

}

public void Update(T item)

{

\_database.Entry(item).State = EntityState.Modified;

}

public void Delete(int id)

{

var data = \_database.Set<T>();

T item = data.Find(id);

if (item != null)

data.Remove(item);

}

}

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title</title>

@Styles.Render("~/Content/css")

</head>

<body>

<div class="b-header">

<div class="b-super-header clearfix">

<div class="b-super-header\_\_nav clearfix">

<ul class="b-super-header\_\_list clearfix">

<li class="b-super-header\_\_item">

<a class="b-super-header\_\_link b-nav-button" href="@Url.Action("Index", "Home")">

<div class="b-nav-button\_\_text">Главная</div>

<div class="b-nav-button\_\_line"></div>

</a>

</li>

<li class="b-super-header\_\_item">

<a class="b-super-header\_\_link b-nav-button" href="@Url.Action("Index", "About")">

<div class="b-nav-button\_\_text">Об аукционе</div>

<div class="b-nav-button\_\_line"></div>

</a>

</li>

<li class="b-super-header\_\_item">

<a class="b-super-header\_\_link b-nav-button" href="@Url.Action("Contacts", "Home")">

<div class="b-nav-button\_\_text">Контакты</div>

<div class="b-nav-button\_\_line"></div>

</a>

</li>

</ul>

</div>

<div class="b-header-account">

@Html.Partial("\_LoginPartial")

</div>

</div>

@if (Request.IsAuthenticated)

{

<div class="b-user-panel">

<ul class="b-user-panel\_\_nav clearfix">

<li class="b-user-panel\_\_item">@Html.ActionLink("Аукционы", "Index", "Home", null, new { @class = "b-user-panel\_\_button" })</li>

<li class="b-user-panel\_\_item">@Html.ActionLink("Добавить аукцион", "CreateAuction", "Auction", null, new { @class = "b-user-panel\_\_button" })</li>

<li class="b-user-panel\_\_item">@Html.ActionLink("Завершившиеся аукционы", "EndedAuctions", "Auction", null, new { @class = "b-user-panel\_\_button" })</li>

<li class="b-user-panel\_\_item">@Html.ActionLink("Купить биды", "EndedAuctions", "Auction", null, new { @class = "b-user-panel\_\_button" })</li>

<li class="b-user-panel\_\_item"><span class="b-user-panel\_\_button">Bid Count: 34</span></li>

</ul>

</div>

}

</div>

<div class="content-container clearfix">

@RenderBody()

</div>

<script type="text/javascript" src="~/Scripts/jquery-3.1.1.min.js"></script>

@RenderSection("scripts", required: false)

</body>

</html>

@model Application.Models.AuctionModels.AuctionDescriptionViewModel

@{

ViewBag.Title = "Auction";

}

<div class="b-auction">

<div class="b-auction\_\_block b-auction-information">

<div class="b-auction-information\_\_image">

<img src="~/Auction/GetImage?id=@Model.Id" alt="">

</div>

<div class="b-auction-information\_\_title">@Model.Title</div>

<div class="b-auction-information\_\_start-time">@Model.TradingStart</div>

</div>

<div class="b-auction\_\_block b-auction-interface">

<div id="auction-timer" class="b-auction-interface\_\_timer">00:12</div>

<div id="auction-price" class="b-auction-interface\_\_price">130</div>

<div id="auction-button" class="b-auction-interface\_\_button">Сделать ставку</div>

<div id="auction-last-rate" class="b-auction-interface\_\_last-rate">@Model.UserName </div>

</div>

<div class="b-auction\_\_block b-user-rates">

<ul class="b-user-rates\_\_list">

<li class="b-user-rates\_\_item">@Model.UserName</li>

<li class="b-user-rates\_\_item">Egor@yandex.ru</li>

<li class="b-user-rates\_\_item">@Model.UserName</li>

<li class="b-user-rates\_\_item">Svetlana@mail.ru</li>

<li class="b-user-rates\_\_item">@Model.UserName</li>

</ul>

</div>

</div>

@section scripts{

<script src="~/Scripts/jquery.signalR-2.2.2.min.js"></script>

<script src="~/signalr/hubs"></script>

<script type="text/javascript">

$(function () {

var auction = $.connection.auctionHub;

auction.client.invokeMessage = function(auctionId, ticks, currentPrice, lastUser) {

if (@Model.Id === auctionId) {

ticks = parseInt(ticks, 10);

let minutes = Math.floor(ticks / 60);

let seconds = ticks % 60;

$('#auction-timer').text(`${minutes}:${seconds}`);

$('auction-price').text(currentPrice);

$('#auction-last-rate').text(lastUser);

}

}

$.connection.hub.start().done(function () {

$('#auction-button').click(function () {

auction.server.placeBet(@Model.Id, '@Model.UserName');

});

});

});

</script>

}

@model Application.Models.RegisterViewModel

@{

ViewBag.Title = "Register";

}

<div class="account-container">

<div class="account-title">Регистрация</div>

<div class="account-form">

@using (Html.BeginForm("Register", "Account", new { ReturnUrl = ViewBag.ReturnUrl }, FormMethod.Post, new { @class = "app-form" }))

{

@Html.AntiForgeryToken()

@Html.ValidationSummary(true, "", new { @class = "form-text-danger" })

<div class="form-group">

@Html.TextBoxFor(m => m.Email, new { @class = "form-input", placeholder = "Введите email" })

@Html.ValidationMessageFor(m => m.Email, "", new { @class = "form-text-danger" })

</div>

<div class="form-group">

@Html.PasswordFor(m => m.Password, new { @class = "form-input", placeholder = "Введите пароль" })

@Html.ValidationMessageFor(m => m.Password, "", new { @class = "form-text-danger" })

</div>

<div class="form-group">

@Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-input", placeholder = "Введите пароль ещё раз" })

</div>

<div class="form-group">

@Html.TextBoxFor(m => m.FirstName, new { @class = "form-input", placeholder = "Ваше имя" })

@Html.ValidationMessageFor(m => m.FirstName, "", new { @class = "form-text-danger" })

</div>

<div class="form-group">

@Html.TextBoxFor(m => m.LastName, new { @class = "form-input", placeholder = "Ваша фамилия" })

@Html.ValidationMessageFor(m => m.LastName, "", new { @class = "form-text-danger" })

</div>

<div class="form-group">

@Html.TextBoxFor(m => m.DateOfBirth, new { @class = "form-input", placeholder = "Дата рождения" })

@Html.ValidationMessageFor(m => m.DateOfBirth, "", new { @class = "form-text-danger" })

</div>

<div class="form-group">

@Html.TextBoxFor(m => m.PhoneNumber, new { @class = "form-input", placeholder = "Ваш номер телефона" })

@Html.ValidationMessageFor(m => m.PhoneNumber, "", new { @class = "form-text-danger" })

</div>

<div class="form-group">

<input type="submit" class="form-submit b-action-button" value="Зарегистрироваться" />

</div>

}

</div>

</div>

@model Application.Models.LoginViewModel

@{

ViewBag.Title = "Log in";

}

<div class="account-container">

<div class="account-title">Login</div>

<div class="account-form">

@using (Html.BeginForm("Login", "Account", new { ReturnUrl = ViewBag.ReturnUrl }, FormMethod.Post, new { @class = "app-form" }))

{

@Html.AntiForgeryToken()

@Html.ValidationSummary(true, "", new { @class = "form-text-danger" })

<div class="form-group">

@Html.TextBoxFor(m => m.Email, new { @class = "form-input", placeholder = "Введите email" })

@Html.ValidationMessageFor(m => m.Email, "", new { @class = "form-text-danger" })

</div>

<div class="form-group">

@Html.PasswordFor(m => m.Password, new { @class = "form-input", placeholder = "Введите пароль" })

@Html.ValidationMessageFor(m => m.Password, "", new { @class = "form-text-danger" })

</div>

<div class="form-group">

@Html.LabelFor(m => m.RememberMe)

@Html.CheckBoxFor(m => m.RememberMe)

</div>

<div class="form-group">

<input type="submit" class="form-submit b-action-button" value="Войти" />

</div>

<div class="form-group">

@Html.ActionLink("Забыли пароль?", "ForgotPassword", null, new { @class = "form-link" })

</div>

}

</div>

</div>

@model Application.Models.ForgotPasswordViewModel

@{

ViewBag.Title = "ForgotPassword";

}

<div class="account-container">

<div class="account-title">Восстановление пароля</div>

@using (Html.BeginForm("ForgotPassword", "Account", new { ReturnUrl = ViewBag.ReturnUrl }, FormMethod.Post, new { @class = "app-form" }))

{

@Html.AntiForgeryToken()

@Html.ValidationSummary(true, "", new { @class = "form-text-danger" })

<div class="form-group">

@Html.TextBoxFor(m => m.Email, new { @class = "form-input", placeholder = "Введите email" })

@Html.ValidationMessageFor(m => m.Email, "", new { @class = "form-text-danger" })

</div>

<div class="form-group">

<input type="submit" class="form-submit b-action-button" value="Отправить" />

</div>

}

</div>