Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

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

Пояснительная записка

к дипломному проекту

на тему

**Программное СРЕДСТВО ОСУЩЕСТВЛЕНИЯ торговых сделок по безналичному расчету**

БГУИР ДП 1-40 01 01 03 086 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | Г.А. Русецкий |
| Руководитель |  | И. М. Марина |
| Консультанты: |  |  |
| *от кафедры ПОИТ* |  | И. М. Марина |
| *по экономической части* |  | Т.Н. Слюсарь |
|  |  |  |
| Нормоконтролер |  | С. В. Болтак |
|  |  |  |
| Рецензент |  |  |

Минск 2016

СОДЕРЖАНИЕ

[Введение 8](#_Toc452526903)

[**1** Анализ аналогов, литературных источников и формирование требований к проектируемому программному средству 9](#_Toc452526904)

[**1.1** Анализ методов разработки архитектуры веб приложений. 9](#_Toc452526905)

[**1.2** Анализ существующих аналогов 15](#_Toc452526906)

[**1.2.1** Анализ сайта onliner.by (раздел «каталог и цены») 15](#_Toc452526907)

[**1.2.2** Анализ сайта onliner.by (раздел «барахолка») 16](#_Toc452526908)

[**1.2.3** Анализ сайта au.by 17](#_Toc452526909)

[**1.3** Назначение разработки 17](#_Toc452526910)

[**1.4** Перечень основных выполняемых функций 18](#_Toc452526911)

[**1.5** Входные данные 19](#_Toc452526912)

[**1.6** Выходные данные 19](#_Toc452526913)

[**1.7** Требования к временным характеристикам 19](#_Toc452526914)

[**1.8** Требования к информационной и программной совместимости 19](#_Toc452526915)

[**1.9** Требования к безопасности 19](#_Toc452526916)

[**1.10** Требования к интерфейсу 19](#_Toc452526917)

[**1.11** Обоснование выбора языка и сред разработки 20](#_Toc452526918)

[**2** Моделирование предметной области и разработка функциональных требований 24](#_Toc452526919)

[**2.1** Разработка функциональной модели 24](#_Toc452526920)

[**2.2** Разработка информационной модели 30](#_Toc452526921)

[**2.3** Разработка спецификации требований 41](#_Toc452526922)

[**3** Проектирование программного средства 42](#_Toc452526923)

[**3.1** Разработка архитектуры программного средства 42](#_Toc452526924)

[**3.2** Техническое проектирование программного средства 45](#_Toc452526925)

[**4** Тестирование программного средства 47](#_Toc452526926)

[**4.1** Функциональное тестирование программного средства 47](#_Toc452526927)

[**4.2** Модульное тестирование программного средства 50](#_Toc452526928)

[**5** Методика использования разработанного программного средства 52](#_Toc452526929)

[**6** Технико-экономическое обоснование разработки программного средства осуществления торговых сделок по безналичному расчету 54](#_Toc452526930)

[**6.1** Характеристика проекта 54](#_Toc452526931)

[**6.2** Расчет затрат на разработку и отпускной цены программного продукта ...……………………………………………………………………………54](#_Toc452526932)

[**6.3** Расчет стоимостной оценки результата от разработки программного продукта 58](#_Toc452526933)

[Заключение 63](#_Toc452526934)

[Список использованных источников 64](#_Toc452526935)

[Приложение А Код программного модуля products\_controller.rb 65](#_Toc452526936)

[Приложение Б Код программного модуля application\_controller.rb 68](#_Toc452526937)

[Приложение В Код программного модуля user.rb 71](#_Toc452526938)

[Приложение Г Код программного модуля 20160327154500\_tables.rb 79](#_Toc452526939)

[Приложение Д Диаграмма прецедентов 88](#_Toc452526940)

[Приложение Е Схема базы данных 89](#_Toc452526941)

[Приложение Ж Внешний вид главной страницы приложения 90](#_Toc452526942)

[Приложение З Схема программы 91](#_Toc452526943)

[Приложение И Схема добавление товара 92](#_Toc452526944)

[Приложение К Схема авторизации пользователя 93](#_Toc452526945)

# **ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ**

В настоящей пояснительной записке применяются следующие определения и сокращения.

*Программа* – данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

*Программирование* – научная и практическая деятельность по созданию программ.

*Программный модуль* – программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

*Программное обеспечение* – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

*Фреймворк* – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

*Спецификация программы* – формализованное представление требований, предъявляемых к программе, которые должны быть удовлетворены при ее разработке, а также описание задачи, условия и эффекта действия без указания способа его достижения.

ООП – Объектно-ориентированное программирование.

ОС – Операционная система.

ПО – Программное обеспечение.

ПС – Программное средство.

MVC (Model view controller) – Модель, представление, контроллер.

API (Application Programming Interface) – Интерфейс программирования приложений.

UML (Unified Modeling Language) — унифицированный язык моделирования.

# **ВВЕДЕНИЕ**

Появление в 90-х гг. XX века электронной коммерции стало возможным благодаря в первую очередь развитию на планете сети Internet. Это, в свою очередь, было причиной возникновения и развития целых отраслей, связанных с обработкой информации. К примеру, деньги в наше время стали в представлении многих людей ассоциироваться не только, да уже и не столько, как бумажки с портретами и памятниками архитектуры, а с виртуальными счетами в интернет и обычных банках, с всевозможными платежными системами, кредитными картами, картами оплаты, балансами и т. д. Именно благодаря развитию информационного пространства, вовлечению в него миллионов пользователей, организаций и структур стало возможным появление в начале в США и Европе, а затем и в остальном мире, интернет-магазинов.

Белорусский Интернет подтягивается за мировым. Проходит повальное увлечение создавать контентные сайты и представительские странички о себе. Утверждается понимание, что Интернет – это еще одна среда для ведения бизнеса, имеющая свою специфику, и весьма многоплановая, но подчиняющаяся общим бизнес-законам. В Интернете ищут партнеров или клиентов, создают решения поддержки собственного бизнеса, автоматизируют операции с партнерами и дилерами, создают и развивают розничные Интернет-магазины.

Интернет развивается довольно стремительно. Быстро растет количество издании, посвященных Сети, что предвещает широкое ее распространение даже в далеких от техники областях. Интернет превращается из большой игрушки для интеллектуалов в полноценный источник всевозможной информации для всех.

Ожидается существенный рост увеличения объемов Интернет-коммерции, особенно в таких областях, как путешествия, розничная торговля, финансы, тематическая реклама, а также в компьютерном секторе.

Преимущества электронной торговли для покупателя очевидны: отсутствие географической привязанности к торговой точке, он доступен 24 часа в сутки, 7 дней в неделю, 365 дней в году, позволяя, будучи в Москве, торговать, например, в Париже без открытия магазина на Елисейских Полях.

Именно поэтому темой своей дипломной работы я выбрал написание программного средства для осуществления торговых сделок через интернет.

# **1 АНАЛИЗ АНАЛОГОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

* 1. Анализ методов разработки архитектуры веб приложений.

Веб-приложение состоит из клиентской и серверной частей, тем самым реализуя технологию «клиент-сервер». Клиентская часть реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него. Серверная часть получает запрос от клиента, выполняет вычисления, после этого формирует веб-страницу и отправляет её клиенту по сети с использованием протокола HTTP. Само веб-приложение может выступать в качестве клиента других служб, например, базы данных или другого веб-приложения, расположенного на другом сервере. Ярким примером веб-приложения является система управления содержимым статей Википедии: множество её участников могут принимать участие в создании сетевой энциклопедии, используя для этого браузеры своих операционных систем (будь то Microsoft Windows, GNU/Linux или любая другая операционная система) и не загружая дополнительных исполняемых модулей для работы с базой данных статей. В настоящее время набирает популярность новый подход к разработке веб-приложений, называемый Ajax. При использовании Ajax страницы веб-приложения не перезагружаются целиком, а лишь догружают необходимые данные с сервера, что делает их более интерактивными и производительными.

Также в последнее время набирает большую популярность технология WebSocket, которая не требует постоянных запросов от клиента к серверу, а создает двунаправленное соединение, при котором сервер может отправлять данные клиенту, без запроса от последнего. Таким образом появляется возможность динамически управлять контентом в режиме   
реального времени.

Для создания веб-приложений на стороне сервера используются разнообразные технологии и любые языки программирования, способные осуществлять вывод в стандартную консоль.

За счет наличия исполняемой части, Web-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 Web-приложениями, относятся:

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

Другими словами, каждое web-приложение отправляет http запросы на web-сервер для получения полезных данных. Программа под управлением web-сервера использует ту или иную модель для хранения данных. В современном мире чаще всего используются базы данных, SQL или NoSQL.

Формально каждое web-приложение можно разбить на 3 взаимно независимые части.

Модуль, который исполняется WEB-браузером. Это приложение может быть написано на любом языке, который поддерживает браузер. Чаще всего используется язык JavaScript, как наиболее поддерживаемый и имеющий большую библиотечную поддержку. Это очень важно, так как позволяет существенно экономить бюджеты проектов.

Модуль, исполняемый на серверной стороне под управлением web-сервера. Это приложение может быть написано на любом языке, интерпретацию которого поддерживает выбранный Вами web-сервер. Последнее время, часто, в качестве языка программирования выбирается язык Java. Этот язык также имеет серьезную библиотечную поддержку.

База данных. В этой области так же существует достаточно широкий выбор. Есть промышленные базы данных, такие как Oracle, DB2, PostgreSQL. Есть легкие базы данных, такие как MySQL. База данных выбирается основываясь на целях и области решаемых задач.

Для достижения цели максимальной независимости между структурными единицами, необходимо чтобы каждая структурная единица оперировала только необходимым ей набором данных. Рассмотрим более подробно.

Браузер – это прикладное программное обеспечение для просмотра web страниц.

HTML – это стандартный язык разметки документов. Большинство современных web-браузеров способны интерпретировать язык HTML.

Web сервер – это программное обеспечение, которое способно принимать HTTP запросы от клиентов, обрабатывать их и отправлять ответ в соответствии со стандартом протокола.

База данных – это представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью ЭВМ.

Для минимизации зависимостей между «Браузером» и Web-сервером необходимо, чтобы язык разметки HTML был задействован только в браузере, а Web-сервер предоставлял интерфейс для получения необходимых данных для страницы.

Для решения этой задачи необходимо:

* определить цели и область решаемых задач, которые будут решаться в рамках создаваемого интерфейса;
* определить API серверной части;
* выбрать протокол взаимодействия между серверной и клиентской частью.

Взаимодействие базы данных и web-сервера возможно организовать на основании двух принципиально разных сценариях:

* бизнес логика находится в базе данных;
* бизнес логика находится в коде web-сервера;
* в первом случае база данных хранит данные и предоставляет интерфейс доступа к данным;
* выборка данных – решается через представления;
* модификация данных – решается через хранимые процедуры.

Программа для web-сервера является драйвером для доступа к бизнес-логике. Т.е она просто связывает Браузер с бизнес логикой, которая реализована в базе данных.

Во втором случае база данных хранит данные, и предоставляет прямой доступ к данным. Бизнес-логика реализована в коде web-сервера. В этом случае база данных предоставляет транзакции для проведения атомарных операций.

Для минимизации зависимостей между Web-Сервером и Базой данных, необходимо, чтобы бизнес-логика была определена только в одном месте. Т.е либо в коде Web-Сервера, либо в Базе данных. Это очень важно, так как ведет к уменьшению затрат на обработку запросов на изменения. Это происходит в силу того, что изменения в одной структурной единицы не выходят за ее рамки.

В последние годы была представлена технология Ajax, позволяющая выполнять перезагрузку частей страницы на основе данных из сервера, без перезагрузки всей страницы, с помощью языка JavaScript. Несмотря на то, что техника частичной перезагрузки страницы появилась задолго до того, как XMLHttpRequest был представлен в Internet Explorer (основа AJAX программирования), эта техника начала массово использоваться.

Каждый веб-разработчик сталкивается с некоторыми проблемами при навигации по страницам веб-приложения, например:

* востребованность в процессорном времени, для генерирования целых html-страниц;
* востребованность в хорошей пропускной способности канала, для передачи html-страниц;
* важность частого использования кэширования;
* проблемы с кнопками вперед/назад, при отправке форм.

К примеру, нередко можно встретить веб-приложения, в которых используется iframe, чтоб избежать проблем с кнопками вперед/назад.

Разработка на основе страниц иногда заставляет прибегать к немного странному стилю кодирования, повторениям кода (мы постоянно должны подключать одни и те же файлы), неэффективному использованию ресурсов (затраты на пропускной канал и вычислительную мощь), это все не присуще разработке настольных приложений.

В области веб-разработки можно выделить два вида решений: веб-приложения и веб-сайты.

В первом случае AJAX используется все более часто, так как данный тип приложений не требует некоторых свойств, которые присущи веб-сайтам. Для веб-сайтов интенсивное использование технологии AJAX часто становится проблемой.

В публичных веб-сайтах все еще используется концепция разработки на основе страниц, так как любой веб-сайт должен обладать следующими характеристиками:

* Закладки: каждая веб-страница имеет собственный URL, этот URL может быть сохранен в закладках. AJAX позволяет изменять части содержимого страниц, но URL страницы в этом случае остается одним и тем же, в следствие чего пользователь не может сохранить конкретный вид (состояние) страницы в закладки.
* Поисковая оптимизация (SEO): любой веб-сайт должен быть полностью проиндексирован поисковыми системами, такими, к примеру, как Google Search. Сегодняшние поисковики рассматривают веб, как веб 1.0, при этом JavaScript код полностью игнорируется, тем самым, любое частичное изменение страницы с помощью AJAX не считается новым посещением.
* Сервисы, основанные на посещении страниц: для таких сервисов, как рекламный сервис Google AdSense и сервис мониторинга посещений Google Analytics, очень важным является показатель количества посещений страниц. При использованием технологии AJAX частичная перезагрузка страницы не учитывается как новое посещение.

Обычно разные веб-страницы имеют разные URL, но следуя парадигме SPI мы должны изменять состояние лишь одной страницы, без ее перезагрузки, но вместе с этим мы должны изменять и URL, чтоб дать возможность пользователям добавить страницу в неком состоянии в закладки. Как мы можем это сделать?

Существует трюк, с помощью которого можно реализовать данную функцию, используя «ссылочную» часть URL («хэш-фрагмент»), это последняя часть ссылки, которая идет после символа «#». Обычно эта часть используется для прокрутки страницы в конкретную позицию, которая определена меткой. При изменении ссылочной части страница не перезагружается, а значит, если URL будет изменен с помощью window.location вместе с основным состоянием страницы, тогда страница перезагружена не будет. Так как URL и основное состояние будут изменены, пользователи смогут сохранить этот URL в закладки. Ссылочная часть URL будет содержать информацию о состоянии страницы, на основе этой информации могут быть запрошены данные из сервера, если пользователь захочет открыть страницу, которую он сохранил в нужном состоянии в закладках. К сожалению, ссылочная часть URL не может быть отправлена на сервер, так как ссылочная часть никак не взаимодействует с удаленным ресурсом по протоколу HTTP, следовательно, нам нужно запрашивать данные у сервера о состоянии страницы уже после загрузки самой страницы. После того, как будет загружена страница, мы можем проверить с помощью JavaScript, содержит ли объект window.location ссылочную часть, в которой должна храниться информация о состоянии. Если содержит, тогда мы можем запросить нужное состояние у сервера на основе ссылочной части URL. После выполнения дополнительного запроса к серверу мы получим страницу в нужном нам состоянии.

Самый простой способ сделать так, чтоб наш веб-сайт был обработан поисковыми системами – сделать две разных модели навигации: SPI – для пользователей; страницы – для поисковых ботов. Следующий пример демонстрирует ссылку, которая соответствует данной идее: <a href= «http://link» onclick=«return false»>…</a>. Эта ссылка ничего не делает в браузерах с поддержкой JavaScript, так как навигация отключена с помощью команды "return false" атрибута onclick, но когда поисковый бот будет индексировать страницы, он проигнорирует атрибут onclick, так как JavaScript код им не выполнится, в результате чего робот сможет пройти по ссылке, занести ее в индекс и проиндексировать следующую страницу. В SPI-приложениях URL, используемый для навигации между состояниями, должен содержать идентификатор состояния, либо в ссылочной части URL, либо URL может быть задан в виде обычной ссылки с параметрами. Второй вариант предпочтительней, так как позволяет избежать лишнего запроса к серверу, также можно использовать и «красивые» URL.

На сегодняшний день Google уже “умеет” сканировать «AJAX-ссылки», с помощью ссылочной части, которая следует за символами “#!”, что позволяет делать AJAX-приложения индексируемыми. В данном случае веб-сайт должен вернуть ожидаемую страницу, которая запрашивается в соответствии с \_escaped\_fragment\_ параметром.

В то же время SPI веб-фреймворк может добавлять специальный код обработчика onclick перед тем, как вызвать «return false», или может назначить ссылке event listener с помощью addEventListener или attachEvent, в зависимости от браузера, для навигации между состояниями. Этот event listener будет выполнять некие действия и отдавать команды серверу, обычно с помощью AJAX, для изменения состояния страницы. Когда пользователь кликнет по ссылке, состояние изменится, но переход на новую страницу не будет осуществлен, так как атрибут onclick=«... return false» предотвратит выполнение поведения по умолчанию.

Описанная выше техника – это простейший способ сделать ссылки SPI-приложения совместимыми с поисковыми ботами. Вы также можете разделить эти функции. Например, вы можете скрыть с помощью JavaScript некоторые ссылки для пользователей, которые будут видны только для ботов, и отдельно сделать кликабельные элементы, которые будут использовать пользователи для навигации между состояниями страницы.

Наиболее важной особенностью SPI-совместимого фреймворка является возможность генерирования целых HTML-страниц в нужном состоянии, и в то же время, должна быть возможность навигации между состояниями с помощью JavaScript с перезагрузкой лишь отдельных частей страницы.

Кнопки вперед/назад часто являются источником проблем для традиционных страничных веб-сайов, которые по возможности должны быть устранены. Следует избегать использования этих кнопок после отправки формы на сервер (так как это, к примеру, может привести к риску дважды купить один и тот же авиабилет или книгу на сайте). Тем не менее, следует учитывать, что использование кнопок вперед/назад широко распространено и привычно для пользователей.

Очевидно, что парадигма SPI ломает традиционный путь навигации по веб-сайту, и кнопки вперед/назад не имеют никакого смысла в SPI-приложениях (так как нет страниц), да и веб-браузеры не предоставляют должного контроля над этими кнопками.

Но это не совсем так. Работу кнопок вперед/назад можно эмулировать. Вместо навигации по страницам (по истории посещений страниц) может использоваться навигация между состояниями. В таком случае, с помощью JavaScript кода, можно определить изменения ссылочной части URL и запросить соответствующие изменения состояний приложения. Так как браузер теперь не посещает разные страницы, ваше приложение полностью отвечает за поведение кнопок вперед/назад, и можно решить типичные проблемы с некорректным использованием этих кнопок при отправке форм, так как теперь нет неконтролируемых посещений различных страниц веб-сайта.

На основе вышесказанного я решил разрабатывать приложение в форме Singlepage Application с использованием AJAX для динамической загрузки страниц с сервера, предоставляющего API.

* 1. Анализ существующих аналогов

В белорусском сегменте интернета уже существуют приложения позволяющие выполнять онлайн торговлю:

### **1.2.1** Анализ сайта onliner.by (раздел «каталог и цены»)

Внешний вид сайта представлен на рисунке 1.1.

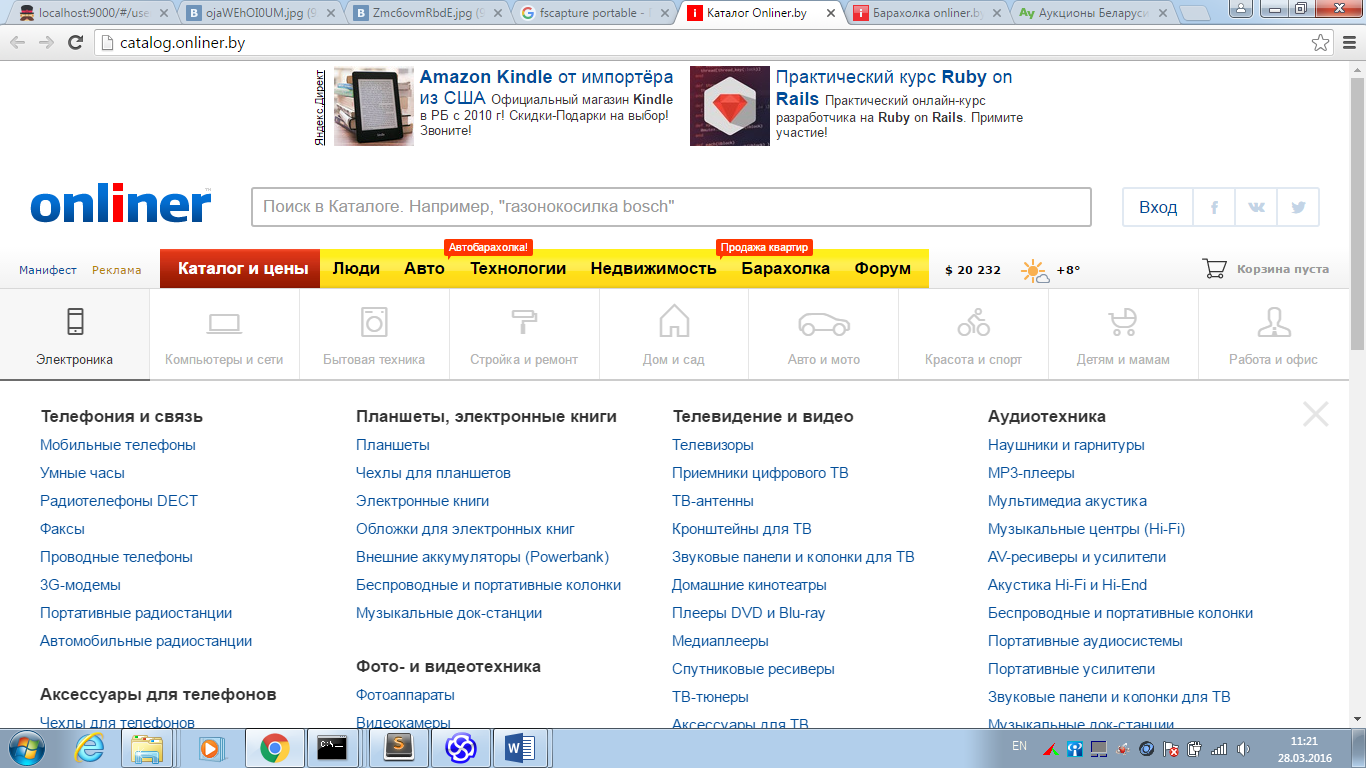


Рисунок 1.1 – Внешний вид сайта onliner.by

Достоинства:

* разделение товаров по категориям;
* большое количество опций сортировки и фильтрации товаров;
* наличие предложений о продаже одного и того же товара от разных продавцов.

Недостатки:

* отсутствие возможности прямой торговли – само приложение не осуществляет торговых операций, а содержит только контакты продавцов;
* возможность размещать объявления есть только у юридических лиц.

### **1.2.2** Анализ сайта onliner.by (раздел «барахолка»)

Внешний вид сайта представлен на рисунке 1.2.

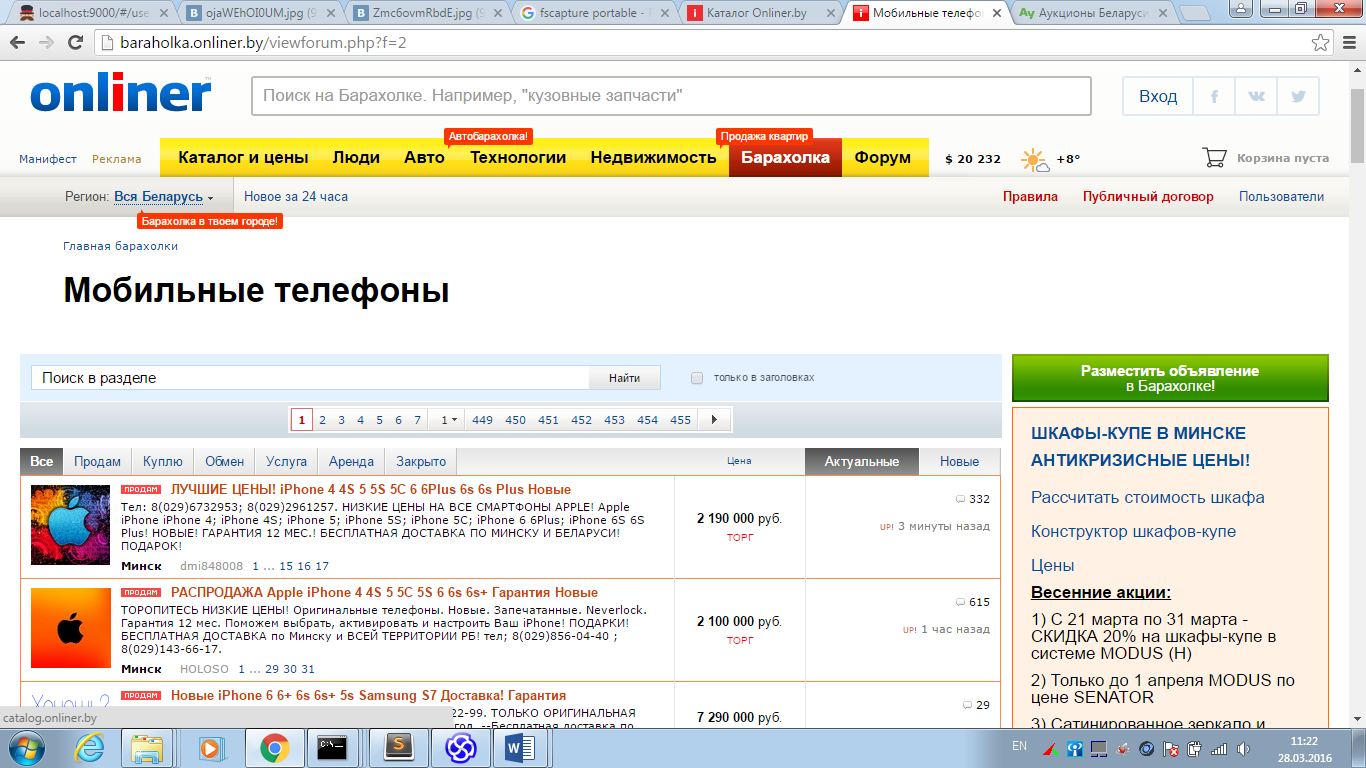


Рисунок 1.2 – Внешний вид сайта onliner.by

Достоинства:

* возможность размещения объявлений как покупателями так и продавцами;
* наличие полнотекстового поиска.

Недостатки:

* отсутствие фильтрации товаров по свойствам;
* отсутствие возможности прямой торговли.

### **1.2.3** Анализ сайта au.by

Внешний вид сайта представлен на рисунке 1.3.

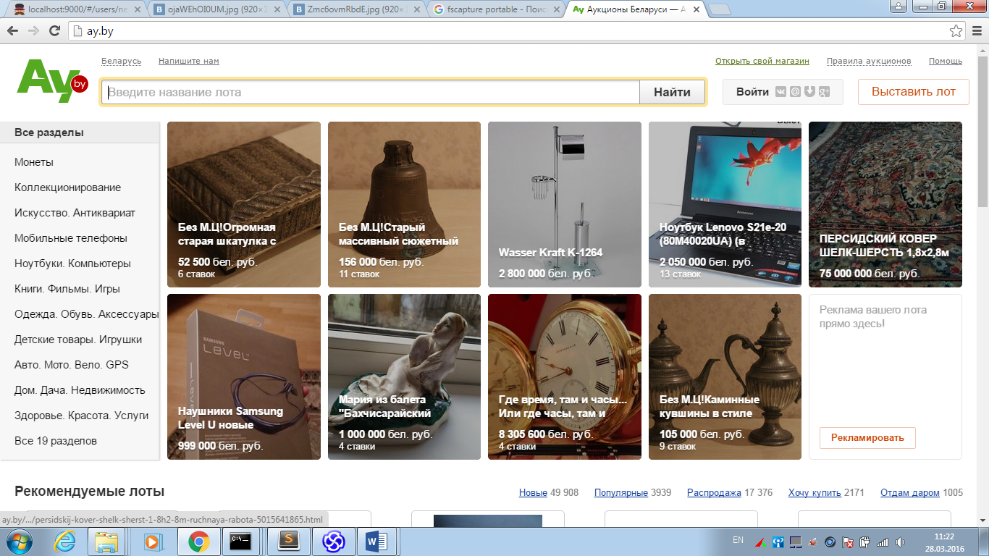


Рисунок 1.3 – Внешний вид сайта ay.by

Достоинства:

* возможность изменения цены (аукциона);
* наличие нескольких уровней категорий.

Недостатки:

* + отсутствие фильтрации товаров по свойствам;
  + отсутствие возможности прямой торговли;
  + отсутствуют расширенные варианты проведения аукционов.

Приняв во внимания достоинства и недостатки существующих аналогов, составим требования к разрабатываемому приложению.

* 1. Назначение разработки

Разработать программное средство, с помощью которого пользователи смогут осуществлять торговлю по безналичному расчету онлайн. Программное средство должно быть браузерным, поддерживать большинство современных браузеров и устройств, иметь приятный внешне и интуитивно понятный пользовательский интерфейс. С помощью программного средства пользователи будут размещать объявление о продаже каких-либо товаров или услуг, а другие пользователи будут покупать эти товары или услуги. Программное средство должно иметь функцию фильтрации товаров по их свойствам, а также задания этих свойств для новых товаров. Программное средство должно обеспечивать гибкость в настройке, должна быть предусмотрена возможность работы с самыми разными видами товаров. Приложение должно иметь систему прав пользователей. Программное средство должно хранить платежные реквизиты пользователей и осуществлять запрос к платежной системе при покупке товаров. Получать прибыль от приложения планируется через введение платы за пользование в размере доли от продаж.

* 1. Перечень основных выполняемых функций

Основные функции приложения:

* Регистрация и авторизация – основные функции системы будут доступны только для зарегистрированных пользователей.
* Создание категорий товаров – товары в каталоге будут делиться на категории.
* Создание свойств категорий – товары в одной категории будут описываться набором свойств, специфичных для этой категории. Свойствами могу быть, например, ширина, длина, вес, год выпуска и др. Программной средство должно позволять пользователю создавать и добавлять к категориям самые разные свойства.
* Создание товаров – товары в одной категории описываются сходным набором свойств. Каждый товар имеет свои значения каждого конкретного свойства.
* Добавление товаров в категории – товары классифицируются по категориям по назначению (или по другому признаку, который пользователь приложения сочтет важным).
* Создание отзывов на товары – пользователь будут иметь возможность оставлять отзывы в виде текстового комментария и числовой оценки от одного до пяти.
* Покупка товаров – покупка товаров в приложении должна осуществляться путем отправки в платежную систему счета, который покупатель в дальнейшем должен оплатить.
* Модерация – пользователь с правами модератора должен иметь возможность контролировать действия других пользователей.
* Отображение статистической информации – пользователь с правами директора должен иметь возможность получать статистическую информацию о работе приложения такую как: количество сделок, количество товаров, общая сумма сделок, прибыль, количество пользователей в приложении и тому подобное.
  1. Входные данные

Входные данные приложения:

* данные пользователей, указанные при регистрации;
* платежные реквизиты пользователей;
* данные указанные пользователями при создании товара;
* данные указанные пользователями при добавлении отзыва.
  1. Выходные данные

Выходные данные приложения:

* запросы к платежной системе;
* статистические данные о количестве пользователей, количестве сделок, количестве товаров, общей сумме сделок, прибыли;
* актуальная информация о товаре, о том если ли товар в наличии, о цене товара, о его характеристиках и тому подобное;
* данные из профилей пользователей.
  1. Требования к временным характеристикам

Серверный код, клиентский код, запросы к базе дынных, дизайн должны быть оптимизированы для лучшей производительности.

* 1. Требования к информационной и программной совместимости

Приложение должно работать через браузер пользователя, должно корректно выполнять свои функции и хорошо выглядеть в последних версиях популярных браузеров.

* 1. Требования к безопасности

Приложение должно обеспечивать защиту конфиденциальных данных пользователей, таких как пароли, платежные реквизиты, данные о совершенных сделках. Приложение должно противостоять попыткам взлома и неправомерного использования.

* 1. Требования к интерфейсу

Приложение должно иметь красивый и удобный интерфейс, работать на большинстве современных устройств.

* 1. Обоснование выбора языка и сред разработки

Для разработки серверной части я выбрал фреймворк Ruby on Rails.

Rails – фреймворк для веб-разработки, написанный на языке программирования Ruby. Он разработан, чтобы сделать программирование веб-приложений проще, так как использует ряд допущений о том, что нужно каждому разработчику для создания нового проекта. Он позволяет писать меньше кода в процессе программирования, в сравнении с другими языками и фреймворками так как реализует концепцию максимального повторного использования кода. Ruby on Rails предоставляет архитектурный образец Model-View-Controller (модель-представление-контроллер) для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером базы данных. Model-View-Controller – схема использования нескольких 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. Данная схема проектирования часто используется для построения архитектурного каркаса, когда переходят от теории к реализации в конкретной предметной области. Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счет такого разделения повышается возможность повторного использования. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. В частности, выполняются следующие задачи:

К одной модели можно присоединить несколько видов, при этом не затрагивая реализацию модели. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы.

Не затрагивая реализацию видов, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных), для этого достаточно использовать другой контроллер.

Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики (модели), вообще не будут осведомлены о том, какое представление будет использоваться.

Ruby on Rails использует REST-стиль построения веб-приложений.

Предпочтительным методом интеграции с веб-серверами является проксирование – использование веб-сервера в качестве прокси-сервера перед сервером приложения. Важную роль играют модули Phusion Passenger для интеграции с серверами Apache и nginx.

Ruby on Rails использует интерфейс RACK, что позволяет использовать менее распространённые механизмы (FCGI, CGI, SCGI). Ruby on Rails может работать с Apache, Lighttpd или любым другим веб-сервером, поддерживающим FastCGI. Для разработки и отладки часто используется веб-сервер WEBrick, встроенный в Ruby, или Mongrel. В качестве сервера базы данных поддерживаются MySQL, Firebird, PostgreSQL, DB2, Oracle и Microsoft SQL Server. Также поддерживается встраиваемая база данных SQLite.

Для разработки AJAX-приложений в RoR по умолчанию используется javascript-фреймворк jQuery, однако вместо него можно использовать и другие библиотеки.

В качестве базы данных я выбрал MySQL.

MySQL – свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

Для написания клиентского кода я использовал язык JavsScript, а также AngularJS. JavaScript-фреймворк с открытым исходным кодом. Предназначен для разработки одностраничных приложений. Его цель – расширение браузерных приложений на основе MVC шаблона, а также упрощение тестирования и разработки.

AngularJS спроектирован с убеждением, что декларативное программирование лучше всего подходит для построения пользовательских интерфейсов и описания программных компонентов, в то время как императивное программирование отлично подходит для описания бизнес-логики. Фреймворк адаптирует и расширяет традиционный HTML, чтобы обеспечить двустороннюю привязку данных для динамического контента, что позволяет автоматически синхронизировать модель и представление. В результате AngularJS уменьшает роль DOM-манипуляций и улучшает тестируемость. Angular придерживается MVC-шаблона проектирования и поощряет слабую связь между представлением, данными и логикой компонентов. Используя внедрение зависимости, Angular переносит на клиентскую сторону такие классические серверные службы, как видозависимые контроллеры. Следовательно, уменьшается нагрузка на сервер и веб-приложение становится легче.

Двустороннее связывание данных в AngularJS является наиболее примечательной особенностью и уменьшает количество кода, освобождая сервер от работы с шаблонами. Вместо этого, шаблоны отображаются как обычный HTML, наполненный данными, содержащимися в области видимости, определенной в модели. Сервис $scope в Angular следит за изменениями в модели и изменяет раздел HTML-выражения в представлении через контроллер. Кроме того, любые изменения в представлении отражаются в модели. Это позволяет обойти необходимость манипулирования DOM и облегчает инициализацию и прототипирование веб-приложений.

Также для клиентского кода использовалась библиотека JQuery.

jQuery – библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX. Точно так же, как 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.

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.

Для работы с коллекциями использовалась библиотека Underscore.

Underscore – библиотека JavaScript, реализующая дополнительную функциональность для работы с массивами, объектами и функциями, изначально отсутствующую в javascript, но имеющую аналоги в других языках. Библиотека умеет делегировать вызовы, если какая-то функциональность реализована разработчиками браузеров.

Для дизайна я использовал препроцессор SASS и верстку HTML.

# **2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## **2.1** Разработка функциональной модели

Для документирования требований к программному средству я использовал UML диаграммы.

UML (англ. Unified Modeling Language – унифицированный язык моделирования) ­– язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

Формальная спецификация последней версии UML 2.0 опубликована в августе 2005 года. Семантика языка была значительно уточнена и расширена для поддержки методологии Model Driven Development – MDD. Последняя версия UML 2.5 опубликована в июне 2015 года.

UML 2.4.1 принят в качестве международного стандарта ISO/IEC 19505-1, 19505-2.

Диаграмма прецедентов (диаграмма вариантов использования, use case диаграмма) в UML – диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Прецедент – возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

Преимущества UML:

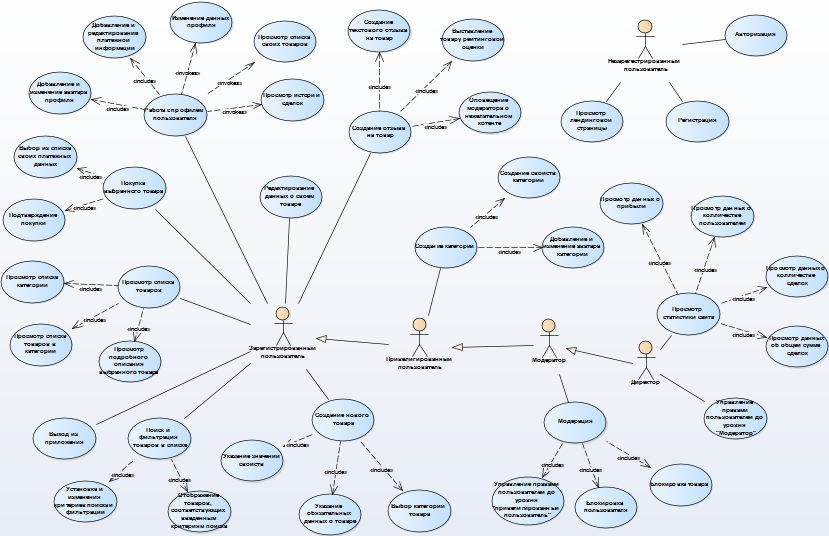
* UML объектно-ориентирован, в результате чего методы описания результатов анализа и проектирования семантически близки к методам программирования на современных объектно-ориентированных языках;
* UML позволяет описать систему практически со всех возможных точек зрения и разные аспекты поведения системы;
* диаграммы UML сравнительно просты для чтения после достаточно быстрого ознакомления с его синтаксисом;
* UML расширяет и позволяет вводить собственные текстовые и графические стереотипы, что способствует его применению не только в сфере программной инженерии;
* UML получил широкое распространение и динамично развивается;
* на основе требований, изложенные в разделе 1, я составил use case диаграмму (рисунок 2.1).

Рисунок 2.1 – Полная диаграмма прецедентов приложения

В приложении есть 5 ролей:

1. Незарегистрированный пользователь – пользователь который не приходил регистрацию на сайте, ему основные функции системы ему не доступны. Он может только открыть лендинговую страницу приложения, на которой будет информация о приложении и также зайти на страницы регистрации и авторизации (рисунок 2.2).

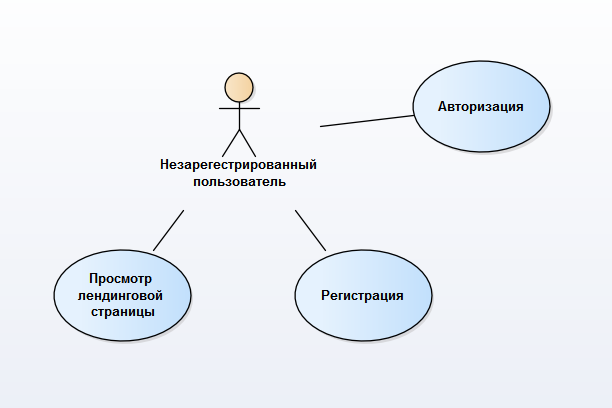


Рисунок 2.2 – Диаграмма прецедентов для незарегистрированного пользователя

1. Зарегистрированный пользователь – пользователь который указал свои данные при регистрации и может использовать основные функции приложения (рисунок 2.3).

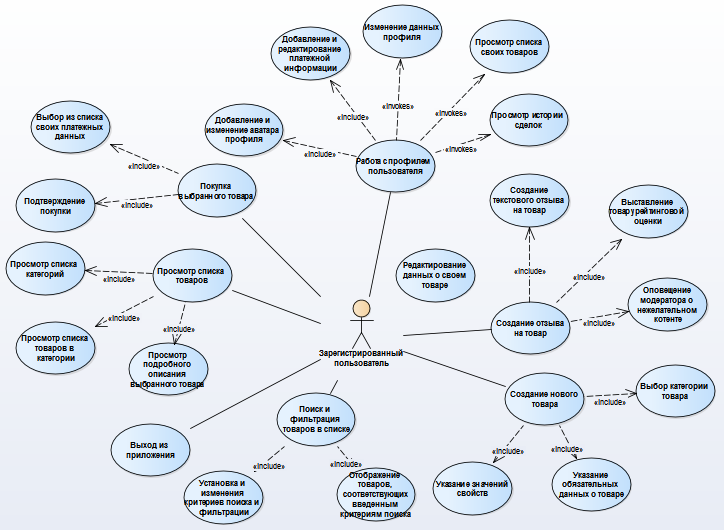


Рисунок 2.3 – Диаграмма прецедентов для зарегистрированного пользователя

Зарегистрированный пользователь имеет доступ к основным функциям приложения, среди которых:

* просмотр списка товаров – товары отображаются в списке, который можно формировать и упорядочивать в зависимости от введенных параметров;
* просмотр списка категорий – товары подразделяются на категории;
* просмотр подробного описания товара – у каждого товара есть подробное описание в котором указаны значения специфичных для товара свойств, таких как вер, длина, ширина и так далее;
* покупка товара – при покупке система отправляет запрос к платежной системе для снятия денег со счета покупателя и переведения на счет продавца;
* выбор из списка платежных данных при покупке – пользователь может иметь несколько платежных средств и про покупке выбирать одно из них;
* подтверждение покупки – платежная система обычно запрашивает у пользователя подтверждение совершаемой операции;
* добавление и изменения данных профиля – пользователь должен иметь возможность изменять данные указанные при регистрации;
* добавление и редактирование платежной информации – пользователь может добавить в приложение несколько платежных средств, с помощью которых совершать покупки;
* добавление или изменения аватара – у каждого пользователя будет картинка аватар;
* просмотр списка своих товаров – пользователю будет доступен список товаров, которые в данный момент он продает;
* просмотр истории сделок – пользователь будет иметь возможность видеть список своих прошлых покупок и продаж;
* поиск и фильтрация товаров в списке – пользователь сможет выбирать из списка товаров интересующие его товары путем указания категории, указания значений свойств товаров;
* создание нового товара – включает в себя указание категории, указание обязательных данные, указание значений свойств категории;
* редактирование данных о своем товаре – пользователь может менять данные о товарах которые он сам добавил;
* создание отзыва о товаре – отзывы обеспечивают коммуникацию пользователей и администрации сайта. Отзыв состоит из числовой оценки, текстового комментария и опционального поля «сообщить модератору о нежелательном контенте»;
* выход из приложения – пользователь имеет возможность нажать кнопку «выйти» после нажатия которой он снова станет незарегистрированным пользователем.

1. Привилегированный пользователь – зарегистрированный пользователь, которому модератор добавил права на создание новых категорий (рисунок 2.4).

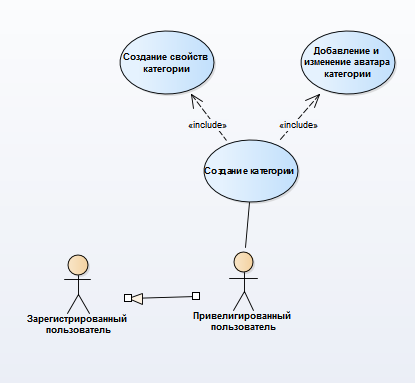


Рисунок 2.4 – Диаграмма прецедентов для привилегированного пользователя

Помимо функций доступных зарегистрированному пользователю, привилегированный пользователь имеет возможность созданий новых категорий. Дополнительные функции привилегированного пользователя включают в себя:

* Создание категории, которое включает в себя добавление аватара категории и создание свойств категории. Каждая категория характеризуется специфичными свойствами (например свойства автомобиля – марка, модель, год выпуски, пробег, цвет, состояние, свойства стола – производитель, материал, размеры, цвет, количество полок и других элементов и т.д.). Создание отдельных свойств для каждой категории обеспечивает гибкость в работе, позволяет системе работать с самыми разными товарами.

1. Модератор – пользователь, который контролирует действия других пользователей (рисунок 2.5).

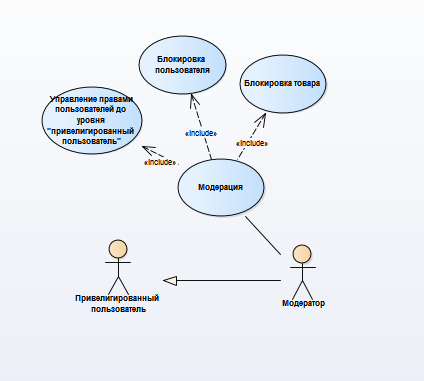


Рисунок 2.5 – Диаграмма прецедентов модератора

В дополнение к функциям привилегированного пользователя модератор имеет функции:

* блокировка пользователей – заблокированный пользователь теряет возможность совершать торговые операции и оставлять отзывы;
* блокировка товаров – заблокированный товар не может быть куплен;
* добавление или удаление у пользователя прав привилегированного пользователя.

При помощи этих инструментов модератор может осуществлять контроль за нежелательным контентом, оскорбительным поведением некоторых пользователей и т.п.

1. Директор – владелец приложения. Имеет максимальные права (рисунок 2.6).

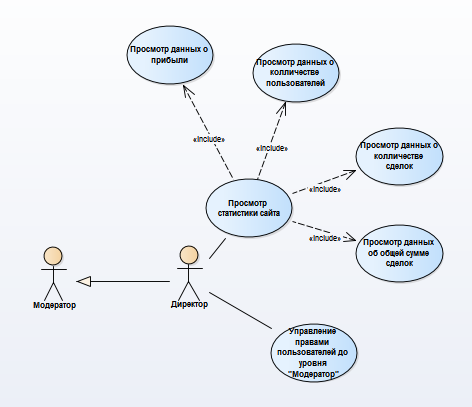


Рисунок 2.6 – Диаграмма прецедентов для директора

Директор наследует права модератора, и в дополнение к ним имеет ещё некоторые.

* просмотр статистики сайта – в целях управления бизнесом директор должен иметь информацию о работе сайта, такую как количество пользователей, количество сделок, общая сумма всех сделок, сумма прибыли;
* добавление и удаление прав модератора у пользователей – директор может назначать и снимать модераторов.

## **2.2** Разработка информационной модели

Далее, принимая во внимание целевые функции приложения, разработаем схему базы данных приложения.

Реляционная модель данных в подавляющем большинстве случаев вполне достаточна для моделирования любых данных. Однако проектирование базы данных в терминах схемы отношений на практике может вызвать большие затруднения, т.к. в этой модели изначально не предусмотрены механизмы описания семантики предметной области. С этим связано появление семантических моделей данных, которые позволяют описать конкретную предметную область гораздо ближе к интуитивному пониманию и, в то же время, достаточно формальным образом.

Часто семантическое моделирование используется только на первой стадии проектирования базы данных. Концептуальная схема будущей БД строится на основе некоторой семантической модели, а затем вручную преобразуется к реляционной схеме.

Существуют методики, четко описывающие все этапы такого преобразования. При таком подходе отсутствует потребность в дополнительных программных средствах, поддерживающих семантическое моделирование. Требуется только владение основами выбранной семантической модели и правилами преобразования концептуальной схемы в реляционную.

Следует заметить, что многие начинающие проектировщики баз данных недооценивают важность семантического моделирования вручную. Зачастую это воспринимается как дополнительная и излишняя работа. Эта точка зрения является абсолютно неверной. Во-первых, построение мощной и наглядной концептуальной схемы БД позволяет более полно оценить специфику моделируемой предметной области и избежать возможных ошибок на стадии проектирования схемы реляционной БД. Во-вторых, на этапе семантического моделирования производится очень важная документация (хотя бы в виде вручную нарисованных диаграмм и комментариев к ним), которая может оказаться полезной не только при проектировании схемы реляционной БД, но и при эксплуатации, сопровождении и развитии уже заполненной БД.

История систем автоматизации проектирования баз данных (CASE-средств) начиналась с автоматизации процесса рисования диаграмм, проверки их формальной корректности, обеспечения средств долговременного хранения диаграмм и другой проектной документации. Конечно, компьютерная поддержка работы с диаграммами очень полезна для проектировщика БД. Наличие электронного архива проектной документации помогает при эксплуатации, администрировании и сопровождении базы данных. Но система, которая ограничивается поддержкой рисования диаграмм, проверкой их корректности и хранением, напоминает текстовый редактор, поддерживающий ввод, редактирование и проверку синтаксической корректности конструкций некоторого языка программирования, но существующий отдельно от компилятора. Кажется естественным желание расширить такой редактор функциями компилятора, и это действительно возможно, поскольку известна техника компиляции конструкций языка программирования в коды целевого компьютера. Но коль скоро имеется четкая методика преобразования концептуальной схемы БД в реляционную схему, то почему бы ни выполнить программную реализацию соответствующего «компилятора» и ни включить ее в состав системы проектирования баз данных?

Существует много разных подходов к семантическому моделированию баз данных. В последние 10 лет одним из наиболее популярных языков семантического моделирования является UML. Проектирование реляционных БД – только одна и не слишком большая область применения этого языка, его возможности гораздо шире, однако подмножество UML (диаграммы классов) успешно применяется именно для таких целей.

Среда «Enterprise Architect» поддерживает генерацию UML диаграмм и автоматическую генерацию кода на SQL, поэтому для разработки схемы я выбрал именно её.

Разработку схемы начнем с таблицы users, которая будет хранить данные о пользователях. Во всех таблицах присутствуют поля created\_at, updated\_at и id. Поля created\_at и updated\_at хранят даты создания и последнего редактирования записи, они автоматически управляются ORM в Ruby on Rails. Поле id содержит номер записи в таблице и используется во всех таблицах в качестве первичного ключа. Индексы на таблицы установлены на поля по которым наиболее вероятно будет производится поиск. Далее я опишу значение каждого поля всех таблиц. Начнем с таблицы users.

* email – адрес электронной почты пользователя;
* login – логин пользователя на сайте;
* first\_name – имя пользователя;
* last\_name – фамилия пользователя;
* passport\_id – номер паспорта пользователя;
* avatar\_url – ссылка на аватар пользователя;
* address\_line\_1 – первая строка адреса;
* address\_line\_2 – вторая строка адреса;
* post\_index – почтовый индекс;
* phone\_number – номер телефона;
* birth\_date – дата рождения;
* is\_blocked – флаг блокировки;
* is\_archived – флаг архивирования;
* role\_id – id роли пользователя, внешний ключ;
* country\_id – id страны пользователя, внешний ключ;
* token – токен пользователя используемый для аутентификации;
* encrypted\_password – хеш пароля пользователя, используемый для авторизации.

Таблица также содержит индексы на поля role\_id, country\_id, token, login, email (рисунок 2.7).

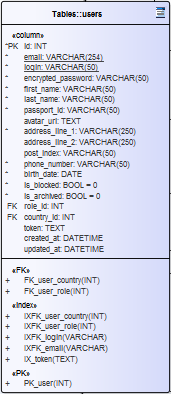


Рисунок 2.7 – UML диаграмма для таблицы users

Таблица countries содержит данные о странах (рисунок 2.8).

* name – название страны;
* phone\_code – телефонный код страны.

Таблица содержит индексы на поля name и phone\_code.

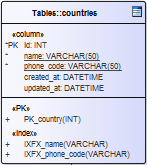


Рисунок 2.8 – UML диаграмма для таблицы countries

Таблица roles содержит данные о ролях пользователей и правах для каждой роли (рисунок 2.9).

* name – название роли;
* can\_see\_products – права на просмотр товаров;
* can\_create\_products – права на создание товаров;
* can\_create\_categories – права на создание категорий;
* can\_buy\_products – права на покупку товаров;
* can\_comment – права на создание отзывов;
* can\_moderate – права на модерацию контента;
* can\_chat – права на отправку сообщений;
* can\_see\_statistics – права на просмотр статистики;
* can\_create\_moderators – права на назначение и удаление модераторов
* can\_create\_priveleged\_users – права на назначение и удаление привилегированных пользователей.

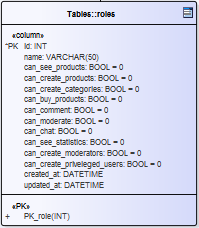


Рисунок 2.9 – UML диаграмма для таблицы roles

Таблица comments содержит данные об оставленных пользователями отзывах (рисунок 2.10).

* rate – числовая оценка;
* text – текстовый комментарий пользователя;
* product\_id – id товара, внешний ключ;
* user\_id – id пользователя, внешний ключ.

Таблица содержит индексы на поля product\_id и user\_id.

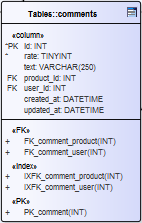


Рисунок 2.10 – UML диаграмма для таблицы comments

Таблица payment\_infos содержит данные об платежных реквизитах пользователей (рисунок 2.11).

* type – тип платежной информации;
* data – сериализованные данные о реквизитах оплаты;
* currency\_id – id валюты, внешний ключ;
* user\_id – id пользователя, внешний ключ.

Таблица содержит индексы на поля currency\_id и user\_id.

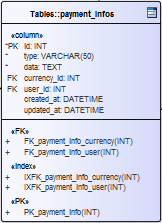


Рисунок 2.11 – UML диаграмма для таблицы payment\_infos

Таблица products содержит данные о продаваемых товарах (рисунок 2.12).

* name – название товара;
* description– описание товара;
* price– цена за единицу;
* quantity– количество в продаже;
* rating – сумма всех оценок пользователей;
* rates\_count – количество оценок;
* is\_paused – флаг, показывающий остановлена ли торговля;
* is\_archived – флаг, показывающий помещен ли товар в архив;
* is\_blocked – флаг блокировки;
* currency\_id – id валюты, внешний ключ;
* category\_id – id категории, внешний ключ;
* owner\_id – id владельца товара, внешний ключ.

Таблица содержит индексы на поля currency\_id, category\_id, is\_archived и owner\_id.

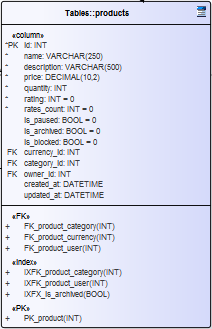


Рисунок 2.12 – UML диаграмма для таблицы products

Таблица transactions содержит данные о проведенных транзакциях (рисунок 2.13).

* sender\_id – id отправителя, внешний ключ;
* receiver\_id– id получателя, внешний ключ;
* price– сумма транзакции;
* deal\_id– количество в продаже.

Таблица содержит индексы на поля deal\_id, receiver\_id, sender\_id.

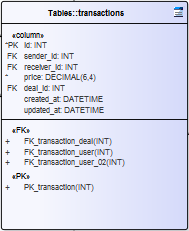


Рисунок 2.13 – UML диаграмма для таблицы transactions

Таблица currency содержит данные об используемых в приложении валютах (рисунок 2.14).

* name – название валюты;
* rate – курс валюты по отношению к доллару США;
* code– сокращенное название валюты.

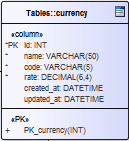


Рисунок 2.14 – UML диаграмма для таблицы currency

Таблица properties содержит данные о свойствах категорий. Для каждой категории определен уникальный набор свойств и записей для них в таблице (рисунок 2.15).

* name – название свойства;
* description – описание свойства;
* category\_id – id категории, внешний ключ;
* property\_type\_id – id типа свойства, внешний ключ.

Таблица содержит индексы на поля category\_id и property\_type\_id.

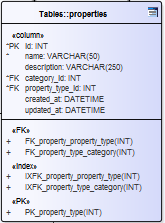


Рисунок 2.15 – UML диаграмма для таблицы properties

Таблица property\_types содержит данные о типах свойств категорий. (например текстовое, числовое, ссылка, картинка и др.) Список типов свойств определен изначально и не редактируется пользователями (рисунок 2.16).

* name – имя типа свойства используемое в коде;
* description – описание типа свойства;
* display\_name – название свойства которое отображается пользователям.

Таблица содержит индексы на поля category\_id и property\_type\_id.

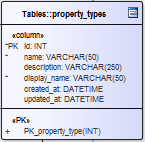


Рисунок 2.16 – UML диаграмма для таблицы property\_types

Таблица property\_parameters содержит информацию о параметрах для свойств. У каждого типа свойств есть некоторое число параметров (например у типа свойств «диапазон» есть параметры «минимум» и «максимум» и т.д.). Для свойств конкретной категории эти параметры имеют свои значения. (у разных категорий минимум и максимум в диапазоне может быть разным)

* key – ключ параметра;
* value – значение параметра;
* property\_id – id свойства, внешний ключ.

Таблица содержит индекс на поле property\_id (рисунок 2.17).

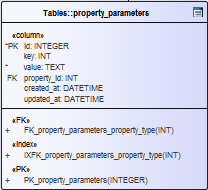


Рисунок 2.17 – UML диаграмма для таблицы property\_parameters

Таблица deals содержит информацию об уже совершенных сделках (рисунок 2.18).

* seller\_id – id продавца, внешний ключ;
* buyer\_id – id покупателя, внешний ключ;
* product\_id – id товара, внешний ключ;
* price – сумма сделки;
* currency\_id – id валюты, внешний ключ.

Таблица содержит индексы на поля seller\_id, buyer\_id, product\_id, currency\_id.

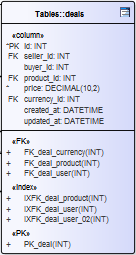


Рисунок 2.18 – UML диаграмма для таблицы deals

Таблица property\_values содержит информацию о значениях свойств товаров (у каждого товара свои значения свойств) (рисунок 2.19).

* value – значение свойства;
* property\_id – id свойства, внешний ключ;
* product\_id – id товара, внешний ключ.

Таблица содержит индексы на поля property\_id, product\_id

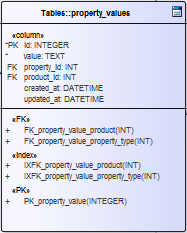


Рисунок 2.19 – UML диаграмма для таблицы property\_values

Таблица categories информацию о категориях товаров (рисунок 2.20).

* name – название категории;
* description – описание категории;
* avatar\_link – ссылка на аватар категории;
* is\_archived – флаг архивирования.

Таблица содержит индекс на поле is\_archived

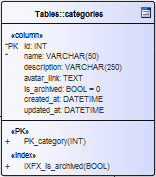


Рисунок 2.20 – UML диаграмма для таблицы categories

## **2.3** Разработка спецификации требований

Основной задачей данной работы является разработка веб приложения на технологиях Ruby on Rails, AngularJS, MySQL, JavaScript, UnderscoreJS, Javascript, HTML, SASS. Приложение должно корректно выполнять функции, описанные в подразделе 1 раздела 2, использовать информационную модель, описанную в подразделе 2 раздела 2, а также соответствовать нефункциональным требованиям, изложенным в разделе 1.

# **3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## **3.1** Разработка архитектуры программного средства

Фреймворк Ruby on Rails обычно используется для создания полноценных приложений. В этом случае Rails позволяет описывать клиентскую часть приложения, HTML и CSS генерируется на сервере и клиенту отсылается уже готовая страница. Для реализации данного проекта я использовал другую схему построения архитектуры: на Ruby on Rails реализована только серверная часть приложения. Сервер в данном случае предоставляет API для клиентской части. API предоставляет методы по работе с ресурсами сервера. Обмен данными происходит в формате JSON. Параллельно с сервером Rails, будет работать сервер с AngularJS. Приложение на AngularJS будет использовать ресурсы, предоставляемые сервером Rails путем вызова соответствующих методов API. При этом часть вычислений переносится на машину клиента, что уменьшает нагрузку на сервер, что особенно важно для высоконагруженных приложений.

Архитектура приложения представлена на рисунке 3.1.

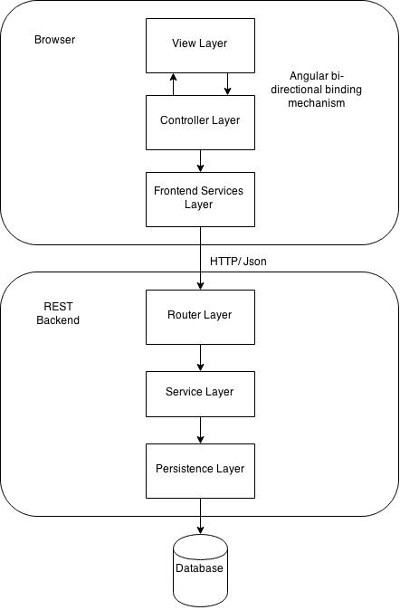


Рисунок 3.1 – Общая архитектура приложения

Для построения взаимодействия между клиентом и сервером я решил использовать стиль REST.

REST (сокр. от англ. Representational State Transfer – «передача состояния представления») – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине.

В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно GET или POST; такой запрос называют REST-запрос), а необходимые данные передаются в качестве параметров запроса.

Для построение архитектуры в стиле REST необходимо выполнить следующие требования:

* Единый интерфейс между клиентом и сервером. Такое разделение подразумевает отсутствие связи между клиентами и хранилищем данных. Это хранилище остаётся внутренним устройством сервера, таким образом переносимость клиентского кода увеличивается, что способствует упрощению сервера и его масштабируемости. Серверы и клиенты могут быть мгновенно заменены независимо друг от друга, так как интерфейс между ними не меняется.
* Отсутствие состояния. Серверы не связаны с интерфейсами клиентов и их состояниями. На стороне сервера не сохраняется пользовательский контекст между двумя разными запросами. Каждый запрос содержит всю информацию, необходимую обработчику, а состояние сессии хранится на клиенте. Состояние сессии может быть передано сервером на другой сервис благодаря поддержке постоянного состояния базой данных. Клиент отсылает запросы, когда готов совершить транзакцию на изменение состояния;
* Кеширование. Как и во Всемирной паутине, каждый из клиентов, а также промежуточные узлы между сервером и клиентами могут кэшировать ответы сервера. В каждом запросе клиента должно явно содержаться указание о возможности кэширования ответа и получения ответа из существующего кэша. В свою очередь, ответы могут явно или неявно определяться как кэшируемые или некэшируемые для предотвращения повторного использования клиентами в последующих запросах сохранённой информации. Правильное использование кэширования в REST-архитектуре устраняет избыточные клиент-серверные взаимодействия, что улучшает скорость и расширяемость системы.
* Единообразие интерфейса. Ограничения на унифицированный интерфейс являются фундаментальными в дизайне REST-сервисов. Каждый из сервисов функционирует и развивается независимо.
* Слои. Клиент может взаимодействовать не напрямую с сервером, а через промежуточные узлы (слои). При этом клиент может не знать об их существовании, за исключением случаев передачи конфиденциальной информации. Промежуточные серверы выполняют балансировку нагрузки и могут использовать дополнительное кэширование.

Приложение полностью удовлетворяющее всем этим требованиям получит следующие преимущества:

* надёжность (за счёт отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна);
* производительность (за счёт использования кэша);
* масштабируемость;
* прозрачность системы взаимодействия (особенно необходимая для приложений обслуживания сети);
* простота интерфейсов;
* портативность компонентов;
* лёгкость внесения изменений;
* способность эволюционировать, приспосабливаясь к новым требованиям (на примере Всемирной паутины).

И Rails и AngularJS полностью поддерживают REST архитектуру. Также и AngularJS и Rails поддерживают архитектуру MVC. Принципы и преимущества использования MVC описаны в разделе 1.

Архитектура MVC представлена на рисунке 3.2.

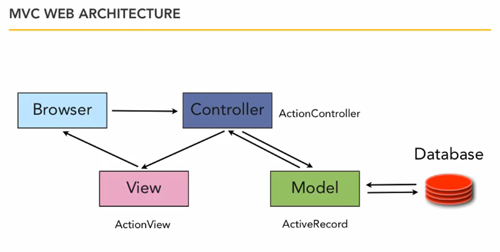


Рисунок 3.2 – Архитектура приложения на MVC.

## **3.2** Техническое проектирование программного средства

Проектирование программного средства я начал с создания базы данных. Для манипуляции со структурой базы данных Rails использует механизм миграций. Миграции базы пишутся на языке Ruby DSL, которые не зависит от синтаксиса конкретной СУБД. Для работы с CRUD (Create Read Update Delete) операций с базой используется ORM фреймворк под названием Active Record. Использование этих двух механизмов позволяет написать сделать модель (уровень в MVC) независимой от конкретной СУБД. Это повышает переносимость приложения, а также освобождает программиста от работы связанной с взаимодействием приложения и базы данных.

В предыдущих разделах я разработал схему базы данных в среде Enterprise Architect. Эта среда позволяет генерировать SQL скрипт, создающий в базе таблицы, поля, связи и другие объекты. Созданный скрипт я импортировал в пустую MySQL базу тем самым физически создав базу данных для приложения. Далее с помощью консольной команды «rake db:schema:dump» я создал схему базы данных и поместил её в проект Rails приложения. Далее в целях контроля версий я использовал созданную схему для написания миграции базы. В дальнейшем если возникнет необходимость внести изменения в структуру базы это можно будет легко сделать написав ещё одну миграцию, вносящую необходимые изменения.

Далее я связи на уровне модели. Rails использует соглашения об именовании таблиц, классов, полей и других элементов и автоматически находит таблицы, соответствующий каждой модели, внешние ключи в таблицах и другое. Если следовать соглашению об именовании это убирает необходимость явно указывать соответствие между таблицами и моделями. Для создания связей на уровне модели используются ключевые слова «has\_one», «belongs\_to», «has\_many» и другие в зависимости от вида связи. Далее указывается модель, с которой устанавливается связь. Rails автоматически установит соответствие между таблицами, найдет внешние ключи и будет извлекать или модифицировать нужные записи при обращении к соответствующим свойствам модели. К примеру если установить для модели User связь «belongs\_to :role», получение роли пользователя упрощается от «Role.where(role\_id => user.id)» до «user.role» и так далее.

Когда связи между моделями установлены я занялся разработкой методов моделей. Методы моделей позволяют структурировать код, уменьшить его количество за счет повторного использования. При написании как серверного так и клиентского кода я руководствовался принципами, изложенными в книге Боба Мартина «Чистый код». В частности использованием «говорящих» имен методов и переменных, а также использованием маленьких, узкоспециализированных методов. Использование этих принципов позволяет добиться наглядности и хорошей читабельности кода, что упрощает отладку и улучшает поддерживаемость приложения. К примеру операция изменения прав пользователя меняется с «user.role = Role.where(name => «manager»); user.save» до «user.make\_manager» и так далее.

Далее я написал таблицу маршрутизации, на основе url запроса и его вида направляющую запрос в нужный метод контроллера.

Далее я разработал код контроллеров. При использовании стиля REST, контроллеры выполняют только функции отправки данных клиенту и получения данных от клиента, вся бизнес логика реализована в методах модели.

Далее я занялся разработкой клиентской части приложения. Составил список страниц приложения (страниц в фигуральном смысле, в одностраничном приложении нет страниц. Под страницами понимаются части сайта с разными видами контента) и написал для них список маршрутизации – каждый введенный а адресную строку адрес соответствует своему контроллеру и своему HTML шаблону.

Далее создал файлы контроллеров и шаблонов в AngularJS приложении, написал HTML разметку (верстку) каждой страницы. Написал код для контроллеров, реализующий клиентскую логику приложения. Закончил разработку приложения созданием дизайна на SASS и отладкой, совмещенной со смоук тестированием.

# **4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## **4.1** Функциональное тестирование программного средства

Тестирование проводится с целью обеспечить качество разрабатываемого программного продукта. Стандарт ISO-8402, посвященный описанию систем обеспечения качества программного обеспечения, под качеством понимает «совокупность характеристик программного продукта, относящихся к его способности удовлетворять установленные и предполагаемые потребности клиента». Основным параметром качества программы является надёжность. Надёжность определяется как вероятность его работы без отказов в течении определённого периода времени, рассчитанная с учётом стоимости для пользователя каждого отказа. Отказ программного обеспечения – это проявление ошибки в нём. Отсюда тестирование ПО – это процесс выполнения программы с целью обнаружения в ней ошибок. «Удачным» тестом является такой, на котором выполнение программы завершилось с ошибкой. Напротив, «неудачным» называется тест, не позволивший выявить ошибку в программе.

Ввиду того, что данное программное средство является веб приложением основным браузером, в котором проводилось тестирование, является Google Chrome версии 50.0.2661.102 m. Так как данное приложение является веб-приложением, то для доступа к нему пользователю необходим только браузер.

Для обнаружения ошибок в работе приложения проводилось функциональное тестирование. Тестирование готового программного средства проводилось несколькими пользователями на разных компьютерах с разными операционными системами. Тесты, проведенные над программой, представлены в таблице 4.1.

Таблица 4.1 – Тестирование программы

|  |  |  |  |
| --- | --- | --- | --- |
| Страница веб-приложения | Содержание теста | Ожидаемый результат | Тест пройден |
| Страница регистрации | Ввод данных и нажатие кнопки «Регистрация» | Создание нового пользователя, отображение сообщения об успешной регистрации. | Да |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Страница веб-приложения | Содержание теста | Ожидаемый результат | Тест пройден |
| Станица авторизации | Ввод логина и пароля и нажатие кнопки «Авторизация» | Авторизация пользователя, отображения сообщения об успешной авторизации | Да |
| Станица авторизации | Ввод некорректных данных, нажатие кнопки «Авторизация» | Отображение сообщения об ошибке | Да |
| Страница  списка товаров | Выбор желаемых параметров товара, нажатие кнопки «Показать» | Отображение списка товаров с желаемыми параметрами | Да |
| Страница  списка товаров | Нажатие кнопки «Добавить товар» | Переход на страницу создания товара | Да |
| Страница списка товаров | Нажатие кнопки «Описание» | Переход на страницу описания выбранного товара | Да |
| Страница описания товара | Нажатие кнопки «Купить» | Создание новой транзакции, отображение сообщения об успешной покупке | Да |
| Страница описания товара | Нажатие кнопки «Удалить» | Удаление товара, отображение сообщения об успешно выполненной операции | Да |
| Страница описания товара | Нажатие кнопки «Редактировать» | Переход на страницу редактирования товара | Да |
| Страница описания товара | Нажатие кнопки «Отмена» | Переход на страницу списка товаров | Да |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Страница веб-приложения | Содержание теста | Ожидаемый результат | Тест пройден |
| Страница описания товара | Ввод комментария, нажатие кнопки «Отправить» | Создание нового комментария, отображение нового комментария в списке комментариев | Да |
| Страница создания товара | Ввод данных, нажатие кнопки создать | Создание нового товара, отображение сообщения об успешно выполненной операции | Да |
| Страница создания товара | Нажатие кнопки «Отмена» | Переход на страницу описания товара | Да |
| Страница редактирования товара | Ввод данных, нажатие кнопки «Сохранить» | Редактирования данных о товаре, отображение сообщения об успешно выполненной операции. | Да |
| Страница редактирования товара | Нажатие кнопки «Отмена» | Переход на страницу описания товара | Да |
| Страница списка транзакций | Переход на страницу | Отображение списка совершенных пользователем транзакций | Да |
| Страница списка категорий | Переход на страницу | Отображение списка категорий | Да |
| Страница списка категорий | Нажатие кнопки «Создать | Переход на страницу создания категории | Да |
| Страница создания категории | Ввод данных, нажатие кнопки «Создать» | Создание категории, отображение сообщения об успешно выполненной операции | Да |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Страница веб-приложения | Содержание теста | Ожидаемый результат | Тест пройден |
| Страница профиля пользователя | Переход на страницу | Отображение данных о пользователе | Да |

## **4.2** Модульное тестирование программного средства

Модульное тестирование, или юнит-тестирование – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Цель модульного тестирования – изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. Этот тип тестирования обычно выполняется программистами.

Преимущества:

* Модульное тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно (регрессионное тестирование). Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.
* Модульное тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируя отдельные части программы, а затем программу в целом.
* Модульные тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера.

Для тестирования серверного функционала я использовал гем RSpec. RSpec – модуль расширение (гем) для фреймворка Ruby On Rails, предназначенный для написания автоматизированных тестов, ставший стандартом среди разработчиков. Для проверки функциональности и улучшения поддерживаемости приложения я написал тесты для ключевых участков кода, обеспечивающих основной функционал приложения.

Для написания тестов клиентской части приложения я использовал библиотеку KarmaJS. Обычно автоматические модульные и интеграционные тесты рекомендуется создавать для выполненного функционала для того, чтобы уменьшить риск регрессионных ошибок при изменении кода в будущем. В случае с JavaScript подобные тесты могут существенно упростить проверку работоспособности системы в различных браузерах путем автоматизации действий по обеспечению такой проверки. Кроме того, добрую службу может сослужить написание модульного или интеграционного теста на каждый закрытый баг в продукте.

# **5 МЕТОДИКА ИСПОЛЬЗОВАНИЯ РАЗРАБОТАННОГО ПРОГРАММНОГО СРЕДСТВА**

В начале работы с приложением пользователь попадает на страницу списка товаров – это главная страница приложения. В начале работы пользователь ещё не авторизован и ему доступны только функции просмотри списка товаров. Для авторизации в приложении пользователю сначала нужно зарегистрироваться. На страницу регистрации можно перейти нажав кнопку «Регистрация».   
 На странице регистрации пользователь может ввести свои данные для регистрации и нажать кнопку «Зарегистрироваться», тем самым создав в базе нового пользователя, а может нажать кнопку «Отмена» и вернуться на главную страницу.

Когда пользователь уже зарегистрирован он может войти на сайт. Для этого ему нужно нажать на кнопку «Вход» на главной странице и он будет перенаправлен на страницу авторизации. На странице авторизации пользователю нужно ввести свой адрес электронной почты и пароль, указанные при регистрации и нажать кнопку «Войти». Если адрес электронной почты и пароль введены верно пользователь увидит сообщение об успешном входе на сайт, если неверно – сообщение об ошибке. Также на странице авторизации пользователь может нажать кнопку «Отмена» чтобы вернуться на главную страницу.

Авторизованному пользователю доступен основной функционал приложения. С главной страницы пользователь может попасть на страницу описания товара нажав на кнопку «Описание» рядом с желаемым товаром, он может подбирать нужные товары вводя свойства товаров в соответствующие поля. С главной страницу пользователь может попасть на страницу списка транзакций, нажав на кнопку «Транзакции», на страницу списка категорий, нажав на кнопку «Категории», на страницу списка пользователей, нажав но кнопку «Пользователи», на страницу добавления товара, нажав на кнопку «Добавить товар».

На странице описания товара пользователь может видеть подробное описание товара, а также кнопку «Купить», нажав на которую он может оформить покупку выбранного товара. Если товар на странице принадлежит пользователю или пользователь является администратором он может удалить товар нажав на кнопку «Удалить». Для создателя товара доступна также функция редактирования товара – нажав но кнопку «Редактировать» пользователь может перейти на страницу редактирования. Также на странице описания товара пользователь видеть список комментариев, а также оставить свой комментарий, ещё пользователь может перейти обратно на страницу списка товаров нажав кнопку «Отмена».

На странице создания товара пользователь может ввести данные о товаре и нажать кнопку «Создать», тем самым создав новый товар, а может ажать кнопку «Отмена» и вернуться на страницу списка товаров.

На странице редактирования товара пользователь может ввести новые данные о товаре и нажать кнопку «Сохранить», а может нажать кнопку «Отмена» и вернуть на страницу описания товара.

На странице списка категорий пользователь может видеть список категорий и кнопку «Создать категорию», нажав на которую он перейдет на страницу создания категории.

На странице создания категории пользователь вводит данные о новой категории и нажимает кнопку «Создать» либо нажимает кнопку «Отмена» и возвращается на главную.

На странице списка транзакций пользователь видит список транзакций с его участием.

На странице списка пользователей отображается список зарегистрированных в приложении пользователей.

# **6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА ОСУЩЕСТВЛЕНИЯ ТОРГОВЫХ СДЕЛОК ПО БЕЗНАЛИЧНОМУ РАСЧЕТУ**

## **6.1** Характеристика проекта

Интернет развивается довольно стремительно. Быстро растет количество издании, посвященных Сети, что предвещает широкое ее распространение даже в далеких от техники областях. Интернет превращается из большой игрушки для интеллектуалов в полноценный источник всевозможной информации для всех.

Ожидается существенный рост увеличения объемов Интернет-коммерции, особенно в таких областях, как путешествия, розничная торговля, финансы, тематическая реклама, а также в компьютерном секторе.

Преимущества электронной торговли для покупателя очевидны: отсутствие географической привязанности к торговой точке, он доступен 24 часа в сутки, 7 дней в неделю, 365 дней в году.

Проект позволяет осуществлять торговлю онлайн, выполняет следующие функции: добавление товара, покупка товара, продажа товара, подбор и сортировка товаров по интересующим критериям, просмотр истории транзакций, оставление комментария к товару, просмотр комментариев других пользователей, просмотр описания товара, создание категорий товара, добавление товара в категорию.

## **6.2** Расчет затрат на разработку и отпускной цены продукта

Основная заработная плата исполнителей (Зо) рассчитывается по формуле:

K, (6.1)

где n – количество исполнителей, занятых разработкой;

Тчi – часовая тарифная ставка i-го исполнителя, руб.;

Тч – количество часов работы в день, ч.;

Фпi – плановый фонд рабочего времени i-го исполнителя, дн.;

К – коэффициент премирования.

Месячная тарифная ставка первого разряда на 01.02.2016 г. составляет 1 988 000 рублей. Среднемесячная расчетная норма рабочего времени для пятидневной рабочей недели на 2016 год составляет 170 часа. Коэффициент премирования равен 1,4.

Учитывая число разработчиков *n* = 1, определим основную заработную плату по формуле (6.1):

1.4 = 30 255 021 руб.

Дополнительная заработная плата включает выплаты, предусмотренные законодательством о труде, и определяется по формуле

, (6.2)

где Нд – норматив дополнительной заработной платы, 10%;

Размер дополнительной заработной платы исполнителей составит:

руб.

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

, (6.3)

где Нсз – норматив отчислений в фонд социальной защиты населения и на обязательное страхование, 34,6%.

Размер отчислений в фонд социальной защиты населения и на обязательное страхование составит:

руб.

Расходы по статье «Машинное время» (Рм) определяются по формуле:

, (6.4)

где Цм – цена одного часа машинного времени, 4500 руб.;

*V*о – общий объем программного средства (строк исходного кода);

Нмв – норматив расхода машинного времени на отладку 100 строк исходного кода, 12%.

Расходы на использование машинного времени составят, с учетом того, что общий объем программного средства составляет 13 020 строк кода, а также с учетом понижающего коэффициента 0.5 составят:

руб.

Расходы по статье «Научные командировки» (Рнк) на программное средство (ПС) определяются по формуле:

, (6.5)

где Нрнк – норматив расходов на командировки в целом по организации, 15%.

Расходы по статье «Научные командировки» составят:

руб.

Расходы по статье «Прочие затраты» (Пз) на ПС включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы и определяются по формуле:

, (6.6)

где Нпз – норматив прочих затрат в целом по организации, 20%.

Расходы по статье «Прочие затраты» составят:

руб.

Затраты по статье «Накладные расходы» (Рн) определяются с помощью следующей формулы:

, (6.7)

где Нрн – норматив накладных расходов в целом по организации, 50%.

Затраты по статье «Накладные расходы» составят:

руб.

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

. (6.8)

Общая сумма расходов по всем статьям сметы на разработку ПС составит:

руб.

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию ПС (Рса), которые определяются по формуле:

, (6.9)

где Нрса – норматив расходов на сопровождение и адаптацию, 5%;

Ср – общая сумма расходов по всем статьям сметы на разработку ПС.

Затраты на сопровождение и адаптацию программного продукта составят:

руб.

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

. (6.10)

Полная себестоимость программного средства составит:

руб.

**6.3** Расчет стоимостной оценки результата от разработки программного продукта

Для определения экономического эффекта от использования нового ПО необходимо сравнить расходы по всем основным статьям сметы затрат на эксплуатацию нового ПО (расходы на заработную плату с начислениями, материалы, машинное время) с расходами по соответствующим статьям при использовании прежнего варианта ПО. При сравнении базового и нового вариантов ПО в качестве экономического эффекта будет выступать общая экономия всех видов ресурсов относительно базового варианта. При этом создание нового ПО окажется экономически целесообразным лишь в том случае, если все капитальные затраты окупятся за счет получаемой экономии в ближайшие 2–3 года.

Общие капитальные вложения заказчика (потребителя) рассчитываются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.11) |

где Кпр – затраты пользователя на приобретение ПО по отпускной цене, руб.;

Кос – затраты пользователя на освоение ПС, руб.;

Кс – затраты пользователя на оплату услуг по сопровождению ПО, руб.;

Ктс – затраты на доукомплектование ВТ техническими средствами в связи с внедрением нового ПО, руб.;

Коб – затраты на пополнение оборотных средств в связи с использованием нового ПО, руб.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Экономия затрат на заработную плату (Сз) при использовании нового ПО в расчете на объем выполненных работ определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.12) |

где Сзе – экономия затрат на заработную плату на 1 задачу, руб.;

А2 – объем выполненных работ с использованием нового ПО (задач).

Экономия затрат на заработную плату в расчете на 1 задачу рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.13) |

где Зсм – среднемесячная заработная плата одного программиста, руб.;

Тс1, Тс2 – трудоемкости работ в расчете на 1 задачу, человеко-часов;

Тч – количество часов работы в день, ч;

Др – среднемесячное количество рабочих дней.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Тогда, подставляя значения в формулу (6.25), рассчитаем экономию затрат на заработную плату:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Экономия с учетом начисления на зарплату вычисляется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.14) |

где Кнз – коэффициент начислений на зарплату (1,5).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Экономия затрат на оплату машинного времени (См) в расчете на выполненный объем работ в результате применения нового ПО:

|  |  |  |
| --- | --- | --- |
|  |  | (6.15) |

где Сме – экономия затрат на оплату машинного времени на 1 задачу.

Экономия затрат на оплату машинного времени в расчете на 1 задачу определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.16) |

где Цм – цена одного машино-часа работы ЭВМ;

Мв1, Мв2 – средний расход машинного времени при применении соответственно базового и нового ПО.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Таким образом, по формуле (6.28) определим экономию затрат на оплату машинного времени в расчете на выполненный объем работ:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Общая годовая экономия текущих затрат, связанных с использованием нового ПО является важным фактором, влияющим в дальнейшем на расчеты, производимые в данном разделе и вычисляется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.17) |

Подставляя ранее полученные значения в формулу (6.30), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Внедрение нового ПО позволит заказчику сэкономить на текущих затратах 131 579 651 рубля. Для заказчика в качестве экономического эффекта выступает лишь чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении, которая определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.18) |

где Нп – ставка налога на прибыль (18%).

|  |  |  |
| --- | --- | --- |
|  |  |  |

В процессе использования нового ПО 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 2016 год) путем умножения результатов и затрат за каждый год на коэффициент дисконтирования α*t*, который рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.19) |

где *E* – норматив приведения разновременных затрат и результатов;

*t* – номер года, результаты и затраты которого приводятся к расчётному

*tp* – номер расчётного года (2016).

Таким образом, получим следующие значения коэффициентов дисконтирования:

2016 год: 

2017 год: 

2018 год: 

2019 год: 

Сведем данные расчета экономического эффекта в таблицу 6.1.

Таблица 6.1 – Расчет экономического эффекта от использования нового ПС

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Ед. изм. | Годы | | | |
| 2016 | 2017 | 2018 | 2019 |
| 1 | 2 | 3 | 4 | 5 | 6 |
| *Результаты:* | | | | | |
| Прирост прибыли за счет экономии затрат (Пч) | руб. |  | 107 895 314 | 107 895 314 | 107 895 314 |
| То же с учетом фактора времени | руб. |  | 87 012 350 | 70 171 250 | 56 589 718 |
| *Затраты:* | | | | | |
| Приобретение ПО (Кпр) | руб. | 109 826 620 |  |  |  |
| Освоение ПО (Кос) | руб. | 7 626 849 |  |  |  |

Продолжение таблицы 6.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Ед. изм. | Годы | | | |
| 2016 | 2017 | 2018 | 2019 |
| Сопровождение (Кс) | руб. | 15 253 697 |  |  |  |
| Доукомплектование ВТ техническими средствами (Ктс) | руб. | 7 626 849 |  |  |  |
| Пополнение оборотных средств (Коб) | руб. | 2 500 000 |  |  |  |
| То же с учетом фактора времени | руб. | 142 834 014 |  |  |  |
| *Экономический эффект:* | | | | | |
| Показатели | Ед. изм. | Годы | | | |
| 2016 | 2017 | 2018 | 2019 |
| Превышение результатов над затратами | руб. | -142 834 014 | 87 012 350 | 70 171 250 | 56 589 718 |
| То же нарастающим итогом | руб. | -142 834 014 | -55 821 664 | 14 349 586 | 70 939 304 |
| Коэффициент приведения | ед. | 1 | 0,8065 | 0,6504 | 0,5245 |

В данном разделе была рассчитана себестоимость проекта, на основании которой была сформирована рыночная цена. Был рассчитан экономический эффект для разработчика: чистая прибыль от реализации ПО составила 12 508 032 рублей.

Кроме этого, была рассчитана экономическая эффективность для заказчика, выраженная в чистом дисконтированном доходе, составившем 70 939 304 рублей. Стоит отметить, что затраты заказчика окупятся менее, чем за два года. Таким образом, учитывая снижение трудоёмкости, значительную экономию затрат, достаточно быстрый срок окупаемости, можно сделать вывод, что проект является полезным и экономически эффективным для заказчика.

# **ЗАКЛЮЧЕНИЕ**

При выполнении данной работы я разработал программное средство, позволяющее выполнять торговые операции онлайн по безналичному расчету. Я более детально изучил технологии такие как Ruby on Rails, AngularJS, JavaScript, MySQL, HTTP, HTML, CSS, научился проектировать архитектуру приложения, проектировать требования к приложению, разрабатывать проектную документация документацию на всех этапах, рассчитывать экономическую эффективность проекта.

# **Список ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. At-www [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.at-www.ru/rol-html-v-sozdanii-web-stranic.html
2. Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: http://ru.wikipedia.org/wiki/JavaScript.
3. Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: http://ru.wikipedia.org/wiki/Model-View-Controller.
4. Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: http://ru.wikipedia.org/wiki/AngularJS
5. 3DNews [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.3dnews.ru/920530.
6. Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: http://ru.wikipedia.org/wiki/UML.

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Код программного модуля products\_controller.rb**

class Api::ProductsController < ApplicationController

def index

if current\_user.can\_see\_products?

products = Product.where\_params(params[:product])

products = products.map{ |product| product.as\_json }

render\_ok({ products: products }.as\_json)

else

render\_forbidden

end

end

def show

if current\_user.can\_see\_products?

product = product.where(id: params[:product][:id]).first if params[:product][:id]

render\_ok product.as\_json if product.present?

render\_not\_found if product.blank?

else

render\_forbidden

end

end

def create

if current\_user.can\_create\_products?

product = Product.new\_from\_params(params[:product])

product.owner\_id = current\_user.id

if product.save

render\_created(product.as\_json)

else

render\_bad\_request("Some parameters are invalid")

end

else

render\_forbidden

end

end

def update

if current\_user.can\_create\_products? && current\_user.owns\_product?(params[:product]) || current\_user.can\_moderate?

product = Product.where(id: params[:product][:id]).first

if product.present?

product.update\_allowed\_fields(params[:product])

if product.save

render\_ok(product.as\_json)

else

render\_bad\_request("Some parameters are invalid")

end

else

render\_not\_found

end

else

render\_forbidden

end

end

def destroy

if current\_user.can\_create\_products? && current\_user.owns\_product?(params[:product][:id]) || current\_user.can\_moderate?

product = Product.where(id: params[:product][:id]).first

if product.present?

product.is\_archived = true

product.save

render\_ok

else

render\_not\_found

end

else

render\_forbidden

end

end

def check\_params

render\_bad\_request if params[:product].blank? || !params[:product].is\_a?(Hash)

end

end

# **ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Код программного модуля application\_controller.rb**

class ApplicationController < ActionController::Base

before\_action :authenticate

protected

rescue\_from Exception do |exception|

render\_exception(exception)

end

#Renderers

def render\_ok(json\_object)

render json: json\_object, status: :ok

end

def render\_created(json\_object)

render json: json\_object, status: :created

end

def render\_bad\_request(message = nil)

responce = { message: message || "Incorrect parameters", stacktrace: caller[0, 10] }

render json: responce.as\_json, status: :bad\_request

end

def render\_error(message = nil)

responce = { message: message || "Internal server error", stacktrace: caller[0, 10] }

render json: responce.as\_json, status: :internal\_server\_error

end

def render\_forbidden(message = nil)

responce = { message: message || "Access denied", stacktrace: caller[0, 10] }

render json: responce.as\_json, status: :forbidden

end

def render\_not\_found(message = nil)

responce = { message: message || "Resource not found", stacktrace: caller[0, 10] }

render json: responce.as\_json, status: :not\_found

end

def render\_exception(e)

error\_info = {

:error => "Internal server error",

:exception => "#{e.class.name} : #{e.message}",

}

error\_info[:stacktrace] = e.backtrace[0,10] if Rails.env.development?

render :json => error\_info.to\_json, :status => :internal\_server\_error

end

#

#Autentication

def current\_user

token = request.headers['auth-token']

user = User.where(token: token).first

return user

end

def authenticated?

authenticate\_token

end

def authenticate

authenticate\_token || render\_unauthorized

end

def authenticate\_token

token = request.headers['auth-token']

user = User.where(token: token).first

return user.present? && !user.token\_expired?

end

def render\_unauthorized

responce = { message: "Unauthorized access", stacktrace: caller[0, 10] }

render json: responce.as\_json, status: :unauthorized

end

#

end

# **ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Код программного модуля user.rb**

class User < ActiveRecord::Base

belongs\_to :country

belongs\_to :role

has\_many :incoming\_transactions, class\_name: "Transaction", foreign\_key: "receiver\_id"

has\_many :outcoming\_transactions, class\_name: "Transaction", foreign\_key: "sender\_id"

has\_many :sales, class\_name: "Deal", foreign\_key: "seller\_id"

has\_many :purchases, class\_name: "Deal", foreign\_key: "buyer\_id"

has\_many :payment\_infos

has\_many :products

has\_many :comments

validates :email, uniqueness: true

validates :login, uniqueness: true

validates :token, uniqueness: true

UPDATE\_FIELDS = [

"first\_name",

"last\_name",

"passport\_id",

"avatar\_url",

"address\_line\_1",

"address\_line\_2",

"post\_index",

"phone\_number",

"birth\_date",

"country\_id",

"is\_blocked",

"is\_archived"

]

SEARCH\_FIELDS = [

"login",

"email",

"first\_name",

"last\_name",

"phone\_number",

"country\_id",

"role\_id",

"is\_blocked",

"is\_archived"

]

def self.new\_from\_params(params)

user = User.new

user.login = params[:login]

user.email = params[:email]

user.update\_allowed\_fields(params)

user.update\_password(params)

user.add\_users\_permissions

user.generate\_token

if params[:payment\_infos].present? && params[:payment\_infos].is\_a?(Array)

params[:payment\_infos].each{ |payment\_info\_params|

payment\_info = PaymentInfo.new\_from\_params(payment\_info\_params)

payment\_info.save

user.payment\_infos.push(payment\_info)

}

end

return user

end

def update\_allowed\_fields(params)

UPDATE\_FIELDS.each{ |field|

self[field] = params[field] if params[field].present?

}

if params[:payment\_infos].present? && params[:payment\_infos].is\_a?(Array)

existing\_ids = self.payment\_infos.pluck(:id)

updated\_payment\_infos = params[:payment\_infos].select{ |payment\_info| payment\_info[:id] }

updated\_ids = update\_payment\_infos.map{ |payment\_info| payment\_info[:id] }

new\_payment\_infos = params[:payment\_infos].select{ |payment\_info| !payment\_info[:id] }

deleted\_ids = existing\_ids - updated\_ids;

updated\_payment\_infos.each{ |payment\_info\_params|

payment\_info = PaymentInfo.where(id: payment\_info\_params[:id]).first

payment\_info.update\_allowed\_fields(payment\_info\_params)

payment\_info.save

}

new\_payment\_infos.each{ |payment\_info\_params|

payment\_info = PaymentInfo.new\_from\_params(payment\_info\_params)

payment\_info.save

self.payment\_infos.push(payment\_info)

}

PaymentInfo.where(id: deleted\_ids).delete\_all

end

end

def self.where\_params(params)

clean\_params = {}

SEARCH\_FIELDS.each{ |field|

clean\_params[field.to\_sym] = params[field.to\_sym] if params[field.to\_sym]

}

puts clean\_params

return User.where(clean\_params)

end

def update\_password(params)

self.encrypted\_password = Digest::SHA2.hexdigest(params[:password])

end

def add\_users\_permissions

self.role = Role.find\_by\_name("User")

end

def add\_privileged\_permissions

self.role = Role.find\_by\_name("Privileged")

end

def add\_moderators\_permissions

self.role = Role.find\_by\_name("Moderator")

end

def add\_admins\_permissions

self.role = Role.find\_by\_name("Admin")

end

def self.password\_valid?(password)

password.present? && password.is\_a?(String) && password.length > 5

end

def generate\_token

self.token = loop do

random\_token = SecureRandom.urlsafe\_base64(nil, false)

break random\_token unless User.exists?(token: random\_token)

end

self.token\_expires\_at = Time.now + 1.day

end

def token\_expired?

self.token\_expires\_at <= Time.now

end

def expire\_token

self.token = Time.now

end

def owns\_product?(product)

product[:id].in?(self.products.pluck(:id))

end

def blocked?

self.is\_blocked

end

def archived?

self.is\_archived

end

def active?

!self.is\_blocked && !self.is\_archived

end

def can\_see\_products?

self.role.can\_see\_products

end

def can\_create\_products?

self.role.can\_create\_products

end

def can\_create\_categories?

self.role.can\_create\_categories

end

def can\_buy\_products?

self.role.can\_buy\_products

end

def can\_comment?

self.role.can\_comment

end

def can\_moderate?

self.role.can\_moderate

end

def can\_chat?

self.role.can\_chat

end

def can\_see\_statistics?

self.role.can\_see\_statistics

end

def can\_create\_moderators?

self.role.can\_create\_moderators

end

def can\_create\_priveleged\_users?

self.role.can\_create\_priveleged\_users

end

end

# **ПРИЛОЖЕНИЕ Г**

**(обязательное)**

**Код программного модуля 20160327154500\_tables.rb**

class Tables < ActiveRecord::Migration

def change

create\_table "categories", force: :cascade do |t|

t.string "name", limit: 50, null: false

t.string "description", limit: 250

t.text "avatar\_link", limit: 65535

t.boolean "is\_archived", default: false

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "categories", ["is\_archived"], name: "IXFX\_is\_archived", using: :btree

create\_table "comments", force: :cascade do |t|

t.integer "rate", limit: 1, null: false

t.string "text", limit: 250

t.integer "product\_id", limit: 4

t.integer "user\_id", limit: 4

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "comments", ["product\_id"], name: "IXFK\_comment\_product", using: :btree

add\_index "comments", ["user\_id"], name: "IXFK\_comment\_user", using: :btree

create\_table "countries", force: :cascade do |t|

t.string "name", limit: 50, null: false

t.string "phone\_code", limit: 50, null: false

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "countries", ["name"], name: "IXFX\_name", using: :btree

add\_index "countries", ["phone\_code"], name: "IXFX\_phone\_code", using: :btree

create\_table "currencies", force: :cascade do |t|

t.string "name", limit: 50, null: false

t.string "code", limit: 5, null: false

t.decimal "rate", precision: 10, scale: 4, null: false

t.datetime "created\_at"

t.datetime "updated\_at"

end

create\_table "deals", force: :cascade do |t|

t.integer "seller\_id", limit: 4

t.integer "buyer\_id", limit: 4

t.integer "product\_id", limit: 4

t.decimal "price", precision: 12, scale: 2, null: false

t.integer "currency\_id", limit: 4

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "deals", ["buyer\_id"], name: "IXFK\_deal\_user\_02", using: :btree

add\_index "deals", ["currency\_id"], name: "FK\_deal\_currency", using: :btree

add\_index "deals", ["product\_id"], name: "IXFK\_deal\_product", using: :btree

add\_index "deals", ["seller\_id"], name: "IXFK\_deal\_user", using: :btree

create\_table "payment\_infos", force: :cascade do |t|

t.string "name", limit: 50, null: false

t.text "data", limit: 65535, null: false

t.integer "currency\_id", limit: 4

t.integer "user\_id", limit: 4

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "payment\_infos", ["currency\_id"], name: "IXFK\_payment\_info\_currency", using: :btree

add\_index "payment\_infos", ["user\_id"], name: "IXFK\_payment\_info\_user", using: :btree

create\_table "products", force: :cascade do |t|

t.string "name", limit: 250, null: false

t.string "description", limit: 500, null: false

t.decimal "price", precision: 12, scale: 2, null: false

t.integer "quantity", limit: 4, null: false

t.integer "rating", limit: 4, default: 0, null: false

t.integer "rates\_count", limit: 4, default: 0, null: false

t.boolean "is\_paused", default: false

t.boolean "is\_archived", default: false

t.boolean "is\_blocked", default: false

t.integer "currency\_id", limit: 4

t.integer "category\_id", limit: 4

t.integer "owner\_id", limit: 4

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "products", ["category\_id"], name: "IXFK\_product\_category", using: :btree

add\_index "products", ["currency\_id"], name: "FK\_product\_currency", using: :btree

add\_index "products", ["is\_archived"], name: "IXFX\_is\_archived", using: :btree

add\_index "products", ["owner\_id"], name: "IXFK\_product\_user", using: :btree

create\_table "properties", force: :cascade do |t|

t.string "name", limit: 50, null: false

t.string "description", limit: 250

t.integer "category\_id", limit: 4, null: false

t.integer "property\_type\_id", limit: 4, null: false

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "properties", ["category\_id"], name: "IXFK\_property\_type\_category", using: :btree

add\_index "properties", ["property\_type\_id"], name: "IXFK\_property\_property\_type", using: :btree

create\_table "property\_parameters", force: :cascade do |t|

t.integer "key", limit: 4

t.text "value", limit: 65535, null: false

t.integer "property\_id", limit: 4

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "property\_parameters", ["property\_id"], name: "IXFK\_property\_parameters\_property\_type", using: :btree

create\_table "property\_types", force: :cascade do |t|

t.string "name", limit: 50, null: false

t.string "description", limit: 250

t.string "display\_name", limit: 50, null: false

t.datetime "created\_at"

t.datetime "updated\_at"

end

create\_table "property\_values", force: :cascade do |t|

t.text "value", limit: 65535, null: false

t.integer "property\_id", limit: 4

t.integer "product\_id", limit: 4

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "property\_values", ["product\_id"], name: "IXFK\_property\_value\_product", using: :btree

add\_index "property\_values", ["property\_id"], name: "IXFK\_property\_value\_property\_type", using: :btree

create\_table "roles", force: :cascade do |t|

t.string "name", limit: 50

t.boolean "can\_see\_products", default: false

t.boolean "can\_create\_products", default: false

t.boolean "can\_create\_categories", default: false

t.boolean "can\_buy\_products", default: false

t.boolean "can\_comment", default: false

t.boolean "can\_moderate", default: false

t.boolean "can\_chat", default: false

t.boolean "can\_see\_statistics", default: false

t.boolean "can\_create\_moderators", default: false

t.boolean "can\_create\_priveleged\_users", default: false

t.datetime "created\_at"

t.datetime "updated\_at"

end

create\_table "transactions", force: :cascade do |t|

t.integer "sender\_id", limit: 4

t.integer "sender\_payment\_info\_id", limit: 4

t.integer "receiver\_id", limit: 4

t.integer "reseiver\_payment\_info\_id", limit: 4

t.decimal "price", precision: 10, scale: 4, null: false

t.integer "deal\_id", limit: 4

t.integer "currency\_id", limit: 4

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "transactions", ["currency\_id"], name: "IXFK\_transactions\_currencies", using: :btree

add\_index "transactions", ["deal\_id"], name: "FK\_transaction\_deal", using: :btree

add\_index "transactions", ["receiver\_id"], name: "FK\_transaction\_user\_02", using: :btree

add\_index "transactions", ["reseiver\_payment\_info\_id"], name: "IXFK\_transactions\_payment\_infos\_02", using: :btree

add\_index "transactions", ["sender\_id"], name: "FK\_transaction\_user", using: :btree

add\_index "transactions", ["sender\_payment\_info\_id"], name: "IXFK\_transactions\_payment\_infos", using: :btree

create\_table "users", force: :cascade do |t|

t.string "email", limit: 254, null: false

t.string "login", limit: 50, null: false

t.string "encrypted\_password", limit: 64, null: false

t.string "first\_name", limit: 50, null: false

t.string "last\_name", limit: 50, null: false

t.string "passport\_id", limit: 50, null: false

t.text "avatar\_url", limit: 65535

t.string "address\_line\_1", limit: 250, null: false

t.string "address\_line\_2", limit: 250

t.string "post\_index", limit: 50

t.string "phone\_number", limit: 50, null: false

t.date "birth\_date", null: false

t.boolean "is\_blocked", default: false, null: false

t.boolean "is\_archived", default: false, null: false

t.integer "role\_id", limit: 4

t.integer "country\_id", limit: 4

t.string "token", limit: 50

t.datetime "token\_expires\_at"

t.datetime "created\_at"

t.datetime "updated\_at"

end

add\_index "users", ["country\_id"], name: "IXFK\_user\_country", using: :btree

add\_index "users", ["email"], name: "IXFK\_email", using: :btree

add\_index "users", ["login"], name: "IXFK\_login", using: :btree

add\_index "users", ["role\_id"], name: "IXFK\_user\_role", using: :btree

add\_index "users", ["token"], name: "IX\_token", using: :btree

add\_foreign\_key "comments", "products", name: "FK\_comment\_product", on\_update: :cascade, on\_delete: :cascade

add\_foreign\_key "comments", "users", name: "FK\_comment\_user", on\_update: :cascade, on\_delete: :nullify

add\_foreign\_key "deals", "currencies", name: "FK\_deal\_currency", on\_update: :cascade

add\_foreign\_key "deals", "products", name: "FK\_deal\_product", on\_update: :cascade, on\_delete: :nullify

add\_foreign\_key "deals", "users", column: "seller\_id", name: "FK\_deal\_user", on\_update: :cascade, on\_delete: :nullify

add\_foreign\_key "payment\_infos", "currencies", name: "FK\_payment\_info\_currency", on\_update: :cascade, on\_delete: :cascade

add\_foreign\_key "payment\_infos", "users", name: "FK\_payment\_info\_user", on\_update: :cascade, on\_delete: :nullify

add\_foreign\_key "products", "categories", name: "FK\_product\_category", on\_update: :cascade, on\_delete: :cascade

add\_foreign\_key "products", "currencies", name: "FK\_product\_currency"

add\_foreign\_key "products", "users", column: "owner\_id", name: "FK\_product\_user", on\_update: :cascade, on\_delete: :cascade

add\_foreign\_key "properties", "categories", name: "FK\_property\_type\_category", on\_update: :cascade, on\_delete: :cascade

add\_foreign\_key "properties", "property\_types", name: "FK\_property\_property\_type", on\_update: :cascade, on\_delete: :cascade

add\_foreign\_key "property\_parameters", "properties", name: "FK\_property\_parameters\_property\_type", on\_update: :cascade, on\_delete: :cascade

add\_foreign\_key "property\_values", "products", name: "FK\_property\_value\_product", on\_update: :cascade, on\_delete: :cascade

add\_foreign\_key "property\_values", "properties", name: "FK\_property\_value\_property\_type", on\_update: :cascade, on\_delete: :cascade

add\_foreign\_key "transactions", "currencies", name: "FK\_transactions\_currencies"

add\_foreign\_key "transactions", "deals", name: "FK\_transaction\_deal", on\_delete: :nullify

add\_foreign\_key "transactions", "payment\_infos", column: "reseiver\_payment\_info\_id", name: "FK\_transactions\_payment\_infos\_02", on\_update: :cascade

add\_foreign\_key "transactions", "payment\_infos", column: "sender\_payment\_info\_id", name: "FK\_transactions\_payment\_infos", on\_update: :cascade

add\_foreign\_key "transactions", "users", column: "receiver\_id", name: "FK\_transaction\_user\_02", on\_delete: :nullify

add\_foreign\_key "transactions", "users", column: "sender\_id", name: "FK\_transaction\_user", on\_update: :cascade, on\_delete: :nullify

add\_foreign\_key "users", "countries", name: "FK\_user\_country", on\_update: :cascade

add\_foreign\_key "users", "roles", name: "FK\_user\_role", on\_update: :cascade

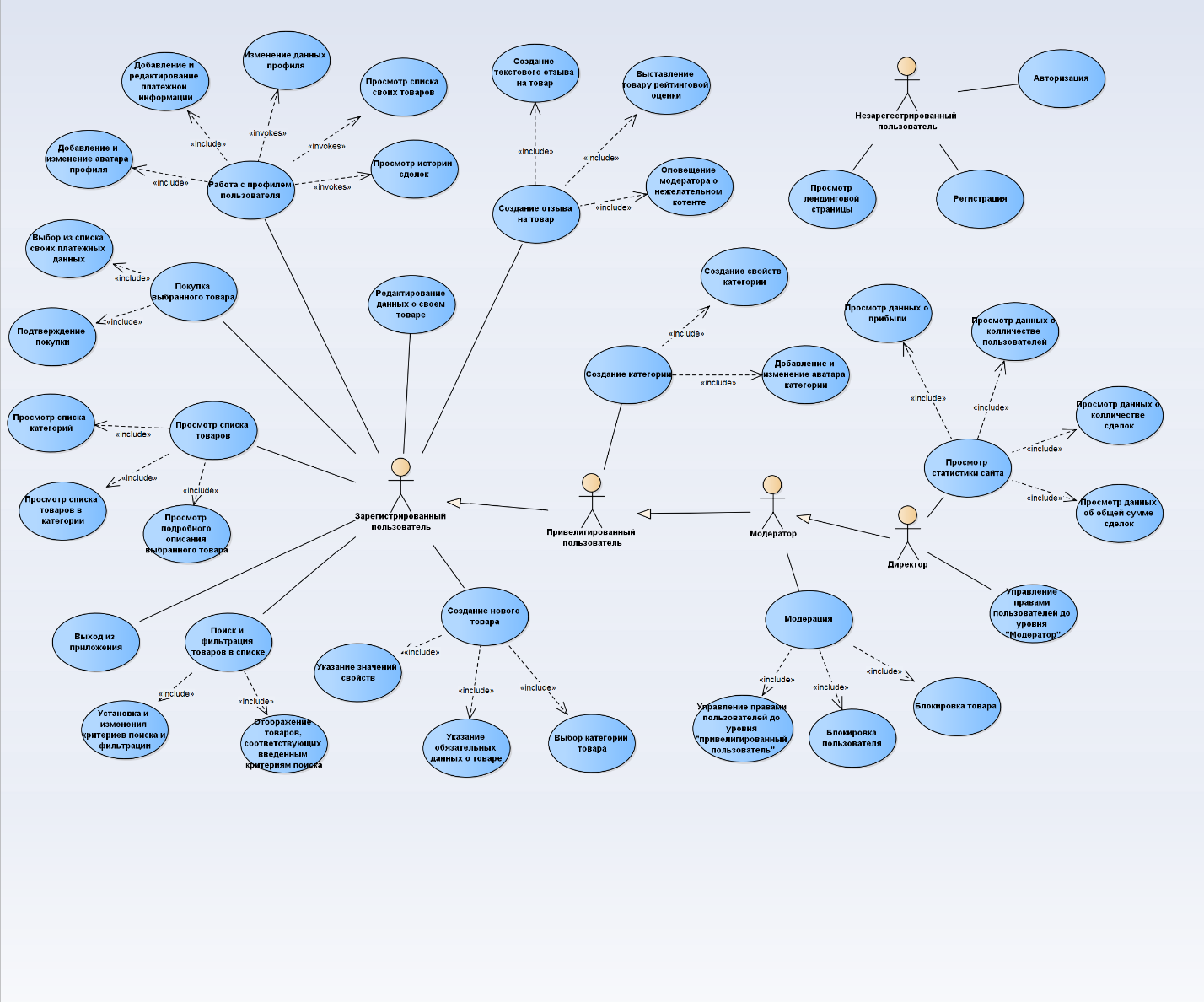
end

end

# **ПРИЛОЖЕНИЕ Д**

**(обязательное)**

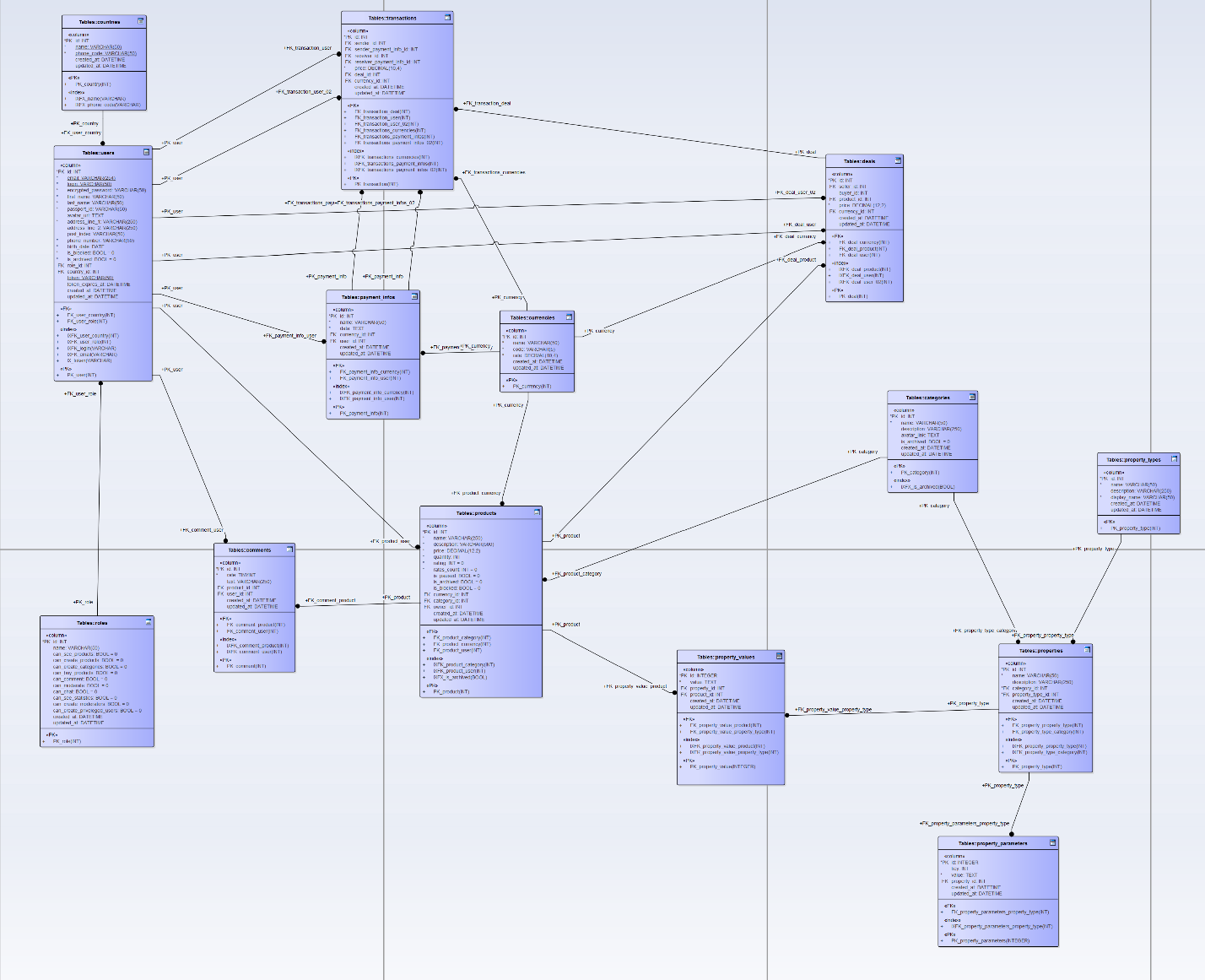
**Диаграмма прецедентов**

****

# **ПРИЛОЖЕНИЕ Е**

**(обязательное)**

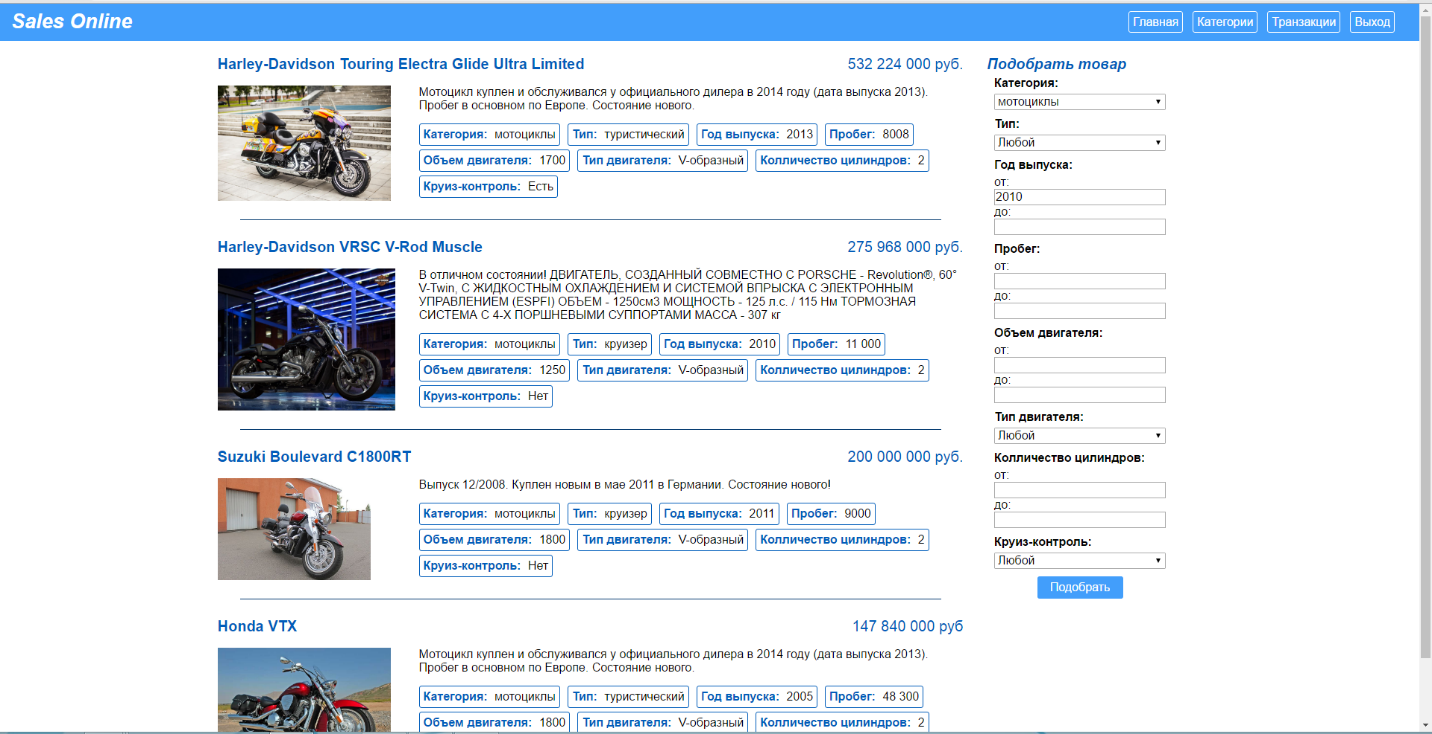
**Схема базы данных**

****

# **ПРИЛОЖЕНИЕ Ж**

**(обязательное)**

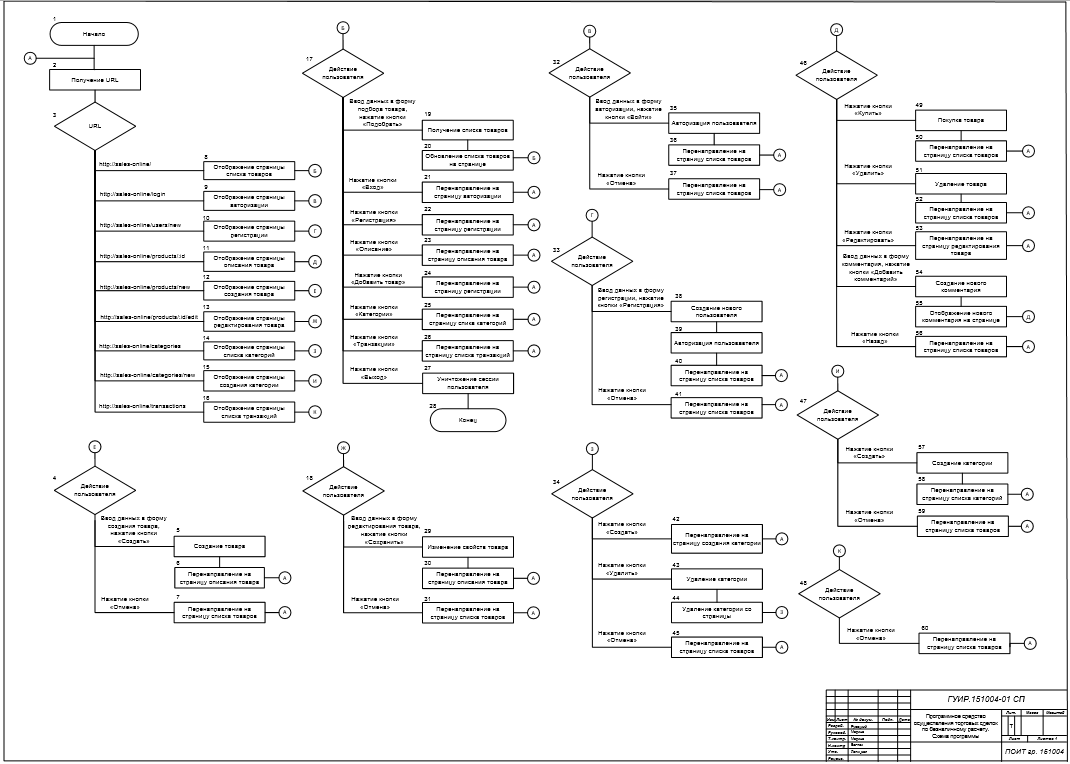
**Внешний вид главной страницы приложения**

****

# **ПРИЛОЖЕНИЕ З**

**(обязательное)**

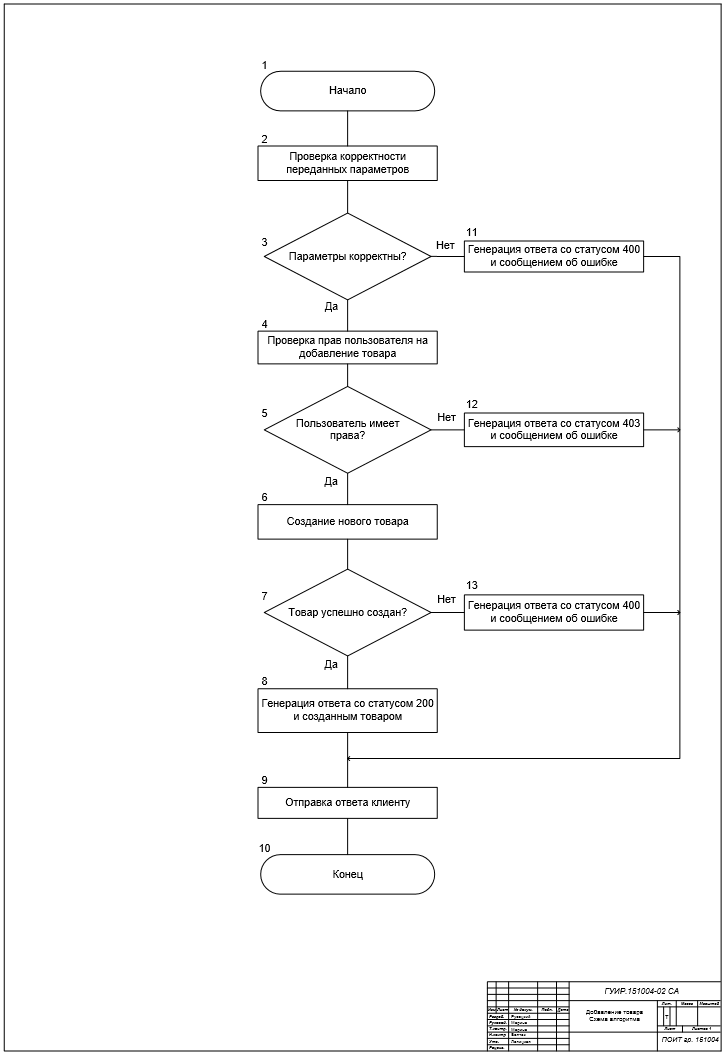
**Схема программы**

****

# **ПРИЛОЖЕНИЕ И**

**(обязательное)**

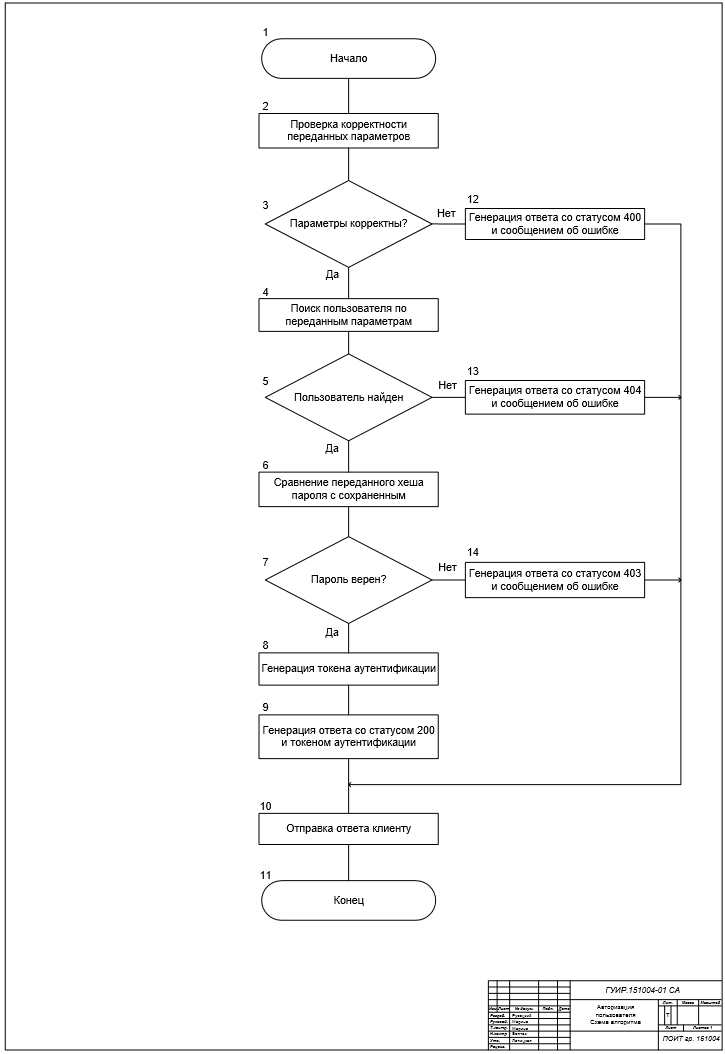
**Схема добавление товара**

****

# **ПРИЛОЖЕНИЕ К**

**(обязательное)**

**Схема авторизации пользователя**

****