

## СОДЕРЖАНИЕ

Введение.....	6
1    Описание предметной области.....	7
2    Постановка задач и обзор методов ее решения .....	9
3    Функциональное моделирование на основе стандарта IDEF0.....	14
3.1  Методология IDEF0.....	14
3.2  Описание разработанной функциональной модели .....	15
4    Информационная модель системы и ее описание .....	20
5    Описание алгоритмов, реализующих бизнес-логику .....	23
6    Руководство пользователя .....	25
6.1  Алгоритм работы регистрации и авторизации .....	25
6.2  Алгоритм работы покупателя.....	25
6.3  Алгоритм работы продавца.....	27
6.4  Алгоритм работы администратора.....	28
7    Архитектурный шаблон проектирования.....	29
7.1  Диаграмма вариантов использования.....	29
7.2  Диаграмма классов.....	30
7.3  Диаграмма последовательностей .....	31
7.4  Диаграмма состояний .....	32
8    Результаты тестирования разработанной системы .....	33
Заключение .....	34
Список использованных источников .....	35
Приложение А (обязательное) Антиплагиат.....	36
Приложение Б (обязательное) Листинг кода.....	37
Приложение В (обязательное) Ведомость документов.....	44

## ВВЕДЕНИЕ

Жизнь современного геймера немислима без интернет-магазинов компьютерных игр. Время, когда игроки бегали на рынки и в уличные магазины за дисковыми изданиями видеоигр, давно минуло, и сегодня абсолютное большинство из них предпочитает приобретать цифровые ключи в интернет-магазинах игр для ПК. Или там же оформлять заказы на физические (чаще всего коллекционные) издания лучших игр. Сегодня это общедоступно, очень удобно и безопасно.

И хотя современные игры являются цифровой продукцией, деньги за них выплачиваются вполне реальные и очень часто довольно немаленькие. По этой причине покупатель хочет быть уверенным, что точно получит желаемый товар, и тот будет надлежащего качества, а в случае каких-либо проблем можно будет без труда вернуть обратно свои деньги.

Создание интернет-магазина, становится все более популярной услугой. По последним данным аудитория в интернете стремительно растет, а продажи через интернет в крупных городах, достигают до 25%, при этом специалисты подчеркивают тенденцию к росту продаж именно через интернет. Сайт интернет-магазина – является современным торговым каналом. С помощью интернет-магазина, можно иметь возможность продавать товары или услуги огромной аудитории, использующей доступ в Интернет. Интернет-магазин для покупателя это: экономия времени, денег и сил. Именно поэтому, по статистике, все больше и больше людей в мире совершает свои покупки через интернет-магазин.

Ежегодно количество интернет-магазинов увеличивается, так как это прибыльно и удобно для всех, так же интернет-магазин экономит бюджет и время. Интернет-магазин работает круглые сутки и может продавать определенные товары в автоматическом режиме без участия продавца. Так же не надо закупать товар заранее, а это существенная экономия, на складских помещениях. Достаточно договориться с поставщиками, и в нужный момент, просто выкупить товар, который у вас закажут. По сравнению с обычным магазином, территория продаж которого ограничивается населением города или района, территория охвата интернет-магазина увеличивается на всю страну и аудиторию в другие страны, ведь товар можно доставлять не только курьерской службой, но и почтой.

Целью проекта является создать интуитивно понятный сайт для пользователей, облегченной системой покупок и добавления игр. Верстка и программирование сайта интернет-магазина с понятным графическим пользовательским интерфейсом, с облегченной системой покупки для пользователей и облегченной страницей добавления игр для продавцов.

## 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Интернет-магазин (англ. online shop или e-shop) – сайт, торгующий товарами в интернете. Позволяет пользователям сформировать заказ на покупку, выбрать способ оплаты и доставки заказа в сети Интернет.

Выбрав необходимые товары или услуги, пользователь обычно имеет возможность тут же на сайте выбрать метод оплаты и доставки. Совокупность отобранных товаров, способ оплаты и доставки представляют собой законченный заказ, который оформляется на сайте путем сообщения минимально необходимой информации о покупателе. Информация о покупателе может храниться в базе данных магазина если бизнес-модель магазина рассчитана на повторные покупки, или же отправляться разово. В интернет-магазинах, рассчитанных на повторные покупки, также ведется отслеживание возвратов посетителя и история покупок. Часто при оформлении заказа предусматривается возможность сообщить некоторые дополнительные пожелания от покупателя продавцу.

Интернет-магазины создаются с применением систем управления контентом сайтов, оснащенных необходимыми модулями. Крупные интернет-магазины работают на специально для них разработанных или адаптированных типовых системах управления. Средние и малые магазины обычно используют типовое коммерческое и свободное ПО. К примеру, широко известен свободный движок osCommerce.

Система управления контентом сайта интернет-магазина может быть коробочным продуктом, самостоятельно устанавливаемым на хостинг-площадку, может быть частной разработкой веб-студии, ей же обслуживаемой, или может быть программным сервисом, предоставляемым с ежемесячной оплатой.

Нужды администраторов интернет-магазина в складском, торговом, бухгалтерском и налоговом учете должны поддерживаться невидимой посетителям частью интернет-магазина – бэк-офисом. Экономически эффективной практикой создания интернет-магазинов является применение специализированных систем учета. Интернет-магазин обычно интегрирован с такими системами учета.

Есть две разновидности интернет-магазинов, в зависимости от вида торговли:

1) Магазины, которые продают товар со своего склада. Такой магазин – прекрасный вариант дополнительного сбыта товара, обычно дают более низкую цену, чем даже в своем реальном магазине; [1]

2) Магазины, которые продают товар других магазинов/людей. Это может быть торговля внутри страны, либо международная торговля. В этом случае интернет-магазин зарабатывает на комиссии, которую платят продавцы за выставление товара. Здесь интернет-магазин выступает гарантом сделки между продавцом и покупателем. Такие магазины используют систему "репутация" продавца. Кроме того, покупатель может пожаловаться админи-

страции сайта на продавца и получить необходимую помощь по возвращению денег, в случае обмана.

Также магазины могут отличаться по способу продажи:

1) Фиксированная цена товара - с доставкой, включенной в стоимость, либо с доставкой, которая считается отдельно, после оформления заказа (очень часто эффект низкой цены бывает испорчен из-за высокой стоимости доставки, продавцы нарочно могут ставить низкую стоимость на товар, а на доставку наоборот - высокую, на чем и зарабатывают).

2) Система аукциона - на товар объявляется аукцион. Кроме начальной цены, продавец может объявлять так называемую блиц-цену – стоимость, за которую продавец готов отдать товар без торга. Есть такой нюанс, как скрытая цена - продавец ставит очень низкую цену на товар (чтобы при поиске товара, клиент заметил именно его лот), но включает опцию "минимальная ставка" – это минимальная цена, которая скрыта от глаз покупателя, и он должен повышать ставки, пока не достигнет ее, иначе ставка не будет принята.

Существует множество информационных систем для создания интернет-магазина. Одна из известных бесплатных систем – движок osCommerce.

Основные возможности этого движка:

- совместимо с PHP 4.x, 5.x и MySQL 4.x, 5.x;
- совместимость со всеми основными браузерами;
- встроенная многоязычность, по умолчанию установлены английский, немецкий, испанский языки. Доступны русский, украинский и многие другие;

- администрирование / База;
- поддерживает неограниченное количество продуктов и разделов категорий;

- поддержка физических и виртуальных (загружаемых) товаров;
- легкость резервного копирования и восстановления данных;
- статистика товаров и заказчиков;
- многоязычная поддержка;
- поддержка нескольких валют.

Клиентская часть:

- регистрация покупателей;
- все заказы хранятся в базе данных для быстрого и эффективного поиска (история покупок для покупателей);
- клиенты могут просматривать историю и статусы своих заказов;
- временная корзина для гостей и постоянная для клиентов;
- быстрый и дружелюбный интерфейс поиска;
- удобная навигация по сайту;
- система оплаты и доставки;
- настройка методов оплаты для разных областей;
- расчет доставки на основе веса и цены товара, зоны доставки. Множество модулей расчета доставки.

## 2 ПОСТАНОВКА ЗАДАЧ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ

В курсовом проекте необходимо автоматизировать работу, продажу и анализ добавленных игр.

Для реализации данного проекта необходимо выполнить следующие задачи:

- изучить предметную область;
- реализовать клиент-серверное приложение;
- разработать базу данных MySQL;
- связать базу данных с контроллерами;
- создать удобный интерфейс для продавца и покупателя;
- распределить роли (администратор, продавец и покупатель);
- разработать собственную иерархию классов;
- реализовать не менее двух паттернов проектирования;
- использовать сокрытие данных, перегрузку методов, переопределение методов;
- предусмотреть обработку исключительных ситуаций;
- предоставить продавцам и администраторам аналитическую информацию в виде статистики.

Общие возможности всех ролей:

- регистрация;
- просмотр каталога;
- пользоваться поисковиком;
- добавление комментария к играм.

Со стороны администратора программа должна предусматривать следующие возможности:

- редактирование и удаление пользователей;
- добавление, редактирование удаление всех игр;
- просмотр статистики продаж своих игр.

Продавец должен иметь возможность реализовать следующие задачи:

- работа с корзиной;
- покупка игр;
- скачивание купленных игр;
- добавление, редактирование удаление только своих игр;
- просмотр статистики продаж своих игр.

Пользователь имеет следующие возможности:

- покупка игры;
- работа с корзиной;
- скачивание купленных игр.

Для решения поставленных задач используются MySQL-сервер, IntelliJ IDEA, AllFusion ERwin Data Modeler, Spring Boot, Spring MVC, Spring Security, Thymeleaf, Maven, язык программирования Java.

**MySQL.** Свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, кото-

рая 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.

Преимущества:

- легко использовать;
- предоставляет большой функционал;
- хорошие функции безопасности;
- легко масштабируется и подходит для больших баз данных;
- обеспечивает хорошую скорость и производительность;
- обеспечивает хорошее управление пользователями и множественный

контроль доступа.

**IntelliJ IDEA.** Интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains. Первая версия появилась в январе 2001 года и быстро приобрела популярность как первая среда для Java с широким набором интегрированных инструментов для рефакторинга, которые позволяли программистам быстро реорганизовывать исходные тексты программ.

Преимущества:

- автодополнение кода и качественная отладка;
- удобная навигация;
- безопасный рефакторинг – применить изменения во всем проекте можно за пару кликов;
- функция Live Edit позволяет мгновенно посмотреть все изменения в браузере;
- интерфейс будет понятен даже новичкам.

**AllFusion ERwin Data Modeler.** Это компьютерная программа для проектирования и документирования баз данных. Модели данных помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких аспектов деятельности предприятия, как уровень сложности данных, технологий баз данных и среды развертывания. Изначально разработанный компанией Logic Works, erwin был приобретен рядом компаний, прежде чем был выделен частной инвестиционной фирмой Parallax Capital Partners, которая приобрела и объединила его в качестве отдельной организации, erwin, Inc., с генеральным директором Адамом Famularo.

Преимущества:

- поддержка стандартной нотации IDEF1x для ER-диаграмм моделей данных, нотации IE и специальной нотации, предназначенной для проектирования хранилищ данных – Dimensional;
- поддержка проектирования информационных хранилищ (на основе Red Brick и Teradata);
- поддержка совместного проектирования (версия для ModelMart);

- поддержка триггеров, хранимых процедур и шаблонов;
- развитые средства проверки корректности моделей данных Reverse Engineering (генерация модели данных на основе анализа существующей базы данных), включая восстановление связей по индексам;
- автоматическая генерация SQL DDL для создания баз данных;
- полная совместимость и поддержка 20-ти типов СУБД на основе прямого доступа к системному каталогу баз данных (отпадает потребность в использовании ODBC).

**Spring Framework.** Универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET. Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «Expert One-on-One Java EE Design and Development».

Основное преимущество Spring – возможность разработки приложения как набора слабосвязанных (loose-coupled) компонентов. Чем меньше компоненты приложения знают друг о друге, тем проще разрабатывать новый и поддерживать существующий функционал приложения. Классический пример – управление транзакциями. Spring позволяет вам управлять транзакциями совершенно независимо от основной логики взаимодействия с БД. Изменение этой логики не порушит транзакционность, равно как изменение логики управления транзакциями не сломает логику программы. Spring поощряет модульность. Компоненты можно добавлять и удалять (почти) независимо друг от друга. В принципе, приложение можно разработать таким образом, что оно даже не будет знать, что управляется Spring. Также Spring заметно упрощает модульное тестирование (unit-testing): в компонент, разработанный для работы в IoC контейнере очень легко инжектировать фейковые зависимости и проверить работу только этого компонента. Ну, и в качестве приятного дополнения, Spring сильно облегчает инициализацию и настройку компонентов приложения, позволяя гибко настраивать приложение без существенных изменений Java-кода.

**Паттерны проектирования** (шаблоны проектирования) – это готовые к использованию решения часто возникающих в программировании задач. Это не класс и не библиотека, которую можно подключить к проекту, это нечто большее. Паттерны проектирования, подходящий под задачу, реализуется в каждом конкретном случае. Следует, помнить, что такой паттерн, будучи примененным неправильно или к неподходящей задаче, может принести немало проблем. Тем не менее, правильно примененный паттерн поможет решить задачу легко и просто.

Типы паттернов:

- порождающие паттерны;
- структурные паттерны;
- шаблонные паттерны.

**Порождающие паттерны.** Порождающие паттерны предоставляют механизмы инициализации, позволяя создавать объекты удобным способом.

Abstract Factory – позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

Builder – позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов.

Factory Method – определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

Prototype – позволяет копировать объекты, не вдаваясь в подробности их реализации.

Singleton – гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

**Структурные паттерны.** Структурные паттерны определяют отношения между классами и объектами, позволяя им работать совместно.

Adapter – позволяет объектам с несовместимыми интерфейсами работать вместе.

Bridge – разделяет один или несколько классов на две отдельные иерархии – абстракцию и реализацию, позволяя изменять их независимо друг от друга.

Composite – позволяет сгруппировать объекты в древовидную структуру, а затем работать с ними так, как будто это единичный объект.

Decorator – позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».

Facade – предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку.

Flyweight – позволяет вместить большее количество объектов в отведённую оперативную память. Легковес экономит память, разделяя общее состояние объектов между собой, вместо хранения одинаковых данных в каждом объекте.

Proxy – позволяет подставлять вместо реальных объектов специальные объекты-заменители. Эти объекты перехватывают вызовы к оригинальному объекту, позволяя сделать что-то до или после передачи вызова оригиналу.

**Поведенческие паттерны.** Поведенческие паттерны используются для того, чтобы упростить взаимодействие между сущностями.

Chain of Responsibility – позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи.

Command – превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций.

Iterator – даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления.

Mediator – позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс-посредник.



Memento – позволяет делать снимки состояния объектов, не раскрывая подробностей их реализации. Затем снимки можно использовать, чтобы восстановить прошлое состояние объектов.

Observer – создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

State – позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.

Strategy – определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы.

Template Method – определяет скелет алгоритма, переключая ответственность за некоторые его шаги на подклассы. Паттерн позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.

Visitor – позволяет создавать новые операции, не меняя классы объектов, над которыми эти операции могут выполняться.

Chain of Responsibility – позволяет избежать жесткой зависимости отправителя запроса от его получателя, при этом запрос может быть обработан несколькими объектами.

Interpreter – определяет грамматику простого языка для проблемной области.

Разумное использование паттернов проектирования приводит к повышению надежности обслуживания кода, поскольку в дополнение к тому, чтобы быть хорошим решением общей проблемы, паттерны проектирования могут быть распознаны другими разработчиками, что уменьшает время при работе с определенным кодом.

### **3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0**

#### **3.1 Методология IDEF0**

IDEF0 – метод функционального моделирования, а также графическая нотация, которая используется для описания и формализации бизнес-процессов. Особенность IDEF0 заключается в том, что эта методология ориентирована на соподчиненность объектов. IDEF0 была разработана для автоматизации предприятий еще в 1981 году в США.

**Функциональная модель компании.** IDEF0 – метод функционального моделирования, а также графическая нотация, которая используется для описания и формализации бизнес-процессов. Особенность IDEF0 заключается в том, что эта методология ориентирована на соподчиненность объектов. IDEF0 была разработана для автоматизации предприятий еще в 1981 году в США.

##### **Типы стрелок.**

Входящими ставятся задачи.

Исходящими выводятся результат деятельности.

Управляющие (стрелки сверху вниз) – это механизмы управления.

Механизмы (стрелки снизу вверх) используются для проведения необходимых работ.

При работе с функциональной моделью приняты следующие правила. К примеру, стрелки получают названия именами существительными (правила, план и т.д.), блоки – глаголами (провести учет, заключить договор).

IDEF0 позволяет обмениваться информацией, при этом благодаря универсальности и наглядности участники обмена легко поймут друг друга. IDEF0 тщательно разрабатывался и совершенствовался, работать с IDEF0 можно с помощью различных инструментов, к примеру, ERWIN, VISIO, Bussines studio.

У IDEF0 есть еще одно неоспоримое преимущество. Эта методика была разработана сравнительно давно, и за три десятилетия она прошла тщательную шлифовку и корректировку. Поэтому создать функциональную модель компании можно быстро и минимальной вероятностью ошибки.

Естественно, есть и другие методологии, так почему мы рекомендуем именно IDEF0? Отпилить кусок металлической трубы можно и ножовкой, но, согласитесь, сделать это гораздо проще с помощью болгарки. Так и с IDEF0: нет более функционального инструмента для моделирования, с ним получить вы легко и быстро получите нужный вам результат.

##### **Стандарт IDEF0 и его требования.**

О базовых требованиях IDEF0 мы говорили чуть выше.

– главный элемент – в верхнем левом углу;

– каждый элемент должен иметь входящие и исходящие стрелки. Причем входящие стрелки находятся слева, справа – исходящие;

– сверху располагаются управляющие элементы, снизу – механизмы;

- при расположении нескольких блоков на одном листе или экране последующие размещаются справа внизу от предыдущего;
- схемы следует создавать так, чтобы стрелки пересекались минимальное количество раз.

Естественно, в стандарте IDEF0 есть общепринятые нормы, требования и обозначения. Подробно на них останавливаться не будем, при желании эту информацию несложно найти.

### 3.2 Описание разработанной функциональной модели

На диаграмме IDEF0 представлено описание процесса управления финансами предприятия. На рисунке 3.2.1 представлена контекстная диаграмма системы.



Рисунок 3.2.1 – Контекстная диаграмма системы

Входные данные: информация о пользователях и информация о игре. Результатом является ссылка на игру, для дальнейшего его скачивания. Механизмами являются: администратор, продавец и покупатель. Управлениями будут база данных клиентов, база данных игр, закон защиты авторских прав.

На следующем рисунке 3.2.2 представлена декомпозиция работы «Электронная коммерческая деятельность и разработка интернет-магазина».

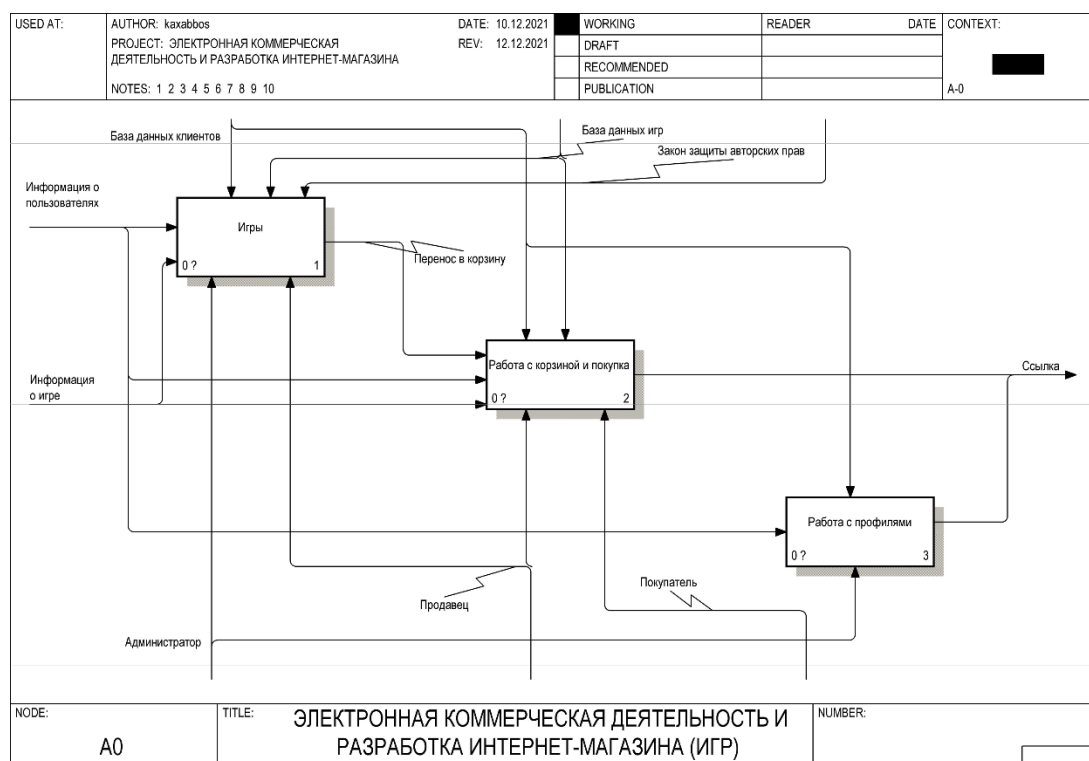


Рисунок 3.2.2 – Декомпозиция работы «Электронная коммерческая деятельность и разработка интернет-магазина»

Данная декомпозиция разбита на 3 блока:

- игры;
- работа с корзиной и покупки;
- работа с профилями.

Стрелки управления:

- «база данных клиентов» входит во все блоки;
- «база данных игр» входит в первые два блока;
- «закон защиты авторских прав» входит в первый блок.

Стрелки ввода:

- «информация о пользователях» входит во все блоки;
- «информация о игре» входит в первые два блока.

Стрелки механизма:

- «администратор» входит в первый и последний блок;
- «продавец» входит в первые два блока;
- «покупатель» входит в блок «работа с корзиной и покупки».

Стрелка вывода «ссылка» выходит из последних двух блоков.

В блоке «игры» происходит работа с играми со стороны администратора и продавца: добавление, редактирование и удаление игр. Выводом из блока является готовая игра, которую можно перенести в корзину или купить.

В блоке «работа с корзиной и покупки» идет процесс покупки игры и основная бизнес-логика текущего курсового проекта.

В блоке «работа с профилями» идет процесс редактирования профилей со стороны администратора.

На следующем рисунке 3.2.3 показана декомпозиция работы «Игры».

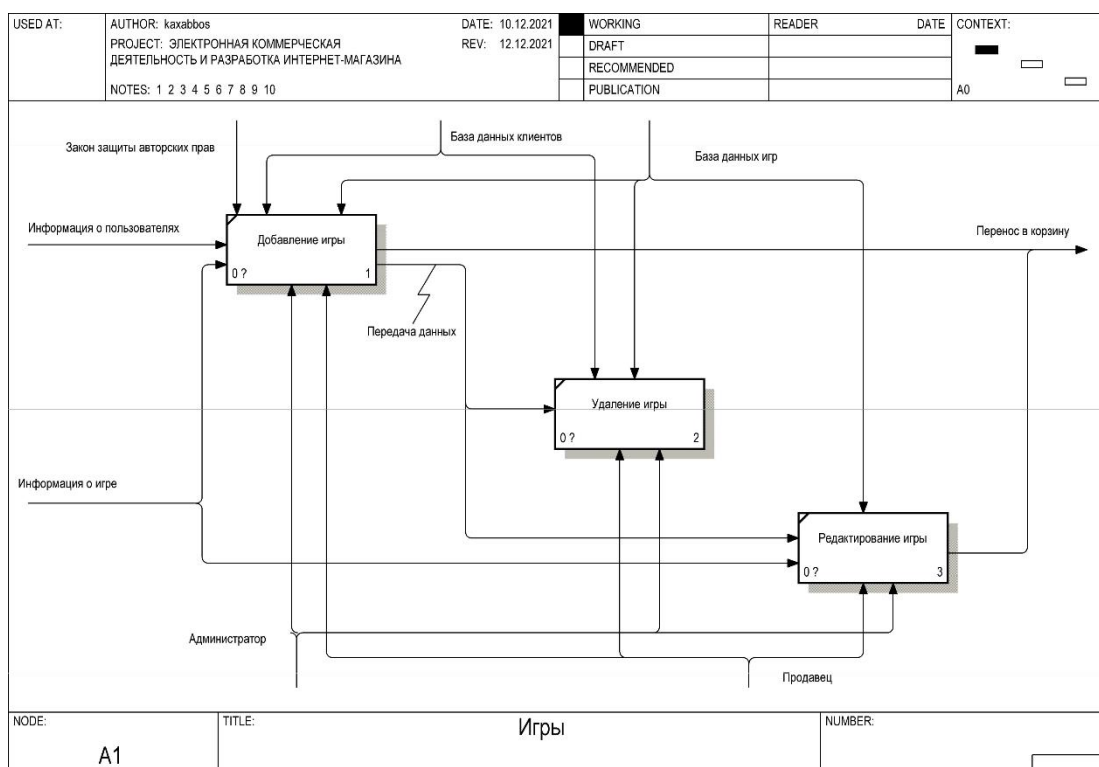


Рисунок 3.2.3 – Декомпозиция работы «Игры»

Данная декомпозиция разбита на 3 блока:

- добавление игры;
- удаление игры;
- редактирование игры.

Стрелки управления:

- «база данных клиентов» входит в первые два блока;
- «база данных игр» входит во все блоки;
- «закон защиты авторских прав» входит в первый блок.

Стрелки ввода:

- «информация о пользователях» входит в первый блок;
- «информация о игре» входит в первый и последний блок.

Стрелки механизма «администратор» и «продавец» входит во все блоки.

Стрелка вывода «ссылка» выходит из первого и последнего блока.

В блоке «добавление игры» происходит добавление игры в базу данных учитывая закон защиты авторских прав. Из блока идет вывод «передача данных» в остальные блоки и «перенос в корзину» для возможности дальнейшей покупки и получения ссылки.

В блоке «удаление игры» происходит удаление игры, удаляются все данные с базы данных игр и данные у пользователей из корзины и купленные.

В блоке «редактирование игры» происходит редактирование игры изменяя ее в базе данных. Из блока идет вывод «перенос в корзину» для возможности дальнейшей покупки и получения ссылки.

На следующем рисунке 3.2.4 показана декомпозиция работы «Работа с корзиной и покупка».

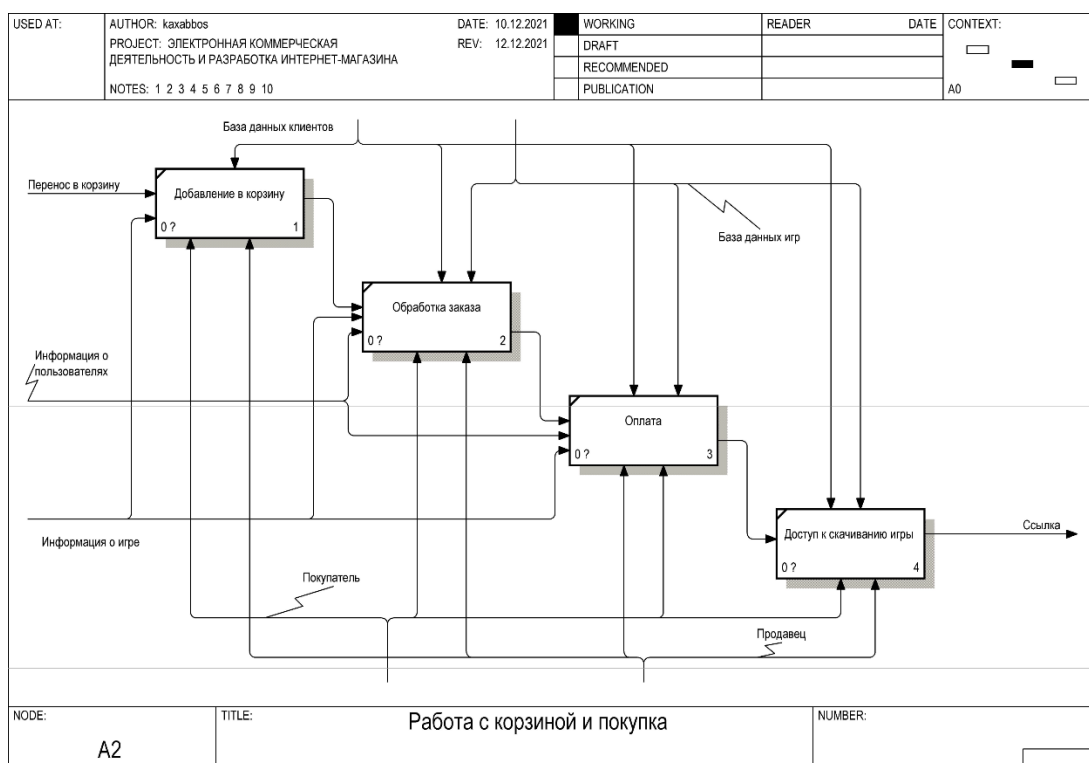


Рисунок 3.2.4 – Декомпозиция работы «Работа с корзиной и покупка»

Данная декомпозиция разбита на 4 блока:

- добавление в корзину;
- обработка заказа;
- оплата;
- доступ к скачиванию игры.

Стрелки управления:

- «база данных клиентов» входит во все блоки;
- «база данных игр» входит в последние три блока.

Стрелки ввода:

- «перенос в корзину» входит в первый блок;
- «информация о пользователях» входит во второй и третий блок;
- «информация о игре» входит в первые три блока.

Стрелки механизма «продавец» и «покупатель» входит во все блоки.

Стрелка вывода «ссылка» выходит из последнего блока.

На следующем рисунке 3.2.5 показана декомпозиция работы «Работа с профилями».

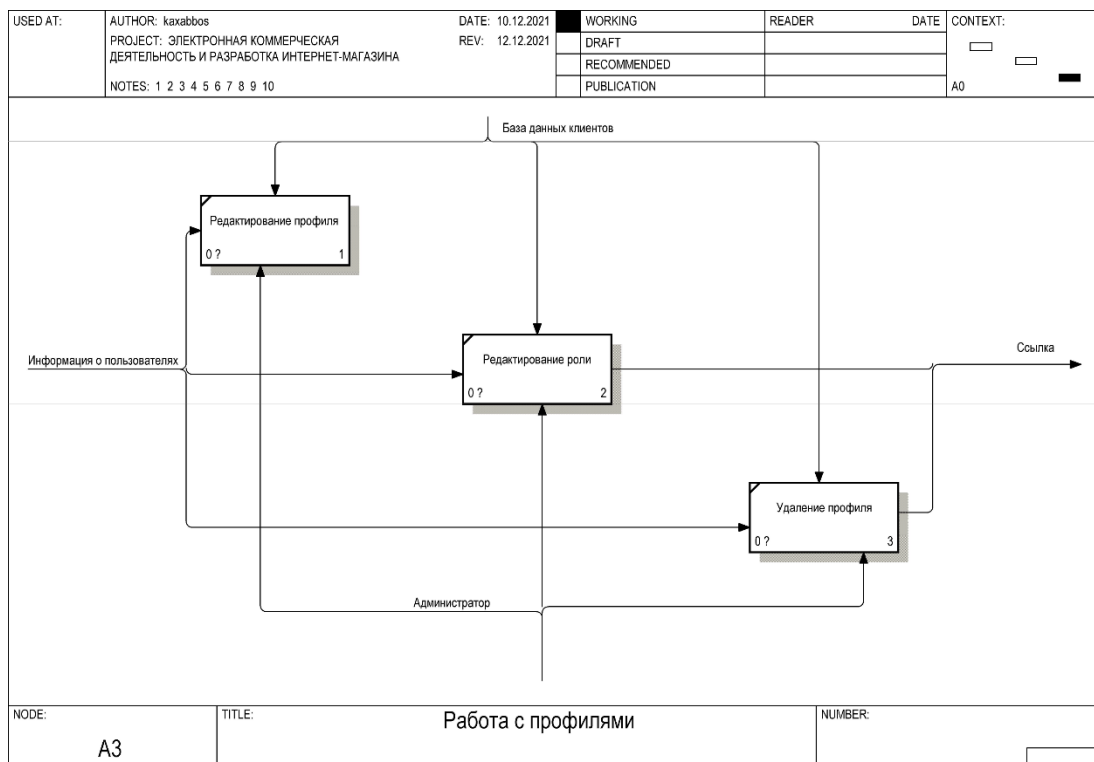


Рисунок 3.2.5 – Декомпозиция работы «Работа с профилями»

Данная декомпозиция разбита на 3 блока:

- редактирование профиля;
- редактирование роли;
- удаление профиля.

Стрелка управления «база данных клиентов» входит во все блоки.

Стрелка ввода «информация о пользователях» входит во все блоки.

Стрелки механизма «администратор» входит во все блоки.

Стрелка вывода «ссылка» выходит из последних двух блоков.

В блоке «Редактирование профиля» происходит изменения логина и пароля в базе данных пользователей со стороны администратора.

В блоке «Редактирование роли» происходит изменение роли в базе данных клиентов, и в зависимости от изменения роли изменяются игры в базе данных, например, при изменении роли с продавца на покупателя, то все игры, добавленные продавцом до этого момента будут удалены.

## **4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ**

Компьютерная информационная система (ИС) – система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию (ISO/IEC 2382:2015). Предназначена для своевременного обеспечения надлежащих людей надлежащей информацией, то есть для удовлетворения конкретных информационных потребностей в рамках определённой предметной области, при этом результатом функционирования компьютерных информационных систем является информационная продукция – документы, информационные массивы, базы данных и информационные услуги.

Современное понимание информационной системы предполагает использование в качестве основного технического средства переработки информации персонального компьютера (сервера, периферийного оборудования и т.д.).

Необходимо понимать разницу между компьютерами и информационными системами. Компьютеры, оснащенные специализированными программными средствами, являются технической базой и инструментом для информационных систем. Информационная система немыслима без персонала, взаимодействующего с компьютерами и телекоммуникациями.

Говоря об информационной системе, следует рассмотреть следующие вопросы: структура ИС, классификации ИС.

Структура ИС обычно рассматривается как совокупность различных подсистем. Все подсистемы можно рассматривать как по отдельности, так и во взаимосвязи друг с другом.

Для обеспечения минимальной избыточности и физического объёма данных, а также для более быстрого доступа, модель приведена к 3 нормальной форме.

Для моделирования системы существует 3 этапа проектирования:

- концептуальное проектирование;
- логическое проектирование;
- физическое проектирование.

Концептуальное проектирование представляет собой начальный этап разработки программного обеспечения (ПО), когда определяется базовая структура информационной системы (ИС), ее компоненты, их назначение и взаимосвязь.

Следующим этапом является логическое проектирование, оно представляет собой описание логической структуры данных посредством системы управления базами данных (СУБД), для которой проектируется БД (см. рис. 4.1).



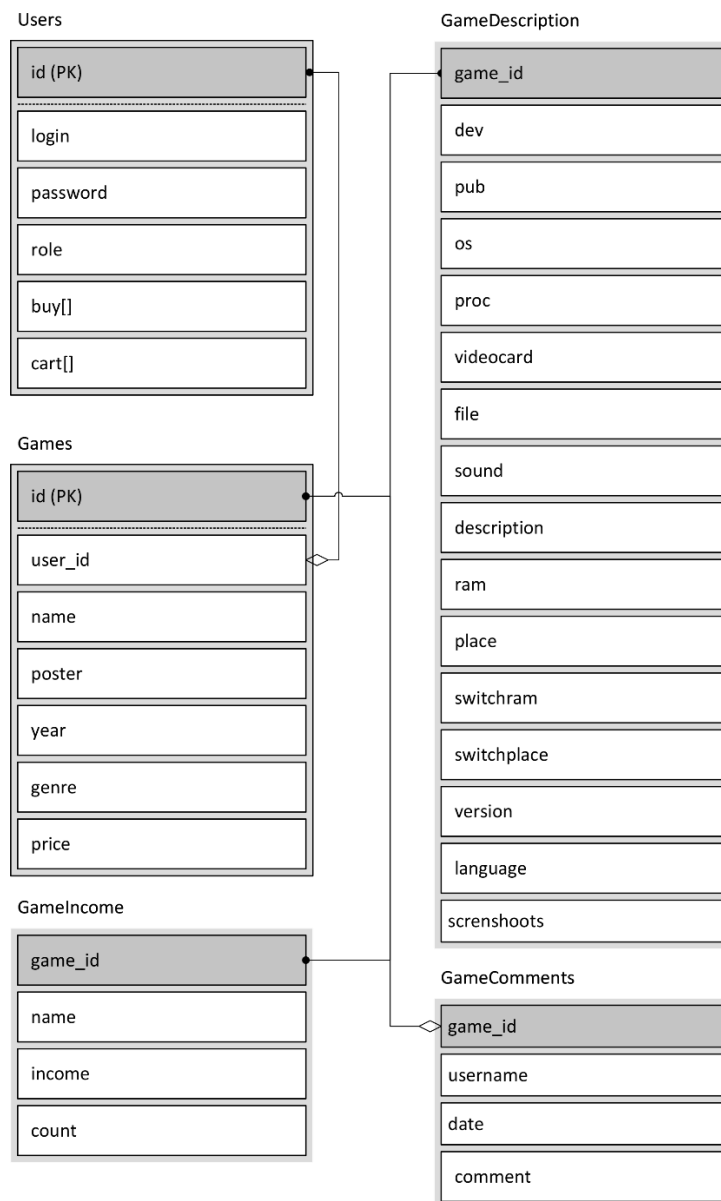


Рисунок 4.1 – Логическая модель системы

Логическая модель системы разработана с помощью редактора Microsoft Visio. Система содержит 5 сущностей: пользователи (Users), игры (Games), описание игры (GameDescription), Комментарии к игре (GameComments), Статистика продаж (GameIncome).

Сущность – это любой различимый объект, о котором необходимо хранить информацию в базе данных. Каждая сущность содержит атрибуты. Они отображают качества и свойства объекта.

Сущность «пользователи» содержит 6 атрибута: id – уникальный идентификационный номер (отдельно от игр), логин, пароль, роль, два массива cart (корзина) и buy (купленные) в которых находятся id игр.

Сущность «игры» содержит 7 атрибутов: id – уникальный идентификационный номер (отдельно от пользователей), имя игры, картину постер, год выпуска, жанр и цена.

Сущность «комментарии» содержит 4 атрибута: id игры, имя пользователь, дата и комментарий. Сущность связана с сущностью «игра».

Сущность «статистика» содержит 4 атрибута: id игры, название игры, суммарный заработок и количество проданных копий. Сущность связана с

Сущность «описание» содержит 16 атрибутов: id игры, разработчик, издатель, операционная система, процессор, видеокарта, ссылка на игру, звук, описание, размер оперативной памяти, занимаемое место после установки, выбор между гб и мб для оперативной памяти, выбор между гб и мб для необходимого места, версия, язык, скриншоты игры.

Этап физического проектирования. Это описание физической структуры базы данных. Физическая модель системы создана при помощи программы Microsoft Visio (см. рис. 4.2).

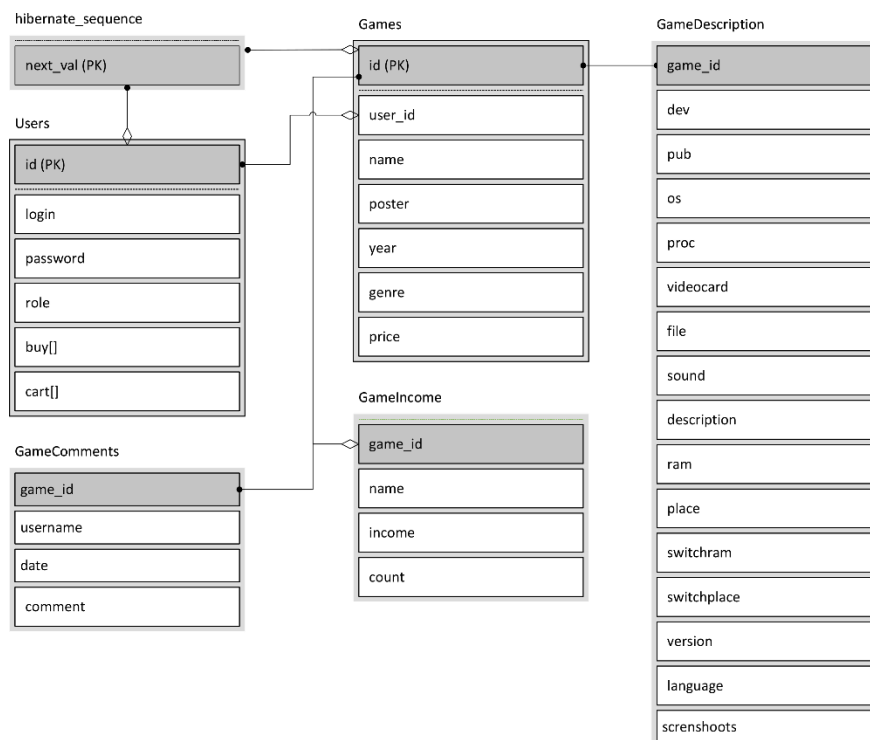


Рисунок 4.2 – Физическая модель системы

База данных была приведена к третьей нормальной форме, была добавлена дополнительная сущность «hibernate\_sequence» в котором хранится следующий id для нового пользователя или игры и у каждой таблицы имеется всего один первичный ключ, а каждое не ключевое поле не транзитивно зависит от первичного ключа, то есть изменив значение в одном столбце, не потребуется изменение в другом столбце.

## 5 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОЕКТИРУЕМОЙ СИСТЕМЫ

Рассмотрим алгоритм работы купли-продажи игр (см. рисунок 5.1).

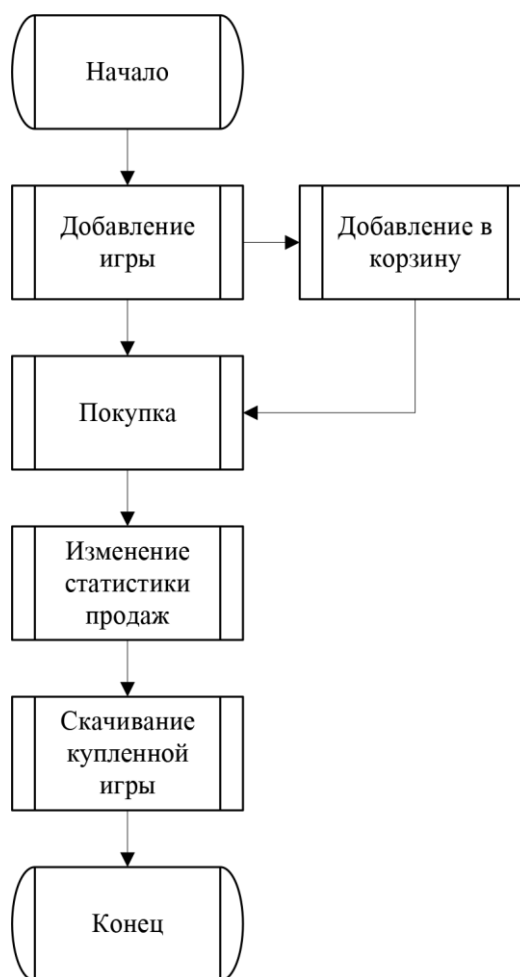


Рисунок 5.1 – Алгоритм купли-продажи игр

На первом администратор или продавец добавляет игру в каталог для дальнейшего получения выгоды.

Далее покупатель или продавец (не владелец) покупает игру или добавляет ее в корзину.

Основным этапом является алгоритм добавления ИД игр в корзину или в купленные игры пользователя.

В случае удачной покупки из корзины, ИД из корзины Пользователя переходит в купленные игры.

У продавца данной игры изменяется статистика о продаже данной игры.

Пользователь, успешно совершивший покупку игры, может зайти в страницу с купленной игрой и скачать ее.

На следующем рисунке 5.2 показано алгоритма регистрации.

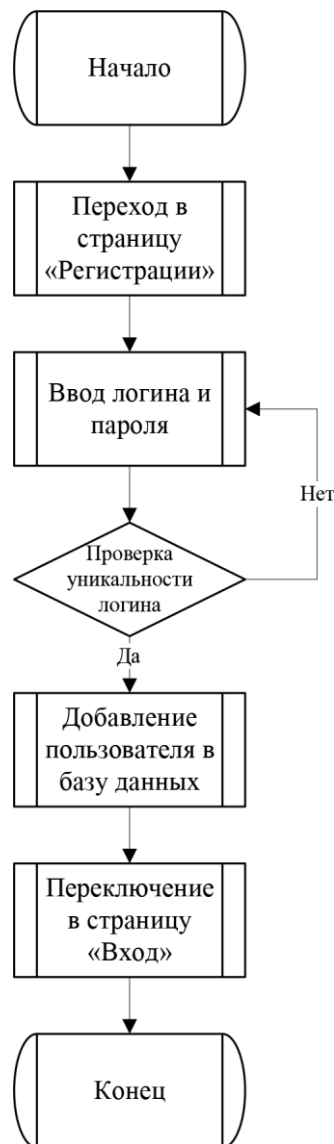


Рисунок 5.2 – Алгоритм регистрации

Сперва новый пользователь должен перейти в страницу «Регистрации» через верхнюю панель или через кнопку «Регистрация» в странице «Вход».

Новый пользователь вводит данные (логин и пароль) и нажимает на кнопку «Регистрация».

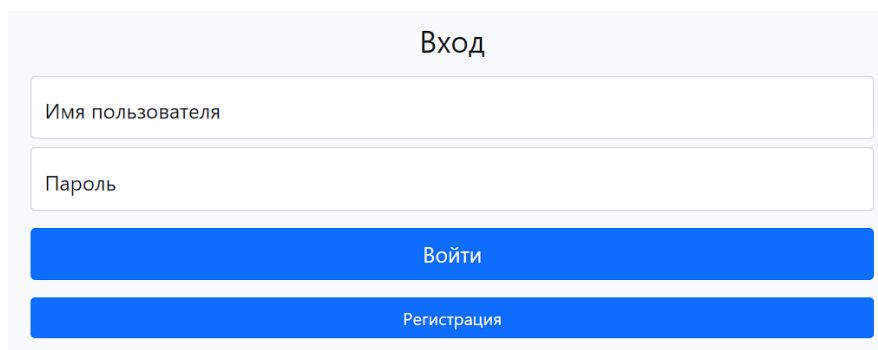
Далее идет проверка логина на уникальность, при не прохождении проверки пользователя перекинет в страницу «Регистрация», при успешной проверки данные сохраняются в базе данных и пользователь переправляется в страницу «Вход».

## 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данное сайт интернет-магазин представляет собой электронная коммерческая деятельность и разработка интернет-магазина.

### 6.1 Алгоритм работы регистрации и авторизации

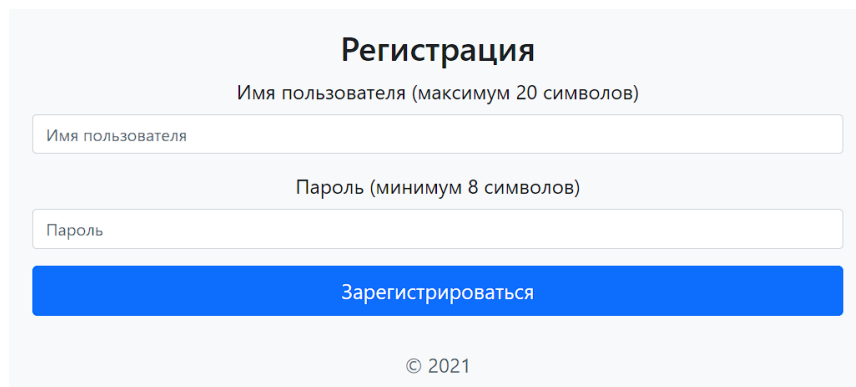
При переходе на страницу «Входа», пользователь должен ввести логин и пароль (см. рис. 6.1.1), или перейти на регистрацию. При добавление нового пользователя ему присваивается роль покупателя, только пользователи с ролью админа могут изменять роли другим пользователям.



The screenshot shows a login form with the title 'Вход' at the top. It contains two input fields: 'Имя пользователя' (Username) and 'Пароль' (Password). Below these fields are two blue buttons: 'Войти' (Login) and 'Регистрация' (Registration).

Рисунок 6.1.1 – Страница «Входа»

При регистрации нового пользователя, длина логина должна быть не больше 20 символов и длина пароля не меньше 8 символов (см. рис. 6.1.2).



The screenshot shows a registration form with the title 'Регистрация'. It includes two input fields with labels: 'Имя пользователя (максимум 20 символов)' (Username (maximum 20 symbols)) and 'Пароль (минимум 8 символов)' (Password (minimum 8 symbols)). Below the fields is a blue button labeled 'Зарегистрироваться' (Register). At the bottom of the form, there is a copyright notice '© 2021'.

Рисунок 6.1.2 – Страница «Регистрации»

В случае успешной регистрации логин не должен совпадать с уже зарегистрированными пользователями, пользователь будет перенаправлен в страницу входа в случае успешной регистрации.

### 6.2 Алгоритм работы покупателя

Покупатель имеет возможность работать с собственной корзиной (см. рис. 6.2.1), просматривать каталог (см. рис. 6.2.2), искать игры, покупать игры (см. рис. 6.2.3) и скачивание после успешной покупки (см. рис. 6.2.4).

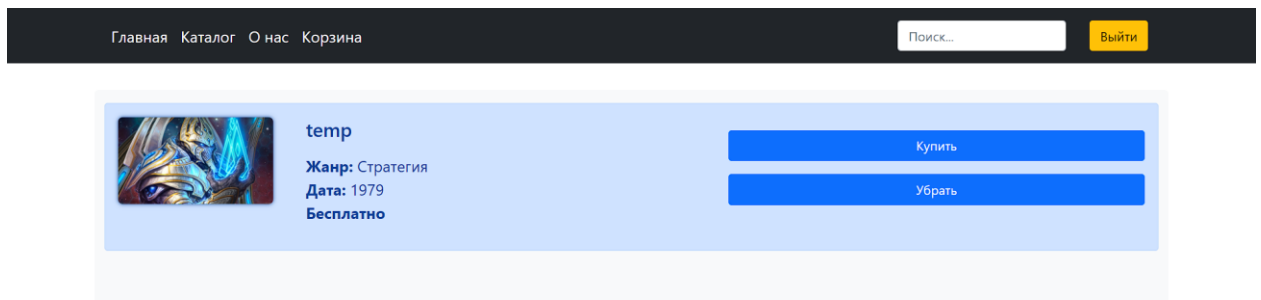


Рисунок 6.2.1 – Работа с корзиной

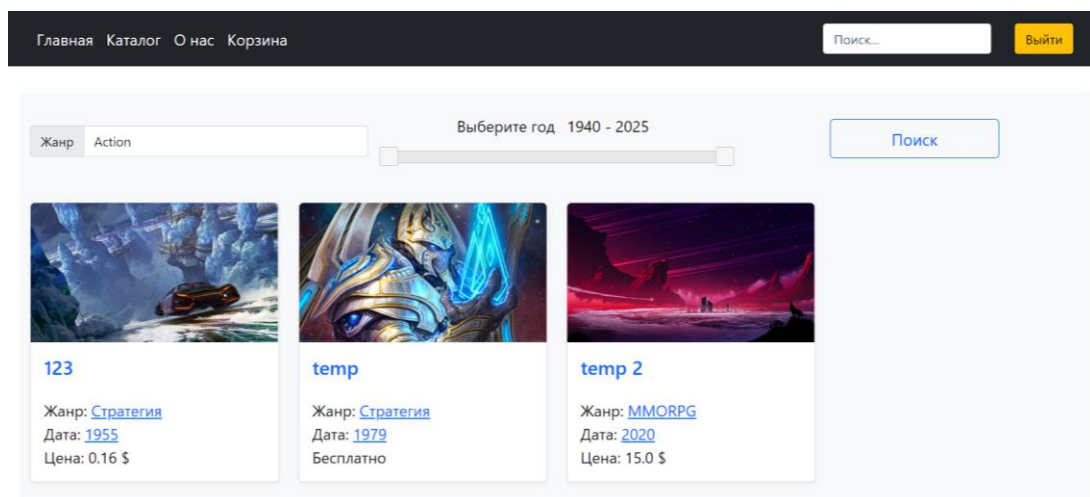


Рисунок 6.2.2 – Каталог

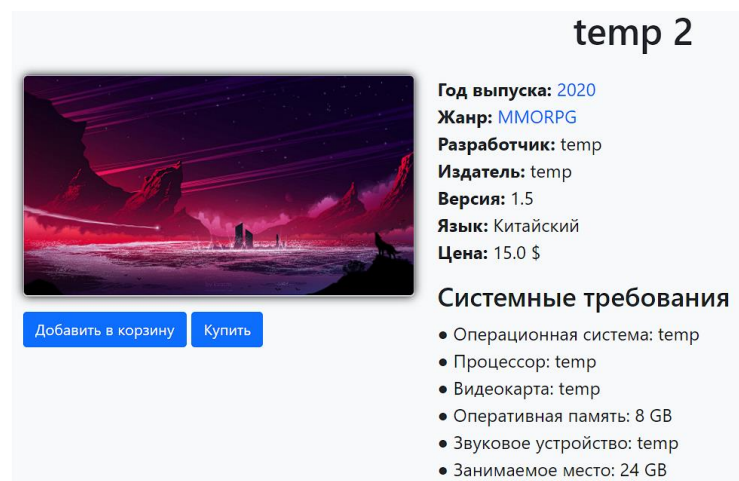


Рисунок 6.2.2 – Кнопки добавления в корзину и прямой покупки

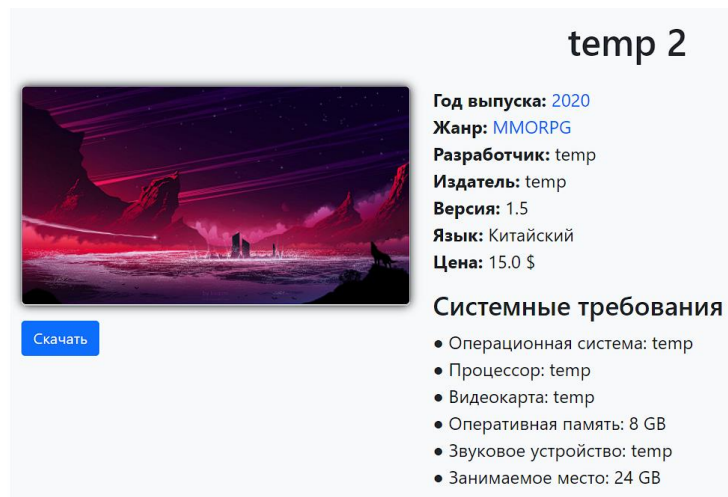


Рисунок 6.2.2 – После покупки появляется кнопка скачать

### 6.3 Алгоритм работы продавца

Продавец имеет те же возможности что и покупатель, но имеет еще возможности добавлять игры (см. рис. 6.3.1), редактировать добавленную игру (см. рис. 6.3.2) и смотреть свою статистику продаж игр (см. рис. 6.3.3).

[Главная](#)
[Каталог](#)
[О нас](#)
[Добавить игру](#)
[Корзина](#)

[Статистика](#)
[Выйти](#)

Название игры

Название игры

Разработчик

Разработчик

Издатель

Издатель

Постер

Выбор файла

Не выбран ни один файл

Скриншоты

Выбор файлов

Не выбран ни один файл

Выберите жанр

Action

Выберите язык

Английский

Год выпуска

Год выпуска

Версия

Версия

Цена, \$

Цена

Системные требования

Операционная система

Операционная система

Процессор

Процессор

Видеокарта

Видеокарта

Оперативная память

Оперативная память

GB

Звуковое устройство

Звуковое устройство

Занимаемое место

Занимаемое место

GB

Описание

Рисунок 6.3.1 – Страница добавления игры





## 7 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ

### 7.1 Диаграмма вариантов использования

Диаграмма вариантов использования (см. рис. 7.1.1) строится во время изучения технического задания, она состоит из графической диаграммы, описывающей действующие лица и прецеденты (варианты использования), а также спецификации, представляющего собой текстовое описание конкретных последовательностей действий (потока событий), которые выполняет пользователь при работе с системой.

Диаграмма вариантов использования

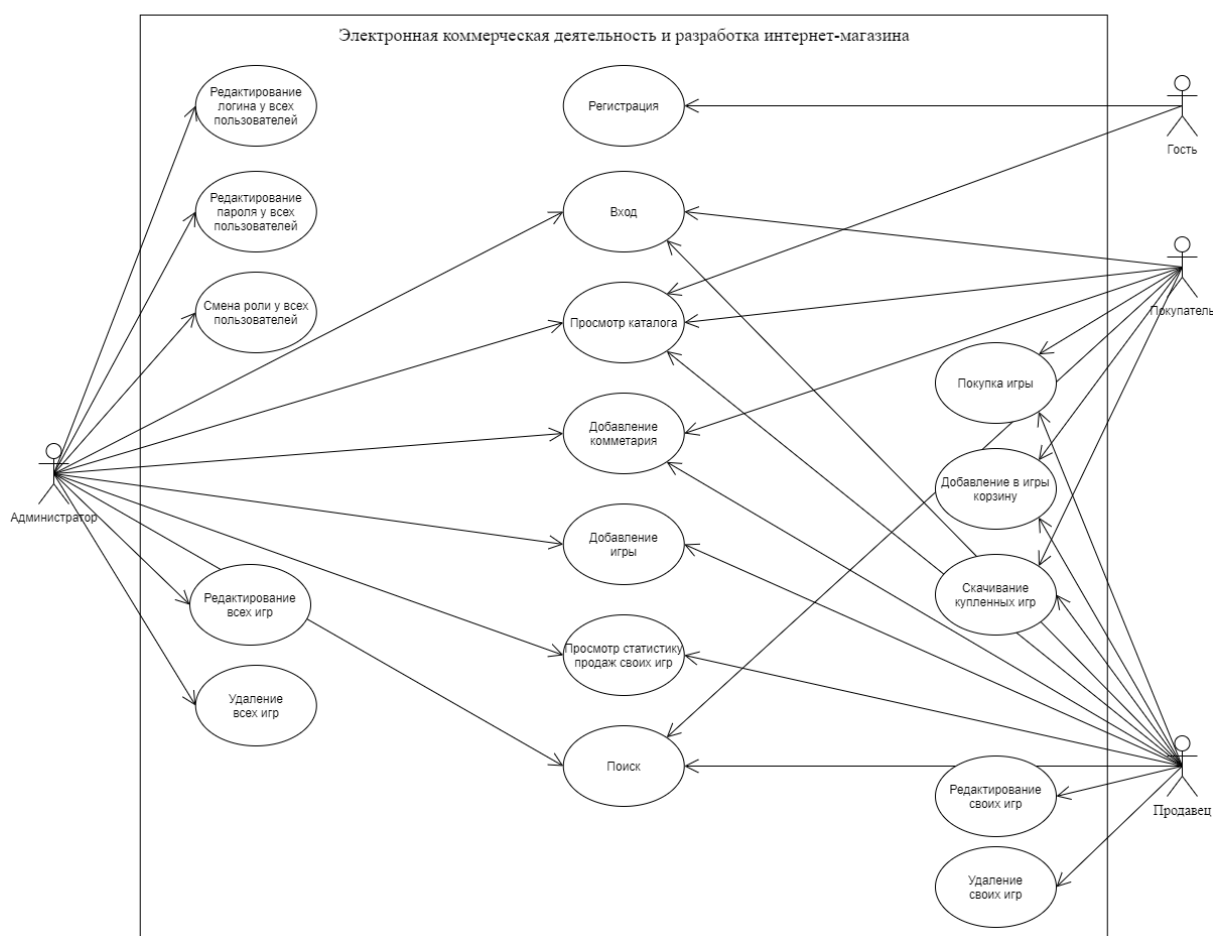


Рисунок 7.1.1 – Диаграмма вариантов использования

Гость может только просматривать каталог и регистрироваться.

Покупатель имеет такие же права как у покупатель, еще добавочно может покупать игры, скачивать их, добавлять комментарии и работать с корзиной.

Продавец имеет те же права что покупатель, еще добавочно может добавлять игры в каталог, просматривать статистику, редактировать свои игры и удалять их.

Администратор имеет все права доступа и все возможности других ролей, также имеет возможности редактировать профили, редактировать все игры и удалять их.

## 7.2 Диаграмма классов

Диаграмма классов (см. рис. 7.2.1) – это набор статических, декларативных элементов модели. Диаграммы классов могут применяться и при прямом проектировании, то есть в процессе разработки новой системы, и при обратном проектировании - описании существующих и используемых систем.

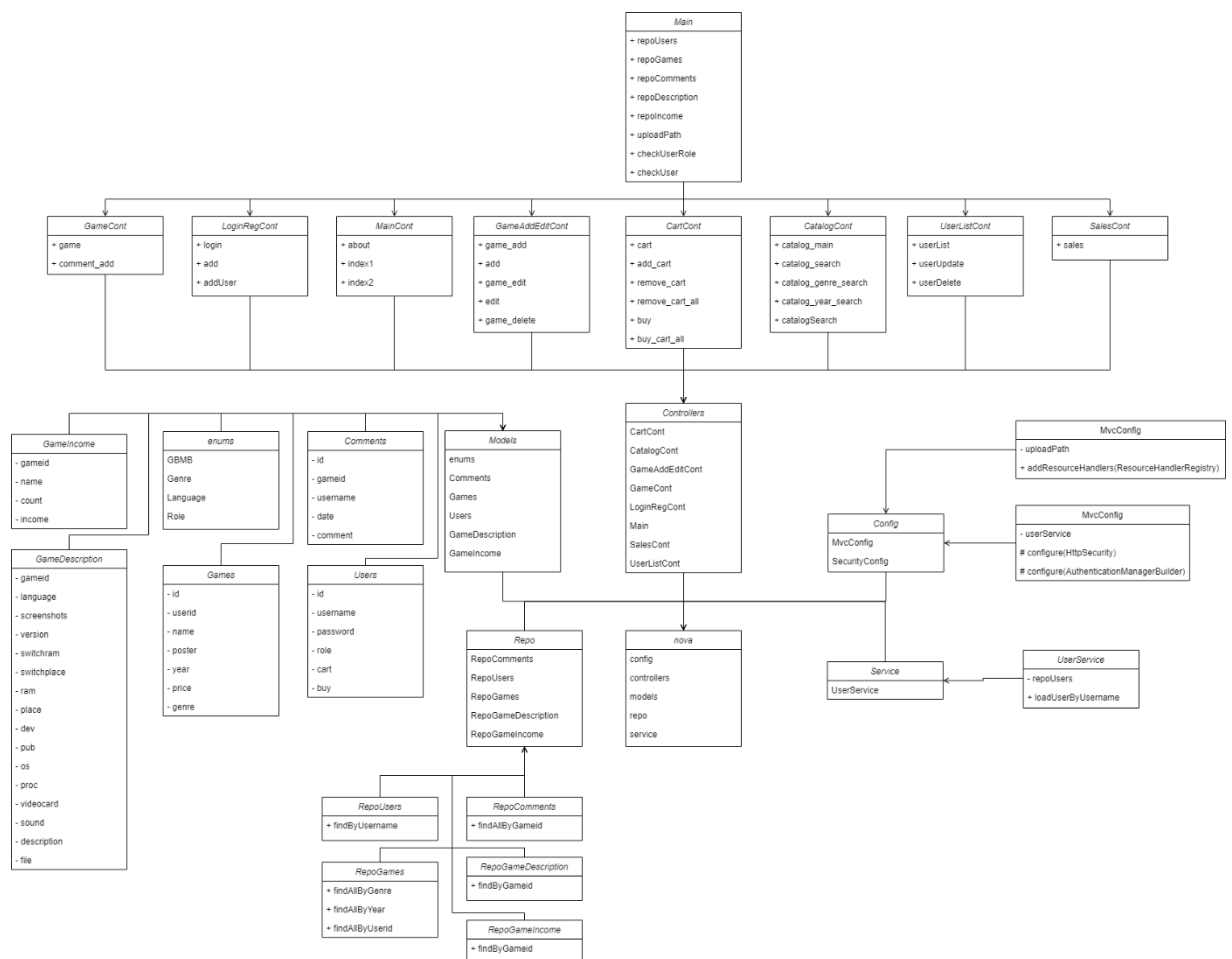


Рисунок 7.2.1 – Диаграмма классов

В диаграмме классов показана структура проекта (nova):

- config – хранятся конфигурационные файлы Spring Security и раздачи статических файлов и изображений;
- controllers – находятся вся логика сайта;
- models – хранит модели базы данных;
- repo – содержит уникальные sql запросы в базу данных игр и пользователей;
- service – находится конфигурация для корректной работы Spring Security.

### 7.3 Диаграмма последовательностей

Диаграмма последовательности (см. рис. 7.3.1) – UML-диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актеров (действующих лиц) информационной системы в рамках прецедента.

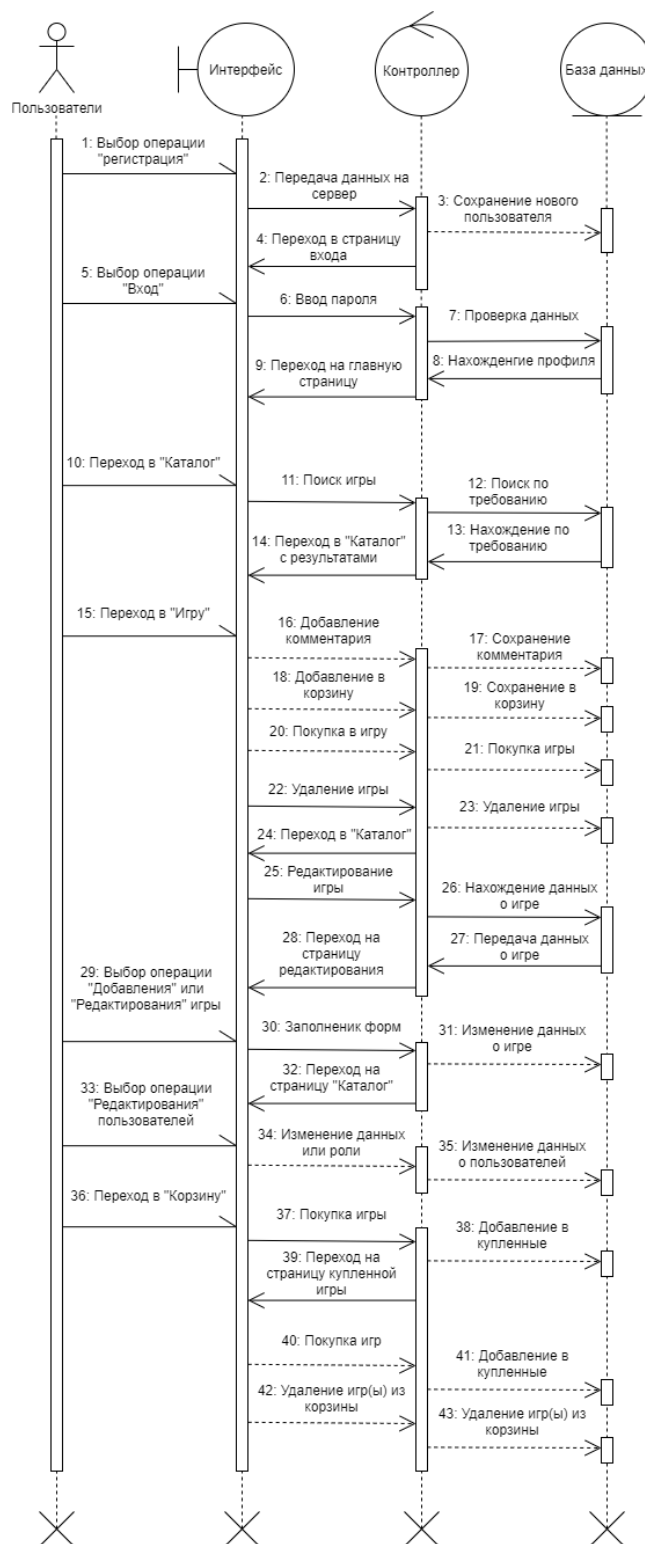


Рисунок 7.3.1 – Диаграмма последовательностей

На диаграмме показан алгоритм работы пользователей в страницах регистрации, вход, каталог, игра, редактирование игры, редактирование пользователей и корзина. При работе через интерфейс происходит запуск логики контроллера и взаимодействие с базой данных игр и пользователей.

## 7.4 Диаграмма состояний

Диаграмма состояний (см. рис. 7.4.1) – это, по существу, диаграмма состояний из теории автоматов со стандартизированными условными обозначениями, которая может определять множество систем от компьютерных программ до бизнес-процессов.

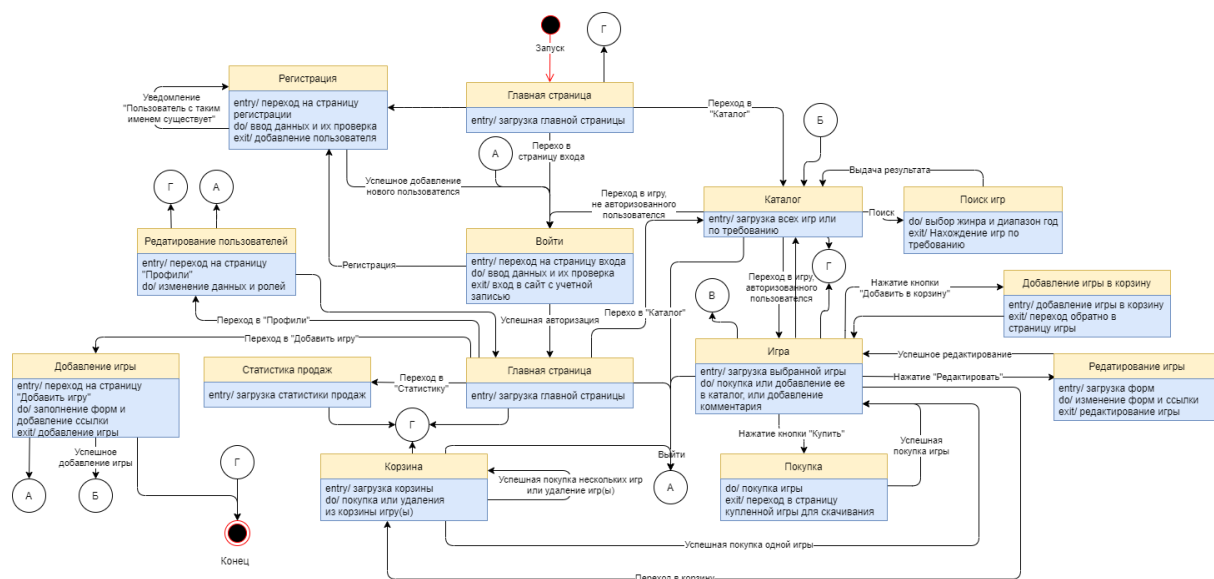


Рисунок 7.4.1 – Диаграмма состояний

В диаграмме показывает, как объект переходит из одного состояния в другое. Например, для входа в систему нужно авторизоваться, если нету, то новый пользователь переходит в страницу регистрации и создает новый профиль, при создании проверяет уникальность логина, после успешной регистрации нового пользователя, контроллер перенаправляет на страницу вход, после входа пользователь перенаправляется в главную страницу, или, администратор захотел изменить роль продавца у пользователя на покупателя, после изменения все добавленные игры этим пользователем до этого момента будут удалены, или администратор захотел сам изменить свою роль на покупателя, то так все игры добавленные до этого момента будут удалены и пользователя контроллер перенаправит в главную страницу.

## 8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

Тестирование сайта – это последний и обязательный этап технической разработки сайта. Он играет важнейшую роль в процессе создания ресурса, так как именно от качества тестирования зависит дальнейшая жизнь ресурса. К сожалению, очень часто разработчики не уделяют должного внимания этому этапу, полагаясь на свой опыт. В результате существующие ошибки приводят к колоссальным затратам времени и денег. Ведь ресурс, который имеет ошибки, вызывает негатив у посетителей и, как следствие, их потерю. В итоге владелец ресурса вынужден платить за доработку (а иногда за повторную разработку ресурса), а сотрудничество с бывшими разработчиками некачественного сайта оставляет только неприятный осадок.

В процессе разработки были обработаны исключения со стороны контроллера и интерфейса при регистрации, добавлении игры, удаление собственного профиля.

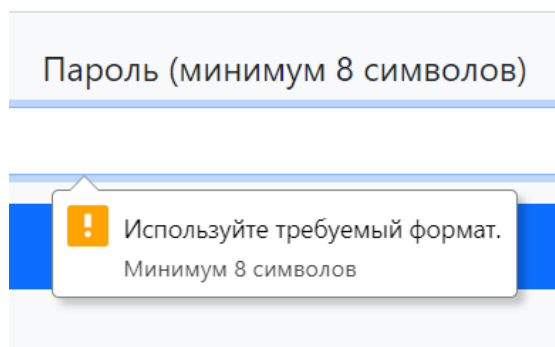


Рисунок 8.1 – Уведомление при регистрации нового пользователя

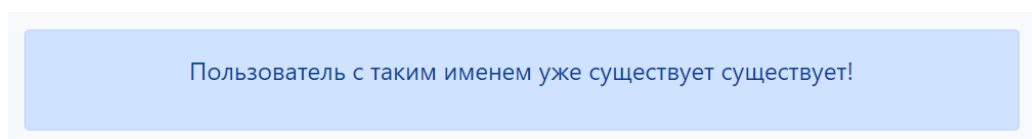


Рисунок 8.2 – Уведомление контроллера при регистрации нового пользователя с уже существующим именем

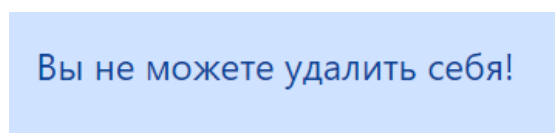


Рисунок 8.3 – Уведомление контроллера при попытке удаления Администратором свой профиль

Обработаны исключения при добавление новой игры, например при добавлении, HTML5 проверяет не пустые ли все ячейки, и правильны ли они заполнены, идет проверка цифр при введении полей год выпуска, версия, цена, оперативная память и занимаемое память, цифры не могут быть ниже 0, и в некоторых случаях выше 1024 или 1000.

## ЗАКЛЮЧЕНИЕ

При выполнении курсового проекта был произведен анализ предметной области, реализовано программное и информационное обеспечение, подготовлены контрольные примеры и произведены тестирования.

Спроектированная система имеет интуитивно понятный интерфейс для покупателя и продавца, программа автоматизирует работу, как и издателей так и разработчиков игр с геймерами, для удобной реализации продажи игр.

При завершении разработки программы были выполнены все поставленные задачи. Данная программа является законченной, но имеет возможность обновления и усовершенствования при необходимости в будущем, для более масштабного проекта, такие как: внутренняя валюта; рейтинг игр; поиск игр по рейтингу; поиск игр по издателям или разработчикам; добавление видео обзора для игры.

В результате написания курсового проекта были усовершенствованы навыки владения языком программирования Java, с помощью которого были реализованы алгоритмы данного проекта написанные с помощью платформы Spring Framework, и усовершенствованы навыки работы с базами данных, используя систему управления базами данных MySQL, также были изучена работа с гипертекстовой разметкой страниц сайта и их визуальное обработаны стили (HTML/CSS), был использован для облегченной и корректной разметки страниц проекта с помощью разделений на сетки благодаря платформе Bootstrap, дополнительно были использованы сторонние платформы Spring Framework, такие как паттерн проектирования Spring MVC, работа с пользователями Spring Security и набор настроенных модулей Spring Boot упрощающих конфигурацию приложений, написанного на платформе Spring Framework.

Выполнены следующие, поставленные задачи при выполнении данного проекта. Создан интуитивно понятный визуально минимальным интерфейсом сайта для пользователей, облегченной системой покупок и добавления игр. Добавлена система облегченной покупки игр для пользователей, облегченной страницей добавления игр для продавцов и администраторов. Было реализовано адаптивность вэб-дизайн для разных устройств, такие как: широкоформатные экраны; телефоны; ноутбуки.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Tproger [Электронный ресурс]. – Режим доступа:  
<https://tproger.ru/translations/java-intro-for-beginners/>
- [2] Bauman National Library [Электронный ресурс]. – Режим доступа:  
[https://ru.bmstu.wiki/Spring\\_Security](https://ru.bmstu.wiki/Spring_Security)
- [3] Хабр [Электронный ресурс]. – Режим доступа:  
<https://habr.com/ru/post/350864/>
- [4] Skillbox [Электронный ресурс]. – Режим доступа:  
[https://skillbox.ru/media/code/chto\\_takoe\\_git\\_obyasnyаем\\_na\\_skhemakh/](https://skillbox.ru/media/code/chto_takoe_git_obyasnyаем_na_skhemakh/)
- [5] Ruseller [Электронный ресурс]. – Режим доступа:  
<https://ruseller.com/lessons.php?id=666>
- [9] spring [Электронный ресурс]. – Режим доступа:  
<https://spring.io/guides/gs/serving-web-content/>
- [10] spring [Электронный ресурс]. – Режим доступа:  
<https://spring.io/guides/gs/securing-web/>
- [11] spring [Электронный ресурс]. – Режим доступа:  
<https://spring.io/guides/gs/accessing-data-mysql/>
- [12] javarush [Электронный ресурс]. – Режим доступа:  
<https://javarush.ru/groups/posts/spring-framework-java-1>
- [13] it-brain [Электронный ресурс]. – Режим доступа:  
[https://ru.it-brain.online/tutorial/spring/spring\\_quick\\_guide/](https://ru.it-brain.online/tutorial/spring/spring_quick_guide/)
- [14] proglib [Электронный ресурс]. – Режим доступа:  
<https://proglib.io/p/java-spring/>
- [15] javastudy [Электронный ресурс]. – Режим доступа:  
<https://javastudy.ru/interview/jee-spring-questions-answers/>
- [16] itproger [Электронный ресурс]. – Режим доступа:  
<https://itproger.com/course/java-spring>
- [17] Calltouch [Электронный ресурс]. – Режим доступа:  
<https://www.calltouch.ru/glossary/elektronnaya-kommertsiya/>
- [18] habr [Электронный ресурс]. – Режим доступа:  
<https://habr.com/ru/company/audiomania/blog/375533/>

## ПРИЛОЖЕНИЕ А

(обязательное)

### Антиплагиат

Оригинальность

91,5%

Заимствования

8,5%

#### СЕМАНТИЧЕСКИЕ ХАРАКТЕРИСТИКИ

---

##### Рубрикатор ВАК документа

05.13.18 Математическое моделирование, численные методы и комплексы программ: 26.37%

08.00.05 Экономика и управление народным хозяйством (по отраслям и сферам деятельности): 20.24%

08.00.14 Мировая экономика: 19.43%

05.13.06 Автоматизация и управление технологическими процессами и производствами (по отраслям): 17.04%

05.13.13 Телекоммуникационные системы и компьютерные сети: 16.91%

##### Рубрикатор ГРНТИ документа

50 Автоматика. Вычислительная техника: 58.63%

06 Экономика и экономические науки: 41.37%

##### Рубрикатор УДК документа

00 Наука в целом: 52.87%

33 Экономика. Народное хозяйство. Экономические науки: 37.26%

51 Математика: 9.87%

Оценка связности текста: 80.1980198019802

Наличие описания результатов исследования: да

Наличие описания метода исследования: да

Наличие введения: нет

Наличие выводов: нет

Наличие библиографии: да

Наличие аннотации: да

Доля общей лексики: 0

Доля научной лексики: 100



## ПРИЛОЖЕНИЕ Б

(обязательное)

### Листинг программного кода

```
CartCont
@GetMapping("/cart")
public String cart(Model model) {
    Users userFromDB = checkUser();

    if (userFromDB.getCart() != null) {
        long[] cart = userFromDB.getCart();
        float summary = 0;
        List<Games> games = new ArrayList<>(), temp = repoGames.findAll();

        for (Games g : temp) for (long c : cart) if (g.getId() == c) games.add(g);
        for (Games g : games) summary += g.getPrice();

        model.addAttribute("summary", summary);
        model.addAttribute("games", games);
        int i = 0;
        for (Games g : games) {
            i++;
            if (i == 2) {
                model.addAttribute("more", i);
                break;
            }
        }
    }

    model.addAttribute("role", checkUserRole());
    return "cart";
}

@PostMapping("/game/{id}/add_cart")
public String add_cart(@PathVariable(value = "id") Long id) {
    Users userFromDB = checkUser();

    long[] cart;
    if (userFromDB.getCart() == null) cart = new long[]{id};
    else {
        cart = new long[userFromDB.getCart().length + 1];
        for (int i = 0; i < userFromDB.getCart().length; i++) cart[i] = userFromDB.getCart()[i];
        cart[userFromDB.getCart().length] = id;
    }

    userFromDB.setCart(cart);

    repoUsers.save(userFromDB);
    return "redirect:/game/{id}";
}

@PostMapping("/game/{id}/remove_cart")
public String remove_cart(@PathVariable(value = "id") Long id) {
    Users userFromDB = checkUser();

    if (userFromDB.getCart().length == 1) userFromDB.setCart(null);
    else {
        long[] cart = new long[userFromDB.getCart().length - 1];
        int i = 0;
        for (long c : userFromDB.getCart()) {
```

```

        if (id == c) continue;
        cart[i] = c;
        i++;
    }
    userFromDB.setCart(cart);
}

repoUsers.save(userFromDB);
return "redirect:/cart";
}

@PostMapping("/game/remove_cart_all")
public String remove_cart_all(Model model) {
    Users userFromDB = checkUser();

    userFromDB.setCart(null);

    repoUsers.save(userFromDB);
    return "redirect:/cart";
}

@PostMapping("/game/{id}/buy")
public String buy(@PathVariable(value = "id") Long id) {
    Users userFromDB = checkUser();

    if (userFromDB.getCart() != null) {
        if (userFromDB.getCart().length == 1) userFromDB.setCart(null);
        else {
            long[] cart = new long[userFromDB.getCart().length - 1];
            int i = 0;
            for (long c : userFromDB.getCart()) {
                if (id == c) continue;
                cart[i] = c;
                i++;
            }
            userFromDB.setCart(cart);
        }
    }

    long[] buy;
    if (userFromDB.getBuy() == null) buy = new long[]{id};
    else {
        buy = new long[userFromDB.getBuy().length + 1];
        for (int i = 0; i < userFromDB.getBuy().length; i++) buy[i] = userFromDB.getBuy()[i];
        buy[userFromDB.getBuy().length] = id;
    }

    Optional<Games> temp = repoGames.findById(id);
    List<Games> games = new ArrayList<>();
    temp.ifPresent(games::add);

    for (Games g : games) {
        g.setCount(g.getCount() + 1);
        g.setIncome(g.getIncome() + g.getPrice());
        repoGames.save(g);
        break;
    }

    userFromDB.setBuy(buy);

    repoUsers.save(userFromDB);
    return "redirect:/game/{id}";
}

```

```

@PostMapping("/game/buy_cart_all")
public String buy_cart_all(Model model) {
    Users userFromDB = checkUser();

    long[] buy;
    if (userFromDB.getBuy() == null) {
        buy = new long[userFromDB.getCart().length];
        for (int i = 0; i < userFromDB.getCart().length; i++) buy[i] = userFromDB.getCart()[i];
    } else {
        buy = new long[userFromDB.getBuy().length + userFromDB.getCart().length];
        for (int i = 0; i < buy.length; i++) {
            for (int j = 0; j < userFromDB.getBuy().length; j++) {
                buy[i] = userFromDB.getBuy()[j];
                i++;
            }
            for (int j = 0; j < userFromDB.getCart().length; j++) {
                buy[i] = userFromDB.getCart()[j];
                Optional<Games> temp = repoGames.findById(userFromDB.getCart()[j]);
                List<Games> games = new ArrayList<>();
                temp.ifPresent(games::add);
                for (Games g : games) {
                    g.setCount(g.getCount() + 1);
                    g.setIncome(g.getIncome() + g.getPrice());
                    repoGames.save(g);
                    break;
                }
                i++;
            }
        }
    }

    userFromDB.setBuy(buy);
    userFromDB.setCart(null);

    repoUsers.save(userFromDB);

    return "redirect:/cart";
}

```

#### GameAddEditCont

```

@GetMapping("/game/add")
public String game_add(Model model) {
    model.addAttribute("role", checkUserRole());
    return "game_add";
}

```

```

@PostMapping("game/add")
public String add(@RequestParam String name, @RequestParam String dev, @RequestParam("poster") Multi-
partFile poster, @RequestParam("screenshots") MultipartFile[] screenshots, @RequestParam String pub,
@RequestParam int year, @RequestParam float version, @RequestParam float price, @RequestParam Genre genre,
@RequestParam Language language, @RequestParam String os, @RequestParam String proc, @RequestParam
String videocard, @RequestParam String sound, @RequestParam int ram, @RequestParam int place,
@RequestParam GBMB switchram, @RequestParam GBMB switchplace, @RequestParam String[] description,
@RequestParam String file) throws IOException {
    Games newGames = new Games(name, dev, pub, year, version, price, genre, language, os, proc, videocard,
sound, ram, place, switchram, switchplace, file);

```

```

    StringBuilder des = new StringBuilder();
    for (String s : description) des.append(s);
    newGames.setDescription(des.toString());
    String uuidFile = UUID.randomUUID().toString();

```

```

if (poster != null && !poster.getOriginalFilename().isEmpty()) {
    File uploadDir = new File(uploadPath);
    if (!uploadDir.exists()) uploadDir.mkdir();
    String result_poster = uuidFile + "_" + poster.getOriginalFilename();
    poster.transferTo(new File(uploadPath + "/" + result_poster));
    newGames.setPoster(result_poster);
}

if (screenshots != null && !screenshots[0].getOriginalFilename().isEmpty()) {
    uuidFile = UUID.randomUUID().toString();
    String result_screenshot;
    String[] result_screenshots = new String[screenshots.length];
    for (int i = 0; i < result_screenshots.length; i++) {
        result_screenshot = uuidFile + "_" + screenshots[i].getOriginalFilename();
        screenshots[i].transferTo(new File(uploadPath + "/" + result_screenshot));
        result_screenshots[i] = result_screenshot;
    }
    newGames.setScreenshots(result_screenshots);
}

newGames.setUserid(checkUser().getId());

repoGames.save(newGames);
return "redirect:/catalog/all";
}

@GetMapping("/game/{id}/edit")
public String game_edit(@PathVariable(value = "id") Long id, Model model) {
    if (!repoGames.existsById(id)) return "redirect:/catalog";
    Optional<Games> temp = repoGames.findById(id);
    ArrayList<Games> games = new ArrayList<>();
    temp.ifPresent(games::add);
    model.addAttribute("games", games);
    model.addAttribute("role", checkUserRole());
    return "game_edit";
}

@PostMapping("/game/{id}/edit")
public String edit(@PathVariable(value = "id") Long id, @RequestParam String name, @RequestParam String dev,
    @RequestParam("poster") MultipartFile poster, @RequestParam("screenshots") MultipartFile[] screenshots,
    @RequestParam String pub, @RequestParam int year, @RequestParam float version, @RequestParam float price,
    @RequestParam Genre genre, @RequestParam Language language, @RequestParam String os, @RequestParam
    String proc, @RequestParam String videocard, @RequestParam String sound, @RequestParam int ram,
    @RequestParam int place, @RequestParam GBMB switchram, @RequestParam GBMB switchplace,
    @RequestParam String[] description, @RequestParam String file) throws IOException {
    Games g = repoGames.findById(id).orElseThrow();
    StringBuilder des = new StringBuilder();
    for (String s : description) des.append(s);

    g.setDescription(des.toString());
    g.setName(name);
    g.setDev(dev);
    g.setPub(pub);
    g.setYear(year);
    g.setVersion(version);
    g.setPrice(price);
    g.setGenre(genre);
    g.setLanguage(language);
    g.setOs(os);
    g.setProc(proc);
    g.setVideocard(videocard);
    g.setSound(sound);

```

```

g.setRam(ram);
g.setPlace(place);
g.setSwitchram(switchram);
g.setSwitchplace(switchplace);
g.setFile(file);
String uuidFile = UUID.randomUUID().toString();

if (poster != null && !poster.getOriginalFilename().isEmpty()) {
    File uploadDir = new File(uploadPath);
    if (!uploadDir.exists()) uploadDir.mkdir();
    String result_poster = uuidFile + "_" + poster.getOriginalFilename();
    poster.transferTo(new File(uploadPath + "/" + result_poster));
    g.setPoster(result_poster);
}

if (screenshots != null && !screenshots[0].getOriginalFilename().isEmpty()) {
    uuidFile = UUID.randomUUID().toString();
    String result_screenshot;
    String[] result_screenshots = new String[screenshots.length];
    for (int i = 0; i < result_screenshots.length; i++) {
        result_screenshot = uuidFile + "_" + screenshots[i].getOriginalFilename();
        screenshots[i].transferTo(new File(uploadPath + "/" + result_screenshot));
        result_screenshots[i] = result_screenshot;
    }
    g.setScreenshots(result_screenshots);
}

repoGames.save(g);
return "redirect:/game/{id}"/";
}

@GetMapping("/game/{id}/delete")
public String game_delete(@PathVariable(value = "id") Long id) {
    repoGames.deleteById(id);
    List<Users> users = repoUsers.findAll();

    for (Users user : users) {
        if (user.getCart() != null) for (long carts : user.getCart())
            if (id == carts) {
                if (user.getCart().length == 1) user.setCart(null);
                else {
                    long[] cart = new long[user.getCart().length - 1];
                    int i = 0;
                    for (long c : user.getCart()) {
                        if (id == c) continue;
                        cart[i] = c;
                        i++;
                    }
                    user.setCart(cart);
                }
            }
        if (user.getBuy() != null) for (long carts : user.getBuy())
            if (id == carts) {
                if (user.getBuy().length == 1) user.setBuy(null);
                else {
                    long[] cart = new long[user.getBuy().length - 1];
                    int i = 0;
                    for (long c : user.getBuy()) {
                        if (id == c) continue;
                        cart[i] = c;
                        i++;
                    }
                    user.setBuy(cart);
                }
            }
    }
}

```

```

    }
}

for (Users user : users) repoUsers.save(user);
return "redirect:/catalog/all";
}

SalesCont
@GetMapping("/sales")
public String sales(Model model) {
    List<Games> games = repoGames.findAllByUserid(checkUser().getId());
    float income = 0;
    for (Games g : games) income += g.getIncome();

    model.addAttribute("income", income);
    model.addAttribute("games", games);
    model.addAttribute("role", checkUserRole());
    return "sales";
}

LoginRegCont
@GetMapping("/login")
public String login(Model model) {
    model.addAttribute("role", checkUserRole());
    return "login";
}

@GetMapping("/reg")
public String reg(Model model) {
    model.addAttribute("role", checkUserRole());
    return "reg";
}

@PostMapping("/reg")
public String addUser(Users user, Model model) {
    model.addAttribute("role", checkUserRole());
    Users userFromDB = repoUsers.findByUsername(user.getUsername());
    if (userFromDB != null) {
        model.addAttribute("message", "Пользователь с таким именем уже существует!");
        return "reg";
    }
    user.setRole(Role.USER);
    repoUsers.save(user);

    return "redirect:/login";
}

Main
@Autowired
RepoUsers repoUsers;

@Autowired
RepoGames repoGames;

@Autowired
RepoComments repoComments;

@Value("${upload.path}")
String uploadPath;

```

```

String checkUserRole() {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetail = (UserDetails) auth.getPrincipal();
        if (userDetail != null) {
            Users userFromDB = repoUsers.findByUsername(userDetail.getUsername());
            return String.valueOf(userFromDB.getRole());
        }
        return "NOT";
    }
    return "NOT";
}

Users checkUser() {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetail = (UserDetails) auth.getPrincipal();
        if (userDetail != null) {
            return repoUsers.findByUsername(userDetail.getUsername());
        }
    }
    return null;
}

```

SecurityConfig

@Autowired

private UserService userService;

@Override

```

protected void configure(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        .authorizeRequests()
        .antMatchers("/", "/reg", "/catalog/all", "/static/**", "/img/**").permitAll()
        .anyRequest()
        .authenticated()
        .and()
        .formLogin()
        .loginPage("/login").permitAll()
        .defaultSuccessUrl("/index")
        .and()
        .logout()
        .logoutRequestMatcher(new AntPathRequestMatcher("/logout", "POST"))
        .invalidateHttpSession(true)
        .clearAuthentication(true)
        .deleteCookies("JSESSIONID")
        .logoutSuccessUrl("/login");
}

```

@Override

```

protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .userDetailsService(userService)
        .passwordEncoder(NoOpPasswordEncoder.getInstance());
}

```

**ПРИЛОЖЕНИЕ В**  
(обязательное)  
**Ведомость документов**