Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационных компьютерных систем

Дисциплина "Объектно-ориентированное программирование"

*К защите допустить*:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
04.12.2024

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему

# Разработка интернет-магазина канцелярских товаров

БГУИР КР 1-40 05 01-10 № 152 ПЗ

Студент (подпись студента) Н.В. Рассолов

Курсовая работа
представлена на проверку
04.12.2024
_____
(подпись студента)

Минск 2024

# Реферат

БГУИР КР 1-40 05 01-10 № 152 ПЗ, гр. 814302

Н.В. Рассолов, Разработка интернет-магазина канцелярских товаров, Минск: БГУИР - 2024.

Пояснительная записка 123684 с., 17 рис., 0 табл.

Ключевые слова: интернет-магазин , товары , Ruby on Rails

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

*Объект: шаблоны проектирования, разработка пользовательского интерфейса. Цель: разработка интернет-магазина канцелярских товаров. Методология проведения работы: разработка дизайна сайта, разработка логики приложение в соответствии с архитектурным паттерном MVC. Ссылка на проект: https://github.com/big-qqq/shopoop Результаты работы: разработан удобный сайт интернет-магазин, для заказа продукции на дом или самовывозом. Область применения результатов: предназначено для потребителей, которые нуждаются в простом и удобном сайте для быстрых покупок.*

*Object: design patterns, user interface development. Goal: development of an online stationery store. Work methodology: development of the site design, development of the application logic in accordance with the MVC architectural pattern. Link to the project: https://github.com/big-qqq/shopoop Results: a convenient online store website was developed for ordering products at home or at pick-up. Scope of results: intended for consumers who need a simple and convenient site for quick purchases.*

# Содержание



# Введение

В наше время интернет является очень удобным и популярным местом для осуществления коммуникаций с потребителями, а также инструментом для упрощения жизней как продавцов, так и клиентов. Большое количество людей ищет нужную информацию в интернете, а не в газетах, журналах и брошюрах. Основные преимущества интернет-магазина по сравнению с физическим: • он доступен покупателю 24 часа в сутки, 7 дней в неделю, что позволяет вам не упустить клиентов в нерабочие часы; • ваш бизнес не привязан к конкретному региону – вы можете осуществлять торговлю не только в своем городе, но и во всей стране, и даже на международном уровне; • не нужно арендовать или покупать торговую площадь, можно обойтись просто складом для товаров, из которого и отгружать их напрямую; • «витрина»

интернет-магазина безгранична, т.е. у вас не возникнет проблем с недостатком места для представления товаров потенциальным покупателям; • on-line торговля не требует большого количества наемных сотрудников – если лично у вас не хватает времени на обработку заказов, вы можете нанять для этого всего лишь одного менеджера, и он будет выполнять ту работу, которую в обычном магазине делают условно пятеро консультантов; • это во многом объясняется тем, что в интернет-магазины обращаются целевые покупатели; • интернет торговля имеет гораздо меньший порог вхождения, по сравнению со стационарным магазином; • on-line магазин может быть как основным источником доходов, так и дополнительным к основному бизнесу; • популярность интернет-магазинов все больше и больше растет среди потребителей из-за экономии времени, легкости поиска нужного товара, конфиденциальности покупок, доступности в любое время суток. Еще одним преимуществом интернет-магазина является то, что он, будучи полноценным сайтом, содержит полную информацию о вашей компании: кто вы, чем занимаетесь, контактные данные, новости фирмы и т.д. Это является дополнительным методом рекламы для вашего бизнеса. Притом к этим данным, содержащимся на сайте, у пользователя есть доступ круглосуточно. Чего не скажешь о рекламных листовках или визитках, которые всегда теряются или выбрасываются. В курсовой работе поставлена цель создать интернет-магазин. Для достижения цели необходимо решить следующие задачи: 1 Изучить современные инструменты для разработки сайтов. 2 Разработать макета сайта. 3 Создать базу данных для хранения информации.

# 1 ОПИСАНИЕ ПРОЕКТА

### 1 Клиентская часть

Клиентская составляющая — это то, что видит пользователь на экране браузера. Веб-проект строится из текста, изображений, ссылок, списков, форм ввода данных, таблиц и тому т.п. По сути, вебразработка интернет-сайта заключается в размещении всех его текстовых и графических элементов в нужном месте и придание им необходимых форм/свойств.

Например, простой текст можно оформить в виде списка или заголовка и опубликовать его по средине экрана. Изображение можно увеличить или уменьшить, сделать его ссылкой и поместить в нужном месте. Таким образом, создание клиентской части вебсайта — это редактирования текста/графики веб-страницы, только не традиционным способом (как это делают мышью), а через написание специального кода на HTML, CSS. Процедура делится на следующие этапы:

Проектирование макета.
Вёрстка внешнего вида веб-ресурса по макету.
Программирование интерактивных возможностей.

HTML (Hypertext Markup Language) - это язык для описания структуры веб-страниц. Язык разметки HTML используется для определения текстового документа внутри тегов, который определяет структуру веб-страницы. HTMLиспользуется браузером для манипулирования текстом, изображениями и другим содержимым для отображения его в требуемом формате. Элементы HTML являются «строительными блоками» страниц. С помощью конструкций HTML изображения и другие объекты, такие как интерактивные формы, могут быть встроены в

отображаемую страницу. HTML предоставляет средства для создания структурированных документов, определяя структурную семантику для текста, такого как заголовки, абзацы, списки, ссылки, цитаты и другие элементы. HTML-элементы обозначены тегами, которые непосредственно вводят контент на страницу. Другие теги окружают и предоставляют информацию о тексте документа и могут включать другие теги в качестве подэлементов. Браузеры не отображают теги HTML, но используют их для интерпретации содержимого страницы.

CSS – это формальный язык, служащий для описания оформления внешнего вида документа, созданного с использованием языка разметки (HTML, XHTML, XML). Название происходит от английского Cascading Style Sheets, что означает «каскадные таблицы стилей». Назначение CSS – отделять то, что задает внешний вид страницы, от ее содержания. Если документ создан только с использованием HTML, то в нем определяется не только каждый элемент, но и способ его отображения (цвет, шрифт, положение блока и т. д.). Если же подключены каскадные таблицы стилей, то HTML описывает только очередность объектов. А за все их свойства отвечает CSS.

Такая технология:

обеспечивает относительно простую и быструю разработку, потому что однажды созданное оформление можно применять ко многим страницам;
повышает гибкость и удобство редактирования – достаточно внести правку в CSS, чтобы оформление изменилось везде;
делает код более простым, снижая повторяемость элементов. Его проще читать программистам и поисковым ботам;
ускоряет время загрузки, потому что CSS может кэшироваться при первом открытии, а в последующих считываются только структура и данные;
увеличивает количество визуальных решений для представления содержимого;
обеспечивает возможность легко применять к одному документу разные стили (например, создавать адаптированную версию для мобильных устройств или специальные стили для слабовидящих).

То есть каскадные таблицы служат не только для воплощения дизайна, но и кардинально меняют подход к сайтостроению, упрощая труд разработчиков и обеспечивая гибкость реализации.

Приложение, разрабатываемое в рамках курсовой работы, состоит из множества html-страниц. Эти страницы – главная страница, страница с различными категориями товаров, корзина и личный кабинет. Пользователь имеет доступ только к уже созданным админом категориям и товарам.

Для просмотра товаров магазина не обязательно быть авторизованным пользователем (см. рис. 1). Авторизация необходима только для заказа товаров. При нажатии на кнопку личного кабинета, если пользователь не авторизован или не зарегистрирован, он встречает страницу регистрации с кнопками авторизации и регистрации. После регистрации и\или авторизации URL-ы для работы с корзиной становятся доступными.

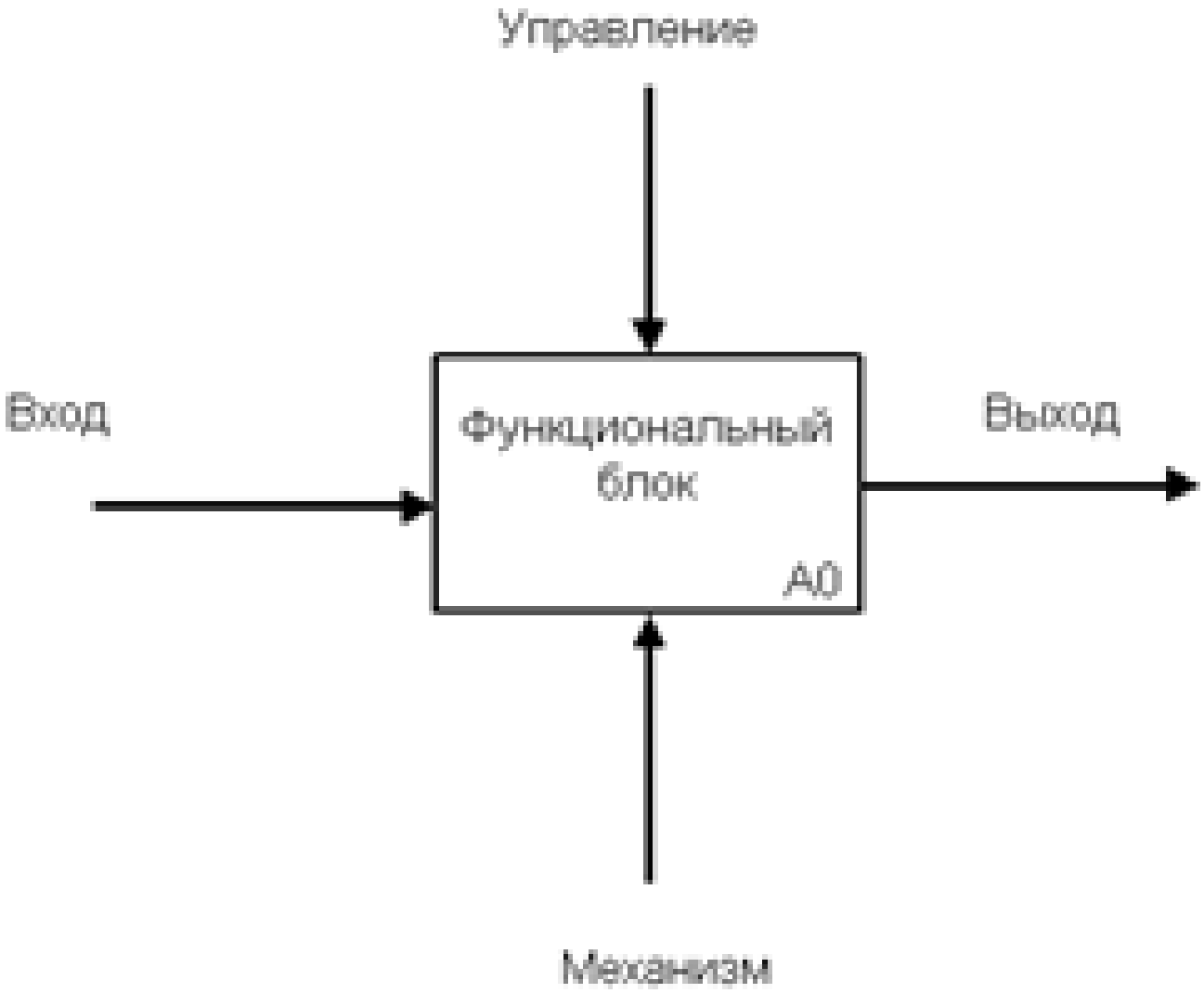


Рисунок 1 - Главная страница

С левого края страницы представлено поле, в котором пользователь может отсортировать товары по любым критериям (см. стр. 2), будь то максимальная цена или какая-либо определенная категория. Для примера выберем только ручки (см. стр. 3).

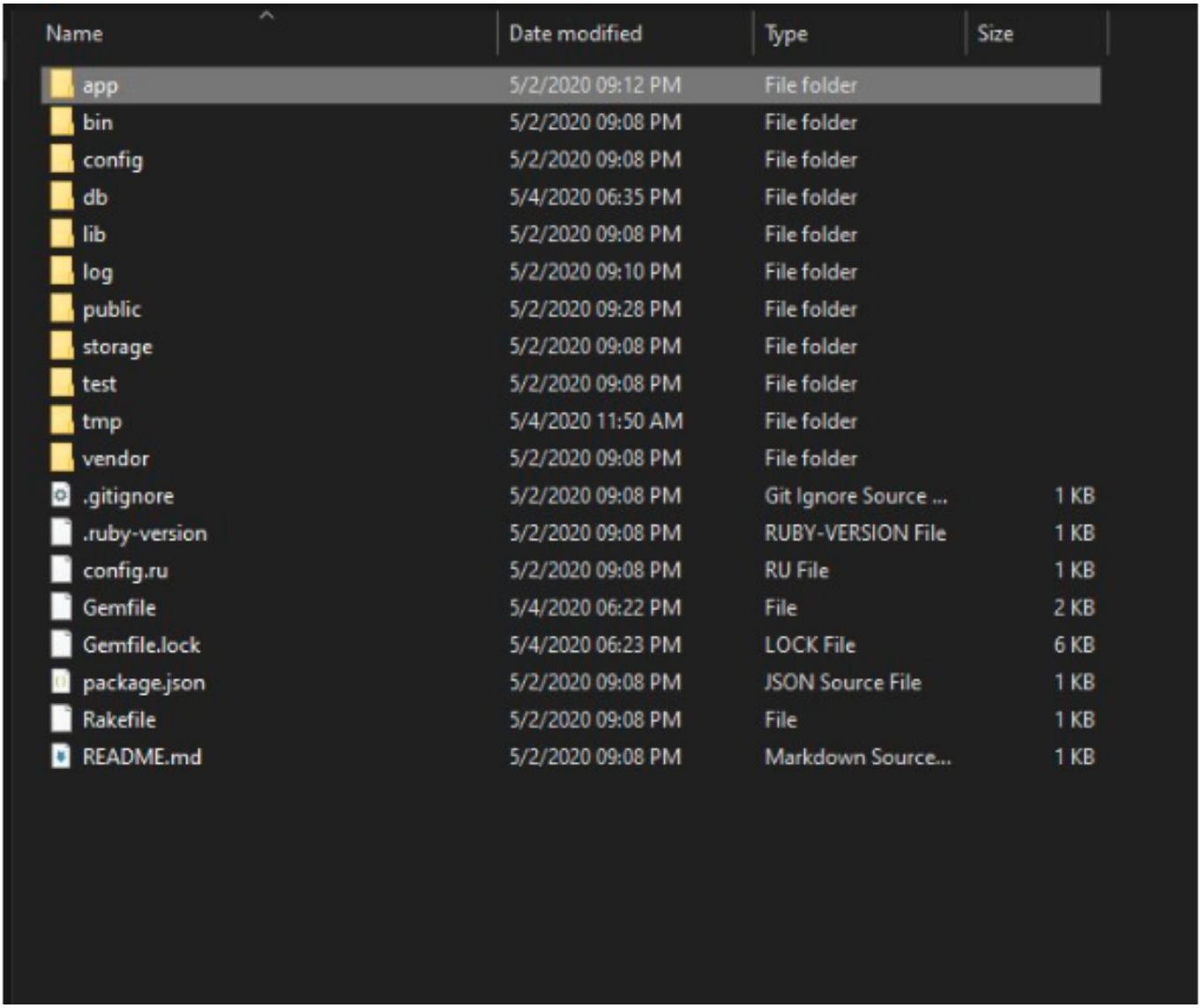


Рисунок 2 – Меню сортировки

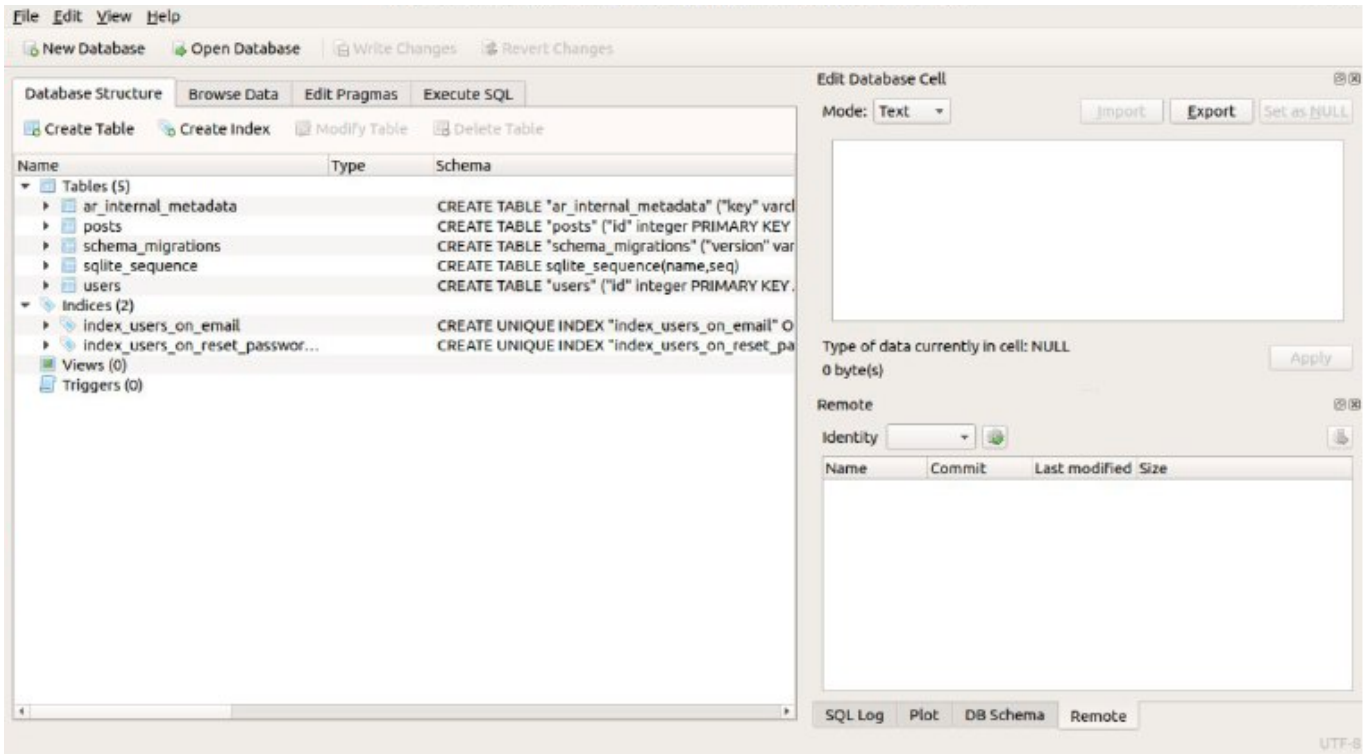

Рисунок 3 – Критерий “Ручки”

При нажатии на кнопку “Войти” или на иконку личного кабинета открывается страница с формой для авторизации (см. рис.4).

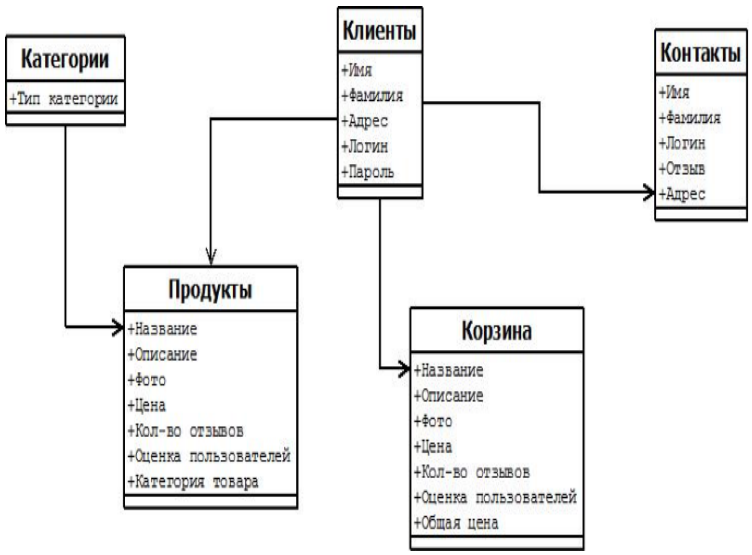


Рисунок 4 – Форма для авторизации

Если у пользователя еще нет аккаунта, то он должен зарегистрироваться, нажав на кнопку “Зарегистрироваться” (см.рис.5), после прохождения регистрации пользователь моментально заходит в свой аккаунт.

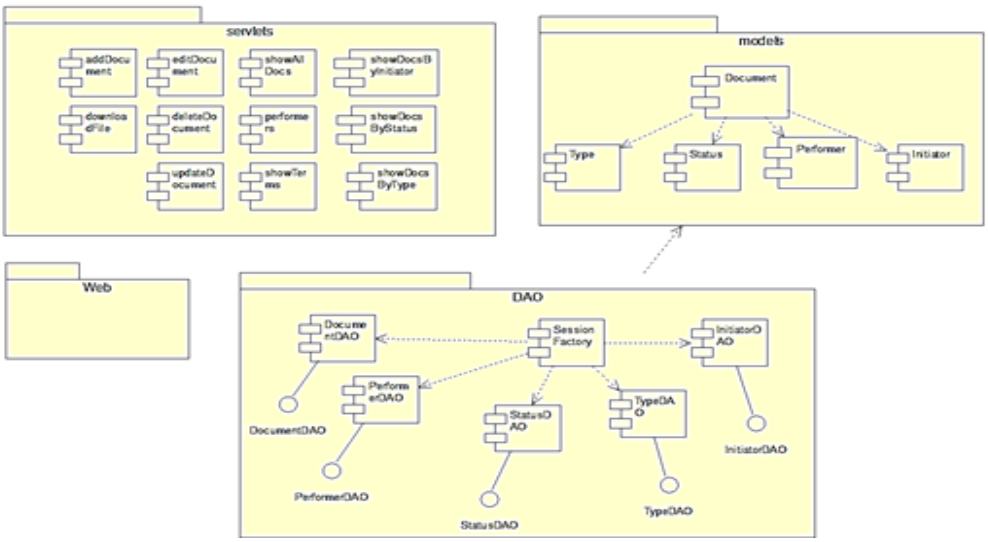


Рисунок 5 – Форма регистрации

После успешной авторизации\регистрации пользователь попадает на главную страницу магазина, где можно увидеть все товары.

Для уточнения деталей по товару надо нажать на его название и перейти на страницу описания и характеристик (см. рис.6). На этой странице мы видим сам товар, его название, характеристи, цену, описание.

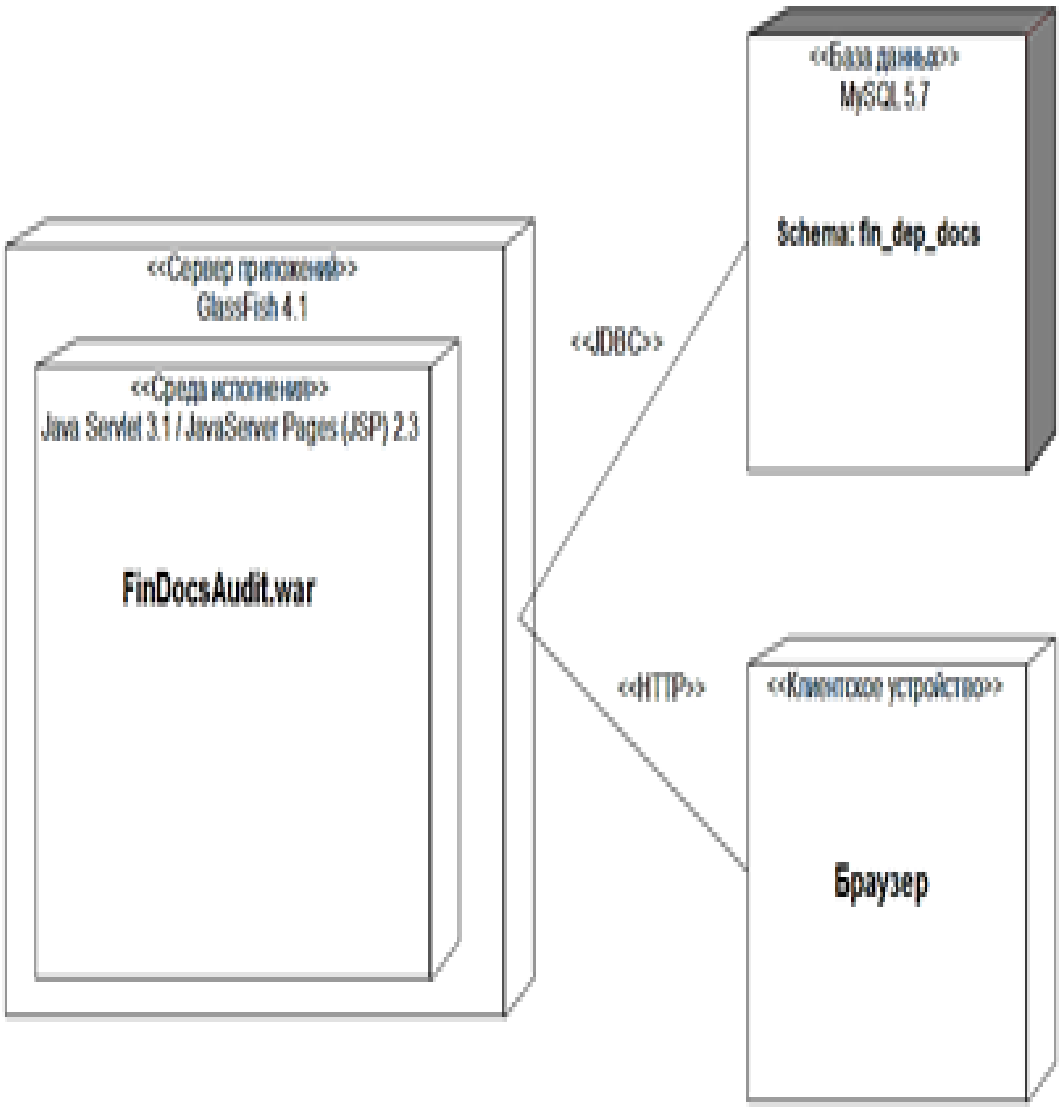


Рисунок 6 – Детальная страница товара

При нажатии на кнопку «Добавить в корзину», пользователь попадает на страницу, где он выбирает необходимое количество товара (см. рис.7).

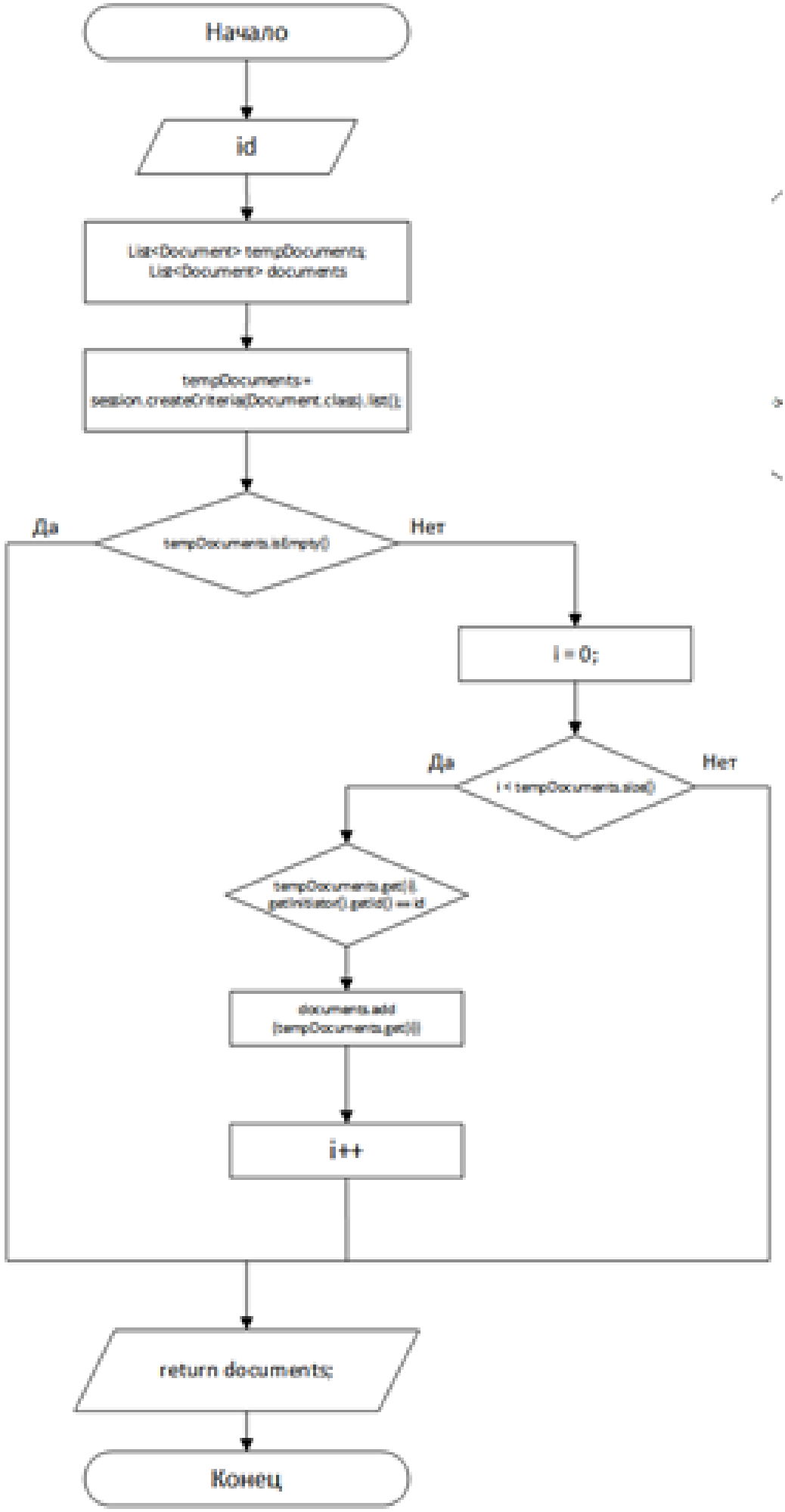
Начало
id
List<Document> tempDocuments;
List<Document> documents
tempDocuments =
session.createCriteria(Document.class).list();
Да
tempDocuments.isEmpty()
Нет
i = 0;
Да
i < tempDocuments.size()
Нет
tempDocuments.get(i).
getInitiator().getId() == id
documents.add
(tempDocuments.get(i))
i++
return documents;
Конец

Рисунок 7 – Страница добавления товара в корзину

После добавления всех нужных нам товаров следует нажать на кнопку “Корзина”, в корзине находятся все заказанные пользователем товары. В ней мы видим: общее количество позиций, количество каждой позиции, цену за единицу товара и сумму общего заказа. Если пользователь передумал покупать тот или иной товар, то он может без проблем удалить его (см.рис.8).

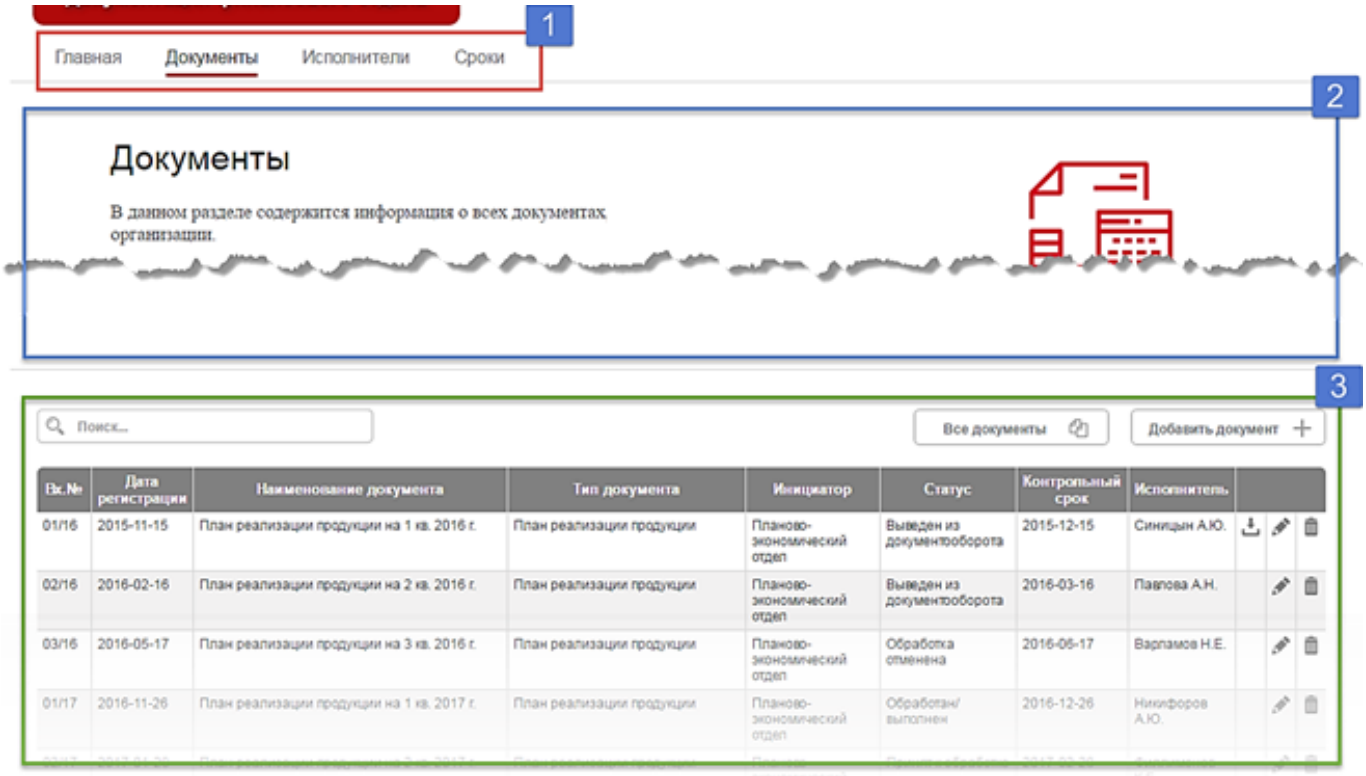


Рисунок 8 – Страница корзины

Если человека всё устраивает, то далее следует нажать кнопку “Оформить заказ”. После этого мы попадаем на страницу с формой для внесения наших данных (см.рис.9).

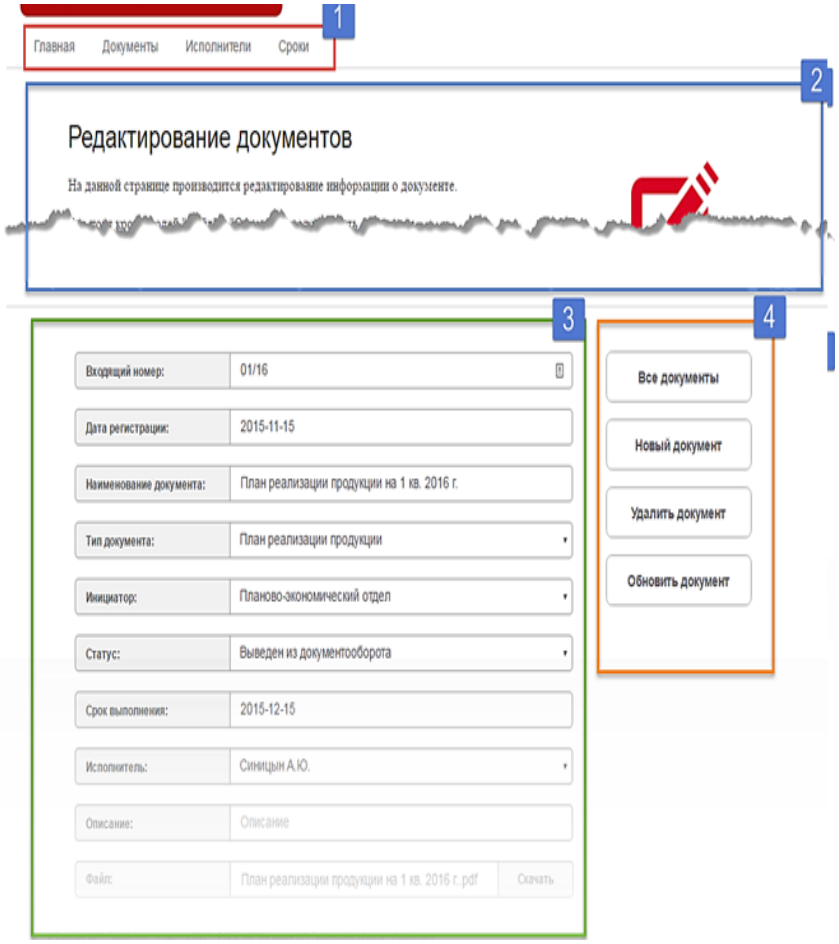


Рисунок 9– Форма оформления заказа

После оформления заказа мы попадаем в личный кабинет, где отображаются все заказы определённого пользователя (см.рис.10). Человек имеет возможность очистить историю покупок.

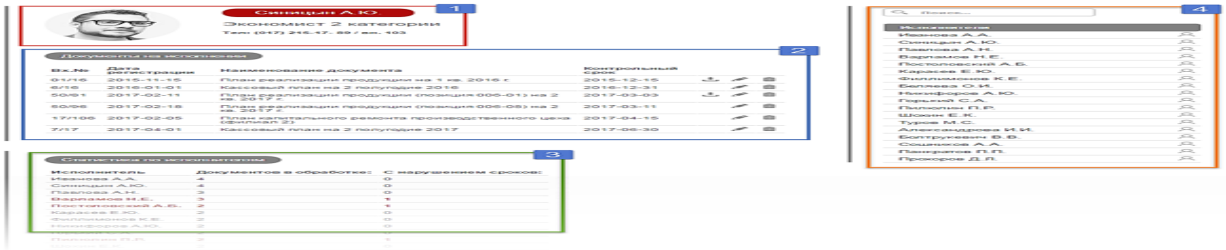

Рисунок 10 – Список покупок

Существует два вида аккаунтов: админ и обычный пользователь. После авторизации под именем админа открывается ряд функций, недоступных обычному пользователю, а именно: добавление\редактирование товара, добавление\редактирование категорий и удаление\редактирование пользователей (см. рис. 11-15).

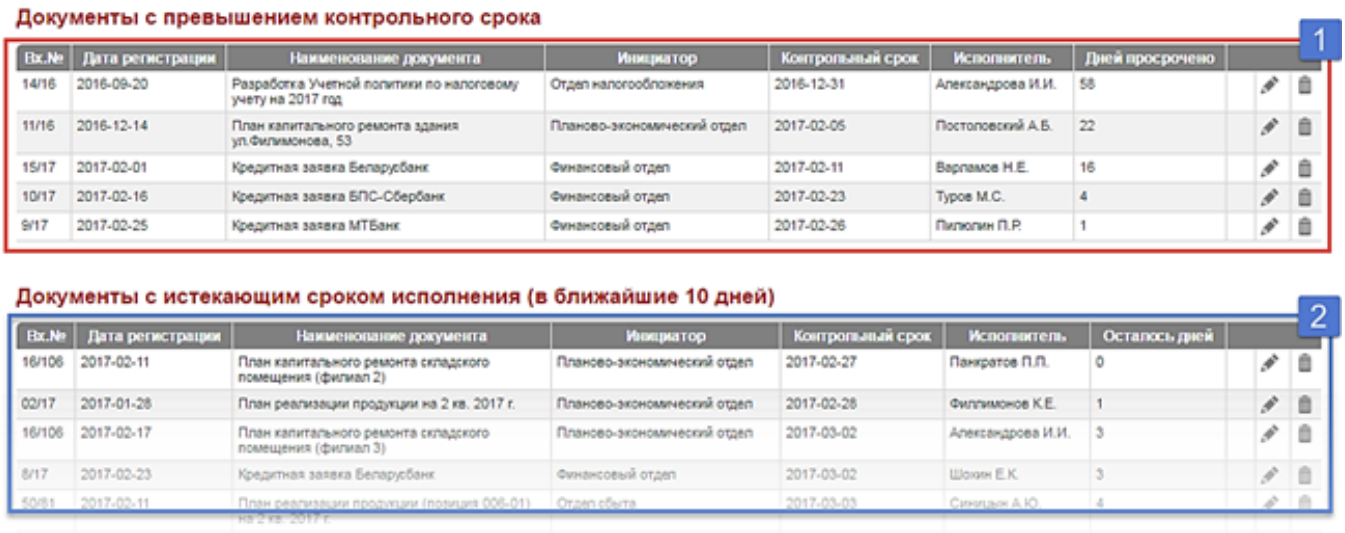


Рисунок 11 – Управление категориями

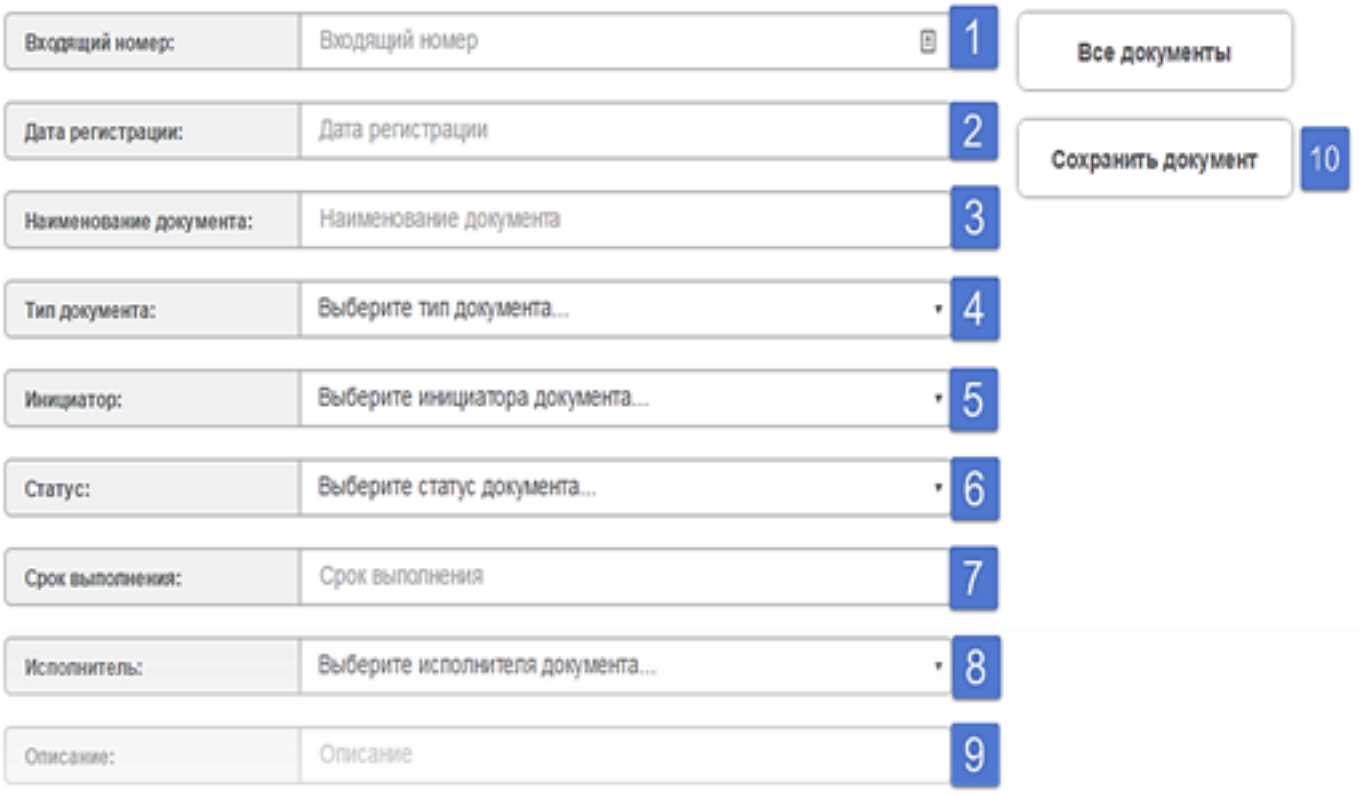


Рисунок 12 – Добавление\Редактирование категории

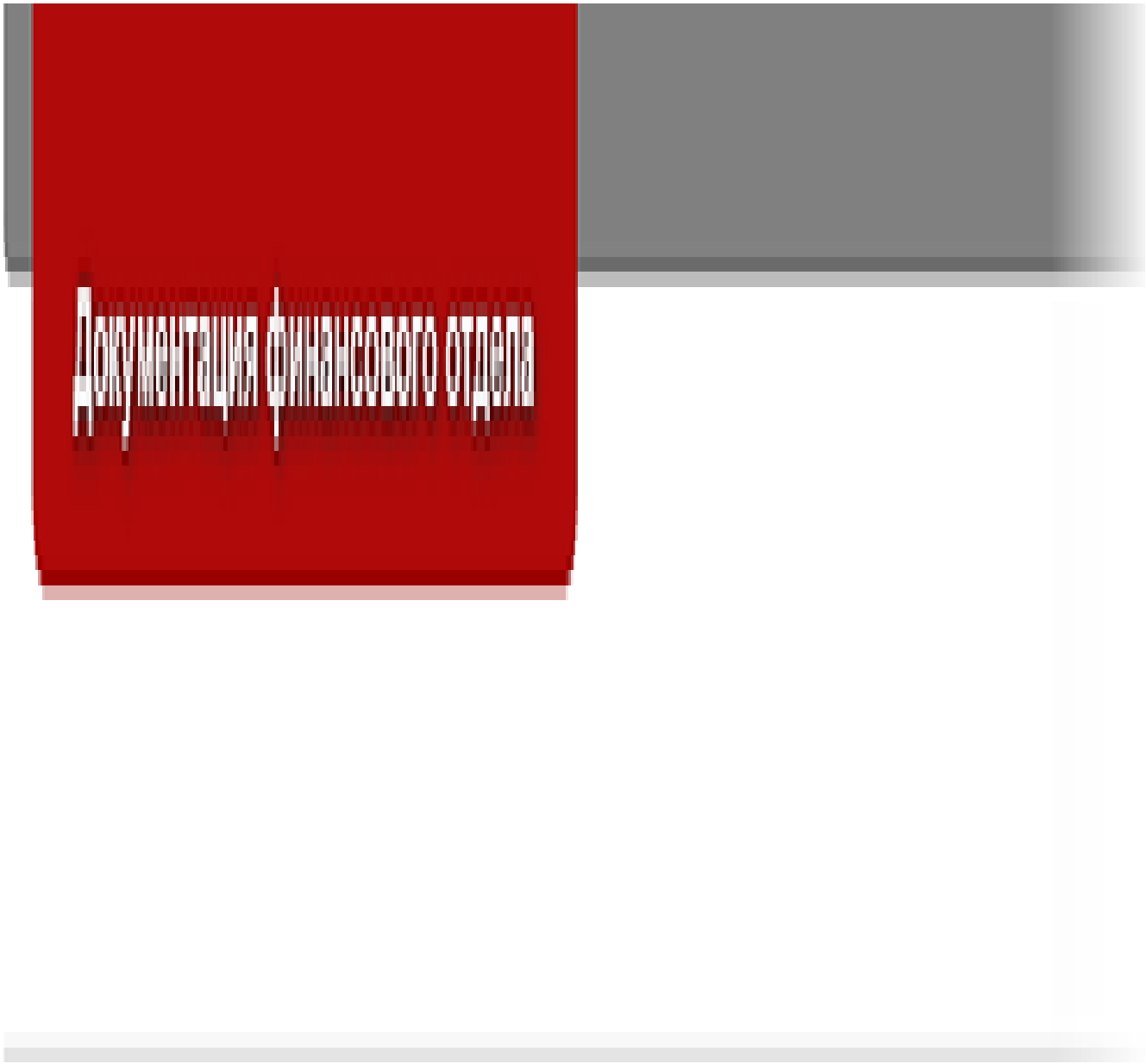


Рисунок 13 – Добавление\Редактирование товара


Рисунок 14 – Управление аккаунтами

Как пример проведем изменение над вторым аккаунтом. Изменим имя пользователя, поставим новый пароль и дадим привилегию админа (см. рис 15).

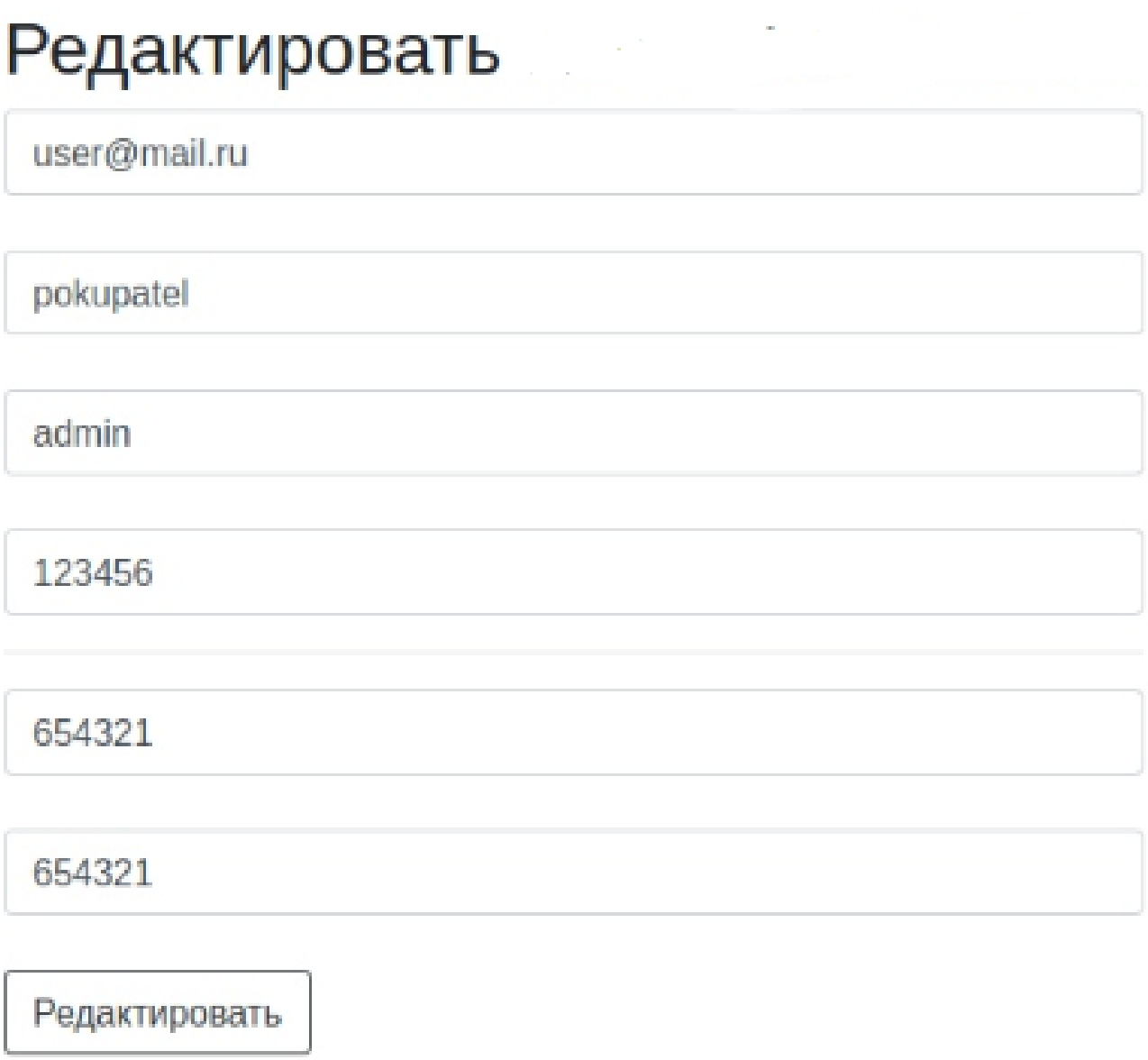


Рисунок 15 – Редактирование аккаунта

**2 Серверная часть**

Серверная часть получает запрос от клиента, выполняет вычисления, после чего формирует веб-страницу и отправляет её клиенту по сети с использованием протокола HTTP.

Протокол передачи гипертекста (HTTP) — это протокол прикладного уровня для распределенных, совместных, гипермедиа информационных систем. Это основа для передачи данных для Всемирной паутине (то есть Интернета) с 1990 года. HTTP — это протокол связи на основе TCP / IP, который используется для доставки данных (файлы HTML, файлы изображений, результаты запросов и так далее) в World Wide Web. Порт по умолчанию – TCP80, но можно использовать и другие порты. Он обеспечивает стандартизированный способ связи компьютеров друг с другом. Спецификация HTTP определяет, как данные запросов клиентов будут создаваться и отправляться на сервер, и как серверы отвечают на эти запросы.

Есть три основных функции, которые делают HTTP простым, но мощным протоколом:

HTTP-клиент, то есть браузер, инициирует HTTP-запрос, и после того, как запрос выполнился, клиент ожидает ответа. Сервер обрабатывает запрос и отправляет ответ, после чего клиент отключает соединение. Дальнейшие запросы отвечают на новое соединение, как клиент и сервер являются новыми друг для друга.
HTTP не зависит от носителя: это означает, что любой тип данных может быть отправлен по HTTP, если и клиент, и сервер знают, как обрабатывать содержимое данных. Как клиенту, так и серверу необходимо указать тип контента, используя соответствующий MIME-тип.
Сервер и клиент знают друг друга только во время текущего запроса. После этого они оба забывают друг о друге. Из-за такой природы протокола ни клиент, ни браузер не могут сохранять информацию между выполнением различных запросов на веб-страницах.

Как и на стороне клиента, «сторона сервера» означает все, что происходит на сервере, а не на клиенте. В прошлом почти вся бизнес-логика работала на стороне сервера, и это включало рендеринг динамических веб-страниц, взаимодействие с базами данных, аутентификацию личности и push-уведомления. Проблема с размещением всех этих процессов на стороне сервера заключается в том, что каждый запрос, связанный с одним из них, должен каждый раз проходить весь путь от клиента до сервера. Это вносит большую задержку. По этой причине современные приложения запускают больше кода на стороне клиента, один из вариантов такого взаимодействия - рендеринг динамических веб-страниц в реальном времени путем запуска скриптов в браузере, которые вносят изменения в контент, который видит пользователь.

Тем не менее, серверная часть во многом незаменима. Но если бы разработчику приходилось вручную заниматься серверной частью, то разработка одного просто приложения занимала бы очень много времени и сил. Для того, чтобы облегчить разработку веб-приложений и сделать ее удобной и, главное, эффективной, придумали веб-фрэймворки. Говоря простыми словами, framework – это каркас, который состоит из множества различных библиотек, которые облегчают разработку программного продукта или сайта. То есть фреймворк это набор из нескольких библиотек. Фреймворки различны по своим возможностям и функциям. Их наполнение зависит от поставленных задач. Они нужны для того, чтобы программист или дизайнер могли сосредоточиться на уникальных задачах, а не заново изобретали колесо.

Ruby on Rails— это многоуровневый MVC-фреймворк для построения веб-приложений, использующих реляционные и NoSQL базы данных (например, MySQL, MariaDB, PostgeSQL, MongoDB). Фреймворк написан на языке программирования Ruby. Rails подходит как для разработки обычных сайтов, которые должны быть реально быстрыми, отказоустойчивыми и работающими под высокой нагрузкой, так и для веб-приложений со сложной бизнес-логикой и динамичными web-интерфейсами. Ruby on Rails является открытым программным обеспечением и распространяется под лицензией MIT.

Основными компонентами приложений на Ruby on Rails являются модель (англ. model), представление (англ. view) и контроллер (англ. controller). Ruby on Rails использует REST-стиль построения веб-приложений.

Модель предоставляет остальным компонентам приложения объектно-ориентированное отображение данных (таких как каталог продуктов или список заказов). Объекты модели могут

осуществлять загрузку и сохранение данных в реляционной базе данных, а также реализуют бизнес-логику.

Для хранения объектов модели в реляционной СУБД по умолчанию в Rails 3 использована библиотека ActiveRecord. Конкурирующий аналог — DataMapper. Существуют плагины для работы с нереляционными базами данных, например Mongoid для работы с MongoDB. Представление создаёт пользовательский интерфейс с использованием полученных от контроллера данных.

Представление также передает запросы пользователя на манипуляцию данными в контроллер (как правило, представление не изменяет непосредственно модель).

В Ruby on Rails представление описывается при помощи шаблонов ERB — файлов HTML с дополнительными включениями фрагментов кода Ruby (Embedded Ruby, или ERb). Вывод, сгенерированный встроенным кодом Ruby, включается в текст шаблона, после чего получившаяся страница HTML возвращается пользователю. Кроме ERB возможно использовать ещё около 20 шаблонизаторов, в том числе Haml.

Контроллер в Rails — это набор логики, запускаемой после получения HTTP-запроса сервером. Контроллер отвечает за вызов методов модели и запускает формирование представления.

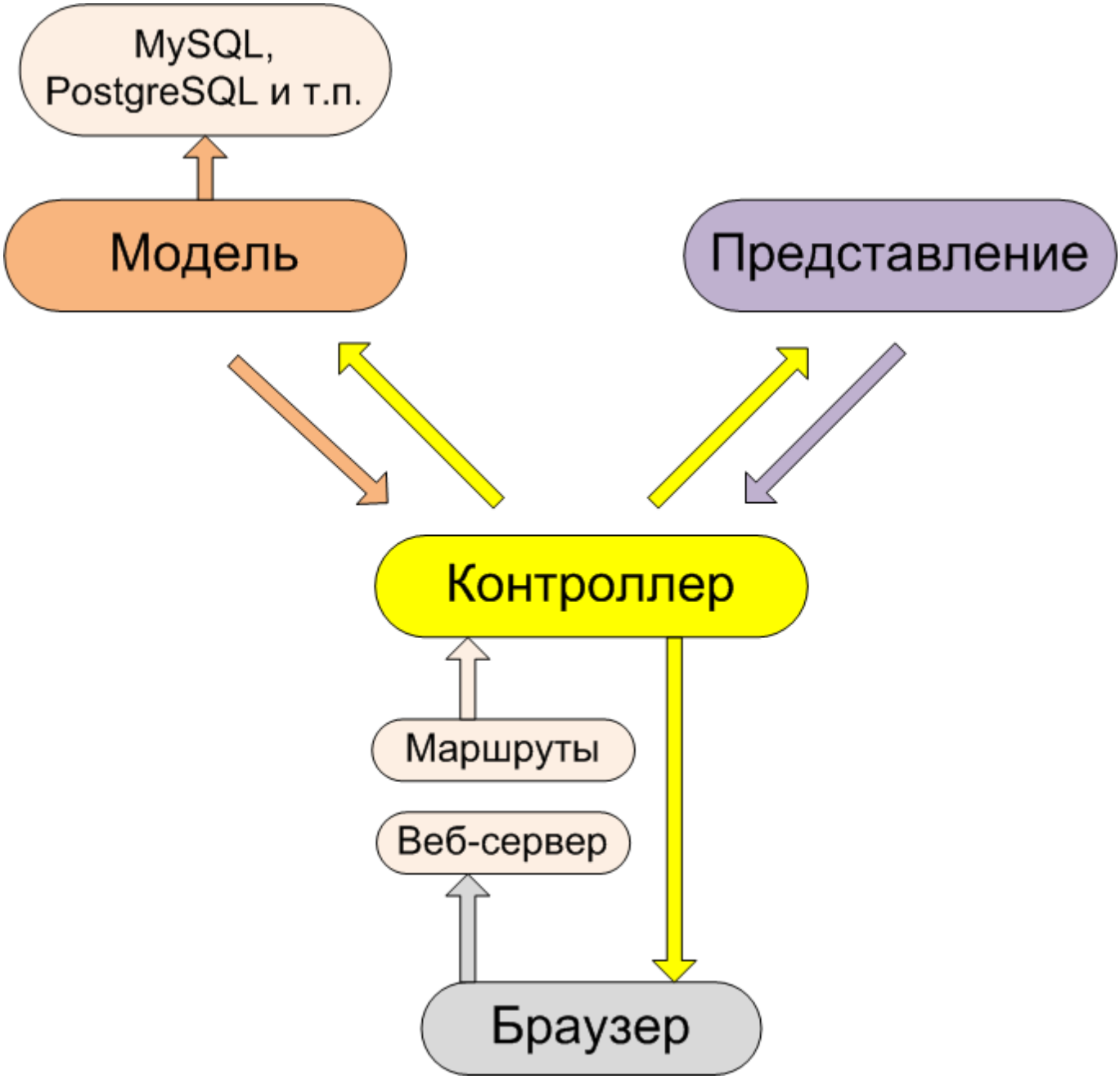


Рисунок 16 - Схематическое представление архитектуры модель-представление-контроллер с дополнительными компонентами.

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 используется веб-сервер Puma (ранее WEBrick, встроенный в Ruby, или Mongrel). В качестве сервера базы данных поддерживаются MySQL, PostgreSQL, Firebird, DB2, Oracle и Microsoft SQL Server. Также поддерживается встраиваемая база данных SQLite.

Для Windows существует дистрибутив Instant Rails с настроенной и готовой к работе сразу после установки рабочей средой для разработки Rails-приложений, которая включает в себя сервер Apache и СУБД MySQL, а также дистрибутив RubyInstaller, включающий последние версии Ruby и инструменты разработчика. Для платформ Windows, Linux, Mac OS X имеется комплексный установщик BitNami RubyStack[5], включающий в себя все необходимое для

разработки в среде Rails, включая Ruby, RubyGems, Ruby on Rails, MySQL, Apache, Mongrel и Subversion.

В качестве репозитория плагинов Ruby on Rails использует экосистему пакетов RubyGems, которые также называются «джемы» (gem с англ. — «самоцвет»). Некоторые плагины со временем были включены в базовую поставку Rails, например Sass и CoffeeScript; другие же, хотя и не были включены в базовую поставку, являются стандартом де-факто для большинства разработчиков.

Для создания сайта на Ruby on Rails я использовал текстовый редактор Visual Studio Code.

# 2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

### 2.1 Объектно-ориентированное программирование

Объектно-ориентированное программирование (в дальнейшем ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов. В центре ООП находится понятие объекта.

Объект — это сущность, экземпляр класса, которой можно посылать сообщения, и которая может на них реагировать, используя свои данные. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования. Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм, то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

Абстрагирование — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения не значимые. Соответственно, абстракция — это набор всех таких характеристик.

В контексте ООП абстракция — это обобщение данных и поведения для типа, находящегося выше текущего класса по иерархии. Перемещая переменные или методы из подкласса в супер класс, вы обобщаете их. Но язык добавляет также понятия абстрактных классов и абстрактных методов.

Абстрактный класс является классом, для которого нельзя создать экземпляр.

Инкапсуляция — свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента, а взаимодействовать с ним посредством предоставляемого интерфейса, а также объединить и защитить жизненно важные для компонента данные. При этом пользователю предоставляется только спецификация (интерфейс) объекта.

Цель инкапсуляции – уйти от зависимости внешнего интерфейса класса (то, что могут использовать другие классы) от реализации. Чтобы малейшее изменение в классе не влекло за собой изменение внешнего поведения класса.

Существует 4 вида модификаторов доступа: **public, protected, private** и **default**.

Public – уровень предполагает доступ к компоненту с этим модификатором из экземпляра любого класса и любого пакета.

Protected – уровень предполагает доступ к компоненту с этим модификатором из экземпляров родного класса и классов-потомков, независимо от того, в каком пакете они находятся.

Default – уровень предполагает доступ к компоненту с этим модификатором из экземпляров любых классов, находящихся в одном пакете с этим классом.

Private – уровень предполагает доступ к компоненту с этим модификатором только из этого класса.

Наследование — это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов. Применение иерархии классов делает управляемыми большие потоки информации.

Полиморфизм — это способность объекта использовать методы производного класса, который не существует на момент создания базового.

В более общем смысле, концепцией полиморфизма является идея “один интерфейс, множество методов“. Это означает, что можно создать общий интерфейс для группы близких по смыслу действий. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компилятор.

Абстракция — это обобщение данных и поведения для типа, находящегося выше текущего класса по иерархии.

Упоминая ООП, следует сказать про принципы SOLID.

SOLID — это аббревиатура пяти основных принципов проектирования в объектно-ориентированном программировании. Она была предложена Робертом Мартином. Эти принципы позволяют строить на базе ООП масштабируемые и сопровождаемые программные продукты с понятной бизнес-логикой.

Вот как расшифровывается акроним SOLID:

S: Single Responsibility Principle (Принцип единственной ответственности).
O: Open-Closed Principle (Принцип открытости-закрытости).
L: Liskov Substitution Principle (Принцип подстановки Барбары Лисков).
I: Interface Segregation Principle (Принцип разделения интерфейса).
D: Dependency Inversion Principle (Принцип инверсии зависимостей).

***Принцип единственной ответственности*** (single responsibility principle / SRP) обозначает, что каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс. Все его сервисы должны быть направлены исключительно на обеспечение этой обязанности

***Принцип открытости / закрытости*** (open-closed principle / OCP) декларирует, что программные сущности (классы, модули, функции и т. п.) должны быть открыты

для расширения, но закрыты для изменения. Это означает, что эти сущности могут менять свое поведение без изменения их исходного кода.

***Принцип подстановки Барбары Лисков*** (Liskov substitution principle / LSP) в формулировке Роберта Мартина: «функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом».

***Принцип разделения интерфейса*** (interface segregation principle / ISP) в формулировке Роберта Мартина: «клиенты не должны зависеть от методов, которые они не используют». Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

***Принцип инверсии зависимостей*** (dependency inversion principle / DIP) — модули верхних уровней не должны зависеть от модулей нижних уровней, а оба типа модулей должны зависеть от абстракций; сами абстракции не должны зависеть от деталей, а вот детали должны зависеть от абстракций.

### 2.2 Язык программирования Ruby

Ruby — интерпретируемый, полностью объектно-ориентированный язык программирования с четкой динамической типизацией. Он сочетает в себе Perl-подобный синтаксис с объектно-ориентированным подходом. Также некоторые черты заимствованы из языков программирования Python, Lisp, Dylan и CLU. Кроссплатформенная реализация интерпретатора языка Ruby распространяется на условиях открытого программного обеспечения. Код, написанный на Ruby, может быть понятен даже человеку, который не разбирается в программировании. На RoR были созданы такие проекты, как Redmine, Twitter, Shopify, Basecamp, GitHub, Kickstarter, Airbnb и другие.

**Преимущества Ruby**

Многочисленное и доброжелательное комьюнити.
Довольно высокий порог входа, что означает, что Ruby-разработчик с большой вероятностью имеет опыт работы как минимум с еще одним языком программирования.
Вы используете только те библиотеки и модули, которые необходимы.
Существует большое количество полезных библиотек, которые уже готовы к использованию (Ruby Gems).
В интернете есть много информации по Ruby, в структурированном и отсеянном виде.
В контексте обсуждения Ruby нельзя не упомянуть популярнейший фреймворк Ruby on Rails.

А теперь поговорим о некоторых преимуществах Ruby более подробно.

**Скорость разработки**

Одним из плюсов Ruby и фреймворка RoR считается скорость разработки. Личный опыт и опыт коллег показывает, что решение задач на Rails происходит в полтора раза быстрее по сравнению с другими фреймворками подобного плана. Ruby очень прост как инструмент, а также существует огромное количество готовых решений для различного рода задач.

**Штатные средства кеширования данных**

При разработке масштабного проекта одним из самых важных моментов является кеширование. Ruby on Rails в основной комплектации имеет штатные средства кеширования данных. То есть у вас сразу будут в наличии инструменты для кеширования данных на проекте, и вы можете легко кешировать отдельные блоки кода или даже целые страницы.

**Вначале тесты, потом код**

Часто в процессе разработки крупных проектов возникает вопрос о тестировании, и не редкость, когда нет дополнительных средств на отдельную команду тестировщиков. В Rails есть решение и этой проблемы. Если сравнивать RoR с другими фреймворками в контексте тестирования приложения, то вы найдете массу готовых решений для любого вида тестов, будь то интеграционные или юнит. Все эти библиотеки работают «из коробки». В идеале в проекте на Ruby on Rails код не пишется до тех пор, пока под него не написаны тесты. RoR идеология предполагает изначальное использование методов BDD (Behavior Driven Development) или TDD (Test Driven Development).

**Общепринятые стандарты процесса разработки у Ruby-разработчиков**

Говоря о преимуществах Ruby, я не могу снова не упомянуть сообщество рубистов. Оно постоянно растет, развивается и всегда готово прийти на помощь. Всегда есть кто-то, кто подскажет, как лучше решить проблему, поделится опытом в каком-либо вопросе. Также очень важный момент — в Ruby-сообществе уже много лет есть стандарты процесса разработки, некие правила/соглашения сообщества, по которым ведется разработка, что очень сильно упрощает работу. За счет этих стандартов каждый проект очень структурируется, соответственно, новый разработчик в команде быстро войдет в курс дела и уже с первых дней работы сможет быть полезен. И даже больше: если проект начинала одна команда, а заканчивает другая — это тоже совсем не проблема. Поскольку разработка ведется по уже упомянутым правилам и соглашениям сообщества, новая команда быстро и без трудностей вникнет в проект и успешно его закончит без особых потерь времени. Также в Ruby on rails есть большое количество самых разных готовых решений в открытом доступе. Большинство решений уже были реализованы кем-то до вас, а также протестированы сообществом, что уменьшает необходимость разработки с нуля. Это могут быть системы аутентификации, авторизации, комментирования, системы платежей, почтовые рассылки и так далее.

**Готовые решения для многоязычности проекта**

Rails в базовой комплектации имеет очень мощные средства для локализации проекта. Есть возможность как предусмотреть поддержку нескольких языков изначально, так и осуществить её позже. В проекте присутствуют специальные файлы для переводов терминов, инструменты для отображения шаблонов на разных языках и многое другое.

**Высокий уровень защиты данных**

Сейчас нередко в сети публикуются статьи о взломах различных ресурсов. Разработчики фреймворка Ruby on Rails очень серьезно отнеслись к проблеме защиты данных. В RoR изначально присутствует шифрование паролей, данных кредитных карт и других личных данных пользователя, также исключены SQL инъекции и XSS атаки. Все входные параметры экранируются по умолчанию.

### 2.3 Технологии фронтенд разработки

Фронтенд - это набор технологий, которые используются при разработке пользовательского интерфейса веб-приложений и веб-страниц. С помощью передовых технологий разработчики создают дизайн, структуру, анимацию, поведение и все, что мы видим на экране при открытии веб-приложения, веб-сайта или мобильного приложения. Разработка фронтенда всегда была основной частью сети, и технологии фронтенда заметно выросли в последние годы. Кроме того, в связи с растущим спросом на высокопроизводительные веб-приложения и мобильные приложения, компании начали концентрироваться именно на разработке интерфейса. Они стремятся улучшить взаимодействие с пользователем, эффективность, интерактивность и внешний вид своего приложения. Основная цель передовых инструментов и технологий разработки - помочь веб-разработчикам повысить их эффективность и ускорить, упростить и улучшить процесс разработки.

Для разработки интерфейса я использовал HTML и CSS.

HTML (или язык гипертекстовой разметки) - это компьютерный язык, предназначенный для создания веб-сайтов, которые впоследствии могут быть получены каждому, кто имеет доступ к Интернету. HTML обычно используется для структурирования веб-документа. Он определяет такие элементы, как заголовки, абзацы, списки, таблицы, и позволяет встраивать изображения, видео и другие медиафайлы.

HTML состоит из серии шорткодов, называемых тегами, которые преобразованы создателем сайта в текстовый файл. Текст сохраняется в виде файла HTML и просматривается через браузер. Браузер сканирует файл и интерпретирует текст в видимой форме и отображает страницу так, как планировал дизайнер.

Гипертекст - это способ, с помощью которого мы перемещаемся по страницам путем нажатия на гиперссылки.

Текстовая разетка определяет качества, которые теги HTML применяют к тексту внутри них. Теги помечают его как определенный тип текста. Как язык, он содержит кодовые слова и синтаксис, как и любой другой язык.

CSS - это язык каскадных таблиц стилей. Он применяется для определения того, как элементы HTML должны представляться на веб-странице с точки зрения дизайна, макета и вариантов для различных устройств с различными размерами экрана. CSS управляет макетом множества различных веб-страниц одновременно.

CSS взаимодействует с элементами HTML, компонентами веб-страницы. Чтобы общаться с HTML, CSS использует селекторы. Селектор - это часть кода CSS, определяющая, на какую часть HTML повлияет стилизация CSS. Объявление содержит свойства и значения, которые используются селектором. Свойства определяют размер шрифта, цвет и поля.

Внешние таблицы стилей хранятся в виде файлов с расширением .css и могут применяться для определения внешнего вида всего веб-сайта через один файл, вместо того, чтобы помещать дополнительные экземпляры кода CSS в каждый элемент HTML, который необходимо изменить.

Внутренние таблицы стилей - это инструкции CSS, помещаемые прямо в заголовок конкретной

страницы .html. Встроенные стили - это фрагменты CSS, записанные в самом коде HTML.

# 3 ИНСТРУМЕНТАРИЙ

## 3.1 Редактор кода Visual Studio Code

Studio Code — это редактор исходного кода. Он поддерживает ряд языков программирования, подсветку синтаксиса, IntelliSense, рефакторинг, отладку, навигацию по коду, поддержку Git и другие возможности. Многие возможности Visual Studio Code недоступны через графический интерфейс, зачастую они используются через палитру команд или JSON файлы (например, пользовательские настройки). Палитра команд представляет собой подобие командной строки, которая вызывается сочетанием клавиш.

Studio Code был разработан компанией Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации.

## 3.2 Фрэймворк Ruby on Rails

Rails on Rails — это прежде всего среда разработки, которая великолепно подходит для создания любого типа веб-приложений: систем для управления веб-сайтами и платформ для ведения электронной торговли, программ для организации совместной работы и для веб-сервисов для осуществления коммуникации, для учетных и ERP-систем, статистических и аналитических систем.

Основными принципами разработки на Rails являются:

*Принцип DRY (Don’t repeat yourself)* — фреймворк предоставляет механизмы повторного использования программного кода. Это позволяет не только минимизировать дублирование кода, но и повысить скорость разработки.
*Принцип Convention over configuration* — по умолчанию во фреймворке используются многочисленные соглашения по конфигурации, типичные для большинства приложений. Это очень упрощает создание приложений, так как явная спецификация конфигурации требуется только в нестандартных случаях.
*Автоматизированное тестирование* — в составе RoR поставляются средства для проведения полностью автоматического модульного, интеграционного и функционального тестирования, а идеология Ruby on Rails предполагает использование методов разработки через тестирование (TDD — Test Driven Development). Всё это делает разработанные приложения реально надёжными.

Основным преимуществом языка программирования Ruby и фреймворка Ruby on Rails является скорость разработки. На практике скорость разработки проектов на RoR выше на

30-40 процентов по отношению к любому другому языку программирования или фреймворку. Такой прирост скорости разработки объясняется обширным набором готовых к работе штатных инструментов RoR, возможностью использовать готовые решения других разработчиков, ну и, конечно, удобством программирования на Ruby.

### 3.3 База данных SQLite

База данных — это организованная структура, предназначенная для хранения информации. Каждая база данных имеет один или несколько отдельных API для создания, доступа, управления, поиска и репликации данных, которые она содержит.

В настоящее время системы управления реляционными базами данных используются для хранения и управления огромным объемом данных. Это называется реляционной базой данных, поскольку все данные хранятся в разных таблицах, а отношения устанавливаются с использованием первичных или внешних ключей.

SQLite — компактная встраиваемая СУБД.

Слово «встраиваемый» означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

### 3.4 Система контроля версий GIT

**Система контроля версий (Version Control System, VCS) представляет собой программное обеспечение, которое позволяет отслеживать изменения в документах, при необходимости производить их откат, определять, кто и когда внес исправления и т.п.**

Существуют разные типы систем контроля версий. Централизованная система управления версиями требует, чтобы каждый пользователь проверял или синхронизировал файлы с центральным хранилищем при их редактировании. Чаще всего разработчики программного

обеспечения используют распределенные системы контроля версий. Наиболее распространенной считается Git.

Git - это бесплатная распределенная система контроля версий с открытым исходным кодом, предназначенная для быстрой и эффективной работы как с небольшими, так и очень крупными проектами. Git легок в освоении и имеет молниеносную производительность.

Репозиторий Git — это место, где хранится код и вся информация о его изменениях. Репозитории могут находиться на компьютере или к на удалённом сервере.

И в этой ситуации в игру вступают распределённые системы контроля версий (РСКВ). В таких системах как Git, Mercurial, Bazaar или Darcs клиенты не просто выгружают последние версии файлов, а полностью копируют весь репозиторий. Поэтому в случае, когда "умирает" сервер, через который шла работа, любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. Каждый раз, когда клиент забирает свежую версию файлов, он создаёт себе полную копию всех данных.

Кроме того, в большей части этих систем можно работать с несколькими удалёнными репозиториями, таким образом, можно одновременно работать по-разному с разными группами людей в рамках одного проекта. Так, в одном проекте можно одновременно вести несколько типов рабочих процессов, что невозможно в централизованных системах.

Главные преимущества Git по сравнению с другими системами управления версиями:

Распределённая разработка;
Транзакционный подход в управлении пакетами;
Простота управления исходным кодом;
Простота и удобство создания патчей;
Интеграция в VCS апстрима, в том числе и по истории;
Прозрачная сборка как локально, так и на сервере;
Быстрое бэкпортирование;
Малый трафик при постановке на сборку в репозиторий;
Разнообразные проверки собираемых пакетов.

# 4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC

**Архитектурный шаблон** — это общее и повторяющееся решение часто возникающей проблемы архитектуры приложений в пределах заданного контекста. Архитектурные шаблоны схожи с шаблонами программного дизайна, однако имеют более широкий охват.

Шаблон не является законченным образцом проекта, который может быть прямо преобразован в код, скорее это описание или образец для того, как решить задачу, таким образом, чтобы это можно было использовать в различных ситуациях. Объектно-ориентированные шаблоны зачастую показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться. Алгоритмы не рассматриваются как шаблоны, так как они решают задачи вычисления, а не проектирования.

Самые универсальные — *архитектурные шаблоны*, которые можно реализовать практически на любом языке. Они нужны для проектирования всей программы, а не отдельных

её элементов.

MVC расшифровывается как модель-представление-контроллер (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

MVC — это паттерн проектирования веб-приложений, который включает в себя несколько более мелких шаблонов. При использовании MVC на три отдельных компонента разделены модель данных приложения, пользовательский интерфейс и логика взаимодействия пользователя с системой, благодаря чему модификация одного из этих компонентов оказывает минимальное воздействие на остальные или не оказывает его вовсе.

Основная цель применения MVC состоит в разделении данных и бизнес-логики от визуализации. За счет такого разделения повышается возможность повторного использования программного кода: например, добавить представление данных какого-либо существующего маршрута не только в виде HTML, но и в форматах JSON, XML, PDF, XLSX становится очень просто и не требует изменений слоя бизнес-логики исходного маршрута. Также упрощается и сопровождение программного кода: внесение изменений во внешний вид, например, не отражаются на бизнес-логике, а изменения бизнес-логики не затрагивают визуализацию.

Компоненты MVC:

Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.
Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.
Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения.

**Модель**

**Модель отвечает за данные, которые хранятся и обрабатываются на сервере.**

**Представление**

Это HTML-шаблон, который возвращает сервер после обработки запроса. Если запрос корректно обрабатывается, вы получаете веб-страницу со списком друзей. Если запрос некорректный, вы попадаете на страницу ошибки 404.

**Контроллер**

Контроллер обрабатывает входящие запросы. В фреймворке это может заключаться в определении конкретных URL, на которые попадает пользователь при переходе по ссылке или при нажатии кнопки.

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

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

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

Когда вы используете рабочую среду, базовая структура MVC уже подготовлена, и вам остаётся только расширить структуру, размещая ваши файлы в соответствующих директориях для соответствия шаблону MVC. Кроме того, у вас будет набор функций, которые уже написаны и хорошо протестированы.

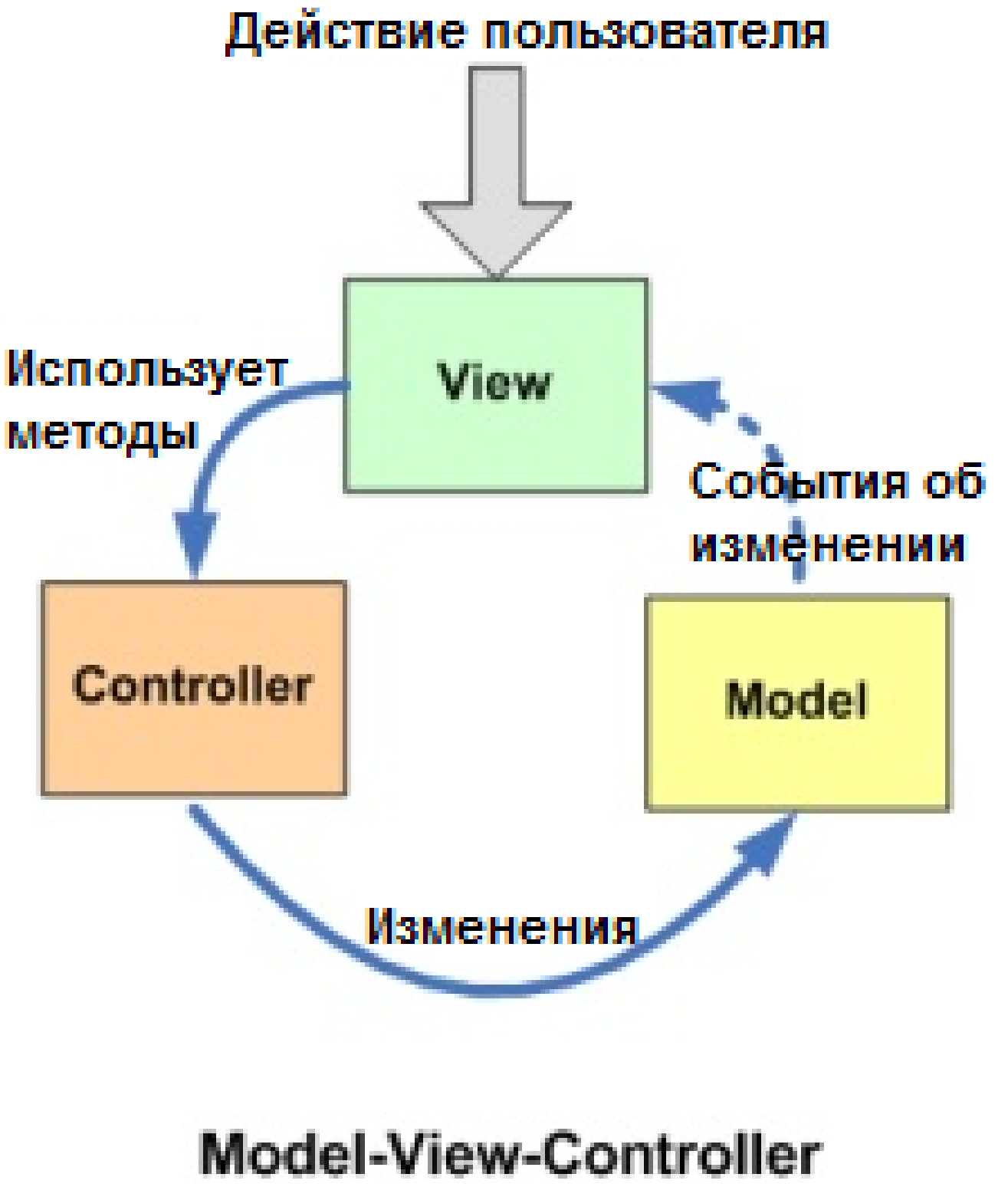


Рисунок 17 - MVC

# 5 ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

Шаблон проектирования, или паттерн, в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования, в рамках некоторого часто возникающего контекста.

Шаблоны бывают следующих трех видов:

1. Порождающие;
2. Структурные;
3. Поведенческие.

Если говорить простыми словами, то это шаблоны, которые предназначены для создания экземпляра объекта или группы связанных объектов

Порождающие шаблоны — шаблоны проектирования, которые имеют дело с процессом создания объектов. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять наследуемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту.

Существуют следующие порождающие шаблоны:

простая фабрика (Simple Factory);
фабричный метод (Factory Method);
абстрактная фабрика (Abstract Factory);
строитель (Builder);
прототип (Prototype);
одиночка (Singleton).

В объектно-ориентированном программировании (ООП), фабрика — это объект для создания других объектов. Формально фабрика — это функция или метод, который возвращает объекты изменяющегося прототипа или класса из некоторого вызова метода, который считается «новым».

**Фабричный метод** — порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Полезен, когда есть некоторая общая обработка в классе, но необходимый подкласс динамически определяется во время выполнения. Иными словами, когда клиент не знает, какой именно подкласс ему может понадобиться.

**Абстрактная фабрика** — порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки). Затем пишутся классы, реализующие этот интерфейс.

Фабрика, которая группирует индивидуальные, но связанные/зависимые фабрики без указания их конкретных классов.

**Строитель** — порождающий шаблон проектирования, который предоставляет способ создания составного объекта. Предназначен для решения проблемы антипаттерна «Телескопический конструктор».

Шаблон позволяет вам создавать различные виды объекта, избегая засорения конструктора. Он полезен, когда может быть несколько видов объекта или когда необходимо множество шагов, связанных с его созданием.

**Прототип** относится к классу порождающих шаблонов. Прототип позволяет копировать объекты, не вдаваясь в подробности их реализации. Он используется для задания вида создаваемых объектов на основе объекта прототипа, от которого происходит передача внутреннего состояния. Он сродни фабричному методу, позволяет избавиться от жесткой привязки к классам, задаёт виды создаваемых объектов с помощью экземпляра-прототипа и создаёт новые объекты путём копирования этого прототипа. Используется когда необходим объект, похожий на существующий объект, либо когда создание будет дороже клонирования.

**Одиночка** — порождающий шаблон проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру. Обеспечивает тот факт, что создаваемый объект является единственным объектом своего класса.

Структурные шаблоны проектирования отвечают за построение удобных в поддержке иерархий классов. Известны следующие шаблоны этой группы:

**Adapter** или Адаптер. Позволяет объектам с несовместимыми интерфейсами работать вместе. Типичным примером использования шаблона Адаптер можно назвать создание классов, приводящих к единому интерфейсу функции языка PHP обеспечивающие доступ к различным СУБД.
**Bridge** или Мост. Разделяет один или несколько классов на две отдельные иерархии — абстракцию и реализацию, позволяя изменять их независимо друг от друга. При частом изменении класса преимущества объектно-ориентированного подхода становятся очень полезными, позволяя делать изменения в программе, обладая минимальными сведениями о реализации программы. Шаблон мост является полезным там, где часто меняется не только сам класс, но и то, что он делает.
**Composite** или Компоновщик. Структурный шаблон проектирования, объединяющий объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково. Паттерн определяет иерархию классов, которые одновременно могут состоять из примитивных и сложных объектов, упрощает архитектуру клиента, делает процесс добавления новых видов объекта более простым.
**Decorator** или Декоратор. Позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».
**Facade** или Фасад. Структурный шаблон проектирования, позволяющий скрыть сложность системы путём сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы. экземпляр в разных местах программы, по факту не является таковым.
**Proxy** или Заместитель. 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.
**Flyweight или приспособленец. Структурный шаблон проектирования, при котором объект, представляющий себя как уникальный экземпляр в разных местах программы, по факту не является таковым.**

Поведенческие шаблоны проектирования

**Chain of Responsibility** или Цепочка обязанностей. Поведенческий шаблон проектирования предназначенный для организации в системе уровней ответственности.
**Command** или Команда. Позволяет вам инкапсулировать действия в объекты. Основная идея, стоящая за шаблоном — это предоставление средств, для разделения клиента и получателя.
**Iterator** или Итератор. Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из агрегированных объектов
**Mediator** или Посредник. Позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс-посредник.
**Memento** или Хранитель. Позволяет сохранять и восстанавливать прошлые состояния объектов, не раскрывая подробностей их реализации.
**Observer** или Наблюдатель. Создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними.
**State** или Состояние. Позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.
**Strategy** или Стратегия. Предназначен для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путём определения соответствующего класса. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.
**Template Method** или Шаблонный метод. Определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Паттерн позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.
**Visitor** или Посетитель. Позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться.

В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем.

# 6 ПРИЛОЖЕНИЕ А

```
class ApplicationController < ActionController::Base

before_action :configure_permitted_parameters, if: :devise_controller?
```

```
    protected
    def configure_permitted_parameters
    devise_parameter_sanitizer.permit(:sign_up) {|u|
    u.permit(:username,:email,:password,:password_confirmation,:roll,:about) }
    devise_parameter_sanitizer.permit(:sign_in) {|u|
    u.permit(:username,:email,:password,:password_confirmation,:roll,:about) }

    end

    end


  class BoughtsController < ApplicationController

def new
@wains = Wain.where(user_id: current_user.id)
@bought = current_user.boughts.build
end



def create
@bought = current_user.boughts.build(bought_params)
@bought.save
redirect_to users_page_path(current_user)
end
def destroy

@bought = Bought.find(params[:id])
@bought.destroy
redirect_to users_page_path(current_user)
end



private def bought_params
params.require(:bought).permit(:name,:number_tel,:street,:payment,:deliver,:last_name,:email,:user
name)
end



end


  class CategoriesController < ApplicationController
```

```
before_action :authenticate_user!, except:[:index,:show]
before_action :set_category ,only: [:show,:edit,:update,:destroy]

def index
@categories=Category.all
end

def new
@category=Category.new
end

def show
end

class BoughtsController < ApplicationController

def new
@wains = Wain.where(user_id: current_user.id)
@bought = current_user.boughts.build
end



def create
@bought = current_user.boughts.build(bought_params)
@bought.save
redirect_to users_page_path(current_user)
end
def destroy

@bought = Bought.find(params[:id])
@bought.destroy
redirect_to users_page_path(current_user)
end



private def bought_params
params.require(:bought).permit(:name,:number_tel,:street,:payment,:deliver,:last_name,:email,:user
name)
end
```

```
end

def edit
end

def update
if @category.update_attributes(category_params)
redirect_to categories_path
else
render :edit
end
end

def create
@category = Category.new(category_params)

if @category.save
redirect_to @category
flash[:success] = "Успешно"
else
render :new
end
end

def destroy
@category.destroy
redirect_to categories_path
end

private def category_params
params.require(:category).permit(:name)
end
def set_category
@category=Category.find(params[:id])
end

end


    class CharacteristicsController < ApplicationController
before_action :authenticate_user!, except:[:index,:show]
```

```
before_action :set_product, only: [:create, :update, :destroy]
def index

end

def new
@characteristic=Characteristic.new
end



def show
@product = Product.find(params[:id])
@characteristic = Characteristic.where(product_id: @product.id)
end

def createclass CategoriesController < ApplicationController
before_action :authenticate_user!, except:[:index,:show]
before_action :set_category ,only: [:show,:edit,:update,:destroy]

def index
@categories=Category.all
end

def new
@category=Category.new
end

def show
end

class BoughtsController < ApplicationController

def new
@wains = Wain.where(user_id: current_user.id)
@bought = current_user.boughts.build
end



def create
@bought = current_user.boughts.build(bought_params)
```

```
@bought.save
redirect_to users_page_path(current_user)
end
def destroy

@bought = Bought.find(params[:id])
@bought.destroy
redirect_to users_page_path(current_user)
end



private def bought_params
params.require(:bought).permit(:name,:number_tel,:street,:payment,:deliver,:last_name,:email,:user
name)
end



end

def edit
end

def update
if @category.update_attributes(category_params)
redirect_to categories_path
else
render :edit
end
end

def create
@category = Category.new(category_params)

if @category.save
redirect_to @category
flash[:success] = "Успешно"
else
render :new
end
end
```

```
def destroy
@category.destroy
redirect_to categories_path
end

private def category_params
params.require(:category).permit(:name)
end
def set_category
@category=Category.find(params[:id])
end

end

@characteristic = @product.characteristics.create(characteristic_params)
if @product.save
redirect_to @product
else
render :new
end
end

def destroy

@characteristic = Characteristic.find(params[:id])
@characteristic.destroy
redirect_to @product
end


def set_product
@product = Product.find(params[:product_id])
end

def characteristic_params
params.require(:characteristic).permit(:name,:about)
end

end
```

```
    class ProductsController < ApplicationController
before_action :authenticate_user!, except:[:index,:show]

def index
@products=Product.where(["name LIKE ?","%#{params[:search]}%"]).order(params[:sort])
end



def show
@product=Product.find(params[:id])
end

def new

@product = Product.new
end
def edit
@product=Product.find(params[:id])
endclass CharacteristicsController < ApplicationController
before_action :authenticate_user!, except:[:index,:show]
before_action :set_product, only: [:create, :update, :destroy]
def index

end

def new
@characteristic=Characteristic.new
end



def show
@product = Product.find(params[:id])
@characteristic = Characteristic.where(product_id: @product.id)
end

def createclass CategoriesController < ApplicationController
before_action :authenticate_user!, except:[:index,:show]
before_action :set_category ,only: [:show,:edit,:update,:destroy]

def index
```

```
@categories=Category.all
end

def new
@category=Category.new
end

def show
end

class BoughtsController < ApplicationController

def new
@wains = Wain.where(user_id: current_user.id)
@bought = current_user.boughts.build
end



def create
@bought = current_user.boughts.build(bought_params)
@bought.save
redirect_to users_page_path(current_user)
end
def destroy

@bought = Bought.find(params[:id])
@bought.destroy
redirect_to users_page_path(current_user)
end



private def bought_params
params.require(:bought).permit(:name,:number_tel,:street,:payment,:deliver,:last_name,:email,:user
name)
end



end

def edit
```

```
end

def update
if @category.update_attributes(category_params)
redirect_to categories_path
else
render :edit
end
end

def create
@category = Category.new(category_params)

if @category.save
redirect_to @category
flash[:success] = "Успешно"
else
render :new
end
end

def destroy
@category.destroy
redirect_to categories_path
end

private def category_params
params.require(:category).permit(:name)
end
def set_category
@category=Category.find(params[:id])
end

end

@characteristic = @product.characteristics.create(characteristic_params)
if @product.save
redirect_to @product
else
render :new
end
```

```
end

def destroy

@characteristic = Characteristic.find(params[:id])
@characteristic.destroy
redirect_to @product
end


def set_product
@product = Product.find(params[:product_id])
end

def characteristic_params
params.require(:characteristic).permit(:name,:about)
end

end


def update
@product=Product.find(params[:id])
if @product.update_attributes(product_params)
redirect_to @product
else
flash[:error] = "Ошибка"
render :edit
end
end



def create
@product = Product.new(product_params)

if @product.save
redirect_to @product
flash[:success] = "Успешно"
else
flash[:error] = "Ошибка"
```

```
render :new
end
end


def destroy
@product=Product.find(params[:id])
@product.destroy
redirect_to new_search_path

end


private def product_params
params.require(:product).permit(:name,:body,:picture,:about,:cost,:url,:category_id,:image)
end




end


    class SearchesController < ApplicationController

def new
@products=Product.where(["name LIKE ?","%#{params[:search]}%"]).order(params[:sort])
@search=Search.new

end
def create
@search=Search.create(search_params)
redirect_to @search
end
def show
@search=Search.find(params[:id])
end
private
def search_params
params.require(:search).permit(:keywords,:min_price,:max_price,:category_id)
end
```

```
end


    class UsersPageController < ApplicationController
def index
@users=User.all
end

def show

@user = User.find(params[:id])
@boughts = Bought.where(user_id: @user.id)
end



def destroy
if current_user.roll =="admin" || current_user.username == @user.username
@user=User.find(params[:id])
@user.destroy
redirect_to new_search_path
else
redirect_to new_search_path
end
end

def edit
@user=User.find(params[:id])
end

def update
@user=User.find(params[:id])
if current_user.roll =="admin" || current_user.username == @user.username
if @user.update_attributes(user_params)
redirect_to users_page_path
else
render :edit
end
else
redirect_to users_page_path
end
end
```

```
private def user_params
params.require(:user).permit(:email,:username,:roll)
end
end


    class WainsController < ApplicationController

before_action :set_post, only: [:new,:create, :update,:edit,:show]

def index

@wains = Wain.where(user_id: current_user.id)

end
def new

@wain = current_user.wains.build
end

def create
@wain = current_user.wains.build(wain_params)
@wain.save
redirect_to wains_path
end

def destroy

@wain=Wain.find(params[:id])
@wain.declass UsersPageController < ApplicationController
def index
@users=User.all
end

def show

@user = User.find(params[:id])
@boughts = Bought.where(user_id: @user.id)
end
```

```
def destroy
if current_user.roll =="admin" || current_user.username == @user.username
@user=User.find(params[:id])
@user.destroy
redirect_to new_search_path
else
redirect_to new_search_path
end
end

def edit
@user=User.find(params[:id])
end

def update
@user=User.find(params[:id])
if current_user.roll =="admin" || current_user.username == @user.username
if @user.update_attributes(user_params)
redirect_to users_page_path
else
render :edit
end
else
redirect_to users_page_path
end
end


private def user_params
params.require(:user).permit(:email,:username,:roll)
end
end
stroy
redirect_to wains_path

end

private def wain_params
params.require(:wain).permit(:name,:count,:id_product,:image,:cost)
end
```

```
def set_post
@product = Product.find(params[:product_id])
end

end
```

ПОЛНОЦЕННЫЙ код программы находится по ссылке: https://github.com/big-qqq/shopoop

## Заключение

Многие пользователи интернета давно оценили преимущества интернет-магазина. Ведь посещение такого магазина избавляет вас от давней проблемы – тратить время на многочасовые походы по торговым центрам. Часто на такой шопинг уходил единственный выходной. Теперь же есть замечательная возможность делать покупки, не отходя от компьютера. Можно не спеша рассмотреть товар, подробно прочитать об его характеристиках, сравнить цены на разных сайтах и сделать свой выбор. Пользователь может заказать товар из другого города и даже страны. Да, зайти в интернет-магазин вы можете в любое время суток. Вот зачем пользователям нужен интернет-магазин. Открытие такого магазина удачное решение не только для покупателей, но и для продавцов. Нет необходимости арендовать торговую площадь и набирать персонал. При этом, если магазин грамотно раскручен, то посещать его будет гораздо больше людей, чем посещали бы реальный. В результате выполнения курсовой работы были изучены материалы, связанные с разработкой веб-приложений на фреймворке Ruby on Rails, с помощью объектно-ориентированного шаблона проектирования MVC. В ходе работе был разработан веб-сайт по продаже канцелярских товаров. Код моего интернет-магазина написан на языке Ruby, с помощью языка текстовой разметки HTML и каскадных таблиц стилей CSS. В результате работы был создан удобный пользовательский интерфейс, понятный любому обывателю интернета.

## Список использованных источников


1. [url] **Разработка интернет-магазина канцелярских товаров**
2. [url] **ERUD.BY** erud.by/object_orient_program
3. [url] **Tproger** tproger.ru
4. [url] **Wikipedia** ru.wikipedia.org/wiki/SQLite
5. [url] **Ruby on Rils по-русски** rusrails.ru/
6. [url] **SQL** hostinfo.ru/


## Приложения

1. [электронный документ] 5ebdabdc4c9e1_ZAPISKA.docx
2. [электронный документ] 5ebdac349ce4a_zadanie.docx
3. [электронный документ] 5ebdae4a64e44_polz_interf.docx