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
27.06.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

РАЗРАБОТКА ИНТЕРНЕТ-МАГАЗИНА КОСМЕТИКИ

БГУИР КР 1-40 05 01-10 № 143 ПЗ

Студент (подпись студента) Д.Д.Кириенко

Курсовая работа
представлена на проверку
27.06.2025
_____
(подпись студента)

Минск 2025

Реферат

БГУИР КР 1-40 05 01-10 № 143 ПЗ, гр. 814302

Д.Д.Кириенко, РАЗРАБОТКА ИНТЕРНЕТ-МАГАЗИНА КОСМЕТИКИ, Минск: БГУИР - 2025.

Пояснительная записка 117047 с., 17 рис., 0 табл.

Ключевые слова: интернет-магазин, Ruby, Ruby on Rails, HTML, CSS

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

Предмет: создание веб-приложения Объект: шаблоны проектирования, разработка пользовательского интерфейса. Цель: верстка и программирование интернет-магазина косметики с использованием архитектурного шаблона проектирования MVC. Методология проведения работы: в процессе решения поставленных задач спроектирован и разработан простой и удобный интерфейс веб-приложения, разработана логика приложения с использованием паттерна MVC, изучена и применена работа с базой данных. Результаты работы: разработан интернет-магазин с удобным функционалом. Интерфейс сайта был максимально упрощен и в меру информативен. Область применения результатов: удовлетворение пользователей Интернета, нуждающихся в заказе косметики, не выходя из дома, а также для оптимизации их времени. Ссылка на онлайн-репозиторий GitHub: https://github.com/danusyaaaa/online-shop

Subject: creating a web application Object: design patterns, user interface development. Purpose: layout and programming of the online cosmetics store using the MVC architectural design pattern. Methodology of the work: in the process of solving the tasks, a simple and convenient web application interface was designed and developed, the application logic using the MVC pattern was developed, the database was studied and applied. Results: an online store with convenient functionality was developed. The site interface was maximally simplified and reasonably informative. Scope of the results: the satisfaction of Internet users who need to order cosmetics, without leaving home, as well as to optimize their time. Link to the GitHub online repository: https://github.com/danusyaaaa/online-shop

Содержание

Введение
1 ОПИСАНИЕ ПРОЕКТА
2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ
3 ИНСТРУМЕНТАРИЙ
4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC
5 ШАБЛОН ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ
6 ПРИЛОЖЕНИЕ А
Заключение
Список использованных источников
Приложения

Введение

Невозможно представить жизнь современного человека без Интернета. Если раньше им могло пользоваться только небольшое количество людей, то сейчас доступ открыт всем. Интернет представляет собой необходимый инструмент, помогающий в работе, общении и отдыхе. Миллионы сайтов позволяют человеку, не отходя от компьютера, совершать сотни действий. Уже давно стало привычкой узнавать последние новости и прогноз погоды в Интернете. Электронная почта давно потеснила почту привычную, да и платежные системы в сети становятся все более популярными. Благодаря интернету, человек может свободно совершать покупки в различных магазинах мира, заказать билеты на самолет, в театр или на выставку, спланировать путешествие, не выходя из дома. У любой современной компании существует сайт. В сегодняшнем мире это элемент необходимости, ведь именно в Интернете потенциальные клиенты будут в первую очередь искать информацию о фирме. Актуальность создания сайта состоит также в том, что это наиболее удобный способ донести информацию максимально быстро до большого количества людей. Веб-ресурс позволяет представить информацию о компании и ее товарах или услугах сжато и одновременно полноценно. Также сайт может сообщать о новостях фирмы, об изменениях в прайсе или режиме работы, содержать отзывы благодарных клиентов. Актуальность разработки сайта объясняется следующими факторами: • Быстрота подачи информации широкому кругу лиц; • Улучшение имиджа компании и повышение ее популярности; • Возможность организовать обратную связь с клиентами; • Организация маркетинговых исследований; • Реклама и привлечение покупателей и клиентов; • Увеличение трафика. В курсовой работе поставлена цель: проектирование интернет-магазина косметики с использованием архитектурного шаблона MVC на фреймворке Ruby on Rails. Для достижения поставленной цели необходимо решить следующие задачи: 1 Проектирование пользовательского интерфейса. 2 Построение логической модели приложения. 3 Верстка приложения по разработанному дизайну. 4 Программирование бэкенда приложения.

1 ОПИСАНИЕ ПРОЕКТА

1. **Клиентская часть**

Клиентская часть реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него. В веб-разработке «сторона клиента» относится ко всему в веб-приложении, что отображается или происходит на устройстве конечного пользователя. Это включает в себя то, что видит пользователь, например, текст, изображения, формы, кнопки и остальную часть пользовательского интерфейса, а также любые действия, которые приложение выполняет в браузере пользователя. Языки разметки, такие, как HTML и CSS, интерпретируются браузером на стороне клиента. Клиентская сторона также известна как фронтенд, хотя эти два термина не означают одно и то же. Клиентская сторона относится исключительно к месту, где выполняются процессы, в то время как фронтенд относится к видам процессов, которые выполняются на стороне клиента.

Несомненно, важным компонентом любого приложения является пользовательский интерфейс, так называемая оболочка – то, как выглядит приложение и насколько удобным для

использования оно является. Классическим и наиболее популярным методом создания веб-интерфейсов является использование HTML с применением CSS.

Пользовательский интерфейс разрабатываемого интернет-магазина написан на этом языке текстовой разметки с использованием каскадных таблиц стилей.

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. HTML используется 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.

Каскадные таблицы стилей (CSS) - это язык таблиц стилей, используемый для описания представления документа, написанного на языке разметки HTML. CSS предназначен для разделения представления и содержимого, включая макет, цвета и шрифты. Такое разделение может улучшить доступность контента, обеспечить большую гибкость и контроль в спецификации характеристик представления, дать возможность нескольким веб-страницам совместно использовать форматирование, указав соответствующий CSS в отдельном файле, а также уменьшить сложность и повторяемость структурного контента. CSS также имеет правила для альтернативного форматирования, если доступ к контенту осуществляется на мобильном устройстве. Каскадирование имен происходит из указанной схемы приоритетов, чтобы определить, какое правило стиля применяется, если конкретному элементу соответствует несколько правил.

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.

На главной странице сайта мы видим новинки и хиты продаж, а также бар с меню, который закреплен для эффективной навигации по страницам интернет-магазина (см. рис.1.).

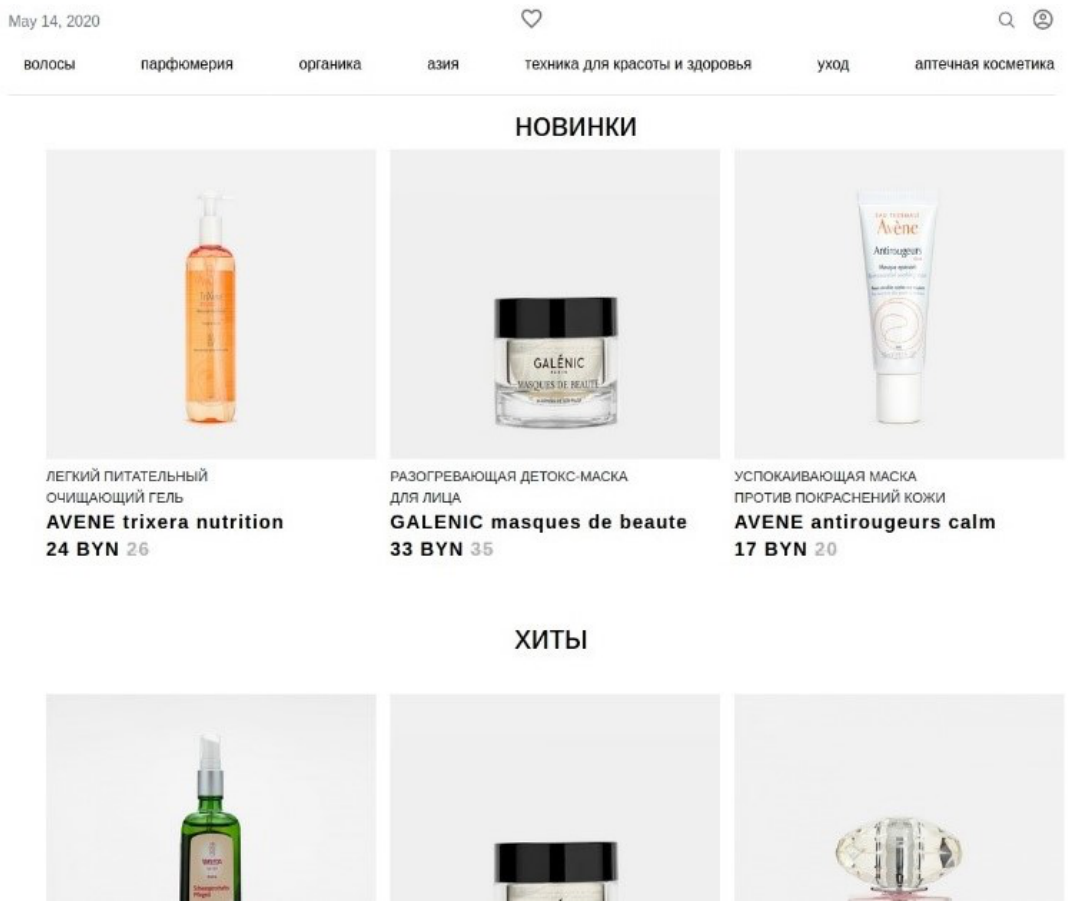

Рисунок 1 – Главная страница интернет-магазина

Для просмотра сайта не обязательно быть авторизованным. Авторизация необходима для дальнейшего заказа товаров. При нажатии на иконку «Личный кабинет», если пользователь не авторизован или не зарегистрирован, его приветствует страница авторизации с полем «Войдите в систему» (см. рис.2) или кнопка «Создайте аккаунт», после нажатия на которую открывается страница регистрации (см. рис.3). После прохождения авторизации или регистрации URL-ы для работы с корзиной становятся доступными.

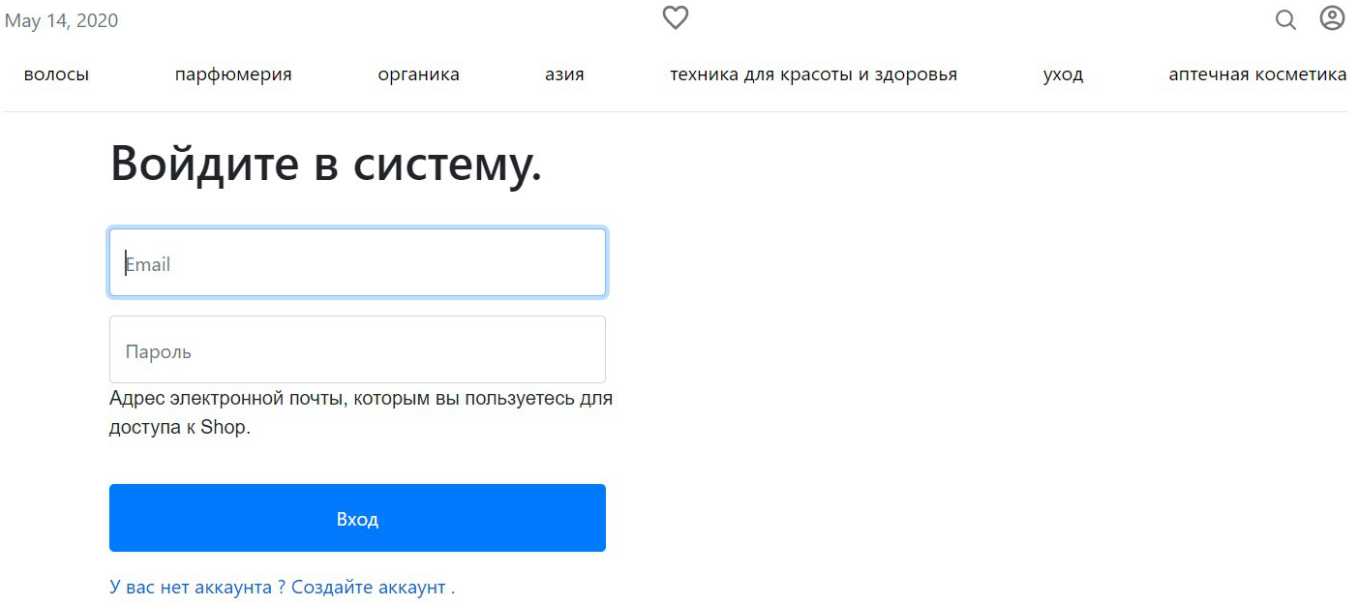

Рисунок 2 - Страница для выполнения авторизации

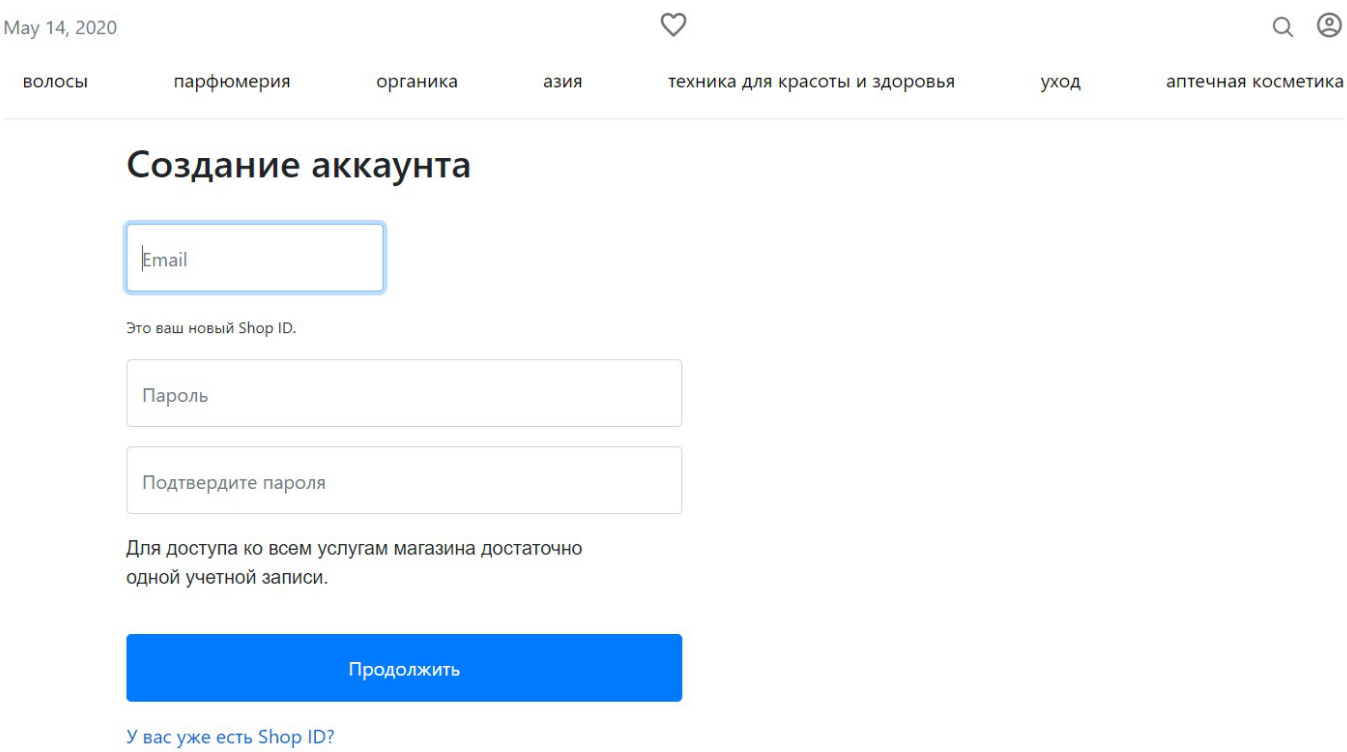

Рисунок 3 - Страница регистрации

После успешной авторизации пользователь попадает на главную страницу интернет-магазина, где он может продолжить дальнейшее изучение представленного товара.

Рассмотрим некоторые разделы сайта.

Например, одна из представленных категорий – «Органика».

При нажатии на эту категорию в баре, мы попадаем на отдельную html-страничку, на которой располагаются товары органической косметики (см. рис.4).

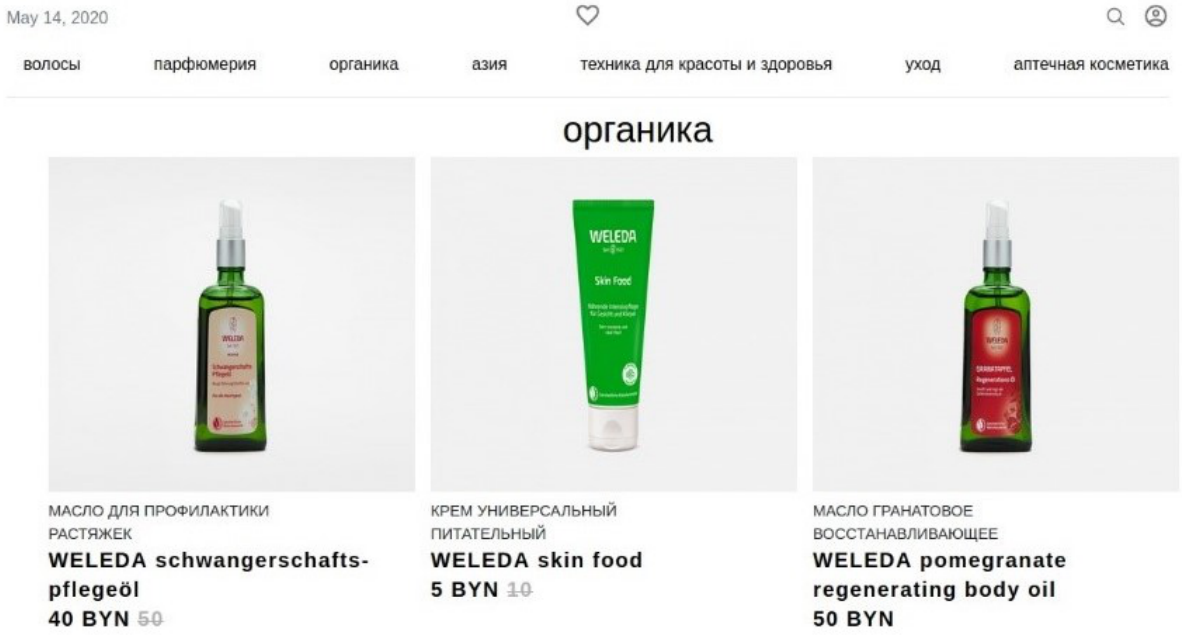

Рисунок 4 – Страница категории «Органика»

Для более детального просмотра какого-то продукта, мы можем нажать на его название и

перейти на страницу описания (см. рис.5). На этой странице мы видим фотографию товара, его название, объём, цену, описание.

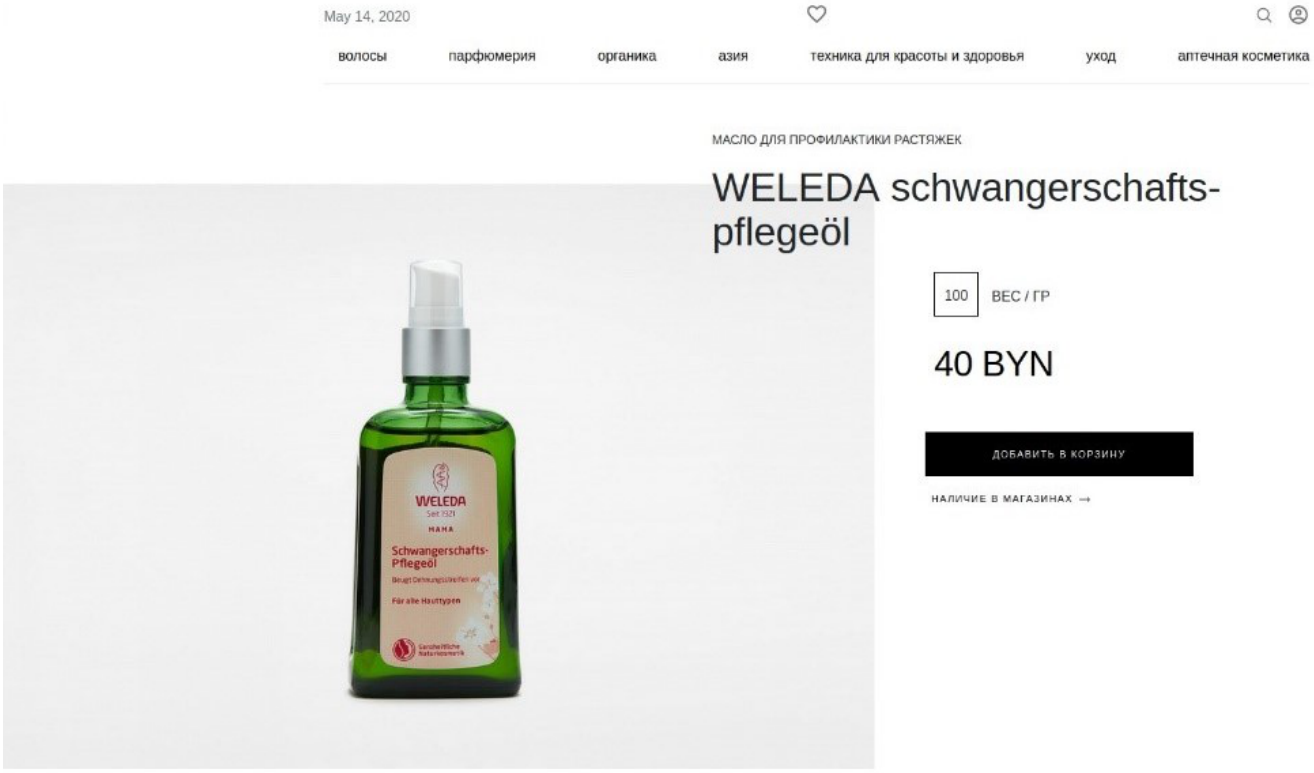

Рисунок 5 – Страница описания товара

В свою очередь, при переходе к описанию товара, пользователь также имеет возможность увидеть товары, которые могли бы его заинтересовать, что очень удобно (см. рис.6).

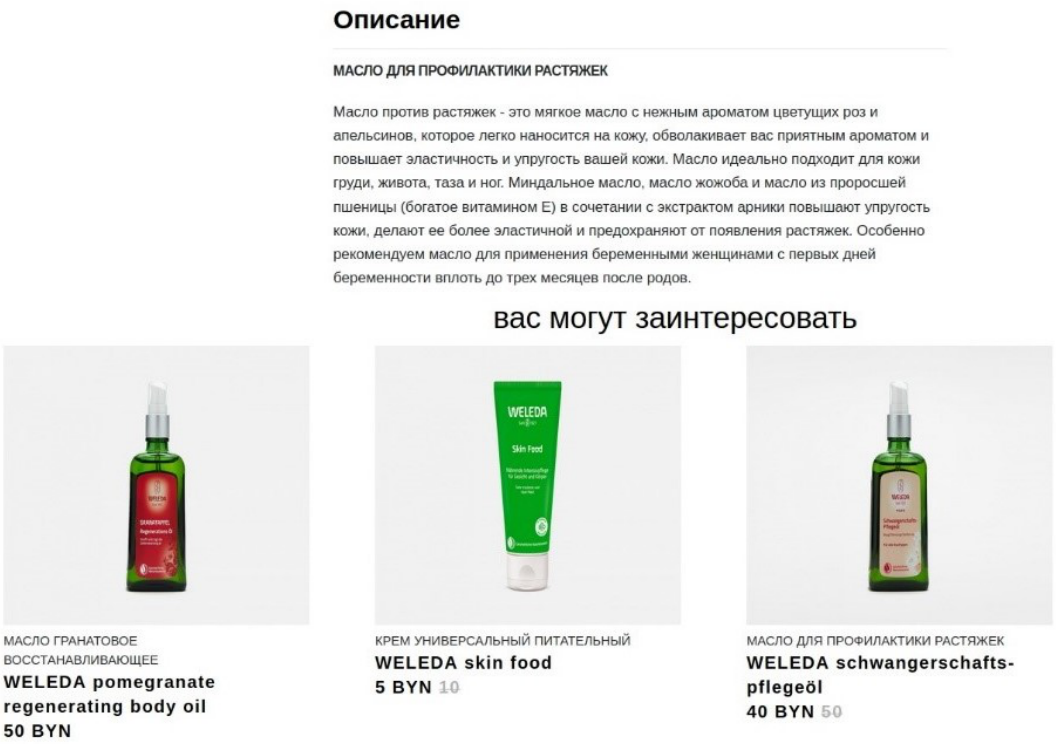

Рисунок 6 – Товары, которые могут заинтересовать

Также на странице описания есть кнопка «Добавить в корзину», при нажатии на которую пользователя переадресовывает на html-страницу, где он выбирает необходимое количество товара (см. рис.7).

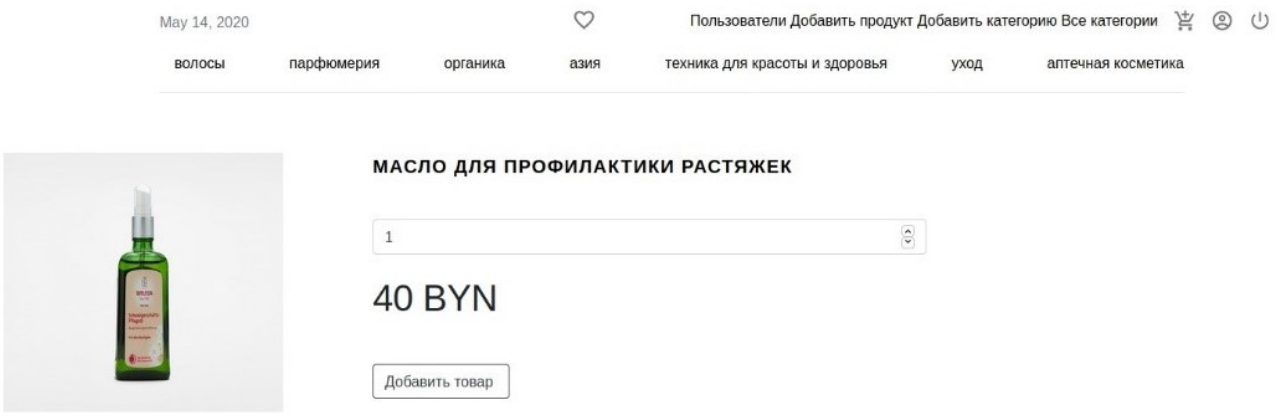

Рисунок 7 – Страница добавления товара в корзину

После добавления товара мы попадаем на страницу Корзины, где находятся все выбранные пользователем товары. Здесь мы видим: название продукта, его количество, конечная цена за несколько единиц продукта, а также итоговая сумма всего заказа. Ещё в Корзине предусмотрена возможность удаления (см.рис.8).

Рисунок 8 – Страница корзины

Далее мы переходим к следующему этапу – оформление заказа. На этой странице

пользователю необходимо внести личные данные. Для удобства также справа находятся товары, которые заказываются (см.рис.9).

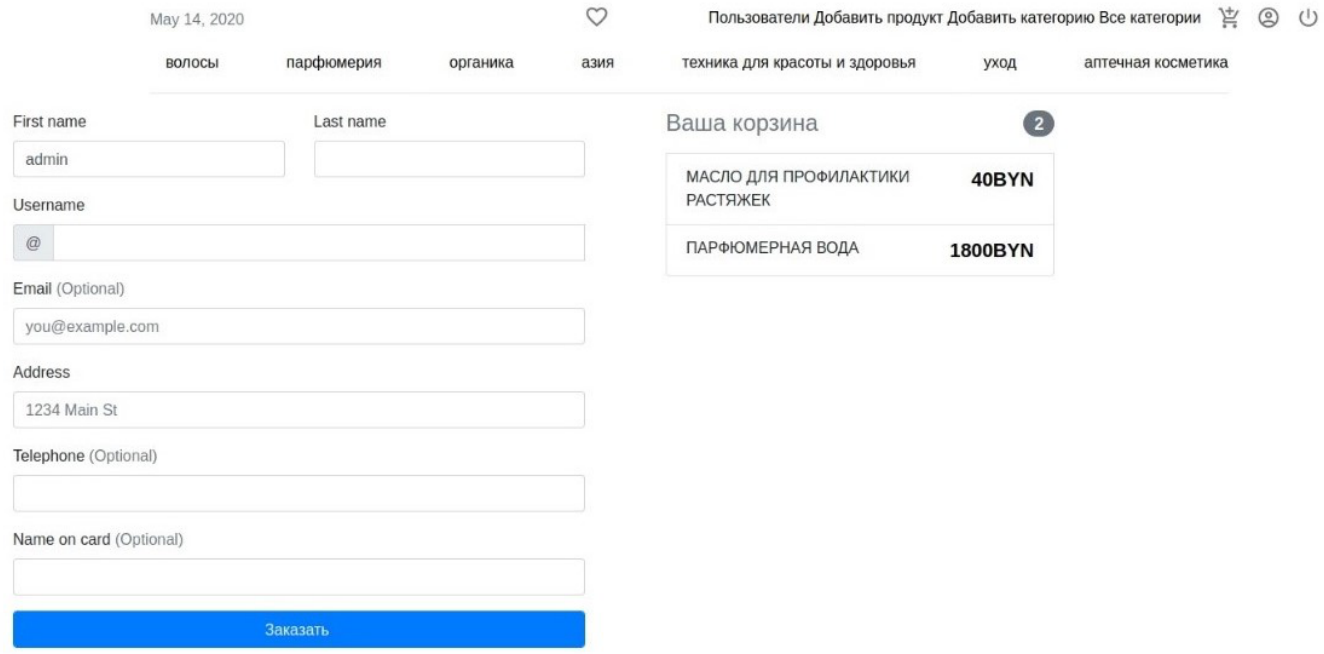

Рисунок 9 – Страница оформления заказа

После оформления заказа мы попадаем в личный кабинет, где и отображается список покупок с товарами, которые были куплены этим пользователем (см.рис.10). В свою очередь, на странице создана возможность удаления заказа.

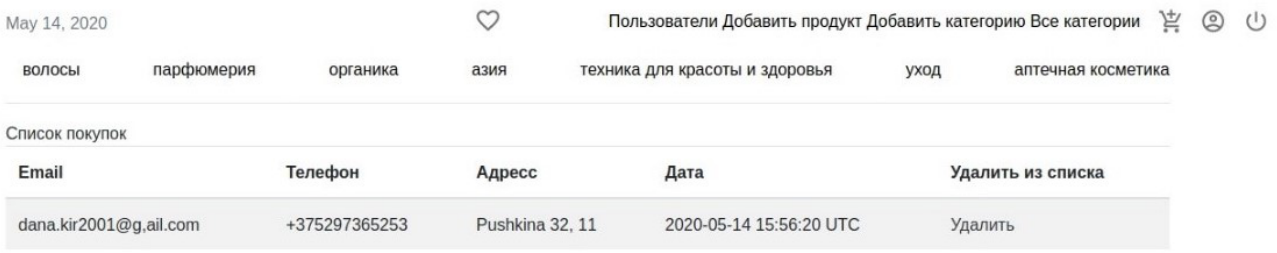

Рисунок 10 – Личный кабинет

На всех страницах бар с меню закреплен для эффективной навигации по страницам. Он представляет собой совокупность категорий, а также после авторизации появляются иконки корзины, личного кабинета и выхода. (см.рис.11).

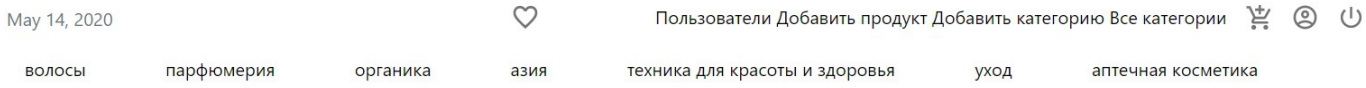

Рисунок 11 – Навигационный бар

Следует отметить, что в системе разграничены права пользователей и рассматриваются 2 группы пользователей, которые должны пройти регистрацию: администратор и покупатель. К статистической информации доступ имеет только покупатель. При авторизации админа

становятся активными такие ссылки как «Пользователи», «Добавить категорию», «Все категории», «Добавить продукт».

Если нажать на ссылку «Пользователи», то мы попадём на html-страницу, где отображаются все пользователи, которые зарегистрированы на сайте (см. рис.12). При разработке также была предусмотрена возможность редактирования аккаунта (см.рис.13), при этом админ может изменять роль пользователя, то есть, он может присвоить роль админа, а также и вовсе удалить аккаунт.

Email	Username	Role	Redaction	Destroy
dana.kir2001@gmail.com	admin	admin	Edit	×
illya.qwerty23@gmail.com	Dread	user	Edit	×
nikita.rassolov228@gmail.com	Nikita	user	Edit	×

Рисунок 12 – Пользователи

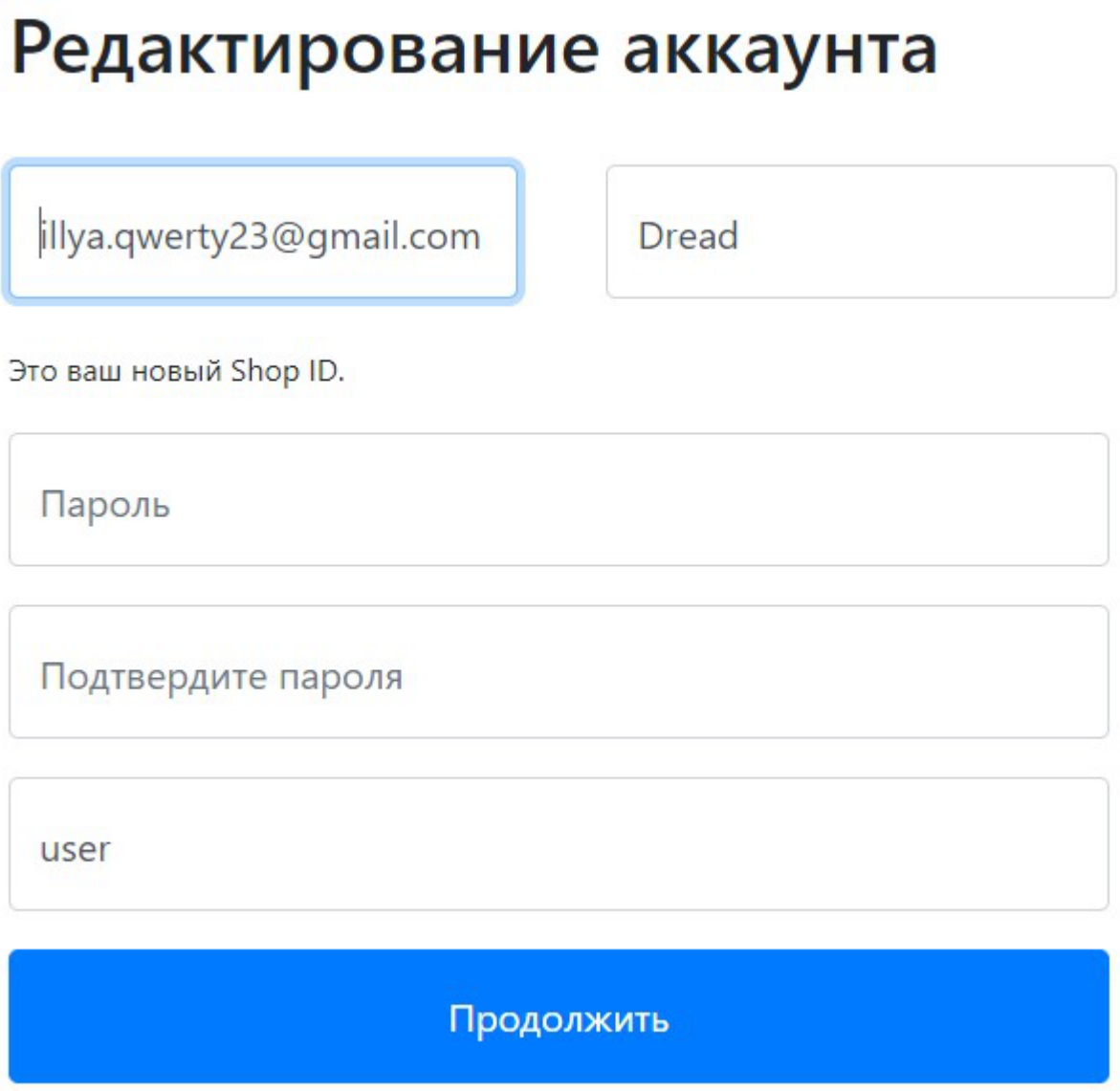

Рисунок 13 – Страница редактирования аккаунта

В свою очередь, при нажатии на ссылку «Добавить продукт», нас перенаправит на форму

для добавления товара (см.рис.14).

Рисунок 14 – Форма добавления продукта

Если же кликнуть на ссылку «Добавить категорию», мы окажемся на форме добавления категории. Для удобства справа находится блок со всеми уже существующими категориями (см.рис.15). При нажатии на ссылку «Все категории», нас перенаправит на html-страницу со всеми уже существующими категориями (см.рис.16).

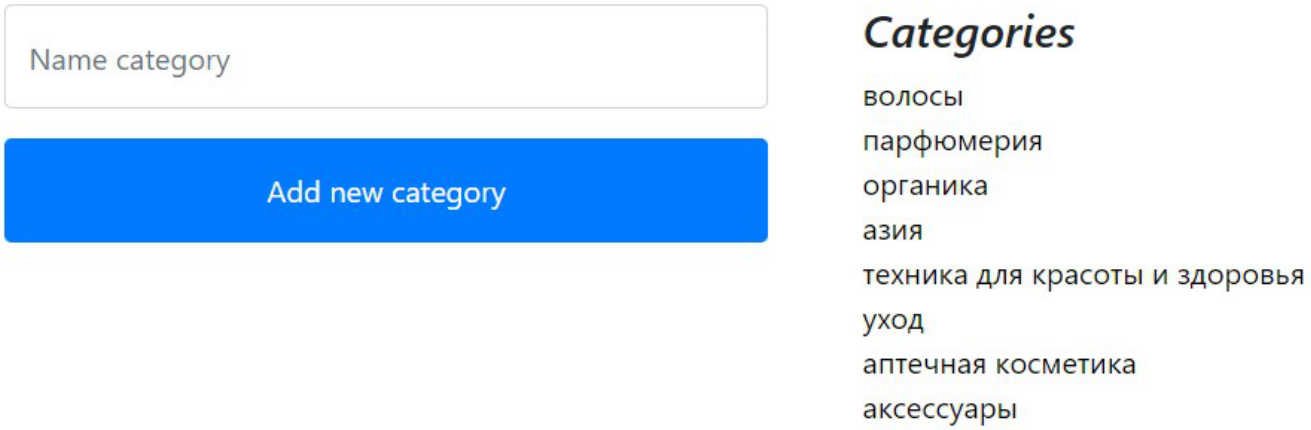

Рисунок 15 –Добавление категорий

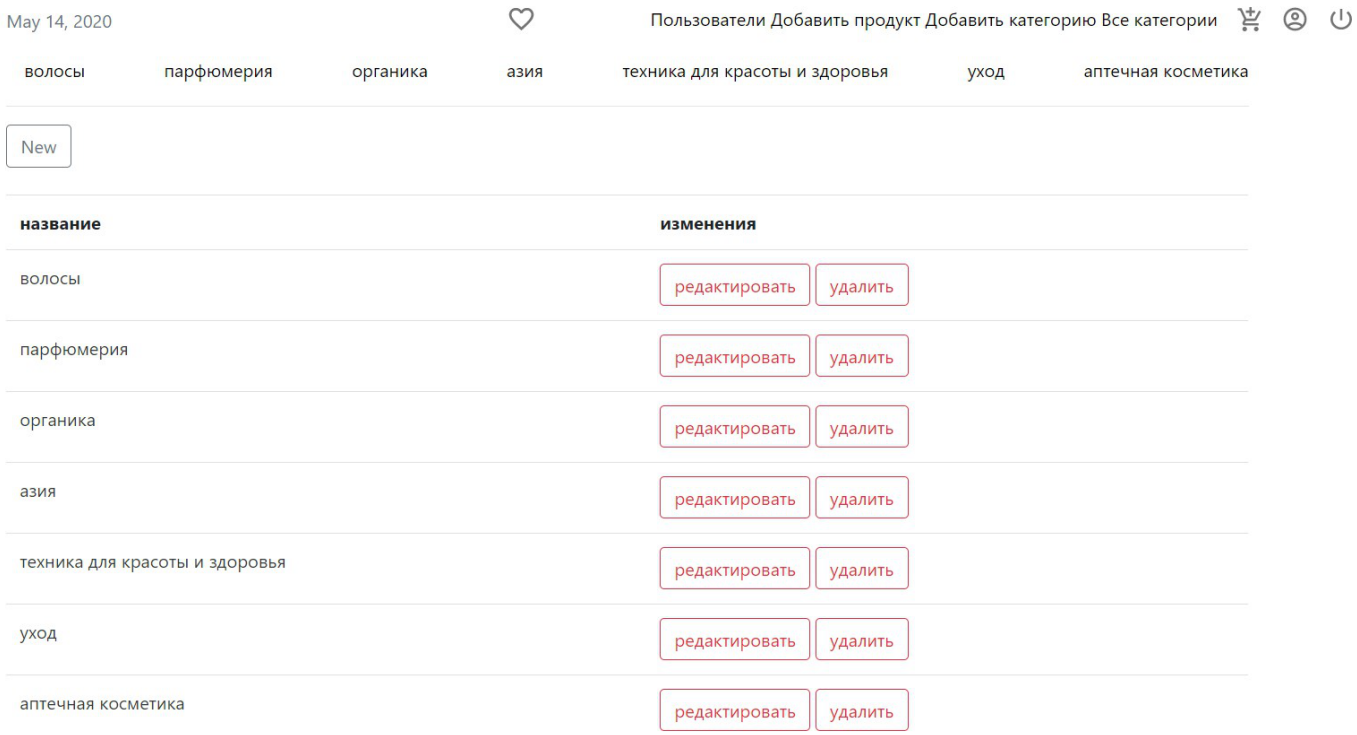

Рисунок 16 – Страница всех категорий

Благодаря серверной части пользовательские данные не удаляются после завершения сеанса, а сохраняются в базе данных на сервере.

1. **Серверная часть**

Веб-браузеры взаимодействуют с веб-серверами при помощи протокола передачи гипертекста (HTTP). Когда вы кликаете на ссылку на странице, заполняете форму или запускаете поиск, браузер отправляет на сервер HTTP-запрос.

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Аббревиатура HTTP расшифровывается как HyperText Transfer Protocol, «протокол передачи гипертекста». В соответствии со спецификацией OSI, HTTP является протоколом прикладного (верхнего, 7-го) уровня.

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту. После этого клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом.

Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером. На данный момент именно благодаря протоколу HTTP обеспечивается работа Всемирной паутины.

Как правило, передача данных по протоколу HTTP осуществляется через TCP/IP-соединения. Серверное программное обеспечение при этом обычно использует TCP-порт 80 (и, если порт не указан явно, то обычно клиентское программное обеспечение по умолчанию использует именно 80-й порт для открываемых HTTP-соединений), хотя может использовать и любой другой.

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. Фрэймворк – это рабочая среда, которая помогает разработчику быстро и качественно создавать программный продукт, не отвлекаясь на мелочи.

Ruby on Rails (RoR) — фреймворк, написанный на языке программирования Ruby, реализует архитектурный шаблон Model-View-Controller для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных. Является открытым программным обеспечением и распространяется под лицензией MIT.

Базируется на следующих принципах разработки приложений:

> максимальное использование механизмов повторного использования, позволяющих минимизировать дублирование кода в приложениях (принцип Don’t repeat yourself);
> по умолчанию используются соглашения по конфигурации, типичные для большинства приложений (принцип Convention over configuration).

В RoR сразу после установки есть сервисы аутентификации, а также он обеспечивает простой способ авторизации пользователя. Для этого необходимо всего лишь подключить такой gem как Devise, после чего будут сразу созданы такие представления как: registration, sessions, то есть без особого вмешательства разработчика уже будут готовы страницы регистрации, авторизации и редактирования аккаунта.

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.

Контроллер в Rails — это набор логики, запускаемой после получения HTTP-запроса сервером. Контроллер отвечает за вызов методов модели и запускает формирование представления. Отличительная черта Rails заключается в том, что они предоставляют

возможность создавать контроллеры и модели не только вручную, но и с помощью команды терминала rails g controller(model). Все классы контроллеров хранятся в отдельных файлах в папке app/controllers и наследуются от класса ApplicationController. В курсовом проекте представлены такие контроллеры как: users_page_controller, products_controller, boughts_controller, categories_controller, characteristics_controller, wains_controller.

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. Это как раз-таки то, что отображает всю клиентскую часть, и помогает пользователю эффективно взаимодействовать с серверной частью.

Никакое веб-приложение практически невозможно построить без использования базы данных, где хранятся данные пользователей. RoR поддерживает такие БД как: MySQL, PostgreSQL, Firebird, DB2, Oracle и Microsoft SQL Server, а также SQLite.

При разработке сайта интернет-магазина использовалась база данный SQLite. Основные таблицы из базы данных приложения: products, boughts, categories, users и др. Каждой из таблиц соответствует модель в приложении: product, bought, category, user. Модели позволяют запрашивать данные из таблиц, а также вставлять новые записи в таблицу.

2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

2.1 Объектно-ориентированное программирование

Объектно-ориентированное программирование, или ООП, - это парадигма программирования, которая концентрирует основное внимание программиста на связях между объектами, а не на деталях их реализации. Объектно-ориентированная парадигма программирования разрабатывалась как попытка связать поведение сущности с её данными и спроецировать объекты реального мира и бизнес-процессов в программный код.

Объект – это абстрактная сущность, наделенная характеристиками объектов окружающего нас реального мира. Создание объектов и манипулирование ими – результат методологии программирования, воплощающей в кодовых конструкциях описания объектов и операции над ними. Каждый объект программы, как и любой реальный объект, отличается собственными атрибутами и характерным поведением.

Класс является шаблоном или формальным описанием объекта, объект же является его реальным воплощением. Каждый класс занимает определенное место в иерархии классов. Любой класс определяет некоторую категорию объектов, а всякий объект есть экземпляр некоторого класса.

Обычно объект находится в некотором уникальном состоянии, определяемом текущими

значениями его атрибутов. Функциональность объектного класса определяется возможными операциями над экземпляром этого класса.

Концепция ООП направлена на создание кода для многократного использования. Основными принципами ООП являются следующие механизмы: инкапсуляция, наследование, полиморфизм и абстракция, которую выделяют отдельно. Для создания программ, обрабатывающих информацию в виде объектов, необходимо понимание, а также комплексное соблюдение всех четырех парадигм.

В объектно-ориентированном программировании придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы, называется абстракцией. Основная идея абстракции состоит в том, чтобы работать с объектами, не вдаваясь в особенности их реализации. Абстракция выполняет основную задачу ООП, позволяя программисту формировать объекты определенного типа с различными характеристиками и поведением, благодаря применению классов.

То, как должен выглядеть класс, какие методы в себе содержать, какими переменными и типами данных манипулировать, определяет интерфейс.

Интерфейс описывает ссылочный тип, который не может иметь какой-либо реализации. Это означает, что все его методы и свойства являются абстрактными. Интерфейсы описывают формальный открытый контракт между классами, которые реализуют интерфейс, и классами, которые используют объекты класса, реализующего интерфейс.

Инкапсуляция - это механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. В ООП код и данные могут быть объединены вместе (в так называемый “черный ящик”) при создании объекта. Внутри объекта коды и данные могут быть закрытыми или открытыми. Закрытые коды или данные доступны только для других частей того же самого объекта и, соответственно, недоступны для тех частей программы, которые существуют вне объекта. Открытые коды и данные, напротив, доступны для всех частей программы, в том числе и для других частей того же самого объекта.

Инкапсуляция существенно повышает надежность разрабатываемых программ, так как локализованные в объекте функции обмениваются с программой сравнительно небольшими объемами данных, причем количество и тип этих данных обычно тщательно контролируются. Другим немаловажным следствием инкапсуляции является легкость обмена объектами, переноса их из одной программы в другую.

Наследование позволяет, практически без ограничений, последовательно строить и расширять классы. Начиная с самых простых классов, можно создавать производные классы по возрастающей сложности, которые не только легки в отладке, но и просты по внутренней структуре.

Новый, или производный класс может быть определен на основе уже имеющегося, или базового класса. При этом новый класс сохраняет все свойства старого: данные объекта базового класса включаются в данные объекта производного, а методы базового класса могут быть вызваны для объекта производного класса, причем они будут выполняться над данными включенного в него объекта базового класса. Иначе говоря, новый класс наследует как данные

старого класса, так и методы их обработки. Если объект наследует свои свойства от одного родителя, то говорят об одиночном наследовании. Если же объект наследует данные и методы от нескольких базовых классов, то говорят о множественном наследовании.

Последовательное проведение в жизнь принципа наследования, особенно при разработке крупных программных проектов, хорошо согласуется с техникой нисходящего структурного программирования. При этом сложность кода программы в целом существенно сокращается. Производный класс (потомок) наследует все свойства, методы и события своего базового класса (родителя) и всех его предшественников в иерархии классов.

При наследовании базовый класс дополняется новыми атрибутами
и операциями. В производном классе обычно объявляются новые свойства и методы. При работе с объектами программист обычно подбирает наиболее подходящий класс для решения конкретной задачи
и создает одного или нескольких потомков от него, которые приобретают способность делать не только то, что заложено в родителе.

Полиморфизм - это свойство, которое позволяет один и тот же идентификатор (одно и то же имя) использовать для решения двух и более схожих, но технически разных задач. Целью полиморфизма, применительно к ООП, является использование одного имени для задания действий, общих для ряда классов объектов. Такой полиморфизм основывается на возможности включения в данные объекта также и информации о методах их обработки (в виде указателей на функции). Принципиально важно, что такой объект становится “самодостаточным”. Будучи доступным в некоторой точке программы, даже при отсутствии полной информации о типе этого объекта, он всегда может корректно вызвать свойственные ему методы. Таким образом, полиморфная функция - это семейство функций с одним и тем же именем, но выполняющих различные действия в зависимости от условий вызова.

Для изменения метода необходимо перегрузить его в потомке, т.е. объявить в потомке одноименный метод и реализовать в нем нужные действия. В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода, имеющие разную кодовую реализацию и, следовательно, придающие объектам разное поведение. Благодаря полиморфизму, потомки могут перегружать общие методы родителя, чтобы реагировать специфическим образом на одно и то же событие.

Каждый класс должен создаваться для достижения определенных целей. Также немаловажно, что объявление класса должно предшествовать созданию его экземпляра. Программист, использующий объектно-ориентированную парадигму программирования, должен четко понимать и разграничивать функции каждого класса. Чтобы избежать дублирования кода и создания классов, выполняющих похожее назначение, следует определить, что объединяет классы и на основе этого создать базовый класс и классы-наследники. Таким образом, можно использовать разработанные классы и в других программах.

В идеале классы должны располагаться в отдельных файлах, имеющих идентичные с классом названия. Например, в RoR все классы контроллеров располагаются в одном каталоге, но в разных файлах с теми же названиями, что и сами классы контроллеров. Каждый

контроллер наследуется от класса ApplicationController.

Пять основных принципов проектирования в объектно-ориентированном программировании сгруппированы в обобщенный принцип ООП, который называется SOLID — Single responsibility, Open-closed, Liskov substitution, Interface segregation и Dependency inversion. В переводе на русский: принципы единственной ответственности, открытости и закрытости, подстановки Барбары Лисков, разделения интерфейса и инверсии зависимостей.

Аббревиатура SOLID была предложена Робертом Мартином, автором нескольких книг, широко известных в сообществе разработчиков. Эти принципы позволяют строить на базе ООП масштабируемые и сопровождаемые программные продукты с понятной бизнес-логикой.

Принцип единственной обязанности трактует, 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 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 в формулировке Роберта Мартина звучит так: «функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом».

Принцип разделения интерфейса 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 — 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 – это современный, объектно-ориентированный язык. В любой программе на Ruby все является объектом. Ruby также является динамическим языком, что означает следующее:

> Ruby интерпретируется динамически (например, как и PHP), поэтому нет никакой компиляции (как с C или Java). Это ускоряет разработку.
> В Ruby все переменные динамического типа, т.е. информация в них автоматически перезаписывается и вам нет необходимости определять и настраивать переменные, как во многих других языках.
> Программа на Ruby может генерировать код динамически, изменив себя во время выполнения.

Ruby является удивительно «чистым» языком. Его легко читать, на нем легко писать, но это не все его преимущества. Ruby можно легко расширить с помощью фреймворка Rails, который

способен его превратить в, своего рода, специальный язык для создания веб-приложений.

Ruby – язык программирования с открытым исходным кодом, то есть, он является бесплатным.

Ruby имеет встроенную поддержку для работы с SQLite, но это не означает, что нельзя использовать Ruby с другими системами управления базами данных. Можно использовать Ruby с MySQL, PostgreSQL, Firebird, DB2, Oracle и Microsoft SQL Server и т.д.

Ruby кроссплатформенный. Это означает, что можно развернуть свое приложение в различных операционных системах, таких, как Windows, Linux, MacOS.

2.3 Технологии front-end разработки

Front-end - 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.

Базовый набор инструментов для разработки интерфейса четко определен: HTML, CSS. Однако технологии разработки веб-интерфейса могут быть расширены с помощью менеджеров пакетов, CSS-препроцессоров, фреймворков и многого другого.

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.

С момента первого выпуска в 1991 году HTML претерпел множество обновлений. В 2014

году был выпущен HTML5. В него добавлены такие функции, как поддержка автономного хранения мультимедиа, более точные элементы контента (например, заголовок, нижний колонтитул, навигация) и поддержка встраивания аудио и видео.

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 Использование веб-фрэймворка Ruby on Rails

Ruby on Rails — полноценный, многоуровневый фреймворк для построения веб-приложений, использующих базы данных, который основан на архитектуре Модель-Представление-Контроллер (Model-View-Controller, MVC).

Для языка Ruby самый популярный фреймворк — это Rails, более 90% веб-приложений, которые написаны на Ruby, используют именно Рельсы.

Культура разработки на Ruby on Rails

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.

С точки зрения бизнеса разработка на RoR весьма эффективна по следующим причинам:

Высокая скорость разработки — проекты на Рельсах разрабатываются действительно

быстрее аналогов на PHP, Python или Java, это подтверждает и наш опыт, и опыт наших коллег по цеху. Обусловлено это как техническими особенностями архитектуры фреймворка (например, продуманные соглашения упрощают конфигурацию), и инструментами для разработки (консольные утилиты и генераторы, готовые библиотеки, расширения и модули). Время разработки — это деньги Заказчика, чем больше времени занимает разработка — тем она дороже.

Сложная бизнес-логика проще и прозрачнее реализуется — конвенции написания программного кода на базе Rails позволяют писать действительно понятный программный код, который впоследствии проще сопровождать и модифицировать в адекватные сроки. Соблюдение заложенных во фреймворк соглашений и стандартов кодирования делает программный код сопровождаемым не только изначальными разработчиками, но и любыми другим специалистами. Отчуждаемость — отсутствие привязки к изначальному разработчику — это очень важная составляющая проекта, разрабатываемого Заказчиком силами внешней компании, а не собственными силами.

Высокая надёжность и сопровождаемость решений — в Rails-разработке обычно используется TDD-подход, а инструментарий поддерживает широкие возможности для тестирования, что делает создаваемые решения более стабильными и сопровождаемыми. Функциональность самого фреймворка также покрыта автоматическими тестами, что делает его использование действительно надёжным — есть уверенность, что ничего не сломается. Для бизнес-систем эта составляющая крайне важна — от стабильности работы приложения часто зависит эффективность работы бизнеса в целом.

Масштабируемость, производительность и высокие нагрузки — фреймворк «заточен» под разработку приложений, к которым предъявляются высокие требования к доступности: Rails-приложения отлично разворачиваются и работают в кластерах серверов или в «облаках». Для веб-сервисов это очень важный критерий, так как для собственной эффективности и для привлекательности в глазах пользователя они должны обладать серьёзным аптаймом и хорошей скоростью работы.

3.2 Использование системы контроля версий GIT

Система контроля версий представляет собой программное обеспечение, которое позволяет отслеживать изменения в документах, при необходимости производить возврат старых данных, определять, кто и когда внес изменения и т.д.

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, однако есть и другие: Mercurial, Subversion и Perforce.

Git-распределенная система контроля версий, разработанная Линусом Торвальдсем для работы над операционной системой Linux. Среди крупных проектов, в рамках которых используется git, можно выделить ядро Linux, QT, Android. Git свободно распространяется под лицензией GNU GPL2 и доступен практически на всех операционных системах. Благодаря таким

достоинствам, как высокая скорость работы, возможность интеграции с другими системами

контроля версий, удобный интерфейс и очень активному сообществу, сформировавшемуся вокруг этой системы, git находится в лидерах на рынке распределенных систем контроля версий.

После установки программного обеспечения для контроля версий, такого как Git, и инициализации репозитория на компьютере, в созданном репозитории добавляется невидимая папка. Она управляет версиями содержимого в папке. При перемещении отслеживания Git в другую папку, невидимая папка git должна переноситься в ту же папку. Добавляя файлы в Git и фиксируя их, Git делает моментальный снимок зафиксированных файлов в этот момент времени. При фиксации другого изменения, Git создает еще один снимок. Если необходимо вернуться к версии файла в определенный момент времени, это можно сделать по снимкам. Это является основной идеей создания версий документов.

В распределенных системах отсутствует четко выделенное центральное хранилище версий, которое еще называют репозиторием. В случае распределенных систем набор версий может быть полностью или частично распределен между различными хранилищами, которые могут быть в том числе и удаленными. Такая система может быть удобна для команды разработчиков, которые располагаются по всему миру, но работают над одним открытым исходным кодом. Любой из них может скачать себе всю информацию о версиях и после этого работать уже на своем локальном компьютере. Как только работа достигнет определенного момента, данные заливаются в центральный репозиторий, и все остальные разработчики могут обновить свою копию хранилища версий. Однако минусом может стать то, что система плохо организована и нет одного центрального хранилища.

3.3 Редактор кода Visual Studio Code

Visual Studio Code — редактор исходного кода, разработанный 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. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом

Visual Studio Code отличный выбор для начинающего программиста, имеет необходимый минимум:

- неплохую документацию
- автодополнение кода (с использованием IntelliSense)
- подсветка синтаксиса
- встроенный отладчик
- расширение функционала за счет плагинов
- управление системой контроля версий git
- кроссплатформенный
- бесплатный, с открытым исходным кодом

Visual Studio Code - отличный редактор для разработки на Ruby. В VS Code Marketplace доступно множество расширений языка Ruby, и в настоящее время создаются и другие.

3.4 База данных SQLite

SQLite - это библиотека на языке C, которая реализует небольшой, быстрый, автономный, высоконадежный, полнофункциональный механизм базы данных SQL. SQLite - самый распространенный в мире движок баз данных. SQLite встроен во все мобильные телефоны и большинство компьютеров и поставляется внутри множества других приложений, которые люди используют каждый день.

Формат файла SQLite является стабильным, кроссплатформенным и обратно совместимым, и разработчики обязуются сохранять его таким, по крайней мере, до 2050 года. Файлы базы данных SQLite обычно используются в качестве контейнеров для передачи богатого контента между системами и в качестве долговременного архивного формата данных. В активном использовании на данный момент более 1 триллиона баз данных SQLite .

Исходный код SQLite находится в свободном доступе и может использоваться всеми для любых целей.

4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC

Концепция MVC (Model-View-Controller: модель-вид-контроллер) очень часто упоминается в мире веб программирования в последние годы. Каждый, кто хоть как-то связан с разработкой веб приложений, так или иначе сталкивался с данным акронимом.

MVC — это архитектурный шаблон проектирования, который описывает способ построения структуры нашего приложения, сферы ответственности и взаимодействие каждой из частей в данной структуре.

Впервые она была описана в 1979 году, конечно же, для другого окружения. Тогда не существовало концепции веб приложения. Tim Berners Lee (Тим Бернерс Ли) посеял семена World Wide Web (WWW) в начале девяностых и навсегда изменил мир. Шаблон, который мы используем сегодня, является адаптацией оригинального шаблона к веб разработке.

Бешеная популярность данной структуры в веб приложениях сложилась благодаря её включению в две среды разработки, которые стали очень популярными: Struts и Ruby on Rails. Эти две среды разработки наметили пути развития для сотен рабочих сред, созданных позже.

Идея, которая лежит в основе архитектурного шаблона MVC, очень проста: нужно чётко разделять ответственность за различное функционирование в наших приложениях. Приложение разделяется на три основных компонента, каждый из которых отвечает за различные задачи.

Контроллер (Controller)

Контроллер управляет запросами пользователя (получаемые в виде запросов HTTP GET или POST, когда пользователь нажимает на элементы интерфейса для выполнения различных действий). Его основная функция — вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно

контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.

Например, Products_controller будет обрабатывать все взаимодействия и входные данные из представления товаров и обновлять базу данных, используя модель товаров. Тот же контроллер будет использоваться для просмотра данных товаров.

Модель (Model)

Модель - это данные и правила, которые используются для работы с данными, которые представляют концепцию управления приложением. В любом приложении вся структура моделируется как данные, которые обрабатываются определённым образом. Модель даёт контроллеру представление данных, которые запросил пользователь (сообщение, страницу книги, фотоальбом, и тому подобное). Модель данных будет одинаковой, вне зависимости от того, как мы хотим представлять их пользователю. Поэтому мы выбираем любой доступный вид для отображения данных.

Модель содержит наиболее важную часть логики нашего приложения, логики, которая решает задачу, с которой мы имеем дело. Контроллер содержит в основном организационную логику для самого приложения.

Например, объект User будет извлекать информацию о пользователе из базы данных, манипулировать ею, обновлять данные и помещать их обратно в базу данных или использовать ее для визуализации данных.

Вид (View)

Вид обеспечивает различные способы представления данных, которые получены из модели. Он может быть шаблоном, который заполняется данными. Может быть несколько различных видов, и контроллер выбирает, какой подходит наилучшим образом для текущей ситуации.

Например, представление Users_page будет включать шаблон с данными о пользователе сайта, а также представлять страницы, где пользователь может зарегистрироваться или авторизоваться.

Таким образом, модель управляет фундаментальным поведением и данными приложения. Она может отвечать на запросы о предоставлении информации, отвечать на инструкции по изменению состояния информации и даже уведомлять наблюдателей в управляемых событиями системах при изменении информации. Это может быть база данных или любое количество структур данных или систем хранения. Представление эффективно предоставляет элемент пользовательского интерфейса приложения. Оно будет отображать данные из модели в форму, подходящую для пользовательского интерфейса. Контроллер получает пользовательский ввод и выполняет вызовы к модельным объектам и представлению для выполнения соответствующих действий.

С технической точки зрения, при сборке с использованием архитектуры MVC у разработчика есть следующие стратегические преимущества:

> Разделять роли в проекте становится проще. В проекте может быть бэкэнд-разработчик, работающий над логикой контроллера, и фронтенд-разработчик, который работает с представлениями. Это очень распространенный способ работы в компаниях, и наличие MVC делает процесс разделения обязанностей намного проще, чем, когда кодовая база содержит так называемый «спагетти-код».
> Стратегия MVC вынуждает разбивать файлы на логические каталоги, что облегчает

поиск определенных файлов при работе над большими проектами.
Изоляция ответственности. Например, можно вносить изменения в представления и в модели отдельно, потому что они друг от друга независимы.
Полный контроль над URL-адресами приложений. Архитектура MVC позволяет полностью контролировать внешний вид приложения, выбирая маршруты.
С MVC легче следовать принципу SOLID.

У архитектурного шаблона проектирования MVC есть как преимущества, так и недостатки:

Необходимость использования большего количества ресурсов. Сложность обусловлена тем, что все три фундаментальных блока являются абсолютно независимыми и взаимодействуют между собой исключительно путем передачи данных. Controller должен всегда загрузить (и при необходимости создать) все возможные комбинации переменных и передать их в Model. Model, в свою очередь, должна загрузить все данные для визуализации и передать их во View.
Усложнен механизм разделения программы на модули. В концепции MVC наличие трех блоков (Model, View, Controller) прописано жестко. Соответственно каждый функциональный модуль должен состоять из трех блоков, что в свою очередь, несколько усложняет архитектуру функциональных модулей программы.
Усложнен процесс расширения функционала. Проблема очень схожа с вышеописанной. Недостаточно просто написать функциональный модуль и подключить его в одном месте программы. Каждый функциональный модуль должен состоять из трех частей, и каждая из этих частей должна быть подключена в соответствующем блоке.

RoR, который является одним из лучших Ruby-фреймворков для веб-разработки, следует архитектурному шаблону MVC.

Схема работы архитектурного шаблона проектирования MVC представлена на рисунке 17.

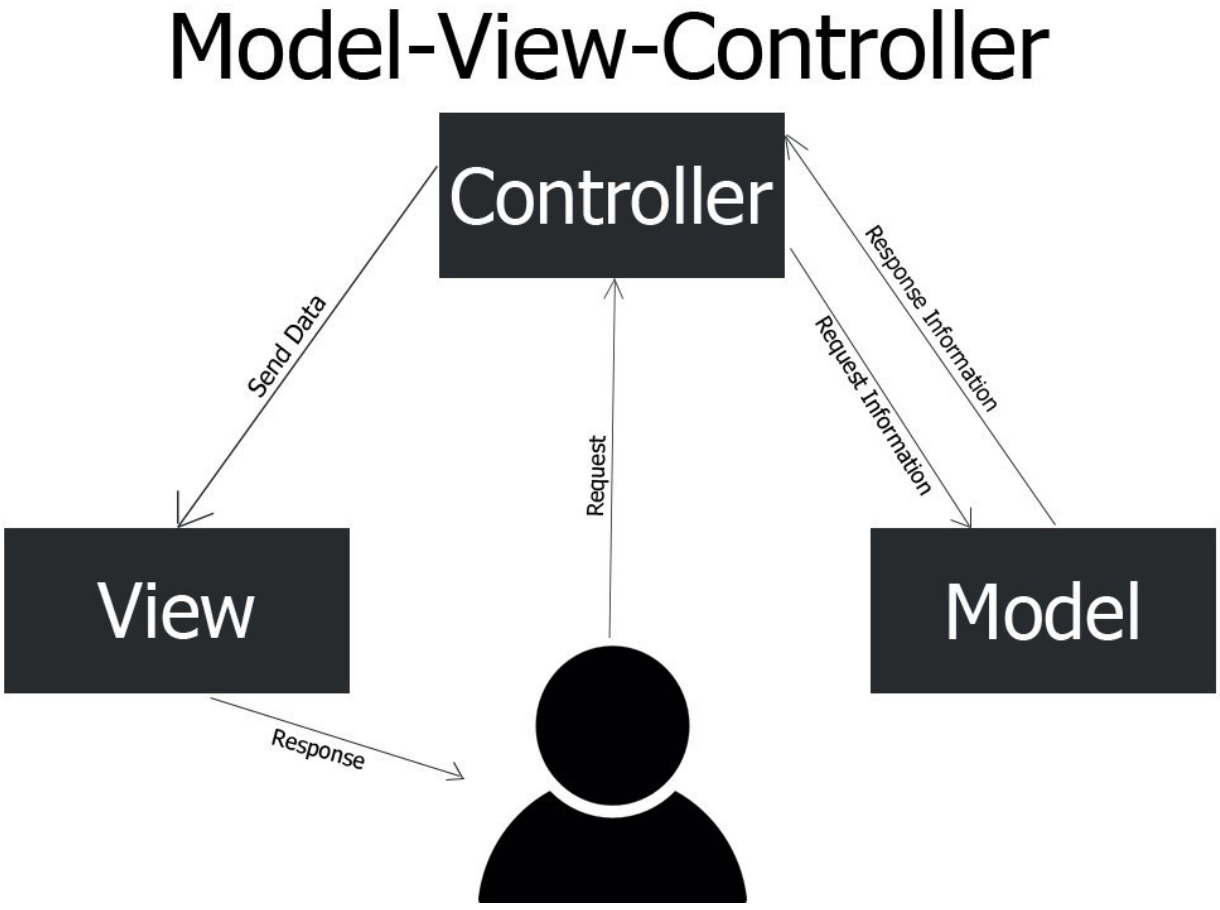

Рисунок 17 - Схема работы шаблона проектирования MVC

Рассмотрим конкретный пример реализации MVC. Предположим, пользователь сайта хочет перейти к странице описания продукта. Пользователь переходит по URL products/16, браузер отправляет get-запрос. Далее переходим в контроллер Products_controller, где находится метод show. В этом методе происходит получение данных о товаре с помощью модели. Этот же метод вызывает представление, передавая данные о товаре: @product=Product.find(params[:id]);

Более бизнес-ориентированный способ описания работы архитектурного шаблона проектирования MVC – изображение на диаграмме последовательности. Диаграмма представлена в графическом материале. Данная диаграмма подходит под любой класс контроллера, под любое представление и, соответственно, модель.

Диаграмма последовательности наиболее точно отражает процессы, происходящие во время взаимодействия пользователя с системой. Согласно диаграмме, пользователь отправляет запрос страницы с целью увидеть какой-то контент. Это может быть главная страница приложения или, если пользователь авторизован, страница с задачами или дневником, например. Промежуточное звено между объектами «Пользователь» и «Контроллер» - это маршрут, который направляет систему на выполнение какого-либо метода класса контроллера. Соответствующий метод после выполнения каких-либо действий, обычно с моделью (изъятие данных из модели, одиночная или множественная вставка в модель и т.д.), вызывает представление, которое в последующем увидит пользователь системой.

Пользователь имеет возможность взаимодействовать с системой. Например, отправлять данные для обработки на сервере с помощью формы. После ввода данных в форму происходит уже описанная выше цепочка, затем метод контроллера выполняет вставку в модель (обычно при отправлении формы данные сохраняются в модели), а в ответ пользователю возвращается первичное представление либо какое-то другое.

На схеме указано, что контроллер взаимодействует с моделью. Это взаимодействие проявляется в работе с данными системы. В модель могут добавляться новые данные, обновляться или удаляться старые. Метод контроллера, в котором происходит работа с данными, может вернуть данные в представление и отобразить пользователю какую-либо информацию из модели.

Например, в базе хранятся фотографии, которые пользователь загрузил раньше. Когда пользователь отправляет запрос страницы, метод контроллера GalleryController gallery собирает фотографии, которые ему принадлежат, и передает эти данные в представление. В представлении они отображаются в таком виде, в котором задумал дизайнер пользовательского интерфейса.

Диаграмма последовательности полезна для описания алгоритма действий, но она не дает представления о поведении определенного объекта в рамках отдельного варианта использования или системы в целом, что необходимо при объектно-ориентированном программировании.

На сегодняшний день при проектировании сложной системы принято делить ее на части, каждую из которых затем рассматривать отдельно. Таким образом, при объектной декомпозиции система разбивается на объекты или компоненты, которые взаимодействуют друг с другом, обмениваясь сообщениями. Сообщения описывают или представляют собой

некоторые события. Получение объектом сообщения активизирует его и побуждает выполнять предписанные его программным кодом действия.

При данном подходе система становится событийно управляемой, поэтому разработчикам зачастую важно знать, как должен реагировать тот или иной объект на определенные события. Инициаторами событий могут быть как объекты самой системы, так и её внешнее окружение.

Описать поведение отдельно взятого объекта помогает диаграмма состояний.

Диаграмма состояний представлена в графическом материале. На диаграмме присутствует два объекта – «пользователь» и «интернет-магазин», и этого вполне достаточно, чтобы описать их взаимодействие. Пользователь заходит на сайт с целью просмотра и возможного заказа товаров. Отклик системы на такое поведение пользователя, если он не авторизован, - запросить ввод логина и пароля. Пользователь, чтобы получить доступ к заказу товара, обязан ввести логин (email) и пароль. Система проверит данные, и, если такой пользователь существует, она выведет страницу добавления товара в корзину. Иначе, система покажет сообщение об ошибке и предложит пользователю либо ввести данные снова.

Диаграмма состояний описывает взаимодействие пользователя и системы на этапе заказа, то есть тогда, когда пользователю необходимо авторизоваться.

5 ШАБЛОН ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ

Шаблон проектирования, или паттерн, - повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Шаблоны проектирования приобрели широкую популярность в связи с тем, что к моменту их появления, программирование уже было довольно сильно развито. Разработчики начали понимать, что нет необходимости изобретать велосипед, когда уже кто-то до тебя сталкивался с такой же определенного рода проблемой.

Выделяют три основных типа шаблона проектирования: порождающий, структурный и поведенческий.

Порождающий шаблон проектирования позволяет сделать систему независимой от способа создания, композиции и представления объектов, то есть этот шаблон создает объект или их группу. К порождающим шаблонам относят: абстрактная фабрика, строитель, прототип, одиночка и другие.

Одним из самых известных из порождающих шаблонов является абстрактная фабрика. Он позволяет разработчику создавать интерфейс для объектов, которые связаны как-либо между собой. При этом нет необходимости указывать конкретные классы, так как работа с каждым из них производится через интерфейс. С помощью такой фабрики создаются группы объектов, реализующие одно поведение. К преимуществам абстрактной фабрики можно причислить то, что она изолирует конкретный класс, благодаря чему легко заменять семейства продуктов. Недостатком можно считать то, что при расширении возможностей фабрики путем добавления нового типа продуктов, придется изменять все конкретные реализации шаблона, а это бывает очень сложно на практике, если создано большое количество фабрик.

Структурные паттерны рассматривают образование более крупных структур из объектов и классов. Шаблоны уровня класса полезны в тех случаях, когда необходимо объединить процессы нескольких библиотек. При этом затрагиваются такие механизмы, как:

- наследование (определение реализации подклассами, а интерфейса - базовым классом);

- композиция, при которой структуры образуются через объединение нескольких объектов.

К структурным шаблонам относятся: адаптер, мост, компоновщик и другие. Рассмотрит такой структурный шаблон проектирования, как адаптер. Он предназначен для преобразования интерфейса одного класса в интерфейс другого. Благодаря реализации данного паттерна мы можем использовать вместе классы с несовместимыми интерфейсами. Адаптер используют, когда необходимо использовать имеющийся класс, но его интерфейс не соответствует потребностям, или уже существует класс совместно с другими классами, интерфейсы которых несовместимы.

Участники:

target-представляет объекты, которые используются клиентом;
client-использует объекты target для реализации своих задач;
adaptee-представляет адаптируемый класс, который мы хотели бы использовать у клиента вместо target;
adapter- непосредственно сам адаптер, который позволяет работать с объектами adaptee как с объектами target.

То есть клиент ничего не знает об adaptee, он знает и использует только объекты target. И благодаря адаптеру мы можем использовать объекты adaptee как target.

Поведенческие шаблоны связаны с распределением обязанностей между объектами. Их отличие от структурных шаблонов заключается в том, что только описывают структуру, но также описывают шаблоны для их взаимодействия. К поведенческим шаблонам относят: цепочка обязанностей, итератор, наблюдатель и другие.

Паттерн Наблюдатель представляет собой поведенческий шаблон проектирования, который использует отношение «один ко многим». В этом отношении есть один наблюдаемый объект и множество наблюдателей. И при изменении наблюдаемого объекта автоматически происходит оповещение всех наблюдателей.

Данный паттерн еще называют издатель-подписчик, поскольку их взаимоотношения схожи с сутью данного шаблона. Подписчик подписывается на издательскую рассылку, а издатель уведомляет всех подписчиков об изменениях. Наблюдатель используют, когда система состоит из множества классов, объекты которых должны находиться в согласованных состояниях; общая схема взаимодействия объектов предполагает две стороны: одна рассылает сообщения и является главной, другая получает сообщения и реагирует на них, т.е. отделение логики обеих сторон позволяет рассматривать их рассматривать независимо друг от друга; существует один объект, рассылающий сообщения и множество подписчиков, которые получают

сообщения, при это точное число подписчиков заранее неизвестно и в процессе работы может меняться.

Участники:

IObservable-представляет собой наблюдаемый объект;

ConcreteObservable-конкретная реализация интерфейса IObservable, которая определяет коллекцию объектов наблюдателей;
IObserver-представляет наблюдателя, который подписывается на все уведомления наблюдаемого объекта;
ConcreteObserver-конкретная реализация интерфейса IObserver.

При этом наблюдаемому объекту не надо ничего знать о наблюдателе кроме того, что тот реализует метод Update(). С помощью отношения агрегации реализуется слабосвязанность обоих компонентов. Изменения в наблюдаемом объекте не влияют на наблюдателя и наоборот. В определенный момент наблюдатель может прекратить наблюдение. И после этого и наблюдатель, и наблюдаемый могут продолжать существовать в системе независимо друг от друга.

Выше были рассмотрены только по одному примеру из каждого типа шаблонов проектирования, однако, естественно, их существует больше.

6 ПРИЛОЖЕНИЕ А

```
  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.order(:created_at).reverse_order.first(3)
      @products_hit=Product.order(:sale).reverse_order.first(3)

    end

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

    def new
      @product = Product.new
    end
    def edit
```

```
      @product=Product.find(params[:id])
    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,:firma,:sale,:volume,:picture,:cost,:category_id,
:image)
      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)
    @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 "http://localhost:3000/users_page"
   else
    redirect_to "http://localhost:3000/users_page"
   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.destroy
      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
```

P.S. В Приложении А приведен код только основной части бэкенда программы. Полный код, с клиентской частью, можно посмотреть по следующей ссылке: https://github.com/danusyaaaa/online-shop.

Заключение

В настоящее время интернет представляет собой высокоинтеллектуальную сферу, где происходит обмен, хранение и обработка огромного массива информации. Потребителем этих

данных является пользователь, то есть человек. В свою очередь, необходимость общаться и обменивать этой информацией породили создание площадок, где люди в свободной форме могут коммуницировать друг с другом. Кроме того, Интернет стал эффективным инструментом для осуществления коммерческой деятельности. Создание Интернет-магазинов - один из наиболее выгодных и перспективных инструментов онлайн-бизнеса. В целом на сегодняшний день системы управления сайтами дают возможность реализовать любые требования пользователя при правильном подходе к выбору системы. В результате выполнения курсовой работы были изучены материалы, связанные с разработкой сайта интернет-магазина на фреймворке Ruby on Rails. В свою очередь, в ходе создания сайта был изучен шаблон проектирования MVC, исследована и применена на практике объектно-ориентированная технология программирования. Интернет-магазин косметики разработан при помощи языка Ruby, с использованием языков разметки HTML и CSS. Вся информация, которая отображается на сайте, хранится при помощи баз данных, доступ к которым осуществляется по запросам. Навигация по сайту довольно проста и интуитивно понятна, использование анимации информирует пользователя о загрузке запрашиваемых данных.

Список использованных источников

1. [url] **ERUD.BY** http://erud.by/object_orient_program
2. [url] **RUBY ON RAILS ПО-РУССКИ** http://rusrails.ru/
3. [url] **HABR** https://habr.com/
4. [url] **GIT ДЛЯ НАЧИНАЮЩИХ. ЧАСТЬ 1**
https://devpractice.ru/git-for-beginners-part-1-what-is-vcs/
5. [url] **ШАБЛОНЫ ПРОЕКТИРОВАНИЯ ПРОСТЫМ ЯЗЫКОМ**
https://tproger.ru/translations/design-patterns-simple-words-2/

Приложения

1. [электронный документ] 5ebd78b541f57_Пояснительная записка ООП.docx
2. [электронный документ] 5ebdb3a90a2aa_Приложение Б.docx