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

Факультет Компьютерных технологий

Кафедра проектирования информационных компьютерных систем

Дисциплина "Современные технологии проектирования информационных систем"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
04.07.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Интернет-каталог косметики

БГУИР КР 1-40 05 01-10 № 172 ПЗ

Студент (подпись студента)

Курсовая работа
представлена на проверку
04.07.2025

_____
(подпись студента)

2025

Реферат

БГУИР КР 1-40 05 01-10 № 172 ПЗ, гр. 784371

, Интернет-каталог косметики, Минск: БГУИР - 2025.

Пояснительная записка 117036 с., 16 рис., 6 табл.

Ключевые слова:

Предмет Современные технологии проектирования информационных систем, А.В.Михалькевич

Содержание

Введение
1 Лист задания
2 Описание проекта
3 Система контроля версий
4 Спецификация вариантов использования системы
5 Модели представления системы и их описание
6 Ruby on Rails
7 Применение шаблона проектирования практических решений
8 Результат работы
9 Список литературы
10 ПРИЛОЖЕНИЕ А
Заключение
Список использованных источников
Приложения

Введение

Курсовая работа выполнена на тему «Интернет-каталог косметики». Целью курсового проекта является разработка интернет-каталога по продаже косметики. В современном мире Интернет становиться все более популярной средой для поиска и привлечения потребителей. В тоже время, существенным является и тот факт, что Интернет становиться удобной и достаточно дешевой «торговой площадкой». Все больше магазинов стараются предоставить свою продукцию о онлайн среде. Это обусловлено не только тем, что содержание интернет-магазина дешевле, чем физического магазина, но и тем, что все больше потребителей ищут требуемую продукцию именно на просторах Интернета. С развитием Интернет-среды развивается и само предложение. Теперь люди могут не только получать интересующую их информацию, но и с помощью Интернет-магазинов совершать покупки. При этом в сети-Интернет расположение большое количество магазинов с различными товарами. Таким образом пользователь, может не вставая с дивана заказать сразу такую разнообразную продукцию как утюг, свежие фрукты и новые ботинки в пару кликов. В курсовой работе поставлена цель: разработка интернет-каталога косметики. В качестве практической части в рамках курсового проекта создаётся интернет-каталог косметики с использованием технологий языка программирования Ruby и СУБД PostgreSQL. Для достижения поставленной цели необходимо решить следующие задачи: Описание проекта; Выбор технологий и инструментария; Выбор архитектурного шаблона проектирования; Программирование; Наполнение интернет-каталога информацией; Размещение на открытый репозиторий.

1 Лист задания

Министерство образования Республики Беларусь
Учреждение образования
Белорусский государственный университет информатики и радиоэлектроники
Факультет непрерывного и инновационного обучения
Кафедра проектирования информационно-компьютерных систем

«УТВЕРЖДАЮ»
Заведующий кафедрой
_____ В.В. Хорошко
«___» _____ 2020

З А Д А Н И Е
к курсовой работе по дисциплине «Современные технологии проектирования информационных систем»

Фамилия, имя, отчество Пенькова Полина Геннадьевна

группа 784371

1.Тема проекта: Интернет-каталог косметики

2.Сроки сдачи студентом законченного проекта: 3 мая 2020 г.

3.Исходные данные к проекту:

3.1.Описание к выполнению Верстка и программирование с использованием архитектурного шаблона проектирования MVC

3.2.Язык и среда программирования – на выбор студента. Однако разработанное программное обеспечение должно быть реализовано на объектно-ориентированном языке.

3.3.В реализации программного обеспечения учесть возможность использования сервера.

3.4.Пояснительную записку и графический материал выполнять по СТП БГУИР 01-2013.

3.5.Другие требования уточняются студентом в процессе работы.

4. Содержание расчётно-пояснительной записки (перечень подлежащих разработке вопросов):

Титульный лист. Заполненный бланк задания с приложением. Содержание (1-2 *стр.*)

Введение (*1 – 3 стр. Актуальность темы курсовой работы; цель и перечень задач,*

которые планируется решить; детальная постановка задачи).

4.1.Описание проекта (*10 – 15 стр. Описание серверной и клиентской части разрабатываемого проекта*).

4.2.Обоснование выбора технологий (*7-15 стр. Технологии программирования, используемые для решения поставленных задач. Реализация объектно-ориентированных технологий программирования в современных программно-математических средах*).

4.3.Инструментарий (*5-7 стр. Обоснование используемых инструментов. Использование системы контроля версий GIT. Обязательна ссылка на репозиторий с проектом, например github.com*.).

4.4.Архитектурный шаблон проектирования MVC (*5-7 стр. Разработка схемы алгоритма, диаграммы последовательности и диаграммы состояний (схемы в Приложении) с детальными пояснениями каждого компонента шаблона проектирования MVC или его модификаций*).

4.5.Шаблон проектирования практических решений (*7-10 стр. Использование шаблонов проектирования практических решений для решения практических задач*).

Заключение (*1 стр. Выводы по курсовой работе*).

Список литературных источников (*1, 2 стр. Перечень литературы и интернет-источников, которые были реально использованы при выполнении курсовой работы).*

Приложения (*3 и более стр. Ведомость документов, листинг программного кода и др.).*

5.Перечень графического материала (с указанием обязательных чертежей и графиков):

5.1.Структура графического пользовательского интерфейса (формат А3 или несколько А4)

5.2.Схема алгоритма (формат А3 или несколько А4)

5.3.Диаграмма последовательности (формат А3 или несколько А4)

5.4.Диаграмма состояний (формат А3 или несколько А4)

6.Консультант по работе: Михалькевич Александр Викторович

7.Дата выдачи задания:

8.Календарный график работы над проектом на весь период проектирования:

№ п/п	Наименование этапов курсового проекта	Срок выполнения этапов проекта	Примечание
1.	1-я опроцентовка (пп. 4.1, 4.2, 5.1)	04.03.2020	40%
2.	2-я опроцентовка (пп. 4.3, 4.4, 5.2, 5.3)	01.04.2020	70%...80%
3.	3-я опроцентовка (пп. 4.5, 4.6, 5.4, приложения)	29.04.2020	95%
4.	Сдача на проверку и защита курсового проекта	03.05.2020	100%
5.	Защита курсового проекта	10-11.05.2020	Согласно графику

Руководитель _____

А.В.Михалькевич

Задание принял к исполнению

2 Описание проекта

1 Описание проекта

Темой проекта является: интернет-каталог косметики. Данный интернет-каталог создается для того, чтобы пользователи могли познакомиться с перечнем продаваемых товаров, информацией о данных товарах, а также оформить заказать на требуемый товар.

1. **Дизайн интернет-каталога**

Дизайн решение разрабатываемого каталога будет соответствовать стандартному оформлению фраймворка Ruby on Rails, с использованием библиотеки spree.

Цветовое решение данной библиотеки –белые и синие цвета.

1. **Технологии и инструментарий проекта**

Технологии проекта: *Ruby on Rails*, *PostgreSQL*, *Puma*.

Обоснование выбора технологии: выбор программ обусловлен тем, что они являются объектно-ориентированными программами, а также веб-оптимизированными программами.

Инструментарий: Программной средой проекта был выбран *Puma*. Ссылка для скачивания: https://github.com/puma/puma

При запуске *Puma* становятся доступны следующие возможности: HTTP 1.1 protocol.

Рисунок 1 – Запуск Puma

В разделе 1 описан проект: дизайн интернет-каталога, обоснованы технологии и инструментарий проекта. Основными технологиями проекта являются *Ruby on Rails, PostgreSQL*. Программная среда проекта - *Puma*.

3 Система контроля версий

Git- мощная и сложная распределенная система контроля версий.

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь СКВ, вы всё испортите или потеряете файлы, всё можно будет легко восстановить.

Зарегистрировались на GitHub.com.

Прописали следующие команды в консоли:

1. Git init – создание репозитория;

2. Git add * - добавляет содержание рабочей директории в индекс (staging area) для последующего изменения;

3. Git commit –m “сообщение”- изменение последнего изменения;

4.Git remote add - добавление удаленного репозитория;

5. Git push project master - вносим изменения в удаленный репозиторий.

В разделе 2 описаны возможности и установка Git.

Git- мощная и сложная распределенная система контроля версий.

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

Также приведена ссылка на удаленный репозиторий, по которой можно найти данный проект.

4 Спецификация вариантов использования системы

Спецификация - документ, который полно описывает элемент проекта или его интерфейсы в терминах требований (к функциям, производительности, ограничениям и устройству), а также условия приёмки и процедуры проверки требований.

Спецификация может содержать:

описательное название, номер или другой идентификатор спецификации;
время последнего пересмотра и отметку, кем был выполнен пересмотр;
логотип или торговую марку, указывающую, кому принадлежит право на копирование, владельца и происхождение документа;
содержание документа, если документ длинный;
ответственное лицо или организацию по вопросам по спецификации, по обновлениям и отклонениям;
важность, область применения спецификации и её назначение;
термины, определения и аббревиатуры для пояснения сути спецификации;
способы проверки для всех установленных требований и характеристик;
материальные требования: физические, механические, электрические, химические и другие. Целевые и допустимые;
требования по эксплуатационному тестированию. Целевые и допустимые;
изображения, фотографии или технические иллюстрации;
требования по мастерству;
требования к сертифицированности;
требования по технике безопасности;
экологические требования;
контроль по обеспечению качества, образец для проверки, проверка, критерий приёма работы;
лицо или организация ответственное за выполнение спецификации;
выполнение и доставка;
условия по отклонениям, перепроверке, пересмотре, корректировке измерений и характеристик;

ссылки и цитаты в тексте спецификации, которые могут потребоваться для установки ясности документа;
подписи и разрешения, если они необходимы;
контроль изменений (с помощью специальных компьютерных программ) для последовательной разработки, проверки и выполнения, если документ предназначен для внутреннего использования;
приложения, которые раскрывают детали, добавляют ясности, или пояснения по оплате.

Сценарий использования, вариант использования, прецедент использования (англ. use case) – в разработке программного обеспечения и системном проектировании это описание поведения системы, когда она взаимодействует с кем-то (или чем-то) из внешней среды. Система может отвечать на внешние запросы Актёра (англ. actor) (может применяться термин Актант), может сама выступать инициатором взаимодействия. Другими словами, сценарий использования описывает, «кто» и «что» может сделать с рассматриваемой системой, или что система может сделать с «кем» или «чем». Методика сценариев использования применяется для выявления требований к поведению системы, известных также как пользовательские и функциональные требования.

3.1 Диаграмма вариантов использования

Диаграмма вариантов использования — диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

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

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

При моделировании системы с помощью диаграммы прецедентов системный аналитик стремится:

чётко отделить систему от её окружения;
определить действующих лиц (актёров), их взаимодействие с системой и ожидаемую функциональность системы;
определить в глоссарии предметной области понятия, относящиеся к детальному описанию функциональности системы (то есть прецедентов).

Диаграмма вариантов использования представлена на рисунке 3.1.

Рисунок 3.1 - Диаграмма вариантов использования

5 Модели представления системы и их описание

4 Модели представления системы и их описание

4.1 Диаграмма последовательности

Диаграмма последовательности — диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актёров (действующих лиц) ИС в рамках какого-либо определённого прецедента (отправка запросов и получение ответов).

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

Диаграмма последовательности показана на рисунке 4.1.

Рисунок 4.1 - Диаграмма последовательности

4.2 Диаграмма состояний

Диаграмма состояний предназначена для отображения состояний объектов системы, имеющих сложную модель поведения. Она показывает пространство состояний системы или ее элементов, события, которые влекут переход из одного состояния в другое, действия, которые происходят при изменении состояния. Объекты меняют своё состояние в ответ на происходящие события и стечением времени. Диаграмма состояний представляет состояния объекта и переходы между ними, а также начальное и конечное состояние объекта.

Диаграмма состояний показана на рисунке 4.2.

Рисунок 4.2 - Диаграмма состояний

4.3 Диаграмма классов

Диаграммы классов – это наиболее часто используемый тип диаграмм, которые создаются при моделировании объектно-ориентированных систем. Показывают набор классов, интерфейсов и кооперации, а также их связи. На практике диаграммы классов применяют

для моделирования статического представления системы (по большей части это моделирование словаря системы, кооперацій или схем). Кроме того, диаграммы данного типа служат основой для целой группы взаимосвязанных диаграмм – диаграмм компонентов и диаграмм размещения.

Диаграмма классов описывает структуру системы, показывая её классы, их атрибуты и операторы, а также взаимосвязи этих классов.

Диаграмма классов представлена на рисунке 4.3.

Рисунок 4.3 - Диаграмма классов

4.4 Диаграмма компонентов

Диаграмма компонентов (англ. Component diagram) – элемент языка моделирования UML, статическая структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.

Диаграмма компонентов представлена на рисунке 4.4.

Рисунок 4.4 - Диаграмма компонентов

4.5 Диаграмма развертывания

Диаграмма развёртывания, Deployment diagram в UML моделирует физическое развертывание артефактов на узлах. Например, чтобы описать веб-сайт диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, JDBC, REST, RMI).

Диаграмма развертывания представлена на рисунке 4.5.

Рисунок 4.5 - Диаграмма развертывания

6 Ruby on Rails

5 Ruby On Rails

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.

Создан Давидом Хейнемейером Ханссоном на основе его работы в компании 37signals над средством управления проектами Basecamp и выпущен в июле 2004 года. 23 декабря 2008 года команда проекта Merb объединилась с командой Rails с целью создания следующей версии Rails 3, которая объединит в себе лучшие черты обоих фреймворков.

Основными компонентами приложений на Ruby on Rails являются модель (англ. model), представление (англ. view) и контроллер (англ. controller).

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.

Библиотека Ruby on Rails Spree Commerce - это решение для электронной коммерции с открытым исходным кодом на основе Ruby on Rails . Он был создан Шоном Шофилдом 2007 году и с тех пор имеет более 740 участников и более 626 тысяч загрузок из RubyGems.

Расширения являются основным механизмом настройки сайта Spree. Они позволяют разработчикам Spree делиться друг с другом кодом многократного использования. Использование расширений, функций, которые в противном случае потребляли бы довольно много усилий и времени, можно легко добавить на веб-сайт Spree без необходимости создавать их с нуля. Обширные функции, такие как подписка на продукт, управление активами, маркетинг, отчеты администратора, роли и разрешения, подарочные карты и рекламные акции, слоты доставки, возврат товаров, маркетинг в социальных сетях, проверка на одну страницу и многое другое, легко доступны в GitHub для различных версий Spree.

21 сентября 2015 года он был приобретен First Data. После первого сбора данных разработчики из Spark Solutions и VinSol теперь поддерживают и развивают проект Spree Commerce Open Source.

5.1 Теория MVC

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Рисунок 2 – Модель MVC

Концепция MVC была описана Трюгве Реенскаугом в 1978 году, работавшем в научно-исследовательском центре «Xerox PARC» над языком программирования «Smalltalk». Позже, Стив Бурбек реализовал шаблон в Smalltalk-80.

Окончательная версия концепции MVC была опубликована лишь в 1988 году в журнале Technology Object.

Впоследствии, шаблон проектирования стал эволюционировать. Например, была представлена иерархическая версия HMVC; MVA, MVVM.

Дальнейший виток популярности привнесло развитие фреймворков, ориентированных на быструю развёртку, на языках Python и Ruby, Django и Rails. На момент 2017 года, фреймворки с MVC заняли заметные позиции по отношению к остальным фреймворкам без этого шаблона.

С развитием объектно-ориентированного программирования и понятия о шаблонах проектирования — был создан ряд модификаций концепции MVC, которые при реализации у разных авторов могут отличаться от оригинальной. Так, например, Эриан Верми в 2004 году описал пример обобщённого MVC.

В предисловии к диссертации «Naked objects» Ричарда Поусона (Richard Pawson), — Трюгве Реенскауг упоминает свою неопубликованную наиболее раннюю версию MVC, согласно которой:

1. Модель относилась к «разуму» пользователя;
2. Под представлением имелся в виду редактор, позволяющий пользователю просматривать и обновлять информацию;
3. Контроллер являлся инструментом для связывания представлений воедино и применялся пользователем для решения его задач.

Концепция MVC позволяет разделить модель, представление и контроллер на три отдельных компонента.

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает, как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем) просто предоставляя доступ к данным и управлению ими.

Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление «наблюдателей».

Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Представление может влиять на состояние модели, сообщая модели об этом.

Контроллер обеспечивает «связи» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

5.2 Авторизация

Модуль авторизации поставляется совместно с библиотекой Devise. Файл конфигурации авторизации находится в файле app/Models/User.*rb*.

По умолчанию, для сохранения пользовательских данных, *Rails* использует модель User.

Дополнительные настройки можно производить в модели User и с помощью дополнительных методов.

Для создания модели пользователя необходимо выполнить следующие команды:

rails g spree:install --user_class=Spree::User

rails g spree:auth:install

rails g spree_gateway:install

Эти команды создадут необходимые миграции (db/migrations) и роуты (config/routes.rb).

Для создания админа, выполнили следующую команду:

rake spree_auth:admin:create

Рисунок 3 – Авторизация

После успешной авторизации пользователь перенаправляется в /admin. Чтобы перенаправить пользователя на другую страницу, в routes.rb.

7 Применение шаблона проектирования практических решений

6 Применение шаблона проектирования практических решений

Шаблон проектирования (*design patterns*) – это многократно используемые решения распространённых проблем, возникающих при разработке программного обеспечения.

Главная польза каждого отдельного шаблона состоит в том, что он описывает решение целого класса абстрактных проблем. Таким образом, за счёт шаблонов производится унификация терминологии, названий модулей и элементов проекта. Однако есть мнение, что слепое применение шаблонов из справочника, без осмысления причин и предпосылок выделения шаблона, замедляет профессиональный рост программиста, так как подменяет творческую работу механической подстановкой.

Что бы программист ни разрабатывал, на каком бы языке он ни писал, если он стремится к хорошему коду, он будет использовать шаблоны проектирования. Он стремится повторно воспользоваться решениями, которые оказались удачливыми ранее.

Паттерны проектирования представляют наилучшие решения часто встречаемых задач и упрощают повторное использование удачных решений. Шаблоны проектирования не должны ограничивать.

В зависимости от назначения выделяют:

1. Структурные шаблоны (*structural patterns*) – показывают, как объекты и классы объединяются для образования сложных структур.

2. Порождающие шаблоны (*creational patterns*) – контролируют процесс создания и жизненный цикл объектов.

3. Шаблоны поведения (*behavioral patterns*) – используются для организации, управления и объединения различных вариантов поведения объектов.

Каждый шаблон проектирования описывает задачи, с которыми программисту часто приходится сталкиваться. И затем описывает основу решения этой задачи таким образом, что вы можете воплотить это решение при разработке других программ, ни разу не повторившись.

Практические решения, в свою очередь, подразделяются на порождающие паттерны, структурные паттерны и паттерны поведения.

1) Порождающие паттерны или паттерны создания объектов: абстрактная фабрика, одиночка, прототип, строитель, фабричный метод.

Первая группа - это *creational* паттерны. Они в той или иной степени работают с механизмами создания объектов.

Singleton - обеспечиваем существование в системе ровно одного экземпляра некоторого класса;

Factory Method - делегируем процесс создания объектов классам-наследникам;

Prototype - клонируем объекты на основании некоторого базового объекта;

Builder - отделяем процесс создания комплексного объекта от его представления;

Abstract Factory - описываем сущность для создания целых семейств взаимосвязанных объектов.

2) Структурные паттерны: адаптер, декоратор, заместитель, компоновщик, мост, приспособленец, фасад. Они описывают создание более сложных объектов, либо упрощают работу с другими объектами сиситмы.

Adapter - на основании некоторого класса создаем необходимый клиенту интерфейс;

Facade - описываем унифицированный интерфейс для облегчения работы с набором подсистем;

Composite - работаем с базовыми исоставными объектами единым образом;

Decorator - динамически добавляем новую функциональность некоторому объекту, сохраняя его интерфейс;

Proxy - создаем объект, который перехватывает вызовы к другому объекту;

Bridge - разделяем абстракцию от интерфейса, позволяя им меняться независимо;

Flyweight - эффективно работаем с огромным количеством схожих объектов.

3) Паттерны поведения: интерпретатор, итератор, команда, наблюдатель, посетитель, посредник, состояние, стратегия, хранитель, цепочка обязанностей, шаблонный метод.

Они определяют эффективные способы взаимодействия различных объектов в системе.

Strategy - описываем набор взаимозаменяемых алгоритмов с единым интерфейсом;

Iterator - обеспечиваем доступ к коллекциям объектов без раскрытия внутреннего устройства этих коллекций;

Observer - создаем объект для отслеживания изменений в подсистеме и нотификации других подсистем;

Memento - сохраняем внутреннее состояние объекта для последующего использования без

нарушения инкапсуляции;

Command - описываем объект, представляющий собой некоторое действие, которое можно выполнить в необходимый момент;

Interpreter - определяем способ вычисления выражений некоторого языка;

Mediator - создаем объект, которые регулирует взаимодействие между набором подсистем;

State - позволяем объекту менять свое поведение при изменении его внутреннего состояния;

Template method - описываем алгоритм, возлагая реализацию некоторых частей алгоритма на подклассы;

Visitor - отделяем алгоритм от структуры, с которыми алгоритм работает;

Chain of responsibility - пропускаем некоторый запрос через набор обработчиков событий, до тех пор пока запрос не будет обработан.

В *Ruby*, *Java* и других объектно-ориентированных языках программирования программистами всего мира уже реализованы и описаны эти практические решения.

Сразу же стоит указать на ограничения по их применению.

Во-первых, шаблоны группы практических решений необходимо применять только тогда, когда имеется четкое понимание необходимости их использования и дополнительная гибкость действительно необходима. Если за гибкость приходится платить усложнением дизайна либо ухудшением производительности, либо подгоном решения под выбранный паттерн, тогда применение паттернов практических решений необоснованно. Необоснованное применение сложных паттернов при решении простых задач усложняет задачу. Поэтому дальнейшее проектирование системы зависит от ответа на этот важный вопрос: использовать или не использовать при решении конкретной задачи готовое практическое решение.

Шаблоны проектирования нужны для того, чтобы помочь реализовать какую-то идею, а не для того, чтобы уместить идею в рамки некоторого паттерна.

Во-вторых, использовать шаблоны практических решений рационально только после изучение конкретного языка программирования.

6.1 CRUD

Create, Read, Update and Delete (создание, чтение, обновление и удаление)

Рассмотрим разработку *CRUD* по пунктам.

1 Маршрутизация (Get-данные)

Прописываем команду

rails generate controller Products create

Это создаст нам контроллер в папке app/controllers/products.rb и в нём будет метод index. И папку для отображения view/products и в ней можно создать имя совпадающие с методом чтобы он использовался автоматический, в нашем случаи создастся create.html.rb (.html.rb используется для view engine в RoR). В нашем случаи библиотека spree сделал всё это за нас командами, описанными выше и создала папкм view/spree (admin – для страниц

администратора и другие страницы для общего отображения обычным пользователям).

2 Создание форм

Создана форма в app/view/spree/*admin/products/new.html.erb*. В которой мы используем хелпер form_for для создания формы.

Код:

new.html.*rb*

```
<%= form_for [:admin, @product], html: { multipart: true } do |f| %>
  <fieldset data-hook="new_product">
    <%= f.field_container :name, class: ['form-group'] do %>
      <%= f.label :name, Spree.t(:name) %> <span class="required">*</span>
      <%= f.text_field :name, class: 'form-control title', required: :required %>
      <%= f.error_message_on :name %>
    <% end %>

    <div data-hook="new_product_attrs" class="row">
      <% unless @product.has_variants? %>
        <div data-hook="new_product_sku" class="col-xs-12 col-md-4">
          <%= f.field_container :sku, class: ['form-group'] do %>
            <%= f.label :sku, Spree.t(:sku) %>
            <%= f.text_field :sku, size: 16, class: 'form-control' %>
            <%= f.error_message_on :sku %>
          <% end %>
        </div>
      <% end %>

      <div data-hook="new_product_prototype" class="col-xs-12 col-md-4">
        <%= f.field_container :prototype, class: ['form-group'] do %>
          <%= f.label :prototype_id, Spree.t(:prototype) %>
          <%= f.collection_select :prototype_id, Spree::Prototype.all, :id, :name, {include_blank: true}, {class:
'select2'} %>
        <% end %>
      </div>

      <div data-hook="new_product_price" class="col-xs-12 col-md-4">
        <%= f.field_container :price, class: ['form-group'] do %>
          <%= f.label :price, Spree.t(:master_price) %> <span class="required">*</span>
          <%= f.text_field :price, value: number_to_currency(@product.price, unit: ''), class: 'form-control',
required: :required %>
          <%= f.error_message_on :price %>
        <% end %>
      </div>

      <div data-hook="new_product_available_on" class="col-xs-12 col-md-4">
        <%= f.field_container :available_on, class: ['form-group'] do %>
          <%= f.label :available_on, Spree.t(:available_on) %>
          <%= f.error_message_on :available_on %>
          <%= f.text_field :available_on, class: 'datepicker form-control' %>
        <% end %>
      </div>

      <div data-hook="new_product_shipping_category" class="col-xs-12 col-md-4">
        <%= f.field_container :shipping_category, class: ['form-group'] do %>
          <%= f.label :shipping_category_id, Spree.t(:shipping_categories) %><span class="required">*</span>
          <%= f.collection_select(:shipping_category_id, @shipping_categories, :id, :name, { include_blank:
Spree.t('match_choices.none') }, { class: 'select2', required: :required }) %>
          <%= f.error_message_on :shipping_category_id %>
        <% end %>
      </div>

    </div>

    <div data-hook="product-from-prototype" id="product-from-prototype">
      <%= render file: 'spree/admin/prototypes/show' if @prototype %>
    </div>

    <%= render partial: 'spree/admin/shared/new_resource_links' %>

  </fieldset>
<% end %>
```

3 Создание миграции в spree

Миграция, модель созданы в разделе 4.

Для создания миграций в RoR можно использовать команду rails migration:create название_миграции

Для миграции необходимо выполнить команду rails db:migrate

Всю схему бд можно найти в db/schema.rb и все миграции в папке db/migrations

4 Работа с мультиязычностью

Для настройки русского интерфейса надо использовать файл config/application.rb:

Код application.rb

```
require_relative 'boot'

require 'rails/all'

# Require the gems listed in Gemfile, including any gems
# you've limited to :test, :development, or :production.
Bundler.require(*Rails.groups)

module ECommerce
  class Application < Rails::Application

    config.to_prepare do
      # Load application's model / class decorators
      Dir.glob(File.join(File.dirname(__FILE__), "../app/**/*_decorator*.rb")) do |c|
        Rails.configuration.cache_classes ? require(c) : load(c)
      end

      # Load application's view overrides
      Dir.glob(File.join(File.dirname(__FILE__), "../app/overrides/*.rb")) do |c|
        Rails.configuration.cache_classes ? require(c) : load(c)
      end
    end

    # Initialize configuration defaults for originally generated Rails version.
    config.load_defaults 5.1
    config.i18n.default_locale = :ru

    # Settings in config/environments/* take precedence over those specified here.
    # Application configuration should go into files in config/initializers
    # -- all .rb files in that directory are automatically loaded.
  end
end
```

Для создания мультиязычных фраз используем встроенную в rails библиотеку i18n и файлы с переводом размещаем в config/locales/ru.yml. Все фразы указываются в натации yml

```
ru:
  hello: 'Привет'
```

Для вставки такой фразии надо в папке view прописать в файле: Spree.t('hello') (функция t означает в RoR translite).

5 Создание ссылок

Для работы с ссылками в rails используется хелпер link в который мы передаём название роута (rails создаёт их автоматический при генерации ресурсов). Пример:
<%= link_to Spree.t(:products), spree.admin_products_url %> # Создаст ссылку на роут admin/product_url

6 Использование partials в шаблонах html

Для переиспользования определённых участков кода можно использовать партиал файлы (это файлы который содержат небольшой html код для определённого участка).

<%= render partial: 'spree/admin/shared/error_messages', locals: { target: @product } %>

Будет использовать код для отображения ошибок

Код 'spree/admin/shared/error_messages'

```
<% if target && target.errors.any? %>
  <div id="errorExplanation" class="errorExplanation alert alert-danger" data-hook>
    <p>
      <strong><%= Spree.t(:errors_prohibited_this_record_from_being_saved, count:
target.errors.size) %>:</strong><br/>
    </p>
    <ul class="ul-with-markup">
     <% target.errors.full_messages.each do |msg| %>
       <li><%= msg %></li>
     <% end %>
    </ul>
  </div>
<% end %>
```

Демонстрация работы

Добавили товар, с именем ESS Палетка теней для век:

Рисунок 6.1.1 – Добавление товара

В базе добавился данный товар с id=25.

Рисунок 6.1.2 – База данных

При нажатии кнопки «Удалить» данный товар удаляется с базы данных и со страницы сайта.

При нажатии кнопки «Редактировать» отображается следующее окно:

Рисунок 6.1.3 – Редактирование товара

8 Результат работы

В данном разделе представлены скриншоты готового каталога.

Рисунок 7.1- Главная страница

Рисунок 7.2 – Раздел «Для губ»

Рисунок 7.3 - Раздел «Для глаз»

Рисунок 7.4 – Раздел «Для бровей»

Рисунок 7.5 – Страница товара

Рисунок 7.6 - Корзина

Рисунок 7.7 - Оформление товара

Рисунок 7.8 - Поиск товара

9 Список литературы

1. Метц, Б. Ruby. Объектно-ориентированное проектирование / Б. Метц – Спб. : Издательство «Питер», 2018. – 304с.

2. Хартл, М. Ruby on Rails для начинающих / М. Хартл, - Москва. : Издательство «ДМК», 2017 - 572с.

3. Никхил, А. Веб-программирование для чайников / А. Никхил - Киев. : Издательство «Диалектика», 2016 - 304с.

4. Фултон, Х. Путь Ruby / Х. Фултон, А. Арко - Москва. : - Издательство «ДМК», 2016 - 664с.

5. Макгаврен, Дж. Head First. Изучаем Ruby / Джей Макгаврен - Спб. : - Издательство «Питер», 2016 - 554с.

6. Гойвертс, Я. Регулярные выражения. Сборник рецептов. / Я. Гойвертс, С. Левитан - Москва. : - Издательство «Символ-Плюс», 2015 - 704с.

7. Макконнелл, Дж. Анализ алгоритмов. / Дж. Макконнелл - Москва. : - Издательство «Техносфера», 2009 - 416с.

8. Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа : http://ru.wikipedia.org/wiki/Веб-приложение/.

9. Blojek [Электронный ресурс]. – Электронные данные. – Режим доступа : http://blojek.info/preimushhestva-i-nedostatki-veb-prilozhenij/.

10. Web-приложения – преимущества и недостатки [Электронный ресурс]. – Электронные данные. – Режим доступа : http://mydiv.net/arts/view-web-prilozhenija_preimuxhestva_i_nedostatki.html.

11. Ожегов, С. И. Толковый словарь русского языка / С. И. Ожегов, Н. Ю. Шведова. – М. : ООО «ИТИ Технологии», 2006. – 944 стр.

12. Буч, Г. Язык UML. Руководство пользователя. 2-е изд. / Г. Буч, Д. Рамбо, И. Якобсон. – М. : ДМК Пресс, 2006. – 496с.

13. Гамма, Э. П75 Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – СПб : Питер, 2001. – 368 с.: ил.

14. Philosoft [Электронный ресурс]. – Электронные данные. – Режим доступа : http://www.philosoft.ru/gost34asconcept.zhtml.

15. Хэнссон, Д. Rails 4. Гибкая разработка веб-приложений. / Д. Хэнссон, С. Руби, Д. Томас - Спб. : - Издательство «Питер Пресс», 2014 - 448с.

16. Мацумото, Ю. Язык программирования Ruby. / Ю. Мацумото, Д. Флэнаган - Спб. : - Издательство «Питер Пресс», 2011 - 496с.

10 ПРИЛОЖЕНИЕ А

ПРИЛОЖЕНИЕ А

(обязательное)

Шаблон home/index.*html.erb (главная страница)*

```
<% @body_id = 'home-page' %>
<% content_for :sidebar do %>
  <div data-hook="homepage_sidebar_navigation">
    <%= render partial: 'spree/shared/taxonomies' %>
  </div>
<% end %>
<div data-hook="homepage_products">
  <% cache(cache_key_for_products) do %>
    <%= render partial: 'spree/shared/products', locals: { products: @products } %>
  <% end %>
</div>
```

Листинг кода создания схемы баз данных

```
ActiveRecord::Schema.define(version: 20180607090520) do
  # These are extensions that must be enabled in order to support this database
  enable_extension "plpgsql"
  create_table "friendly_id_slugs", id: :serial, force: :cascade do |t|
    t.string "slug", null: false
    t.integer "sluggable_id", null: false
    t.string "sluggable_type", limit: 50
    t.string "scope"
    t.datetime "created_at"
    t.datetime "deleted_at"
    t.index ["deleted_at"], name: "index_friendly_id_slugs_on_deleted_at"
    t.index ["slug", "sluggable_type", "scope"], name: "index_friendly_id_slugs_on_slug_and_sluggable_type_and_scope", unique: true
    t.index ["slug", "sluggable_type"], name: "index_friendly_id_slugs_on_slug_and_sluggable_type"
    t.index ["sluggable_id"], name: "index_friendly_id_slugs_on_sluggable_id"
    t.index ["sluggable_type"], name: "index_friendly_id_slugs_on_sluggable_type"
  end
  create_table "spree_addresses", id: :serial, force: :cascade do |t|
    t.string "firstname"
    t.string "lastname"
    t.string "address1"
    t.string "address2"
    t.string "city"
    t.string "zipcode"
    t.string "phone"
    t.string "state_name"
    t.string "alternative_phone"
    t.string "company"
    t.integer "state_id"
    t.integer "country_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["country_id"], name: "index_spree_addresses_on_country_id"
    t.index ["firstname"], name: "index_addresses_on_firstname"
    t.index ["lastname"], name: "index_addresses_on_lastname"
    t.index ["state_id"], name: "index_spree_addresses_on_state_id"
  end
  create_table "spree_credit_cards", id: :serial, force: :cascade do |t|
    t.string "month"
    t.string "year"
    t.string "cc_type"
```

```
  t.string "last_digits"
  t.integer "address_id"
  t.string "gateway_customer_profile_id"
  t.string "gateway_payment_profile_id"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.string "name"
  t.integer "user_id"
  t.integer "payment_method_id"
  t.boolean "default", default: false, null: false
  t.index ["address_id"], name: "index_spree_credit_cards_on_address_id"
  t.index ["payment_method_id"], name: "index_spree_credit_cards_on_payment_method_id"
  t.index ["user_id"], name: "index_spree_credit_cards_on_user_id"
end
create_table "spree_customer_returns", id: :serial, force: :cascade do |t|
  t.string "number"
  t.integer "stock_location_id"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.index ["number"], name: "index_spree_customer_returns_on_number", unique: true
  t.index ["stock_location_id"], name: "index_spree_customer_returns_on_stock_location_id"
end
create_table "spree_gateways", id: :serial, force: :cascade do |t|
  t.string "type"
  t.string "name"
  t.text "description"
  t.boolean "active", default: true
  t.string "environment", default: "development"
  t.string "server", default: "test"
  t.boolean "test_mode", default: true
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.text "preferences"
  t.index ["active"], name: "index_spree_gateways_on_active"
  t.index ["test_mode"], name: "index_spree_gateways_on_test_mode"
end
create_table "spree_orders", id: :serial, force: :cascade do |t|
  t.string "number", limit: 32
  t.decimal "item_total", precision: 10, scale: 2, default: "0.0", null: false
  t.decimal "total", precision: 10, scale: 2, default: "0.0", null: false
  t.string "state"
```

t.decimal "adjustment_total", precision: 10, scale: 2, default: "0.0", null: false
t.integer "user_id"
t.datetime "completed_at"
t.integer "bill_address_id"
t.integer "ship_address_id"
t.decimal "payment_total", precision: 10, scale: 2, default: "0.0"
t.string "shipment_state"
t.string "payment_state"
t.string "email"
t.text "special_instructions"
t.datetime "created_at", null: false
t.datetime "updated_at", null: false
t.string "currency"
t.string "last_ip_address"
t.integer "created_by_id"
t.decimal "shipment_total", precision: 10, scale: 2, default: "0.0", null: false
t.decimal "additional_tax_total", precision: 10, scale: 2, default: "0.0"
t.decimal "promo_total", precision: 10, scale: 2, default: "0.0"
t.string "channel", default: "spree"
t.decimal "included_tax_total", precision: 10, scale: 2, default: "0.0", null: false
t.integer "item_count", default: 0
t.integer "approver_id"
t.datetime "approved_at"
t.boolean "confirmation_delivered", default: false
t.boolean "considered_risky", default: false
t.string "guest_token"
t.datetime "canceled_at"
t.integer "canceler_id"
t.integer "store_id"
t.integer "state_lock_version", default: 0, null: false
t.decimal "taxable_adjustment_total", precision: 10, scale: 2, default: "0.0", null: false
t.decimal "non_taxable_adjustment_total", precision: 10, scale: 2, default: "0.0", null: false
t.index ["approver_id"], name: "index_spree_orders_on_approver_id"
t.index ["bill_address_id"], name: "index_spree_orders_on_bill_address_id"
t.index ["canceler_id"], name: "index_spree_orders_on_canceler_id"
t.index ["completed_at"], name: "index_spree_orders_on_completed_at"
t.index ["confirmation_delivered"], name: "index_spree_orders_on_confirmation_delivered"
t.index ["considered_risky"], name: "index_spree_orders_on_considered_risky"
t.index ["created_by_id"], name: "index_spree_orders_on_created_by_id"
t.index ["guest_token"], name: "index_spree_orders_on_guest_token"
t.index ["number"], name: "index_spree_orders_on_number", unique: true

```
    t.index ["ship_address_id"], name: "index_spree_orders_on_ship_address_id"
    t.index ["store_id"], name: "index_spree_orders_on_store_id"
    t.index ["user_id", "created_by_id"], name: "index_spree_orders_on_user_id_and_created_by_id"
  end
  create_table "spree_payments", id: :serial, force: :cascade do |t|
    t.decimal "amount", precision: 10, scale: 2, default: "0.0", null: false
    t.integer "order_id"
    t.string "source_type"
    t.integer "source_id"
    t.integer "payment_method_id"
    t.string "state"
    t.string "response_code"
    t.string "avs_response"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "number"
    t.string "cvv_response_code"
    t.string "cvv_response_message"
    t.index ["number"], name: "index_spree_payments_on_number", unique: true
    t.index ["order_id"], name: "index_spree_payments_on_order_id"
    t.index ["payment_method_id"], name: "index_spree_payments_on_payment_method_id"
    t.index ["source_id", "source_type"], name: "index_spree_payments_on_source_id_and_source_type"
  end
  create_table "spree_stores", id: :serial, force: :cascade do |t|
    t.string "name"
    t.string "url"
    t.text "meta_description"
    t.text "meta_keywords"
    t.string "seo_title"
    t.string "mail_from_address"
    t.string "default_currency"
    t.string "code"
    t.boolean "default", default: false, null: false
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index "lower((code)::text)", name: "index_spree_stores_on_lower_code", unique: true
    t.index ["default"], name: "index_spree_stores_on_default"
    t.index ["url"], name: "index_spree_stores_on_url"
  end
  create_table "spree_taxons", id: :serial, force: :cascade do |t|
```

```
    t.integer "parent_id"
    t.integer "position", default: 0
    t.string "name", null: false
    t.string "permalink"
    t.integer "taxonomy_id"
    t.integer "lft"
    t.integer "rgt"
    t.string "icon_file_name"
    t.string "icon_content_type"
    t.integer "icon_file_size"
    t.datetime "icon_updated_at"
    t.text "description"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "meta_title"
    t.string "meta_description"
    t.string "meta_keywords"
    t.integer "depth"
    t.index ["lft"], name: "index_spree_taxons_on_lft"
    t.index ["name"], name: "index_spree_taxons_on_name"
    t.index ["parent_id"], name: "index_taxons_on_parent_id"
    t.index ["permalink"], name: "index_taxons_on_permalink"
    t.index ["position"], name: "index_spree_taxons_on_position"
    t.index ["rgt"], name: "index_spree_taxons_on_rgt"
    t.index ["taxonomy_id"], name: "index_taxons_on_taxonomy_id"
  end
  create_table "spree_users", id: :serial, force: :cascade do |t|
    t.string "encrypted_password", limit: 128
    t.string "password_salt", limit: 128
    t.string "email"
    t.string "remember_token"
    t.string "persistence_token"
    t.string "reset_password_token"
    t.string "perishable_token"
    t.integer "sign_in_count", default: 0, null: false
    t.integer "failed_attempts", default: 0, null: false
    t.datetime "last_request_at"
    t.datetime "current_sign_in_at"
    t.datetime "last_sign_in_at"
    t.string "current_sign_in_ip"
    t.string "last_sign_in_ip"
```

```
    t.string "login"
    t.integer "ship_address_id"
    t.integer "bill_address_id"
    t.string "authentication_token"
    t.datetime "reset_password_sent_at"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "spree_api_key", limit: 48
    t.datetime "remember_created_at"
    t.datetime "deleted_at"
    t.string "confirmation_token"
    t.datetime "confirmed_at"
    t.datetime "confirmation_sent_at"
    t.index ["bill_address_id"], name: "index_spree_users_on_bill_address_id"
    t.index ["deleted_at"], name: "index_spree_users_on_deleted_at"
    t.index ["email"], name: "email_idx_unique", unique: true
    t.index ["ship_address_id"], name: "index_spree_users_on_ship_address_id"
    t.index ["spree_api_key"], name: "index_spree_users_on_spree_api_key"
  end
end
```

Заключение

В данной курсовой работе был разработан сайт-визитка с каталогом товаров для мебельного магазина. Основными технологиями проекта являются Ruby on Rails, PostgreSQL. Программная среда проекта - Puma. В результате работы, поставленные задачи были выполнены. А именно, описан проект: интернет-каталог косметики для того чтобы познакомить пользователя с перечнем продаваемых товаров, информацией о данных товарах. Установлен Git. Git- мощная и сложная распределенная система контроля версий. Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Также проект размещен на удаленном репозитории. Ссылка на репозиторий: https://github.com/viktoriya-gr/website_of_shop. Проект создан на основе фреймворка Ruby on Rails. Он берет на себя аутентификацию, роутинг, работу с сессиями, кеширование, внедрение зависимостей и многое другое. Также организован модуль авторизации с помощью данного фреймворка. Для решения распространённых проблем, возникающих при разработке программного обеспечения используется - шаблон проектирования. В данной пояснительной записки показано на примерах создания CRUD и отправки оповещений на e-mail. В целом предлагаемый интерне-каталог позволит значительно повысить эффективность деятельности магазина, так как интернет-каталог содержит основную информацию о товарах, прайс-листы, что способствует привлечению новых покупателей и как следствие получение наибольшей выгоды.

Список использованных источников

Приложения

1. [электронный документ] 5ed0327e076ec_Курсовая СТПИС.docx
2. [электронный документ] 5ed0327e473fd_Титул ПЗ СТПИС.docx
3. [электронный документ] 5ed373ffe2402_List_zadania.docx