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
03.07.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Разработка интернет-магазина по продаже обуви

БГУИР КР 1-40 05 01-10 № 129 ПЗ

Студент (подпись студента) Д.О.Савицкий

Курсовая работа
представлена на проверку
03.07.2025
_____
(подпись студента)

Минск 2025

Реферат

БГУИР КР 1-40 05 01-10 № 129 ПЗ, гр. 814303

Д.О.Савицкий, Разработка интернет-магазина по продаже обуви, Минск: БГУИР - 2025.

Пояснительная записка 54641 с., 14 рис., 0 табл.

Ключевые слова: ИНТЕРНЕТ-МАГАЗИН, ВЕБ-ПРИЛОЖЕНИЕ, ИНТЕРФЕЙС

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

- Предмет: создание веб-приложения Объект: шаблоны проектирования, разработка пользовательского интерфейса. Цель: верстка и программирование интернет-магазина по продаже обуви 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, чтобы вся важная информация, была на главной странице. Область применения результатов: удовлетворение пользователей Интернета, нуждающихся в заказе обуви, не выходя из дома, а также для оптимизации их времени. Ссылка на онлайн-репозиторий GitHub: https://github.com/dmitrysavitski/Shoes-store

- Subject: creating a web application Object: design templates, user interface development. Goal: layout and programming of an online Shoe store using the MVC architectural design template. Methodology of 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, and work with the database was studied and applied. Results: developed an online store with convenient functionality. The site interface was simplified as much as possible and reasonably informative, so that all important information was on the main page. The scope of the results: to satisfy Internet users who need to order shoes from the comfort of their homes, as well as to optimize their time. Link to the github online repository: https://github.com/dmitrysavitski/Shoes-store

Содержание

Введение
1 Описание проекта
2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ
3 ИНСТРУМЕНТАРИЙ
4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC
5 ШАБЛОНЫ ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ
6 Приложение А
Заключение
Список использованных источников
Приложения

Введение

В настоящее время Интернет становиться все более развитой средой для осуществления коммуникаций с потребителями, а также инструментом для упрощения их жизней. В то же время, существенным является и тот факт, что Интернет становиться удобным и достаточно доступным сервисом для покупки любых вещей. Интернет-магазины приобретают всю большую популярность у покупателей, и в настоящее время online торговля имеет ряд преимуществ. Потребитель может удалённо выбрать соответствующий товар и приобрести его, так же интернет-магазин доступен в любое время суток. Преимущества для компаний: развитие бизнеса, нет необходимости покупать торговую площадь, минимальное количество рабочих кадров. При этом, с помощью интернет-порталов можно с легкостью находить информацию, представленную в доступном визуале, который сможет понять любой пользователь. Большим преимуществом создания интернет-магазина является то, что стартовый капитал, необходимый для открытия и запуска торгового предприятия может быть в разы ниже, нежели при организации такого же магазина в реальности. В курсовой работе поставлена цель: разработка интернет магазина. Для достижения поставленной цели необходимо решить следующие задачи: 1 Изучить теоретическую базу по разработке веб-приложений; 2 Обосновать выбор метода разработки; 3 Установить и настроить инструментарий; 4 Разработать и реализовать веб-приложение с удобным интерфейсом.

1 Описание проекта

Курсовой проект представляет собой интернет-магазин обуви с возможностью регистрации и авторизации в личный кабинет, просмотра каталога товаров, добавления товара в «Корзину».

Веб-приложение разработано с помощью языка C#, технологии .Net, фреймворка Bootstrap.

Вся информация, которая отображается на сайте, хранится при помощи баз данных, доступ к которым осуществляется по запросам.

Сайт реализован только на одном языке: английском.

В приложении реализован архитектурный шаблон MVC.

1.1 Клиентская часть

На главной странице интернет-магазина представлены вкладки, что обеспечивает простую навигацию по сайту и простоту использования (рис. 1).

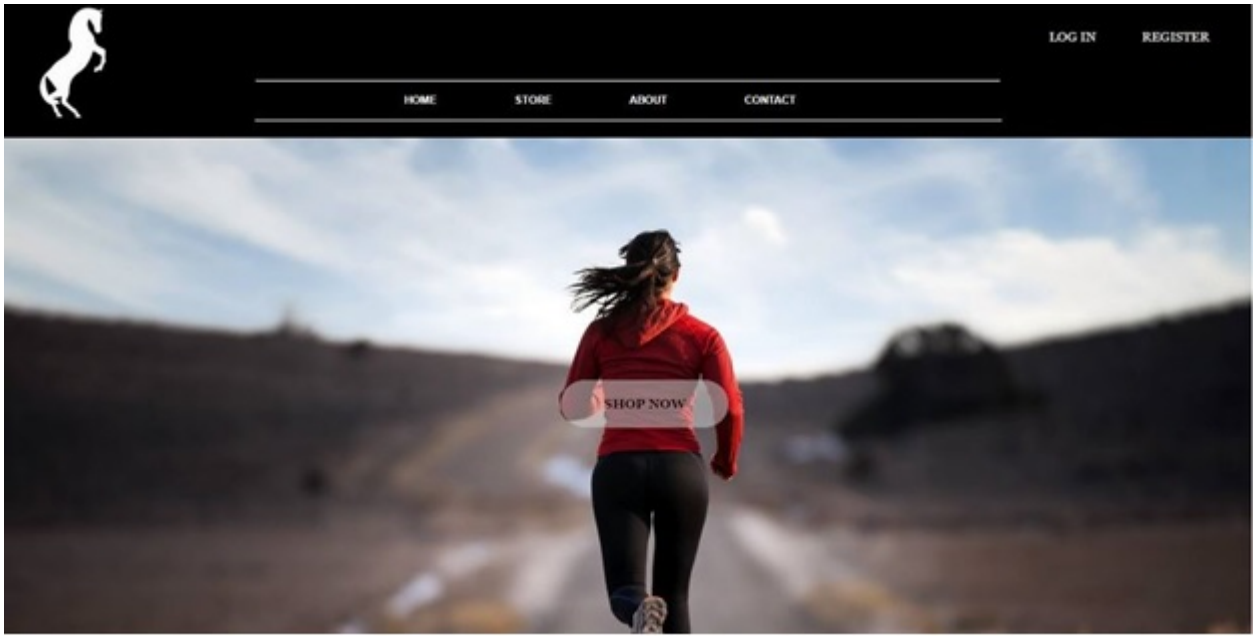

Рисунок 1 – Главная страница

В верхнем правом углу экрана есть возможность выбрать действие, связанное с аккаунтом пользователя: войти на сайт или зарегистрироваться.

Рисунок 2 – Клавиши, связанные с аккаунтом

При нажатии на клавишу “Store” или “Shop now” пользователя перенаправляет в каталог товаров (рис. 3).

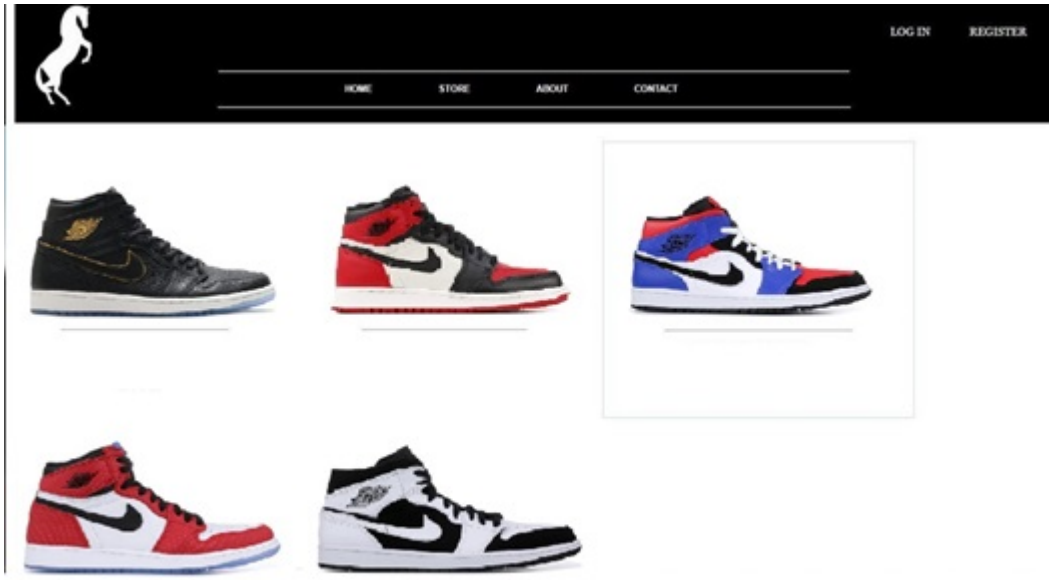

Рисунок 3 – Каталог товаров

При выборе товара, пользователь перенаправляется на страницу выбранного товара, где он

может увидеть фотографию, название модели и цену товара (Рис 4).

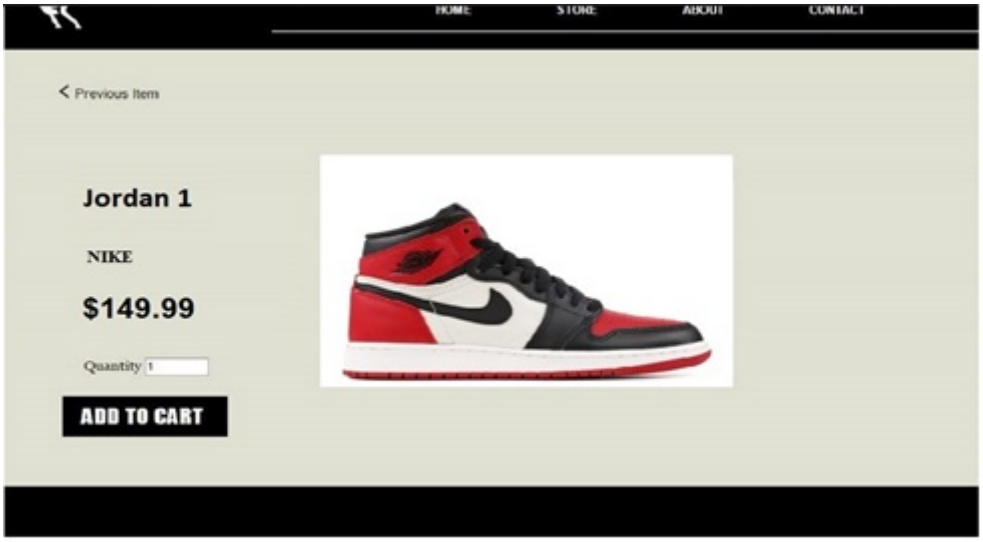

Рисунок 4 – Страница товара

При нажатии кнопки “ADD TO CART” данный товар добавляется в корзину, где он может увидеть цену товара, количество, которое он хотел бы купить и итоговую цену, а также кнопку “Continue Shopping” (Рис 5).

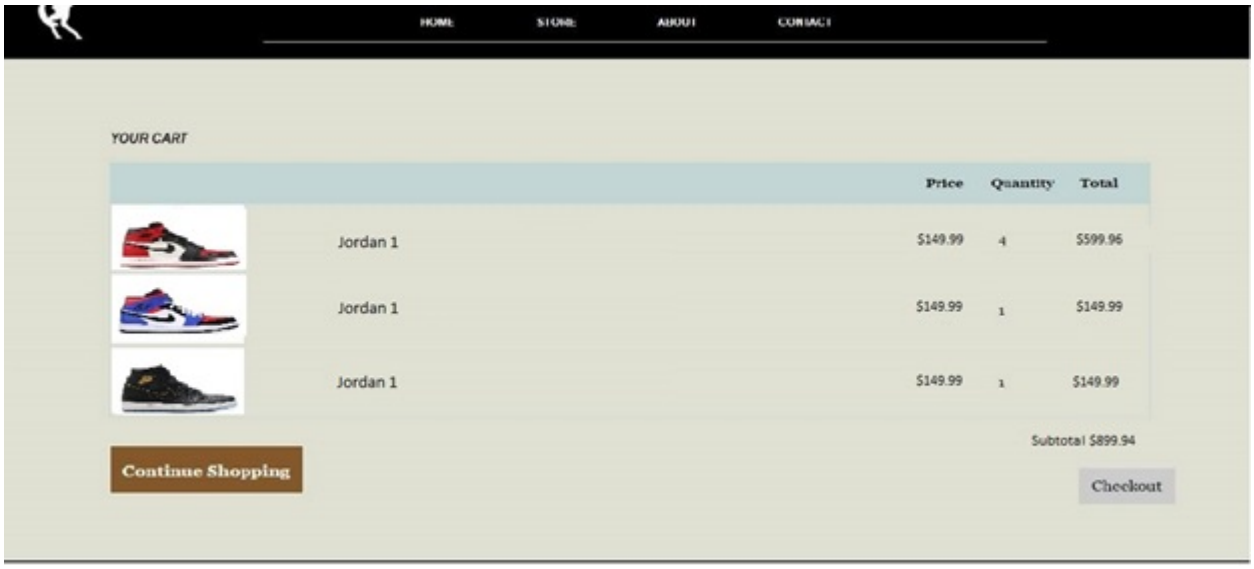

Рисунок 5 – Интерфейс “Корзины” покупателя

После нажатия кнопки “Checkout” пользователя перенаправляет на страницу заполнения данных о доставке (рис. 6).

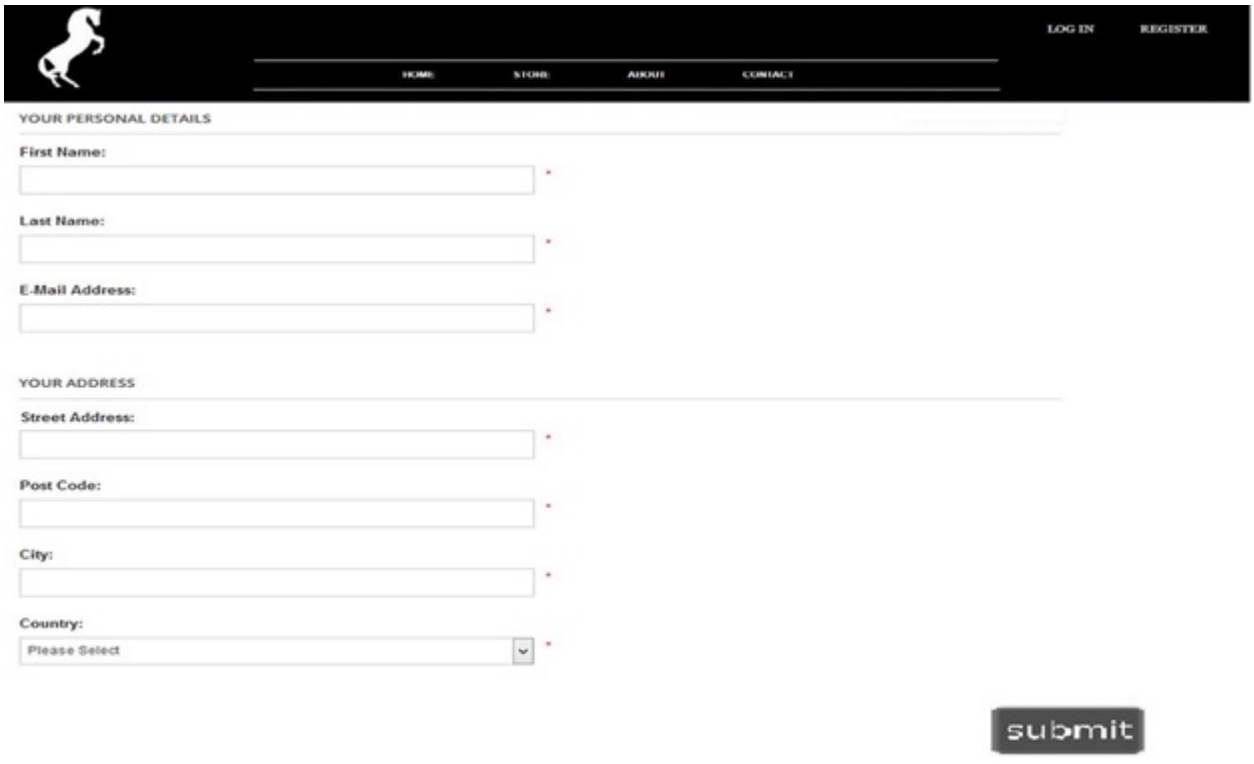

Рисунок 6 – Интерфейс вкладки заполнения данных о доставке

При нажатии кнопки “Register” пользователя перенаправляется на страницу регистрации.

На рисунке 7 показано какие данные пользователю требуется ввести для регистрации.

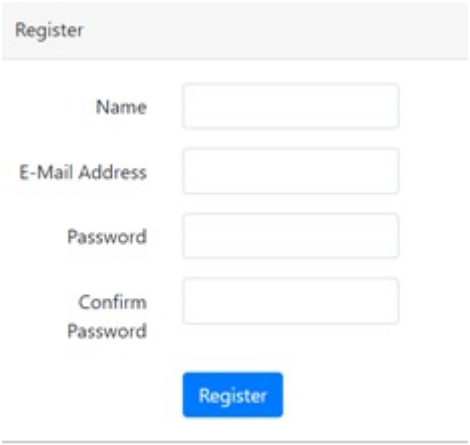

Рисунок 7 – Интерфейс страницы регистрации

1.2 Серверная часть

При авторизации, есть возможность входа в систему как администратора. Администратор имеет различные возможности, которые отличатся от возможностей обычного пользователя.

Интерфейс для администратора не сильно отличается от обычного пользовательского, однако администратор может добавлять новые товары, удалять и редактировать уже имеющиеся товары, что показано на рисунках 8,9,10.

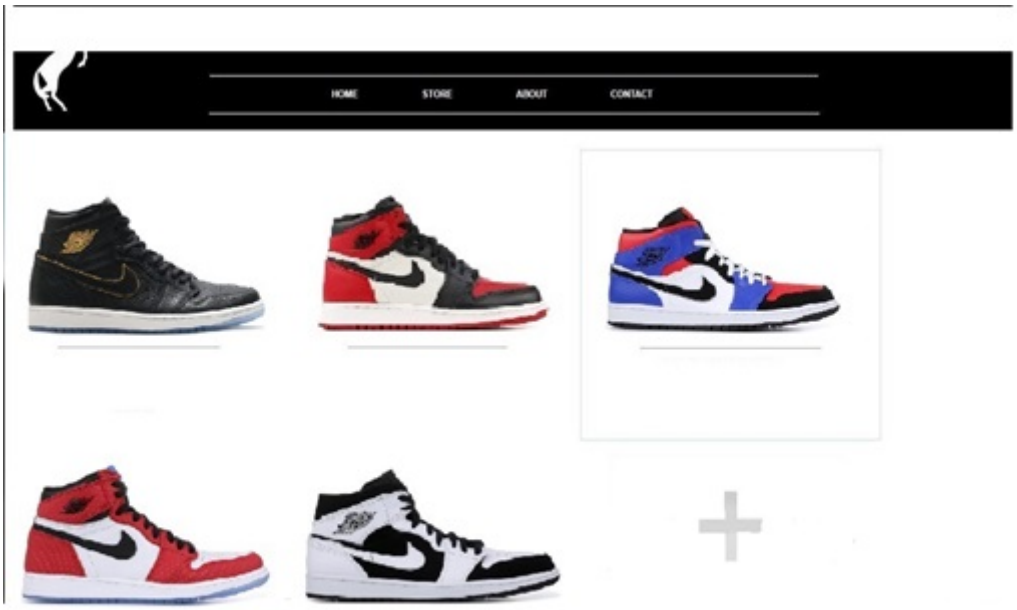

Рисунок 8 – Интерфейс интернет-магазина для администратора

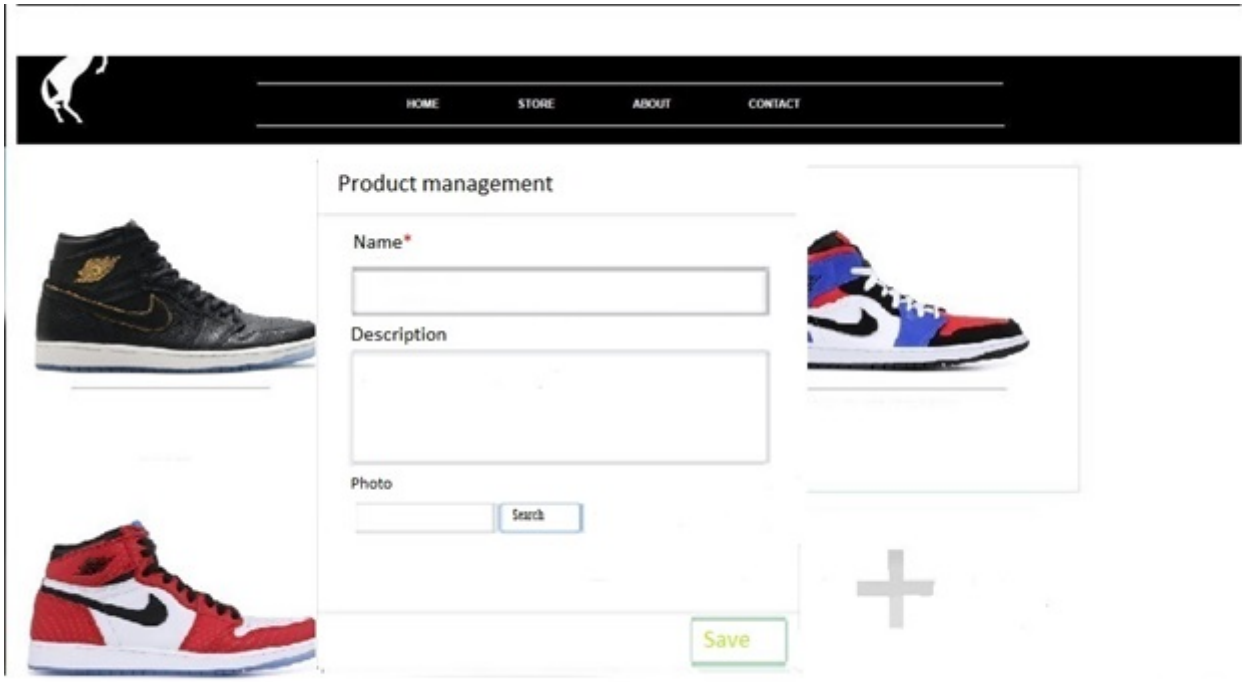

Рисунок 9 – Создание нового товара

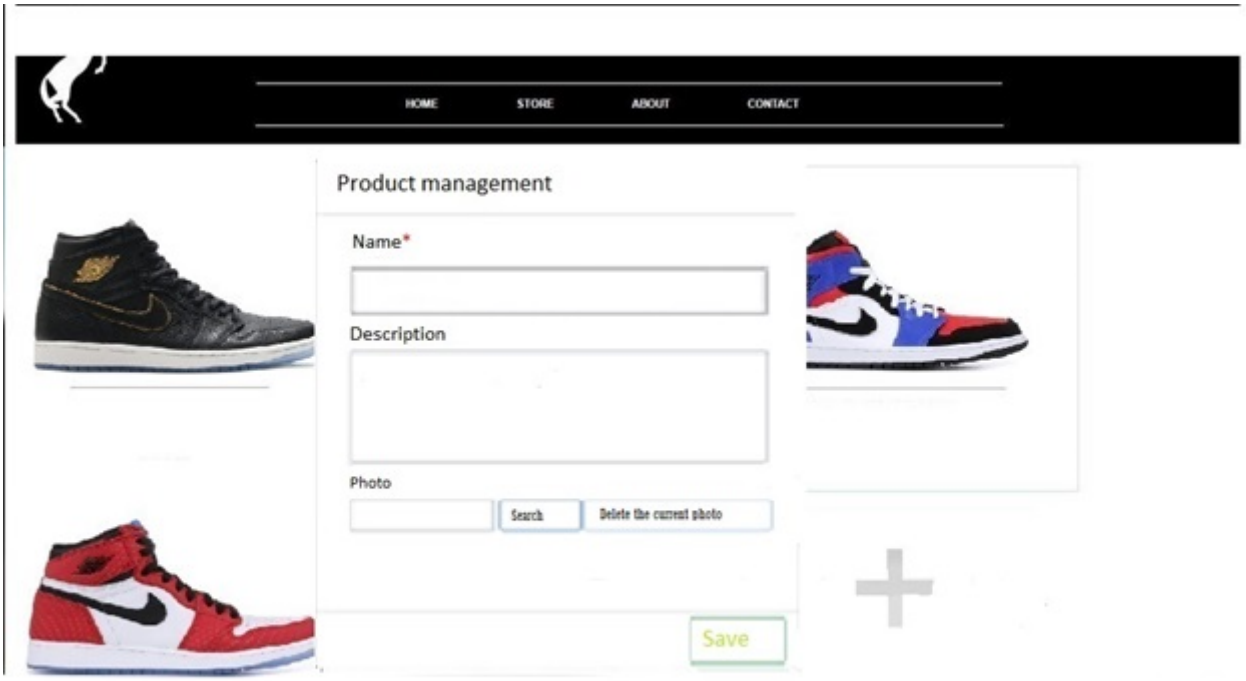

Рисунок 10 – Редактирование имеющегося товара

2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

2.1 Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) - это метод программирования, при использовании которого главными элементами программ являются объекты. В языках программирования понятие объекта реализовано как совокупность свойств (структур данных, характерных для данного объекта), методов их обработки (подпрограмм изменения их свойств) и событий, на которые данный объект может реагировать и, которые приводят, как правило, к изменению свойств объекта. Объединение данных и свойственных им процедур обработки в одном объекте, называется инкапсуляцией и является одним из важнейших принципов ООП.

Другим фундаментальным понятием является класс.

Класс - это шаблон, на основе которого может быть создан конкретный программный объект, он описывает свойства и методы, определяющие поведение объектов этого класса. Каждый конкретный объект, имеющий структуру этого класса, называется экземпляром класса.

Следующими важнейшими принципами ООП являются наследование и полиморфизм.

Наследование - такая организация классов, которая предусматривает создание новых классов на базе существующих и позволяет классу потомку иметь (наследовать) все свойства класса – родителя.

Полиморфизм - означает, что рожденные объекты обладают информацией о том, какие методы они должны использовать в зависимости от того, в каком месте цепочки (дерева классов) они находятся, иными словами это концепция, реализующая "множество методов в одном интерфейсе".

Другим важнейшим принципом ООП является модульность. Модульность - это такая организация объектов, когда они заключают в себе полное определение их характеристик, никакие определения методов и свойств не должны располагаться вне его, это делает возможным свободное копирование и внедрение одного объекта в другие.

2.2 Язык программирования C#

C# - современный объектно-ориентированный**.** C# поддерживает все три «столпа» объектно-ориентированного программирования: инкапсуляцию, наследование и полиморфизм. Кроме того, в нем была реализована автоматическая «сборка мусора», обработки исключений, динамическое связывание.

Как и Java, C# изначально предназначался для веб-разработки, и примерно 75% его синтаксических возможностей аналогичны языку программирования Java. Его также называют «очищенной версией Java». 10% позаимствовано из C++, а 5% – из Visual Basic. И около 10%

C# – это реализация собственных идей разработчиков.

2.3 Технология .Net

Платформа .NET Framework — это технология, которая поддерживает создание и выполнение веб-служб и приложений Windows. При разработке платформы .NET Framework учитывались следующие цели:

Обеспечение согласованной объектно-ориентированной среды программирования для локального сохранения и выполнения объектного кода, для локального выполнения кода, распределенного в Интернете, либо для удаленного выполнения.
Обеспечение среды выполнения кода, минимизирующей конфликты при развертывании программного обеспечения и управлении версиями.
Обеспечение среды выполнения кода, гарантирующей безопасное выполнение кода, включая код, созданный неизвестным или не полностью доверенным сторонним изготовителем.
Обеспечение среды выполнения кода, исключающей проблемы с производительностью сред выполнения сценариев или интерпретируемого кода.
Обеспечение единых принципов разработки для разных типов приложений, таких как приложения Windows и веб-приложения.
Взаимодействие на основе промышленных стандартов, которое гарантирует интеграцию кода платформы .NET Framework с любым другим кодом.

Библиотека классов платформы .NET Framework представляет собой коллекцию типов, которые тесно интегрируются со средой CLR. Библиотека классов является объектно-ориентированной. Она предоставляет типы, от которых управляемый код пользователя может наследовать функции. Это не только упрощает работу с типами .NET Framework, но и сокращает время изучения новых средств платформы .NET Framework. Кроме того, компоненты независимых производителей можно легко объединять с классами платформы .NET Framework.

2.4 Фреймворк Bootstrap

Bootstrap это фреймворк включающий в себя HTML– и CSS– шаблоны для оформления веб–интерфейса. Изначально Bootstrap носил название Twitter Blueprint, так как создавался компанией Twitter. Это отличный пример того, как разрабатываемый для внутреннего использования инструмент становится в итоге востребованным по всему миру. Основатели Bootstrap – сотрудники компании Twitter Марк Отто и Якоб Торнтон – по мере разработки поняли, что проект может стать чем–то большим, чем просто одним из внутренних инструментов компании. После того, как проект был выложен в открытый доступ, его стали

использовать многие разработчики.

Использование фреймворка позволяет уменьшить количество времени, затрачиваемого на разработку сайта, сделать сайт адаптивным, позволяет сайту отображаться одинаково во всех браузерах и также он довольно прост и удобен в использовании. Однако, при этом все сайты, написанные на Bootstrap шаблонны и похожи друг на друга. Он не позволяет поддерживать сайт или добавлять новый функционал так как при добавлении чего–либо нового весь код написанный на фреймворке придется переписывать.

3 ИНСТРУМЕНТАРИЙ

3.1 IDE Microsoft Visual Studio 2019

При реализации курсовой работы я использовал IDE Microsoft Visual Studio 2019.

IDE(Integrated Drive Electronics или же интегрированная среда разработки) – комплекс программных средств, используемый программистами для разработки программного обеспечения. Среда разработки включает в себя:

- текстовый редактор;
- компилятор;
- отладчик и прочее.

Суть IDE заключается в объединении нескольких компонентов для максимальной концентрации программиста на решении алгоритмических задач, без потерь времени на сохранение файла в текстовом редакторе, затем вызове компилятора и так далее. Таким образом, повышается производительность труда разработчика. Также огромным преимуществом среды разработки является наличие статического анализатора кода, который позволяет выявить ошибки в синтаксисе и другие мелкие ошибки по ходу написания программы.

Microsoft Visual Studio 2019 – это среда разработки от компании Microsoft. Она включает в себя поддержку многих компонентов таких как: Visual Basic, Visual C++, Visual C#, Python, .NET и её компоненты, имеет встроенный веб–сервер который может пригодиться при веб–разработке, присутствует интеграция с Unity и Unreal Engine. Можно разрабатывать как консольные приложения, так и приложения с графическим интерфейсом. Visual Studio позволяет создавать и подключать сторонние дополнения для расширения функциональности при помощи NuGet, также присутствует интеграция с системой контроля версий Git.

Основные возможности и преимущества программы:

- Имеются встроенные отладчик и командная строка.
- Поддержка практически всех языков программирования.
- Наличие встроенной библиотеки элементов кода.
- Автозавершение при вводе кода.
- Добавление в библиотеку собственных сниппетов.
- Подсветка синтаксиса.

Одновременная работы с несколькими проектами.
Поддержка многооконного и двухпанельного режимов.
Расширение функционала с помощью плагинов.
Интеграция с Visual Studio Team Services, GitHub и GIT.
Наличие встроенных средств для тестирования, сборки, упаковки и развертывания приложений.
Публикация созданных программных продуктов в Microsoft Azure (через посредство Visual Studio Team Services).
Интегрированная система подсказок.
Командная работа над проектами.
Широкий набор настроек и кроссплатформенность.

3.2 Использование системы контроля версий GIT

Система контроля версий(СКВ) – это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Она позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда и многое другое.

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.

На рисунке 11 представлена схема локального контроля версий.

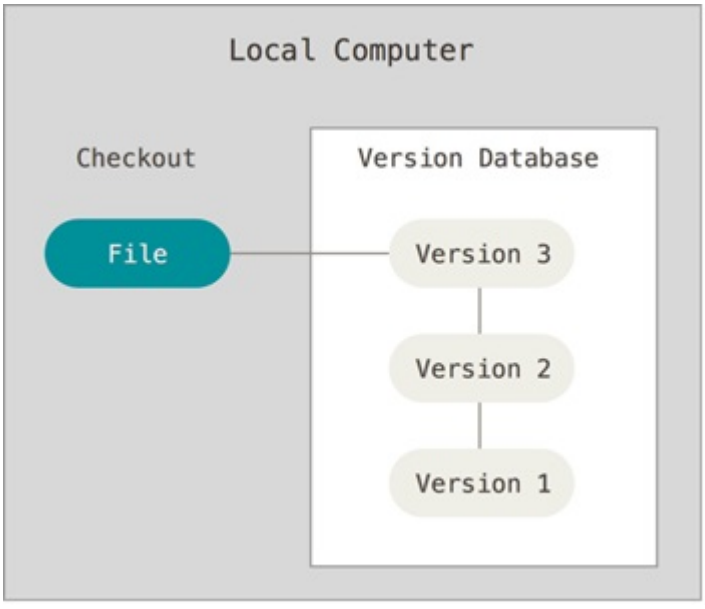

Рисунок 11 – Локальный контроль версий

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

Следующая серьёзная проблема, с которой сталкиваются люди, – это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.

На рисунке 12 представлена схема централизованного контроля версий.

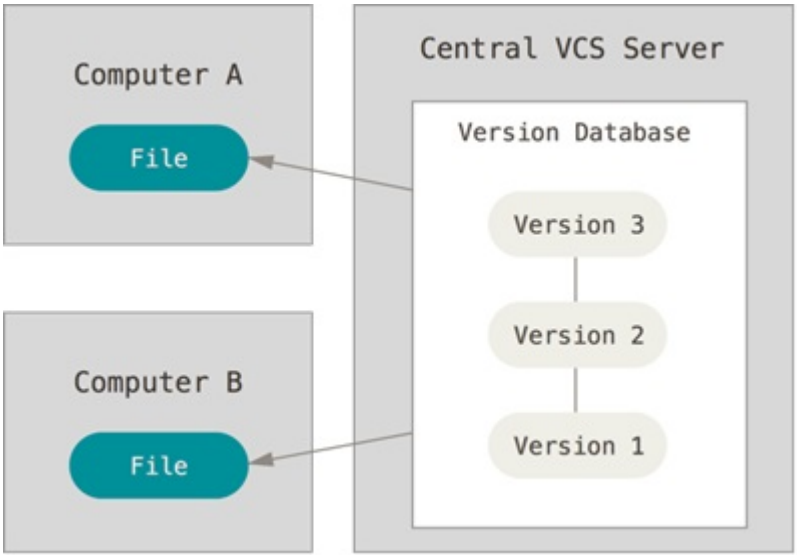

Рисунок 12 – Централизованный контроль версий

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус – это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками.

Существуют распределённые системы контроля версий (РСКВ). В РСКВ (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) – они полностью копируют репозиторий. В этом

случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.

На рисунке 13 изображена схема распределённого контроля версий.

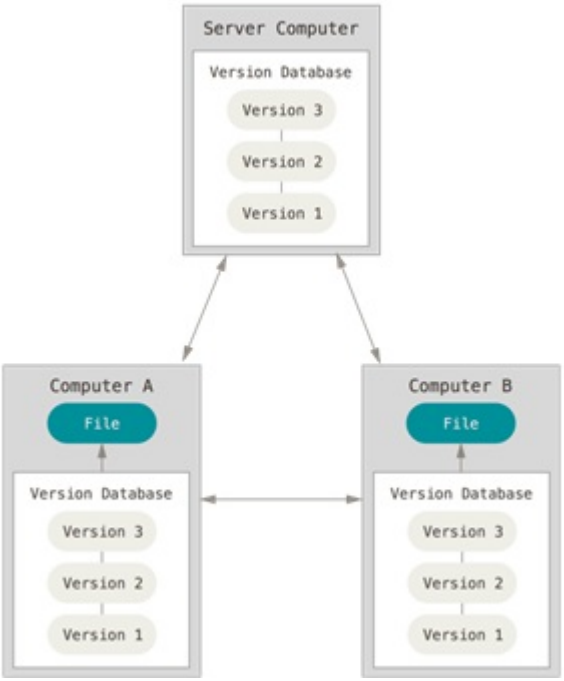

Рисунок 13 – Распределённый контроль версий

Более того, многие РСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому разработчики могут работать с различными группами людей, применяя различные подходы единовременно в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

В нашем курсовом проекте использовался онлайн-репозиторий GitHub. Ссылка на него: https://github.com/dmitrysavitski/Shoes-store

4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

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.

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. Что такое пользователь для приложения — сообщение или книга? Только данные, которые должны быть обработаны в соответствии с правилами (дата не может указывать в будущее, e-mail должен быть в определённом формате, имя не может быть длиннее X символов, и так далее).

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 (форум, магазин, банк, и тому подобное). Контроллер содержит в основном организационную логику для самого приложения (очень похоже на ведение домашнего хозяйства).

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.

Веб приложение обычно состоит из набора контроллеров, моделей и видов. Контроллер может быть устроен как основной, который получает все запросы и вызывает другие контроллеры для выполнения действий в зависимости от ситуации.

Рассмотрим классическую схему веб-приложения, которая отражена на рисунке 14:

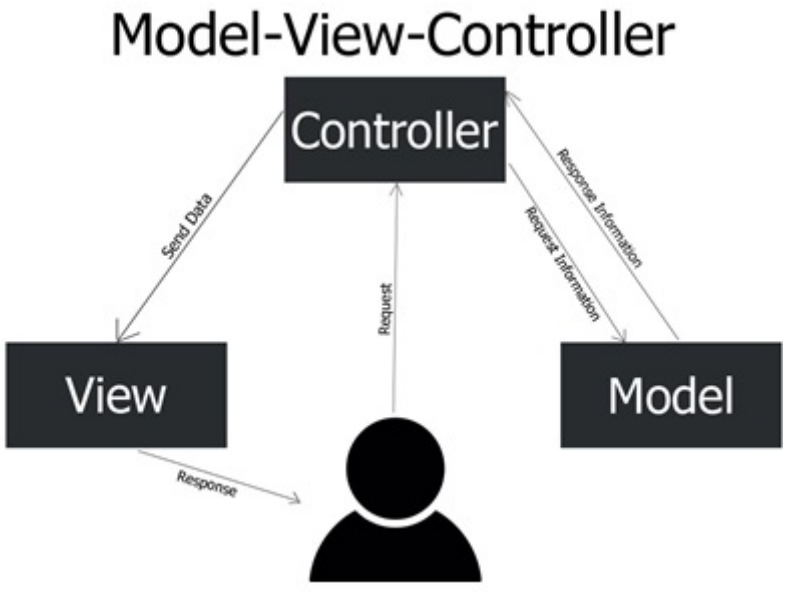

Рисунок 14 – Принцип работы контроллера, модели и элементов представления в шаблоне проектирования MVC

На этом и последующем рисунках пунктирными линиями показана управляющая информация (такая, например, как ID запрашиваемой записи блога или товара в магазине), а сплошными – собственно данные приложения (которые могут храниться в БД, или в виде файлов на диске, или даже, возможно, в оперативной памяти – этот вопрос лежит за пределами

паттерна MVC). В применении к вебу запрос и ответ ходят по HTTP, поэтому можно условно считать, что на этом рисунке пунктиром обозначены заголовки HTTP-запроса и ответа, а сплошными линиями – их тела.

Вопрос о том, кто должен проверять на валидность и права доступа входные данные (Контроллер или Модель), является предметом достаточно многочисленных споров, поскольку паттерн MVC не описывает таких деталей. Это значит, что в этом вопросе выбор за вами (или за вас его сделали авторы вашего любимого фрейvворка или CMS).

На практике придерживаются такого подхода: контроллер проверяет входные данные на предмет «общей» (т.е. независящей от конкретного запроса) корректности, соответствие требованиям Модели к валидности сохраняемых данных проверяет соответствующий метод Модели, а права доступа – метод Access отдельного класса User.

Для вызова Представления в PHP иногда проектируется специальный класс (а то и несколько классов), например, View (это часто встречается в описаниях MVC в реализации того или иного фреймворка), однако это не является требованием MVC.

Также файлы представлений часто называют шаблонами, а при использовании так называемых шаблонизаторов, роль Представления играет сам шаблонизатор, а шаблоны (т.е. файлы, содержащие непосредственно HTML-разметку) в некоторых фреймворках называют layouts.

5 ШАБЛОНЫ ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ

Паттерн представляет определенный способ построения программного кода для решения часто встречающихся проблем проектирования.

Существует множество различных паттернов, которые решают разные проблемы и выполняют различные задачи. Но по своему действию их можно объединить в ряд групп. В основу классификации основных паттернов положена цель или задачи, которые определенный паттерн выполняет.

Порождающие паттерны — это паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов. Среди них выделяются следующие:

Абстрактная фабрика (Abstract Factory)
Строитель (Builder)
Фабричный метод (Factory Method)
Прототип (Prototype)
Одиночка (Singleton)
Простая фабрика (Simple Factory)

Другая группа паттернов - структурные паттерны - рассматривает, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты. К таким шаблонам относятся:

Адаптер (Adapter);

Мост (Bridge);
Компоновщик (Composite);
Декоратор (Decorator);
Фасад (Facade);
Приспособленец (Flyweight);
Заместитель (Proxy).

Третья группа паттернов называются поведенческими - они определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение. Среди подобных шаблонов можно выделить следующие:

Цепочка обязанностей (Chain of responsibility);
Команда (Command);
Интерпретатор (Interpreter) и т.д.

Существуют и другие классификации паттернов в зависимости от того, относится паттерн к классам или объектам.

Паттерны классов описывают отношения между классами посредством наследования. Отношения между классами определяются на стадии компиляции. К таким паттернам относятся:

Фабричный метод (Factory Method);
Интерпретатор (Interpreter);
Шаблонный метод (Template Method);
Адаптер (Adapter).

Другая часть паттернов - паттерны объектов описывают отношения между объектами. Эти отношения возникают на этапе выполнения, поэтому обладают большей гибкостью. К паттернам объектов относят следующие:

Абстрактная фабрика (Abstract Factory);
Строитель (Builder);
Прототип (Prototype) и т.д.

Использование шаблонов проектирования отличает сильного программиста. Использование шаблонов дает возможность тратить меньше времени используя уже проверенные решения, без изобретения костылей. Вы делаете меньше просчетов при проектировании используя стандартные типовые решения. При работе в команде вам не требуется объяснять другим программистам что именно вы сделали так как вы можете просто назвать паттерн.

Однако у паттернов есть и свои минусы. Например, неэффективное использование. При знакомстве с паттернами множество программистов начинает их использовать не особо задумываясь, подходят они под конкретную задачу или нет. Также существует критическая необходимость приспосабливать паттерны к реалиям проекта, а не реализовывать их просто ко книжке.

Каждый из паттернов решает свою проблему, но при этом может разрушать принципы программирования. Рассмотрим по одному паттерну из каждого шаблона.

Одиночка или Singleton – это довольно популярный порождающий паттерн проектирования. Этот паттерн решает сразу две проблемы: гарантирует наличие единственного экземпляра

класса, предоставляет глобальную точку доступа. При этом он нарушает *принцип единственной ответственности* – принцип ООП, обозначающий, что каждый объект должен иметь одну ответственность и эта ответственность должна быть полностью инкапсулирована в класс.

Для реализации одиночки достаточно скрыть конструктор по умолчанию и создать публичный статический метод, который и будет контролировать жизненный цикл объекта.

Применять этот паттерн стоит, когда в программе должен быть единственный экземпляр какого–то класса, доступный всем клиентам. Или, когда вам хочется иметь больше контроля над глобальными переменными.

Стратегия(Strategy) – это поведенческий паттерн. Он предлагает определить семейство схожих алгоритмов, которые часто изменяются или расширяются, и вынести их в собственные классы, называемые *стратегиями*. Вместо того, чтобы изначальный класс сам выполнял тот или иной алгоритм, он будет играть роль контекста, ссылаясь на одну из стратегий и делегируя ей выполнение работы. Чтобы сменить алгоритм, вам будет достаточно подставить в контекст другой объект-стратегию. Важно, чтобы все стратегии имели общий интерфейс. Используя этот интерфейс, контекст будет независимым от конкретных классов стратегий. С другой стороны, вы сможете изменять и добавлять новые виды алгоритмов, не трогая код контекста.

Использовать этот класс стоит, когда вам нужно использовать разные вариации какого-то алгоритма внутри одного объекта, когда у вас есть множество похожих классов, отличающихся только некоторым поведением, когда вы не хотите показывать детали реализации алгоритмов для других классов. Однако использование этого паттерна связано с некоторыми неудобствами. Он усложняет программу за счет дополнительных классов. Помимо этого, для использования этого паттерна клиент должен знать в чем состоит разница между стратегиями чтобы выбрать подходящую.

Адаптер – это структурный паттерн проектирования. С помощью этого паттерна вы можете создать объект-переводчик, который трансформирует интерфейс или данные одного объекта в такой вид, чтобы он стал понятен другому объекту. При этом адаптер оборачивает один из объектов, так что другой объект даже не знает о наличии первого. Например, вы можете обернуть объект, работающий в метрах, адаптером, который бы конвертировал данные в футы. Адаптеры могут не только переводить данные из одного формата в другой, но и помогать объектам с разными интерфейсами работать сообща. Это работает так:

1. Адаптер имеет интерфейс, который совместим с одним из объектов.
2. Поэтому этот объект может свободно вызывать методы адаптера.
3. Адаптер получает эти вызовы и перенаправляет их второму объекту, но уже в том формате и последовательности, которые понятны второму объекту.

Применять этот паттерн стоит когда вы хотите использовать сторонний класс, но его интерфейс не соответствует остальному коду приложения и Когда вам нужно использовать несколько существующих подклассов, но в них не хватает какой–то общей функциональности, причём расширить суперкласс вы не можете.

6 Приложение А

https://github.com/dmitrysavitski/Shoes-store

Заключение

Целью курсового проекта являлась разработка интернет-магазина по продаже обуви. В ходе выполнения проекта было изучено большое количество технологий, таких как ASP.NET MVC, Bootstrap, шаблон проектирования MVC, базы данных и другие. При работе над курсовой работой я изучил большое количество материалов по теме для грамотного и качественного решения задач, поставленных вначале разработки. Убедившись в корректности работы программы, можно сказать, что результат курсового проекта «Интернет-магазин по продаже обуви» полностью соответствует всем требованиям, поставленным вначале разработки программы.

Список использованных источников

1. [url] **Введение в C#** https://metanit.com/sharp/
2. [url] **Интернет магазин на ASP.Net** https://professorweb.ru/my/ASP_NET/gamestore/level1/
3. [url] **Git. Википедия – свободная энциклопедия** https://ru.wikipedia.org/wiki/Git
4. [url] **Model-View-Controller** https://ru.wikipedia.org/wiki/Model-View-Controller
5. [url] **Основы паттернов проектирования** https://metanit.com/sharp/patterns/1.1.php
6. [печатное издание] **Cochran, D. Bootstrap Site Blueprints / D. Cochran, I. Whitley. – Packt, 2014. – 304 p.**

Приложения

1. [электронный документ] **записка** 5ebce8b59971f_Savitski_OOP.docx
2. [Задание] **Задание** 5ebce8eaba892_задание.docx
3. [электронный документ] 5ebcf780e3912_Диаграмма последовательности и состояния.docx