

Специальность: «Программист Python»

ДИПЛОМНЫЙ ПРОЕКТ НА ТЕМУ:

**«Разработка современного интернет-магазина**

**для упрощения и ускорения технологический процессов.»**

Автор - Белик Сергей Николаевич

Саратов – 2023

Дипломный проект «Разработка современного интернет-магазина

для упрощения и ускорения технологический процессов.»

Содержание:

|  |  |
| --- | --- |
| Введение | 2 |
| Теоретическая часть | 30 |
| Практическая часть | 14 |
| Заключение | 1 |
| Список литературы и ресурсов | 1 |
| Презентация | 5 |
|  |  |

1. **Введение:**

Концепция проекта заключается в создании современного и функционального интернет-магазина, предоставляющего уникальную платформу для продажи разнообразных строительных материалов, включая трубы и другие важные строительные изделия. Моей задачей при разработке проекта является создание привлекательного и интуитивно понятного веб-приложения, которое удовлетворит потребности как корпоративных клиентов, так и индивидуальных заказчиков.

Разработка интернет-магазина осуществлена с целью - обеспечить максимальное удобство и удовлетворение потребностей наших пользователей. Это платформа, предоставляющая уникальный и привлекательный пользовательский опыт, позволяющий клиентам находить необходимые строительные материалы быстро, легко и с удовольствием.

Важным аспектом разработки является выбор подходящей технологической платформы. Для этого и используются мощности и гибкости языка программирования Python в сочетании с прогрессивным фреймворком Django. Это обеспечит быструю и отзывчивую работу интернет-магазина, а также облегчит интеграцию сторонних сервисов, таких как системы оплаты и доставки.

Одним из главных аспектов, на которые нужно было обратить особое внимание - информативность интернет-магазина. Широкий спектр описания продуктов, детальные характеристики и фотографии(клиенты должны получить всю необходимую информацию о товаре до совершения покупки, но и добавить отзывы и рейтинг продуктов) должны создать доверие к качеству и надежности товаров.

Для достижения максимального удобства пользователей старался уделить особое внимание оптимизации пользовательского интерфейса и навигации. Создать интуитивно понятный дизайн и структурировать информацию таким образом, чтобы пользователи могли легко ориентироваться на сайте и совершать покупки без лишних усилий. Кроме того, проводил оптимизацию загрузки страниц и обеспечение отзывчивости сайта на различных устройствах, что позволит пользователям наслаждаться бесперебойным взаимодействием с интернет-магазином, где бы они ни находились.

На этапе планирования и анализа изучал рынок строительных материалов, проводил анализ потребительского спроса и конкурентного окружения. Это помогло определить наиболее востребованные товарные категории и ключевые функциональные возможности интернет-магазина. В результате создано уникальное и привлекательное предложение для клиентов.

Проектирование интернет-магазина сосредоточилось на разработке привлекательного и современного дизайна, который обеспечит удобство и понятность пользовательского интерфейса. Уделено особое внимание эстетике, чтобы создать уникальную визуальную идентичность для магазина, позволяющую выделиться и создать положительное впечатление у клиентов.

Уделено внимание безопасности и защите данных. Применены лучшие практики по обеспечению конфиденциальности информации клиентов и обеспечению безопасность платежных транзакций, что повысит доверие к магазину.

Важным этапом в процессе разработки является тщательное тестирование функциональности, производительности и безопасности интернет-магазина. Систематические проверки на различных устройствах и браузерах, гарантирующие отличный пользовательский опыт для всех клиентов, проводятся постоянно. Это основной фактор безопасности и обеспечения его стабильной и надежной работы, что очень важно для заключительного этапа - внедрение интернет-магазина и его запуск на хостинге. Обеспечение его стабильной и надежной работы.

Моя цель - создать выдающийся интернет-магазин для строительных материалов, который привлечет множество клиентов и сделает их покупки удобными и приятными.

**2. Теоретическая часть**

Python - высокоуровневый интерпретируемый язык программирования с простым и понятным синтаксисом, который делает его очень популярным среди начинающих и опытных разработчиков. Был создан Гвидо ван Россумом в конце 1980-х годов. Он начал работать над языком программирования в декабре 1989 года, когда работал в канцелярии по математике и компьютерным наукам (CWI) в Нидерландах.

Гвидо ван Россум хотел создать новый язык программирования, который был бы простым, читаемым и имел четкую структуру. Он назвал свой проект в честь популярного комедийного шоу "Монти Пайтон" (Monty Python), поэтому язык получил своё название "Python".

Первая публичная версия Python, версия 0.9.0, была выпущена в феврале 1991 года. В этот период основные особенности Python и были заложены, включая простой и читаемый синтаксис.

Python 2 (2000-2010): В 2000 году вышла версия Python 2.0, в которой были добавлены новые функции, такие как list comprehensions и garbage collection. Версии 2.x были активно разрабатывались и использовались в течение следующего десятилетия.

Python 3 (2008-настоящее время): Разработка Python 3.x началась в 2006 году и была ориентирована на улучшение языка, избавление от устаревших конструкций и улучшение Unicode-поддержки. Python 3.0 был выпущен в декабре 2008 года, но переход от Python 2.x занял некоторое время из-за несовместимости. Но постепенно сообщество перешло на Python 3.x.

Рост популярности и экосистемы: в последние годы Python стал одним из самых популярных языков программирования благодаря своей простоте, многофункциональности и поддержке различных платформ. Python также стал широко используемым в различных областях, таких как веб-разработка, научные вычисления, искусственный интеллект, анализ данных и многие другие.

Некоторые ключевые особенности Python:

Читабельность кода Python - одна из ключевых особенностей языка, подчеркивающая его философию "читаемости кода". Это означает, что код на Python должен быть написан таким образом, чтобы он был понятен и легко читаем другими разработчиками (включая вас самого), что облегчает совместную работу и поддержку проекта. Вот некоторые аспекты, которые способствуют читабельности кода Python:

Отступы - Python использует отступы вместо фигурных скобок или ключевых слов для определения блоков кода. Правильное форматирование отступов делает структуру кода очевидной и легко читаемой.

Именование переменных - имена переменных в Python должны быть описательными и понятными. Используйте смысловое название для переменных, которые отражают их содержание или назначение.

Понятные комментарии - добавление комментариев к коду помогает объяснить его логику и дает контекст для понимания. Хорошие комментарии объясняют сложные части кода и действия, которые он выполняет.

Простой и понятный синтаксис - Python предоставляет чистый и простой синтаксис, что делает код более понятным. Избегайте излишней сложности и используйте языковые конструкции, которые делают код более выразительным и лаконичным.

Длина строк - постарайтесь не делать строки кода слишком длинными. Рекомендуется придерживаться PEP 8 (Python Enhancement Proposal 8), который предлагает максимальную длину строки в 79 символов.

Использование функций и модулей - разделение кода на функции и модули помогает сделать его более структурированным и облегчает его переиспользование.

Избегайте избыточности - не повторяйте код без необходимости.

Динамическая типизация - это свойство языка программирования, которое позволяет переменным принимать значения различных типов данных во время выполнения программы. Python является языком с динамической типизацией, что делает его гибким и удобным для разработки.

Давайте подробнее рассмотрим основные аспекты динамической типизации в Python:

Без объявления типов: в отличие от некоторых языков программирования, где нужно явно указывать тип переменной при её объявлении (например, int, float, string и т. д.), в Python тип переменной определяется автоматически во время присваивания значения.

Изменение типов: переменные в Python могут менять свой тип в процессе выполнения программы. Например, переменная, которая сначала содержала числовое значение, может позже принять строковое значение без необходимости предварительного объявления новой переменной.

Динамический вывод типов: Python динамически определяет тип переменных, и вы можете узнать тип переменной с помощью функции type(). Например, type(5) вернет <class 'int'>, а type("hello") вернет <class 'str'>.

Гибкость и удобство: динамическая типизация делает Python гибким и удобным для быстрой разработки, так как вы можете изменять типы данных переменных в зависимости от требований вашей программы без ограничений.

Однако, следует помнить, что с динамической типизацией приходит и ответственность за правильное использование переменных и их типов. Неправильное использование может привести к ошибкам во время выполнения программы. Хорошее знание типов данных и общих правил Python поможет вам избегать таких проблем.

Обширная библиотека: Python имеет обширную стандартную библиотеку, которая включает в себя большое количество модулей и функций, предоставляющих различные инструменты для разработки. Эти модули покрывают широкий спектр областей, таких как работа с файлами, сетевое программирование, работа с базами данных, обработка строк, математические вычисления, веб-разработка, научные вычисления и многое другое. Вот некоторые из наиболее популярных модулей из стандартной библиотеки Python:

os: Модуль для взаимодействия с операционной системой, предоставляет функции для работы с файловой системой, директориями, путями и т. д.

sys: Модуль для работы с интерпретатором Python, предоставляет доступ к аргументам командной строки, стандартным потокам ввода/вывода и другой информации.

re: Модуль для работы с регулярными выражениями позволяет выполнять поиск и обработку текста на основе шаблонов.

datetime: Модуль для работы с датами и временем предоставляет классы для работы с датами, временем, интервалами и т. д.

urllib: Модуль для работы с URL-адресами и сетевым программированием позволяет отправлять HTTP-запросы, работать с веб-страницами и т. д.

json: Модуль для работы с JSON (JavaScript Object Notation) позволяет сериализовывать и десериализовывать данные в формате JSON.

math: Модуль для выполнения математических операций предоставляет функции для работы с числами, константами и т. д.

random: Модуль для работы с случайными числами и выборками позволяет генерировать случайные числа и выбирать случайные элементы из последовательности.

sqlite3: Модуль для работы с базами данных SQLite предоставляет функции для создания, подключения и выполнения SQL-запросов.

http.server: Модуль для простого создания HTTP-сервера используется для разработки простых веб-приложений и тестирования API.

Это лишь небольшой обзор функциональности стандартной библиотеки Python. Благодаря ей разработчики могут избегать необходимости создавать функциональность "с нуля" и использовать готовые инструменты для решения различных задач. Кроме того, Python также имеет множество сторонних библиотек, которые значительно расширяют возможности языка в различных областях разработки.

Поддержка различных стилей программирования: Python обладает гибкостью и позволяет разработчикам использовать различные стили программирования в зависимости от их предпочтений и требований проекта. Ниже я расскажу о нескольких распространенных стилях программирования, которые часто используются в сообществе Python:

Процедурное программирование: это стиль программирования, в котором программа разделяется на набор процедур (функций), каждая из которых выполняет определенную задачу. В Python можно легко писать процедурный код, определяя функции, которые выполняют различные операции.

Объектно-ориентированное программирование (ООП): это парадигма программирования, которая позволяет организовать код в виде объектов, взаимодействующих между собой. ООП предоставляет способ моделирования реального мира, а также упрощает разработку сложных программных систем.

Основные принципы ООП:

Инкапсуляция: это принцип, согласно которому данные (переменные) и методы (функции) для их обработки объединяются в единый объект (класс). Таким образом, данные скрыты от прямого доступа извне, и для работы с ними используются методы класса. Это обеспечивает контроль доступа и защиту данных, что упрощает поддержку кода и уменьшает его зависимость от конкретных реализаций.

Наследование - позволяет создавать новый класс (подкласс) на основе уже существующего (родительского класса). Подкласс наследует атрибуты и методы родительского класса и может расширять или изменять их поведение. Это позволяет повторно использовать код и создавать иерархии классов для упрощения кода и его организации.

Полиморфизм - предполагает, что один интерфейс может быть реализован разными способами. Это означает, что объекты разных классов могут иметь одинаковые методы, но выполнять различные действия в зависимости от своей конкретной реализации. Полиморфизм позволяет обобщать операции на различных типах данных, что упрощает и облегчает код.

Python предоставляет множество средств для реализации ООП, включая объявление классов с помощью ключевого слова class, использование наследования с помощью super(), а также поддержку инкапсуляции через соглашения о наименованиях (public, protected, private) и декораторы (например, @property).

Python также поддерживает функциональное программирование. Функции считаются "первоклассными объектами", что позволяет передавать их как аргументы другим функциям, возвращать из функций и сохранять в переменных. Этот стиль программирования обращается к функциям как к основным строительным блокам программы.

Императивное программирование - этот стиль программирования описывает последовательность инструкций, которые изменяют состояние программы. Python позволяет писать императивный код, используя операторы присваивания, циклы и условные операторы.

Функциональное реактивное программирование (FRP) - этот стиль программирования акцентирует внимание на потоках данных и реакции на изменения. В Python существуют сторонние библиотеки, позволяющие писать FRP-код.

Использование подходов структурного программирования: Python также поддерживает структурное программирование, которое ограничивает использование переходов и операторов goto, что облегчает чтение и понимание кода.

Важно отметить, что в большинстве проектов Python разработчики часто комбинируют несколько стилей программирования в зависимости от потребностей проекта и личных предпочтений. Правильный выбор стиля программирования может значительно повысить понятность, структурированность и эффективность вашего кода.

Кроссплатформенность: Python работает на различных операционных системах, таких как Windows, macOS, Linux и других.

Широкое применение: Python применяется в различных областях, включая веб-разработку, научные вычисления, искусственный интеллект, анализ данных и автоматизацию задач.

Интерпретируемый язык: Код Python выполняется интерпретатором в режиме реального времени, что упрощает разработку и отладку.

Помимо стандартной библиотеки существует также множество сторонних библиотек и фреймворков, которые расширяют функциональность Python для специфических задач.

Python продолжает развиваться и улучшаться благодаря активному сообществу разработчиков, которое поддерживает язык и создает новые библиотеки и фреймворки. Эта поддержка делает Python идеальным выбором для решения разнообразных задач.

Важной особенностью Python является его удобство в изучении и использовании. Начинающим программистам часто легче понять основные концепции программирования, используя Python, что обусловлено его простотой и понятным синтаксисом.

Python также привлекателен для опытных разработчиков, так как позволяет создавать сложные программы с помощью компактного кода. Он способствует увеличению производительности разработчиков, так как позволяет достигать результатов с меньшим объемом кода, чем многие другие языки.

Еще одним преимуществом Python является его обширное сообщество пользователей и разработчиков. В мире Python существует огромное количество ресурсов для обучения, форумов для обсуждения и поддержки, а также открытых проектов, на которых можно работать. Это позволяет разработчикам получать помощь, обмениваться опытом и находить новые возможности для роста.

Python также часто выбирается в качестве языка для преподавания основ программирования в школах и университетах. Его популярность в образовательных учреждениях способствует распространению знаний о программировании и привлекает новых людей в IT-сферу.

Следует также отметить, что Python обладает активной поддержкой в области научных вычислений и анализа данных.

Библиотеки для научных вычислений и анализа данных: NumPy предоставляет мощные инструменты для работы с многомерными массивами и математическими функциями. SciPy расширяет его функциональность для научных и инженерных приложений. Pandas предоставляет структуры данных и функции для анализа и манипулирования табличными данными.

Визуализация данных: Matplotlib позволяет создавать разнообразные графики и диаграммы для визуализации данных. Seaborn и Plotly предоставляют более высокоуровневые интерфейсы для создания стильных графиков.

Статистический анализ: StatsModels предоставляет статистические модели и тесты, необходимые для анализа данных. Это важно для извлечения смысла из данных и проверки гипотез.

Интерактивные вычисления и визуализации: Jupyter Notebook предоставляет интерактивное окружение для кодирования, выполняя код по частям и отображая результаты встроенных графиков и визуализаций.

Python также активно применяется в области искусственного интеллекта и машинного обучения. Библиотеки TensorFlow, PyTorch, scikit-learn, Keras, NLTK, SpaCy и другие становятся ключевыми инструментами для исследования и применения методов машинного обучения в различных областях, включая компьютерное зрение, обработку естественного языка и автономные технологии.

TensorFlow - открытая библиотека машинного обучения и глубокого обучения, разработанная компанией Google. Она предоставляет инструменты для создания и обучения различных типов нейронных сетей, моделей машинного обучения и искусственного интеллекта. Вот некоторые ключевые аспекты TensorFlow:

Граф вычислений: Одной из особенностей TensorFlow является использование графа вычислений для представления операций и данных. Граф описывает поток данных и операций, что позволяет эффективно выполнять вычисления на графических процессорах (GPU) и центральных процессорах (CPU).

Гибкость и масштабируемость: TensorFlow предоставляет гибкость в выборе архитектур нейронных сетей и моделей машинного обучения. Он поддерживает как простые нейронные сети, так и сложные архитектуры, такие как сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN) и генеративные состязательные сети (GAN).

Высокоуровневые API: TensorFlow предоставляет высокоуровневые API для создания и обучения моделей более простым способом. Например, Keras, который является частью TensorFlow, обеспечивает удобный интерфейс для создания нейронных сетей без необходимости написания большого количества кода.

Распределенное обучение: TensorFlow позволяет обучать модели на множестве устройств, включая несколько GPU и даже кластеры серверов. Это обеспечивает возможность распределенного обучения для ускорения процесса обучения на больших наборах данных.

Инструменты для визуализации: TensorFlow предоставляет инструменты для визуализации графов вычислений, обучения моделей и мониторинга процесса. Это помогает разработчикам лучше понимать, как проходит обучение моделей и выявлять возможные проблемы.

Поддержка различных платформ: TensorFlow может быть использован на различных платформах, включая серверы, персональные компьютеры, мобильные устройства и веб-приложения.

TensorFlow оказал огромное влияние на область глубокого обучения и искусственного интеллекта, благодаря своей гибкости, масштабируемости и обширной документации. Он широко используется как исследователями, так и инженерами для решения разнообразных задач в области анализа данных и создания интеллектуальных систем.

PyTorch разработана и поддерживается Facebook. Она предоставляет инструменты для создания и обучения нейронных сетей, а также обработки и анализа данных. Вот некоторые ключевые характеристики PyTorch:

Динамические вычисления. Одной из отличительных особенностей PyTorch является его динамическая вычислительная графика. В отличие от TensorFlow, где вычисления описываются статическим графом, в PyTorch граф формируется динамически по мере выполнения операций. Это упрощает процесс отладки и разработки моделей.

Интуитивный интерфейс. PyTorch обладает простым и интуитивным интерфейсом, который делает его очень доступным для новичков в области глубокого обучения. Он позволяет быстро создавать и настраивать модели без необходимости писать большое количество кода.

Автоматическое дифференцирование. PyTorch автоматически вычисляет градиенты для параметров моделей, что делает процесс обучения гораздо проще. Это особенно полезно при работе с методами оптимизации и обучения градиентным спуском.

Модульность и гибкость. Библиотека построена на основе модульной структуры, позволяющей разработчикам комбинировать различные слои и операции для создания сложных моделей. Это дает большую гибкость в настройке архитектуры нейронных сетей.

Совместимость с NumPy. PyTorch совместим с библиотекой NumPy, что облегчает процесс обработки и анализа данных перед использованием их в моделях PyTorch.

Работа с графиками вычислений. В PyTorch также есть функциональности для создания и визуализации графиков вычислений, что помогает лучше понимать поток данных и операций в моделях.

Scikit-learn. Библиотека машинного обучения, разработанная для общих задач анализа данных и построения моделей. Scikit-learn предоставляет широкий спектр алгоритмов для классификации, регрессии, кластеризации, визуализации данных и многого другого. Она удобна в использовании и хорошо подходит для начинающих в области машинного обучения.

Keras - высокоуровневый API для построения нейронных сетей, работающий поверх библиотек TensorFlow и Theano (поддержка Theano официально завершена). Keras упрощает создание, обучение и оценку моделей глубокого обучения. Он подходит как для новичков, так и для опытных исследователей в области нейронных сетей.

NLTK (Natural Language Toolkit). Библиотека для обработки естественного языка, предоставляющая инструменты для анализа и обработки текстовых данных. NLTK содержит различные ресурсы, такие как корпусы текстов, лексиконы и алгоритмы для анализа текста, токенизации, стемминга и морфологического анализа.

SpaCy. Библиотека для обработки естественного языка, которая предоставляет эффективные инструменты для токенизации, лемматизации, извлечения именованных сущностей, анализа зависимостей и многое другое. SpaCy спроектирована для обработки текстов большими объемами данных с быстрой скоростью.

Каждая из этих библиотек предоставляет инструменты для конкретных задач в области анализа данных и обработки естественного языка. Их гибкость и функциональность делают их незаменимыми инструментами для разработчиков и исследователей, работающих в этих областях.

Еще одним преимуществом Python является его гибкость в интеграции с другими языками программирования.

Многоязыковая платформа. Python является платформой с открытым исходным кодом, что облегчает интеграцию с различными языками программирования.

Foreign Function Interface (FFI). Python предоставляет механизм FFI, позволяющий вызывать функции, написанные на других языках (например, C, C++, Fortran), из Python-кода.

Работа с библиотеками. Множество языков предоставляют богатый выбор библиотек и инструментов. Python может использовать эти библиотеки, расширяя свои возможности.

Биндинги и обёртки. Возможность создания биндингов (привязок) и обёрток позволяет использовать существующий код на других языках в Python, делая его доступным для Python-разработчиков.

Embedding. Python может быть встроен в приложения, написанные на других языках, для добавления сценариев и возможностей расширения.

Модули на C. Расширения на C могут быть подключены к Python как модули, что улучшает производительность и предоставляет доступ к системным вызовам.

Jupyter Notebook. Интеграция Python с другими языками в Jupyter Notebook позволяет создавать интерактивные документы, объединяя различные языки и выводы в единый поток.

Web-интеграция. Python может быть интегрирован с другими языками для создания веб-приложений с разнообразными компонентами.

Гибкий формат данных. Python поддерживает работу с различными форматами данных (JSON, XML, CSV), что упрощает обмен информацией между различными языками.

Python занимает высокие позиции в различных рейтингах языков программирования. Вот некоторые ключевые аспекты рейтинга:

TIOBE Index. Рейтинг TIOBE Index оценивает популярность языков программирования на основе поисковых запросов и статистики.

PYPL Popularity of Programming Language. Этот рейтинг учитывает интерес разработчиков к языкам программирования на основе поисковых запросов в обучающих материалах.

RedMonk Ranking. Этот рейтинг объединяет данные с GitHub и Stack Overflow для определения популярности языков программирования среди разработчиков.

IEEE Spectrum. В этом рейтинге учитываются различные факторы, включая популярность в поисковых запросах, разнообразие применений и прочие.

Еще одной важной характеристикой Python является его многоплатформенность. Python код может быть написан один раз и запущен на различных операционных системах, таких как Windows, macOS, Linux и даже более экзотические платформы, такие как BSD и AIX, без необходимости внесения значительных изменений.

Кросс-платформенные библиотеки Python имеют множество библиотек и фреймворков, которые являются кросс-платформенными, то есть они работают на разных ОС без изменений в коде. Примеры включают библиотеку GUI Tkinter, фреймворк для веб-разработки Django и библиотеку для научных вычислений NumPy.

Официальные интерпретаторы Python (CPython) и множество других альтернативных интерпретаторов (например, Jython, IronPython) доступны для разных платформ. Это позволяет использовать Python на разных устройствах и встроенных системах.

Python позволят создавать виртуальные среды, которые изолируют зависимости и библиотеки для каждого проекта. Это упрощает переносимость кода между разными платформами.

Большинство библиотек и модулей Python также являются многоплатформенными, что позволяет использовать один и тот же код на разных ОС.

Python поддерживает совместимость между версиями 2.x и 3.x, что облегчает перенос кода между разными версиями языка.

Python также известен разработчикам своим активным и дружественным сообществом. Python - разработчики часто обмениваются опытом, обсуждают идеи и предлагают решения через различные форумы, конференции и онлайн-сообщества. Это содействует развитию языка и распространению знаний в сообществе.

Python широко применяется в веб-разработке благодаря своей удобной синтаксису, богатому набору библиотек и фреймворков, а также разнообразным возможностям интеграции. Основные направления применения Python в веб-разработке включают:

Фреймворки для веб-разработки. Python предоставляет несколько мощных фреймворков, которые упрощают создание веб-приложений. Django, Flask, и FastAPI - это некоторые из наиболее популярных фреймворков. Django предоставляет полный стек инструментов для сложных проектов, Flask - более гибкий и позволяет выбирать компоненты, а FastAPI - быстрый фреймворк для создания API.

Разработка веб-приложений. С помощью фреймворков можно легко создавать полноценные веб-приложения с функциями, такими как аутентификация, авторизация, административные панели и многое другое. Фреймворки предоставляют структуру для обработки запросов, управления маршрутами и отображения данных.

Взаимодействие с базой данных. Python предоставляет богатый выбор библиотек для работы с базами данных. Фреймворки также обеспечивают интеграцию с базой данных через ORM (Object-Relational Mapping), что упрощает взаимодействие с данными и уменьшает необходимость написания SQL-запросов вручную.

Работа с шаблонами. Чтобы создавать динамические веб-страницы, Python предоставляет возможность использовать шаблоны. С его помощью можно встраивать данные из вашего кода в HTML-шаблоны для отображения на веб-страницах.

Обработка форм. Python облегчает обработку данных, отправляемых пользователем через веб-формы. Фреймворки предоставляют инструменты для валидации данных, обработки ошибок и сохранения информации.

Создание API и микросервисов. С помощью фреймворков, таких как Flask и FastAPI, вы можете создавать API для обмена данными между разными приложениями. Это особенно актуально в современной архитектуре микросервисов.

Тестирование. Python облегчает написание тестов для кода. Используется при создавать юнит-тестов, интеграционных тестов и функциональных тестов для обеспечения качества приложения.

Веб-скрапинг и автоматизация. Python может быть использован для создания скриптов, которые извлекают данные с веб-сайтов (веб-скрапинг) и автоматизируют повторяющиеся задачи и автоматизации задач, связанных с вебом.

Веб-сайты и блоги. Python часто используется для разработки корпоративных веб-сайтов, персональных блогов, новостных порталов и статических сайтов.

Python широко используется для работы с данными из социальных сетей. С помощью библиотеки Tweepy можно взаимодействовать с Twitter API для сбора данных, например, твитов, пользователей или трендов. Для работы с Facebook API можно использовать библиотеку PyFacebook. Instagram Graph API позволяет получать данные с Instagram.

Для анализа текстовых данных из социальных сетей, таких как комментарии или посты, можно использовать библиотеки для обработки текста, например, NLTK или spaCy. Для анализа настроений и эмоций часто используют библиотеку TextBlob.

При визуализации данных из социальных сетей библиотеки Matplotlib и Seaborn помогут создать графики и диаграммы. Для более интерактивных визуализаций можно воспользоваться Plotly.

Если вы хотите создать бота для социальных сетей, то библиотека python-telegram-bot для Telegram поможет вам.

Боты для Telegram на Python стали популярными благодаря библиотеке python-telegram-bot. Вот краткий обзор истории, развития и возможностей библиотеки. Библиотека python-telegram-bot была создана в 2015 году. Она предоставляет разработчикам удобные инструменты для создания ботов на платформе Telegram, облегчая процесс взаимодействия с Telegram API.

С течением времени библиотека стала все более популярной благодаря своей простоте использования и обширной функциональности. Разработчики активно поддерживали библиотеку, выпуская регулярные обновления с целью включения новых возможностей и исправления ошибок.

Библиотека предоставляет широкий спектр функций, включая:

Отправку и получение текстовых сообщений, изображений, аудио и видео файлов.

Создание клавиатур для взаимодействия с пользователем.

Обработка команд и обычных текстовых сообщений.

Работа с обновлениями и событиями бота.

Интеграция с вебхуками для более эффективного получения обновлений.

Поддержка асинхронных операций для улучшения производительности.

Возможность обработки встроенных запросов и многое другое.

Библиотека стала популярной среди разработчиков благодаря своей простоте и обширной документации. Она получила значительное количество звезд на GitHub и активное сообщество пользователей и разработчиков. Python-telegram-bot стал одной из наиболее используемых библиотек для создания ботов на платформе Telegram.

Python и его история в интернет-магазинах началась в конце 1980-х. Однако активное развитие инструментов и библиотек для веб-разработки на Python началось позже, в 2000-х годах. В этот период появились такие фреймворки, как Django и Flask, которые значительно упростили создание веб-приложений, включая интернет-магазины.

Python привлекателен для интернет-магазинов из-за своей простоты и читаемости кода, что способствует быстрой разработке и сопровождению проектов. Богатство библиотек позволяет легко интегрировать разнообразные функциональные возможности.

Распространение Python в интернет-магазинах быстро росло благодаря развитию экосистемы инструментов и успешным кейсам использования. Этот язык стал популярным выбором для создания MVP и прототипов, а также для проектов с ограниченными бюджетами.

Важные факторы, способствующие его использованию, включают отказоустойчивость, так как Python обладает множеством библиотек и фреймворков для обработки ошибок и управления исключениями. Благодаря активному сообществу разработчиков, поддержка и обновления инструментов доступны на протяжении времени.

Python также предоставляет возможности для сравнения с другими языками. Например, в сравнении с PHP, Python имеет более структурированный и организованный код. По сравнению с Java, Python более компактен и требует меньше строк кода для реализации функциональности.

Обработка заказов и уведомления. Python может автоматически обрабатывать входящие заказы, обновлять базу данных о товарах и клиентах, а также отправлять уведомления о статусе заказа по электронной почте или через SMS. Это позволяет оперативно информировать покупателей о ходе доставки.

Управление инвентарем. Скрипты на Python могут отслеживать количество товаров в наличии и автоматически обновлять информацию о доступных количествах на сайте. Это предотвращает ситуации, когда товар заканчивается, а он всё ещё доступен для заказа.

Генерация отчетов. Python может автоматически создавать разнообразные отчеты, включая статистику продаж, тренды спроса, популярные товары и многое другое. Это помогает принимать информированные решения по развитию бизнеса.

Поддержка клиентов через чат-ботов. Python позволяет разрабатывать чат-ботов, которые могут отвечать на часто задаваемые вопросы, помогать с выбором товаров и даже оформлять заказы прямо в чате.

Оптимизация цен. Скрипты на Python могут отслеживать цены у конкурентов и на основе этой информации автоматически корректировать цены на товары в вашем магазине для поддержания конкурентоспособности.

Персонализированные рекомендации. Python используется для анализа поведения покупателей и предоставления персонализированных рекомендаций о товарах, которые могут заинтересовать клиента.

Скорость и результаты автоматизации зависят от сложности задачи и эффективности написанных скриптов. Преимущества автоматизации включают более быстрое и точное выполнение задач, снижение риска человеческих ошибок, повышение эффективности работы, а также более оперативное реагирование на изменения в бизнес-процессах. Однако важно также учесть, что правильная настройка и тестирование автоматизированных решений требует времени и усилий.

Игры на Python – это популярная область, которая привлекает разработчиков своей доступностью и множеством инструментов.

1. Инструменты:

- Pygame: Библиотека для разработки 2D игр. Предоставляет функции для работы с графикой, звуком и вводом.

- Panda3D: Движок для создания 3D игр. Имеет готовые компоненты для работы с графикой и физикой.

- Godot Engine: не связан напрямую с Python, но поддерживает его в качестве скриптового языка.

2. Основные этапы разработки:

- Проектирование: определение механик, структуры и визуального стиля игры.

- Реализация: написание кода для игровых объектов, управления и логики игры.

- Тестирование: проверка игры на наличие ошибок, уточнение баланса и играбельности.

- Оптимизация: улучшение производительности и оптимизация кода.

- Релиз: подготовка игры к публикации, создание дистрибутивов.

3. Особенности:

- Простой синтаксис: Python обладает читаемым и понятным кодом, что упрощает разработку и поддержку.

- Богатая экосистема: множество библиотек и инструментов для графики, звука и физики.

- Сообщество: активное сообщество разработчиков, готовое помочь и поделиться опытом.

Игровая разработка на Python предоставляет возможность создания разнообразных игр, начиная с небольших проектов и заканчивая более сложными и амбициозными.

Python становится все более популярным инструментом для разработки мобильных приложений благодаря своей простоте, гибкости и богатой экосистеме.

Мобильная разработка на Python:

Kivy - фреймворк Python для создания мультимедийных приложений, включая те, которые работают на Android. Он позволяет создавать кроссплатформенные приложения с интерфейсом, адаптированным под различные устройства.

BeeWare - набор инструментов, предоставляющих возможность разрабатывать приложения на Python и запускать их на разных платформах, включая Android.

PyQt for Android - набор инструментов для создания графического интерфейса на Python. С его помощью также можно разрабатывать приложения для Android, но процесс может быть сложным.

SL4A (Scripting Layer for Android) - проект, позволяющий запускать скрипты Python на Android-устройствах. Однако он предназначен в большей степени для автоматизации и скриптовых задач.

QPython - среда разработки на Python для Android, которая также предоставляет доступ к некоторым системным функциям.

Chaquopy - плагин для Android Studio, который позволяет встраивать Python-код в приложение Android, обеспечивая легкую интеграцию.

Python предоставляет удобные средства для мобильной разработки, позволяя разработчикам быстро создавать качественные и функциональные приложения для разных платформ.

Благодаря всем этим преимуществам, Python стал выбором многих крупных компаний и стартапов для разработки программного обеспечения. Он используется в таких компаниях, как Google, Facebook, Dropbox, Instagram и многих других. Вот несколько исторических фактов:

Google начал активно использовать Python в начале 2000-х годов. Внутренние инструменты и проекты Google часто были написаны на Python из-за его простоты, читаемости и эффективности. Один из первых больших проектов, написанных на Python, - фреймворк для создания веб-приложений Google, известный как "Google Web Framework" (GWF).

Google App Engine, платформа для разработки и развертывания веб-приложений, также предоставляет поддержку для приложений, написанных на Python, начиная с ее запуска в 2008 году. Это позволило разработчикам создавать и размещать веб-приложения на Python в облачной инфраструктуре Google.

Guido van Rossum в Google: в 2005 году Гвидо ван Россум, создатель языка Python, присоединился к Google и работал над различными проектами, связанными с Python. Он оставался в Google до 2012 года, после чего вернулся в сообщество Python.

TensorFlow, библиотека для машинного обучения и глубокого обучения, была разработана командой Google Brain. Хотя она в основном написана на C++ и CUDA, TensorFlow также предоставляет Python API для более удобного использования. Это сделало Python популярным языком для разработки моделей машинного обучения с использованием TensorFlow.

Google Colab - интерактивная среда для выполнения кода на Python в браузере, которая предоставляет доступ к вычислительным ресурсам Google. Она позволяет научным исследователям и разработчикам создавать и выполнять код на Python для анализа данных и обучения моделей машинного обучения.

Связь между языком программирования Python и компанией Facebook (теперь известной как Meta) является важной и интересной.

Движок Django. Django - популярный фреймворк для разработки веб-приложений на Python. Он был разработан внутри компании Lawrence Journal-World, которая впоследствии была приобретена Facebook. Django успешно использовался во многих проектах и продуктах компании, включая Instagram.

Instagram. Одним из самых известных проектов, основанных на Python, является Instagram. Instagram начал свою историю как небольшое веб-приложение на Django, созданное Кевином Систем и Майком Кригером. Позже оно было приобретено Facebook (Meta), и с тех пор Instagram стал одним из самых популярных социальных медиа-платформ в мире.

PyTorch - фреймворк для глубокого обучения и искусственного интеллекта, разработанный Facebook AI Research (FAIR). Он стал популярным среди исследователей и разработчиков благодаря своей гибкости и легкости использования.

Важно также упомянуть о недостатках Python. Одним из них является его относительно низкая производительность по сравнению с некоторыми компилируемыми языками, такими как C++. Это может быть проблемой для высоконагруженных приложений или при работе с большими объемами данных. Тем не менее, существуют способы оптимизации кода на Python и интеграции с быстрыми библиотеками на других языках для решения этой проблемы.

Большой вклад в популярность Python внесли Flask и Django.

Flask является более легковесным и гибким фреймворком, который позволяет разработчикам выбирать только необходимые компоненты для своего проекта.

Django фреймворк для веб-разработки, обладающий всеми необходимыми инструментами для быстрой и удобной работы.

Согласно историческим фактам Django был создан в 2003 году журналистами, разработчиками из компании "Lawrence Journal-World". Условия требовали, чтобы новые возможности и целые приложения были разработаны максимально быстро, часто счёт шёл на дни и часы. В результате чего Саймон и Адриан создали среду разработки, экономящую время, исходя из своих потребностей - это был единственный способ, с помощью которого они могли создавать управляемые приложения в жёстких рамках сроков.

В июле 2005 года Django, названный в честь джазового гитариста Джанго Рэйнхарда (Django Reinhardt), была выпущена в виде программного обеспечения с открытым исходным кодом.

Основная цель Django - ускорение процесса разработки веб-приложений. Он предоставляет множество готовых компонентов и инструментов для решения типовых задач, таких как работа с базами данных, обработка форм и маршрутизация URL-ов. Это позволяет разработчикам сосредотачиваться на бизнес-логике приложения, минимизируя необходимость писать множество повторяющегося кода.

Преимущества и особенности Django.

Простота. Джанго обладает интуитивно понятным синтаксисом, что упрощает создание приложений даже для новичков.

ORM (Object-Relational Mapping). Встроенный ORM позволяет работать с базами данных через объекты Python, упрощая взаимодействие с данными.

Автоматическая административная панель. Джанго предоставляет готовый функционал для создания админ-панели, что позволяет управлять данными и приложением без необходимости писать дополнительный код.

Масштабируемость. Фреймворк поддерживает расширение функционала через плагины и приложения, что упрощает создание сложных проектов.

Безопасность. Джанго предоставляет множество инструментов для защиты приложения от уязвимостей, таких как CSRF, SQL-инъекции и другие.

URL-маршрутизация. Гибкая система маршрутизации позволяет определять, какие функции будут вызываться при обращении к определенным URL.

Шаблоны. Встроенная система шаблонов упрощает создание визуального оформления веб-страниц.

Сообщество. Джанго имеет большое и активное сообщество разработчиков, что обеспечивает доступ к множеству ресурсов и документации.

Готовые решения. благодаря стандартным компонентам и функционалу, Джанго позволяет быстро создавать приложения без необходимости писать много кода "с нуля".

REST API. Фреймворк предоставляет средства для создания RESTful API, что позволяет строить современные веб-приложения с клиент-серверной архитектурой.

Модели в контексте веб-разработки с использованием фреймворка Django представляют структуры данных, которые определяют способ организации и хранения информации в базе данных. Модели Django обеспечивают абстракцию для создания и управления данными в приложении. Они определяют типы данных, атрибуты и отношения между объектами.

Каждая модель Django обычно создается в виде класса Python, который наследуется от базового класса models.Model. Этот класс определяет поля (переменные), представляющие атрибуты объекта, а также методы для управления данными.

Пример модели Django, представляющей объект "Книга":

from django.db import models

class Book(models.Model):

title = models.CharField(max\_length=100)

author = models.CharField(max\_length=50)

publication\_date = models.DateField()

isbn = models.CharField(max\_length=13)

В этом примере Book - модель, а title, author, publication\_date и isbn - это поля модели. Каждое поле определяет тип данных и ограничения для хранения соответствующих атрибутов книги.

Модели Django также поддерживают определение отношений между объектами, таких как один-ко-многим, многие-ко-многим и другие.

Представления (views) в фреймворке Django, представляют собой Python-функции или методы классов, которые обрабатывают HTTP-запросы и возвращают HTTP-ответы. Они играют ключевую роль в обработке запросов от пользователей и определении того, какие данные и какой контент должны быть показаны на веб-странице.

Представления принимают в себя HTTP-запрос и могут взаимодействовать с моделями, шаблонами и другими компонентами фреймворка для создания ответа. Они определяют бизнес-логику приложения и преобразуют данные из моделей в форму, пригодную для отображения в шаблонах или в формате JSON для API.

Пример простого представления Django:

from django.shortcuts import render

from django.http import HttpResponse

def hello(request):

return HttpResponse("Привет, мир!")

В данном примере функция hello принимает HTTP-запрос (request) и возвращает HTTP-ответ (HttpResponse) с текстом "Привет, мир!".

Представления также могут обрабатывать данные форм, параметры URL и другие данные из запроса. Они могут взаимодействовать с базой данных через модели, проводить валидацию данных, исключать ошибки и многое другое.

Для визуализации данных, полученных в представлениях, используются шаблоны (templates), которые определяют структуру и внешний вид веб-страницы.

Важно понимать, что представления организуют бизнес-логику и обработку данных, но они не занимаются прямой генерацией HTML-кода. Эту роль выполняют шаблоны, которые подставляют данные в соответствующие места на веб-странице.

URL-маршрутизация (или маршрутизация URL) в фреймворке Django, процесс определения, каким образом HTTP-запросы будут связаны с определенными представлениями (views) для их обработки. Django использует механизм URL-маршрутизации для определения, какой код должен выполняться при обращении к определенным URL-адресам.

Конфигурация URL-маршрутов осуществляется с использованием файла urls.py, который находится в каждом Django-приложении. В этом файле определяются соответствия между URL-путями и представлениями, которые должны обрабатывать запросы для этих путей.

Пример конфигурации URL-маршрутов в файле urls.py:

from django.urls import path

from . import views

urlpatterns = [ path('hello/', views.hello),

path('articles/<int:article\_id>/',

views.article\_detail), ]

В данном примере urlpatterns является списком путей (URL-шаблонов), которые связаны с соответствующими представлениями. Функция path используется для определения соответствия между URL и представлением. Например, URL-путь 'hello/' будет связан с представлением views.hello, а путь 'articles/<int:article\_id>/' будет связан с представлением views.article\_detail, где <int:article\_id> - это переменная в URL, которая будет передана в представление.

Конфигурация URL может включать различные типы переменных в URL-шаблонах, такие как целочисленные значения (<int:variable\_name>), строки (<str:variable\_name>), и другие.

Эффективная маршрутизация URL позволяет определить структуру вашего веб-приложения, задать понятные и интуитивные URL-адреса и связать их с соответствующими представлениями для обработки запросов.

Шаблоны (templates), используются для создания пользовательского интерфейса веб-приложения. Шаблоны представляют собой файлы, в которых определена структура и внешний вид веб-страницы, а также места, куда будут подставляться данные, полученные из представлений (views).

Основные задачи шаблонов:

Разделение логики и дизайна. Шаблоны позволяют разделять бизнес-логику (которая находится в представлениях) и дизайн веб-страницы. Это упрощает сопровождение и изменение кода.

Динамическое подставление данных. Шаблоны содержат "местозаполнители" или переменные, которые заменяются реальными данными при отображении страницы. Например, можно использовать {{ variable\_name }} для вставки переменных.

Условия и циклы. Шаблоны поддерживают логические условия и циклы, что позволяет выводить различный контент в зависимости от данных.

Пример шаблона Django:

htmlCopy code

<!DOCTYPE html>

<html>

<head>

<title>{{ page\_title }}</title>

</head>

<body>

<h1>Добро пожаловать, {{ user\_name }}!</h1>

<ul> {% for article in articles %}

<li><a href="{{ article.url }}">{{ article.title }}</a></li>

{% endfor %}

</ul>

</body>

</html>

В данном примере {{ page\_title }}, {{ user\_name }} и {% for article in articles %} - это местозаполнители и конструкции шаблонизации. Переменные подставляются в шаблоне на основе данных, переданных из представления, итерации выполняются по списку articles, и т.д.

Шаблоны облегчают создание красивых и динамических пользовательских интерфейсов, а использование шаблонных тегов и фильтров дает гибкость для форматирования и обработки данных.

Встроенная административная панель (Django Admin) предоставляет интерфейс для управления данными вашего веб-приложения через веб-браузер. Эта админ-панель предоставляет мощный инструментарий для управления моделями и данными в вашей базе данных без необходимости написания дополнительного кода.

Некоторые ключевые особенности встроенной административной панели Django:

Автоматическое создание интерфейса. После определения моделей в приложении, админ-панель автоматически создает интерфейс для их управления. Это включает в себя создание, редактирование, удаление и просмотр объектов.

Настройка отображения полей. Можно настраивать, какие поля отображаются для каждой модели в админ-панели, а также порядок их отображения.

Поиск и фильтрация. Административная панель предоставляет возможность поиска и фильтрации объектов на основе значений полей.

Создание пользовательских действий. Можно определить пользовательские действия, которые можно выполнять над выбранными объектами, например, массовое обновление полей.

Импорт и экспорт данных. Административная панель позволяет импортировать и экспортировать данные в различных форматах, таких как CSV или JSON.

Безопасность. Доступ к административной панели может быть ограничен различными уровнями прав доступа, что обеспечивает безопасность управления данными.

Кастомизация. Можно настраивать внешний вид админ-панели, добавлять пользовательские стили и скрипты, а также создавать собственные страницы для управления данными.

Для включения встроенной административной панели в Django-приложение, нужно зарегистрировать модели в файле admin.py, который находится внутри приложения. В этом файле можно определить, как модели будут отображаться и управляться в административной панели.

В целом, Django Admin предоставляет удобный и быстрый способ управления данными и администрирования ваших приложений без необходимости писать дополнительный код для создания административных интерфейсов.

Работа с базой данных:

Django поддерживает несколько СУБД (систем управления базами данных), включая PostgreSQL, MySQL, SQLite и другие.

Создание моделей в Django ключевой шаг при разработке приложения. Модели определяют структуру данных вашей базы данных.

Миграции в Django представляют собой способ автоматически создавать и обновлять структуру базы данных в соответствии с вашими моделями. Это позволяет вам безопасно изменять схему базы данных, не теряя данные.

Вот основные шаги работы с миграциями:

Создание миграции. После определения новой модели или внесения изменений в существующие модели необходимо запустить команду python manage.py makemigrations. Django создаст файл миграции, который содержит инструкции о том, как изменить структуру базы данных.

Применение миграции. При запуске команды python manage.py migrate фактически применяется миграция к базе данных. Django автоматически обновит схему базы данных в соответствии с изменениями, описанными в миграции.

Откат миграции (по желанию). Если необходимо откатить изменения, необходимо выполнить python manage.py migrate <app\_name> <migration\_name>. Это вернет базу данных к состоянию до применения указанной миграции.

Миграции должны быть частью версионного контроля проекта, чтобы коллеги или другие разработчики могли применить их на своих системах.

Работа с формами:

В веб-разработке необходимо собирать информацию от пользователей: от регистрации и авторизации до заполнения профилей и отправки запросов. Формы позволяют собирать, проверять и сохранять информацию в базе данных.

Создание форм в Django начинается с определения класса формы. Можно использовать стандартные поля, такие как CharField, EmailField, IntegerField и многие другие, чтобы определить типы данных, получаемых от пользователя.

После того как пользователь отправит данные через форму, Django предоставляет встроенный механизм валидации. Этот механизм проверяет данные на соответствие заданным правилам, таким как обязательное заполнение, формат электронной почты и другие. Если данные не соответствуют правилам, можно легко показать ошибки пользователю.

Безопасность:

Django, как популярный фреймворк для веб-разработки, предоставляет множество инструментов и мер безопасности. Вот некоторые ключевые аспекты безопасности в Django:

Django предоставляет встроенную защиту от атак CSRF(Cross-Site Request Forgery). Каждый раз, когда форма отображается, в неё добавляется уникальный токен. При отправке данных этот токен проверяется, чтобы удостовериться, что запрос действительно пришел с ожидаемой страницы.

Django по умолчанию использует подготовленные запросы и параметры, чтобы избежать атак SQL-инъекций.

Защита от XSS (Cross-Site Scripting). Django автоматически экранирует данные, чтобы предотвратить атаки на основе внедрения вредоносных скриптов в страницы.

Django предоставляет защиту от атак, которые пытаются обмануть пользователя, скрывая определенные элементы страницы под другими, имеет готовые механизмы аутентификации и авторизации, позволяющие управлять доступом к различным частям приложения, использует хеширование паролей с добавлением "соли" для обеспечения безопасности хранения паролей пользователей.

Предоставляет удобные способы управления загрузкой и предотвращения доступа к файлам, которые не должны быть общедоступными. В режиме разработки Django может отображать подробную информацию об ошибках. В production-среде этот режим следует отключить, чтобы не выставлять наружу лишнюю информацию.

Для поддержания безопасности приложения важно следить за релизами Django и устанавливать обновления, в которых могут быть исправлены уязвимости. Однако, важно также понимать основы безопасности и соблюдать bewst practices при разработке приложений на Django, чтобы обеспечить защиту данных пользователей и предотвратить атаки.

Работа со статическими файлами.

Django предоставляет удобные инструменты для управления такими файлами, такими как CSS, JavaScript, изображения и другие.

Django имеет встроенный механизм для обработки статических файлов. По умолчанию, в режиме разработки, будет обслуживать статические файлы напрямую. Однако, в production-среде, это не рекомендуется, и следует использовать веб-сервер (например, Nginx или Apache) для обслуживания статики.

Обычно статические файлы размещаются в директории с именем "static" внутри каждого Django приложения. Каждое приложение может иметь свою собственную папку "static". Например:

arduinok

myapp/

static/

myapp/

css/

js/

images/

Для подключения статических файлов в HTML-шаблонах, можно использовать тег {% static %}. Например:

html

{% load static %}

<link rel="stylesheet" type="text/css" href="{% static 'myapp/css/style.css' %}">

<script src="{% static 'myapp/js/script.js' %}"></script>

<img src="{% static 'myapp/images/logo.png' %}" alt="Logo">

Перед тем как развернуть приложение в production-среде, рекомендуется собрать статические файлы в одной папке. Для этого используется команда collectstatic. Она скопирует все статические файлы из разных приложений в одну общую директорию.

Чтобы использовать статические файлы сторонних библиотек (например, Bootstrap, jQuery), можно подключить их либо локально, либо через CDN.

В файле настроек settings.py можно настроить параметры работы со статическими файлами, такие как STATIC\_URL (URL для статических файлов), STATIC\_ROOT (путь для собранных статических файлов) и другие.

В заключение можно сказать, что Django - мощный фреймворк для веб-разработки, обеспечивающий эффективное создание и поддержку веб-приложений. Его интегрированные инструменты, такие как ORM, административная панель и система маршрутизации, значительно упрощают процесс разработки. Django также обладает хорошей документацией и активным сообществом, что обеспечивает поддержку и ресурсы для разработчиков.

**3. Практическая часть**

При работе над проектом я использовал PyCharm. Он позаботился о рутинных задачах, что помогло сосредоточиться на более важных вещах, обеспечил точное автоматическое дополнение, поиск ошибок и быстрые исправления, удобную навигацию по коду и другие полезные функции.

И конечно же, как не сказать о Python. Он является основой проекта и помогает решить проблему управления зависимостями проектов. В Python уже есть модуль venv для создания виртуальных сред, который можно использовать как в разработке, так и в производстве. Установил виртуальную среду, Python 3.10.

Итак, основным инструментом при создании проекта использовался Django. Концепция Модель-Представление-Шаблон (MVT) - основа фреймворка Django. Она разделяет приложение на три основных компонента: модель (Model), представление (View) и шаблон (Template). Установил фреймворк используя команду pip install django. Помимо самого фреймворка было установлено несколько обязательных зависимостей. Сам Django содержит в себе около 15 пакетов, позволяющих решать большинство задач разработки без установки дополнительных пакетов и модулей.

В Django различаются термины проект и приложение. Проект представляет из себя пакет Python с базовыми настройками. Приложение также является пакетом. Но он входит в состав проекта. При этом каждый проект может состоять из нескольких приложений. Для создания своего проекта я использовал команду

django-admin startproject web\_project

команда создала структуру проекта, которая содержит все необходимые файлы и папки для работы с фреймворком. У меня получилось вот так:

web\_project

manage.py

web\_project

\_\_init\_\_.py

settings.py

urls.py

asgi.py

wsgi.py

manage.py - файл, который используется для управления проектом. С его помощью можно запустить сервер, создать миграции, создать суперпользователя и т.д

web\_project - директория, которая содержит основные файлы проекта.

\_\_init\_\_.py - файл, который сообщает Python, что директория web\_project является пакетом

urls.py - файл, который содержит маршруты приложения.

asgi.py - файл, который используется для запуска проекта в ASGIсовместимых серверах.

wsgi.py - файл, который используется для запуска проекта в WSGIсовместимых серверах.

Для проверки работоспособности сервера мною использовалась команда:

python manage.py runserver

Эта команда запускает сервер на локальном хосте и порту 8000. Если есть проблемы можно изменить порт командой:

python manage.py runserver 8001

Сервер Django используется лишь при разработке и тестировании проекта. Он способен раздавать статику в виде изображений, CSS файлов, JavaScript кода и т.п. Так же сервер отслеживает изменения, которые вносятся в файлы проекта. После каждого такого изменения сервер автоматически перезагружается.

В процессе работы над проектом использовался режим отладки. В файле

настроек settings.py есть строка кода, означающая, что включена отладка кода: DEBUG = True

Данная настройка помогла мне с настройкой проекта.

Далее перешёл к созданию приложения, для чего было необходимо создать отдельный модуль, который будет содержать логику и шаблоны для определенной функциональности. Для создания приложения выполнил команду:

python manage.py startapp djapp

Получил вот такую структуру:

web\_project /

djapp/

migrations/

\_\_init\_\_.py

\_\_init\_\_.py

admin.py

apps.py

models.py

tests.py

views.py

web\_project /

\_\_init\_\_.py

settings.py

urls.py

asgi.py

wsgi.py

db.sqlite

manage.py

Добавились новые данные:

djapp / - директория приложения

migrations/ - директория для хранения миграций базы данных

\_\_init\_\_.py - файл, указывающий на то, что директория является пакетом Python

\_\_init\_\_.py - файл, указывающий на то, что директория является пакетом Python

admin.py - файл для настройки административного интерфейса приложения

apps.py - файл для настройки приложения

models.py - файл, содержащий модели данных приложения

tests.py - файл для написания тестов приложения

views.py - файл, содержащий представления (views) приложения

db.sqlite - файл базы данных SQLite

manage.py - файл для управления проектом Django (запуск сервера, создание миграций и т.д.)Чтобы приложение запускалось, его необходимо добавить в файл settings.py. В список INSTALLED\_APPS, что и было проделано:

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'djapp'

]

Следующий пункт работы - создать представление. Для создания представления нужно определить функцию в файле views.py, которая будет обрабатывать запрос на эту страницу. Приведу часть кода, которая создана для создания основных страниц:

from django.shortcuts import render

def index(request):

return render(request, 'djapp/index.html', )

def about\_us(request):

return render(request, 'djapp/about.html')

def FAQ(request):

return render(request, 'djapp/FAQs.html')

Для настройки URL в Django необходимо определить маршруты (routes), связывающие определенные URL с соответствующими представлениями (views) в приложении. Маршруты определяются в файле urls.py, который находится в корневой директории проекта и в директории приложения. Файл, по умолчанию, уже содержит необходимые данные. Добавляю маршрут для работы своего приложения:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('djapp.urls')),

]

Этот код импортирует модуль admin из пакета django.contrib.

Импортируется функция path и модуль include из пакета django.urls. Обе функции нужны для формирования url адресов, на которые будет отвечать сервер

Создается список urlpatterns, который будет содержать маршруты (routes) для обработки URL-адресов.

Добавляется маршрут для административной панели Django, который будет обрабатывать URL-адрес, начинающийся с префикса "admin/" и передавать управление в модуль admin.site.urls.

Добавляется маршрут для приложения djapp, который будет обрабатывать пустой URL-адрес и передавать управление в модуль djapp.urls. Маршрут включается с помощью функции include. Затем добавил одну строку в список urlpatterns. В path('', include('djapp.urls')) ничего не указываю('' пустые кавычки), чтобы при обращении по адресу вызывался маршрут к моему приложению автоматически.

При создании приложения в нем нет файла urls.py, соответственно его создал сам. В процессе создания проекта добавлял в него данные. Вот маршрут для запуска моей первой страницы:

from django.urls import path

from . import views

urlpatterns = [

path('', views.index, name='main\_page'),

Здесь импортируется функция path, которая принимает два аргумента: первый аргумент - URL-адрес, а второй – представление, отвечающее за обработку запроса на этот URL путь до файла views. Задаётся имя маршрута с помощью параметра name. Маршрут связывает корневой URL с представлением index. И тут я не указываю адрес, чтобы автоматически запускался данный маршрут.

Для логирования запуска моей первой страницы добавил изменение в код. В файл settings.py:

LOGGING = {

'version': 1,

'disable\_existing\_loggers': False,

'handlers': {

'console': {

'class': 'logging.StreamHandler',

},

'file': {

'class': 'logging.FileHandler',

'filename': 'log/django.log',

},

},

'loggers': {

'django': {

'handlers': ['console', 'file'],

'level': 'INFO',

},

'djapp': {

'handlers': ['console', 'file'],

'level': 'DEBUG',

'propagate': True,

},

},

}

Добавил директорию Log и в ней файл django.log. В этом файле буду храниться необходимые мне логи. Теперь первое представление выглядит так:

from django.shortcuts import render

from .models import Trubi

import logging

logger = logging.getLogger(\_\_name\_\_)

def index(request):

logger.info('Index page accessed')

return render(request, 'djapp/index.html', )

Путь до представления прописан, теперь переходим к созданию базового шаблона. Это то, как будет выглядеть моя страница на сайте.

Шаблоны в Django представляют собой файлы, содержащие HTML-код с

дополнительными тегами и переменными, которые могут быть заменены на

значения из контекста.

Прежде чем создавать первый шаблон, внутри каталога приложения создал каталог templates. Далее в нём создал каталог с именем приложения djapp.

Создал base.html:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

B верху располагается «голова», где прописываются основные и дополнительные настройки, например, name=”viewport” означает, что шаблон будет пытаться отображаться более красиво на других устройствах помимо ПК (без этой настройки на телефонах будет требоваться прокрутка по экрану в ширь, чтобы увидеть весь сайт)

<link rel="stylesheet" type="text/css" href="{% static 'hellodj/style.css'%}">

означает, что к файлу присоединён файл css, который увеличивает красоту шаблона.

Base.html является главным и основным шаблоном для всех остальных html-файлов. В него входит: Верхняя часть (header), Нижняя часть (footer) и nav-bar для перемещения между файлами.

Всё это находится в классе wrapper, где используется flexbox и ставится минимальный размер 100% (Нужно, чтобы footer всегда оставался внизу страницы, даже если страница не заполнена)

Классы header\_bar и header\_container служат “базой” для размещения кнопки для меню перехода между страницами (dropup, dropdown,dropdown-button nav nav-pills) где находится 3 ссылки: на Главную страницу, на страницу частых вопросов и страницу о нас.

<div>  
 <footer class="d-flex flex-wrap justify-content-between align-items-center py-3 my-4 border-top">  
 <p class="col-md-4 mb-0 text-body-secondary">© 2023 Стальной шедевр, все права защищены Inc</p>

В конце шаблона есть footer, который завершает шаблон, добавляя большей красоты. Сделан таким образом, что даже если на странице меньше контента, чем длина экрана пользователя, он всё равно окажется в самом низу.

<div class="container">

{% block content %} {% endblock %}

</div>

Эти блоки позволяют встраивать уникальный контент для каждой страицы.

Чтобы передать данные в шаблон, использовал функцию render из модуля django.shortcuts:

from django.shortcuts import render

def about\_us(request):

return render(request, 'djapp/about.html')

В этом примере создал функцию about\_us, которая использует шаблон about.html из приложения djapp. Функция render заменяет переменные в шаблоне на значения из контекста и возвращает готовую HTML-страницу.

Далее использовал возможность наследования шаблонов, что позволило создать базовый шаблон и на его основе создавать другие, которые имеют общий вид и функциональность. Наследование шаблонов позволило избежать дублирования кода и упростить процесс разработки:

{% extends 'djapp/base.html' %}

{% block title %}

{%load static%}

Уголки

{%endblock%}

{% block content %}

{%load static%}

<div class="Characteristics">

Характеристики

<div class="CBox">

<div class="CCBox">

<div class="picture">

<img src="{% static 'images/Shveller.png' %}">

</div>

<div class="other">

<div class="Info">Высота

<div>18 мм</div>

</div>

<div class="Info">Ширина

<div>18 мм</div>

</div>

<div class="Info">Толщина

<div>9 мм</div>

</div>

<div class="Info">ГОСТ

<div>8732-78 Группа В62</div>

</div>

</div>

</div>

</div>

</div>

{% endblock %}

Приведенный файл наследуется от base.html и вносит изменения в основной части.

Перешел к созданию необходимых стилей, создал папку static в ней папку hellodj, файл style.css, после чего получил следующее:

\* { }

Отвечает за свойства всего сайта (Стиль шрифта)

.dropup, .dropdown, .dropdown-button, { }

Отвечает за свойства меню перехода (На страницы сайта) на header-е

Body, html { }

Отвечают за свойства контейнеров Body и html (цвет фона)

a { }

Отвечают за свойства контейнера a (цвет шрифта)

.Header bar, d-flex, .header\_container { }

Отвечают за свойства Верхней части сайта (Цвет, фон, расположения других контейнеров,

DivHead1 { }

Отвечают за свойства этого контейнера и ему похожему:

Img { }

Отвечает за свойства картинок на сайте

.wrapper { }

Отвечает за свойства контейнера, в котором лежат все другие

.navpills, nav-items { }

Отвечают за кнопки, через которые осуществляется переход на другие страницы сайта

.row, .col, .card, .card-body, .card-text, { }

Отвечают за корректное отображение карточек с информацией о продукте

Btn-sm, btn-group { }

Отвечают за корректное отображения кнопки на карточках

Footer,footer-font { }

Отвечают за корректное отображения «подвала»

.image\_holder { }

Отвечает за правильное отображение контейнера, содержащего картинку

.Characteristics, CBox, CCbox, .other, .Info { }

Отвечают за корректное отображение информации о продукте

.h2 { }

Отвечает за более насыщенный шрифт для букв, написанных на страницах «О нас» и «Частые вопросы».

В папке static создал папку images для хранения изображений, связанных с проектом. Она создана для удобства организации медиа-файлов, таких как фотографии, рисунки и другие графические файлы. Эти изображения загружены и обработаны в рамках Django-приложения. В первую очередь поместил графические элементы, которые потребовал мой проект. Это помогло в организации и управлении медиа-контентом приложения.

Для удобной работы решил создать небольшую базу данных, для этого использовал модель, позволяющую создавать, изменять и удалять записи в базе данных без необходимости написания SQL-запросов. Используемые модели позволяют описывать отношения между таблицами, что делает работу с базой данных удобной и эффективной.

В файле models.py создал:

from django.db import models

class Trubi(models.Model):

name = models.CharField('Тип/название трубы', max\_length=100)

wall = models.CharField('Cтенка (мм)', max\_length=10)

length = models.CharField('Длина (Метры)', max\_length=10)

weight = models.CharField('Вес 1 единицы (в погонных метрах)', max\_length=10)

price = models.CharField('Цена 1 единицы (в погонных метрах)', max\_length=10)

priceT = models.CharField('Цена 1 Тонны (в погонных метрах)', max\_length=25)

def \_\_str\_\_(self):

return self.name

class Meta:

verbose\_name = 'трубу в таблицу "Трубы"'

class Image(models.Model):

image = models.ImageField(upload\_to='media/hellodj/')

Каждый атрибут класса соответствует полю таблицы, а экземпляры класса представляют записи в таблице.

С помощью команды:

python manage.py makemigrations

создал миграцию, которая автоматически сгенерировала скрипты для изменения структуры базы данных в соответствии с изменениями в моделях приложения.

Далее использовал команду:

python manage.py migrate.

Команда "migrate" выполнила все накопленные миграции в порядке их создания и применила изменения к базе данных.

Для возможности создания, редактирования и удаления записей в базе данных занялся созданием административной панели. Во-первых, она значительно упрощает работу, позволяя быстро и удобно управлять данными.

Во-вторых, административная панель может быть использована в качестве удобного инструмента для администрирования приложения.

В urls.py автоматически уже прописан путь и импортированы необходимые модули:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('djapp.urls')),

]

Создал суперпользователя командой: python manage.py createsuperuser

Подключил модель к административной панели. В файле admin.py приложения:

from django.contrib import admin

from .models import Trubi

admin.site.register(Trubi)

Административная панель Django обладает достаточно широким набором

инструментов, позволяя пользоваться административной панелью написав минимум строк кода.

**Заключение**

В ходе разработки проекта мною был создан полнофункциональный веб-сайт на базе фреймворка Django. Проект включает в себя следующее. Разработана функциональность для просмотра и поиска товаров. Каждый товар имеет описание, изображение. Администраторы имеют доступ к панели управления, где могут просматривать и обрабатывать заказы, управлять товарами и пользователями. Сайт адаптирован под различные устройства, что обеспечивает удобство использования как на компьютерах, так и на мобильных устройствах. Применены меры безопасности, такие как хеширование паролей пользователей и защита от атак на уровне приложения.

Важно отметить, что при разработке были использованы передовые технологии и лучшие практики веб-разработки. Код проекта был организован в соответствии с принципами чистого и эффективного программирования. Такие как: Django 4.2: Фреймворк, используемый для разработки веб-приложений на языке Python. Django предоставляет множество готовых компонентов, что ускоряет процесс разработки. Python 3.10. HTML5/CSS3: Стандартные языки разметки и стилей для создания пользовательского интерфейса. SQLite: Встроенная реляционная база данных, выбранная для простоты развертывания и тестирования.

Применены архитектурные решения MVC (Model-View-Controller): Проект организован в соответствии с архитектурным шаблоном MVC, что способствует разделению логики данных, бизнес-логики и пользовательского интерфейса.

Создание этого сайта подтверждает полученные навыки в разработке веб-приложений и демонстрирует понимание принципов проектирования и программирования.

Данный проект служит примером профессионального и творческого подхода к решению задач в области веб-разработки.

**Список литературы и ресурсов**

1. "Django Girls Tutorial" by Django Girls.
2. "Django RESTful Web Services" by Gaston C. Hillar.
3. Официальная документация Django.
4. Django for Beginners" by William S. Vincent.
5. "Django 3 By Example" by Antonio Mele.
6. "Two Scoops of Django 3.x: Best Practices for the Django Web Framework" by Daniel Roy Greenfeld and Audrey Roy Greenfeld.
7. "Obey the Testing Goat: Using Django, Selenium, and JavaScript" by Harry J.W. Percival.
8. "Django Design Patterns and Best Practices" by Arun Ravindran.
9. "Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, and JavaScript" by Harry J.W. Percival.