

«Bärendreck»,  
онлайн-словарь вариантов немецкого языка

**Специальность, Geekbrains**

**Щербак Сергей Андреевич**

**2024**

СОДЕРЖАНИЕ:

1. Введение.
2. Теоретическая часть.
   1. Немецкий язык
      1. Краткие сведения о немецком языке.
      2. Диалекты и региональные варианты немецкого языка.
   2. Инструменты проекта
      1. HTML и CSS
      2. Язык программирования JavaScript.
      3. СУБД данных SQLite.
3. Практическая часть.
   1. Разработка и создание дизайна для страницы проекта.
   2. Наполнение базы данных
   3. Размещение проекта в интернете
4. Заключение.
5. Список использованной литературы.
6. Приложения.
   * + 1. Введение.

Настоящий дипломный проект – «Bärendreck», онлайн-словарь вариантов и диалектов немецкого языка (основной литературный

Тема проекта: Исследование особенностей организации процесса ручного тестирования веб-приложений в условиях отсутствия технического задания на примере тестирования интернет-магазина парфюмерии и косметики.

Цель: Изучить особенности ручного тестирования веб-приложений в условиях отсутствия спецификации к программному продукту и разработать предложения по улучшению процесса ручного тестирования веб-приложения РИВ ГОШ.

Какую проблему решает: Случается такое, что необходимо протестировать и выпустить продукт без требований, или на их изучение нет времени.

Задачи:

1. Изучить литературу, касающуюся темы исследования.

2. Рассмотреть основные виды и методы ручного тестирования веб-приложений.

3. Ознакомиться с основными принципами составления, такой тестовой документации, как чек-листы, тест-кейсы, тест-планы, баг-репорты.

4. Составить план ручного тестирования веб-приложения РИВ ГОШ с использованием техник тест-дизайна.

5. Выполнить ручное тестирование веб-приложения РИВ ГОШ.

6. Разработать предложения по улучшению ручного тестирования веб-приложения РИВ ГОШ.

Инструменты: Postman, Qase.io, TestRail или Jira, DevTools, Git, Charles Proxy, FastStone Capture 7.2.

Состав команды: ФИО (Тестировщик)

* + - 1. Теоретическая часть.
  1. Немецкий язык.
     1. Краткие сведения о немецком языке.

Немецкий язык (Deutsch) принадлежит к западногерманской группе германских языков и является одним из самых распространенных языков в мире. Германская языковая семья, к которой относится немецкий, известна своей богатой историей и значительным влиянием на культуру и науку.

Одной из особенностей немецкого языка является его грамматическая структура. Немецкий язык обладает грамматическим родом, числовыми и падежными формами, а также строгими правилами согласования между подлежащим и сказуемым. Благодаря этим особенностям, немецкий язык оправдывает репутацию языка, в котором словообразование и синтаксис должны быть внимательно проверены и соблюдены.

Немецкий язык также известен своим богатым словарным запасом. В нем содержатся множество слов и выражений, которые воплощают разнообразные концепции и идеи, что отражает культурные и интеллектуальные традиции тех, кто говорит на этом языке. Немецкие авторы, философы и ученые внесли значительный вклад в мировую литературу, философию и науку, и их работы переведены на множество языков, включая русский.

Немецкий язык имеет обширный словарный запас, состоящий из слов различных происхождений. Он включает основной немецкий лексикон, наследованный от германской языковой группы, а также лексику, заимствованную из других языков, таких как латинский, французский, английский и другие.

Этимология слов в немецком языке может быть очень интересной, поскольку она иногда позволяет проследить исторические и культурные связи между разными языками. Например, многие слова в немецком языке имеют общую основу с английскими словами, что объясняется их общим происхождением от германского языка.

Семантика — это другой важный аспект лексикологии. Она изучает значения слов и способы их организации в лексической системе. В немецком языке существует множество понятийных полей, где группы слов связаны общими значениями. Например, в поле "фрукты" собраны слова, обозначающие различные виды фруктов, такие как "Apfel" (яблоко), "Banane" (банан), "Orange" (апельсин) и т.д.

Кроме этого, в немецком языке существуют синонимы и антонимы, которые обогащают разнообразие выражения. Например, слова "fröhlich" (веселый) и "glücklich" (счастливый) имеют схожее значение, а слова "heiß" (горячий) и "kalt" (холодный) являются примером антонимов.

Также стоит отметить, что в немецком языке много сложных слов, образованных путем соединения нескольких лексических элементов. Например, слово "Dampfschifffahrtskapitän" (капитан парохода) состоит из трех основных частей, каждая из которых вносит свой смысл к общему значению слова.

Немецкий язык также богат различными группами сленга и жаргона, которые могут варьироваться в зависимости от региона и социальной группы. Это создает множество вариантов и выражений в рамках немецкого языка и позволяет говорящим выразить свою индивидуальность и принадлежность к определенной группе.

Важно отметить, что немецкий язык часто используется в научной и академической среде. Многие важные научные работы и исследования изначально публикуются на немецком языке, и изучение немецкого может открыть дополнительные возможности для научного и интеллектуального развития. Также стоит отметить богатство лексического состава немецкого языка. В немецком есть множество слов, которые имеют свои аналоги в других языках, но при этом они часто обладают уникальными нюансами и оттенками значения. Это делает немецкий язык интересным для изучения и позволяет расширить словарный запас.

Еще одной интересной особенностью немецкого языка является наличие диалектов. В различных регионах Германии, Швейцарии и Австрии употребляются различные диалекты, которые могут отличаться по произношению, лексике и грамматике. Это может создавать сложности для общения между представителями разных регионов, но в то же время придает языку большую эстетическую разнообразность.

Немецкий язык также играет важную роль в международном бизнесе и туризме. Германия является крупной экономической державой, и знание немецкого языка может предоставить широкий спектр возможностей для карьерного роста и открыть двери к сотрудничеству с немецкими компаниями.

В заключение, немецкий язык имеет свои уникальные особенности, которые делают его интересным для изучения и использования. Он является частью богатой культурной и языковой традиции и имеет большое значение в науке, литературе, бизнесе и международных отношениях. Изучение немецкого языка может быть увлекательным и позволить расширить границы коммуникации и культурного обмена.

* + 1. Диалекты и региональные варианты немецкого языка.

Немецкий язык обладает множеством диалектов и региональных вариантов, которые различаются по произношению, лексике и грамматике. Эти различия могут быть настолько значительными, что иногда даже представители разных регионов могут испытывать трудности во взаимопонимании.

Одним из известных диалектов немецкого языка является баварский, который наиболее распространен в южной части Германии и Австрии. Баварский диалект характеризуется особым произношением, особенно в звуковом сочетании "ch". Например, слово "ich" (я) в стандартном немецком произносится [ɪç], в то время как в баварском оно звучит как [i]. Другой известный диалект - алеманский, преимущественно употребляется в южной части Германии, Швейцарии и Австрии. Алеманский диалект также имеет свои особенности в произношении и грамматике. Например, в алеманском диалекте слово "ich" произносится как [ɪg] или [iɡ]. Также стоит упомянуть франконский диалект, распространенный в Северной Баварии и некоторых частях ФРГ, Вюртембергский диалект, используемый в Штутгарте и окрестностях, а также рейнский диалект, характерный для Рейнской области и численных приграничных регионов.

Каждый диалект имеет свои уникальные особенности, которые отражают богатство и многообразие немецкого языка. Они также могут включать в себя специфические лексические выражения, фразы и даже грамматические конструкции, которые могут отличаться от стандартного немецкого языка. Однако несмотря на эти различия, стандартный немецкий язык всегда остается общим средством коммуникации между говорящими на разных диалектах.

В заключение, диалекты и региональные варианты немецкого языка придают языку его уникальность и культурное наследие. Они являются важным аспектом изучения немецкого языка и позволяют лучше понять культуру и традиции различных регионов, где используется немецкий язык.

Помимо описанных ранее диалектов, существует множество других местных вариантов немецкого языка, которые имеют свою уникальность. Например, в Северной Германии распространен верхненемецкий диалект, который включает в себя региональные варианты, такие как нижненемецкий, кельский и хессенский диалекты. Восточная Германия характеризуется наличием сильной прослойки лексических и грамматических особенностей, которые отличаются от западных региональных вариантов.

Каждый диалект и региональный вариант немецкого языка имеет свою историю и культурные корни. Их ареалы определяются географическими, историческими и социокультурными факторами. Различные диалекты и языковые варианты часто отражают историческую сложность и разнообразие регионов, в которых они зародились и сохранились.

Диалекты немецкого языка также находят отражение в литературе, музыке и других культурных проявлениях. Множество писателей, поэтов и композиторов использовали диалекты для выражения своего личного стиля и отражения культурного наследия своих родных регионов. Это позволяет нам узнать больше о различных культурах и традициях внутри Германии и других немецкоязычных стран.

Однако не следует забывать, что стандартный немецкий язык (Hochdeutsch) остается государственным и общим для всех немецкоязычных регионов. Это язык образования, делового общения и официального общения в Германии и других странах. В школах и университетах преподаются основы стандартного немецкого языка, а также ведется его стандартизация через учебники и правила грамматики. Стандартный немецкий язык также используется в СМИ и в официальных документах.

В заключение, немецкий язык богат диалектами и региональными вариантами, которые отражают культурное разнообразие и историческое наследие немецкоязычных регионов. Понимание и изучение этих различий помогает нам более глубоко погрузиться в немецкую культуру и лучше понять многообразие языка и стилей, которые сопутствуют его использованию в разных регионах.

Особенности немецкого языка в Германской Демократической Республике (ГДР) отличались от использования немецкого языка в других германоязычных регионах. Во время существования ГДР (1949-1990), язык испытал влияние социально-политических факторов и отдельных политических лидеров, что привело к некоторым особенностям в его использовании.

1. Социалистический словарь: В ГДР стремились к созданию своего уникального лексикона, отличавшегося от использования немецкого языка на западе. Были введены новые термины и выражения, связанные с социалистической и марксистско-ленинской идеологией. Некоторые слова имели специфический семантический оттенок, относящийся к социалистической системе.

2. Использование советской нормы: Влияние советского стиля и норм русского языка также было заметно в использовании немецкого языка в ГДР. Например, применение конструкций, подвижных приставок и словосочетаний, аналогичных русским языковым образцам.

3. Отличия в произношении: Некоторые говорящие на немецком языке в ГДР могли иметь отличия в произношении некоторых звуков и интонаций из-за региональной вариации или влияния русского языка.

4. Ограничения на использование слов и выражений: В определенные периоды существования ГДР существовали цензурные ограничения на использование определенных слов и выражений. Власти стремились контролировать язык и его использование, чтобы поддерживать специфическую политическую атмосферу.

5. Влияние других региональных вариантов немецкого языка: В ГДР встречались различные диалекты и варианты немецкого языка. Некоторые из них могли иметь свои уникальные особенности в произношении, грамматике и лексике.

6. Идеологическая терминология: В ГДР было распространено использование специфической идеологической терминологии, связанной с коммунистической идеологией. Например, часто употреблялись термины, такие как "Klassenkampf" (борьба классов), "Sozialismus" (социализм) и "Realsozialismus" (реальный социализм).

7. Влияние русского языка: Благодаря близким связям с СССР, русский язык оказал значительное влияние на немецкий язык в ГДР. Введение советской нормы и использование русских заимствований сделало некоторые выражения и конструкции уникальными для ГДР.

8. Административная лексика: В связи с социалистической организацией государства и экономики, в ГДР была развита специфическая административная лексика. Были введены новые термины и выражения, связанные с работой государственных органов, планированием и контролем экономики.

9. Культурные отличия: В ГДР поддерживалась специфическая культура и искусство. Это отразилось и в использовании языка. В ГДР развивалась своя литературная традиция, к которой относились такие выдающиеся писатели, как Бертольд Брехт и Штефан Гейслер. Их произведения и стиль письма имели свои особенности, которые отличались от используемого на Западе.

10. Устаревшие выражения: Со временем, после объединения Германии, некоторые выражения и фразы, использовавшиеся в ГДР, стали устаревшими в современном немецком языке. Однако, они могут быть все же встречены в текстах, документах и литературе, отражая исторический контекст и время создания.

В целом, немецкий язык в ГДР имел свои уникальные особенности, вызванные социально-политическим контекстом времени и стремлением властей формировать специфическую идеологическую и культурную среду. Эти особенности создавали определенную отличительную атмосферу в использовании языка в ГДР.

Стоит отметить, что многие из этих особенностей постепенно ушли в прошлое после объединения Германии в 1990 году. Однако, ряд слов и выражений, используемых в ГДР, продолжает присутствовать в современном немецком языке, особенно в восточных регионах Германии.

Общение и изучение немецкого языка позволяют погрузиться в его богатую лексику и открыть для себя множество интересных слов и выражений. Лексикология немецкого языка предлагает множество увлекательных аспектов для изучения и исследования. В целом, изучение лексикологии и семантики немецкого языка позволяет развить более глубокое понимание его словарного запаса и организации слов. Кроме того, это может помочь в изучении немецкого языка и повышении своего уровня владения им.

* 1. Инструменты проекта.
     1. HTML и CSS.

HTML (HyperText Markup Language) — это язык разметки, используемый для создания и структурирования веб-страниц. Он состоит из набора тегов, которые определяют различные элементы и их взаимодействие на странице. HTML является основным языком разметки веб-страниц и является частью разработки веб-сайтов. Он позволяет создавать структуру и содержимое страницы, определять иерархию элементов и добавлять функциональность через другие языки, такие как CSS и JavaScript. HTML (HyperText Markup Language) является языком разметки, используемым для создания веб-страниц. HTML был разработан в начале 1990-х годов Тимом Бернерсом-Ли (Tim Berners-Lee) и его коллегами в ЦЕРНе (Европейский центр физических исследований) в Женеве, Швейцария.

В то время, когда первые веб-страницы стали появляться, нужен был способ для представления текста и других элементов на странице таким образом, чтобы пользователи могли легко переходить по ссылкам и навигировать по информации. HTML был разработан как простой иструктивный язык разметки, который позволял разработчикам создавать структурированный контент с помощью тегов.

Первая версия HTML, известная как HTML 1.0, была выпущена в 1991 году. Она включала основные элементы разметки, такие как заголовки (h1, h2 и так далее), параграфы (p), списки (ul, ol), ссылки (a) и теги для форматирования текста (bold, italic).

С течением времени были выпущены новые версии HTML, включая HTML 2.0, HTML 3.2, HTML 4.01 и HTML5, каждая из которых вносила дополнительные функции и возможности. В настоящее время HTML5 является последней версией и включает в себя множество новых элементов и атрибутов, которые позволяют разработчикам создавать более интерактивные и богатые по функционалу веб-страницы.

HTML стал фундаментальным языком для разработки веб-сайтов и по-прежнему широко используется веб-разработчиками по всему миру.

Вот некоторые основные элементы HTML:

1. Теги: HTML-элементы обозначаются с помощью открывающего и закрывающего тегов. Например, <p>текст параграфа</p> задает параграф, а <h1>заголовок</h1> - заголовок первого уровня.

2. Элементы: HTML поддерживает различные элементы, такие как заголовки, параграфы, списки, таблицы, изображения и формы. Например, <ul> и <li> используются для создания неупорядоченного списка, а <img> для вставки изображений.

3. Атрибуты: Теги могут иметь атрибуты, которые предоставляют дополнительную информацию о элементе. Атрибуты обычно указываются в начале открывающего тега. Например, <a href="https://www.example.com">ссылка</a> задает гиперссылку с атрибутом href, указывающим адрес URL.

4. Структура: HTML-страница имеет определенную структуру. Она начинается с объявления DOCTYPE, которое указывает на версию HTML, и затем содержит <html>, <head> и <body>. <head> содержит метаданные страницы, такие как заголовок и ссылки на внешние стили CSS, а содержимое страницы размещается внутри <body>.

5. Стили и форматирование: HTML предоставляет базовую возможность стилизации с помощью атрибутов стиля, но более распространенное использование для этой цели - это использование CSS (Cascading Style Sheets). CSS позволяет изменять внешний вид HTML-элементов, определяя различные свойства, такие как цвет, шрифт, размер и расположение.

6. Верстка и компоновка: С помощью HTML вы можете создавать различные компоновки элементов на странице. Например, вы можете использовать таблицы (<table>), блочную модель (<div>) или сетку для распределения и организации содержимого.

CSS (Cascading Style Sheets) — это язык стилей, используемый для определения внешнего вида (стилизации) элементов HTML на веб-странице. С помощью CSS вы можете изменять цвета, шрифты, размеры, раскладку и многое другое для создания эстетически приятных и привлекательных веб-страниц. CSS был разработан группой специалистов веб-разработки, включая Хакона Виума Лиебека (Håkon Wium Lie), Берта Боса (Bert Bos) и Тима Бернерс-Ли (Tim Berners-Lee). Работа по спецификации CSS началась в конце 1994 года, и первая редакция CSS была выпущена в 1996 году. С тех пор CSS претерпел множество изменений и стал одним из основных инструментов для стилизации веб-страниц.

Вот некоторые основные концепции CSS:

1. Селекторы: CSS использует селекторы для определения, к каким элементам HTML должны быть применены определенные стили. Например, вы можете выбрать все абзацы (p) или все элементы с определенным классом (#id или .class).

2. Свойства: Свойства CSS определяют внешний вид элементов. Например, свойство color устанавливает цвет текста, а font-size устанавливает размер шрифта. Существует огромное количество свойств CSS, которые позволяют настроить практически все аспекты элемента.

3. Значения: CSS-свойства имеют значения, определяющие конкретные параметры. Например: color: red; задает красный цвет текста, а font-size: 16px; устанавливает размер шрифта в 16 пикселей.

4. Каскадирование и наследование: Каскадирование в CSS означает, что стили могут быть определены несколько раз для одного элемента, и их влияние будет комбинироваться и применяться в порядке приоритета. Наследование означает, что определенные свойства стилей, примененные к родительскому элементу, также могут наследоваться дочерними элементами.

5. Единицы измерения: CSS поддерживает различные единицы измерения, такие как пиксели (px), проценты (%), em, rem и другие. Эти единицы используются для задания размеров, отступов, отступов и других значений в стилях.

6. Медиазапросы: Медиазапросы позволяют создавать стили, которые применяются только при определенных условиях экрана или устройства. Например, вы можете задать особый стиль для мобильного устройства или планшета, используя медиазапросы.

7. Бокс-модель: В CSS каждый элемент имеет бокс-модель, которая определяет пространство вокруг элемента. Бокс-модель включает в себя границу (border), отступы (margin) и внутренние отступы (padding). Вы можете использовать свойства CSS, чтобы настроить размеры и внешний вид каждого компонента бокс-модели.

8. Позиционирование: CSS предоставляет различные методы позиционирования элементов на веб-странице. С помощью свойства position вы можете задать, как элемент должен быть расположен - абсолютно, относительно или фиксированно. Вы также можете использовать свойства top, left, right, bottom для точного позиционирования элементов.

9. Анимации и переходы: CSS позволяет создавать анимации и переходы для элементов на веб-странице. Вы можете задать свойства анимации, такие как animation-name, animation-duration, animation-delay и другие, чтобы создать плавные переходы между состояниями элементов.

10. Верстка на сетке: CSS предоставляет мощные инструменты для создания респонсивной верстки на сетке. Вы можете разбить страницу на горизонтальные и вертикальные столбцы, используя свойства grid или flexbox, что обеспечивает гибкость в расположении элементов на веб-странице в зависимости от размера экрана.

CSS является важной частью разработки веб-страниц и позволяет создавать привлекательный и эстетически приятный дизайн. Путем применения правильных стилей и свойств вы можете предоставить удобный и легко читаемый контент для ваших пользователей.

CSS предоставляет мощные инструменты для улучшения внешнего вида веб-страницы, делая ее более привлекательной и понятной для пользователей. Он часто используется вместе с HTML для создания полноценных и красивых веб-сайтов.

* + 1. Язык программирования JavaScript.
    2. СУБД данных SQLite.

СУБД (Система Управления Базами Данных) - это программное обеспечение, которое обеспечивает организацию, хранение, управление и доступ к данным в базе данных. СУБД предоставляет средства для создания, изменения, удаления и извлечения данных, а также для выполнения различных операций над данными, таких как сортировка, фильтрация и агрегация.

СУБД предоставляет интерфейс для взаимодействия с базой данных с помощью структурированного языка запросов, такого как SQL (Structured Query Language). SQL позволяет программистам и администраторам баз данных выполнять операции по созданию и управлению таблицами, индексами, представлениями и другими составными элементами базы данных.

Преимущества использования СУБД включают:

1. Эффективное хранение данных: СУБД обеспечивает оптимальное использование дискового пространства, благодаря чему данные занимают меньше места и обрабатываются быстрее.

2. Безопасность: СУБД предоставляет механизмы для обеспечения безопасности данных, такие как права доступа и шифрование, что позволяет ограничить доступ к данным только авторизованным пользователям.

3. Целостность данных: СУБД обеспечивает соблюдение ограничений целостности данных, таких как ограничения на уникальность значений и ссылочные целостность, что гарантирует надежность и согласованность данных.

4. Многопользовательская поддержка: СУБД позволяет нескольким пользователям одновременно работать с базой данных, выполняя операции добавления, изменения и извлечения данных, при этом обеспечивая согласованность и последовательность выполнения операций.

Примеры популярных СУБД включают MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL и SQLite. Каждая СУБД имеет свои особенности и подходит для разных сценариев использования в зависимости от требований проекта.

SQLite была разработана командой программистов во главе с Ричардом Хиппом. Хипп является создателем и основным разработчиком SQLite с момента ее начала в 2000 году. SQLite изначально разрабатывалась как небольшая, эффективная, встраиваемая СУБД для использования в приложениях с ограниченными ресурсами и поддержкой доступа через SQL. Эта легковесная и легко настраиваемая СУБД была предназначена для работы в автономном режиме, с возможностью хранения данных в одном файле без необходимости в отдельном сервере. SQLite быстро стала популярной и на данный момент является одной из наиболее широко используемых СУБД в мире. Благодаря своей простоте, эффективности и надежности SQLite активно применяется в различных сферах, включая мобильные приложения, веб-сайты и встроенную электронику.

SQLite - это легковесная встроенная СУБД (система управления базами данных), которая предоставляет надежное хранение и доступ к данным. Она отличается от большинства других СУБД тем, что база данных SQLite хранится в одном файле на диске и не требует отдельного сервера для своей работы.

Особенности SQLite:

1. Простота использования: SQLite имеет простой и понятный синтаксис языка запросов SQL. Это делает ее очень легкой в использовании, даже для начинающих разработчиков.

2. Переносимость: База данных SQLite может работать на различных платформах, включая Windows, macOS, Linux и мобильные операционные системы, такие как Android и iOS. Это обеспечивает высокую степень переносимости и позволяет использовать SQLite в различных типах приложений.

3. Надежность: SQLite обеспечивает целостность данных, поддерживает атомарные операции записи и предоставляет механизмы для обеспечения безопасности базы данных. Благодаря своей встроенной природе, SQLite обеспечивает отказоустойчивость и защиту данных от потери.

4. Экономичность: SQLite не требует больших ресурсов, таких как память или процессорное время. Она имеет небольшой размер, что позволяет эффективно использовать ресурсы системы.

5. Масштабируемость: SQLite может обрабатывать базы данных разного размера - от небольших до очень больших. Она поддерживает индексы, что позволяет ускорить поиск и сортировку данных.

SQLite часто используется в мобильной разработке, особенно для создания локальных баз данных в приложениях. Она также может использоваться в небольших веб-приложениях, прототипировании, тестировании и других сценариях, где простота и независимость от сервера являются важными факторами.

1. Практическая часть.
   1. Разработка и создание дизайна для страницы проекта.

Создание веб-страницы включает в себя несколько основных шагов. Вот общий обзор процесса:

1. Планирование: В первую очередь, перед тем как приступить к созданию веб-страницы, важно провести подробное планирование. Определите цели, аудиторию, функциональность, дизайн и структуру страницы. Разработайте концепцию и схему сайта.

2. Верстка: После планирования переходите к верстке страницы. Верстка представляет собой создание HTML-структуры страницы. Используйте язык разметки HTML для определения структуры контента и его размещения на странице. Разделите элементы на блоки, задайте им соответствующие теги и атрибуты.

3. Стилизация: Следующий шаг - стилизация страницы с помощью CSS. Используйте CSS для определения внешнего вида страницы, задания цветов, шрифтов, размеров и расположения элементов. Создавайте стилистические правила и применяйте их к соответствующим элементам на странице.

4. Добавление контента: После создания структуры и стилизации страницы добавьте контент. Это может быть текст, изображения, видео, таблицы или другие элементы, которые несут информацию или предоставляют функциональность на странице. Разместите контент в соответствующих разделах страницы.

5. Навигация и интерактивность: Добавьте навигационные элементы, такие как меню и ссылки, чтобы пользователи могли легко перемещаться по страницам и взаимодействовать с контентом. Реализуйте интерактивные элементы с помощью JavaScript, если необходимо, чтобы страница была более динамичной и функциональной.

6. Тестирование: Прежде чем опубликовать веб-страницу, проведите тестирование. Проверьте, что все элементы и функциональности работают корректно в разных браузерах и на разных устройствах. Отладьте ошибки и убедитесь, что страница отображается и функционирует должным образом.

7. Развертывание и публикация: После успешного тестирования разверните веб-страницу на сервере, чтобы она была доступна в Интернете. Зарегистрируйте доменное имя, выберите хостинг-провайдера и загрузите файлы вашей страницы на сервер.

Это общий обзор процесса создания веб-страницы. Конкретные шаги и детали могут варьироваться в зависимости от требований вашего проекта.

1. Планирование: Начните с определения структуры и содержимого вашей веб-страницы. Размышляйте о различных разделах и элементах, которые должны присутствовать на странице. Например, заголовки, параграфы, списки, изображения, таблицы и т. д.

2. Создание базовой структуры: Создайте базовую структуру вашей веб-страницы с использованием HTML-тегов. Это включает в себя создание объявления DOCTYPE, открывающего и закрывающего тегов <html>, <head> и <body>. Внутри <head> вы можете добавить различные метаданные, такие как заголовок страницы, описание, ключевые слова, ссылки на CSS файлы и другие.

3. Разметка контента: Добавьте различные HTML-теги для разметки контента на вашей странице. Например, используйте <h1> - <h6> для заголовков разных уровней, <p> для параграфов, <ul> и <li> для списков, <img> для изображений, <table> для таблиц и другие теги в зависимости от ваших потребностей.

4. Форматирование и стилизация: Добавьте CSS-стили или встроенные стили для форматирования и стилизации вашей веб-страницы. Вы можете использовать классы и идентификаторы, чтобы применить стили к определенным элементам на странице. CSS позволяет задавать различные свойства, такие как цвет текста, шрифты, размеры, отступы, рамки и т. д.

5. Создание ссылок и навигации: Добавьте ссылки для перемещения по вашей веб-странице или на другие страницы. Используйте тег <a> для создания гиперссылки. Вы также можете создать навигационное меню, используя список ссылок или другие подходящие элементы.

6. Работа с формами: Если вам необходимо создать формы, используйте тег <form> в HTML. Вы можете добавить элементы формы, такие как текстовые поля, полосы прокрутки, флажки, кнопки отправки и другие, чтобы пользователи могли отправлять данные с вашей веб-страницы.

7. Проверка и отладка: Проверьте свой HTML-код на ошибки и правильность работы. Воспользуйтесь инструментами проверки валидности кода и исправьте ошибки при необходимости. Также убедитесь, что ваша веб-страница корректно отображается на разных браузерах и устройствах.

* 1. Создание проекта на Django.

Это лишь общая схема процесса разработки сайта на Django. В реальности могут быть дополнительные шаги или детали, которые зависят от особенностей вашего проекта. Важно проводить тестирование и следовать лучшим практикам разработки, чтобы создать надежный и эффективный сайт на Django.

Создание сайта на Django — это процесс, который включает в себя несколько основных шагов. Вот общая последовательность действий при разработке сайта на Django:

1. Установка Django: Прежде чем начать работу, вам потребуется установить Django на своем компьютере. Это можно сделать с помощью инструментов установки пакетов Python, таких как pip.

2. Создание проекта Django: После установки Django вы можете создать новый проект с помощью команды django-admin startproject. Это создаст основную структуру проекта, включая файлы управления настройками, маршрутизацией и т.д.

3. Конфигурация базы данных: Django поддерживает различные типы баз данных. Вы можете настроить подключение к базе данных в файле settings.py вашего проекта, указав соответствующие параметры.

4. Создание приложений: В Django приложение - это модуль, который содержит функционал относящийся к конкретной части вашего сайта. Вы можете создать новые приложения с помощью команды python manage.py startapp.

5. Определение моделей: Модели в Django определяют структуру данных, которые будут храниться в вашей базе данных. Вы можете определить модели в файлах models.py в соответствующих приложениях. Django автоматически создаст таблицы базы данных на основе ваших моделей.

6. Создание представлений: Представления в Django определяют, как данные будут отображаться на веб-страницах. Вы можете создать представление в файлах views.py вашего приложения. В представлениях вы можете обрабатывать запросы, извлекать данные из базы данных и передавать их на страницы.

7. Создание шаблонов: Шаблоны в Django определяют внешний вид страниц вашего сайта. Вы можете создать шаблоны в формате HTML с использованием специальных шаблонных тегов Django, которые позволяют вставлять данные и логику в страницы.

8. Разработка URL-маршрутов: URL-маршрутизация в Django определяет, какие представления должны быть вызваны для определенных URL-адресов. Вы можете определить маршруты в файле urls.py вашего приложения, указав, какой view будет ассоциирован с каждым URL-адресом.

9. Тестирование: Django предоставляет мощные инструменты для тестирования вашего сайта. Вы можете писать автоматические тесты для проверки правильности работы вашего кода.

10. Развертывание сайта: После завершения разработки вы можете развернуть ваш сайт на сервере. Существуют различные способы развертывания Django-приложений, включая использование специализированных хостинг-провайдеров или самостоятельное конфигурирование сервера.

9. Создание статических файлов: Django позволяет использовать статические файлы, такие как изображения, CSS-стили и JavaScript, для оформления и функциональности вашего сайта. Вы можете сохранить статические файлы в папке static в каждом приложении и использовать их в шаблонах.

10. Тестирование и отладка: После того, как вы создали основную функциональность своего сайта, рекомендуется провести тестирование и отладку. Django предоставляет встроенные инструменты для тестирования, а также возможности отладки.

11. Развертывание сайта: После завершения разработки вы можете развернуть свой сайт на сервере. Django имеет множество возможностей для развертывания, включая интеграцию с популярными серверами, такими как Apache или Nginx.

Это лишь общий обзор процесса разработки сайта на Django. Конкретные шаги и детали могут варьироваться в зависимости от требований вашего проекта. Если вы новичок в Django, рекомендуется изучить документацию Django, которая содержит подробные инструкции и примеры кода.

* 1. Тестирование проекта на Django.

Django предоставляет встроенные инструменты для написания и запуска тестовых случаев и проверки работоспособности вашего приложения. Тестирование является важной частью разработки, поскольку оно позволяет проверить, что ваше приложение работает правильно и соответствует ожидаемым результатам.

В Django вы можете писать различные виды тестов, такие как модульные тесты, функциональные тесты и интеграционные тесты. Вот несколько шагов для того, чтобы начать писать тесты в Django:

1. Создайте файл tests.py в директории вашего приложения Django, если он еще не создан.

2. Импортируйте необходимые модули для написания тестов. Обычно это включает TestCase из модуля django.test, а также модели и функции вашего приложения, которые вы хотите протестировать.

3. Определите тестовые классы, наследующиеся от django.test.TestCase. Внутри классов вы можете определить методы, которые будут тестировать определенные аспекты вашего приложения.

4. В методах тестов используйте различные методы-ассерты, предоставляемые Django, для проверки ожидаемых результатов. Например, вы можете использовать assertEqual для проверки равенства значений.

5. Запустите тесты с помощью команды python manage.py test в корневой директории вашего проекта Django. Django выполнит все тестовые случаи, которые вы определили, и выведет результаты на консоль.

6. Анализируйте результаты тестов и исправляйте возникающие ошибки, если они есть. Повторно запустите тесты, чтобы убедиться, что все работает правильно.

Таким образом, вы можете писать и запускать тесты в Django, чтобы проверить, что ваше приложение работает корректно. Тестирование помогает выявить ошибки и обеспечивает надежность вашего приложения.

В Django существует несколько методов, которые помогают вам писать и запускать тесты. Ниже перечислены некоторые из них:

* Client: Класс Client позволяет создавать HTTP-запросы к вашему приложению без необходимости запуска сервера. Это особенно полезно для тестирования представлений и URL-шаблонов. Вы можете использовать методы, такие как get, post, put и delete, чтобы отправить запросы и проверить ожидаемые ответы.
* TestCase: Класс TestCase предоставляет набор функций для написания тестовых случаев. Вы можете использовать методы, такие как setUp и tearDown, чтобы настроить и очистить ресурсы перед и после выполнения каждого тестового случая. Внутри методов тестов вы можете вызывать различные методы-ассерты для проверки ожидаемых результатов.
* Client и TestCase вместе: Эти инструменты можно комбинировать для более сложного и точного тестирования. Вы можете использовать Client для отправки запросов и получения ответов, а затем проверять эти ответы внутри методов тестового случая TestCase.
* RequestFactory: Класс RequestFactory позволяет создавать объекты HttpRequest внутри тестов. Это полезно, когда вам нужно проверить какие-либо аспекты работы с запросами, но вы не хотите отправлять реальные HTTP-запросы.
* Client и RequestFactory вместе: Вы можете комбинировать эти инструменты, чтобы создавать запросы с помощью RequestFactory и отправлять их через Client, чтобы проверить ожидаемые ответы.

Тестирование в Django помогает вам убедиться, что ваше приложение работает корректно и соответствует вашим ожиданиям. Хорошо написанные тесты обеспечивают надежность и помогают вам быстро обнаруживать и исправлять ошибки. Тестирование проекта на Django - это процесс проверки и валидации функциональности и корректности работы вашего приложения. Это важная часть разработки, которая помогает убедиться, что ваше приложение работает правильно и соответствует вашим требованиям.

В Django есть встроенные инструменты, которые облегчают написание и запуск тестов. Следуя некоторым соглашениям и передовым практикам, вы можете создать надежные и устойчивые тесты для вашего проекта.

Вот некоторые основные шаги процесса тестирования проекта на Django:

1. Определение тестовой структуры: Перед тем как начать писать тесты, определите структуру вашей тестовой папки. Обычно вы создаете папку tests в корневом каталоге вашего проекта и разделяете тесты на несколько файлов или модулей.

2. Написание тестовых случаев: Django предоставляет класс TestCase, который служит основой для создания тестовых случаев. Вы можете создать наследующийся от TestCase класс и определить в нем методы, которые проверят ожидаемое поведение вашего приложения. Можно написать тесты для представлений, моделей, форм, URL-шаблонов и других элементов вашего проекта. Например, вы можете использовать методы-ассерты, такие как assertEqual, assertTrue и assertRaises, чтобы проверить ожидаемые значения, условия и исключения. Важно покрыть различные аспекты вашего приложения, чтобы убедиться, что все функциональные требования выполняются правильно.

3. Запуск тестов: Django предоставляет утилиту командной строки manage.py test, которая автоматически обнаруживает и запускает тесты в вашем проекте. Вы можете указать путь к папке с тестами или конкретный модуль для запуска. Утилита выводит результаты тестов и сообщает об ошибках или несоответствиях.

Вы также можете использовать флаги командной строки, такие как --failfast (досрочное завершение тестирования при первой ошибке), --keepdb (сохранение базы данных после выполнения тестов) и другие для настройки запуска тестов.

4. Покрытие кода тестами: Хорошей практикой является покрытие вашего кода тестами. Django предоставляет инструменты для измерения покрытия кода тестами, такие как coverage.py. Вы можете использовать эти инструменты, чтобы определить, насколько ваш код покрыт тестами, и найти потенциальные пробелы, которые требуют дополнительных тестов.

5. Интеграционное тестирование: Кроме модульных тестов, вы также можете проводить интеграционное тестирование для проверки взаимодействия различных компонентов вашего приложения. Django предоставляет инструменты, чтобы эмулировать HTTP-запросы и проверять ответы сервера.

Класс Client позволяет создавать запросы к вашему приложению без необходимости запуска сервера. Вы можете использовать методы, такие как get, post, put и delete, чтобы отправить запросы и проверить ожидаемые ответы.

Тестирование проекта на Django помогает вам гарантировать, что ваше приложение работает правильно и сохраняет свою функциональность при внесении изменений. Чем больше вы покрываете свой код тестами, тем более надежным и стабильным будет ваш проект.

* 1. Создание и наполнение базы данных.

Для создания базы данных с помощью Django вам понадобится выполнить несколько простых шагов:

1. Установите Django, если вы еще этого не сделали, с помощью команды установки pip: pip install django.

2. Создайте новый проект Django с помощью команды: django-admin startproject projectname. Здесь "projectname" - это имя вашего проекта, которое вы можете выбрать самостоятельно.

3. Перейдите в директорию проекта с помощью команды: cd projectname.

4. Создайте новое приложение Django с помощью команды: python manage.py startapp appname. Здесь "appname" - это имя вашего приложения, которое вы также можете выбрать самостоятельно.

5. Откройте файл settings.py в директории проекта и найдите переменную DATABASES. Здесь вы можете настроить параметры вашей базы данных, такие как тип базы данных, имя, пользователя, пароль и хост.

6. Укажите тип базы данных, который вы хотите использовать. Django поддерживает различные типы баз данных, такие как SQLite, MySQL, PostgreSQL и другие. Например, для использования SQLite вы должны установить sqlite3 в переменной ENGINE и указать путь к файлу базы данных в переменной NAME.

7. Запустите миграцию для создания таблиц базы данных на основе ваших моделей с помощью команды: python manage.py migrate.

Поздравляю! Вы только что создали базу данных с помощью Django. Теперь вы можете определить свои модели данных внутри вашего приложения и использовать СУБД для сохранения и извлечения данных.

Чтобы создать модели Django, необходимо выполнить следующие шаги:

1. Откройте файл models.py в директории вашего приложения Django.

2. Импортируйте класс models из библиотеки Django: from django.db import models.

3. Определите класс модели, наследующийся от models.Model. Этот класс будет представлять таблицу базы данных. Например:

class MyModel(models.Model):

field1 = models.CharField(max\_length=50)

field2 = models.IntegerField()

4. Внутри класса модели определите поля, которые будет содержать таблица. Django предоставляет различные типы полей, такие как CharField, IntegerField, DateField, ForeignKey, и другие. Установите необходимые атрибуты для каждого поля, такие как max\_length, default и другие, в зависимости от ваших требований. Например, CharField указывает на текстовое поле с максимальной длиной 50 символов, а IntegerField указывает на целочисленное поле.

5. После определения полей модели, запустите команду миграции для создания соответствующих таблиц в базе данных. Выполните команду: python manage.py makemigrations. Это создаст файл миграции для вашей модели.

6. Примените миграцию, чтобы создать таблицу в базе данных. Запустите команду: python manage.py migrate.

Теперь у вас есть модель Django, которая отображает таблицу в базе данных. Вы можете использовать эту модель для создания, чтения, обновления и удаления данных в вашем приложении Django.

1. Заключение.

В заключение необходимо включить следующее:

1. Краткие и ёмкие теоретические и практические выводы, которые были получены во время анализа теоретической базы и практического исследования.

2. Оценка проведённого исследования, описание его результатов.

3. Практическая значимость работы, рекомендации и планы на дальнейшие исследования.

4. Общий итог — достижение цели, выполнение задач, доказательство гипотезы.

5. Предложения по совершенствованию объекта исследования.

1. Список использованной литературы.
2. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. - СПб.: Питер, 2017.
3. Мейер Э., Уэйл Э. CSS: полный справочник, 4-е изд.: Пер. с англ. - СПб.: ООО «Диалектика», 2019.
4. Учурова С. А. Лексикология немецкого языка: конспект лекций. Lexikologie der deutschen Sprache : vorlesungsskripten / С. А. Учурова. – Екатеринбург : Изд-во Урал. ун-та, 2014.