Лекція 21
Створення серверних застосунків

Можна виділити 5 основних методів розробки

1. Ручна за допомогою HTML
2. За допомогою програмних засобів розробки сайтів
3. За допомогою інструментальних систем таких як CMS
4. З використанням фреймворків
5. На SaaS-платформах у CLOUD

21.1 Використання HTML та програмних систем для розробки

Ручна розробка за допомогою HTML

При появі стандарту HTML, цей метод був найпоширенішим. Основною програмою для розробки був Notepad. Але у цього методу є істотні недоліки. Цей спосіб досить трудомісткий. І до того ж зробити нормальний Web-сайт без CSS, JavaScript та інших мов програмування досить важко.

HTML5 – це специфікація мови розмітки, що використовується в створенні веб-сторінок. На відміну від попередніх версій це не просто специфікація мови для гіпертекстової розмітки, а набір різнопланових модулів – від HTML-елементів до відео-, аудіо-, векторної графіки SVG, растрової JavaScript-графіки Canvas, локальних баз даних і навіть різних API браузера.

Весь цей список модулів дозволяє HTML5 успішно конкурувати з технологіями Flash і Silverlight. Причому успішна конкуренція з Flash можлива ще й тому, що HTML5 потенційно набагато менше навантажує процесор комп'ютера, ніж Flash, не вимагає установки плагінів і оновлень, а значить, менш вразливий для хакерських атак.

Adobe Flash була технологією розробки веб-додатків та мультимедійного контенту, яка включала у себе серію інструментів та технологій для створення візуально привабливих, інтерактивних елементів веб-сторінок.

Однак, у 2020 році Adobe оголосила про припинення підтримки Flash та рекомендувала користувачам припинити використання цієї

технології у своїх проектах. Таким чином, у розробці серверних застосунків сьогодні Adobe Flash не використовують. Замість цього, розробники можуть використовувати інші технології, такі як HTML, CSS та JavaScript для створення інтерактивних елементів веб-сторінок та серверних застосунків.

Silverlight - це технологія розробки веб-додатків, яка була створена компанією Microsoft для створення багатомедійного контенту та веб-застосунків з високою інтерактивністю.

Silverlight була розроблена на базі технологій .NET Framework та Windows Presentation Foundation (WPF) та могла бути використана для створення веб-додатків, що працюють на різних платформах, включаючи Windows, MacOS та Linux.

Проте, в 2013 році Microsoft оголосила про припинення розвитку та підтримки Silverlight. Замість цього, компанія рекомендує використовувати HTML5 та інші відкриті веб-стандарти для створення інтерактивних веб-додатків та мультимедійного контенту.

Отже, як і у випадку з Adobe Flash, у розробці серверних застосунків Silverlight сьогодні не використовують і рекомендується замінити цю технологію на більш сучасні.

Фактично саме поява в HTML5 нових тегів <video> та <audio> робить його потенційним конкурентом існуючих технологій від Adobe і Microsoft.

HTML 5 вводить кілька нових елементів і атрибутів. Деякі з них технічно є еквівалентами і , але мають своє семантичне значення, наприклад <nav> (навігаційна панель) і <footer>. Ці теги мають полегшувати роботу пошукачам, а також обробку сайту з персонального комп'ютера або програмам-читачам. Інші елементи надають нову функціональність, такі як <audio> і <video>. Деякі застарілі елементи HTML 4, такі як <font> і , були видалені з HTML 5.

Крім визначення розмітки, в HTML 5 визначені application programming interface, API. Існуючі інтерфейси DOM (Document Object Model - «об'єктна модель документа») розширені, також були додані нові API:

• Малювання 2D-картинок в реальному часі;

• Контроль над відтворенням медіафайлів, який може використовуватися, наприклад, для синхронізації субтитрів з відео;

• Зберігання даних у браузері;

• Редагування;

• Drag-and-drop;

• Робота з мережею.

HTML 5-сумісні браузери дуже гнучкі під час обробки помилок, на відміну від XHTML. HTML 5 розроблений так, що не підтримують його браузери можуть спокійно ігнорувати елементи HTML 5. На відміну від четвертої, п'ята версія чітко прописує правила лексичного розбору, щоб різні браузери відображали один і той же результат в разі некоректного синтаксису.

Canvas (англ. Canvas - «полотно») - елемент HTML5, призначений для створення растрового двомірного зображення за допомогою скриптів, зазвичай на мові JavaScript. Початок відліку блоку знаходиться зліва зверху. Від нього і будується кожен елемент блоку. Використовується, як правило, для відтворення графіків для статей і ігрового поля в деяких браузерних іграх. Але також може використовуватися для вбудовування відео в сторінку і створення повноцінного плеєра. Використовується в WebGL – стандарті на базі OpenGL ES 2.0, що дозволяє розробникам вебконтенту вбудовувати до вебоглядачів, які підтримують HTML5, повноцінну 3D-графіку, не вдаючись до посередництва плагінів – для апаратного прискорення 3Dграфіки.

21.2 Розробка сайту за допомогою програмних засобів

Існує багато готових рішень, для більш швидкої і зручної розробки сайтів. Вони надають можливість генерувати html код, розробляти сайт у візуальному режимі і мають багато інших можливостей.

Виділимо декілька інструментальних систем для розробки HTML:

• *програми, що мають у своєму складі візуальні редактори (design-based editor)* – це засоби, які автоматично формують необхідний HTML-код, дозволяючи розробляти Web-сторінки в режимі WYSIWYG;

• *програми-редактори (code-based editors)*, які надають редактор і допоміжні засоби для автоматизації написання коду.

Розглянемо найбільш популярні design-based редактори:

• *Adobe DreamWeaver* – це один з кращих візуальних редакторів, що генерують HTML код. Він дозволяє працювати в декількох режимах одночасно, з HTML кодом або у візуальному режимі. Але основним недоліком є те, що програма генерує занадто "важкий" код, додаючи багато зайвого. Але, якщо знайомі з HTML, тоді текст HTML можна відредагувати. Ця програмна система випускалася до 2005 року компанією Macromedia, після чого була придбана фірмою Abode.

• *Microsoft FrontPage* – це простий в опануванні та зручний Web-редактор для проектування, підготовки і публікації Web-сайтів. Завдяки інтеграції з сімейством продуктів MS Office, звичного інтерфейсу і великої кількості шаблонів програма дозволяє швидко почати роботу навіть початківцям, які знайомі з основами роботи в MS Word. Проте FrontPage надає широкі функціональні можливості та різноманітні засоби оптимізації при колективній розробці. Вона дозволяє швидко створювати динамічні комплексні Web-вузли практично будь-якої складності.

Розглянемо популярні code-based редактори:

• *Adobe HomeSites* – це потужний пакет, до складу якого входить багато корисних функцій і підпрограм. Об'ємний дистрибутив редактора включає в себе, крім самого редактора, редактор TopStyle для редагування таблиць CSS, перевірку орфографії та багато іншого.

• *HotDog* – це цілком професійний редактор. Вбудована підтримка дуже широкого набору інструментів, що використовуються в Web-дизайні: HTML, CSS, JavaSript, VBScript, ASP, а також DOM - об'єктну модель документа, що використовується при програмуванні на VBScript і JavaScript. При цьому перевірка синтаксису цих інструментів може налаштовуватися в досить широких межах. Наприклад, HTML можна

перевіряти на відповідність версії 3.2, 4, або на "перегляді" тільки в Internet Explorer та інше.

• *AceHTML* - основні функціональні можливості - подібно HomeSite і FirstPage. Головною перевагою AceHTML є вбудована утиліта для перегляду графічних файлів у комп'ютері. У просторому вікні відображаються ескізи всіх картинок в директорії, а також їх параметри і розмір у пікселях.

21.3 Використання інструментальних систем CMS для створення динамічних Web-застосунків

Для створення динамічного сайту можливі два способи.

Перший спосіб – це написання власних програм, які відповідають за створення потрібних шаблонів і підтримують необхідні функції. При цьому створена система буде повністю відповідати потребам, проте можливо вимагатиме великих програмістських зусиль і часу.

Другий спосіб – це скористатися вже існуючими системами, які і називаються системами управління Web-контентом. Перевагою цього шляху є зменшення витрат часу і сил. До його недоліків можна віднести зниження гнучкості, надання недостатнього або надмірного набору можливостей.

Другий шлях є основним на цей час для створення складних, сучасних сайтів, порталів, Веб-додатків. Це метод з використанням CMS. Вікіпедія дає наступне визначення. CMS це система керування вмістом (контентом) (англ. Content management system, CMS) — комп'ютерна програма чи система, що використовується для забезпечення і організації сумісного процеса створення, редагування та управління текстовими і мультимедійними документами (вмісту чи контента). Звичайно цей вміст розглядається як не структуровані данні предметної задачі в протилежність структурованим даним, що звичайно знаходяться під керуванням СУБД. Звичайно, що встановлення CMS робиться вже на вибраному хостингу. При цьому як мінімум вимагається FTP доступ та дозвіл роботи MySQL.

Таким чином, відділення дизайну від контенту є головною відміною особливістю динамічних сайтів від статичних. На цій основі можливі подальші удосконалення структури сайту, такі як визначення різних призначених для користувача функцій і автоматизація бізнес-процесів, а саме головне, контроль контенту, що надходить на сайт.

22.3 Системи управління Web-контентом

Типова структура системи управління Web-контентом надана на рисунку 21.1.

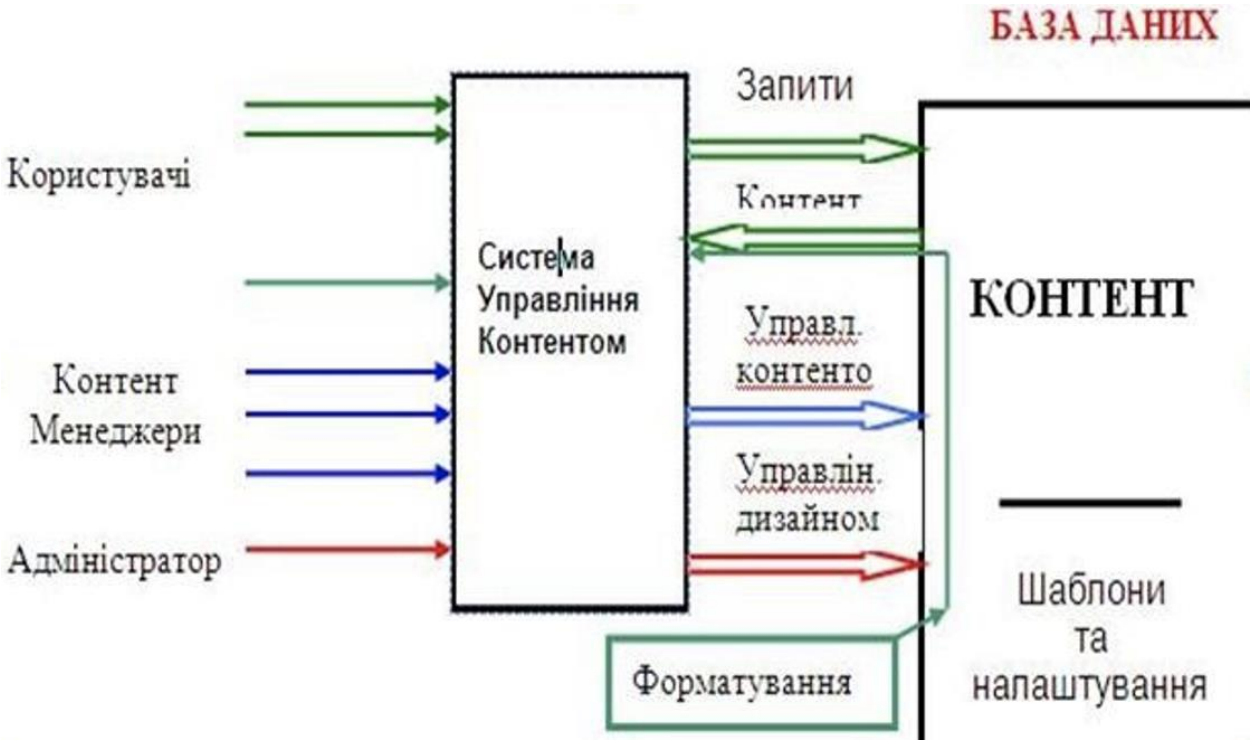


Рисунок 21.1 - Типова структура системи управління Web-контентом

Приклади CMS: WordPress, Joomla !, OpenCart та ін. За різними даними на WordPress працюють від 30% до 40% всіх сайтів, які використовують CMS. Інші CMS, які увійшли в топ 5 на різних сайтах: Joomla !, Drupal, Wix, Shopify, Squarespace.

На практиці CMS - це веб-додаток, в якому люди створюють і обслуговують сайти. Основна перевага CMS - щоб створити сайт і працювати з ним не обов'язково знати програмування.

Основні функції систем управління Web-контентом

Виробники систем управління Web-контентом одностайні лише в самому загальному визначенні того, що повинні надавати такі системи. Їх основні параметри можна згрупувати в три категорії:

1. Розробка контенту

Розробка контенту є одним з ключових компонентів всієї системи. Саме тут починається життєвий цикл будь-якого матеріалу що публікується на сайті. На цьому етапі відбувається створення, редагування та затвердження контенту, а роль системи полягає в автоматизації цих процесів. Завдання підтримки спільної роботи авторів, редакторів, програмістів та менеджерів повністю покладається на систему. Це завдання здійснюється завдяки розподілу контенту і дизайну. Всі компоненти сайту, включаючи шаблони і наповнення, зберігаються у визначених місцях сховища даних. Система ж автоматично звертається в потрібні місця сховища, дозволяючи великої кількості користувачів, які навіть не є технічними фахівцями, працювати над підготовкою контенту до публікації, включаючи перевірку його достовірності.

2. Управління сайтом

На цьому рівні відбувається розробка самого сайту, попередній перегляд і публікація підготовленого контенту. Тут розробляється зовнішній вигляд, готуються шаблони, розподіляються ролі користувачів і класифікація необхідної бізнес-інформації (наприклад, товари, ціни). Важливими компонентами цього рівня є служби що підтримують своєчасність надходження необхідного контенту.

3. Доставка контенту

Коли сайт повністю підготовлений до публікації, необхідні ресурси для динамічного формування Web-сторінок в залежності від виду конкретних користувачів. У зв'язку з цим, одним з важливих компонентів даного етапу є персоналізація або розподіл профілів, щоб кожен користувач отримував тільки ту інформацію, яка відповідає його ролі.

Треба відзначити, що хоч і не існує абсолютно однакових систем управління Web-контентом, експерти сходяться в одному. При розвитку

Web-технологій системи повинні будуть концентруватися більше на управлінні контентом, ніж на Web-паблішинг.

Дуже швидко CMS перетворилися із просто систем управління контентом в повнофункціональні системи управління сайтом. При цьому популярні і широко розповсюдженні скрипти, які окремо використовувалися на сайтах, були об'єднані загальною програмою під єдиним інтерфейсом. В багатьох випадках в інтегровані зручні автоматичні інсталятори і візуальні HTML редактори. В результаті отримали на 100% готовий до застосування повнофункційний сайт — портал. Інсталювати такий портал на сервер, встановити (вибрати) необхідні функціональні модулі (наприклад, модуль публікації статей з уже вбудованою можливістю додавання коментаріїв відвідувачами), наповнювати інформаційним контентом свій сайт уже може будь-яка людина без спеціальних знань.

Часто програмні можливості CMS дозволяють (в повністю автоматичному режимі) змінювати розташування виведення інформації на Web-сторінці, вибирати схеми кольорів, дизайн із шаблонів, що є в наявності.

В наш час за допомогою сучасних CMS, будь-яка людина, яка має бажання відкрити своє представництво в Інтернет, може без спеціальних знань, «титанічної» праці, особливих матеріальних витрат організувати свій якісний і багатофункційний сайт. Організації також отримали більш дешеву і мобільну можливість вести свій Інтернет-бізнес. Це ж стосується і рекламних агентств, значна кількість яких в наш час є розробниками своїх власних CMS.

В більшості випадків CMS будуються із програмного ядра — «движка» та модулів, що підключаються.

«Движок» CMS — це програма, яка підпорядкована певному алгоритму введення і виведення інформації. Цей алгоритм у кожної CMS різний, свій, в відповідності з ідеєю і ціллю розробника. Це дає можливість користувачу вибирати CMS, що найбільш підходить. В зв'язку з цим, зовнішній вигляд інформації що виводиться (наприклад, новин), в кожній CMS різний, індивідуальний і змінити його досить складно (або неможливо, без переробки, зрозуміло, самого «движка»),

тому якщо зовнішній вигляд контента початково не влаштовує, краще вибрати іншу CMS, ніж займатися його «підгонкою».

Універсальність і, багато в чому популярність CMS, залежить від кількості наявних модулів, можливості и простоти створення нових. Наявність, кількість, оновлення модулів є одним із критеріїв оцінки CMS. Модульна структура дозволяє розробникам охопити велике коло користувачів з різними вимогами до функційності порталу, програмістам — створювати свої модулі під конкретні задачі чи замовлення, а користувачам — отримати зручність і можливість мобільно реалізовувати різні свої ідеї.

Багато які CMS, крім функційних модулів, мають шаблону структуру. За допомогою шаблонів можна швидко змінити зовнішній вигляд всього порталу або окремих модулів. Крім того, наявність шаблоної структури припускає і дозволяє корекцію і створення своїх, індивідуальних шаблонів.

CMS з одного боку дає широкі можливості відвідувачам-користувачам, а з іншого - адміністраторам, які керують ресурсами, адмініструють інформацію, керують загальним виглядом сайту, спілкуються з відвідувачами і клієнтами. Більшість CMS забезпечені системою авторизації, і адміністратор може дозволяти, забороняти, обмежувати доступ до інформації, або частини інформації на своєму ресурсі окремим відвідувачам.

Багато які CMS, крім функційних модулів, мають шаблону структуру. За допомогою шаблонів можна швидко змінити зовнішній вигляд всього порталу або окремих модулів. Крім того, наявність шаблоної структури припускає і дозволяє корекцію і створення своїх, індивідуальних шаблонів.

4. Оформлення Веб-сайту

Художнє та дизайнерське оформлення сайтів беруть на себе так звані "шаблони". Це розроблене дизайнерське та художнє рішення сайту, що пропонується кожним CMS при його встановленні. Розробка нового свого шаблону досить трудомісткий процес. Використання стандартних шаблонів значно полегшує створення сайтів, але зменшує їх індивідуальність.

Значно полегшують роботу користувачів з CMS при введенні та редагуванні інформації вбудовані в них редактори (їх часто називають WYSIWYG). Не всі редактори, що пропонуються, бездоганні - можливі і помилки, і неадекватна їх робота. Виправлення часто можливе на рівні HTML і ця можливість надається цими редакторами. Якщо проаналізувати інші можливості WYSIWYG, то можна виділити наступні блоки функцій: - редагування і форматування (значно менші ніж у MS Word), уставлення гіперпосилань, уставлення малюнків, приєднання файлів та декілька інших.

Загальною практичною порадою є грамотне використання буфера вирізаного зображення (Ctrl+C, Ctrl+V).

Як і в усіх потужних програмних розробках в CMS додаються модулі, що значно розширюють їх функціональні можливості. Можливе включення Форумів, Гостьових книг, Форм та іншого. У деяких CMS можливе створення Блогів і на їх основі внутрішніх самостійних сайтів.

Поняття і сутність системи управління контентом.

Сьогодні виділяють два види сайтів: статичні і динамічні. Статичні сайти створюються відносно просто і дешево, але трудомісткі і дорогі в супроводі. Замовні динамічні сайти складні і дорогі в розробці, але при вдалій постановці завдання можуть бути достатньо дешевими у супроводі.

Системи управління контентом, або движки, пропонують компроміс між цими двома варіантами. За рахунок обмежень, накладених на логічну структуру контенту, зовнішній вигляд (дизайн) і функціональні можливості створюваних динамічних сайтів, такі системи дозволяють радикально знизити трудомісткість розробки та підтримки.

Скрипти CMS розташовуються на самому сайті і щоб скористатися ними потрібно просто зайти на спеціальну сторінку сайту, захищену паролем. Відповідно все, що потрібно для оновлення сайту з встановленою системою, - це наявність підключеного до Інтернету комп'ютера і браузера. Оновлення сайту за допомогою движка відбувається швидше і простіше, ніж редагування коду, і може здійснюватися особами, абсолютно не знайомими з HTML і іншими

мовами розмітки. Багато сучасних CMS настільки автоматизовані, що при роботі з системою не потрібен фахівець.

Контент – це будь-яка інформація, графічна чи текстова, розміщена на сервері.

Система управління контентом:

1) це система управління інформацією на сайті;

2) це система, за допомогою якої власник може керувати змістом свого ресурсу, не вдаючись при цьому до послуг професійних html-верстальників і програмістів;

3) це спеціалізоване програмне забезпечення для автоматизованої розробки і підтримки динамічних сайтів.

Такі системи дозволяють розробляти і підтримувати сайти, не вимагаючи при цьому знання HTML.

Основна ідея систем управління контентом - поділ дизайну сайту і його змісту. При створенні сайту за допомогою такої системи розробляються кілька шаблонів сторінок, в яких згодом розміщується інформація, тобто система містить шаблони сторінок і сховище (базу даних) інформації, з якої формується інформаційне наповнення сайту.

Незважаючи на те, що всі системи управління контентом спроектовані по-різному, вони призначені для однієї мети – оновлення змісту сайту, і важливу роль тут відіграє модуль документообігу, що входить в систему управління сайтом.

У центрі системи знаходиться база даних, в якій зберігається весь контент, шаблони сторінок і інформація про управління користувачами/продуктами.

Переваги CMS:

- можна створити сайт самостійно і за короткий проміжок часу;
- не потрібно розбиратися в дизайні і програмуванні;
- розробка сайту буде коштувати дешевше;
- зручно керувати вмістом сайту.

Недоліки CMS:

- потрібно стежити за оновленнями і сумісністю нових версій з доповненнями;

- продуктивність зазвичай знижується, якщо на сайті багато доповнень;
- не весь функціонал можна реалізувати;
- не підходять для нетипових завдань.

21.4 Класифікація систем управління контентом.

В основу класифікації можуть бути покладені наступні критерії:

користувацький сервіс - наявність тих чи інших функцій і модулів, зрозумілість і доступність користувачеві;

технологічність - використання технологій, що дозволяють підвищити пропускну здатність і надійність системи;

сумісність (апаратна і програмна) - можливість функціонування системи на різних платформах, сумісність з СУБД, можливість підключення додаткових модулів;

масштабованість - можливість розвитку та нарощування системи.

На основі вищевикладених критеріїв виділяють: *просту, шаблонну, професійну та універсальну системи управління контентом.*

Проста система управління контентом

Інтерфейс користувача контент-системи збирається з програмних модулів, набір яких визначається в індивідуальному порядку для кожного окремого проекту. Модулі один раз налаштовуються розробником, чим жорстко закріплюється структура проекту. Від користувача системи потрібне знання розмітки HTML. Система функціонує на основі динамічного формування сторінок (щоразу при запиті користувача) і обмеженій пропускній спроможності (в залежності від апаратного забезпечення) - до 3 000-5 000 відвідувачів на добу.

Проста система сумісна з певними платформами і типами СУБД. Спроба впровадження додаткових модулів, залежно від технологій що використовуються, може привести до повної переробки проекту. Масштабованість досягається шляхом перекладу сайту на нову, більш пристосовану до поточних завдань, версію системи управління контентом.

Шаблонна система управління контентом

Інтерфейс користувача системи представлений єдиним модулем або набором модулів з жорстко закріпленою структурою сайту. Шаблонна система функціонує на основі динамічного формування сторінок або використання кешування даних. Залежно від методів формування сторінок - від 5 000 до 50 000 відвідувачів на добу. Система сумісна з певними платформами і типами СУБД. Спроба впровадження додаткових модулів, також, як і в попередньому випадку, обмежена. Масштабованість відбувається, як і в простій системі управління контентом.

Професійна система управління контентом

Система представлена інтуїтивно зрозумілим для користувача інтерфейсом і розширеними можливостями редагування. На основі таких систем управління контентом створюються самі різноманітні сайти. Надається можливість підключення додаткових модулів, як від розробника, так і прикладного ПЗ. Формовані динамічні сторінки кешуються, що веде до безмежної пропускної здатності, яка залежить тільки від апаратного забезпечення. Система сумісна з різними програмно-апаратними платформами. Рівень пропонованої в системі масштабованості дозволяє підключати додаткові модулі без порушення структури та ідеології керування веб-ресурсом.

Універсальна система управління контентом

Користувацький сервіс представлений передовими засобами управління контентом, система надає засоби для розробки нових сервісів і можливостей. За технічними характеристиками дана система повністю пристосована до внутрішньокорпоративного використання в зв'язці з ERP-пакетами, що забезпечено наявністю сертифікованої системи безпеки - розмежування прав доступу до контент-системи на внутрішньокорпоративному рівні. У процесі функціонування системи проводиться кешування формованих динамічних сторінок і при цьому пропускна здатність - не обмежена.

Система сумісна з різними програмно-апаратними платформами. Можливості масштабування дозволяють підключати додаткові модулі, що розширюють функціональність ресурсу, у тому числі готові модулі

інтеграції з внутрішньокорпоративними системами. Можливе розширення за рахунок кластеризації.

21.5 Розповсюджені CMS

WordPress дозволяє створювати сайти будь-якого типу, так як має гарну розширюваність. Він простий в установці, дозволяє легко змінювати шаблони і теми оформлення, має високі SEO-характеристики (Search Engine Optimization тобто пошукова оптимізація).

Drupal – безкоштовна CMS, зі значною SEO-адекватністю, безпекою та розширюваністю. Движок широко використовується при побудові сайтів різного призначення. Його використано при розробці сайтів сенату США та потужного сайту рейтингування Університетів світу Webometrics

Joomla – це система, яка має здатність до розширюваності, захищеністю і відкритістю, має великий набір шаблонів. Недоліком є високе навантаження на сервер та невисока швидкість роботи. Проблеми з SEO-адекватністю.

Bitrix – це професійна платна CMS, яку використовують під час створення корпоративних сайтів. Досить складна в налаштуванні і редагуванні, сильно навантажує сервер, є проблеми з дублюванням сторінок і безпекою.

DataLifeEngine – це платна CMS, призначена для розробки інформаційних і новинних сайтів. Цей движок має багато вразливостей, тому часто піддається злому.

21.6 Використання фреймворків

Фреймворк це програмний продукт, який є основою для створення сайтів, але він не має готових рішень для побудови сайтів, не має рішень для виконання певних функцій. Це більш низький рівень ніж CMS. Розробники на фреймворках створюють і інтерфейсну частину, і базу даних, і алгортми та програмні рішення проблемно орієнтованої

частини і скоріше не сайту , а Веб додатку. Створюючи також його адміністративний інтерфейс.

Фреймворк (в інформаційних системах) – це структура програмної системи, що полегшує розробку і об'єднання різних компонентів великого програмного проекту. На відміну від бібліотек, які об'єднують набір підпрограм близької функціональності, фреймворк містить в собі велику кількість різних за призначенням бібліотек. Вживається також слово каркас, а деякі автори використовують його в якості основного, в тому числі не базуючись взагалі на англомовному аналогу. Можна також говорити про каркасний підход як про підхід до побудови програм, де будь-яка конфігурація програми будується з двох частин: перша, постійна частина - каркас, незалежний від конфігурації до конфігурації і несе в собі гнізда, в яких розміщується друга, змінна частина - змінні модулі (або точки розширення).

Існують наступні види фреймворків:

Фреймворки програмної системи;

Фреймворки додатків;

Фреймворки концептуальної моделі;

Фреймворк програмної системи - це каркас системи або підсистеми. Він може включати допоміжні програми, мови сценаріїв, все, що полегшує розробку і об'єднання різних компонентів. Від бібліотеки він відрізняється виконанням коду, який написаний для нього, але не виконується сам. До цього виду фреймворків відносяться і фреймворки для WEB.

Фреймворк додатку має стандартну структуру. З ростом необхідності в графічних інтерфейсах користувача з'явилася і необхідність у фреймворках додатків. З їх допомогою простіше створювати засоби для створення графічних інтерфейсів автоматично. Для створення фреймворку додатків використовують об'єктно-орієнтоване програмування. Перший такий Фреймворк написала компанія Apple для Macintosh. Спочатку він був створений за допомогою Паскаль, потім же перероблений в C ++.

Фреймворк концептуальної моделі – це абстрактне поняття даної структури для визначення способів вирішення конкретної проблеми.

WEB фреймворки – це каркас, призначений для створення динамічних веб-сайтів, мережевих додатків, сервісів або ресурсів. Він спрощує розробку і позбавляє від необхідності написання рутинного коду. Багато фреймворків спрощують доступ до баз даних, розробку інтерфейсу, а також зменшують дублювання коду .

Існує п'ять типів веб-фреймворків: Request-based, Component-based, Hybrid, Meta and RIA-based.

Request-based: фреймворки, які безпосередньо обробляють вхідні запити. Збереження стану відбувається за рахунок серверних сесій. Приклади: Django, Ruby на Rails, Struts, Grails.

Component-based: фреймворки, які абстрагують обробку запитів всередині стандартних компонентів і самостійно стежать за станом. Своєю поведінкою дані каркаси нагадують стандартні програмні графічні інтерфейси.. Приклади: JSF, Tapestry, Wicket.

Hybrid-based: фреймворки, які комбінують Request-based та Component-based фреймворки, беручи під свій контроль всі дані і логічний потік в заснованої на запиті моделі. Розробники мають повний контроль над URL, формами, параметрами, cookies і pathinfos. Однак замість того, щоб відобразити дії і контролери безпосередньо до запиту, гібридні фреймворки забезпечують об'єктну модель компонентів, яка поводиться тотожно в багатьох різних ситуаціях, таких як окремі сторінки, перервані запити, подібні порталу фрагменти сторінок і інтегровані віджети. Компоненти можуть розподілятися окремо і ефективно інтегруватися в інші проекти. Приклади: RIFE.

Meta -based: у фреймворків є ряд базових інтерфейсів для загального обслуговування і основу яка легко розширюється, для інтегрування компонентів і служб. Приклад: Keel.

RIA-based: фреймворки для розробки Rich Internet Applications (RIA). Служать для розробки повноцінних додатків, що запускаються всередині браузера. Приклад: Flex.

Найбільш поширеними є Request-based і Component-based веб-фреймворки. Більшість WEB-фреймворків побудовані на архітектірі MVC.Model View Controller (MVC, «модель- представлення-контролер», «модель-вид- контролер») – схема використання декількох шаблонів

проектування, за допомогою яких модель додатки, інтерфейс і взаємодія з користувачем розділені на три окремі компоненти таким чином, щоб модифікація одного з компонентів чинила мінімальний вплив на інші . Дана схема проектування часто використовується для побудови архітектурного каркаса, коли переходять від теорії до реалізації в конкретній предметної області.

Архітектурний шаблон Модель-Вид-Контролер (MVC) поділяє програму на три частини. У тріаді до обов'язків компоненту Модель (Model) входить зберігання даних і забезпечення інтерфейсу до них. Вигляд (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача, і повідомляє про зміни компоненту Модель. Така внутрішня структура в цілому поділяє систему на самостійні частини і розподіляє відповідальність між різними компонентами.

MVC поділяє цю частину системи на три самостійні частини: введення даних, компонент обробки даних і виведення інформації. Модель, як вже було відмічено, інкапсулює ядро даних і основний функціонал з їх обробки. Також компонент Модель не залежить від процесу введення або виведення даних. Компонент виводу Вигляд може мати декілька взаємопов'язаних областей, наприклад, різні таблиці і поля форм, в яких відображається інформація. У функції Контролера входить моніторинг за подіями, що виникають в результаті дій користувача (зміна положення курсора миші, натиснення кнопки або введення даних в текстове поле).

Зареєстровані події транслюються в різні запити, що спрямовуються компонентам Моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач через Контролер внесе зміни до Моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

21.7 SaaS-платформи

SaaS-платформи для створення сайтів - це можливість запустити досить простий веб-проект дуже швидко і досить дешево (ще і на умовах оренди і без необхідності хостінгу). Щось подібне до Гугл сайтів, але зі значно більшими можливостями. Рішення підходить для простих сайтів, тимчасових проектів і для перевірки бізнес-ідей. SaaS-платформи, як і CMS, бувають специфічними (наприклад, тільки для інтернет-магазинів) і універсальними (для всіх типових видів сайтів). Цей метод зараз активно розвивається та використовується.

SaaS-рішення мають свої переваги і недоліки, тому не кожен проект може бути реалізований подібним способом. На SaaS має сенс створювати прості сайти, які не особливо вимогливі до дизайну і ні відразу, ні в перспективі не зажадають яких-небудь доопрацювань логіки роботи та навігації. SaaS відмінно підходить для запуску проектів, мета яких полягає в перевірці на практиці бізнес-гіпотез. Можна і для створення простих проектів, вимога до яких одне - «потрібен сайт», а також для створення «заглушок» при розробці серйозних проектів. Хоча простий сайт доцільно створювати, наприклад, за допомогою засобів Гугл сайт.

Алгоритм створення сайтів на SaaS-платформах

Крок 1 Зареєструватися;

Крок 2 Вибрати тип сайту (наприклад, сайт-візитка, інформативний сайт або інтернет-магазин)

Крок 3 Вибрати потрібні модулі (новини, каталог, форум та інше);

Крок 4 Вибрати шаблон дизайну зі переліку шаблонів;

Крок 5 Ввести необхідну інформацію в систему управління контентом;

Крок 6 Ввести контент

SaaS-рішення входить все необхідне для повноцінної роботи проекту - не треба вибирати, встановлювати і налаштовувати CMS, не треба замовляти хостинг і налаштовувати сервер, а в подальшому не треба займатися технічним супроводом проекту. Все це робить за вас SaaS-рішення

Недоліки SaaS-платформ для створення сайтів

Фактично це шаблонний дизайн - оформлення сайтів на SaaS-платформах проводиться за готовими шаблонами (часто не дуже високої якості), які можна тільки «розфарбувати» і на деяких платформах можна окремі блоки місцями поміняти. Для сайтів, до яких пред'являються вимоги до оформлення, такі рішення не підходять.

Рамки функціональних можливостей - якщо платформа «не має якихось можливостей», то це ніяк не виправити. Програмний продукт типовий і його налаштування під індивідуальні побажання обмежена. Якщо проект відразу або в перспективі повинен вирішувати специфічні завдання і гнучко налаштовуватися, то SaaS-платформа для його розробки не підходить.

Приклади SaaS-платформ

UMI, WIX, InSales, Shopify, Setup, uCoz - деякі з цих платформ специфічні (тільки для простих сайтів або тільки для інтернет-магазинів), а деякі - досить універсальні. Якщо ваш проект не має ніяких істотних вимог до дизайну і до функціональності, то має сенс звернути увагу на ці SaaS-рішення. В іншому випадку, вибирайте іншу платформу для створення сайтів

21.8 . Створення серверних додатків за допомогою Node.js

Розглянемо корисні посилання

1 загальні відомості про середовище Node.js

https://www.javatpoint.com/nodejs-tutorial

2 робота з MySQL у середовищі Node.js

https://www.javatpoint.com/nodejs-mysql-create-connection

Node.js є середовищем виконання JavaScript, яке можна використовувати для розробки серверних додатків. Коли мова йде про роботу з MySQL (або іншою реляційною базою даних), Node.js можна використовувати як сервер, який взаємодіє з базою даних.

Одна з головних переваг Node.js полягає в його асинхронному підході до програмування. Це дозволяє розробникам створювати ефективні серверні додатки, які можуть обслуговувати багато запитів одночасно. Крім того, Node.js має багату екосистему модулів, які дозволяють легко підключати різноманітні функції до додатків.

Під час роботи з MySQL, Node.js можна використовувати бібліотеку, такою як "mysql", щоб взаємодіяти з базою даних. Ця бібліотека дозволяє виконувати запити до бази даних, отримувати та зберігати дані, а також контролювати транзакції.

Node.js є потужним інструментом для розробки серверних додатків, що взаємодіють з MySQL. Використання Node.js дозволяє розробникам створювати швидкі, ефективні та масштабовані додатки. JavaScript можна використовувати для створення запитів до бази даних MySQL. Node.js має ряд бібліотек, таких як "mysql", "sequelize", "knex" та інші, які дозволяють зв'язуватись з базою даних MySQL і виконувати різноманітні операції з даними, такі як додавання, оновлення, видалення та вибірка даних з бази даних.

Бібліотека "mysql" є однією з найпоширеніших для взаємодії з базою даних MySQL з Node.js. Вона дозволяє виконувати SQL-запити з Node.js, з'єднуватись з базою даних, взаємодіяти з таблицями та рядками даних, контролювати транзакції та інші операції.

Окрім того, деякі ORM-бібліотеки, такі як "sequelize" та "knex", дозволяють розробникам використовувати JavaScript-синтаксис для створення запитів до бази даних. ORM-бібліотеки дозволяють використовувати моделі даних замість написання SQL-запитів вручну, що спрощує процес взаємодії з базою даних та зменшує ризик помилок.

21.8.1 Можливості, які надає бібліотека "mysql"

Бібліотека "mysql" є однією з найпоширеніших бібліотек для взаємодії з базою даних MySQL з Node.js. Вона надає розробникам можливості для підключення до бази даних MySQL, виконання запитів до бази даних та обробки результатів запитів. Наведемо можливості, які надає бібліотека "mysql":

1 Підключення до бази даних MySQL: бібліотека "mysql" дозволяє встановити з'єднання з базою даних MySQL з Node.js, використовуючи відповідні параметри, такі як адреса сервера бази даних, ім'я користувача та пароль.

2 Виконання SQL-запитів: бібліотека "mysql" дозволяє виконувати SQL-запити до бази даних MySQL, такі як SELECT, INSERT, UPDATE та DELETE, використовуючи методи, такі як query та execute. Виконання запитів можливе як за використання параметрів, або без використання параметрів.

Наведемо приклад коду для виконання SQL-запитів за допомогою бібліотеки "mysql" середовища Node.js:

```
const mysql = require('mysql');

// створення підключення до бази даних
const connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'username',
  password : 'password',
  database : 'database_name'
});

// виконання SELECT запиту
connection.query('SELECT * FROM users', function (error, results, fields) {
  if (error) throw error;
  console.log('The solution is: ', results);
});

// виконання INSERT запиту
const user = { name: 'John', email: 'john@example.com' };
connection.query('INSERT INTO users SET ?', user, function (error, results,
fields) {
  if (error) throw error;
```

```
  console.log('User added to database with ID:', results.insertId);
});

// закриття підключення до бази даних
connection.end();
```

```
const mysql = require('mysql');

// створення підключення до бази даних
const connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'username',
  password : 'password',
  database : 'database_name'
});

// виконання SELECT запиту
connection.query('SELECT * FROM users', function (error, results, fields) {
  if (error) throw error;
  console.log('The solution is: ', results);
});

// виконання INSERT запиту
const user = { name: 'John', email: 'john@example.com' };
connection.query('INSERT INTO users SET ?', user, function (error, results,
  if (error) throw error;
  console.log('User added to database with ID:', results.insertId);
});
```

```
// закриття підключення до бази даних
connection.end();
```

У цьому прикладі створюється з'єднання з базою даних MySQL з використанням параметрів підключення. Потім виконуються два SQL-запити - SELECT і INSERT - за допомогою методу query. Результати запиту можна обробити у функції зворотнього виклику. Нарешті, після виконання запитів підключення закривається за допомогою методу end.

Зауваження. Наведений вище код призначений лише для навчальних цілей і потребує додаткової обробки помилок та забезпечення безпеки даних, таких як валідація введених даних перед виконанням запитів.

3 Обробка результатів запитів: бібліотека "mysql" дозволяє отримати результати SQL-запиту до бази даних MySQL у вигляді JavaScript-об'єктів або масивів, що дозволяє легко обробляти ці дані в Node.js.

4 Управління транзакціями: бібліотека "mysql" дозволяє управляти транзакціями, які дозволяють здійснювати групу зв'язаних запитів як одну транзакцію. Це дозволяє забезпечити цілісність даних та запобігти помилкам у випадку, якщо один із запитів не виконується.

5 Підтримка пулів з'єднань: бібліотека "mysql" дозволяє використовувати пули з'єднань для оптимізації взаємодії з базою даних MySQL і знизити витрати ресурсів.

Пул з'єднань працює наступним чином: коли програма потребує доступу до бази даних, замість того, щоб кожного разу створювати нове з'єднання з базою даних, бібліотека "mysql" перевіряє, чи є вільні з'єднання в пулі. Якщо є, вона використовує одне з них для взаємодії з базою даних. Якщо вільних з'єднань немає, бібліотека "mysql" автоматично створює нове з'єднання та додає його до пулу, а потім використовує його для взаємодії з базою даних.

Підтримка пулів з'єднань дозволяє ефективніше використовувати ресурси сервера і зменшити кількість часу, потрібного для створення та знищення з'єднань з базою даних. Крім того, це може зменшити навантаження на сервер бази даних, оскільки одне і те ж з'єднання може використовуватися для взаємодії з базою даних багатьма запитами з різних клієнтів.

Отже, підтримка пулів з'єднань є важливою можливістю бібліотеки "mysql" для оптимізації взаємодії з базою даних MySQL, що дозволяє знизити витрати ресурсів та поліпшити продуктивність системи.

21.8.2 Можливості, які надає бібліотека "sequelize"

Бібліотека "sequelize" є однією з ORM-бібліотек для роботи з базами даних в середовищі Node.js. Наведемо можливості, які надає бібліотека "sequelize":

1 Підтримка різних типів баз даних: бібліотека "sequelize" підтримує роботу з різними базами даних, такими як MySQL, PostgreSQL, SQLite та MSSQL, що дозволяє розробникам легко переносити свій код між різними системами управління базами даних.

2 Підтримка ORM-функцій: бібліотека "sequelize" дозволяє використовувати ORM-функції для взаємодії з базою даних. Наприклад, ви можете використовувати методи create, findAll, update та destroy для створення, вибірки, оновлення та видалення даних з бази даних.

3 Міграції баз даних: бібліотека "sequelize" дозволяє легко керувати міграціями баз даних, що дозволяє розробникам змінювати структуру бази даних у відповідності з новими вимогами без ризику втрати даних.

4 Підтримка транзакцій: бібліотека "sequelize" дозволяє використовувати транзакції, які дозволяють здійснювати групу зв'язаних запитів як одну транзакцію. Це дозволяє забезпечити цілісність даних та запобігти помилкам у випадку, якщо один із запитів не виконується.

5 Підтримка відносин між таблицями: бібліотека "sequelize" дозволяє легко визначати та керувати відносинами між таблицями бази даних, такими як один до одного, один до багатьох багато до багатьох.

6 Підтримка пулів з'єднань: бібліотека "sequelize" підтримує пули з'єднань, що дозволяє оптимізувати взаємодію з базою даних і знизити витрати ресурсів.

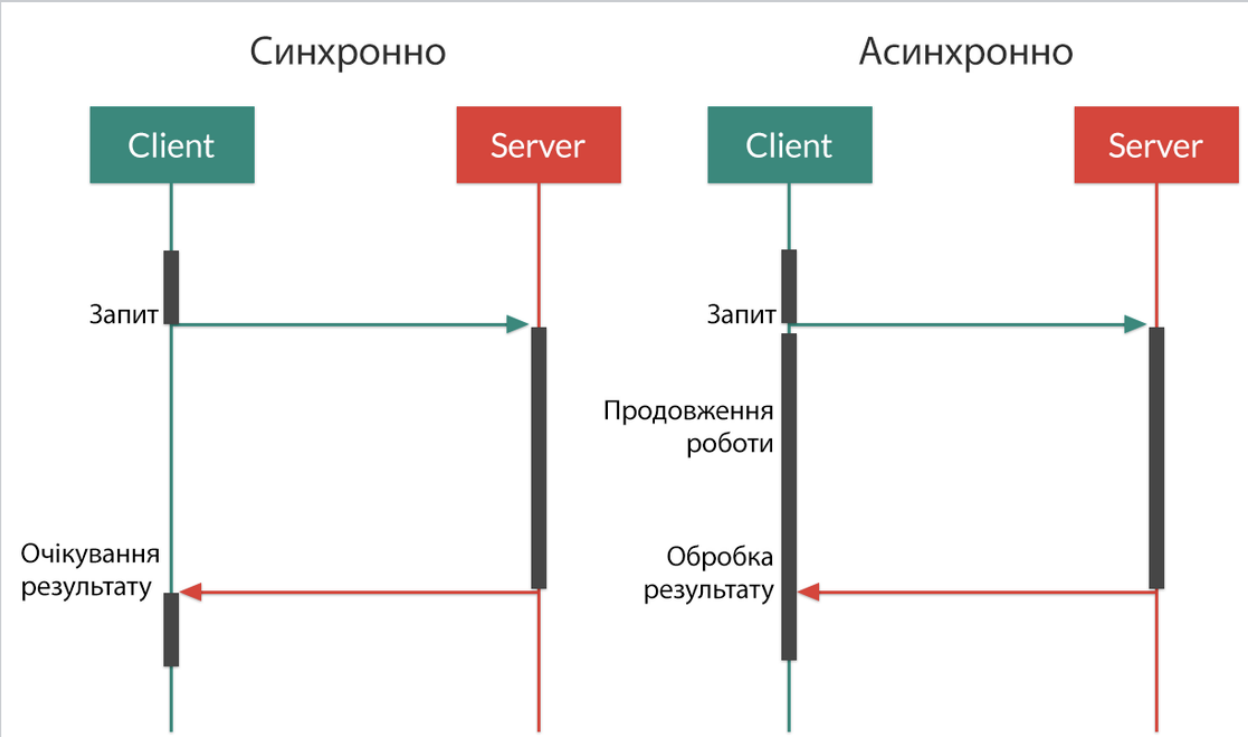
7 Підтримка промісів та асинхронного програмування: бібліотека "sequelize" підтримує проміси та асинхронне програмування, що дозволяє легко обробляти результати запитів та керувати взаємодією з базою даних.

8 Підтримка транзакцій збереження точки відновлення: бібліотека "sequelize" дозволяє зберігати точки відновлення транзакцій, що дозволяє відновлювати базу даних до попереднього стану в разі виникнення помилок.

9 Підтримка кешування запитів: бібліотека "sequelize" підтримує кешування запитів, що дозволяє збільшити продуктивність системи та зменшити навантаження на базу даних.

10 Підтримка оптимізації запитів: бібліотека "sequelize" має можливості для оптимізації SQL-запитів до бази даних, що дозволяє зменшити час виконання запитів та покращити продуктивність системи.

Проміси (Promises) – це спосіб обробки асинхронних операцій в JavaScript. Вони дозволяють виконувати асинхронний код відповідним чином і обробляти результати операцій, які ще не завершилися.



Проміси є об'єктами, які представляють поточний стан асинхронного процесу і обіцяють повернути результат асинхронної операції. Проміси мають два стани: "виконується" (pending) та "виконано" (fulfilled) або "відхилено" (rejected).

Коли ви створюєте проміс, він знаходиться в стані "виконується". Після виконання асинхронної операції проміс може перейти у стан "виконано" або "відхилено", залежно від результату операції. Якщо операція успішна, проміс переходить у стан "виконано" і повертає результат. Якщо операція не вдалася, проміс переходить у стан "відхилено" і повертає помилку.

Для обробки результатів промісів використовують методи then() і catch(). Метод then() виконується, якщо проміс перейшов у стан "виконано", тоді як метод catch() виконується, якщо проміс перейшов у стан "відхилено". Обидва методи приймають функції зворотнього виклику, які можуть обробляти результати асинхронної операції.

Наведемо приклад коду, який демонструє використання промісів:

```
function getData() {
  return new Promise(function(resolve, reject) {
    // Відправляємо запит на сервер за даними
    fetch('https://jsonplaceholder.typicode.com/todos/1')
      .then(function(response) {
        // Якщо запит успішний, повертаємо відповідь у форматі JSON
        if (response.ok) {
          return response.json();
        }
        // Якщо запит не вдається, відхиляємо проміс з помилкою
        throw new Error('Network response was not ok.');
      })
      .then(function(data) {
        // Якщо отримали дані, успішно виконуємо проміс з результатом
        resolve(data);
      })
      .catch(function(error) {
        // Якщо сталася помилка, відхиляємо проміс з помилкою
        reject(error);
      });
  });
}
```

```
// Викликаємо функцію для отримання даних та обробки результату
getData()
  .then(function(data) {
    // Обробляємо отримані дані
    console.log(data);
  })
  .catch(function(error) {
    // Обробляємо помилки
    console.error(error);
  });
```

```
function getData() {
  return new Promise(function(resolve, reject) {
    // Відправляємо запит на сервер за даними
    fetch('https://jsonplaceholder.typicode.com/todos/1')
      .then(function(response) {
        // Якщо запит успішний, повертаємо відповідь у форматі JSON
        if (response.ok) {
          return response.json();
        }
        // Якщо запит не вдається, відхиляємо проміс з помилкою
        throw new Error('Network response was not ok.');
      })
      .then(function(data) {
        // Якщо отримали дані, успішно виконуємо проміс з результатом
        resolve(data);
      })
      .catch(function(error) {
        // Якщо сталася помилка, відхиляємо проміс з помилкою
        reject(error);
      });
  });
}
```

```
// Викликаємо функцію для отримання даних та обробки результату
getData()
  .then(function(data) {
    // Обробляємо отримані дані
    console.log(data);
  })
  .catch(function(error) {
    // Обробляємо помилки
    console.error(error);
  });
```

У цьому прикладі функція getData() повертає новий проміс, який отримує дані з сервера за допомогою методу fetch(). Якщо отримання даних успішне, проміс виконується з отриманими даними, викликаючи функцію resolve(). Якщо сталася помилка, проміс відхиляється з помилкою, викликаючи функцію reject().

Після того, як функція getData() повертає проміс, ми можемо використовувати методи then() і catch() для обробки результатів. Якщо проміс виконується успішно, метод then() виконується з отриманими даними. Якщо проміс відхиляється, метод catch() виконується з помилкою.

Проміс відхиляється, коли асинхронна операція, на яку був створений проміс, не вдалася. Це може статися, наприклад, якщо запит до сервера повернув помилку або якщо код виконувався з помилкою.

Якщо проміс відхиляється, він переходить у стан "відхилено" (rejected) і повертає помилку. Обробка помилок здійснюється за допомогою методу catch(), який приймає функцію зворотного виклику і обробляє помилку. Якщо не використовувати метод catch(), помилка може вивестися в консоль браузера або викликати зупинку виконання коду.

21.8.3 Можливості, які надає бібліотека "knex"

Бібліотека knex є SQL-запитувальною бібліотекою для Node.js, яка дозволяє працювати з реляційними базами даних за допомогою JavaScript. Наведемо деякі з можливостей, які надає бібліотека knex:

1 Підтримка різних СУБД: knex дозволяє працювати з різними реляційними базами даних, такими як MySQL, PostgreSQL, SQLite та Oracle.

2 Створення запитів за допомогою JavaScript: knex дозволяє створювати запити до бази даних з використанням звичайних конструкцій JavaScript, що полегшує роботу з ними та зменшує ризик помилок.

3 Підтримка транзакцій: knex дозволяє використовувати транзакції, які дозволяють забезпечити консистентність даних (дані знаходяться коректному стані та відповідають всім правилам та обмеженням даної БД) в базі в разі незавершених операцій.

4 Підтримка міграцій: knex дозволяє виконувати міграції баз даних, які допомагають зберігати схему бази даних під контролем версій, змінювати її безпечно та забезпечувати консистентність даних.

5 Підтримка пулів з'єднань: knex дозволяє створювати пули з'єднань з базою даних, що дозволяє зменшити кількість підключень до бази даних та забезпечити більш ефективне використання ресурсів.

6 Підтримка більш високорівневих операцій: knex надає можливість виконувати більш високорівневі операції з базою даних, такі як вставка даних, оновлення та видалення, що дозволяє скоротити кількість коду та спростити роботу з базою даних.

Наведемо приклад коду на оновлення даних у таблиці users бази даних з використанням бібліотеки knex:

```
const knex = require('knex')({
  client: 'mysql',
  connection: {
    host: '127.0.0.1',
    user: 'your_database_user',
    password: 'your_database_password',
    database: 'myapp_test'
  }
});

// Оновлюємо дані користувача з id=1
knex('users')
  .where({ id: 1 })
  .update({
    first_name: 'John',
    last_name: 'Doe',
    email: 'johndoe@example.com'
  })
  .then(function(result) {
    console.log(result); // Виводимо кількість змінених рядків
  })
  .catch(function(error) {
    console.error(error); // Виводимо помилку в разі її виникнення
  })
```

```
    .finally(function() {
      knex.destroy(); // Закриваємо з'єднання з базою даних
    });
```

```
const knex = require('knex')({
  client: 'mysql',
  connection: {
    host: '127.0.0.1',
    user: 'your_database_user',
    password: 'your_database_password',
    database: 'myapp_test'
  }
});

// Оновлюємо дані користувача з id=1
knex('users')
  .where({ id: 1 })
  .update({
    first_name: 'John',
    last_name: 'Doe',
    email: 'johndoe@example.com'
  })
  .then(function(result) {
    console.log(result); // Виводимо кількість змінених рядків
  })
  .catch(function(error) {
    console.error(error); // Виводимо помилку в разі її виникнення
  })
  .finally(function() {
    knex.destroy(); // Закриваємо з'єднання з базою даних
  });
```

У цьому прикладі ми використовуємо метод update() для оновлення даних у таблиці users. Метод where() встановлює умови для вибірки рядків, які потрібно оновити. У метод update() передаємо об'єкт зі значеннями, які потрібно змінити у відповідних рядках.

Метод then() оброблює результати запиту, а метод catch() оброблює помилки, які можуть виникнути під час виконання запиту. Метод finally() виконується в будь-якому випадку, якщо запит був виконаний, і дозволяє припинити з'єднання з базою даних.

21.9 Тестування

Мета тестування серверних застосунків полягає в тому, щоб забезпечити високу якість програмного забезпечення та покращити його функціональність, безпеку та продуктивність.

Серверні застосунки зазвичай взаємодіють з іншими системами та компонентами, виконують різноманітні операції з обробки даних, а також використовують різні протоколи та технології мережі. Тому, тестування серверних застосунків дозволяє виявити можливі проблеми та помилки, які можуть вплинути на їхню працездатність та безпеку.

Тестування серверних застосунків дозволяє перевірити наступні аспекти програмного забезпечення:

1 Функціональність: тестування допомагає перевірити, чи працює застосунок відповідно до його вимог та очікувань, і чи відповідає він специфікації.

2 Безпека: тестування дозволяє виявити потенційні уразливості та захистити систему від атак та зломів.

3 Продуктивність: тестування допомагає забезпечити високу продуктивність застосунку, перевіряючи його роботу за різних умов.

4 Сумісність: тестування допомагає перевірити, чи працює застосунок з іншими системами та компонентами, з якими він повинен взаємодіяти.

5 Масштабованість: тестування допомагає перевірити, чи працює застосунок ефективно при збільшенні обсягу даних та навантаження на систему.

Тестування слід проводити на кожному етапі життєвого циклу застосунку, починаючи зі специфікації вимог та закінчуючи етапом експлуатації.

Основні етапи життєвого циклу застосунку, на яких проводять тестування, можуть включати такі:

1 Аналіз та планування: на цьому етапі проводяться вивчення вимог до застосунку, аналіз його функціональності та встановлюються *вимоги до тестування*.

2 Розробка: на цьому етапі проводяться *модульні тести* для перевірки роботи окремих компонентів та їхньої взаємодії.

3 Інтеграція: на цьому етапі проводяться *тести для перевірки взаємодії компонентів* та їхньої взаємодії з іншими системами.

4 Системне тестування: на цьому етапі проводяться *тести для перевірки відповідності застосунка вимогам*, встановленим для нього, та очікуванням користувачів.

5 Приймальне тестування: на цьому етапі проводяться *тести для перевірки готовності застосунка до використання* користувачами та відповідності вимогам, які встановлені для нього.

6 Експлуатація: на цьому етапі проводяться тести *для перевірки* безперебійного функціонування застосунку та задоволення потреб користувачів.

Таким чином, тестування повинно проводитися на кожному етапі життєвого циклу застосунку, щоб забезпечити якість програмного забезпечення та покращити його функціональність, безпеку та продуктивність.

Процес тестування серверного застосунку складається з наступних етапів:

1 Аналіз вимог: на цьому етапі проводиться вивчення вимог до застосунку, включаючи функціональні, нефункціональні та технічні вимоги.

2 Планування тестування: на цьому етапі встановлюються стратегії тестування та визначаються тести, які потрібно провести. Також на цьому етапі встановлюються критерії приймального тестування.

3 Розробка тестових сценаріїв: на цьому етапі створюються тестові сценарії, які описують послідовність дій, необхідних для

проведення тестування. Також на цьому етапі розробляються тестові дані.

Наведемо приклад тестового сценарію для тестування функції додавання користувача до бази даних:

Крок 1: Запуск тестового сценарію.

Крок 2: Ввійти в систему за допомогою валідних облікових даних.

Крок 3: Перейти до сторінки додавання користувача.

Крок 4: Заповнити форму додавання користувача з валідними даними.

Крок 5: Натиснути кнопку "Додати".

Крок 6: Перевірити, що користувача було додано до бази даних та всі відповідні дані були збережені правильно.

Крок 7: Перевірити, що на сторінці списку користувачів новий користувач був доданий та всі відповідні дані відображені правильно.

Крок 8: Вийти з системи.

Крок 9: Завершити тестовий сценарій.

Цей тестовий сценарій включає послідовність дій, необхідних для тестування функції додавання користувача до бази даних. Після запуску тестового сценарію користувач вводить свої облікові дані, переходить до сторінки додавання користувача, заповнює форму додавання користувача з валідними даними, натискає кнопку "Додати" та перевіряє, що користувача було додано до бази даних та всі відповідні дані були збережені правильно. Крім того, тестується, що на сторінці списку користувачів новий користувач був доданий та всі відповідні дані відображені правильно. Після цього користувач виходить з системи та завершує тестовий сценарій.

Крок 4 цього сценарію може бути модифікований завдяки введенню невалідних даних з метою виявлення вразливостей застосунка. За введення таких даних важливо охопити якомога ширший спектр помилкових або злочинних дій користувача.

4 Виконання тестів: на цьому етапі проводяться тести згідно зі створеними тестовими сценаріями та використовуються різні техніки тестування.

5 Аналіз результатів: на цьому етапі аналізуються результати тестів та визначаються проблеми та помилки.

6 Виправлення помилок: на цьому етапі виправляються виявлені проблеми та помилки.

7 Перевірка виправлень: на цьому етапі перевіряється, чи були виправлені виявлені проблеми та помилки.

8 Приймальне тестування: на цьому етапі проводиться приймальне тестування, щоб перевірити, чи відповідає застосунок вимогам, встановленим для нього, та чи відповідає він очікуванням користувачів.

9 Документування результатів: на цьому етапі документуються результати тестування та виправлень, які були зроблені після аналізу результатів.

10 Тестування продуктивності та навантаження: на цьому етапі перевіряється працездатність та швидкодія серверного застосунку під час високого навантаження, яке може бути спричинене одночасним доступом великої кількості користувачів або обробкою великого обсягу даних.

Тестування продуктивності та навантаження дозволяє виявити потенційні проблеми, які можуть виникнути при великому навантаженні на серверний застосунок, такі як:

– Перевищення максимальної кількості запитів до бази даних;

– Відмови в обслуговуванні на піку навантаження;

– Повільна швидкість роботи застосунку при великому обсязі даних;

– Нестабільність та збої в роботі застосунку при високому навантаженні.

Для проведення тестування продуктивності та навантаження використовують різні техніки, наприклад:

– Тестування з використанням засобів автоматизованого навантаження, таких як Apache JMeter, Gatling, LoadRunner та інших;

– Моніторинг серверів та баз даних під час виконання тестів;

– Аналіз результатів, що дозволяє виявити буттєві проблеми та проблеми, пов'язані з продуктивністю та швидкодією серверного застосунку.

– Тестування продуктивності та навантаження є важливою складовою частиною процесу тестування серверних застосунків, щоб забезпечити, що застосунок буде працювати без перебоїв та задовольнятиме вимоги користувачів.