Лекція

з дисципліни: «Клієнт-серверне програмування»

на тему: «*Web Applications Architecture*»

ПЛАН ЛЕКЦІЇ

1. Принципи функціонування World Wide Web.
2. Сервіс WWW.
3. Backend / Frontend
4. AJAX.
5. Application programming interface.
6. Монолітна/мікросервісна архітектура.

1. Принципи функціонування World Wide Web.

Одна людина на ім’я Вільям Генрі Гейтс III багато років тому (в середині 90-х років XX століття) сказав приблизно так: «... Якщо ваш бізнес не представлений в WEB, у вас немає бізнесу…»

Майже все програмне забезпечення в світі – веб-додатки!!!

Система WWW була створена у 1989 році вченими організації CERN (Європейський центр ядерних досліджень) у Женеві. World Wide Web спочатку була призначена для використання різними групами спеціалістів, які за її допомогою могли отримати доступ до заздалегідь підготовленої інформації.

WWW працює за принципом клієнт-сервер: існує велика кількість серверів, які за запитом клієнта надають йому гіпермедійний документ. Такий документ складається із частин з різним представленням інформації (текст, графіка, звук, відео, тривимірні об'єкти тощо). В ньому кожен елемент може бути посиланням на інший документ чи його частину. Такі посилання в WWW організовані так, що кожний інформаційний ресурс в глобальній мережі Internet однозначно адресується, а надісланий сервером документ може посилатися на інші документи на цьому ж сервері, чи на документи на інших комп'ютерах Internet. При цьому користувач не помічає цього і працює з усім інформаційним простором Internet, як з єдиним цілим. Посилання WWW вказують не тільки на документи, специфічні для самої WWW, але й на інші сервіси і інформаційні ресурси Internet. Більш того, більшість програм клієнтів WWW (браузер) не просто розуміють такі посилання, а є додатково програмами-клієнтами відповідних сервісів: FTP, Gopher, новин мережі Usenet, електронної пошти і т. і. Таким чином, World Wide Web — це глобальна система поширення гіпертекстової інформації, яка використовує для транспортування канали Інтернет.

Термін *гіпертекст* було введено задовго до появи Інтернету Тедом Нельсоном приблизно в 1965 році. Аналогом гіпертексту може бути звичайна енциклопедія. Її том складається з невеликих статей на певні теми, а у кожній

з них можуть міститися посилання на інші статті. Якщо вас зацікавила стаття, зазначена у посиланні, ви можете згідно з ним звернутися до потрібного тому.

Гіпертексти, на відміну від друкованої енциклопедії-книги, є електронними документами. З ними можна працювати лише на комп'ютері, бо в друкованому вигляді їх не існує. Прикладом гіпертекстової системи є довідкова система ОС Windows.

Гіпертекст — це спосіб організації тексту, графіки та інших даних, у якому елементи даних пов'язані між собою. Пов'язаними можуть бути як елементи одного документа, так і різних документів. Гіпертекстова структура є основою World Wide Web.

Зв'язки (від англ. «links») в гіпертекстовій структурі здійснюються за допомогою посилань. Керуючись ними, користувач може з одного документа викликати інший, з нього — наступний і т. д.

Основними перевагами гіпертекстів є, насамперед, можливість розмістити на невеликій площі (приблизно на кількох екранах) велику кількість інформації і, по-друге, зрозумілий спосіб пошуку інформації за допомогою посилань.

Гіпертекстові документи у World Wide Web розміщуються на Web-серверах. Web-сервери обробляють запити клієнтів і повертають їм копії потрібних документів.

Гіпермедіа - це термін, який використовується для гіпертексту, який не обмежений текстом: він може включати, наприклад, графіку, відео та звук.

Гіпертекст і гіпермедіа - це концепції, а не продукти.

2. *Сервіс WWW.*

WWW — мережна технологія прикладного рівня стека TCP/IP, побудована на клієнт-серверній архітектурі та використовує інфраструктуру Інтернет для взаємодії між сервером та клієнтом

Сервери www (веб-сервери) - це сховища гіпертекстової (загалом) інформації, керовані спеціальним програмним забезпеченням.

Документи, подані у вигляді гіпертексту, називаються веб-сторінками. Декілька веб-сторінок, об'єднаних загальною тематикою, оформленням, пов'язаних гіпертекстовими посиланнями і зазвичай знаходяться на тому самому веб-сервері, називаються веб-сайтом.

Для завантаження та перегляду інформації з веб-сайтів використовуються спеціальні програми – браузери, здатні обробляти гіпертектову розмітку та відображати вміст веб-сторінок.

Слово «браузер» походить від англійського «browse» — «читати безладно». Можливо, цей термін відображає характер роботи середнього користувача з Web-сторінками у WWW.

Веб-браузер може бути клієнтом, а програма, що працює на комп’ютері, на якому розміщено веб-сайт, може бути сервером.

Клієнт надсилає на сервер повідомлення HTTP-запиту.

Веб-сервер — це програма, запущена на мережевому комп'ютері і чекає на запити клієнта за протоколом HTTP. Браузер може звернутися до веб-сервера за доменним ім'ям або ір-адресою, надаючи в запиті ідентифікатор необхідного ресурсу. Отримавши запит від клієнта, сервер знаходить відповідний ресурс на локальному пристрої зберігання та надсилає його як відповідь. Сервер надає такі ресурси, як файли HTML та інший вміст, або виконує інші функції від імені клієнта, повертає клієнту відповідне повідомлення. Браузер приймає відповідь та обробляє її відповідним чином, залежно від типу ресурсу (відображає гіпертекст, показує зображення, зберігає отримані файли тощо).

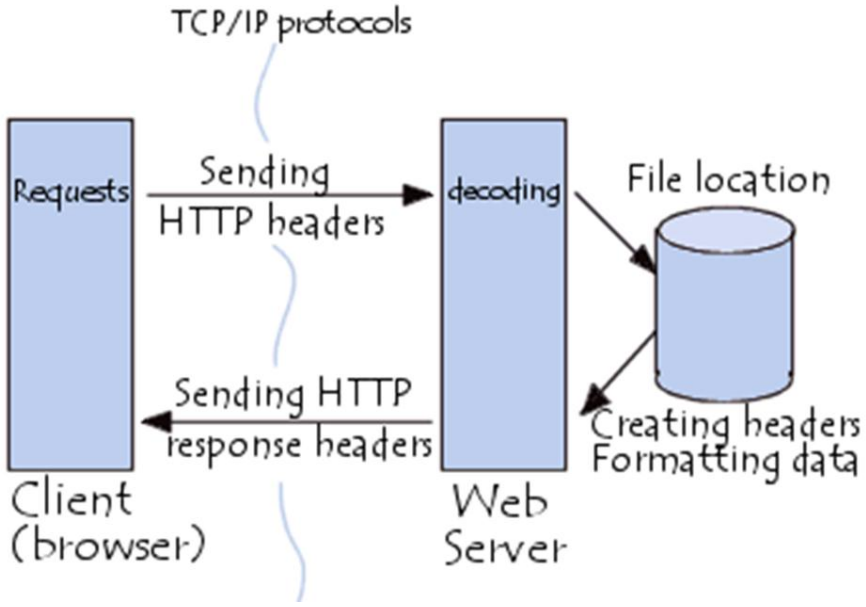

Рис. 1 Схема функціонування моделі клієнт-сервер

Функціонування сервісу забезпечується складовими:

URL/URI - уніфікований спосіб адресації та ідентифікації мережевих ресурсів;

HTML – мова гіпертекстової розмітки веб-документів;

HTTP – протокол передачі гіпертексту;

Адресація веб-ресурсів. URL, URN, URI

Для доступу до будь-яких мережних ресурсів необхідно знати, де вони розміщені і як до них можна звернутися. У Всесвітньому павутинні для звернення до веб-документів спочатку використовується стандартизована схема адресації та ідентифікації, яка враховує досвід адресації та ідентифікації таких мережевих сервісів, як e-mail, telnet, ftp тощо.

URL, Uniform Resource Locator.

URL (RFC 1738) - уніфікований локатор (покажчик) ресурсів, стандартизований спосіб запису адреси ресурсу в www та мережі Інтернет. Адреса URL має гнучку та розширювану структуру для максимально природної вказівки місцезнаходження ресурсів у мережі. Для запису адреси використовується обмежений набір символів ASCII. Загальний вигляд адреси можна представити так:

<схема>://<логін>:<пароль>@<хост>:<порт>/<повний-шлях-до-ресурсу>

Де:

Схема - звернення до ресурсу: http, ftp, gopher, mailto, news, telnet, file, man, info, whatis, ldap, wais тощо.

<логін>:<пароль> ім'я користувача та його пароль, які використовуються для доступу до ресурсу

хост

доменне ім'я хоста або його IP-адресу.

порт

порт хоста для підключення

повний-шлях-до-ресурсу

уточнююча інформація про місце знаходження ресурсу (залежить від протоколу).

У серпні 2002 року RFC 3305 анонсував старіння URL на користь URI (Uniform Resource Identifier), ще більш гнучкого способу адресації, який увібрав можливості як URL, так і URN (Uniform Resource Name, уніфіковане ім'я ресурсу). Уніфікований ідентифікатор ресурсу (URI) – це рядок символів, який використовується для ідентифікації назви веб-ресурсу. URI дозволяє не лише вказувати місцезнаходження ресурсу (як URL), а й ідентифікувати його в заданому просторі імен (як URN). Поточна структура та синтаксис URI регулюється стандартом RFC 3986, що вийшов у січні 2005 року.

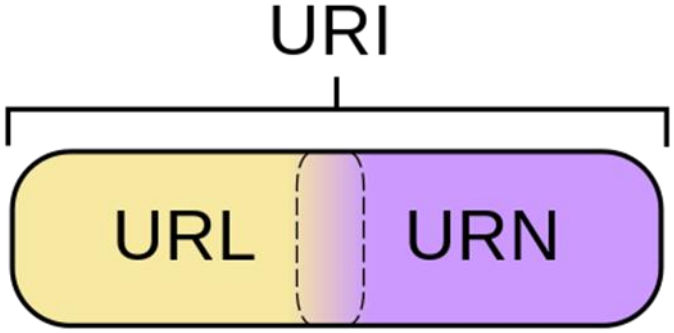

Рис. 2 Uniform Resource Identifier

HTTP протокол

HTTP - це протокол прикладного рівня, розроблений для обміну гіпертекстовою інформацією в мережі Internet. Він функціонує як протокол запиту-відповіді в обчислювальній моделі клієнт-сервер.

HTTP дозволяє реалізувати в рамках обміну даними набір методів доступу, що базуються на специфікації універсального ідентифікатора ресурсів (Universal Resource Identifier), що застосовується у формі універсального локатора ресурсів (Universe Resource Locator) або універсального імені ресурсу (Universal Resource Name).

Структура протоколу визначає, що кожне HTTP-повідомлення складається з трьох частин (рис. 3), які передаються у такому порядку:

- Стартовий рядок (англ. Starting line) - Визначає тип повідомлення;
- Заголовки (англ. Headers) - характеризують тіло повідомлення, параметри передачі та інші відомості;
- Тіло повідомлення (Message Body) - безпосередньо дані повідомлення. Обов'язково має відокремлюватися від заголовків порожнім рядком.

Стартовий рядок є обов'язковим елементом, оскільки вказує на тип запиту/відповіді, заголовки та тіло повідомлення можуть бути відсутніми.

HTTP визначає набір *методів запиту*, щоб вказати бажану дію, яку потрібно виконати для певного ресурсу.

Кожен із них реалізує різну семантику, але деякі мають спільні риси: напр. метод запиту може бути безпечним, ідемпотентним або кешованим.

Метод	Опис
GET	Використовується для запиту на вміст вказаного ресурсу. За допомогою методу GET можна також розпочати будь-який процес. У цьому випадку в тіло повідомлення у відповідь слід включити інформацію про хід виконання процесу. Клієнт може передавати параметри виконання запиту URI цільового ресурсу після символу «?»: GET /path/resource?param1=value1¶m2=value2 HTTP/1.1 Відповідно до стандарту HTTP, запити типу GET вважаються ідемпотентними - багаторазове повторення одного й того ж запиту GET має призводити до однакових результатів (за умови, що сам ресурс не змінився за час між запитами). Це дозволяє кешувати відповіді на запити GET. Існує різновид методу GET - умовний GET. При використанні цього методу сервер відповість на запит тільки в тому випадку, якщо будуть виконані умови передачі. Це дозволяє розвантажити мережу та позбавити її від передачі непотрібної інформації. Умова вказується в полі "if-Modified-Since" заголовка запиту.
HEAD	Аналогічний методу GET, за винятком того, що у відповіді сервера відсутнє тіло. Запит HEAD зазвичай застосовується для отримання метаданих, перевірки наявності ресурсу (валідація URL) і щоб дізнатися, чи не змінився він з моменту останнього звернення. Заголовки відповіді кешуються. При розбіжності метаданих ресурсу з відповідною інформацією в кеші копія ресурсу позначається як застаріла.
POST	Застосовується для передачі користувацьких даних заданому

	ресурсу. Наприклад, у блогах відвідувачі зазвичай можуть вводити свої коментарі до записів у HTML-форму, після чого вони передаються серверу методом POST і він поміщає їх на сторінку. При цьому дані (у прикладі з блогами — текст коментаря) включаються в тіло запиту. Аналогічно, за допомогою методу POST зазвичай завантажуються файли. Не є імпотентним.
PUT	Використовується для завантаження вмісту запиту на вказаний у запиті URI. Якщо за заданим URI не існувало ресурсу, сервер створює його і повертає статус 201 (Created). Якщо ж був змінений ресурс, сервер повертає 200 (Ok) або 204 (No Content). Сервер не повинен ігнорувати некоректні заголовки Content-*, що передаються клієнтом разом з повідомленням. Якщо якийсь із цих заголовків не може бути розпізнаний або не допустимий за поточних умов, необхідно повернути код помилки 501 (Not Implemented). Фундаментальна відмінність методів POST та PUT полягає у розумінні призначень URI ресурсів. Метод POST передбачає, що за вказаним URI буде проводитися обробка вмісту, що передається клієнтом. Використовуючи PUT, клієнт припускає, що завантажуваний вміст відповідає ресурсу, що знаходиться по даному URI.
PATCH	Аналогічно PUT, але застосовується лише до фрагменту ресурсу.
DELETE	Видаляє вказаний ресурс
TRACE	Повертає отриманий запит так, що клієнт може побачити, що проміжні сервери додають або змінюють в запиті.
LINK	Встановлює зв'язок зазначеного ресурсу з іншими.

Усі HTTP-заголовки поділяються на чотири основні групи:

- General Headers (Основні заголовки) – повинні включатися до будь-якого повідомлення клієнта та сервера.
- Request Headers (Заголовки запиту) – використовуються лише у запитах клієнта.
- Response Headers (Заголовки відповіді) – присутні лише у відповідях сервера.
- Entity Headers (Заголовки сутності) супроводжують кожну сутність повідомлення.

Сутності (entity, у перекладах також зустрічається назва "об'єкт") - це корисна інформація, що передається у запиті чи відповіді. Сутність складається з метаінформації (заголовки) та безпосередньо змісту (тіло повідомлення).

Тіло HTTP повідомлення (message-body), якщо воно є, використовується для передачі сутності, пов’язаної з запитом чи відповіддю.

Тіло повідомлення (message-body) відрізняється від тіла сутності (entity-body) тільки в тому випадку, коли при передачі застосовується кодування, вказане в заголовку Transfer-Encoding. В інших випадках тіло повідомлення ідентичне тілу сутності.

HTML

HTML - це мова розмітки, яка представляє прості правила оформлення та компактний набір структурних та семантичних елементів розмітки (тегів). HTML дозволяє описувати спосіб представлення логічних частин документа (заголовки, абзаци, списки тощо) та створювати веб-сторінки різної складності.

Спочатку мова HTML (HyperText Markup Language) була задумана і створена як засіб структурування та форматування документів без прив'язки до засобів відображення.

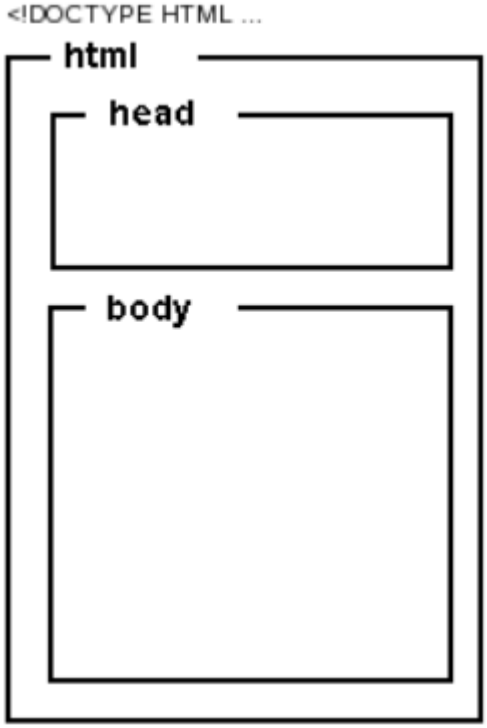

Рис. 3 Загальна структура веб-сторінки

HTML-документ складається з тексту, який є інформаційним вмістом і спеціальних засобів мови HTML — тегів розмітки, які визначають структуру та зовнішній вигляд документа при його відображенні браузером. Структура HTML-документа (рис. 3) досить проста:

- Опис документа починається із зазначення його типу (секція DOCTYPE).
- Текст документа розміщується у тегу <html>. Текст документа складається з заголовка та тіла, які виділяються відповідно до тегів <head> і <body>.

У заголовку (<head>) вказують назву HTML-документа та інші параметри, які браузер використовуватиме під час відображення документа.

Тіло документа (<body>) - це та частина, в яку міститься власне вміст HTML-документа. Тіло включає призначений для відображення текст та керуючу розмітку документа (теги), які використовуються браузером.

Наявність секції DOCTYPE дозволяє вказати браузеру, який тип документа йому належить розбирати, тобто які вимоги потрібно виконувати при обробці гіпертексту.

Заголовок призначений для розміщення метаінформації, яка описує веб-документ як такий.

Блок <body> містить те, що потрібно показати користувачеві: текст, зображення, впроваджені об'єкти та ін.

1. *Backend / Frontend.*

Класичний веб-додаток має структуру, показану на рис. 4.

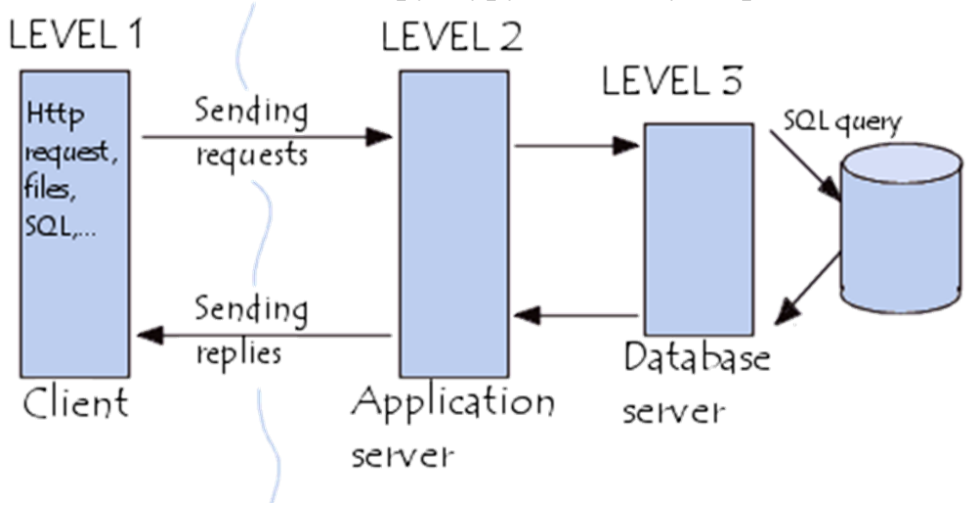

Рис. 4 Структура класичнго веб-додатоку

Сервер бази даних призначений для зберігання даних. Веб-сервер містить програмний код системи, який забезпечує бізнес-логіку та рівень доступу до даних (взаємодія між веб-сервером і базою даних)

Немає чіткого поділу між Frontend і Backend. Усі дії виконуються на сервері (Backend).

Внутрішній код динамічно генерує веб-сторінки на основі даних користувача

Frontend – це публічна частина web-додатків (веб-сайтів), з якою користувач може взаємодіяти і контактувати напряму. У Frontend входить відображення функціональних завдань призначеного для користувача інтерфейсу, що виконуються на стороні клієнта, а також обробка запитів користувачів. По суті, фронтенд – це все те, що бачить користувач при відкритті web-сторінки.

У свою чергу, web-додаток – клієнт-серверний додаток, в якому клієнтом виступає в основному браузер, а сервером – web-сервер. Логіка web-додатку розподілена між сервером і клієнтом, зберігання даних здійснюється переважно на сервері, обмін інформацією відбувається у

мережі. Простіше кажучи, це те, що бачить користувач і які дії виконує кожен раз, коли підключається до мережі інтернет і відкриває будь-який браузер.

Frontend розробка – це робота зі створення публічної частини web-додатку, з якою безпосередньо контактує користувач, і функціоналу, який зазвичай виконується на стороні клієнта. Тобто, фронтенд розробник працює над тим, щоб на сайті кожна кнопочка, іконка, текст і вікно не тільки стояли на своєму місці, не перекривали один одного і виглядали цілісно (це веб-верстка), але і щоб вони виконували своє пряме призначення – підштовхували до якоїсь дії (наприклад, щоб кнопка "купити" відкривала кошик, а "play" – запускала відтворення фільму або музики).

Backend – це програмно-апаратна частина проекту, Frontend ж є клієнтською стороною призначеного для користувача інтерфейсу до програмно-апаратної частини проекту, тобто до бекенду. Іншими словами бекенд – це все те, що відбувається на стороні сервера і що залишається невидимим користувачеві (сам сервер теж є частиною бекенду, тільки апаратного). Звідси і назва front – це видиме спереду, back – це те, що приховано позаду, невидиме.

Наприклад, ви оплачуєте покупку в інтернеті: вводите дані карти, клікаєте "сплатити" і бачите напис "ваш платіж прийнятий в обробку" – це був фронтенд. Те, як рухаються ваші гроші всередині мережі і те, як ваше замовлення надходить в магазин – це бекенд. Відповідно, коли магазин бачить повідомлення про те, що надійшло замовлення, а гроші зараховані на рахунок – це знову робота фронтенда.

Бекенд-розробники мають справу з серверними мовами програмування, такими як Java, Python, PHP, Ruby та інші. Також бекендери повинні знати бази даних, архітектуру, до всього іншого їм знадобляться знання апаратної частини бекенд, тобто сервера, його можливості і характеристики. Бекенд-розробники, як правило, не мають відношення ні з чим, що безпосередньо взаємодіє з користувачем, вони не розбираються в призначених для користувача інтерфейсах UI і не заглиблюються в призначений для користувача досвід взаємодії UX, або в верстку сторінки, хоча загальне розуміння всього цього мають. Вони працюють, в основному, з точним аналізом і обчисленнями, де майже немає творчої, гуманітарної складової. При цьому, їм потрібно вміти обчислювати всі можливі наслідки операцій та розуміти причини помилок, що з'явилися на шляху клієнт-сервер-клієнт.

Розглянемо процес взаємодії frontend і backend:

фронтенд відправляє призначену для користувача інформацію в бекенд;

інформація обробляється;

інформація повертається назад, прийнявши цілісну форму і виконавши оброблений запит.

Варіанти взаємодії frontend і backend:

HTTP-запит відправляється на сервер, сервер в процесі пошуку інформації, вбудовує її в шаблон і повертає назад у вигляді HTML-сторінки.

Випадок із застосуванням інструментарію AJAX (Asynchronous JavaScript and XML). В даному випадку запит відправляє JavaScript, який завантажений в браузер, відповідь же приходить в форматі XML або JSON.

Односторінкові додатки, які завантажують дані без оновлення сторінок. Це робиться за допомогою AJAX або фреймворків Angular і Ember.

Ember або бібліотека React надають допомогу у використанні програми і в клієнтській частині і на сервері. Frontend і backend взаємодіють через AJAX і HTML-код, який обробляється на сервері.

2. *AJAX.*

AJAX (АЯКС - Asynchronous JavaScript And XML) - це технологія яка дозволяє відправляти та отримувати дані з сервер без перезавантаження веб-сторінки.

У назві технології AJAX є слово XML, але це не означає може використовуватися лише XML для передачі даних, але й дані у форматі JSON або простого тексту.

У JavaScript технологія AJAX доступна за допомогою:

XMLHttpRequest

window.fetch

navigator.sendBeacon()

Технологія AJAX дозволяє:

оновлення веб-сторінки без перезавантаження сторінки.

запит і отримання даних з сервера після завантаження сторінки

відправлення даних на сервер у фоновому режимі.

При використані AJAX можна не оновлювати веб-сторінку, а змінити її потрібну частину отримавши дані від сервера у фоному режимі.

AJAX — це синтез декількох технологій, який забезпечує розробників низкою програмних рішень. Назва технології складається з абревіатури, яка розшифровується як Asynchronous Javascript and XML. Таким чином в основі AJAX використовуються дві технології Javascript і XML, які застосовуються на ринку цифрових рішень вже багато років. Крім того, складовими елементами підходу є HTML, CSS, PHP і DOM. В основному AJAX розглядається як веб-додаток нового формату, яке відноситься до Web 2.0.

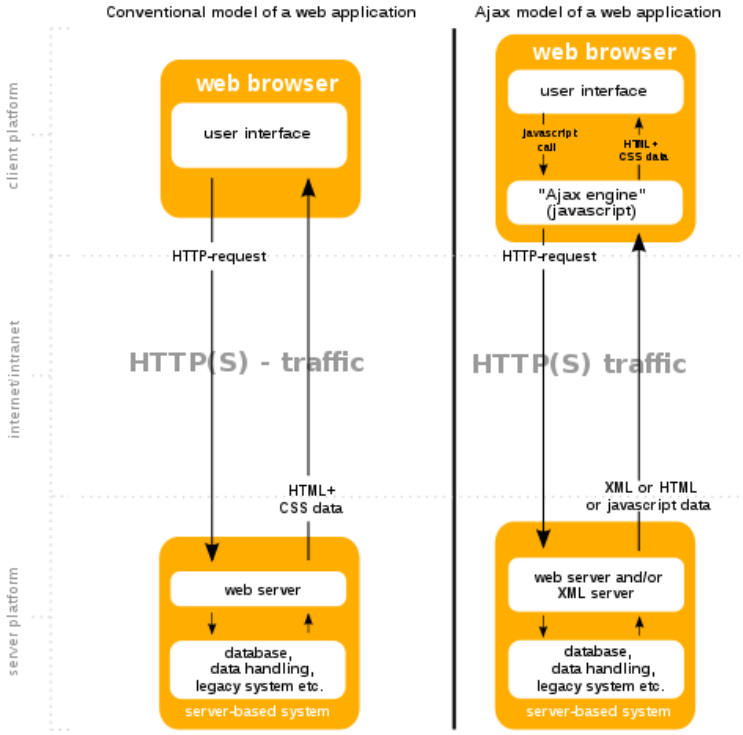

Рис. 5 Модель класичних застосунків для мережі в порівнянні із застосуванням AJAX

Особливості роботи та застосування AJAX

AJAX є одним з підходів до розробки користувацьких веб-інтерфейсів, який передбачає «фоновий» обмін даними між браузером і серверними ресурсами. Асинхронний спосіб оновлення даних веб-сторінок дозволяє не перезавантажувати їх вміст повністю. Завдяки цьому веб-ресурс працює більш плавно і швидко. Також це дозволяє економити трафік, що особливо актуально в умовах, якщо немає безлімітного інтернету.

Але коли на сайті застосовується технологія AJAX, користувачі помічають часткову зміну сторінок. Тому розробникам слід подумати над впровадженням індикації цих змін. Користувач повинен розуміти, що зараз відбувається фоновий обмін даних з сервером.

Ще одна важлива особливість полягає в тому, що не всі браузери працюють з Аяксом. Його не підтримують, наприклад, старі і текстові версії браузерів. А ще іноді Javascript блокується користувацькими налаштуваннями.

Тому девелопери часто рекомендують шукати для технології більш гнучкі альтернативи та інші методи відображення даних на веб-ресурсі.

Основні переваги і недоліки AJAX

AJAX не завжди рекомендується застосовувати розробникам. В окремих випадках технологія принесе свої плюси, але іноді може погано позначитися на роботі веб-ресурсу.

До основних переваг підходу належать:

зменшення трафіку – завдяки частковому оновленню ресурсів не потрібно перезавантажувати сторінку;

зниження навантаження на сервер – технологія дозволяє робити менше запитів до бази даних;

швидка робота сторінок сайту – відгук на дії користувача відбувається більш динамічно;

поліпшення функціональності веб-ресурсу.

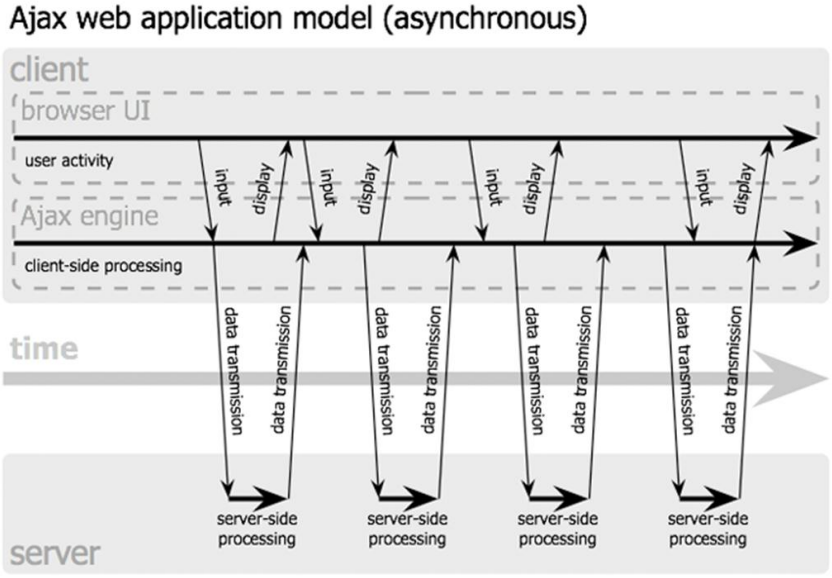

Рис. 6 Модель AJAX web application

Недоліки AJAX

Але при цьому, застосування AJAX може бути загрожує суттєвими недоліками.

Основними мінусами роботи сайту з використанням технології Аякс буде:

відсутність гарантій безпеки – весь програмний код скриптів доступний для перегляду в браузері і цим можуть скористатися зловмисники;

історія відвідування сторінок не працює, і це погано для ведення статистики та аналізу поведінки відвідувачів;

погіршена індексація – сторінки, на яких використовується AJAX будуть значно гірше сприйматися пошуковими роботами. Іноді бувають випадки, коли вони й зовсім випадають з індексу.

3. *Application programming interface*.

Прикладний програмний інтерфейс (інтерфейс програмування застосунків, інтерфейс прикладного програмування) (Application Programming Interface, API) — набір означень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено - це набір чітко означених методів (функцій) для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення оскільки можна скористуватися готовими об’єктами (функціями) іншого програмного забезпечення через означені в останньому правила взаємодії. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

При використанні прикладного програмного інтерфейсу в контексті веб-розробки, як правило, API означується набором повідомлень-запитів HTTP та структурою повідомлень-відповідей. Повідомлення можуть мати різний формат, як правило це XML або JSON. Доступ відбувається до з однієї або декількох загальнодоступних кінцевих точок (endpoints).

Кінцеві точки є важливими аспектами взаємодії з веб-інтерфейсами на стороні сервера, оскільки вони вказують, де знаходяться ресурси, доступ до яких може отримати стороння програма. Зазвичай доступ здійснюється через URI, до якого надсилаються HTTP-запити, і звідки очікується відповідь. Кінцеві точки повинні бути статичними, інакше правильне функціонування програмного забезпечення, яке взаємодіє з нею, не може бути гарантоване. Якщо місце розташування ресурсу змінюється (і разом з ним кінцева точка), то раніше написане програмне забезпечення буде перервано, оскільки потрібний ресурс більше не може бути знайдено в одному місці. Оскільки постачальники API все ще хочуть оновлювати свої веб-API, багато хто з них запровадили систему версій в URI, яка вказує на кінцеву точку, наприклад, Clarifai API: кінцева точка для функцій позначення в Web API має такий URI: "https: //api.google.com/v1/tag/ ". "/ V1 /" частина URI визначає доступ до першої версії веб-API. Якщо Clarifai вирішить оновити до другої версії, вони можуть це зробити, зберігаючи при цьому підтримку стороннього програмного забезпечення, яке використовує першу версію.

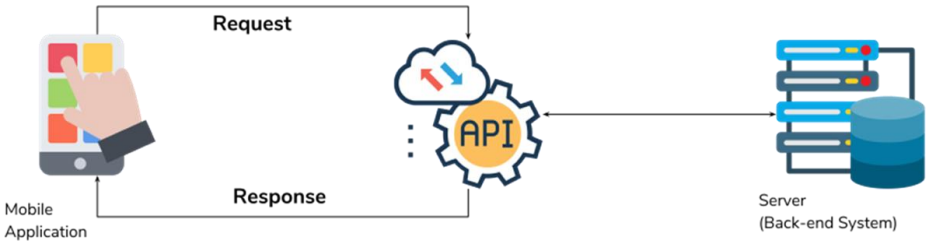

Рис. 7 Модель функціонування API

REST (Representational State Transfer, «передача репрезентативного стану») означує ряд архітектурних принципів проектування Web-сервісів, орієнтованих на системні ресурси, включаючи способи обробки і передачі станів ресурсів по HTTP різноманітними клієнтськими додатками, написаними різними мовами програмування. За останні кілька років REST стала переважаючою моделлю проектування Web-сервісів. Фактично REST зробила настільки великий вплив на Web, що практично витіснила розробку інтерфейсу, заснованого на SOAP і WSDL, через значно більш простий стиль проектування.

Технологія REST не привернула велику увагу в 2000 році, коли Рой Філдінг (Roy Fielding) вперше представив її в Каліфорнійському університеті в Ірвайні в своїй дисертації "Архітектурні стилі і дизайн мережевих архітектур програмного забезпечення", де аналізувався набір принципів архітектури програмного забезпечення, що використовує Web як платформу розподілених обчислень. Однак сьогодні, через багато років, виникли і продовжують розвиватися численні інфраструктури для REST.

Надалі передбачається, що конкретна реалізація Web-сервісів REST слідує чотирьом базовим принципам проектування:

• Явне використання HTTP-методів.

• Незбереження стану.

• Надання URI, аналогічних структурі каталогів.

• Передача даних в XML, JavaScript Object Notation (JSON) або в обох форматах.

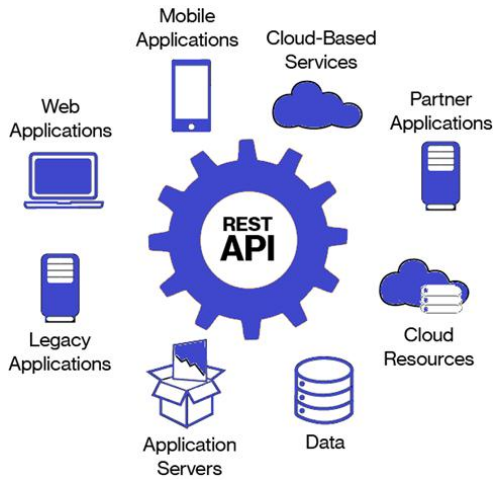

Рис. 8 Модель REST API

Однією з ключових характеристик Web-сервісу RESTful є явне використання HTTP-методів згідно з протоколом, означеним в RFC 2616. Наприклад, HTTP GET означений як метод генерування даних, використовуваний клієнтським додатком для вилучення ресурсу, отримання даних з Web-сервера або виконання запиту в надії на те, що Web-сервер знайде і поверне набір відповідних ресурсів.

REST пропонує розробникам використовувати HTTP-методи явно відповідно до означення протоколу. Цей основний принцип проектування REST встановлює однозначну відповідність між операціями create, read, update і delete (CRUD) і HTTP-методами. Згідно з цим відповідності:

• Для створення ресурсу на сервері використовується POST.

• Для отримання ресурсу використовується GET.

• Для зміни стану ресурсу або його поновлення використовується PUT.

• Для видалення ресурсу використовується DELETE.

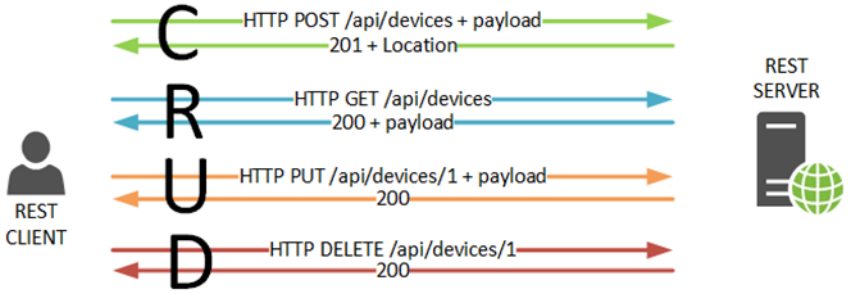

Рис. 9 Модель RESTful

Незбереження стану

Для задоволення постійно зростаючих вимог до продуктивності Web-сервіси REST повинні бути масштабованими. Для формування топології сервісів, що дозволяє при необхідності перенаправляти запити з одного сервера на інший з метою зменшення загального часу реакції на виклик Web-сервісу, зазвичай застосовують кластери серверів з можливістю розподілу навантаження і аварійного перемикання на резерв, проксі-сервери і шлюзи. Використання проміжних серверів для поліпшення масштабованості вимагає, щоб клієнти Web-сервісів REST відправляли повні самодостатні запити, що містять всі необхідні для їх виконання дані, щоб компоненти на проміжних серверах могли перенаправляти, маршрутизувати і розподіляти навантаження без локального збереження стану між запитами.

При обробці повного самодостатнього запиту серверу не потрібно витягувати стан або контекст програми. Застосунок (або клієнт) Web-сервісу REST включає в HTTP-заголовки і в тіло запиту всі параметри, контекст і дані, необхідні серверному компоненту для генерування відповіді. У цьому сенсі незбереження стану (statelessness) покращує продуктивність Web-сервісу і спрощує дизайн і реалізацію серверних компонентів, оскільки відсутність стану на сервері усуває необхідність синхронізації сеансових даних із зовнішнім застосунком.

На рисунку 10 показаний сервіс, що зберігає стан, в якому застосунок може запитати наступну сторінку в багатосторінковому наборі результатів, вважаючи, що сервер зберігає послідовність переходів застосунку з цього набору результатів. У цій моделі зі збереженням стану сервіс нарощує і зберігає змінну previousPage, щоб бути в змозі відповідати на запити наступних сторінок.

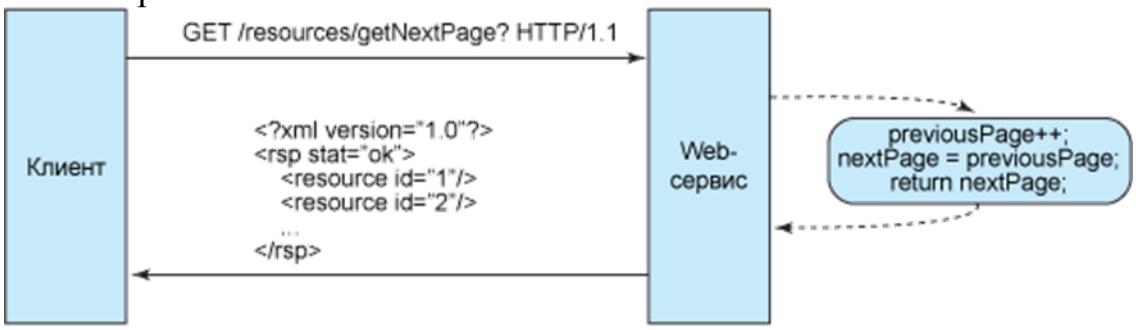

Рис. 10 Модель зі збереженням станів

Подібні сервіси, що зберігають стан, виходять дуже складними. У середовищах такого типу розробники скриптів на серверах стикаються з типовою проблемою пошуку причин виникнення виняткових ситуацій, що може коштувати розробникам декількох днів пошуків в складному графі об'єктів, що визначають стан сервера. Крім того, синхронізація сеансів збільшує накладні витрати, що негативно позначається на продуктивності сервера.

На противагу цьому, серверні компоненти що не зберігають стан більш прості в проектуванні, написанні та розподілі між серверами зі

збалансованим навантаженням. Сервіс, що не зберігає стан, не тільки краще працює, але і покладає основну відповідальність за збереження стану на клієнтську програму. У Web-сервісі RESTful сервер відповідає за генерування відповідей і за надання інтерфейсу, що дозволяє клієнтському застосунку самому зберігати свій стан. Наприклад, при запиті багатосторінкового набору результатів клієнтський застосунок повинен включати в запит номер конкретної сторінки, а не просто запитувати наступну сторінку (див. Рисунок 11).

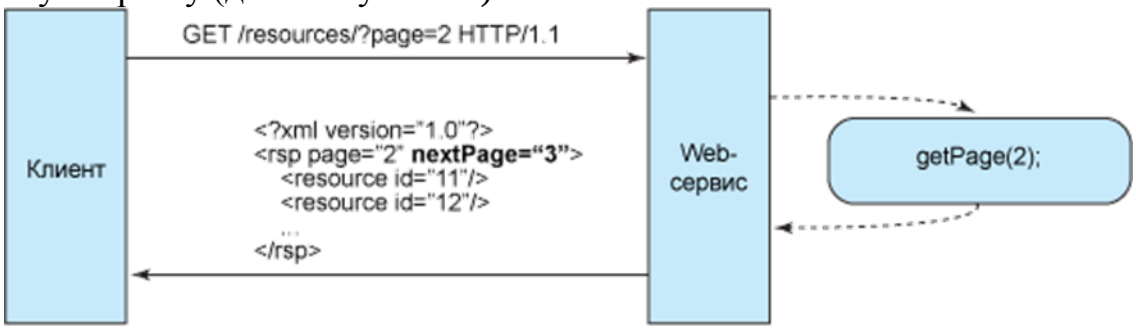

Рис. 11 Модель без збереженням станів

Web-сервіс, що не зберігає стану, генерує відповідь, що містить посилання на номер наступної сторінки в наборі і дозволяє клієнтові самостійно подбати про збереження цього значення.

Відображення URI, аналогічних структурі каталозів

З точки зору звернення до ресурсів з клієнтського за стосунку URI, що надаються, означують наскільки інтуїтивним буде Web-сервіс REST і чи буде він використовуватися так, як припускав розробник. Третя характеристика Web-сервісу RESTful повністю присвячена URI.

URI-адреси Web-сервісу REST повинні бути інтуїтивно зрозумілими. Розглядайте URI як якийсь самодокументований інтерфейс, що майже не вимагає пояснень або звернення до розробника для його розуміння і для отримання відповідних ресурсів. Тому структура URI повинна бути простою, передбачуваною і зрозумілою.

Один із способів досягти такого рівня зручності використання - побудова URI за аналогією зі структурою каталогів. Такого роду URI є ієрархічними, що походить із одного кореневого шляху, розгалуження якого відображають основні функції сервісу. Згідно з цим означенням, URI - це не просто рядок зі слешами як роздільниками, а скоріше дерево з встановленими вище і нижче лежачими гілками, з'єднаними в вузлах. Наприклад, в сервісі обговорень різних тем (від Java до паперу) можна визначити структурований набір URI такого вигляду:

http://www.myservice.org/discussion/topics/{topic}

Корінь /discussion має нижчий вузол /topics. Нижче розташовуються назви тем (наприклад, gossip (чутки), technology (технологія) і т.д.), кожна з яких вказує на свою гілку обговорення. В рамках цієї структури можна легко викликати гілки обговорення простим введенням чогось після /topics/.

У деяких випадках каталого-подібна структура особливо добре підходить для шляхів до ресурсів. Як приклад можна назвати ресурси, впорядковані за датою. Для них дуже добре підходить ієрархічний синтаксис.

Наступний приклад інтуїтивно зрозумілий, оскільки заснований на правилах:

http://www.myservice.org/discussion/2022/12/10/{topic}

Перший фрагмент шляху - чотири цифри року, другий - дві цифри дня і третій - дві цифри місяця. Подібне пояснення може здатися дещо спрощеним, але це саме той рівень простоти, який нам потрібен. Люди і комп'ютери можуть легко генерувати подібні структуровані URI, оскільки вони засновані на правилах. Вказівка фрагментів шляху у відповідних позиціях згідно синтаксису робить URI уніфікованими, оскільки існує закономірність їх створення:

http://www.myservice.org/discussion/{year}/{day}/{month}/{topic}

Передача XML, JSON

Представлення ресурсу, як правило, відображає поточний стан ресурсу (і його атрибутів) на момент його запиту клієнтським додатком. Представлення ресурсів в цьому сенсі є просто знімками в конкретні моменти часу. Ці представлення повинні бути такими ж простими, як представлення запису в базі даних, що складається з відображення між іменами стовпців і XML-тегами, де значення елементів в XML містять значення рядків. Якщо система має модель даних, то згідно з цим визначенням представлення ресурсу є знімком стану атрибутів одного з об'єктів моделі даних системи. Це ті об'єкти, які буде обслуговувати Web-сервіс REST.

Останній набір обмежень, тісно пов'язаний з дизайном Web-сервісів RESTful, відноситься до формату даних, якими обмінюються додаток і сервіс при роботі в режимі запит/відповідь або в тілі HTTP-запиту. Тут особливо важливі простота, читабельність і зв'язаність.

Об'єкти моделі даних зазвичай якось пов'язані, і ці відносини між об'єктами (ресурсами) моделі даних повинні відображатися в способі їх подання для передачі клієнтського додатку. У сервісі обговорень приклад представлень пов'язаних ресурсів може включати в себе кореневу тему обговорення і її атрибути, а також вбудовані посилання на відповіді, надіслані в цю тему.