Назва дипломної: **Розробка системи міграції даних**

**Анотація.** Коротенький зміст основних результатів роботи (2-3 речення) українською і англійською мовами.

**Вступ**

В якому наводиться ґрунтовний опис:

*Проблематики та актуальності* теми роботи, *аналіз існуючих підходів* і методів розв’язання та практичного застосування результатів їх розв’язання ( при цьому **обов’язкові посилання на використані інформаційні джерела**(монографії, статті, Інтернет інформація)), мета роботи, предмет дослідження. Об’єкт дослідження, аргументація новизни і коротка ідея запропонованого Вами підходу.

**Розділ 1. Постановка задачі**

* 1. Основні означення , теореми,леми, твердження, гіпотези.

Багатоаспектність проблеми

Проблема інтеграції даних надзвичайно багатоаспектна і різноманітна. Складність і характер використовуваних методів її рішення істотно залежать від рівня інтеграції, який необхідно забезпечити, властивостей окремих джерел даних і всієї множини джерел в цілому, необхідних способів інтеграції. Системи інтеграції даних можуть забезпечувати інтеграцію даних на фізичному, логічному і семантичному рівні. Інтеграція даних на фізичному рівні з теоретичної точки зору є найбільш простим завданням і зводиться до конверсії даних з різних джерел в необхідний єдиний формат їх фізичного представлення. Інтеграція даних на логічному рівні передбачає можливість доступу до даних, що містяться в різних джерелах, в термінах єдиної глобальної схеми, яка описує їх спільне подання з урахуванням структурних і, можливо, поведінкових (при використанні об'єктних моделей) властивостей даних. Семантичні властивості даних при цьому не враховуються. Підтримку єдиного уявлення даних з урахуванням їх семантичних властивостей в контексті єдиної онтології предметної області забезпечує інтеграція даних на семантичному рівні.

Джерела даних можуть мати різні властивості, істотними для вибору методів інтеграції даних - вони можуть підтримувати представлення даних в термінах тієї чи іншої моделі даних, можуть бути статичними або динамічними і т.п. Безліч джерел інтегрованих даних можуть бути однорідними або неоднорідними щодо характеристик, відповідних використовуваному рівню інтеграції. Що стосується способів інтеграції даних, то можливі два підходи - віртуальне або актуальне (матеріалізоване) уявлення інтегрованих даних. При першому підході створюється механізм доступу, який при обробці призначеного для користувача запиту породжує дані в необхідному поданні безпосередньо з джерел даних. Повне матеріалізоване уявлення інтегрованих даних в термінах єдиного користувальницького інтерфейсу при цьому не підтримується. Віртуальний підхід найчастіше застосовується при використанні часто оновлюваних джерел даних. Навпаки, при другому підході на стадії інтеграції формується повне матеріалізоване уявлення інтегрованих даних, відчужене від вихідних джерел і співіснуюче з ними. Саме це представлення даних використовується для обробки запитів користувачів. Такий підхід використовується, зокрема, в сховищах даних.

Неоднорідність джерел даних

Неоднорідність джерел даних проявляється в системах інтеграції даних в різних аспектах. При цьому, природно, йдеться про неоднорідність характеристик джерел, відповідних використовуваному рівнем інтеграції даних. Так, при інтеграції на фізичному рівні в джерелах даних можуть використовуватися різні формати файлів. На логічному рівні інтеграції може мати місце неоднорідність використовуваних моделей даних для різних джерел або розрізняються схеми даних, хоча використовується одна і та ж модель даних. Одні джерела можуть бути веб-сайтами, а інші - об'єктними базами даних і т.д. При інтеграції на семантичному рівні різних джерел даних можуть відповідати різні онтології. Наприклад, можливий випадок, коли кожен з джерел представляє інформаційні ресурси, що моделюють певний фрагмент предметної області, якому відповідав би своя понятійна система, і ці фрагменти перетинаються.

Завдання, які постають

При створенні системи інтеграції виникає ряд завдань, склад яких залежить від вимог до неї та використовуваного підходу. До них, зокрема, відносяться:

• Розробка архітектури системи інтеграції даних.

• Створення інтегруємої моделі даних, що є основою єдиного призначеного для користувача інтерфейсу в системі інтеграції.

• Розробка методів відображення моделей даних і побудова відображень в інтегруючу модель для конкретних моделей, які підтримуються окремими джерелами даних.

• Інтеграція метаданих, які використовуються в системі джерел даних.

• Подолання неоднорідності джерел даних.

• Розробка механізмів семантичної інтеграції джерел даних.

Архітектура систем інтеграції

У системах інтеграції даних найбільшого поширення набула архітектура з посередником. На посередника покладається завдання підтримки єдиного призначеного для користувача інтерфейсу на основі глобального представлення даних, містяться в джерелах, а також підтримку відображення між глобальним і локальним уявленнями даних. Запит, сформульований в термінах єдиного інтерфейсу, декомпозується на безліч підзапитів, адресованих до потрібних локальних джерел даних. На основі результатів їх обробки синтезується повна відповідь на запит.

Використовуються два різновиди архітектури з посередником - Global as View і Local as View. Перша з них (Global as View) передбачає визначення глобального уявлення інтегрованих даних в термінах заданих уявлень локальних джерел. Такий підхід більш ефективний у разі, коли множину всіх використовуваних джерел визначено. Якщо система інтеграції призначена для підтримки повного матеріалізованого уявлення інтегрованих даних, процеси конверсії даних з джерел в їх єдине глобальне уявлення здійснюються одноразово.

При використанні другого різновиду даної архітектури (Local as View) передбачається, що подання для кожного з локальних джерел даних визначається в термінах заданого інтегруючого глобального уявлення. Хоча в цьому випадку ускладнюється відображення для користувача запитів в середу локальних джерел даних, такий підхід допускає динамічність складу безлічі джерел даних. Кожне нове джерело може підключатися до системи як на стадії розробки, так і на стадії функціонування.

**Засоби семантичної інтеграції даних**

Найбільш поширений підхід до семантичної інтеграції даних заснований на використанні семантичних посередників (Mediators). Засобами посередників підтримуються уніфіковані метаописания інтегрованих джерел даних. Як правило, семантичні посередники розробляються для конкретної вузької предметної області. Механізми посередників спираються на онтологічні специфікації джерел. Для посередника створюється інтегрована онтологія використовуваних джерел. У таких системах необхідна також інтегруюча модель даних з розвиненими можливостями моделювання семантики даних. В останні роки з'явився ряд публікацій, присвячених вирішенню проблеми семантичної інтеграції даних з багатьох джерел, в яких для уявлення глобальної схеми в системі інтеграції даних пропонується використовувати апарат дескриптивних логік, втілений в мові опису онтологій OWL. Перевага такого підходу полягає не тільки в тому, що основою призначеного для користувача інтерфейсу є при цьому високорівнева семантична модель даних, але і можливість міркувань в термінах онтології, що служить концептуальною моделлю підходу

**Інтеграція метаданих**

Інтеграція даних в інформаційній системі природним чином передбачає і інтеграцію в тій чи іншій формі метаданих, що визначають їх джерела. Однією з традиційних завдань інтеграції метаданих в системах інтеграції структурованих даних є завдання інтеграції схем. Труднощі її рішення в конкретних ситуаціях можуть бути пов'язані з наявністю конфліктів, наприклад:

• Конфліктів неоднорідності (використовуються різні моделі даних для різних джерел)

• Конфліктів іменування (в різних схемах використовується різна термінологія, що призводить до омонімії і синонімії в іменуванні)

• Семантичних конфліктів (обрані різні рівні абстракції для моделювання подібних сутностей реального світу)

• Структурних конфліктів (одні і ті ж сутності представляються в різних джерелах різними структурами даних).

Інша типове і досить складне завдання - інтеграція онтологічних специфікацій інформаційних ресурсів.

**Протоколи доступу як засіб інтеграції даних**

Як інструмент інтеграції даних можуть виступати деякі протоколи доступу до інформаційних ресурсів. Як приклад можна назвати протокол доступу до розподілених ресурсів Z39.50. Він підтримує єдине ієрархічне уявлення розподілених інформаційних ресурсів в середовищі архітектури клієнт-сервер і надає користувачеві єдиний інтерфейс для доступу до них. Розроблений спочатку для доступу до бібліографічних даних, цей протокол згодом був адаптований для доступу до інтегрованих розподілених ресурсів іншої природи, зокрема, і до баз даних. В даний час Z39.50 забезпечує доступ до баз даних в термінах мови SQL.

**Інтеграція текстових ресурсів**

Проблема інтеграції колекцій текстових інформаційних ресурсів зводиться, головним чином, до інтеграції метаданих їх джерел, каталогів, класифікаторів, тезаурусів, онтологій і т.д. Як уже зазначалося, ця проблема набула особливої актуальності у зв'язку з розробками електронних бібліотек. Інтеграція тут розуміється, як об'єднання колекцій текстових документів з різних джерел в рамках єдиного джерела. Тут найбільш цікаві методи, що передбачають матеріалізовану інтеграцію метаданих і віртуальну інтеграцію власне контенту колекцій текстових документів. Такий підхід використовується, наприклад, в системі Соціонет (<http://socionet.ru>).

**Роль стандартів в системах інтеграції даних**

У системах інтеграції даних широко використовується ряд офіційних міжнародних стандартів, а також індустріальних стандартів де-факто. Серед них стандарти баз даних ISO / IEC SQL, ISO / IEC SQL / MED, стандарт об'єктних даних консорціуму ODMG, стандарти CORBA і UML консорціуму OMG, стандарти платформи XML консорціуму W3C, стандарт Дублінського ядра консорціуму OCLC та багато інших. Головне призначення стандартів в таких системах полягає у визначенні уніфікованої моделі даних (метаданих), що є основою єдиного інтерфейсу для доступу до інтегрованих даними для додатків і / або кінцевих користувачів. Крім того, деякі стандарти дозволяють «занурити» інтегровані дані в деяку корисну інфраструктуру і користуватися для доступу до них її функціональністю. Такі можливості забезпечують, наприклад, стандарти XML і CORBA. В інтеграції інформаційних ресурсів електронних бібліотек в останні роки активно використовуються стандарти ініціативи відкритих архівів (Open Archives Initiative, OAI; http://www.openarchives.org/).

**Методи реалізації інтеграції даних**

Найбільш поширеними методами реалізації інтеграції даних є:

* обмін на основі файлів;
* реплікація даних;
* технологія Web-сервісів;
* сервіс-орієнтована архітектура (SOA);
* інтеграційні сервери.

Обмін файлами є найбільш поширеним способом інтеграції. З точки зору реалізації - це найпростіший спосіб, але він має недоліки. У разі необхідності обміну складними

структурами, необхідна розробка спеціалізованих форматів файлів, що призводить до великої

Залежно систем один від одного. Також обмін за допомогою файлів має на увазі наявність людини,

який здійснює вивантаження і завантаження файлів. Однак в разі неможливості взаємодії

через мережу перенесення даних на фізичному носії є єдиним можливим рішенням.

Зокрема обмін даними між підсистемами і з додатками з пакету MS Office для автоматизованої інформаційної системи «Прокуратура-Статистика» був використаний саме цей метод.

Було необхідно реалізувати обмін статистичними звітами між територіально віддаленими підсистемами. Для обміну між вузлами був розроблений внутрішній формат представлення даних в двійковому файлі. А для опису структури звіту для додатків MS Word і MS Excel був використаний XML для опису структури форм звітності. Це дозволило універсально описати спосіб відображення

даних у файлі звіту для будь-якої форми.

Реплікація - процес приведення даних електронних таблиць двох БД в ідентичний стан. Процес реплікації заснований на поняттях "видавець" і "передплатник". Видавцем є сервер публікації, тобто сервер, що відправляє інформацію. Передплатником є ​​сервер, який приймає – тобто сервер підписки.

Реплікації зручно розбити на дві великі категорії: реплікація даних в середовищі між серверами

і реплікація даних між сервером і клієнтами. Реплікація даних між серверами виконується для

підтримки таких вимог:

1. Підвищення масштабованості і доступності. Обслуговування безперервно оновлюваних копій даних дозволяє масштабувати операції читання на кілька серверів. Надмірність, яка виникає внаслідок обслуговування декількох копій одних і тих же даних, критично важлива для планового і непланового обслуговування системи.
2. Зберігання даних і складання звітів на сервери сховищ даних і сервери звітів часто використовують дані з інтерактивних обробників транзакцій (OLTP).

Технологія Web-сервісів - зручний засіб інтеграції систем, що дозволяє легко реалізувати міжплатформну взаємодію. Сервіси є об'єктами, що знаходяться в певних відносинах і володіють вкладеною в них функціональністю. Будь-який Web-сервіс можна розглядати як чорний ящик, який має вхідний і вихідний порти. Але технологія Web-сервісів не може розглядатися як загальний підхід до інтеграції систем з кількох причин: Web-сервіси непридатні для обробки великих обсягів даних; в них відсутня підтримка транзакцій; в момент взаємодії інтегровані системи повинні перебувати в працездатному стані. Непридатність для обробки великих обсягів інформації пов'язана з тим, що зазвичай всі дані перетворюються в формат XML, що веде до збільшення обсягу даних і навантаження на систему в цілому.

Дуже поширеною є архітектура, орієнтована на сервіси (Service Oriented Architecture, SOA). Сервіс-орієнтована архітектура (Service-Oriented Architecture або SOA) - підхід до проектування, розгортання та експлуатації розподілених програмних систем. система, реалізована за принципами SOA, являє собою сукупність програмних компонентів, а саме сервісів, що мають стандартні інтерфейси для доступу до них за допомогою мережі і використання цих компонентів. Інтерфейси в SOA незалежні від платформ розгортання сервісів і технологій їх реалізації. Інтерфейс - це ключове поняття сервісу. За допомогою них виконується представлення можливостей сервісу зовнішньому світу і організація взаємодії сервісів. SOA пропонує єдину схему взаємодії сервісів, незалежну від того, де знаходиться сервіс. Передбачається, що сервіс виконує якусь бізнес-дію (business concept), якусь окрему функцію. Завершивши свою роботу, сервіс може викликати інший сервіс. У сервісній архітектурі немає жорстких зв'язків між модулями. Їх заміняють, використовуючи так звану, слабку зв'язність компонентів (loose coupling). Використовуючи такий зв'язок можна на ходу збирати з сервісів таку конфігурацію, яка необхідна в даний момент. Так само наряду зі звичайними даними при зверненні до сервісу викликається модуль, який знає, як його викликати і яку роботу він виконує, передає метадані. Використовуючи сукупність даних і метаданих, сервіс виконує замовлення на обслуговування.

SOA базується на наступних принципах:

* слабка зв'язність компонентів, яка дозволяє виконувати зміну всередині сервісу, що не зачіпає програмні модулі, що його використовують;
* «Грубозерниста» (coarse-grained) структура сервісів, яка забезпечує, завдяки тому,сервіси - модулі бізнес-логіки високого рівня, що не потребують наявності безлічі низькорівневих викликів, які враховують архітектуру сервісів, знизити навантаження на мережу і підвищити продуктивність;

**Advanced Message Queuing Protocol**

Відкритий стандарт протоколу прикладного рівня для проміжного програмного забезпечення, орієнтованого на обробку повідомлень. Семантика обміну повідомленнями налаштовується під потреби конкретного проекту. «Брокер повідомлень», який здійснює маршрутизацію, зазвичай гарантує доставку, розподіл потоків даних, підписку на потрібні типи повідомлень.  
Ідея AMQP була розроблена в 2003 році Джон О'Хара з JPMorgan Chase в Лондоні. JPMorgan Chase уклав контракт з iMatix Corporation, з середини 2004 до середини 2006 року, на розробку C-брокера і його протоколу. У 2005 році JPMorgan Chase звернувся до інших фірм, щоб сформувати робочу групу, яка включала Cisco Systems, Айона Technologies, iMatix, Red Hat і Transaction Workflow Innovation Standards Team (TWIST). У тому ж році JPMorgan Chase укладає партнерство з Red Hat, для створення Apache Qpid, спочатку на Java, а потім і на C++.

**Принцип роботи AMQP**

«Повідомлення» (message) відправляються в «точку обміну». Точка обміну розподіляє повідомлення в одну або кілька «черг». При цьому в точці обміну повідомлення не зберігаються. Точки обміну бувають трьох типів:

* розгалуження - повідомлення передається в усі причеплені до неї черги ;
* прямі - повідомлення передається в чергу з ім'ям, що збігається з ключем маршрутизації (routing key), який вказується при відправці повідомлення ;
* теми (topic ) - щось середнє між розгалуженням і прямим типом, повідомлення передається в черзі, для якої збігається [маска](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%B8%D0%B9_%D0%B2%D0%B8%D1%80%D0%B0%D0%B7) на ключ маршрутизації.

Повідомлення зберігаються в чергах доти, поки не буде забрано клієнтом. Клієнт завжди забирає повідомлення з однієї або декількох черг.

**RabbitMQ**

Однією з найвідоміших систем, що реалізують AMQP є така системи, як RabbitMQ.

RabbitMQ - платформа, яка реалізує систему обміну повідомленнями між компонентами програмної системи (Message Oriented Middleware) на основі стандарту AMQP (Advanced Message Queuing Protocol) [3]. RabbitMQ випускається під Mozilla Public License. RabbitMQ створений на основі випробуваної Open Telecom Platform, що забезпечує високу надійність і продуктивність промислового рівня і написана на мові Erlang.

RabbitMQ складається з:

* RabbitMQ Сервера
* Підтримка протоколи HTTP, XMPP і STOMP
* Клієнтських бібліотек AMQP для Java і .NET Framework
* Різних плагінів (плагін моніторингу та управління через HTTP або веб-інтерфейс, або плагін «Shovel» для передачі повідомлень між брокерами)

Підтримується горизонтальне масштабування для побудови кластерної архітектури.

Як движок бази даних, для зберігання повідомлень, використовується Mnesia.

**Основні поняття RabbitMQ**

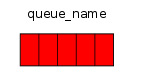
Producing – процес надсилання повідомлень.

Producer – програма, що надсилає повідомлення.



Producer

Queue– черга, яка живе в середині RabbitMQ. Черзі обов'язково повинно бути просвоєнно ім'я (queue\_name).



Queue

Consumer – програма, яка очікує та приймає повідомлення.



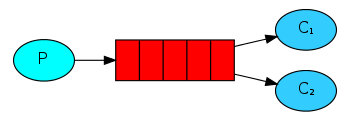
Consumer

Стандартна архітектура системи на базі RabbitMQ, виглядає наступним чином



Producer надсилає повідомлення в чергу. Consumer слухає чергу і отримує повідомлення з неї.

Зазвичай в реальних проектах використовується наступна архітектура



Producer надсилає повідомлення в чергу. Consumer1, Consumer2 і ConsumerN слухають чергу і один з них, або всі отримають повідомлення. Який з Consumer отримає повідомлення залежить від налаштування RabbitMQ, варіцій таких налаштувань надзвичайно багато.

**Протокол транспортного рівня TCP**

Протокол TCP реалізує потокову модель передачі інформації, хоча в його основі, як і в основі протоколу UDP, лежить обмін інформацією через пакети даних. Він являє собою орієнтований на встановлення логічного зв'язку (орієнтований на з'єднання), надійний (забезпечує перевірка контрольних сум, передача підтвердження в разі правильного прийому повідомлення, повторна передачі пакета даних в разі неотримання підтвердження протягом певного проміжку часу, правильна послідовність отримання інформації, повний контроль швидкості передачі даних) двобічний спосіб зв'язку між процесами в мережі.

**TCP-сегмент**

Блок даних TCP називається сегментом, хоча часто також використовують слово пакет, але таке вживання може вносити плутанину з IP-пакетом.

TCP-сегмент складається із TCP-заголовка і поля Дані (Data), яке називають сегментом даних або пейлодом або SDU.

Стандартний розмір TCP-заголовка — 20 байт, але з використанням опцій розмір може зростати до 60 байт. Як правило, опціями хости обмінюються на етапі встановлення з'єднання.

Розмір сегменту даних (поля даних) визначається опцією MSS (Максимальний розмір сегменту, Maximum segment size) на етапі встановлення з'єднання. Якщо обміну опціями не відбулося, то розмір сегменту даних встановлюється за замовчуванням 536 байт.[9] Розмір сегменту даних тісно пов'язаний з MTU (Максимальний блок передачі). Фактично MSS дорівнює MTU з відніманням розміру IP- і TCP-заголовків. Наприклад у сучасній мережі Ethernet MTU дорівнює 1500 байт; тоді оптимальний розмір MSS буде 1460 байтів (1500 мінус 20 байт заголовка IP і 20 байт заголовка TCP).



**WebSocket**

WebSocket — це протокол, що призначений для обміну інформацією між браузером та веб-сервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. WebSocket спроектовано для втілення у веб-браузерах та веб-серверах, але може також використовуватись будь-якою клієнт-серверною системою. Прикладний програмний інтерфейс WebSocket був стандартизований W3C, крім того протокол WebSocket стандартизований IETF як RFC 6455

Перед встановленням з’єднання поверх TCP, відбувається «рукопотискання» (handshake), поверх HTTP:

GET /ws HTTP/1.1

Host: pmx

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Version: 6

Sec-WebSocket-Origin: [http://pmx](http://pmx/)

Sec-WebSocket-Extensions: deflate-stream

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Відповідь сервера

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=

Клієнт надсилає ключ Sec-WebSocket-Key, що є випадковим значенням, закодованим з використанням base64. Щоб сформувати відповідь, до закодованого ключа додають рядок [GUID](https://uk.wikipedia.org/wiki/GUID) 258EAFA5-E914-47DA-95CA-C5AB0DC85B11, а потім (не розкодовуючи ключ) хешують через [SHA1](https://uk.wikipedia.org/wiki/SHA1) та base64. І на завершення, результуючу відповідь вкладають в заголовок Sec-WebSocket-Accept.

WebSocket підтримують такі браузери:

* Google Chrome (починаючи з версії 4.0.249.0);
* Apple Safari (починаючи з версії 5.0.7533.16);
* Mozilla Firefox (починаючи з версії 4);
* Opera (починаючи з версії 10.70 9067);

**Електронна пошта**

Електро́нна по́шта або е-пошта — спосіб обміну цифровими повідомленнями між людьми використовуючи цифрові пристрої, такі як комп'ютери та мобільні телефони, що робить можливим пересилання даних будь-якого змісту (текстові документи, аудіо-, відеофайли, архіви, програми).

Електронна пошта схожа на звичайну пошту. Звичайний лист складається із конверта, на якому зазначена адреса отримувача і стоять штампи поштових відділень шляху слідування, та вмісту — власне листа. Електронний лист складається із заголовків, які містять службову інформацію (про автора листа, отримувача, шлях проходження листа), які служать, умовно кажучи, конвертом, та власне вміст самого листа. За аналогією зі звичайним листом, відповідним методом можна внести в електронний лист інформацію якого-небудь іншого роду, наприклад, фотографію тощо. Як і у звичайному листі можна поставити свій підпис. Звичайний лист може не дійти до адресата або дійти з запізненням, — аналогічно і електронний лист. Звичайний лист доволі дешевий, а електронна пошта — найдешевший вид зв'язку.

Адреса містить ім’я комп’ютера та ім’я конкретного користувача цього комп’ютера, якому адресоване повідомлення.

Електронні скриньки розміщуються на поштових серверах. Для кожної скриньки на поштовому сервері відводиться спеціальне місце. На одному поштовому сервері не може бути двох скриньок з однаковими назвами.

Електронна адреса складається з двох частин, розділених знаком @ Наприклад:

**alexander.mykulych@gmail.com**

alexander.mykulych – ім’я скриньки, gmail.com – доменне ім’я поштового сервера, на якому знаходиться ця скринька. Повідомлення, що надіслані за цією адресою, будуть передані на комп’ютер з іменем gmail.com користувачу на ім’я alexander.mykulych.

**Протоколи передачі електронної пошти**

* IMAP
* POP3
* SMTP
* UUCP

**Веб-сервіс (Web Service)**

Веб-сервіс – програмна система, яка ідентифікується веб-адресою, з стандартизованим інтерфесом.

Веб-сервіси можуть взаємодіяти один з одним і зі сторонніми додатками за допомогою повідомлень, заснованих на певних протоколах (SOAP, XML-RPC, REST і т.д.). Веб-служба є одиницею модульності при використанні сервіс-орієнтованої архітектури.

**Telegram**

Telegram — програмне забезпечення для смартфонів на базі Android та iOS, яке дозволяє обмінюватися текстовими повідомленнями та різноманітними файлами, зокрема графічними файлами та відеофайлами, а також безкоштовно телефонувати іншим користувачам програми.

Підтримує німецьку, англійську, іспанську, італійську, нідерландську, португальську, арабську та китайську. Українська мова не доступна за замовчанням, але її можна встановити власноруч. Додаток має версії для ПК з операційними системами Linux, Windows та OS X.

Обліковий запис користувача прив'язуються до номера мобільного телефона: щоб авторизуватися він повинен ввести код авторизації з отриманного СМС-повідомлення. Надісланий код має обмежений термін придатності, після спливання якого код стає неактуальним. Таким чином, користувач позбавляється необхідності запам'ятовувати чи зберігати десь свій пароль.

Станом на вересень 2016 року месенджером «Telegram» користуються 100 мільйонів осіб.

За допомогою спеціального API сторонні розробники можуть створювати «ботів», спеціальні акаунти, керовані програмами. Типові боти відповідають на спеціальні команди в персональних і групових чатах, також вони можуть здійснювати пошук в інтернеті або виконувати інші завдання, застосовуються задля розваг або в бізнесі.

**Elasticsearch**

Elasticsearch — вільне програмне забезпечення, пошуковий сервер, розроблений на базі Lucene. Надає розподілений, мультиарендний повнотекстовий пошуковий рушій з HTTP веб-інтерфейсом і підтримкою безсхемних JSON документів.

Є найбільш популярним пошуковим рушієм, випереджаючи Apache Solr. Розробляється на Java і випускається під ліцензією Apache License.

Elasticsearch може використовуватись для індексування та пошуку будь-яких типів документів. Він надає масштабовний пошук, має пошук близький до реального часу і підтримку мультиарендності.

Elasticsearch має можливість розподілення, індекси можуть бути розділені по шардах, при чому кожен шард може мати нуль чи більше реплік. Кожен вузол містить один чи більше шардів і діє як координатор делегування операцій на потрібний шард. Балансування та маршрутизація виконується автоматично.

Приклад використання

* Реалізація пошуку по веб-сайту, наприклад пошук товарів в інтернет-магазині. В цьому випадку Elasticsearch індексує каталоги товарів, та надає можливості пошуку та припущення щодо автозаповнення.
* Зберігання журналів подій чи транзакцій, аналізування і добування даних для отримання тенденцій, статистик, висновків, аномалій. В даному випадку можна використовувати Logstash для збору, об'єднання, аналізу даних, і потім перенаправляти ці дані в Elasticsearch для подальшого опрацювання.
* Для розробки, наприклад, платформи по ціновому оповіщенню, що дозволяє досвіченим в цій сфері користувачам визначати правила типу «Я зацікавлений в придбанні електронного ґаджету XXX і я хочу бути оповіщеним якщо ціна у будь-якого постачальника впаде нижче XXX протягом наступного місяця». В даному випадку можна збирати ціни, індексувати їх в Elasticsearch і використовувати функцію зворотнього пошуку: зіставляти коливання цін з запитом користувача і при відповідності до запиту надсилати сповіщення.
* Для впровадження аналітики/Бізнес-аналітики в проект, коли треба швидко досліджувати, аналізувати, візуалізувати надзвичайно великі об'єми даних (мільйони чи мільярди записів). В даному випадку доцільно використання Elasticsearch для збереження даних і Kibana для побудови користувацьких панелей відображення і візуалізації необхідних аспектів. До того ж, можна використовувати агрегаційні функції Elasticsearch для здійснення комплексної бізнес-аналітики даних.

**Kibana**

Kibana - це система з відкритим кодом, для візуалізації даних, плагін для Elasticsearch. Він забезпечує можливість візуалізації даних, які індексуються на кластері Elasticsearch.

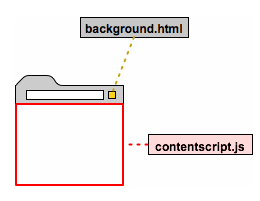
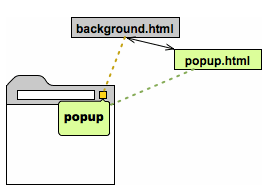
**Розширення Google Chrome**

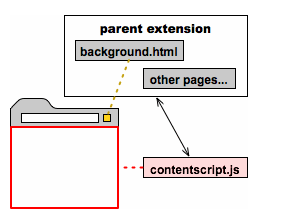
Розширення Google Chrome - це розширення браузера, що змінюють та модифікують його роботу. Розширення написані з використанням веб-технологій, таких як HTML, JavaScript і CSS.

В фізичному представленні розширення – це архів файлів: HTML, CSS, JavaScript, зображень та конфігураційних файлів. Розширення можуть використовувати всі інтерфейси API, що браузер надає для веб-сторінок, від XMLHttpRequest та JSON до HTML5.

В браузері Chrome розширення мають наступну архітектуру:

1. background page
2. UI pages (popup.html)
3. Content scripts
4. Other pages





Background Page

Html сторінка, яка містить JavaScript код, для контролю логіки розширення. Зазвичай вона має назву background.html і ділиться на два типи: «повністю фонова сторінка» (persistent background page) та «сторінка подій» (event page). Різниця між типами полягає в тому, що «persistent background page» завжди знаходиться у відкритому режимі, а «event page» переходить у потрібні режими тільки при певних подіях, таких як: натискання іконки розширення, чи завершення роботи браузера.

UI pages (popup.html)

Сюди входять звичайні html-сторінки, які містить розширення. Прикладом є сторінка, яка появляється при активації розширення (popup.html). Таких сторінок може бути необмежена кількість. Їх завдання надавати користувачу інтерфейс для роботи з функціями розширення або надавати певну інформацію, кожна сторінка має можливість спілкуватись з фоновою сторінкою (background page), передаючи їй потрібну інформацію, та отримуючи відповідь.

Content scripts

До даного типу відноситься js-файл, який забезпечує інтерактивність з веб-сторінкою, яка відкрита в браузері. Браузер забезпечує завантаження даного файла в контексті основної веб-сторінки після чого файл має можливість довантажувати інші файли, модифікувати сторінку та файли, які забезпечують роботу сторінки.

Other pages

До цього типу відносяться усі html-файли, які використовуються фоновою сторінкою чи контент-скріптом.

**Chrome DevTools Protocol**

Chrome DevTools Protocol дозволяє додатку оперувати, досліджувати, дебажити і профілювати Chrome, Chromium і інші Blink-based браузери.

Інструментарій Chrome DevTools Protocol включає в себе певну кількість доменів (DOM, Debugger, Network etc.). Домен – це інструментальна одиниця, яка дозволяє отримувати інформацію про певний аспект роботи браузера в реальному часі. Кожен домен визначає методи і події, які генерують ці методи. Методи і події подаються в серіалізованому представлені JSON-об’єктів, з фіксованою структурою.

Для даної роботи детальніше розглянемо домен – Network, який дає дозволяє відслідковувати мережеву активність веб-сторінки, та надає інформацію про http, файли, дані і інші запити, та відповіді, їхні заголовки і тіла, час і т.п.

Основні методи, які надає домен:

|  |  |
| --- | --- |
| Network.enable | Дозволяє відстежувати мережу, мережеві події будуть відправлені клієнту |
| Network.disable | Вимикає відслідковування активності в мережі, запобігає відправці мережевих подій клієнту. |

Події, які відповідають методу Network.enable:

|  |  |
| --- | --- |
| Network. resourceChangedPriority | Активується, якщо при завантаженні ресурсів пріоритет змінюється |
| Network. requestWillBeSent | Активуэться, коли сторінка збирається послати запит HTTP |
| Network. requestServedFromCache | Активується, якщо запит закінчив завантаження з кеша |
| Network. responseReceived | Активується, коли HTTP відповідь доступна |
| Network. loadingFinished | Виникає, коли запит HTTP завершить завантаження |
| Network. loadingFailed | Виникає, коли запит HTTP не вдалося завантажити |
| Network. webSocketWillSendHandshakeRequest | Виникає, коли WebSocket збирається ініціювати рукопотискання |
| Network. webSocketHandshakeResponseReceived | Активується, коли відповідь рукопотискання закінчено успішно |
| Network. webSocketClosed | Активується, після створення WebSocket |