НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2015 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.050102, 8.050102 Інформаційні технології проектування
(код та назва спеціальності)

на тему: Композитні веб-додатки в наукових дослідженнях

Виконав : студент 4 курсу, групи ДА-11
(шифр групи)

Савельєв Юрій Дмитрович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник к.т.н., доц. Корначевський Я.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант охорона праці доц. Гусєв А.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент д.т.н., проф. Бідюк П.І. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст. викладач Британов О.А.

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2015 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший (Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.050102, 8.050102 Інформаційні технології проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2015 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Савельєву Юрію Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Композитні веб-додатки в наукових дослідженнях
керівник проекту (роботи) Корначевський Ярослав Ілліч, к.т.н., доцент ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» квітня 2015 р. №30/1-ст

2. Строк подання студентом проекту (роботи) 08.06.2015

3. Вихідні дані до проекту (роботи)
SOA
Mashup
Веб-сервіс
Веб-додаток
API
PHP
JavaScript

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Дослідити SOA для побудови композитних веб-додатків.
2. Ознайомитися з концепцією композитних веб-додатків

3. Аналіз технічних проблем концепції
4. Формулювання вимог до веб-додатку
5. Аналіз існуючих композитних веб-додатків
6. Формування ідеї власного веб-додатку
7. Вибір та огляд джерел інформації
8. Розробка власного композитного веб-додатку
9. Зробити загальні висновки по концепції, перевагам та недолікам

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Концепція SOA - плакат
2. Концепція Mashup - плакат
3. Діаграма прецедентів - плакат
4. Приклад роботи програми – плакат
5. Презентація роботи – презентація MS PP

6. Консультанти розділів проекту (роботи)*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Гусєв А.М., доцент		

7. Дата видачі завдання 01.02.2015

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2015	
2	Збір інформації	15.02.2015	
3	Дослідження SOA	28.02.2015	
4	Дослідження процесів композиції веб-сервісів	10.03.2015	
5	Пошук API для композиції	15.03.2015	
6	Робота з WolramAlpha API	25.03.2015	
7	Робота з Google Charts API	25.04.2015	
8	Створення PHP програми	30.04.2015	
9	Оформлення дипломної роботи	31.05.2015	
10	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2015	

Студент _____ Ю.Д. Савельєв
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ Я.І.Корначевський

* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

бакалаврської дипломної роботи Савельєва Юрія Дмитровича
на тему: “Композитні веб-додатки в наукових дослідженнях”

Дана дипломна робота присвячена дослідженню технологій створення композитних веб-додатків та практичній розробці композитного веб-додатку для застосування в наукових дослідженнях.

У роботі була досліджена сервіс-орієнтована архітектура, що лежить в основі композитних веб-додатків, проведено аналіз доступних технологій, що є інструментами для їх створення, та можливих перешкод при компонуванні різних сервісів. Сформульовано основні вимоги, яким має задовольняти створений додаток.

Розроблено композитний веб-додаток, що аналізує числову вибірку, будує її гістограму, шукає значення медіани, дисперсії та середнього значення.

Робота складається з 98 сторінок, 14 рисунків, 4 таблиці, 39 посилань, 1 додатка на 16 сторінок.

Ключові слова: Композитні веб-додатки, mashup, SOA, API, PHP, JavaScript.

ANNOTATION

to the bachelor thesis of Saveliev Yurii Dmytrovich
on “Mashup web-applications in scientific researches ”

This graduate work is addicted to research of technologies of designing mashup web-applications and practical development of creating mashup web-application for scientific research.

The paper researches service-oriented architecture that is fundamental for mashup web-applications, analyzes available technologies, which are instruments for its construction, and potential challenges during compositing different web-services. In addition, it was formulated main requirements for created web-application

It was designed mashup web-application for analyzing sample of numbers, building its histogram, and getting results of mean, median and sample standard deviation.

Work consists of 98 pages, 14 images, 4 tables, 39 references and 1 addition for 16 pages.

Keywords: Mashup, web-applications, SOA, API, PHP, JavaScript.

ЗМІСТ

Перелік умовних позначень, символів, скорочень і термінів

Mashup – композитний веб-додаток.

SOA - сервіс орієнтована архітектура.

Web 2.0 – етап розвитку веб-технологій та Інтернету.

API – прикладний програмний інтерфейс.

REST – Representational State Transfer, архітектурний стиль програмного забезпечення для розподілу систем, таких як World Wide Web.

ПЗ – програмне забезпечення

ЕОМ – електронна обчислювальна машина

Веб-сервіс – та що ідентифікується веб-адресом програмна система з стандартизованими інтерфейсами

ГОСТ – «государственный стандарт» (міждержавний стандарт СНД)

ДСТУ – Державний Стандарт України

МОН – Міністерство освіти і науки України

НАНУ – Національна академія наук України

НТУУ «КПІ» – Національний технічний університет України «Київський політехнічний інститут»

ВСТУП

Оцінка сучасного стану проблеми

Важливою складовою будь-яких наукових досліджень є інструментарій, за допомогою якого вони здійснюються. Для різних обчислень, моделювань, побудови графіків та іншого створено велику кількість програмного забезпечення, яке здатне вирішити будь-які питання. Але основною проблемою є його ціна, складність у використанні, навантаження на ПК, надмірний функціонал. Тому логічним рішенням для вирішення малих задач є використання веб-додатків через їх зручність, доступність з будь-якого пристрою. Але сервісів для таких цілей поки що мало, оскільки технології, що дозволяють їх проектувати з'явились не так давно, а кількість потенціальних користувачів - невелика.

Актуальність

Останнім часом веб-додатки витісняють інстальоване ПЗ, оскільки вони доступні з будь-якого комп'ютера, швидкість Інтернету дозволяє працювати без зайвих проблем, для їх функціонування потрібен лише браузер та вони не створюють ніякого навантаження на пристрій користувача, тому що всі вираховування проводяться на сервері.

Останнє десятиліття – це новий етап розвитку веб-технологій та Інтернету під назвою Web 2.0. Однією з його рис є поява композитних веб-додатків, які інакше називаються mashup. Такий підхід до побудови додатків став можливим завдяки сервіс-орієнтованій архітектурі програмного забезпечення. Mashup використовує інструментарії двох або більше сервісів та, використовуючи їх функціональність, створює новий. Це мало великий поштовх до розвитку різних ресурсів, зокрема соціального напрямку.

В наукових дослідженнях такий підхід застосовується не часто, тому що має малу цільову аудиторію, хоча має великий потенціал. Зазвичай, розробка програмного забезпечення для науки займає більше часу, оскільки потребує реалізації складніших алгоритмів. Саме тому розробка окремих незалежних

компонентів з відкритим програмним інтерфейсом та поєднання їх у mashup має великий потенціал для використання у наукових дослідженнях, такий підхід дозволяє легше розробити складний сервіс.

Мета

Мета даної роботи - аналіз методів та принципів побудови композитних веб-додатків та їх застосування у наукових дослідженнях. В роботі були поставлені такі задачі:

- Вивчення та аналіз mashup концепції побудови веб-додатків, її переваги та недоліки.
- Дослідження засобів, технологій та принципів для побудови композитних веб-додатків.
- Аналіз існуючих композитних веб-додатків. Пошук оптимальних і зручних рішень для побудови власного сервісу.
- Пошук та вибір сервісів для компонування. Ознайомлення з їх API.
- Проектування власного веб-додатку на основі мети та задач зазначених вище.
- Створення композитного веб-додатку
- Аналіз створеного рішення.

1. SOA

1.1 Проблеми інтеграції в сучасних системах обробки даних і історичні шляхи їх вирішення

Протягом всієї історії інформаційних технологій однією з найважливіших проблем була боротьба за повторне використання програмного коду. Першим засобом, що реалізує ідею повторно використання, були процедури і функції - спочатку вбудовані, а потім оформленні як модулі. Модуль являє собою одиницю програмного коду, що окремо компілюється, окремо зберігається та має власний інтерфейс. Інтеграція модулів у єдиний додаток реалізується завдяки механізмам виклику процедур в мовах програмування і механізмам збірки в завантажувачах і редакторах зв'язків. Розробка модулів, реалізує типові технологічні і прикладні функції, що значно полегшує роботу програміста, зменшує обсяг коду і прискорює процес розробки та впровадження програмного продукту, що є важливою і конкурентоздатною перевагою. Розвинені бібліотеки модулів, що надають майже вичерпний набір засобів для вирішення певних класів технологічних або прикладних задач, отримали назву каркасів. Застосування каркасів в ідеалі перетворює створення нового додатка в збірку програми з готових елементів каркаса.[1]

Наступним значним етапом боротьби за повторне використання коду можна вважати появу компонентної архітектури. Компонента - це модуль, який реалізує стандартний інтерфейс. Стандартизація інтерфейсу забезпечує взаємозамінність компонентів і, що найважливіше, взаємодію компонент з проміжним програмним забезпеченням - каркасами, інструментальними засобами розробки, контейнерами. Останнє кардинальним чином змінює процеси розробки та функціонування додатків, значно прискорюючи їх створення та впровадження. Контейнер являє собою деяке проміжне програмне забезпечення, яке підтримує виконання компонентів. Кажуть, що, реалізуючи стандартний інтерфейс, компонент укладає контракт з контейнером: компонент

«зобов'язується» забезпечувати деяку певну поведінку (виконує стандартні методи), а контейнер «за це» надає йому підтримку низького рівня, наприклад, управління життєвим циклом компоненти (певна послідовність виклику її методів), безпека, зв'язок між компонентами, управління транзакціями тощо. Виконання контейнером технологічних функцій низького рівня позбавляє програміста від їх реалізації і дозволяє йому зосередити свої зусилля на прикладній задачі. Особливе значення контрактні відносини між компонентами і контейнером набувають в технологіях розподілених додатків. Частини розподіленого додатку, що виконуються в різних вузлах розподіленої системи, є компонентами, і їх функціонування і взаємодія, що підтримується відповідним контейнером. Інтеграційну функцію, таким чином, виконує контейнер, що підтримує той чи інший тип контракту. Найбільш розвиненими платформами розподілених додатків є Microsoft .NET і Java 2 Enterprise Edition (J2EE). Остання базується на відкритих стандартах і не залежить від апаратної платформи і операційного середовища. Основним виробником засобів технології .NET є фірма Microsoft. Технологія J2EE має більший спектр виробників, в силу відкритості своїх стандартів. Лідерами є фірми Oracle, IBM, Sun, Red Hat, багато засобів цієї технології поширюються вільно або мають відкритий код [1].

Хоча компонентна архітектура забезпечує швидку і зручну інтеграцію компонентів, це стосується тільки компонентів, які були розроблені у відповідності зі специфікаціями даної платформи. Однак сучасні вимоги ринку надзвичайно посилюють вимоги до термінів створення і впровадження систем і додатків інформаційних технологій. Починаючи з середини 1990-х років відзначається безліч випадків, коли розробка і реалізація проекту інформаційної системи «з нуля» закінчувалася крахом, тому що, по-перше, витрати на розробку не виправдовували себе, по-друге, до моменту завершення проекту умови, на підставі яких були складені специфікації проекту, істотно змінювалися. Необхідність швидкого реагування підприємств на мінливі умови привели фірму IBM до концепції бізнесу за вимогою. За визначенням IBM, бізнес за вимогою (on demand business) - це підприємство, бізнес-процеси якого щільно

інтегровані всередині всієї компанії, а також з бізнес-процесами партнерів, постачальників, споживачів, що дає підприємству можливість гнучко і швидко реагувати на мінливі вимоги покупців, зміни ринку і зовнішні впливи. Інформаційні системи, що підтримують бізнес за вимогою, повинні швидко відповідати вимогам того бізнесу, який вони обслуговують. Інтеграція процесів усередині підприємства має на увазі, що окремі процеси підприємства вже автоматизовані, в розробці яких, могли використовуватися різні мови програмування, операційні системи та платформи проміжного програмного забезпечення. Розробка всіх або деякої частини додатків наново відповідно до єдиної прийнятої платформи є занадто довгою і дорогою справою. Підприємство має максимально використовувати наявні активи інформаційних технологій, щоб знизити витрати і зменшити терміни впровадження. Також при розробці нових або модифікації наявних додатків доцільно максимально використовувати вже існуючі компоненти. При інтеграції з інформаційними системами партнерів, постачальників, споживачів тощо, приведення всіх компонентів системи до єдиній платформи просто неможливе. Необхідність вирішення цих проблем заснувала концепцію сервісно-орієнтованої архітектури [1].

Програмні засоби, що реалізують компоненти бізнес-процесів, оформляються як сервіси. Сервіс - це програмний компонент, який реалізує закінчену функцію надання або обробки даних, що переводить їх з одного цілісного стану в інший. Основною відмінністю сервісу від звичайної компоненти є стандартний і переносний незалежний інтерфейс. Клієнт, що звертається до сервісу, не зобов'язаний нічого знати про подробиці реалізації сервісу: якою мовою і якою моделі програмування він створений, на яких апаратних засобах, в якому операційному середовищі, на якій платформі проміжного програмного забезпечення він виконується. Сервісно-орієнтована архітектура, таким чином, дозволяє компонувати бізнес-процеси з компонентів, виконаних на різних платформах (корпоративних J2EE, компонентів .NET, окремих додатків), представляти у вигляді сервісів і повторно використовувати в нових бізнес-процесах успадковані компоненти. В останньому випадку

успадкуванні компоненти не змінюються, але для них робиться оболонка, що адаптує їх інтерфейси до стандартів сервісів [1].

Необхідно звернути увагу на те, що оформлення процесу як сервісу не має будь-яких вимог до розробки процесу, а лише вимоги до його інтерфейсу. Щоб успішно виконувати свої інтегрують функції, платформа, яка втілює сервісно-орієнтовану архітектуру, повинна відповідати таким вимогам:

- забезпечувати специфікації інтерфейсу, які були б прийняті, якщо не всіма, то більшістю розробників компонентів;
- використовувати загальноприйняті протоколи для взаємодії сервісів і клієнтів;
- не використовувати в інтерфейсі які-небудь складні та / або закриті формати представлення інформації;
- не вимагати для своєї підтримки дорогого або ресурсномісткого програмного забезпечення [1].

1.2 Визначення SOA

Існує багато поглядів на концепцію SOA. У найбільш простому вигляді SOA - це підхід до побудови програмних систем, який концентрується на тому, *як* програмне забезпечення створюється [2]. Нижче представлені поширені приклади визначення SOA. Вони обрані з різних джерел і цікаві тим, що ілюструють різні уявлення і погляди на те, що таке SOA, проте всі вони разом дають загальний зміст:

Бізнес-визначення. Набір бізнес-методів, методів процесу, організаційних методів, методів управління, призначених для зменшення або усунення можливості невиконання інформаційними технологіями своїх функцій і для кількісної зміни цінності IT для бізнесу в процесі створення гнучкого бізнес-середовища для отримання конкурентоздатності .

Ще одне бізнес-визначення (за IBM). SOA пропонує можливість зручної роботи з елементами бізнес-процесів і IT-інфраструктурою (що лежить в їх основі) як з безпечними, стандартизованими компонентами (службами), які

можна використовувати багаторазово і комбінувати при зміні пріоритетів бізнесу.

Найширше технічне визначення. IT-архітектура в масштабі підприємства, яка забезпечує слабкі зв'язки, багаторазове використання і взаємодію між системами.

Досить складне технічне визначення. Архітектура додатків, в якій всі функції і служби визначені за допомогою мови програмування і мають інтерфейси виклику, звернення до яких здійснюються при виконанні бізнес-процесів. Кожна взаємодія є незалежною від усіх інших взаємодій і внутрішніх протоколів, що забезпечують з'єднання пристроїв. Оскільки інтерфейси незалежні від платформи, клієнт може використовувати службу, застосовуючи будь-який пристрій, використовуючи будь-яку операційну систему і будь-яку мову програмування.

Визначення - найменший спільний знаменник. Системна архітектура, в якій функції додатків створюються у формі компонентів (служб), які мають слабкі зв'язки і чітко визначені з метою сумісності, підвищення гнучкості і можливості багаторазового використання.

Спеціалізоване визначення. SOA - це синонім таких архітектурних рішень, які використовують технології веб-служб, такі як SOAP, WSDL і UDDI [3].

1.3 Концепції SOA

1.3.1 Сервіс

Сервісні компоненти (чи сервіси) описуються програмними компонентами, що забезпечують прозору мережеву адресацію. Описуючи процеси в інформаційних системах дотримуватимемося визначення, що сервісами називаються відкриті, самовизначальні компоненти, що підтримують швидку побудову розподілених застосувань.

При цьому немає чіткого визначення, який об'єм послуг повинен надавати окремий сервіс, а ці послуги в мережах можуть розрізнятися наступним чином:

Програмне забезпечення як послуга (SaaS, англ. Software-as-a-Service) – модель, в якій споживачеві надається можливість використання прикладного програмного забезпечення провайдера, що працює в хмарній інфраструктурі і доступного з різних клієнтських пристроїв або за допомогою тонкого клієнта, наприклад, з браузеру або інтерфейс програми. Контроль і управління основною фізичною і віртуальною інфраструктурою хмари, у тому числі мережі, серверів, операційних систем, зберігання, або навіть індивідуальних можливостей додатка (за винятком обмеженого набору призначених для користувача налаштувань конфігурації додатка) здійснюється хмарним провайдером [4].

Платформа як послуга (PaaS, англ. Platform-as-a-Service) – модель, коли споживачеві надається можливість використання хмарної інфраструктури для розміщення базового програмного забезпечення для наступного розміщення на нім нових або існуючих застосувань (власних, розроблених на замовлення або придбаних тиражованих застосувань). До складу таких платформ входять інструментальні засоби створення, тестування і виконання прикладного програмного забезпечення – системи управління базами даних, єднальне програмне забезпечення, середовища виконання мов програмування - надаються хмарним провайдером. Контроль і управління основною фізичною і віртуальною інфраструктурою хмари, у тому числі мережі, серверів, операційних систем, зберігання здійснюється хмарним провайдером, за винятком розроблених або встановлених застосувань, а також, по можливості, параметрів конфігурації середовища (платформи) [5].

Інфраструктура як послуга (IaaS, англ. IaaS or Infrastructure-as-a-Service) – надається як можливість використання хмарної інфраструктури для самостійного управління ресурсами обробки, зберігання, мереж і іншими фундаментальними обчислювальними ресурсами, наприклад, споживач може встановлювати і запускати довільне програмне забезпечення, яке може включати операційні системи, платформне і прикладне програмне забезпечення. Споживач може контролювати операційні системи, віртуальні системи зберігання даних і встановлені застосування, а також обмежений контроль набору доступних

сервісів (наприклад, міжмережевий екран, DNS). Контроль і управління основною фізичною і віртуальною інфраструктурою хмари, у тому числі мережі, серверів, типів використовуваних операційних систем, систем зберігання здійснюється хмарним провайдером [6]/

Необхідно враховувати, що при реалізації дрібномодульного сервісу для отримання бажаного результату необхідно забезпечити координовану роботу декількох сервісів. В цьому випадку можна надавати елементарний об'єм функціонального навантаження і забезпечувати високу міру повторного використання. А використання крупномодульних сервісів, дозволяє забезпечити хорошу інкапсуляцію функціональності, але ускладнює повторне використання цих вузькоспеціалізованих сервісів.

Інтерфейс забезпечує опис можливостей і якості послуг, що надаються, конкретним сервісом. У інтерфейсі визначається формат повідомлень, використовуваний для обміну інформацією, а також вхідні і вихідні параметри методів, підтримуваних сервісним компонентом. Від вибору мови і способу опису інтерфейсу залежать можливості програмної сумісності різних реалізацій сервіс-орієнтованої архітектури.

Транспорт забезпечує обмін інформацією між окремими сервісними компонентами. Разом з відкритими стандартами опису інтерфейсів, використання гнучких транспортних протоколів для обміну інформацією між сервісними компонентами дозволяє підвищити програмну сумісність сервіс-орієнтованої системи.

Регістри сервісів використовуються для пошуку сервісних компонентів, що забезпечують необхідну функціональність. Серед усієї множини різних систем, що забезпечують виявлення сервісів [7], можна виділити дві основні категорії: системи динамічного виявлення і системи статичного виявлення.

Статичні системи виявлення сервісів (наприклад, UDDI) орієнтовані на зберігання інформації про сервіси в системах, що рідко змінюються.

Динамічні системи виявлення сервісів [8] орієнтовані на системи, в яких допустима часта поява і зникнення сервісних компонентів. На сьогодні веб-

сервіси найчастіше використовуються для реалізації сервіс-орієнтованої архітектури. Веб-сервіси розроблені для підтримки взаємодії між розподіленими системами за допомогою обчислювальної мережі.

1.3.2 Базова архітектура SOA

Архітектура SOA має форму піраміди, що складається з кількох шарів.

1. Підґрунтям піраміди є базові сервіси і базові операції, а саме, публікація, виявлення, вибір і зв'язування, які націлені на створення і використання описів сервісів.

2. Шар композиції – це консолідація багатьох функціональних сервісів у єдиний складений сервіс, а саме, контроль виконання сервісів і керування потоками даних між ними; публікація подій вищого рівня шляхом фільтрації, підсумовування, кореляції подій компонентів; забезпечення цілісності сервісу та накладання обмежень на його компоненти; досягнення якості композиції сервісів, включаючи показники виконання, секретності, контролю доступу тощо.

3. Шар менеджменту сервісу, а саме, керування платформою сервісу, розгортання, ведення статистики виконання, підвищення ефективності, забезпечення прозорості ходу виконання транзакцій, відстеження стану ходу виконання тощо [9].

1.3.3 Веб-сервіс

Переважна форма реалізації сервісів – це веб-сервіси, які зберігаються та ідентифікуються за URL-адресами і взаємодіють між собою за допомогою мережі Інтернет шляхом віддалених викликів (Remote Procedure Call). Стрімке поширення Інтернету призвело до того, що традиційне єдине інтегроване підприємство минулих поколінь все частіше заміняється мережею бізнесів, які спільно виконують певні функції при тому, що і власність, і менеджмент розподілені між партнерами. Саме інформаційні потреби розподілених бізнесів викликали до життя веб-сервіси як адекватну форму компонентів типу КПВ.

Веб-сервіс має URL-адресу, інтерфейс і механізм взаємодії з іншим сервісом через протоколи Інтернету або зв'язки з іншими програмами, БД і діловими бізнес-операціями. Обмін даними між веб-сервісом і програмою здійснюється за допомогою XML-документів, оформлених у вигляді повідомлень. Веб-сервіси забезпечують розв’язання задачі інтеграції застосувань різної природи, будучи інструментом побудови розподілених систем. Веб-сервіс надається провайдером мережі Інтернету, який має стандартний спосіб взаємодії з розподіленими (.NET, J2EE, CORBA і ін.) і прикладними системами з отримання деякої послуги.

Основні засоби опису веб-сервісів:

- мова XML для опису і побудови SOA–архітектури;
- мова WSDL (веб Services Description Language) для опису веб-сервісів і їхніх інтерфейсів на XML, що стосується типів даних і повідомлень, а також моделей взаємодії і протоколів зв'язування сервісів між собою;
- SOAP (Simple Object Access Protocol) для визначення форматів запитів до веб-сервісів;
- UDDI (Universal Description, Discovery and Integration) для універсального опису, виявлення й інтеграції сервісів, забезпечення їхнього збереження, упорядкування ділової сервісної інформації в спеціальному реєстрі з покажчиками на конкретні інтерфейси веб-сервісів [9].

1.3.4 Концепція слабкого зв'язування

У парадигмі SOA концепція слабкого зв'язування проявляє себе таким чином:

- Вона допомагає організувати рівень абстракції між виробниками і споживачами сервісів.
- Вона сприяє реалізації гнучкості в зміні реалізації сервісів без впливу на споживачів сервісів.
- В архітектурі SOA функціональність організовується як набір модульних загальних сервісів, які використовуються повторно. Ці сервіси мають чітко визначені інтерфейси, що інкапсулюють ключові правила доступу до них. Вони також будуються без будь-яких припущень про те, хто буде використовувати або споживати ці сервіси. Таким чином, вони слабко пов'язані зі споживачами сервісів [3].

1.3.5 Реєстри сервісів

Реєстр сервісів – це каталог сервісів, доступних в системі SOA. Він містить фізичне розташування сервісів, версії і термін дії сервісів, документацію по сервісам і стратегії. Реєстр сервісів є одним з основних будівельних блоків архітектури SOA. Роль реєстру сервісів:

- Реєстр сервісів реалізує обіцяне архітектурою SOA слабке зв'язування. Зберігаючи місця розташування кінцевих точок сервісів, він усуває тісні зв'язки, що приводить до жорсткій прив'язці споживача до провайдера. Він також полегшує потенційні складності заміни однієї реалізації сервісу інший при необхідності.
- Реєстр сервісів добре масштабується; він розвивається відповідно до розвитку системи, яку обслуговує.
- Реєстр сервісів дозволяє системним аналітикам досліджувати корпоративний портфель бізнес-сервісів. Виходячи з цього вони

можуть визначити, які сервіси доступні для автоматизації процесів з метою задоволення актуальних бізнес-потреб, а які ні. Це в свою чергу дозволяє дізнатися, що потрібно реалізувати і додати в портфель, формуючи каталог доступних сервісів.

- Реєстр сервісів може виконувати функцію управління сервісами, зобов'язуючи підписують сервіси бути узгодженими. Це допомагає гарантувати цілісність керівництва сервісами і стратегій.
- Видимість доступних сервісів і їх інтерфейсів прискорює розробку, сприяє повторному використанню додатків, удосконалює керівництво і покращує бізнес-планування і управління. Відсутність реєстру сервісів призводить до надмірності і неефективності.
- Реєстри сервісів допомагають зменшити час, що витрачається на виявлення інформації про сервіс.
- Без реєстру, що стежить за сервісами та їх взаємовідносинами, SOAA середовище не тільки втрачає узгодженість і контроль, але і приходить в стан хаос [3].

Компанії IBM, Microsoft та Ariba створили власні UDDI-реєстри (веб-реєстри), де розробники можуть реєструвати свої Веб-сервіс и. Інформація в UDDI-реєстрах складається з трьох компонентів:

- ***"білі сторінки"*** дозволяють підприємствам реєструвати їх назви і послуги, що забезпечує пошук іншими компаніями згідно з довідниками, які містять їх адресу та інші ідентифікатори;
- ***"жовті сторінки"*** включають класифікатори за галузями та специфікують компанії способами: NAICS-кодами стандартів промисловості, що встановлені американським урядом, кодами Організації Об'єднаних Націй - SPSC-кодуванням та кодами географічного положення;
- ***"зелені сторінки"*** містять технічну інформацію про послуги, що пропонуються компаніями, та адреси для пошуку інформації [10].

1.3.6 Бізнес-процеси

Поняття «бізнес-процес» є багатозначним, на сучасному етапі не існує єдино прийнятого його визначення. Більшість існуючих поглядів на бізнес схиляється до того, що це економічна категорія, тісно пов’язана з теорією реінжинірингу М. Хамера та Дж. Чампі.

У загальному значенні бізнес процес є систематизованим послідовним виконанням логічно пов’язаних та взаємозалежних завдань з використанням ресурсів, що забезпечують виробничу діяльність, з метою створення продукції, яка має споживчі цінності для клієнта.

Бізнес-процес – це комплекс робіт, що мають свої межі і відкриваються первинними постачальниками процесу, тобто входами процесу, якими можуть виступати матеріально-технічні, енергетичні, людські та інформаційні ресурси [11].

У парадигмі SOA бізнес-процес керує потоком сервісів. Бізнес-процес керує потоком подій, викликає і координує сервіси і створює контекст для їх взаємодії. Бізнес-процес являє собою бізнес-абстракцію; хоча він відокремленим від реалізації сервісів, бізнес-процес піклується про хід бізнес-діяльності. Такий поділ завдань не тільки дозволяє краще сконцентруватися на створенні процесу, але й полегшує зміну процесу відповідно до нових вимог без необхідної зміни застосовуваних реалізацій сервісів [3].

UDDI може використовуватися з метою перевірки даних про партнера, пошуку компаній у певній галузі промисловості з конкретним типом обслуговування. UDDI містить елементи таких типів:

- **бізнес об'єкт** (Business Entity) - безпосередньо визначає бізнес;
- **бізнес-сервіс** (Business Service) - містить інформацію про набір послуг;
- **шаблон зв'язування** (Binding Template) - містить інформацію про точки входу послуги;

- **модель технології** (TModel) - визначає окрему специфікацію для послуги.

Бізнес об'єкт (Business Entity) описує ПрО, до якої належить конкретний Веб-сервіс . Цей елемент може включати опис категорій для індустрії, що полегшує детальний пошук послуг.

Бізнес-сервіс (Business Service) - це клас послуг у межах певної галузі промисловості. Кожна галузь належить певному елементу Business Entity.

Шаблон зв'язування та модель технології визначають Веб-сервіс . TModel містить абстрактний опис, а Binding Template - конкретну специфікацію послуги [10].

1.3.7 Орієнтація на стандарти

Як правило, проекти SOA цілком покладаються на стандарти і використовують їх через такі основні переваги:

- Стандарти гарантують можливість взаємодії систем і партнерів.
- Використання стандартів прискорює розробку за допомогою процесів і інструментальних засобів.
- Стандарти покращують керованість і видимість інформаційних активів.
- Стандарти гарантують якість сервісу (quality of service - QoS).
- Стандарти покращують гнучкість завдяки зниженню залежностей від конкретних реалізацій.

Прикладів стандартів, які застосовуються у SOA:

Протокол **WS-Security** заснований на додаванні до заголовка повідомлення SOAP-розширень для зберігання метаданих системи захисту, призначених для реалізації механізмів підтримки цілісності повідомлення, конфіденційності та аутентифікації. Ці розширення дають універсальний механізм асоціювання маркерів системи захисту повідомлення замість

фіксованого захисного механізму. Базова платформа підтримує різні механізми захисту.

Мова **BPEL4WS** (Business Process Execution Language for веб Services) визначається на сайті спільноти OASIS по BPEL наступним чином: «Цей протокол визначає модель і граматику для опису поведінки бізнес-процесу на основі взаємодій між процесом і його партнерами. Він також визначає, як координуються взаємодії між сервісом і його партнерами для досягнення поставленої бізнес-мети, а також задає стан і логіку для цієї координації ».

Протокол BPEL4WS вводить необхідні методи для роботи з винятковими ситуаціями і збоями, а також способи корегування інших фіксованих процесів, які можуть вимагати відкату в разі виникнення помилок. Оскільки BPEL повинен підтримуватися скрізь, він ґрунтується на загальновизнаному протоколі WSDL, який, у свою чергу, заснований на технології XML.

1.3 Архітектура орієнтована на послуги

Поняття архітектури, орієнтованої на послуги, сформувалося упродовж розвитку концепції веб-сервісів. Архітектура веб-сервісів є однією з реалізацій SOA (є також інші підходи до реалізації SOA: Java RMI (від Sun Microsystems), CORBA (від консорціуму OMG), DCOM (від Microsoft), DCE (запропонований асоціацією Open Group) тощо.

Сервіс-орієнтовані обчислення (Service-oriented calculation SOC) - обчислювальна парадигма, яка використовує сервіси як фундаментальні елементи для розробки застосувань. SOC базуються на SOA і забезпечують виконання операцій управління сервісами. Розробка системи SOC - це процес пошуку, підбору і компонування сервісів, що задовольняють вимоги користувача.

Можливість компонування (composability) веб-сервісів часто розглядають як одну з основних переваг їх використання. Компонування полягає у

знаходженні набору елементарних сервісів, необхідних для реалізації функцій, використовуваних у запиті користувача, і визначення порядку їх виконання.

Функціональні можливості веб-сервісів визначаються входом, виходом, попередніми умовами і діями сервісу. їх позначають як ЮРЕ (inputs, outputs, preconditions, and effects). Наприклад, для сервісу купівлі попередня умова - це коректне введення номера кредитної картки, вихід - генерація квитанції, а дія - оплата товарів/послуг; електронний магазин може мати такі входи: назва товару, адреса споживача і номер його кредитної картки з попередньою умовою перевірки справжності цієї кредитної картки.

Виходами можуть бути електронна квитанція та операції з кредитною карткою і відвантаження товару споживачеві. Функціональні атрибути можуть описати показники якості сервісу, такі, наприклад, як час виконання купівлі і час оплати.

Більшість послуг, необхідних користувачам, формується вручну з використанням заснованих на WSDL описів елементарних сервісів. Для автоматичного компонування програми мають бути здатними відбирати потрібні веб-сервіси і компонувати їх.

Інформація, що міститься в реєстрі UDDI, недостатня для автоматичного компонування веб-сервісів, тому що не дає змоги інтерпретувати їх семантику. Тому розробляються механізми відображення семантики сервісів та її автоматизованого зіставлення з семантикою запитів користувачів. Можна розв'язати проблеми автоматичного компонування, зв'язавши параметри веб-сервісів з термінами визначеної ПрО і семантичним обґрунтуванням цих понять.

Інтелектуальні веб-сервіс и (семантичні веб-сервіс и, SW-сервіси) розширюють поняття традиційних веб-послуг. Хоча програми можуть знайти певний веб-сервіс в реєстрі UDDI без допомоги людини, вони не спроможні зрозуміти, як саме ним користуватися.

Мова опису веб-сервісів WSDL надає інструмент для опису того, яким чином взаємодіяти з тим чи іншим веб-сервісом, тоді як семантична розмітка надає інформацію про те, що і як здійснює цей сервіс.

Необхідно забезпечувати веб-сервіси такими описами, щоб можна було автоматично розпізнавати їх значення. Одним із поширених засобів подання семантики веб-сервісів є онтології у межах єдиної системи взаємопов'язаних компонентів.

Онтології полегшують автоматичне компонування послуг. Наявність подання знань про ПрО, до якої належить сервіс, допускає перебудову запитів контекстно-залежним способом і переговори про можливості цього сервісу.

Алгоритми знаходження відповідності між запитом і сервісом, які використовують онтологічне представлення знань, дають можливість автоматизувати знаходження схожих запитів і послуг. Для цього запит узгоджується на основі ієрархії понять ПрО, відображеної в онтології. Відповідність між описом веб-сервісу і запитом виявляється, коли всі виходи запиту узгоджені з виходами опису, і всі входи опису - з усіма входами запиту, а сервіс здатний задовольнити всі входи узгоджених сервісних потреб.

Найбільшою проблемою при виявленні сервісу є їх розподілений характер. Фіксація семантики запитів і досліджень сервісів, так само як контексту запропонованої взаємодії з сервісом, вимагає адекватних засобів подання сервісів і взаємодій. У зв'язку з цим можуть бути застосовані онтології. Для інтеро-перабельного подання онтологій розроблено мову OWL і її модифікацію для сервісів OWL-S (веб Ontology Language for Services).

Інтелектуальний пошук та автоматичне компонування Веб-сервіс ів можуть бути здійснені за допомогою можливостей семантичного опису Веб-сервіс ів, запропонованих у OWLS.

OWL-S забезпечує онтологічний опис веб-сервісів. Мета розробки OWL-S полягає в тому, щоб зробити можливим використання логічного виведення для веб-сервісів, планування автоматичного компонування веб-сервісів, автоматичного використання сервісів програмними агентами.

OWL-S забезпечує декларативні описи властивостей веб-послуги і можливості, які можуть використовуватися для автоматичного виявлення сервісу.

Використовуючи OWL-S, веб-сервіс може повідомляти потенційним користувачам про свої функціональні можливості. Запит на обслуговування може бути узгоджений з оголошенням веб-сервісів за допомогою процесу підбору (matchmaking).

OWL-S забезпечує механізм для моделювання бізнес-процесів, але відрізняється від нього виразністю термінів, уявлень, семантики, підтримки пошуку і виконання, обробки помилок. Опис OWL-S для сервісу складається з профілю сервісу, моделі сервісу та обґрунтування сервісу, тобто пояснення того, що виконує цей сервіс, як він працює, як можна дістати до нього доступ.

Профіль сервісу - абстрактна характеристика функцій сервісу. Профіль побудовано на основі контенту UDDI, що описує властивості сервісу, необхідні для його автоматичного виявлення, наприклад, пропозиція сервісу, його входи і виходи, попередні умови і додаткові дії. На основі профілю, який надає інформацію про провайдера, функціональні можливості, функціональні атрибути сервісу, можуть бути створені описи і запити сервісу.

Для семантичного обґрунтування параметрів веб-сервісів використовують онтології різного рівня. Використовують різну архітектуру для опису семантики джерел інформації: підходи, засновані на єдиній онтології, використовують одну глобальну онтологію, що забезпечує спільний словник для специфікації семантики; підходи, засновані на множинних онтологія х, кожне джерело інформації описується власною онтологією; гібридні підходи подібні підходам, заснованим на множинних онтологіях у тому, що семантика кожного початкового тексту описана її власною онтологією, але для того, щоб зробити локальні онтології порівнянними, формується глобальний словник для загального використовування.

Загальний словник (тезаурус) містить базові терміни (примітиви) ПрО, які комбінуються в локальні онтології для того, щоб описати складнішу семантику. Іноді загальний словник також є онтологією. ПрО може мати декілька онтологій. Будь-яка ПрО характеризується своєю дійсністю, тобто множиною ситуацій, які мали місце у минулому, мають місце у теперішньому і матимуть місце в

майбутньому. Інтеграцію онтологій можна розглядати як процес знаходження схожості між різними онтологіями. Нова онтологія може бути використана як посередник між різними системами. Залежно від змін, які необхідно зробити, щоб одержати нову онтологію, можна розрізняти такі рівні інтеграції: відповідність (alignment), часткова сумісність (partial compatibility), удосконалення і уніфікація (unification).

Основою архітектури, орієнтованої на послуги, є взаємодія її учасників: постачальника, споживача та реєстру послуг (рис. 1.1.).

Рисунок 1.1 – Схема взаємодії учасників SOA [10]

Концепція веб-сервісів означає, що вони мають певну обмежену функціональність. Для вирішення складних завдань потрібно використовувати функціональність кількох послуг. Тому в процесі розвитку архітектури веб-сервісів виникло поняття компонування веб-сервісів і потік веб-послуг, або ще використовують термін оркестровка (веб Service Choreography) і хореографія (веб Service Choreography) веб-сервісів. Ці поняття відображають взаємодію послуг і послідовність їх виконання. Додатки, побудовані з використанням веб-сервісів, базуються на потоках робіт (Workflow-based applications).

Для опису бізнес-систем, що базуються на архітектурі веб-сервісів, IT-компанії запропонували використання різних стандартів: Wf-XML (від Workflow

Management Coalition), WSFL (IBM веб Services Flow Language), XLANG (Microsoft XLANG: Business modeling language for BizTalk), PIPs (Roset-taNet's Partner Interface Process) тощо.

Розподілені обчислення через Internet викликають фундаментальні зміни у веденні бізнесу, і саме веб-сервіс и забезпечують відкритий механізм інтеграції бізнес-процесів. Управління бізнес-процесами відбувається в автоматизованому режимі. Так, за допомогою методів моделювання можна перевіряти коректність виконання бізнес-логіки, представленої в діаграмах, а потім автоматично одержувати опис цих діаграм на XML-мовах управління бізнес-процесами.

Цей підхід допомагає спростити виклик веб-сервісів з будь-якої точки на основі бізнес-правил. Завдяки цьому компанії можуть реалізовувати швидку зміну бізнес-правил.

Стратегічна цінність веб-сервісів полягає у скороченні часу реалізації проектів, підвищенні продуктивності, швидкій інтеграції бізнес-систем та їх застосування.

Тактичні переваги веб-сервісів: проста розробка і впровадження , використання інвестицій, зменшення ризику, пов'язаного з впровадженням проектів у сфері автоматизації послуг та бізнес-процесів, можливість безперервного поліпшення надання послуг, скорочення кількості звертань за технічною підтримкою, підвищення показника повернення інвестицій (ROI) тощо.

Так, низка підприємств з різних галузей економіки, включаючи фінансові послуги, страхування, аерокосмічну галузь, охорону здоров'я, фармацевтику, роздрібну торгівлю, державний сектор і промисловість, впроваджують власні веб-сервіси [10].

1.4 Особливості проектування SOA

SOA не прив'язане до жорсткої методології проектування, впровадження або управління IT-інфраструктурою. SOA обмежується лише рядом принципів,

що характеризують кожен з цих процесів; тому її іноді називають не архітектурою, а архітектурним стилем. Розглянемо деякі з цих принципів [12]:

- Розподілене проектування. Рішення відносно внутрішніх особливостей інформаційних систем приймаються різними групами людей, що мають власні організаційні, політичні і економічні мотиви. – Постійність змін. Окремі ділянки архітектури можуть зазнавати зміни у будь-який момент часу.
- Послідовне вдосконалення. Локальне поліпшення компонентів архітектури повинне призводити до вдосконалення усієї архітектури в цілому – до зростання сумарної корисності компонентів того ж рівня, що і змінюваний, так само як і компонентів нижчого і більш високого рівня. Наприклад, відомий веб-сервіс Google Translate постійно зазнає зміни. Спочатку, він забезпечував тільки веб-інтерфейс для перекладу і обмежений набір мов. Поступово збільшувалися функціональні можливості сервісу: розширювався набір мов, з'явилася можливість голосового відтворення перекладу, при перекладі окремого слова почали видаватися словникові статті з декількома результатами перекладу і т. п. При цьому API і інтерфейс мінявся настільки мало, що клієнти могли використовувати нові можливості за допомогою старого API (наприклад, отримання словникових статей) або ж не міняти використовуваний інтерфейс до тих пір, поки не буде потрібно використання нових можливостей.
- Рекурсивність. Однотипні рішення мають місце на різних рівнях архітектури. З точки зору інформаційних технологій, логіка підприємства [13,14] може бути розділена на декілька шарів, як показано на (рис. 1.2.)

Рисунок 1.2 – Приклад розділення компонентів підприємства на шари [13]

Кожен шар існує окремо від іншого. Структура рішень SOA, показана на рисунку, є еталонною моделлю SOA, що відбиває концептуальний (високорівневий) пристрій рішень SOA. Ця модель, яку іноді також називають "Багатошаровою архітектурою SOA", включає такі шари і поняття, як бізнес-процес, сервіс, сервісний компонент, а також взаємозв'язки між ними. Вона не залежить від технологій, на яких побудована.

Бізнес-логіка є документальною реалізацією бізнес-вимог, які виходять з проблемної області, в якій працює підприємство. Бізнес-логіка, як правило, структурована в процесах, які виражають ці вимоги, а також обмеженнях і залежності від зовнішніх впливів. Логіка додатка – це реалізація бізнес-логіки, організована на основі різних технологічних рішень. Логіка додатка виражає процеси бізнес-логіки за допомогою придбаних або спеціально розроблених програмних систем в умовах обмежених технічних можливостей і залежностей від постачальника рішення. Процес перетворення бізнес-логіки в логіку додатків і реалізація сервісів на основі цих вимог є процесом створення сервісно-орієнтованої інфраструктури для завдань підприємства. Не існує жорстких правил і концепцій принципів побудови SOA, але при реалізації власної

інфраструктури бажано дотримуватися деяких основних підходів, описаних нижче [12, 15].

1.5 Висновки

Архітектура SOA абсолютно незалежна від мов програмування, платформ або протокольних специфікацій, за допомогою яких сервіси розробляються, а також від того, де і за допомогою чого вони розгорнуті. Практично архітектура SOA вимагає наявності не лише сервісів, але і засобів, за допомогою яких ці сервіси можуть бути виявлені і підключені незалежно від інфраструктури. SOA – це не продукт або специфікація, тому потрібне ретельне вибудовування цієї архітектури, що складається з безлічі компонентів, таких, як сервери додатків, єднальне ПО, репозиторій і навіть спеціалізовані пакети централізованого управління SOA.

Строго кажучи, SOA не можна відносити ні до реалізації CORBA, ні до архітектури RMI (Remote Method Invocation). Ключовий компонент SOA-сервіс. Сервіси тут є бізнес- функціями, призначеними для забезпечення узгодженої роботи великих застосувань, що складаються з безлічі частин. А кінцевим місцем, розташування сервісів, являється сервер додатків.

На відміну від звичайних застосувань сервіс в архітектурі SOA призначається для використання усім реалізованим бізнес-функціям. Тоді як звичайні корпоративні застосування містять в собі схожі фрагменти бізнес-логіки або навіть дублюють окремі об'єкти – наприклад, об'єкт клієнтського замовлення, – в архітектурі SOA треба запустити лише єдиний екземпляр такої бізнес-функції. Таким чином, можна повторно використовувати функціональність в середовищі з множинними застосуваннями і швидко.

2. АНАЛІЗ КОНЦЕПЦІЇ ПОБУДОВИ КОМПОЗИЦІЙНИХ ВЕБ-ДОДАТКІВ

2.1 Ознайомлення з концепцією композитних веб-додатків

2.1.1 Поняття композитного веб-додатку

Композитний веб-додаток (mashup) – це веб-додаток, який використовує дані з більше ніж одного джерела для створення нового сервісу, що відображується одним графічним інтерфейсом. Наприклад, комбінуючи картографічні дані Google Maps з відгуками людей про подорожі до якихось місць, можна створити унікальний веб-сервіс з функціональністю, що не передбачало жодне джерело.

Створення таких сервісів включає в себе рішення одразу декількох проблем, таких як обробка даних одразу з декількох ресурсів, аналіз і вибір необхідного, комбінація цього

Принцип створення mashup був запозичений з напряму поп музики, де вперше з'явився напрям музичного жанру mashup, яке полягало в змішуванні вокальних та інструментальних звукових доріжок зібраних з різних музичних композицій.

Mashup створюється на основі веб-служб, які представляють розробникам інтерфейси прикладного програмування API. Творці API дотримуються простоти і оптимальності. Зазвичай термін mashup застосовується тільки до тих проектів, які використовують відкриті інтерфейси API для отримання вище наведених послуг [17]. Іншими методами отримання інформації можуть бути веб-фіди (наприклад RSS або Atom) або парсинг веб-сторінок.

Побудова сервісів за такою концепцією базується на сервіс-орієнтованій архітектурі, більшість розробників веб-додатків останніми роками публікували їх прикладні програмні інтерфейси, дозволяючи розробникам інтегрувати дані і функції у свої проекти, замість того щоб писати їх самостійно.

2.1.2 Історія появи

Історія появи композитних веб-додатків відслідковується від самого початку появи веб. У Web 1.0 компанії зберігали дані користувачів на порталах, постійно їх обновлюючи. Вони контролювали усю інформацію, а користувачам доводилося використовувати їх продукти і сервіси для отримання інформації. Після краху dot-com систем в 2001 році з'являється нове покоління Інтернету і програмного забезпечення, які приносять радикально нові моделі роботи взаємодії з користувачами [18].

Mashup з'явився в інтернеті завдяки появі Web 2.0. Web 2.0 - методика проектування систем, які шляхом обліку мережевих взаємодій стають тим краще, чим більше користувачів ними користуються. По суті, термін позначає проекти і сервіси, які активно розвиваються і покращувані користувачами: блоги, соціальні середовища тощо. Наприклад: веб сервіси: youtube, facebook - якби користувачі не додавали туди свої дані, то ці веб сервіси були б порожні і нудні [18].

У своїй книги, один з винахідників Інтернету Tim Berners-Lee пише: "Ні хто не знає, що це означає ... Якщо веб 2.0. - Це ваші блоги і вікі, тоді це означає "користувачі для користувачів". Але це те ж саме, що сказати - Web 2.0. існує, щоб всі люди були разом " [19]. Тим самим він підкреслює, що Web 2.0 буде розвиватися і ставати цікавіше завдяки користувачам які самі будуть додавати свої дані на веб сервіси.

Вважається, що сайт Chicagocrime був одним з перших mashup сайтів, створеним в 2005 році. Призначення сайту полягало у відображенні на карті Google даних Чиказької поліції про злочини. Принцип роботи сайту полягав у поєднані таких ресурсів як: карта Google і база даних Чиказького поліцейського департаменту. На карту Google накладалися такі дані як календарна дата, тип злочину, координати і адреса. Тим самим користувач міг, вводячи ці параметри, бачити на карті позначені маркерами точки злочинів і визначити ступінь кримінальності району. Звичайно, це був примітивний і вкрай незручний mashup, оскільки, щоб подивитися злочини, треба було ввести дату і тип злочину, а якщо

користувач не знав дати або типу злочину, то він шукав навмання. Тим не менш, цей сайт за своїм принципом роботи вважається першим mashup додатком [20].

2.1.3 Типи композитних веб-додатків

Mashup можна розділити на три основні типи:

Mashup додатки користувача - об'єднують дані з різних відкритих джерел в браузері користувача. Користувач сам може брати участь у створення нових даних, приклад тому сайт earthalbum.com де об'єднуються картографічні дані з сервісу Google Map і фотографії з сервісу Flickr [21].

Mashup даних («enterprise» - mashup) змішують дані близькі за типом з різних джерел, наприклад, об'єднуючи дані з декількох RSS-фідів в один фід з графічним інтерфейсом. «Enterprise» - mashup зазвичай інтегрує дані із зовнішніх і внутрішніх джерел. Такий mashup може, наприклад, створювати звіт про зайнятої частини ринку, об'єднуючи зовнішній список всіх проданих за минулий тиждень будинків з внутрішніми даними про те, які будинки були продані окремим агентством [22].

Бізнес mashup додатки - створюються з використанням технології бізнес-бізнес (b2b). Добре підходять для швидкої розробки проектів, які вимагають співпраці розробників і замовників для визначення та реалізації бізнес-вимог. Збір та аналіз порівняльної інформації щодо ціни на торгових Інтернет-майданчиках дозволяє орієнтуватися в ціновому діапазоні, перебуваючи в одному веб додатку. eBay та Amazon – одні з перших компаній, які створили API для програмного доступу до свого контенту.

Інколи виділяють ще телекомунікаційний та навчальний mashup.

Телекомунікаційний mashup - це телекомунікаційний сервіс, елементи якого зібрані з декількох джерел. Наприклад, хтось може отримувати базовий сервіс від компанії А, тон зворотного дзвінка від компанії Б, сервіс голосової пошти від компанії В тощо [23]

Навчальний mashup -(«Training» -mashup) це навчальний сервіс в Web, який інтегрує дані з різних навчальних джерел в інтернеті. Mashup всередині mashup називають «mashup-монстрами».

Також mashup розрізняють за типом API, якій вони використовують, але вони можуть бути комбіновані між собою або інтегровані в інші сервіси.

За типами використаних даних:

- Індексовані дані
- Картографічні або географічні дані
- Фіди, подкасти

За типами функцій:

- Конвертери інформації
- Комунікації
- Візуалізація інформації
- Безпека
- Редактори

2.1.4 Mashup і портали

Mashup і портали – це дві різні технології агрегації контенту. Портали – це більш старіша технологія проектування, яка є розширенням традиційних динамічних веб-додатків. В ній процес конвертування даних у веб-сторінку розділений на дві фази: генерація індексованих фрагментів і агрегація їх у веб-сторінки. Кожен індексований фрагмент генерується портлетом, які можуть бути розміщені локально або на віддаленому сервері. Портлет – це змінний компонент користувацького інтерфейсу веб-порталу, що видає фрагменти розмітки, які вибудовуються у веб-сторінку.

Технологія порталу визначає повну модель подій, що включає в себе зчитвуання та оновлення інформації. Запит агрегації сторінки для порталу транслюється в індивідуальні запити для всіх портлетів, що формують сторінку. Якщо кнопка підтвердження були натиснута на будь-якому портлеті сторінки

порталу, це транслюється у операцію оновлення лише цього портлету, його оновлення автоматично зчитується усіма іншими портлетами на сторінці.

Отже, між ними полягає у багатьох речах. З боку самого принципу формування сторінок портали агрегують інформацію розділяючи роботу серверу у дві фази: генерація розмітки і агрегація фрагментів розмітки, а mashupи використовують функціонал прикладного програмного інтерфейсу різних сайтів для використання їх контенту у інших цілях. Портали агрегують вже сформованими фрагментами розмітки, а mashup можуть обробляти XML контент. Агрегацією інформації займається лише сервер у портальній технології, а у mashup вона може бути як на стороні сервера, так і на стороні клієнта. Стиль агрегації порталів – це накопичений вміст один за одним з декількох ресурсів без накладок, а mashup дозволяють об'єднувати дані у будь-який спосіб [24].

2.2 Ознайомлення з архітектурою, принципами, спорідненими технологіями композитних веб-додатків.

2.2.1 Архітектурні аспекти

Архітектура будь-якого mashup складається з трьох основних частин, що не перетинаються між собою фізично та логічно:

- Прикладний програмний інтерфейс. Провайдери даних. Це провайдери контенту, звідки береться інформація. Для полегшення обміну даними, провайдери зазвичай представляють контент через веб-протокли, такі як REST, веб Services, and RSS/Atom. Тим не менш, багато потенціально цікавих джерел даних ще не представили власний API.
- Mashup сайт. Це сервер на якому розміщується mashup та інформація від провайдерів контенту, інформація надходить за допомогою відкритих API. З одного боку, mashupи можуть працювати як традиційні Веб-сервіс и, використовуючи динамічну генерацію контенту на стороні серверу технологіями Java, CGI, PHP або ASP.

З іншого, mashup контент може генеруватися на стороні браузера клієнта через клієнтські сценарії або апплети. Логіка генерації контенту на стороні клієнта – це комбінація коду вбудованого прямо в mashup веб-сторінку, сценаріїв API бібліотек або апплетів. Переваги такого підходу полягають у меншому навантаженні на сервер (дані можуть напряму передаватись від провайдеру контенту) і більш органічному використанні ресурсів (сторінки можуть оновлювати дані частково без повного оновлення всієї сторінки)

Зазвичай mashupи використовують комбінацію серверної та клієнтської логіки для досягнення необхідної агрегації даних. Багато mashupів використовують дані, які поставляються напряму від їх бази користувачів, роблячи хоча б один з наборів даних локальним. Додатково, виконуючи комплекс запитів на різні ресурси, інформація потребує обчислень, що можуть бути виконані в браузері клієнта.

Клієнтський браузер, який за певних налаштуваннях браузера може генерувати інформацію як за мовними та регіональними налаштуваннями, так і за останніми пошуковими запитами. Завдяки цьому, при запуску браузера, він сам обирає потрібні налаштування Веб-сервіс ів, позбавляючи користувача від зайвої інформації, незв'язаної з його мовними, регіональними та пошуковими запитами. Приклад реалізації налаштувань браузера добре спостерігаються в API Google Maps. При запуску браузера у користувача відображається інформація на його мові. Одним словом клієнтський браузер - це середовище, в якому додаток інтерпретується в графічному вигляді й взаємодіє з користувачем [25].

2.2.2 Огляд технологій і протоколів, що полегшують створення mashup додатків

Для того, щоб mashup додатки почали свою роботу, використовуються різні технології для їх реалізації:

Ajax - це скоріше модель веб-додатків, аніж специфічна технологія, що складаються з технологій:

- XHTML та CSS – результат співпраці між W3C та веб- додатків генеративної технології робочої групи WHATWG. Заснований на HTML, CSS, DOM, JavaScript;
- API моделі DOM, незалежна інтерфейсна модель - для динамічного відображення і взаємодії HTML;
- XML, розширювана мова розмітки - для асинхронного обміну даними;
- JavaScript - використання сценаріїв на стороні клієнтського браузера.

Використовуючи усі технології разом, досягається ціль створення гладкого, об’єднуючого веб-досвіду для користувача, замість повного перезавантаження та ри-рендеренгу сторінки йде обмін малими частками даних з серверами контенту. Враховуючи, що дані обмінюються між різними серверами і інтерпретуються на користувальницької машині, Ajax дозволяє це робити без перезавантаження браузера під час інтерактивної роботи користувача з веб- додатком.

Веб-протоколи **SOAP і REST** - це платформи незалежних веб-протоколів для зв'язку з віддаленими сервісами. Виступаючи в ролі частини сервіс-орієнтованої парадигми, клієнти можуть використовувати SOAP і REST для взаємодії з віддаленими сервісами без знання їх внутрішньої реалізації, функціональність сервісу цілком виражається описанням запитів та відповідей

SOAP - протокол доступу до об'єктів, фундаментальна технологія парадигми Веб-сервісів. SOAP – це акронім Services-Oriented Access Protocol. Використовується для обміну довільним повідомленнями у форматі XML незалежно від платформи. Структура його повідомлень складається з заголовку та тіла запиту. У заголовку запиту міститься контекстна інформація, що не відноситься до корисної інформації для додатка, наприклад інформація для аутентифікації. Тіло запиту містить специфіковану інформацію для сервісу. SOAP API для Веб-сервісів , що описується WSDL документами, які описують які операції розкриває сервіс, формат повідомлень які приймає, і як їх адресувати.

REST - архітектурний стиль програмного забезпечення для розподілу систем, таких як World Wide Web, який використовується для побудови веб-служб допомогою HTTP і XML. REST – це акронім Representational State Transfer. На відміну від типових інтерфейсів, що базуються на різних наборах запитів для різних даних, які ви можете знайти у сучасних мовах програмування, REST підтримує лише декілька операцій, а саме POST, GET, PUT, DELETE, що приємні для усіх типів даних. Виразність у REST є однією з частин інформації самої по собі, званим ресурсом. Наприклад, ресурс співбесіди для майбутнього співробітника ідентифікується URI, повертається через GET, оновлюється через PUT і так далі. REST подібний до документально-буквального стилю сервісів SOAP.

Клієнти можуть використовувати SOAP і REST для взаємодії з віддаленими сервісами. Функції сервісу повністю передаються через опис повідомлень, за допомогою яких здійснюються запити і відповіді [25].

Як було зазначено вище, недостатність API від провайдерів контенту часто змушувало mashup-розробників парсити веб-сторінки для отримання необхідної інформації для композиції. Процесс полягає в використанні спеціального ПЗ для парсингу та аналізу даних, що початково написані для сприйняття людиною, перетворення семантичних структур даних у форму, що якою може маніпулювати і використовувати програмно. Невелика кількість mashup-ів використовують такий підхід для отримання необхідних даних, особливо з публічних ресурсів.

Парсинг сторінок вважається неелегантним рішенням, і не без підстав. Він має декілька невід’ємних недоліків. Перший, на відміну від даних отриманих з API, парсинг не має структурованої форми обміну інформацією між провайдером контенту і клієнтом. Клієнт має підлагоджувати власні інструменти під джерело контенту і сподіватися, що провайдер дотримувався саме цієї моделі відображення. веб-сайти мають тенденцію до оновлення їх вигляду періодично, щоб залишатися свіжими і стильними, що доставляє велику головну біль через падіння системи парсингу.

Другий недолік – це недостача складного, повторно використовуваних інструментів для парсингу та аналізу даних. Такі інструменти підлаштовуються індивідуально під кожен сервіс. Це призводить до того, що розробники змушені деконструювати контент, розробляти моделі даних, парсити і агрегувати необроблені дані з сайту провайдера контенту [26].

Неелегантність аспектів парсингу відслідковується від факту, що контент створений для сприйняття людиною, а це в свою чергу робить його не пристосованим до автоматизованим сприйняттям комп'ютером. Рішенням є Семантичний веб, який передбачає, що контент сучасного веб може бути доповнений до стану, в якому дані є прийнятним для сприйняття людиною з еквівалентною інформацією, яку здатен сприймати комп'ютер. В цьому контексті Семантичного веб поняття інформації є відмінним від поняття даних, дані становляться інформацією у момент коли вони виражають суть, зміст. Семантичний веб має мету створення веб-інфраструктури, яка поповнює дані метаданими, щоб дати їх зміст, роблячи таким чином їх прийнятними для інтеграції і використання повторно.

Консорціум W3 створив набір специфікацій відомий як RDF. RDF – це акронім Resource Description Framework. Він служить для досягнення цілі забезпечення методології для встановлення синтаксис-структур, які описують дані. XML самого по собі недостатньо, він занадто довільний, ви можете закодувати одні і ті самі дані різними шляхами. RDF-схема додає до RDF можливість закодувати концепти у читабельний для комп'ютера спосіб. Як тільки об'єкти даних можуть бути описані в моделі даних, RDF забезпечує створення відношень між об'єктами даних через відношення суб'єкт-предикат-об'єкт. Комбінація моделей даних і графа відношень дозволяє створення онтологій, що є ієрархічними структурами інформації, які можуть бути знайдені і номінально обґрунтовані. Наприклад, ви можете визначити модель в якій «хижаки» підклас «тварин» з властивістю «з'їсти» іншу «тварину» і створити два екземпляри: перший наповнений даними описуючими гепардів і полярних ведмедів та їх звички, інший описує газелей та пінгвінів і відповідні їх звички.

Система висновків може «замашурити» ці окремі джерела і зрозуміти, що гепарди полюють на газелей але не на пінгвінів.

RDF дані - це швидкий пошук необхідних даних різноманітті доменів, включаючи соціальні мережі і синдикати, такі як RSS. До того ж, ПЗ для технології RDF та його компоненти дозволяють досягти високого рівня завершеності, особливо при використанні мов запитів RDF, програмних фреймворків та невід'ємних двигунів.

Atom і RSS - це сімейство форматів синдикації на основі XML. RSS дозволяє за допомогою онлайн сервісів і різних агрегаторів і каталогів імпортувати і експортувати інформацію, що дозволяє користувачу збирати потрібну для нього інформацію в зручному для нього відео з різних сайтів.

Формат Atom, який врахував недоліки RSS і реалізує обмін інформацією заснований на XML [21].

Ці технології синдикації добре підходять для mashup які агрегують контент, що постійно оновлюється, такий як новини або веб-блоги.

2.2.3 Технічні проблеми

Проблеми інтеграції даних. Семантичне значення і якість даних.

Опитування якості показують, що основний пріоритет IT підприємств є інтеграція даних в межах підприємства віртуальної організації. (У цьому контексті, я використовую термін віртуальна організація, означає складу об'єднання бізнес-одиниць, кожна міститься у його власному адміністративному домені.). Як і багато керівників підприємств IT, які знаходять себе в завдання інтеграції джерел даних (наприклад, створити корпоративні інформаційні панелі, які відображають поточні умови бізнесу), розробники mashupів стикаються з аналогічними проблемами отримання загального семантичного змісту між гетерогенними наборами даних.

Наприклад, системи переводу між моделями даних повинні бути розроблені. Коли конвертується інформація в звичайні форми, мають бути створені припущення щодо певних параметрів, якщо відображення не є повним.

Крім того це може бути ускладнено фактом, що розробник mashupy не є експертом у моделях джерела даних, а тому припущення можуть не бути інтуітивними і зрозумілими.

В добавок до відсутніх даних або не повних відображень , розробник композитного веб-додатку може зустрітися з тим, що інформація, яку він хоче інтегрувати не є прийнятною для автоматизованого сприйняття компютером, вона має бути підготовлена. Семантичні технології моделювання, такі як RDF, мають допомагти полегшити проблеми автоматичного сприйняття між різними наборами даних. Спадкові набори даних потребують більших людьских зусиль в аналізі і підготовці даних перед тим як до них можуть бути застосовані семантичні технології моделювання.

Частиною розробки будь-якого додатку є забезпечення вводу інформації користувачем. Це ніж з двома гранями, з однієї сторони це дуже потужна можливість, що забезпечує відкритий внесок в роботу, з іншої інформація може бути не коректною, не повною і т.д. Це визиває велику кількість сумнівів у введених даних, а mashup має забезпечити коректну їх інтерпретацію [25].

Проблеми компонентів

Модель Ajax розробки може забезпечити значно кращий і багатший досвід використання користувачем, а ніж повне перезавантаження сторінки, але вона також вносить додаткові складнощі. Фундаментально, Ajax використовує сценарні можливості браузеру клієнта і поєднанні з його DOM для досягнення методів отримання контенту, який не цілком передбачається дизайнерами браузеру. Тим не менш, субєкти сервісів побудованих за Ajax вирішують проблеми сумісності інформації, ще з часів Internet Explorer .

Використання JavaScript для асинхронного оновлення контенту сторінки можуть спричинути помилки у інтерфейсі користувача. Через те, що контент не звязаний безпосередньо з URL у панелі адреси браузеру, користувачі можуть не отримати функціонал зазвичай очікуванний при використанні кнопки повернення в браузері, або створення закладки. Також Ajax підвищує затримку відправляючи запити для оновлення контенту, поганий дизайн може справити

погане враження на користувача, тому дизайнер має піклуватися про створення панелей прогресу під час таких операцій.

Розробники mashup та провайдери контенту мають забезпечити передачу захищених даних, як і для будь-якого розподіленого, кросс-доменого сервісу. Поняття ідентифікації може виявитися проблемним, бо традиційний веб в першу чергу створений для анонімного доступу. Одноразовий вхід без аутентифікації в систему є бажаним, але є безліч конкуруючих технологій (від Microsoft Passport до Liberty Alliance), таким чином, створюючи розрізнені простори імен, що необхідно інтегрувати також. Контент-провайдери, швидше за все, використовують схеми аутентифікації і авторизації (які вимагають поняття безпечної ідентифікації або надійно ідентифікованих атрибутів) у своїх API, для забезпечення бізнес-моделі, які включають платні підписки або конфіденційних даних. Конфіденційні дані, ймовірно, також вимагають шифрування, і розробник повинен дбати, коли ви компонуєте їх з іншими джерелами, щоб не скомпрометувати їх. Крім того, інтеграція даних з відбувається як на сервері і на стороні клієнта, тому ідентичність та облік даних при передачі даних від користувача до mashup-сервісу може стати вимогою [25].

2.3 Екосистема Mashup

2.3.1 Поняттня Mashup-екосистеми

На (рис. 2.1) зображено вертикальний стек mashup-екосистеми. Кожен елемент будується на основі попереднього, результат - mashup-додаток. Mashup-додаток є верхнім рівнем всієї екосистеми, але його можна отримати лише за наявності інших елементів екосистеми.

Рисунок 2.1 – Вертикальний стек mashup-екосистеми [26]

Mashup-додаток - це тип ситуативних додатків, що складається з двох або більше різних компонент, які поєднуються для створення нового інтегрованого додатка. Спочатку ситуативні додатки будувалися головним чином на мові командного процесора і на сценаріях Perl, які, при виникненні потреб у ситуативній обробці широко застосовувалися розробниками для швидкого і недорогого створення відповідних рішень. Нажаль, ці технології вимагають наявності досвіду розробки і не підходять для використання досвіду бізнес-користувачів. Однак завдяки еволюції SOA і появі фасадів сервісів виникає основа для mashup-екосистеми, що дозволяє користувачам візуально розробляти ситуативні програми у вигляді mashup-додатків.

Mashup-конструктор - це середовище компоновки для виконання і створення mashup-додатків. Він надає користувачам ефективний спосіб візуальної компонування mashup-додатків шляхом з'єднання загальнодоступної інформації і сервісів з внутрішньо-корпоративною приватною інформацією і сервісами. Користувач може потім візуально маніпулювати цими контентами та інтегрувати їх, незалежно від того, є вони статичним, такими як веб-сторінка, або динамічним, такими як SOAP, REST (Representational State Transfer)-сервіс або RSS-фід. В інструментарії IBM Mashup Starter Kit є mashup-конструктор QEDWiki.

Mashup-конструктор дозволяє швидко візуально компонувати mashup-додатки, надаючи набір віджетів, які є програмними компонентами, що надають доступ до одного або декількох сервісів і контенту. При проектуванні віджетів особлива увага приділяється споживанню та налаштування, щоб гарантувати їх надзвичайну гнучкість, оскільки один з основних принципів веб 2.0 полягає в тому, що ви не можете припускати заздалегідь, як ваш контент буде використовуватися. Віджети можуть бути як візуальними (надають видимий контент, наприклад, діаграму), так і невізуальними (надають певного роду дискретну функцію або доступ до сервісу). Через mashup-конструктор віджет можна перетягнути з палітри на робочий стіл, де можна переглядати властивості віджетів і з'єднувати входи і виходи різних віджетів, створюючи з них mashup-додатки.

Щоб забезпечити наявність достатньої кількості віджетів (тільки тоді компоновка mashup-додатків стане реальністю), повинні бути широко поширені доступні інтерфейси сервісів даних, що представляються віджетами. Інтерфейс сервісу даних - це загальне поняття, що використовується для опису технологічних засобів (зазвичай це фіди, REST, SOAP / веб Services Description Language (WSDL), Asynchronous JavaScript + XML (Ajax) або виклики XML Remote Procedure Calls (XML-RPC)), які провайдер сервісу надає для доступу до свого контенту (інформації, яку ви хочете використовувати або зробити доступною у вашому mashup-додатку). До складу інструментарію IBM Mashup Starter Kit входить Mashup Hub - сервери керування фідамі, що надає широкі можливості для забезпечення доступу mashup-додатків до контенту [26].

2.3.2 Створення ситуативних додатків в рамках mashup-екосистеми

Розглянемо створення ситуативних додатків за допомогою mashup-конструктора, використовуючи модель компонування, з'єднання і розподілу.

Компонування. Підприємство може створити каталог віджетів, доступний mashup-конструктору. Цей каталог містить усі створені на підприємстві і зовнішні доступні віджети, які можуть використовуватися в даній сфері

діяльності. Це дозволить користувачам швидко знаходити і застосовувати сервіси та контент, необхідні в процесі створення mashup-додатки.

З'єднання. Автор використовує сервіси, доступні в каталозі, і з'єднує їх разом через mashup-конструктор, швидко створюючи mashup-додатки у візуальному режимі. Наприклад, на сторінку можна помістити віджет-form, що дозволяє користувачам вводити дані. Ці дані можна з'єднати з входом віджета, що забезпечує виклик Веб-сервіс у, а вихід відповіді Веб-сервіс у можна з'єднати з віджетом, який візуалізує дані.

Публікація. Автор робить mashup-додаток загальнодоступним, щоб його могли використовувати кваліфіковані фахівці і можна було застосувати механізми співтовариства.

У цьому процесі чітко простежуються три ролі:

Mashup enabler (сприяючий). Сприяючий пише віджети і додає їх в каталог. Він спілкується з авторами, щоб зрозуміти їхні вимоги, і додає відповідні сервіси. Зазвичай це співробітник інформаційного відділу або хтось, що вміє писати програми.

Mashup assembler (автор). Це людина, зазвичай не програміст, а є бізнес-користувачем або експертом в предметній області. Автор створює mashup-додатки, з'єднуючи mashup-матеріали, створені сприяючим.

Knowledge workers (кваліфіковані фахівці). Це спільнота, що використовує додаток за його призначенням і застосовує механізми спільноти до додатка (наприклад, ранжування і коментування) для формування зворотного зв'язку з метою можливого поліпшення додатки на наступній ітерації [24].

Відповідальність ролей на різних фазах моделі зображені на (рис. 2.2)

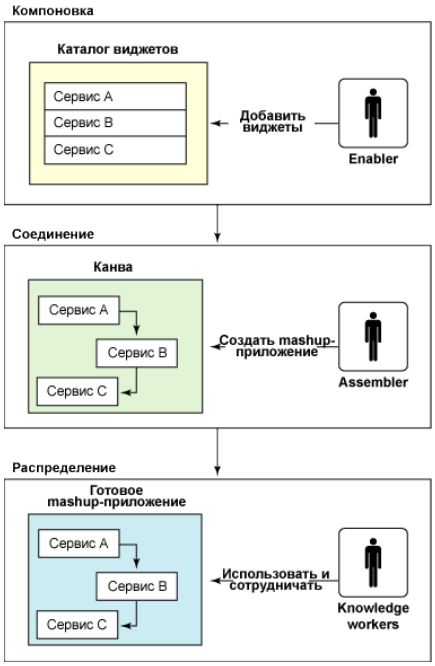

Рисунок 2.2 – Відповідність ролей в mashup-екосистемі [26]

Переваги екосистеми Mashup:

По-перше, значний прогрес, якого досягла індустрія на шляху стандартизації ключових технологій, таких як SOAP, REST і Ajax, привів до бурхливого зростання кількості впроваджень SOA, як в веб, так і на підприємствах. Це привело до досягнення критичної маси технологій і, як наслідок, швидкому поширенню повторно використовуваних веб API і mashup-додатків. У результаті набір контенту, даних і сервісів, доступних для звичайної людини, яка хоче розробляти програми на основі цієї моделі – розширюється постійно.

По-друге, це легкодоступна модель програмування. Вона має дуже низький бар'єр освоєння завдяки широкій базі кваліфікованих користувачів, добре знайомих з інтернет-додатками (і простими мовами сценаріїв, наприклад, PHP) і здатних виконувати завдання, наприклад, писати макроси Excel [21].

2.4 Аналіз процесу композиції веб-сервісів

2.4.1 Критерії аналізу процесу композиції веб-сервісів

Розглядаючи процес композиції сервісів, слід брати до уваги основні артефакти системного інжинірингу складних систем, такі як його фази, дисципліни, розподілені розробки, можливі різні понятійні апарати у провайдерів і, відповідно, семантичні контексти у сервісів тощо. Тому можна виділені такі властивості процесу композиції веб-сервісів:

- Специфікація вимог клієнта і властивостей веб-сервісів;
- Верифікація та валідація (V&V) композитних веб-сервісів;
- Планування процесу композиції.

Для специфікації веб-сервісів і їх композицій можна застосовувати графічні, текстові, формальні і ін. способи. У табл. 3.1 наведено їх короткий аналіз.

Таблиця 3.1 – Характеристики способів специфікації композиції

Способи специфікації	Переваги	Засоби специфікації
Графічна	Наочність, можливість уявлення у вигляді XML-схем	IDEF0, UML, SysML
Текстова	Стандартизація	WS-BPEL, WS-CDL
Формальна	Можливість автоматизованих процесів V&V	Темпоральні логіки (LTL, TLA)

Найбільший інтерес для автоматизованої композиції представляють машинно-орієнтовані способи специфікації. В даний час розвиваються способи і інструментарії автоматизованої генерації специфікацій, програмного коду з специфікацій і їх зворотного перетворення (Rational, StarUML, NetBeans, Altova, Protege, плагіни для Eclipse). При цьому найбільш перспективним вважається

використання діаграм мови UML і похідних від нього мов (наприклад, SysML, SoaML), діалекти мови XML (WSDL, gWSDL, tModel для UDDI, OWL-S, SOAP)

На думку автора статті [27] найбільш важливими є наступні критерії оцінки процесу автоматизованої композиції веб-сервісів:

1) спосіб специфікації (графічна, текстова, формальна);
2) повнота забезпечення V & V композитних сервісів;
3) облік нефункціональних (QoS) вимог;
4) практична застосовність;
5) можливість планування процесу композиції.

Спосіб специфікації - дає можливість визначити, чи передбачає підхід формальну специфікацію композиції і, як наслідок, автоматизації процесів V&V. Можливість забезпечення V&V формальної специфікації необхідна для перевірки коректності композиції і її характеристик. Важливо провести розмежування між поняттями верифікації та валідації. Під верифікацією розуміють перевірку параметрів формальної специфікації, а під валідацією - перевірку коректності формальної специфікації [28].

Облік QoS властивостей композиції дозволяє підвищити якість веб-сервісів, наприклад, параметром QoS може слугувати час відгуку примірника веб-сервісу [29].

Практичне застосовання визначає ефективність підходу в реальних умовах (У разі участі в процесі композиції сотень екземплярів веб-сервісів).

2.4.2 Процес композиції веб-сервісів

Для ілюстрації запропонованих критеріїв розглянемо один з можливих шляхів організації автоматизованої композиції веб-сервісів. Суть цієї організації полягає в послідовному виконанні наступних кроків:

1) графічна специфікація композиції з використанням мови UML;
2) формальна специфікація композиції із застосуванням TLA;
3) верифікація формальної специфікації за допомогою інструментарію TLA Model Checker;

4) валідація моделі композиції в середовищі моделювання DEVS Suite;
5) планування композиції (з оптимізацією за параметром QoS);
6) імітаційне моделювання композиції на основі агентного підходу.

Початковим кроком є графічна специфікація концептуальної моделі композитного веб-сервісу на мові UML, з метою визначення та формального відображення архітектури веб-сервісу, тобто основних його елементів і зв'язків між ними. Формальне графічне представлення архітектури підвищує ймовірність успішної валідації моделі, побудованої на основі формальної специфікації композиції.

Розглядаючи композицію як процес «end-to-end», важливо виробити набір правил трансляції з графічної специфікації в формальну. У більшості підходів, передбачають формальну специфікацію композиції (крок 2), в якості мови специфікації використовують темпоральну логіку LTL (Linear Temporal Logic). Припускається в якості логіки специфікації використовувати TLA (Temporal Logic of Actions). TLA має більш повний набір синтаксичних засобів опису взаємодії паралельних і розподілених процесів, що є критичним для формального опису композиції веб-сервісів. Іншою перевагою використання TLA є можливість формальної верифікації. У разі логіки LTL для формальної верифікації необхідно створення окремої специфікації верифікованих параметрів в пакеті CADP (Construction and Analysis of Distributed Processes). Ця специфіка вимагає додаткових зусиль і підвищує ймовірність помилок.

При використанні TLA специфікація та верифікація можуть бути виконані за допомогою інструментарію TLA Toolbox, який має спеціальний засіб для верифікації специфікацій TLA Model Checker [29]. Використання останнього передбачається на третьому кроці - формальної верифікації композитного веб-сервісу.

Четвертий крок - валідація формальної специфікації веб-сервіс у шляхом створення дискретно-подієвої імітаційної моделі і моделювання за допомогою її функціонування композитного веб-сервісу, наприклад в DEVS Suite. Для цього

необхідно розробити набір правил трансляції формальної специфікації в Java-код, описує DEVS-моделі. Опис можливої зворотної трансляції наведено в [30].

П'ятий крок процесу передбачає планування композиції. Цей крок також включає завдання пошуку примірників веб-сервісів і передбачає оптимізацію по параметру QoS (час відгуку примірника веб-сервісу). В якості заключного кроку виступає імітаційне моделювання процесу композиції на основі агентного підходу, що обґрунтовано введеним критерієм практичної застосовності і найбільшою відповідністю агентного підходу модельованої предметної області.

2.5 Висновки

Mashup – це цілком захоплюючий клас веб-додатків. Комбінація технологій моделювання, що відслідковуються від семантичного вебу та сервіс-орієнтованими, платформо-незалежними архітектурами і протоколами обміну даними, що забезпечують інструментами, необхідними для розробки сервісів, що можуть оперувати великими масивами даних, що доступні у веб. Композитні веб-додатки забезпечують і потребують кращу прозорість роботи, цікаво спостерігати за тим як цей клас сервісів впливає на права використання та інтелектуальну власність, тоді як інші сервіси інтегрують дані в межах організацій, наприклад грід-обчислення та business-to-business управління.

3. РОЗРОБКА КОМПОЗИТНОГО ВЕБ-ДОДАТКУ

3.1 Формулювання вимог до веб-додатку

Метою данної роботи є аналіз методів то принципів побудови композитних веб-додатків та їх застосування у наукових дослідженнях. Відповідно до мети були сформовані такі задачі:

- Вивчення та аналіз mashup концепції при побудові веб-додатків, її переваги та недоліки.
- Дослідження засобів, технологій та принципів для побудови композитних веб-додатків.
- Аналіз існуючих композитних веб-додатків. Пошук оптимальних і зручних рішень для побудови власного сервісу.
- Проектування власного веб-додатку на основі мети та задач зазначених вище.
- Пошук та вибір сервісів для компонування. Ознайомлення з їх API.
- Створення композитного вед-додатку
- Аналіз створеного рішення.

У розділі 2 «АНАЛІЗ КОНЦЕПЦІЇ ПОБУДОВИ КОМПОЗИЦІЙНИХ WEB-ДОДАТКІВ» було розглянуто різні концепції та підходи до побудови композитних веб-додатків. На основі його сформуємо вимоги до програми:

- Створення програми без використання mashup-конструкторів
- Отримання вхідних даних для роботи програми від користувача
- Поєднання генерації mashup контенту на стороні користувача та на стороні серверу
- Використання лише open API

- Використання веб-протоколу REST для обміну даними з провайдером сервісів для композиції

3.2 Аналіз існуючих композитних веб-додатків.

Використання mashup-технології стає все ширшим кожен день, тому сформувалося декілька категорій і типів таких веб-сервісів. Ми можемо виділити чотири основних:

- Картографічні
- Мультимедійні
- Пошукові та для Шоппінгу
- Новинні

Також існує безліч інших категорій, таких як Їжа, Спорт, Наука, Подорожі та інші. На рис. 3.1 зображена статистика mashup-сервісів відповідно до ресурсу programmableweb.com [31].

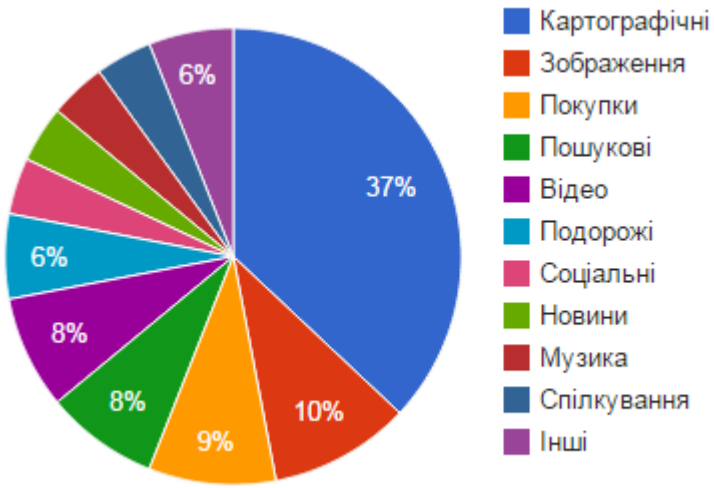

Рисунок 3.1 – Діаграма категорій mashup-сервісів

Як видно, наукові mashup не попадають в окрему категорію, бо їх процент занадто малий і вони внесені в категорію Інші. Проаналізуємо найпопулярніші категорії.

Картографічні композитні веб-додатки є найпопулярнішими поміж існуючих. Вони потребують джерела карт, що забезпечить графічне

відображення території, що відноситься до процесу. Коли найпопулярніші в світі карти Google Maps відкрили API – це дозволило веб-розробникам легко інтегрувати карти у власні сайти. Цей крок мав великий вплив на розвиток mashup і одразу за Google свої API представили такі сервіси як Yahoo Maps, MapQuest, Microsoft's Virtual Earth, роблячи процес використання карт будь-якого рівня тривіальним.

Ось кілька прикладів mashup карт:

- Places.ae – це веб-програма для пошуку практично будь чого Об'єднаних Арабських Еміратах. Вона використовує Google Maps та Twitter
- Gis.chicagopolice.org – це mashup, що інтегрує власну базу злочинів з Google Maps для запобігання злочинам у територіях і попередження громадян про їх місця
- Iamcaltrain.com – це mashup на базі Yahoo Maps. За допомогою розкладів потягів та фотографій сервісу Flickr він допомагає спланувати подорожі Каліфорнією. Інформує про особливості зупинок потягів, такі як парковки для велосипедів та місця їжі.

Фотографії та відео mashup базуються на відповідних провайдерах контенту та інших джерел даних, що можуть бути віднесені до відповідного мультимедійного контенту. Цей контент включає місця та локації де були зроблені відповідні фото та відео, що дозволяє відобразити ці місця на карті. Існують багато API для фото та відео, такі як Flickr, Yahoo Photos, Youtube та інші. Ось приклади відео та фото mashup-сервісів:

- ReelzReview.com – ціль роботи цього веб-додатку – скомпонувати інформацію про певний фільм, щоб допомогти користувачу вирішити, чи варто купувати певний фільм, чи ні. Він використовує відкритий API Amazon, YouTube, Yahoo, Ebay та інших ресурсів, щоб отримати зв'язані дані про нові релізи, огляди, ціни, пропозиції на одній веб-сторінці.
- FlickrMaps це mashup карт Yahoo та Flickr який перетворює вас у віртуального туриста. Оберіть місто і ви отримаєте відповідні фото з Flickr.

Композитні веб-додатки для шоппінгу існували задовго до того, як виникло поняття mashup. Замість використання API, існували порівняльні інструменти, такі як BizRate, PriceGrabber, Google's Froogle, які використовували комбінації business-to-business технологій для агрегації даних для порівняння.

Ідея mashup для пошуку та шоппінгу в порівнянні цін і специфікацій обєктів пошуку різними методами. Дані з декількох джерел відображаються разом для впевненості користувача.

- iPodRadar.com – це композитний веб-додаток для шоппінгу, який допомагає знайти iPod та аксесуари до нього. Окрім того, він повідомляє користувачів про новини пов'язаних блогів, посилань, програмного забезпечення та іншого. Він використовує API Google, Amazon eCommerce, BackPack, eBay, Google AdWords та Technorati API.

Композитні веб-додатки для новин полягають у тому, що вони відображують лише ті категорії новин та з тих ресурсів, які користувач хоче бачити. Це щось накшалт створення власної газети відповідно до його інтересів. У цих випадках використовуються технології RSS та ATOM, про які писалося вище. Приклади таких сервісів:

- Dogghot.us комбінує IT новини з Digg, Slashdot та del.icio.us, об'єднує або видаляє подібні записи та додає кілька власних особливостей.

- Aggreget.com індексує багато сторінок з таких ресурсів як Digg, Stumble, Delicious та подібних та відображає топ-10 популярних посилань, що перетинаються на цих ресурсах між собою, видаючи дійсно найпопулярніші теми.

3.3 Діаграма цілей бакалаврського дослідження

На рис 3.2 зображена діаграма цілей бакалаврського дослідження.

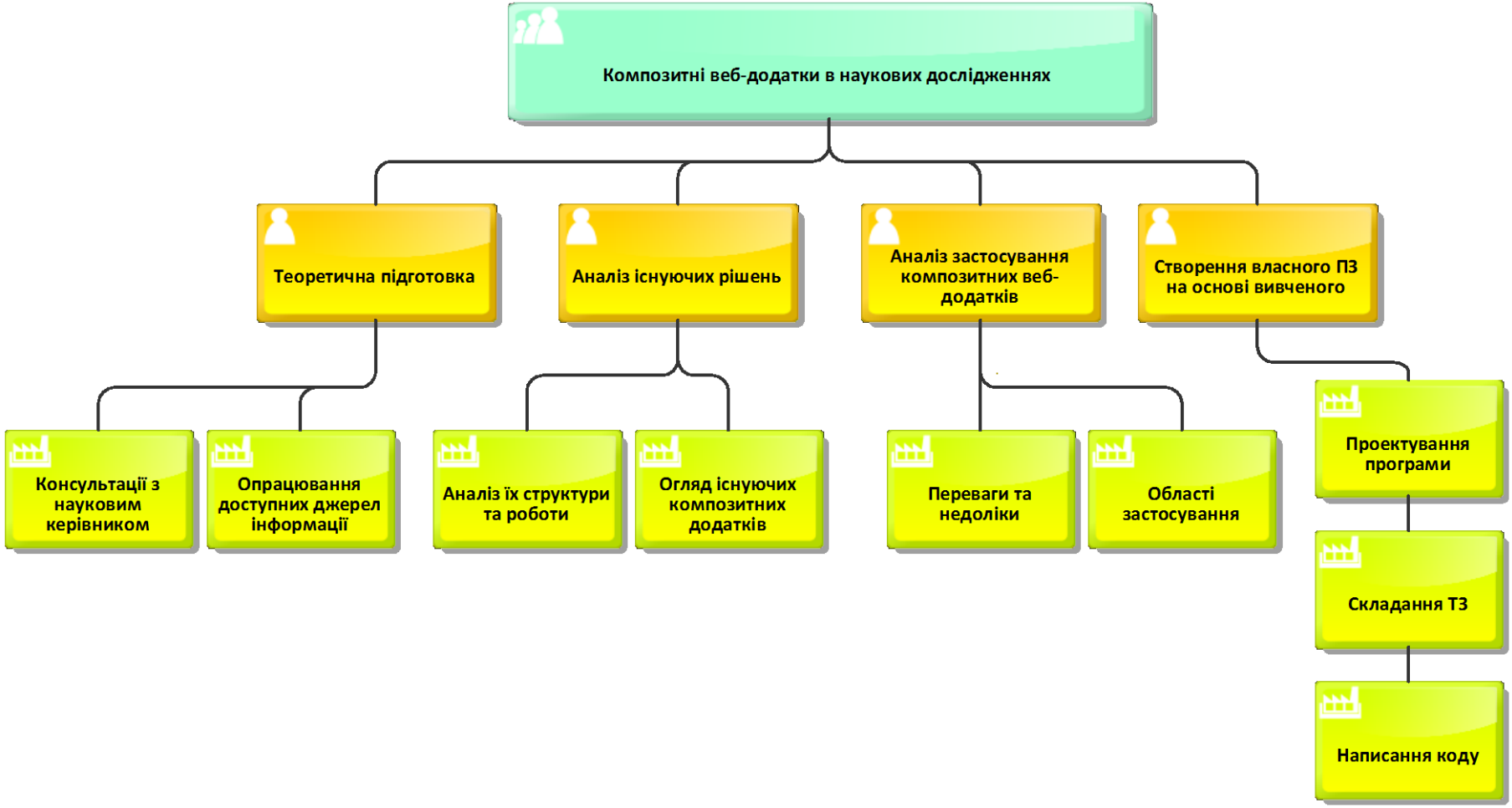

Рисунок 3.2 – Цілі бакалаврського дослідження

3.4 DFD діаграма

На рис 3.3 зображена DFD діаграма бакалаврського дослідження.

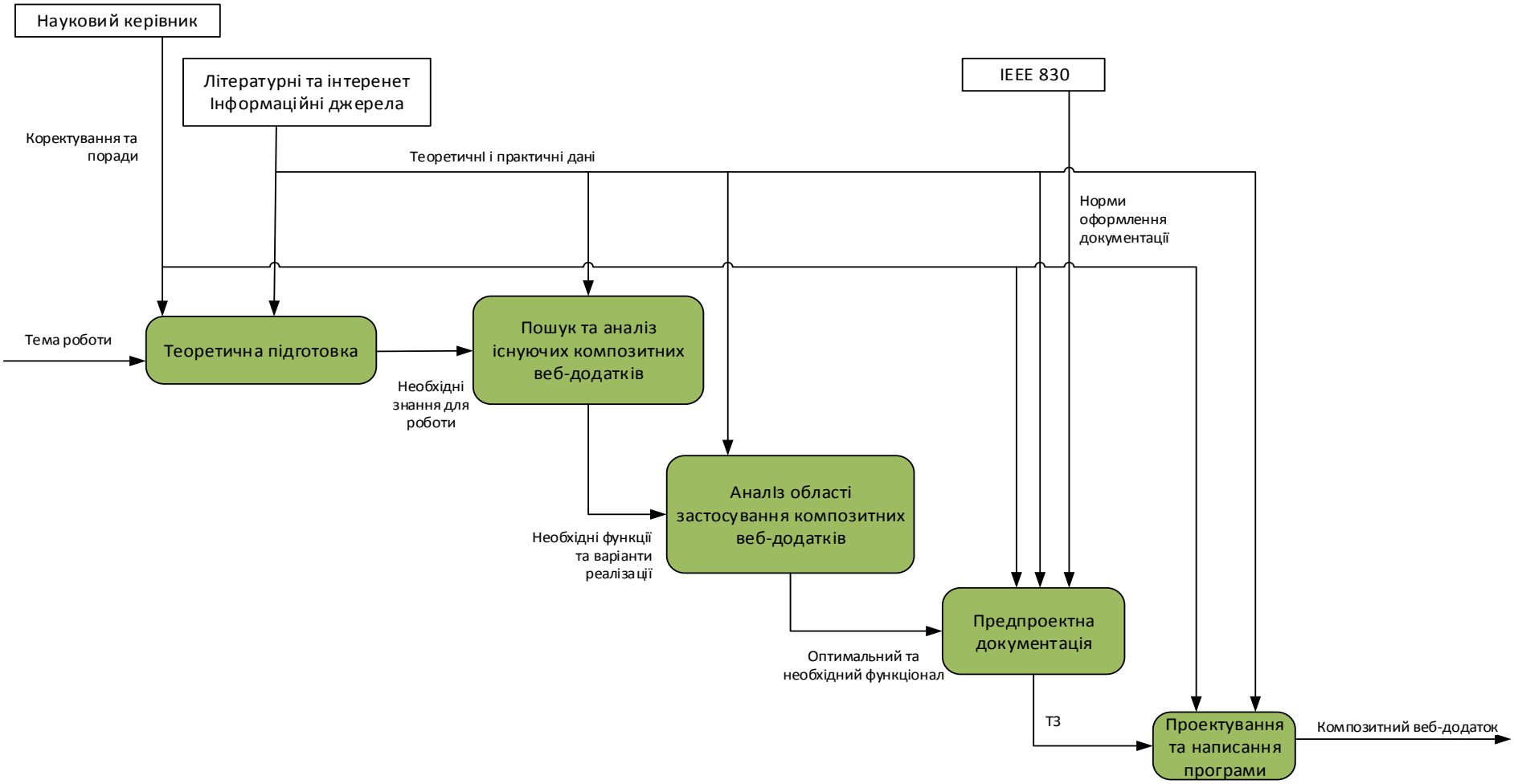

Рис. 3.3 DFD діаграма

3.5 Формування ідеї композитного веб-додатку. Визначення джерел інформації

3.5.1 Ідея веб-додатку

Протягом роботи над теоретичною частиною було вирішено зробити композитний веб-додаток що працював би з вибірками в математичній статистиці. Вона стосується наукових досліджень, а також є хорошим прикладом для демонстрації можливостей застосування mashup у цій області. Отже, відповідно до теми сформуємо функції, які мають бути реалізовані в програмі:

- Введення вибірки користувачем.
- Перевірка правильності введених даних.
- Визначення статистичних властивостей вибірки, а саме: об’єм вибірки, сума вибірки, медіана, середнє значення вибірки, дисперсія.
- Побудова гістограми вибірки.

Відповідно до сформованих задач та функцій було обрано наступні джерела інформації з відкритим API:

- Wolfram Alpha API
- Google Charts API

3.5.2 Огляд WolframAlpha API

WolframAlpha надає дуже потужний API. Він дає можливість працювати у кількох рівнях, починаючи від видання окремих результатів до формування повноцінних веб-сторінок.

Наведемо перелік технічних специфікацій та деталей:

- REST-стиль API
- Структурований XML вивід
- Синхронні та асинхронні операції

- Результати у вигляді тексту, зображень або HTML коду
- Повна підтримка Unicode

API дозволяє користувачу відправляти запити довільної форми, а результати запитів підтримують різноманіття форматів. Запити формуються у стандарті REST протоколу використовуючи запити HTTP GET. Для його використання необхідно зареєструватися, а також отримати AppID. AppID – це спеціальна строка, що ідентифікує ваш додаток, вона має бути у кожному запиті.

Наведемо приклад найпростішого запиту до API:
«http://api.wolframalpha.com/v2/query?input=YYYY&appid=XXXX
&podtitle=ZZZZ&format=RRRR&TTTT=3»

Замість YYYY має стояти ваш запит, а замість XXXX – ваш AppID. Також може передаватись безліч додаткових параметрів, але не будемо на них зупинятись.

З WolframAlpha API можна працювати використовуючи будь яку-мову програмування, що підтримує веб-запити та XML. Детальніше можна ознайомитись з офіційною документацією на сайті [32].

3.5.2 Огляд Google Charts API

Google Charts API забезпечує ідеальну можливість візуалізувати дані для будь-якого веб-сайту. Починаючи від простих діаграм до складних ієрахічних дерев, сервіс надає велику кількість готових типів діаграм.

Найпростіший шлях використання Google Charts – це звичайний JavaScript, який можна вбудувати в сторінку. Усе що для цього потрібно – завантажити Google Charts бібліотеки, мати дані для побудови діаграм та визвати потрібну діаграму за допомогою функції. Згодом вставляється з потрібним ідентифікатором.

Діаграми – це набір JavaScript класів. Ви можете модифікувати їх завласним бажанням. Для побудови зображення використовуються технології HTML5/SVG, щоб забезпечити кросс-браузерну сумісність та кросс-

платформенну підтримку без використання плагінів або іншого програмного забезпечення.

Усі діаграми створюються за допомогою класу DataTable, створюючи можливість легкого переходу від одного типу діаграм до іншого. Також цей клас забезпечує методи сортування, модифікування та фільтрації даних.

Для відображення діаграм необхідні три бібліотеки:

- The Google JSAPI API
- The Google Visualization
- Сама бібліотека для відображення

Наведемо приклад двох скриптів для завантаження цих бібліотек

```
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<script type="text/javascript">
  google.load('visualization', '1.0', {'packages':['corechart']});
  google.setOnLoadCallback(drawChart);
</script>
```

Детальніше з документацією можна ознайомитись на офіційному сайті [33].

3.6 Написання композитного веб-додатку

Серверна частина програми реалізована мовою PHP. Це скриптова мова програмування, що була створена для генерації HTML-сторінок на стороні веб-сервера. PHP є однією з найпоширеніших мов, що використовуються у сфері веб-розробок (разом із Java, .NET, Perl, Python, Ruby). PHP підтримується переважною більшістю хостинг-провайдерів. PHP — проект відкритого програмного забезпечення.

PHP інтерпретується веб-сервером у HTML-код, який передається на сторону клієнта. На відміну від скриптової мови JavaScript, користувач не бачить PHP-коду, бо браузер отримує готовий html-код. Це є перевага з точки зору

безпеки, але погіршує інтерактивність сторінок. Але ніщо не забороняє використовувати PHP для генерування і JavaScript-кодів які виконуються вже на стороні клієнта.

Отже, структура програми виглядає наступним чином:

1. PHP-скрипт на сервері отримує запит від клієнта у формі вибірки
2. PHP-скрипт на основі отриманих даних формує запит до WolframAlpha API. Отримує відповідь від нього. Перевіряє отримані результати. На базі них формує веб-сторінку з результатами відповіді WolframAlpha API.
3. Веб-сторінка містить JavaScript-код, який виконується вже на стороні клієнта. Цей код підключає Google Charts API, що складається з трьох бібліотек, формує запит до нього. Результатом запиту є гістограма вибірки клієнта, яка відображається на сформованій веб-сторінці.

На рис. 3.4 зображена діаграма прецедентів програми.

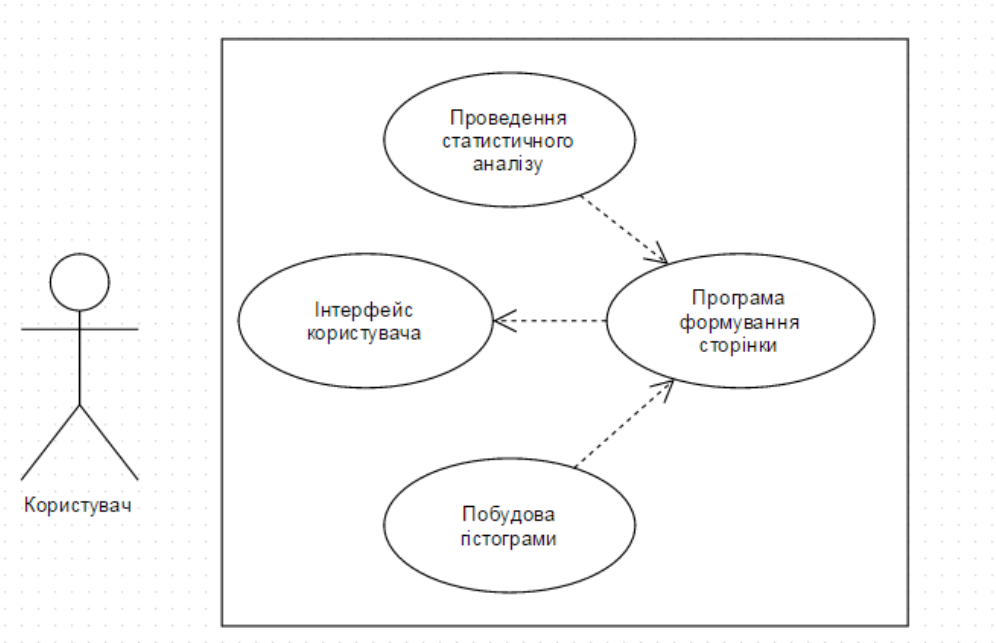

Рисунок 3.4 - Діаграма прецедентів

Код програми наведений у Додатку А.

3.6 Результати роботи програми

На рис 3.5 зображено поле введення вибірки. Елементи вибірки мають розділятися комами. Допустимі як елементи цілочисленні, так і дробові. На рисунку введено вибірку 1, 5, 8, 4, 2, що послужить нам для тестування програми

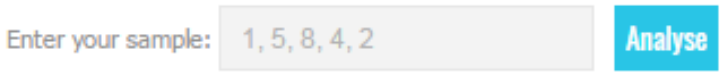

Рисунок 3.5 – Поле введення вибірки

Далі виводяться результати роботи програми. Першими є результати запиту до WolframAlpha API. За звертаннями до цього інтерфейсу виводяться наступні результати, що зображені на рис. 3.6. Цими результатами є вивід вхідної вибірки для забезпечення впевненості користувача у введених даних, сума вибірки. Окремо виводяться статистичні дані: середнє значення, медіана та дисперсія.

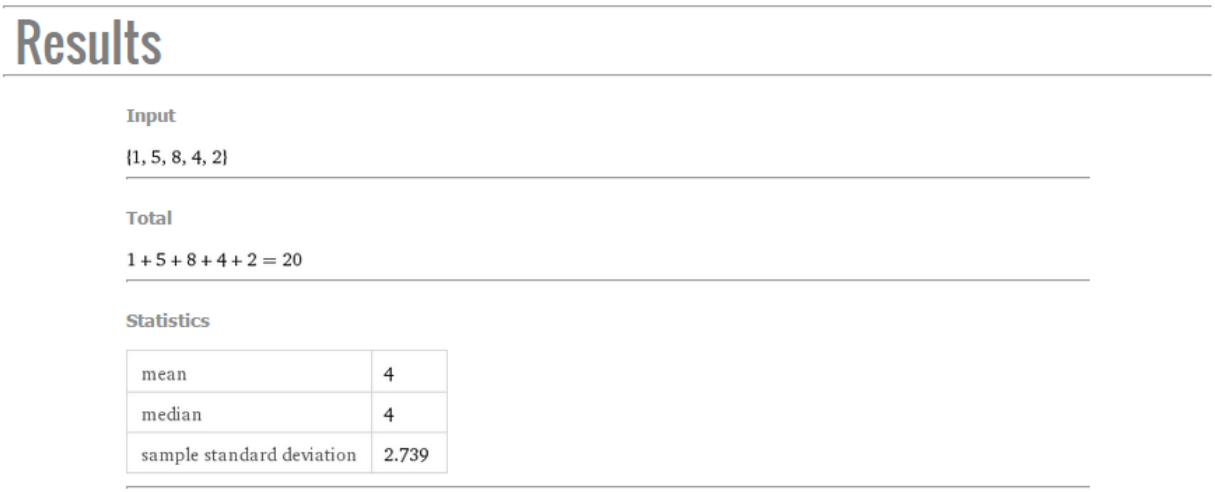

Рисунок 3.6 – Результати отримані від WolframAlpha API

Останніми виводяться результати роботи композитного веб-сервісу з Google Charts API. Цими результатами є гістограма вибірки, що зображена на рис.3.7. При наведенні на певний стовпець висвічується точне його значення, а отже, і точне значення елемента вибірки.

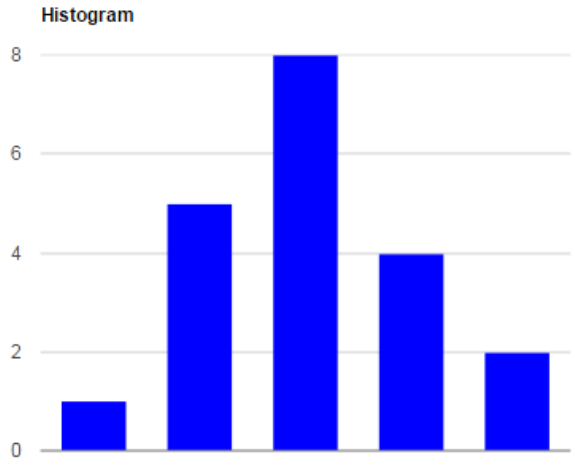

Рисунок 3.7 – Результати отримані від Google Charts API

Інший приклад роботи веб-сервісу на рис. 3.8

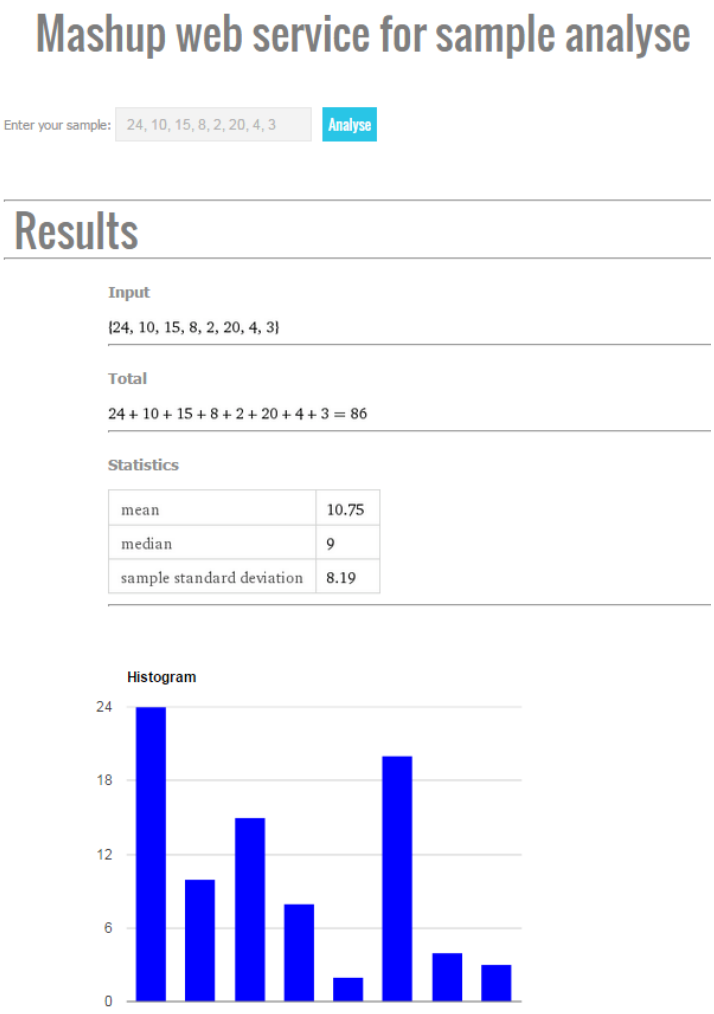

Рисунок 3.8 - Приклад роботи сервісу

3.6 Висновки

У цьому розділі були досліджені існуючі композитні веб-додатки, категорії їх застосування. Як видно, застосування технології у наукових дослідженнях не є популярним, хоча має дуже високий потенціал. На основі отриманих знань і досвіду був створений власний композитний веб-додаток, що може застосовуватись в наукових дослідженнях.

Mashup був створений на основі WolframAlpha API та Google Charts API. Програма була створена за допомогою мов PHP, JavaScript, мови розмітки HTML. Вона аналізує вибірку, будує гістограму, вираховує дисперсію, значення та медіану. Тобто створений композитний веб-додаток цілком відповідає поставленим задачам

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Вступ

Мета даного розділу – аналіз обраного приміщення на відповідність нормам охорони праці і безпеки в надзвичайних ситуаціях, оскільки ці питання мають першочергові значення

Охорона праці є невід’ємним складником умов трудової діяльності. Основні положення закріплено в Законі «Про охорону праці», в якому дано їй визначення - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

Комфортні та безпечні умови для працівника значно підвищують рівень його ефективності. Крім того, іноді вони виступають важливим фактором при виборі робочого місця, тому роботодавець має бути зацікавлений в створенні сприятливих умов та застосуванні сучасних засобів безпеки для своїх підлеглих. Працівник має право відмовитись від роботи поєднаної з небезпекою для життя або в умовах, що не відповідають нормам законодавства.

4.2 Аналіз умов праці. Оцінка санітарно-гігієнічних умов праці

Робоче приміщення – це серверна з двома робочими місцями для персоналу. План приміщення наведений на рис. 4.1.

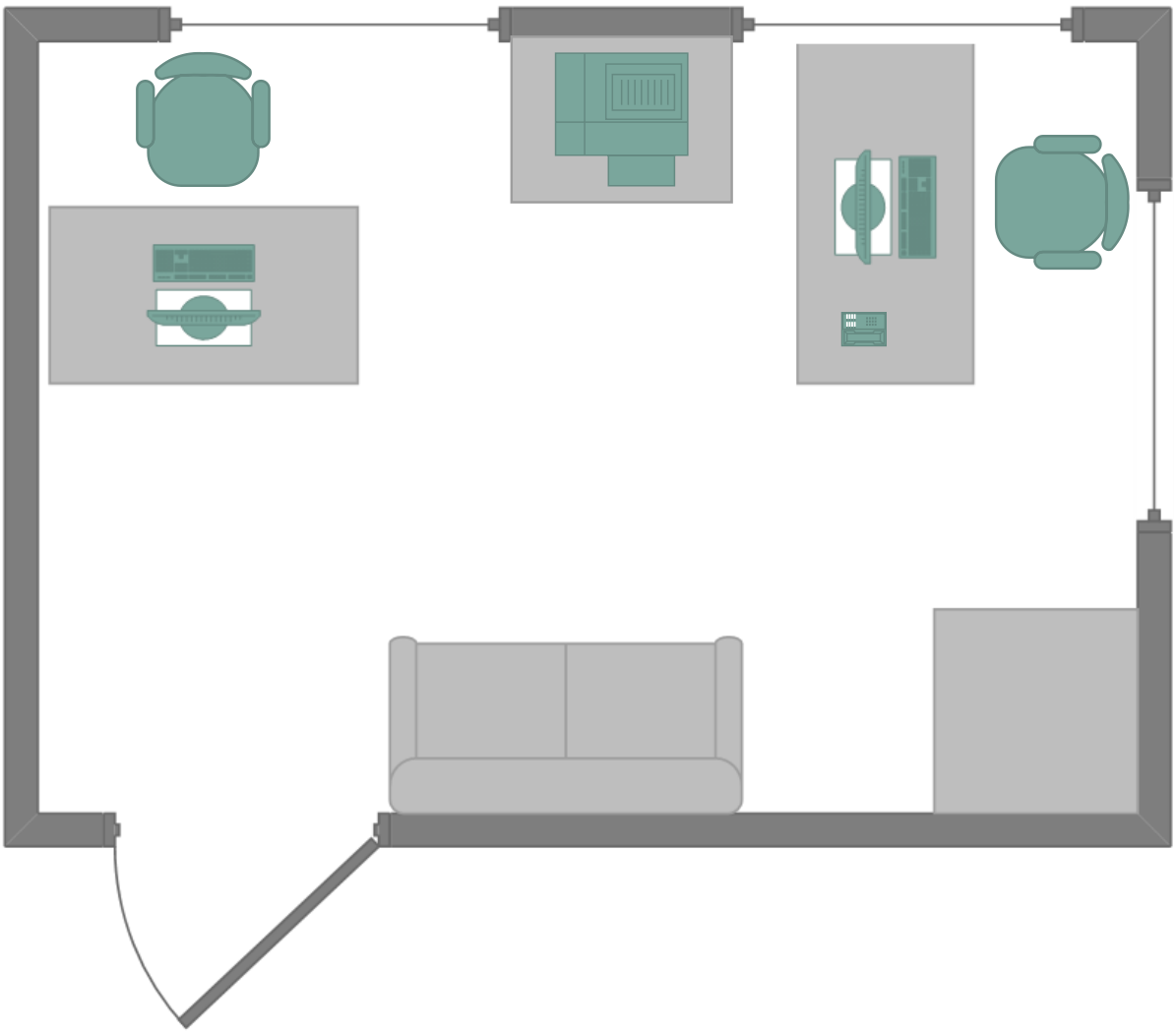

Рисунок 4.1 – План робочого приміщення

Характеристики приміщення:

- Довжина – a=5 м;
- Ширина – b=3.5 м;
- Висота стелі – h=3.2 м;
- Площа – S=a * b=5 * 3.5= 17.5 м;
- Об’єм – V=S * h=17.5 * 3.2= 56 куб.м;
- Кількість робочих місць – N=2;

Відповідно до норм, площа повинна складати не менше 6 кв. м на людину, а об’єм – 20 куб.м. [34].

Розрахуємо ці показники для нашого приміщення:

$$S' = \frac{S}{N} = \frac{17.5}{2} = 8.75(\frac{м^2}{люд.})$$

$$V' = \frac{V}{N} = \frac{56}{2} = 28(\frac{м^3}{люд.})$$

Отже, приміщення відповідає нормам з площі і об’єму.

Тепер розглянемо робоче місці на відповідність нормативам. Для цього побудуємо (табл. 4.1) з фактичними і нормативними значеннями та порівняємо їх.

Таблиця 4.1 – Характеристика робочого місця

Найменування параметра	Значення	
	фактичне	нормативне
Висота робочої поверхні, мм	720	680 ÷ 800
Висота простору для ніг, мм	700	не менше 600
Ширина простору для ніг, мм	800	не менше 500
Глибина простору для ніг, мм	650	не менше 650
Висота поверхні сидіння, мм	470	400 ÷ 500
Ширина сидіння, мм	450	не менше 400
Глибина сидіння, мм	450	не менше 400
Висота поверхні спинки, мм	600	не менше 300
Ширина опорної поверхні спинки, мм	450	не менше 380
Радіус кривини спинки в горизонтальній площині, мм	400	400
Відстань від очей до екрану дисплея, мм	800	700 ÷ 800

Неправильна організація робочого місця сприяє загальній і локальній напрузі м'язів шиї, тулуба, верхніх кінцівок, скривленню хребта й розвитку остеохондрозу.

Відстань між бічними поверхнями комп‘ютерів має бути не меншою за 1,2 м. Відстань між тильною поверхнею одного комп‘ютера та екраном іншого не

повинна бути меншою 2,5 м для. Дані нормативні значення приведено для дисплеїв на основі електронно-променевої трубки, для плоских дискретних дисплеїв (LCD, LED-монітори) в українських нормативних документах не передбачено окремих нормативів, однак плоскі дискретні монітори дають менш потужне електро-магнітне поле. Прохід між рядами робочих місць має бути не меншим 1 м. Приміщення розташування робочих місць цілком задовольняє всім нормам [35].

4.3 Аналіз мікрокліматичних умов

Мікроклімат має значний вплив на працездатність. Мікроклімат робочих приміщень – це клімат внутрішнього середовища цих приміщень, що визначається діючої на організм людини з'єднанням температури, вологості, швидкості переміщення повітря.

Параметри, за якими оцінюється мікроклімат, встановлюється відповідно до пори роки і категорії роботи. У табл. 4.2 наведені оптимальні значення параметрів мікроклімату для категорії робіт 1а, а також фактичні значення цих параметрів у досліджуваному приміщенні [36].

Таблиця 4.2 - Параметри мікроклімату на робочому місці

Період року	Параметр	Оптимальний	Фактичний
Теплий	Температура	23 – 25 °С	24-26 °С
	Вологість	40 – 60 %	40 %
	Швидкість повітря	≤ 0.1 м/с	
Холодний	Температура	22 – 24 °С	21-23 °С
	Вологість	40 – 60 %	50 %
	Швидкість повітря	≤ 0.1 м/с	

Всі показники задовольняють вимогам для робіт категорії легка Іа і є задовільними для здоров’я людини.

Отже, фактичне значення освітленості потрапляє в допустимі 10% відхилення від нормативного показника, тобто – відповідає нормам.

4.4. Шум у робочому приміщенні

Шум в робочому приміщенні негативно впливає на працездатність працюючих. Основними фізичними параметрами звуку, що нормуються є інтенсивність, звуковий тиск і частота коливань. Нормування шумів, ультра- та інфразвуків здійснюється згідно ДСН 3.3.6.037-99. Відповідно до цих норм, рівень шуму не має перевищувати 50 дБ [37].

Можливий список джерел шуму список джерел шуму у нашому приміщенні:

- система охолодження ПЕОМ;
- принтер під час операцій друку.
- шум вуличного транспорту;
- кондиціонер
- система охолодження серверу;

Сумарний рівень інтенсивності звуку можна розрахувати за формулою:

$$L = 10 \cdot lg\left(\frac{1}{T} \cdot \sum_{i=1}^{n} t_i \cdot 10^{0.1 \cdot L_i} \right)$$

де

T – робочий час протягом дня;

ti – час надходження звуку від і-го джерела;

Li – рівень звукового тиску і-го джерела.

В (табл. 4.3) наведені джерела шуму, рівень звукового тиску та час дії протягом робочого дня

Таблиця 4.3 – Джерела шуму

Джерело шуму	Рівень шуму La, дБА	Час дії шуму t, год	Кількість джерел шуму N
Зовнішній шум	35	8	1
Кондиціонер	34	4	1
Струменний принтер	60	0.5	1
Система охолодження ПК	32	8	2
Система охолодження серверу	40	8	1

Визначимо $L_{екв.}$ еквівалентний рівень шуму за 8 робочих годин:

$$L_{\text{екв}} = 10 * lg\frac{1}{T}\sum t_i * 10^{0.1*La} == 10 \cdot lg\frac{1}{8} \cdot (10^{3.5} + 10^{3.4} + 10^{6} + 2 * 10^{3.2} + 10^{4}) \approx 41 \text{ дБА}$$

Отже, максимально можливий рівень звукового тиску та рівень звуку на робочих місцях відповідає вимогам, так як в приміщеннях управління та робочих кімнатах допустимий еквівалентний рівень звуку менше норми.

4.5 Аналіз освітлення

Для приміщень, в яких робочі місця обладнано ПЕОМ, важливо організувати правильні умови освітлення. Нормування умов освітлення здійснюється згідно будівельних норм [38].

Освітлення приміщення здійснюється за допомогою штучного та природного освітлення. Згідно плану приміщення рис. 4.1, кабінет, який ми аналізуємо, має 3 вікна з лінійними розмірами: ширина – 1,6 м, висота – 1,8 м, відповідно площа кожного вікна – 2,88 м^2. Вікна мають регульовані пристрої для відкривання та обладнані жалюзями з можливістю захисту працюючих від прямого попадання сонячних променів і регулювання рівня освітленості в приміщенні.

Стіни обклеєні світлими шпалерами, стеля білого кольору, у якості підлогового покриття використаний матовий ламінат з коефієнтом відбиття 0,3-0,4. Відблискування поверхонь обмежується за рахунок правильного вибору світильників та розташування робочих місць відносно джерел освітлення. Яскравість відблисків на сучасних моніторах не перевищує 35 кд/м2

В досліджуваному приміщенні використовується система загального рівномірного штучного освітлення. Мається два ряди світильників Л201Б 2х40-0.3, у кожному з яких знаходиться по чотири лампи типу ЛБ-40. Їх технічні характеристики:

- потужність – 40 Вт;
- напруга на лампі – 103 В;
- світловий потік: номінальний – 3120 лм, мінімальний – 2810 лм;
- довжина лампи: без штирків – 1199.4 мм, із штирками – 1213.6 мм;
- діаметр – 40 мм.

План освітлення наведений на (рис. 4.2)

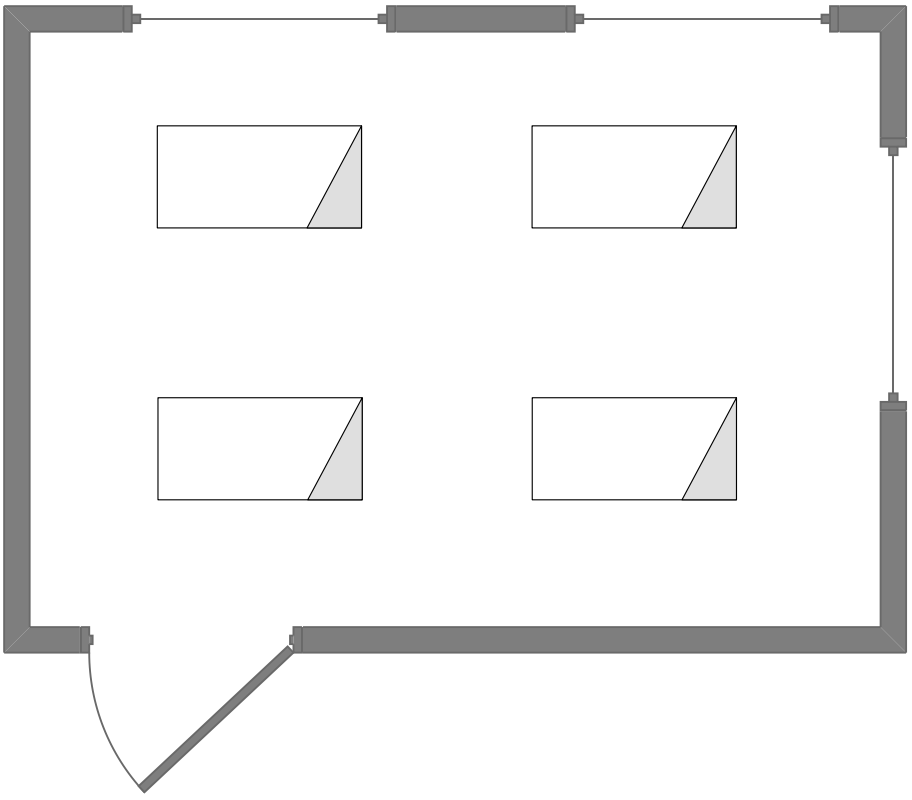

Рисунок 4.2 – План освітлення

Освітлення достатньо рівномірно розподілено в приміщенні, завдяки комбінації штучного та природнього освітлення, уникаються різкі тіні, а системи його регулювання забезпечують такий стан протягом усього дня.

4.6 Пожежна безпека

Будівля II ступені вогнестійкості всі конструкції (стіни, перекриття, покриття, перегородки) виготовлені з негорючих матеріалів з межею вогнестійкості від 0,25 год. до 4 год

В приміщенні, що аналізується, наявні такі потенційні джерела пожежної небезпеки: ПЕОМ, світильники, принтер, кондиціонер, а також електрична проводка і розетки, що забезпечують постачання електроенергії. Також тут присутні меблі з горючих і легкозаймистих матеріалів (ДСП, дерево, пластмаса, синтетичні тканини), папір, ламінат. В робочому приміщенні відсутні горючі рідини.

Згідно з НАПБ Б.03.002-2007 дане приміщення можна віднести до категорії В – приміщення, що містить горючі тверді, волокнисті матеріали [39]. Сама будівля

Відповідно до списку потенційних джерел пожежної небезпеки, наведеного вище, пожежа може виникнути в результаті наступних ситуацій:

- Несправність електромережі, коротке замикання, пошкодження захисних оболонок електромережі, перевантаження електричної мережі, несправності споживачів електроенергії.
- Порушення правил пожежної безпеки збоку працівників: паління в кабінеті, використання побутових нагрівачів, тощо.

Для уникнення такої ситуації, кожен працівник має проходити первинний і повторні інструктажі з правил пожежної безпеки згідно чинних нормативних актів, а також в приміщенні потрібно розміщувати пам’ятку у вигляді витягу з правил пожежної безпеки для постійного нагадування останніх працівникам.

Крім того слід використовувати такий комплекс заходів:

- заборона використання відкритого вогню у приміщенні;
- наявність системи автоматичної пожежної сигналізації з димовими пожежними оповіщувачами;
- ступінь вогнестійкості будівлі, у якій розташовано приміщення – II;
- наявність шляхів евакуації при виникненні пожежі;
- розміщення схеми евакуації людей при пожежі і ознайомлення з нею персоналу.

Згідно з ППБУ, в приміщенні, що аналізується, необхідна присутність наступного:

1. Переносні засоби пожежогасіння: по одному вуглекислотному вогнегаснику з величиною заряду вогнегасної речовини 3кг і більше на кожні 20 $м^2$ площі приміщення, де використовується ПЕОМ. В нашому випадку, це має бути як мінімум 2 вогнегасники марки ВВ-5 (величина заряду вогнегасної речовини 3,5 кг).
2. План евакуації з приміщення.
3. Система протипожежних датчиків, пожежна сигналізація.

Приміщення має один вихід, оскільки в ньому працює менше 25 чоловік. Ширина проходу між робочими місцями у приміщенні перевищує 1 м. Сходова клітка має природне освітлення в комбінації зі штучним. Сходи та приміщення обладнані системою евакуаційного освітлення. Співробітники ознайомлені з порядком і планом евакуації.

Отже, шляхи евакуації з приміщення повністю відповідають нормам.

4.7 Електробезпека

Проаналізуємо приміщення на можливість ураження персоналу електричним струмом. Визначимо групу електронебезпечності даного приміщення. Ознаки підвищеної небезпеки ураження електрострумом:

- наявність вологості;
- наявність температури більш ніж 35 ºС;
- наявність струмопровідного пилу;
- наявність струмопровідної підлоги;
- можливість одночасного дотику до корпусів чи струмопровідних елементів та до елементів, що мають зв’язок з землею.

Ознаки особливої небезпеки ураження електрострумом:

- наявність особливої вогкості;
- наявність хімічно активного середовища.

Наше приміщення не має жодної ознаки особливої або підвищеної небезпеки ураження персоналу струмом. Тому за групою електронебезпечності воно відноситься до приміщень без підвищеної небезпеки ураження струмом.

Споживачі електроенергії: 2 ПЕОМ, 2 дисплеї, 1 принтер, 4 світильники, 1 кондиціонер та 1 сервер. Кожне робоче місце обладнане 4-ма розетками по 220 В, окремо існують розетки для кондиціонеру та серверу. Всі прилади використовують саме цю напругу. Усі кабелі ізольовані. Заземлені конструкції захищені діелектричними сітками від випадкового дотику. Усе електроустаткування має апаратуру захисту від струму короткого замикання.

Лінія електромережі для живлення ЕОМ та периферійних пристроїв ЕОМ виконується як окрема групова три-провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників.

При виконанні робіт по ремонту і обслуговуванню ПЕОМ обслуговуючий персонал зобов'язаний керуватися “Правилами техніки безпеки при експлуатації електроустановок споживачами". До роботи не допускаються особи, які не пройшли навчання з техніки безпеки.

Даний кабінет задовольняє вимоги щодо електробезпеки у приміщенні, в якому встановлені ЕОМ, відображені в НПАОП 0.00-1.28-10.

4.8 Рекомендації щодо поліпшення умов праці

У зв’язку зі специфікою робіт з ЕОМ можна порекомендувати виконання комплексів вправ для психічного та психологічного розвантаження.

При інтенсивній роботі з вхідними даними, редагуванні програм, читанні інформації з екрану монітора безперервна тривалість роботи не повинна перевищувати 4-х годин (при 8-годинному робочому дні). Задля зниження напруженості праці необхідно, якщо це можливо, рівномірно розподіляти навантаження і раціонально чергувати характер діяльності.

Варто щогодини робити перерву на 15 хвилин. Один або кілька разів у годину необхідно виконувати серію легких вправ на розтягування, що можуть зменшити напругу, накопичену в м’язах при тривалій роботі за комп’ютером. Рекомендується робити вправи для м’яз очей, оскільки тривала робота за комп’ютером може значно погіршити його стан.

З інших рекомендацій щодо поліпшення умов праці відповідно до можна навести наступні:

- у приміщенні слід щоденно проводити вологе прибирання;
- у приміщенні повинні бути медичні аптечки першої допомоги.

4.9 Висновки

У даному розділі були розглянуті умови праці для виробничого приміщення, у якому операторами експлуатуються програмний продукт. Розміри приміщення та параметри робочих місць відповідають нормам чинного законодавства з охорони праці.

Приведені рекомендації щодо організації робочого місця на підприємстві дозволяють підвищити рівень безпеки праці, попередити виникнення надзвичайних ситуацій та надати першу медичну допомогу при виникненні надзвичайної ситуації. Служби охорони праці, а саме відповідні служби і структурні підрозділи підприємства повинні здійснювати постійний

контроль за виконанням робіт у відповідності з вимогами з охорони праці, електро-, газо- і пожежобезпеки, не допускати до роботи осіб, які не пройшли інструктаж та не здали заліки по питаннях охорони праці.

Було проведено аналіз шкідливих та небезпечних виробничих чинників, наявних у даному приміщенні. Значення параметрів мікроклімату, виробниче освітлення та засоби пожежної безпеки приміщення відповідають необхідним нормативам.

ВИСНОВКИ

В даній роботі були аналізовані методи та принципи побудови композитних веб-додатків та їх застосування у наукових дослідженнях. В роботі була вивчена архітектура SOA, вивчена та проаналізована mashup концепція побудови концепція побудови веб-додатків. Також були проаналізовані існуючі композитні веб-додатки та був виконаний пошук зручних рішень для побудови власного сервісу. Для побудови власного веб-сервісу було ознайомлено з API відповідних сервісів.

Архітектура SOA абсолютно незалежна від мов програмування, платформ або протокольних специфікацій, за допомогою яких сервіси розробляються, а також від того, де і за допомогою чого вони розгорнуті. Практично архітектура SOA вимагає наявності не лише сервісів, але і засобів, за допомогою яких ці сервіси можуть бути виявлені і підключені незалежно від інфраструктури.

Mashup – це цілком захоплюючий клас веб-додатків. Комбінація технологій моделювання, що відслідковуються від семантичного вебу та сервіс-орієнтованими, платформо-незалежними архітектури і протоколами обміну даними, що забезпечують інструментами, необхідними для розробки сервісів, що можуть оперувати великими масивами даних, що доступні у веб. Композитні веб-додатки забезпечують і потребують кращу прозорість роботи, цікаво спостерігати за тим як цей клас сервісів впливає на права використання та інтелектуальну власність, тоді як інші сервіси інтегрують дані в межах організацій, наприклад грід-обчислення та business-to-business управління.

Mashup був створений на основі WolframAlpha API та Google Charts API. Програма була створена за допомогою мов PHP, JavaScript, мови розмітки HTML. Вона аналізує вибірку, будує гістограму, вираховує дисперсію, значення та медіану. Тобто створений композитний веб-додаток цілком відповідає поставленим задачам.

Працюючи над дипломною робою я набув досвіду, який є корисним у моєму професіональному розвитку і знадобиться мені у подальшому.

ПЕРЕЛІК ПОСИЛАНЬ

1. Дерев'янко А.С. Технології та засоби консолідації інформації: навчальний посібник/ А.С. Дерев'янко, М.Н. Солощук. - Харків: НТУ "ХПІ", 2008. - 432с.
2. Компас у світі сервіс-орієнтованої архітектури (SOA): цінність для бізнесу/ [Біберштейна Н., Боуз С., Джонс К., Фіаммант М., Ша Р]. - М .: КУДИЦ-ПРЕСС, 2007. - 256 с.
3. Фастовский Э. Г. Сервис-ориентванные технологии интеграции информации: [Електронний ресурс]. - Режим доступу : http://khpi-iip.mipk.kharkiv.edu/library/sotii/lectures/Lecture4.pdf. – Дата доступу: 13.02.2015
4. Software as a Service: Strategic Backgrounder: [Електронний ресурс]. – Режим доступу: http://www.siia.net/estore/ssb-01.pdf. – Дата доступу: 28.02.2015
5. UP2010 Virtual Cloud Conference – Microsoft’s PaaS Solution: [Електронний ресурс]. - Режим доступу: http://channel9.msdn.com/posts/UP2010-Virtual-Cloud-Conference--Microsofts-PaaS-Solution. – Дата доступу: 14.02.2015
6. ORACLE Infrastructure as a service (IaaS) with capacity on demand: [Електронний ресурс]. – Режим доступу: http://www.oracle.com/us/products/engineered-systems/iaas/iaasexec-brief-1908773.pdf. – Дата доступу: 24.02.2015
7. Dijkstra E. W. Cooperating Sequential Processes in Programming Languages/ E.W. Dijkstra, F.Genuys. – NY: Academic Press, New York, 1968. – 405p.
8. Математика и САПР / [П. Шенен, М. Коснар, И. Гардан и др.]. – М.: Мир, 1988.- 204 с.
9. Сервісно-орієнтоване програмування: [Електронний ресурс]. - Режим доступу: http://www.programsfactory.univ.kiev.ua/content/books/2/68. – Дата доступу: 14.02.2015

10. Сервіс-орієнтована архітектура: [Електронний ресурс]. - Режим доступу: http://pidruchniki.com/1383121947801/informatika/servis-oriyentovana_arhitektura. – Дата доступу: 15.02.2015
11. Горлачук В. В. Оптимізація бізнес-процесів підприємства / В.В. Горлачук, І.Г. Яненкова/ [Електронний ресурс].- Режим доступу: http://lib.chdu.edu.ua/pdf/posibnuku/294/111.pdf. – Дата доступу: 16.02.2015
12. Радченко Г. И. Распределенные вычислительные системы / Г. И. Радченко. – Челябинск: Фотохудожник, 2012. – 184 с.
13. Обзор терминологии SOA: Часть 1. Сервис, архитектура, управление и бизнес- термины: [Електронний ресурс]. – Режим доступу: http://www.ibm.com/developerworks/ru/library/ws-soa-term1/. – Дата доступу: 10.05.2015
14. Network Virtualisation – Opportunities and Challenges: [Електронний ресурс]. – Режим доступу: http://archive.eurescom.eu/~pub/deliverables/documents/P1900-series/P1956/D1/P1956-D1.pdf. – Дата доступу: 8.05.2015
15. FG-coud-technical-report: [Електронний ресурс].– Режим доступу: http://www.itu.int/en/ITU-T/focusgroups/cloud/Documents/FG-coud-technical-report.zip. – Дата доступу: 05.02.2015
16. Коновалов О.Ю. Сервіс-орієнтована архітектура у розподілених обчислювальних системах: [Електронний ресурс] / О.Ю. Коновалов// Вісник ДУІКТ. – 2013. – №4- сс. 53-61. – Режим доступу: http://www.dut.edu.ua/uploads/p_548_64347041.pdf. – Дата доступу: 11.06.2015
17. 7 Things You Should Know About Mapping Mashups: [Електронний ресурс]. Режим доступу: http://www.educause.edu/library/resources/7-things-you-should-know-about-mapping-mashups. – Дата доступу: 10.03.2015
18. O'Reily T., 2005. What Is Web 2.0. Online: [Електронний ресурс]/ Tim O'Reily. Режим доступу: http://www.oreilly.com/pub/a/ Web2/archive/what-is- web-20.html. – Дата доступу: 8.03.2015
19. Web And Information Security / [Weitzner, Hendler, Berners-Lee, Connolly] / – NY: IRM Press, 2005. – 307 p. . – Дата доступу: 28.03.2015

20.Chang Z. Architectural Design and Prototyping of a web-based Synchronous Collaborative 3D GIS / Z. Chang, S. Li // *Cartography and Geographic Information Science* -2008.- №35(2)-pp. 117-132.

21. Немного о Mashup: [Електронний ресурс].– Режим доступу: http://digipo.eu/nemnogo-o-mashup.html. – Дата доступу: 21.04.2015

22. Personal blog: Why Mashups = (REST + 'Traditional SOA') * веб 2.0: [Електронний ресурс].– Режим доступу: http://blog.sherifmansour.com/?p=187. – Дата доступу: 28.05.2015

23.WEBMASHUP: [Електронний ресурс].– Режим доступу: http://www.webmashup.com/. – Дата доступу: 14.05.2015

24. *Feiler J.* How to Do Everything with Web 2.0 Mashups/*Jesse Feiler*— McGraw Hill Professional, 2008. — 303 p. . – Дата доступу: 01.04.2015

25. Merrill D. Mashups: The new breed of web app. / Duane Merrill / [Електронний ресурс].– Режим доступу: http://www.ibm.com/developerworks/library/x-mashups/. – Дата доступу: 23.04.2015

26.Mashup-приложения - эволюция SOA: Часть 2. Ситуативные приложения и mashup-экосистема / Стивен Уатт/[Електронний ресурс].– Режим доступу: http://www.ibm.com/developerworks/ru/library/ws-soa-mashups2/. – Дата доступу: 09.03.2015

27.Шкарупило В.В.О критериях анализа процессов композиции веб/ В.В. Шкарупило, Р.К. Кудерметов/ Електронний ресурс].– Режим доступу: http://ea.dgtu.donetsk.ua:8080/jspui/bitstream/123456789/12648/1/%D0%A8%D0%BA%D0%B0%D1%80%D1%83%D0%BF%D0%B8%D0%BB%D0%BE%20%D0%92.%D0%92,%20%D0%9A%D1%83%D0%BB%D0%B5%D1%80%D0%BC%D0%B5%D1%82%D0%BE%D0%B2%20%D0%A0.%D0%9A..pdf. – Дата доступу: 09.03.2015

28. Foster, H. Using a Rigorous Approach for Engineering веб Service Compositions: A Case Study [text] / H. Foster, S. Uchitel, J. Magee, J. Kramer, M. Hu // Proceedings of the IEEE International Conference on Cervice Computing (Orlando, Florida, USA, July 11 – 15, 2005). – P. 217 – 224.

29.Bin, X. Efficient Composition of Semantic Web Services with End-to-End QoS Optimization [text] / X. Bin, L. Sen, Y. Yixin // Tsinghua Science and Technology. – 2010. – Vol. 15, No. 6. – P. 678 – 686.

30.Cristia, M. A TLA+ Encoding of DEVS Models [text] / M. Cristia // Proceedings of International Modeling and Simulation Multiconference (Buenos Aires, Argentina, February 8 – 12, 2007). – P. 17 – 22.

31. PROGRAMMABLEWEB RESEARCH CENTER: [Електронний ресурс].– Режим доступу: http://www.programmableweb.com/api-research. – Дата доступу: 15.02.2015

32. Wolfram|Alpha Webservice API Reference: [Електронний ресурс].– Режим доступу: http://products.wolframalpha.com/api/documentation.html. – Дата доступу: 24.03.2015

33. Charts | Google Developers: [Електронний ресурс].– Режим доступу: https://developers.google.com/chart/?hl=ru. – Дата доступу: 23.04.2015

34.Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98 (затверджено Постановою Головного державного санітарного лікаря України від 10.12.1998 р. № 7).

35.Правила охорони праці під час експлуатації електронно-обчислювальних машин. НПАОП 0.00-1.28-10

36.Санітарні норми мікроклімату виробничих приміщень : ДСН 3.3.6.042-99– К. : МОЗ України, 2000. – 42 с. – (Національні стандарти України).

37.Санітарні норми виробничого шуму, ультразвуку та інфразвуку : ДСН 3.3.6.037-99.– К. : МОЗ України, 2000. – 37 с. – (Національні стандарти України).

38.Природнє і штучне освітлення : ДБН В.2.5-28:2006 — К. : Міністерство будівництва, архітектури та житлово-комунального господарства України, 2006. – 68 с.

39.Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою. НАПБ Б.03.002-2007. (затверджено наказом МНС України від 03.12.2007 № 833)

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Файл WolframAlphaEngine.php. Цей класс містить всі необхідні фукції для роботи з API.

```
<?php

include_once 'include/WAResponse.php';
include_once 'include/WAPod.php';
include_once 'include/WASubpod.php';
include_once 'include/WAImage.php';
include_once 'include/WASubstitution.php';
include_once 'include/WAInfo.php';
include_once 'include/WAAssumption.php';

/**
*  This is the Wolfram Alpha API PHP Wrapper.
*  This object will handle all requests and responses to the
*  Wolfram Alpha API.
*  @package WolframAlpha
*/
class WolframAlphaEngine{
  private $APIURL = 'http://api.wolframalpha.com/v1/query.jsp'; // REPLACE THIS WITH FINAL URL WHEN
PUBLIC
  private $appID = '';


  public function WolframAlphaEngine( $appID ) {
    $this->appID = $appID;
  }

  /**
   *  Contact the WolframAlpha API and return results in a PHP data structure.
   *
   *  @param string $input string contains the input query to send to API.
   *
   *  @param array $otherParams    An optional hash array of key value pairs of other parameters
   *                               to pass to the API.
   *
```

```
   *  @return mixed          Returns the results of the query in an OO data structure.
   *                         Returns NULL if no app id or input is specified.
   *                         Refer to the manual for detailed return values.
   */
  public function getResults( $input, $otherParams=array() ) {
    // if no input or appid has been specified, return null
    if ( !$input || !$this->appID )
      return null;

    // get the API URL
    $url = $this->constructURL( $input, $otherParams );

    // Get URL contents and parse XML
    $xml = simplexml_load_file( $url );
    return $this->cleanResponseTree( $xml );
  }
  // PRIVATE FUNCTIONS
  /**
   *  Constructs the API url to be used in this request.
   *
   *  @param string $input      string contains the input query to send to API.
   *
   *  @param array $otherParams An optional hash array of key value pairs of other parameters
   *                            to pass to the API.
   *
   *  @return string            Returns the string URL to be used.
   */
  private function constructURL ( $input, $otherParams=array() ) {
    // construct the API URL
    $url = $this->APIURL ."?appid=". urlencode( $this->appID ) ."&input=".
urlencode($input)."&podtitle=Input&podtitle=Lenght+of+data&podtitle=Total&podtitle=Statisctics&podtimeout=10.0
&scantimeout=10.0&formattimeout=10.0&parsetimeout=10.0";
    foreach ( $otherParams as $key => $value ) {
      $url .= "&". urlencode( $key ) ."=". urlencode( $value );
    }
    return $url;
  }
  /**
   *  This function will take as input a SimpleXMLElement tree and return
   *  a cleanly formatted OO representation of a Wolfram Alpha response.
   *
   *  @param SimpleXMLElement $xml      The XML Tree obtained from an API call
```

```
 *
 *  @return mixed                       A formatted response object.
 */
private function cleanResponseTree( $xml ) {
 $response = new WAResponse();
 // set raw xml if user needs it at all
 $response->rawXML = $xml->asXML();
 // set the global document attributes
 $response->attributes = $this->parseAttributes( $xml );
 // check if the API responded with an error
 if ( $response->isError() ) {
  $response->error = $xml->error;
  return $response;
 }
 foreach ( $xml->pod as $rawpod ) {
  $pod = new WAPod();
  $pod->attributes = $this->parseAttributes( $rawpod );
    // handle markup tag if present
  if ( $rawpod->markup ) {
   $pod->markup = (string) $rawpod->markup;
  }
  // handle subpod tags if present
  if ( $rawpod->subpod ) {
   foreach ( $rawpod->subpod as $rawsubpod ) {
    $pod->addSubpod( $this->saveSubPod( $rawsubpod ) );
   }
  }
  // handle substitutions tag if present
  if ( $rawpod->substitutions ) {
   foreach ( $rawpod->substitutions->substitution as $rawsub ) {
    $sub = new WASubstitution();
    $att = $this->parseAttributes( $rawsub );
    $sub->name = $att['name'];

    $pod->addSubstitution( $sub );
   }
  }
  if ( $rawpod->infos ) {
   foreach ( $rawpod->infos->info as $rawinfo ) {
    $info = new WAInfo();
    $info->text = (string) $rawinfo->asXML();
```

```
      $pod->addInfo( $info );
     }
    }

    $response->addPod( $pod );
   }

   // if this response has a scripts section then add it
   if ( $xml->scripts ) {
    $response->script = (string) $xml->scripts;
   }
   // if this response has a css section then add it
   if ( $xml->css ) {
    $response->css = (string) $xml->css;
   }

   // if this response has any assumptions then loop throught them
   if ( $xml->assumptions ) {
    foreach ( $xml->assumptions->assumption as $rawassumption ) {
     $att = $this->parseAttributes( $rawassumption );
     foreach ( $rawassumption->value as $value ) {
      $valAtt = $this->parseAttributes( $value );
      $assumption = new WAAssumption();
      if (isset($att['type'])) $assumption->type = $att['type'];
      if (isset($att['word'])) $assumption->word = $att['word'];
      if (isset($valAtt['name'])) $assumption->name = $valAtt['name'];
      if (isset($valAtt['desc'])) $assumption->description = $valAtt['desc'];
      if (isset($valAtt['input'])) $assumption->input = $valAtt['input'];

      $response->addAssumption( $assumption );
     }
    }
   }

   return $response;
  }

  /**
   *  Parses a SimpleXMLElement subpod into a WASubpod object
   *  @param SimpleXMLElement $rawsubpod         the xml tree of a subpod
   *  @return WASubpod  a WASubpod representation of the input param
   */
```

```
  private function saveSubPod( $rawsubpod ) {
    $subpod = new WASubpod();
    $subpod->attributes = $this->parseAttributes( $rawsubpod );
    $subpod->plaintext = (string) $rawsubpod->plaintext;
    if ( $rawsubpod->img ) {
      $subpod->image = new WAImage();
      $subpod->image->attributes = $this->parseAttributes( $rawsubpod->img );
    }
    $subpod->minput = (string) $rawsubpod->minput;
    $subpod->moutput = (string) $rawsubpod->moutput;
    if ( $rawsubpod->mathml ) {
      $subpod->mathml = $rawsubpod->mathml->asXML();
    }

    return $subpod;
  }

  /**
   *  Will parse out the attributes of a SImpleXMLElement into an array
   *  @param SimpleXMLElement $attributes          the attributes element
   * @return array()          key, value pairs of the attributes
   */
  private function parseAttributes( $attributes ) {
    $ret = array();
    foreach ( $attributes->attributes() as $key => $val ) {
      $ret[$key] = (string) $val;
    }
    return $ret;
  }
}

?>
```

Файл WASssumption.php. Містить результати допущень, що зроблені з запиту до API WolframAlpha.

```
<?php

/**
 *  The Wolfram Alpha Assumption Object
 *  @package WolframAlpha
 */
```

```
class WAAssumption {
  // define the sections of a response
  public $type = '';
  public $word = '';
  public $name = '';
  public $description = '';
  public $input = '';

  // Constructor
  public function WAAssumption () {
  }

}
?>
```

Файл WAImage.php. Контейнер для зображень, що зроблені з запиту до API WolframAlpha.

```
<?php

/**
 *  The Wolfram Alpha Image Object
 *  @package WolframAlpha
 */
class WAImage {
  // define the sections of a response
  public $attributes = array();

  // Constructor
  public function WAImage () {
  }

}
?>
```

Файл WAInfo.php. Контейнер для інформації, що обробляється з запиту до API WolframAlpha.

```
<?php

/**
 *  The Wolfram Alpha Info Object
 *  @package WolframAlpha
```

```
 */
class WAInfo {
  // define the sections of a response
  public $text = '';

  // Constructor
  public function WAInfo () {
  }

}
?>
```

Файл WAPod.php. Контейнер для інформації про результат роботи запиту, за допомогою якого модна вивести більше інформації.

```
<?php

/**
 *  The Wolfram Alpha Pod Object
 *  @package WolframAlpha
 */
class WAPod {
  // define the sections of a response
  public $attributes = array();
  public $markup = '';

  // private accessors
  private $subpods = array();
  private $substitutions = array();
  private $infos = array();

  // Constructor
  public function WAPod () {
  }

  /**
   *  Add a subpod to this pod
   *  @param WASubpod $subpod  the subpod to be added
   */
  public function addSubpod( $subpod ) {
    $this->subpods[] = $subpod;
  }
```

```
 /**
  *  Add a substitution to this pod
  *  @param WASubstitution $sub   the substitution to be added
  */
 public function addSubstitution( $sub ) {
   $this->substitutions[] = $sub;
 }

 /**
  *  Add an info to this pod
  *  @param WAInfo $info   the info to be added
  */
 public function addInfo( $info ) {
   $this->infos[] = $info;
 }

 /**
  *  Get the subpods associated with this pod
  *  @return array( WASubpod )    An array of subpods
  */
 public function getSubpods() {
   return $this->subpods;
 }

 /**
  *  Get the substitutions associated with this pod
  *  @return array( WASubstitution )        An array of substitutions
  */
 public function getSubstitutions() {
   return $this->substitutions;
 }

 /**
  *  Get the infos associated with this pod
  *  @return array( WAInfo )        An array of infos
  */
 public function getInfos() {
   return $this->infos;
 }
}
?>
```

Файл WAResponse.php. WAResponse – це класс, який сортує отриману інформацію.

```
<?php

/**
 *  The Wolfram Alpha Reponse Object
 *  @package WolframAlpha
 */
class WAResponse {
  // define the sections of a response
  public $attributes = array();
  public $error = array();
  public $rawXML = '';
  public $script = '';
  public $css = '';

  // private accessors
  private $pods = array();
  private $assumptions = array();

  // Constructor
  public function WAResponse () {
  }

  public function isError() {
    if ( $this->attributes['error'] == 'true' ) {
      return true;
    }
    return false;
  }

  /**
   *  Add a pod to this response object
   *  @param WAPod $pod            A WAPod object to be added
   */
  public function addPod( $pod ) {
    $this->pods[] = $pod;
  }

  /**
```

```
   *  Add an assumption to this response object
   *  @param WAAssumption $assumption A WAAssumption object to be added
   */
  public function addAssumption( $assumption ) {
    if ( !isset( $this->assumptions[$assumption->type] ) ) {
      $this->assumptions[$assumption->type] = array();
    }

    $this->assumptions[$assumption->type][] = $assumption;
  }

  /**
   *  Get the pods associated with this response
   *  @return array( WAPod )        An array of pods
   */
  public function getPods() {
    return $this->pods;
  }

  /**
   *  Get the assumptions associated with this response
   *  @return array( WAAssumption )        An array of assumptions
   */
  public function getAssumptions() {
    return $this->assumptions;
  }
}
?>
```

WASubpod.php містить класс, з якого формуються елементи відповідей.

```
<?php

/**
 *  The Wolfram Alpha Subpod Object
 *  @package WolframAlpha
 */
class WASubpod {
  // define the sections of a response
  public $attributes = array();
  public $plaintext = '';
  public $image = '';
  public $minput = '';
```

```
  public $moutput = '';
  public $mathml = '';

  // Constructor
  public function WASubpod () {
  }

}
?>
```

WASubstiotution.php містить класс-контейнер для інформації, що генерує та використовує WolframAlpha API, якщо вхідний набір даних не є вірним.

```
<?php

/**
 *  The Wolfram Alpha Substitution Object
 *  @package WolframAlpha
 */
class WASubstitution {
  // define the sections of a response
  public $name = '';

  // Constructor
  public function WASubstitution () {
  }

}
```

Файл Start.php отримує вхідні дані від клієнта та генерує сторінку результатів, є повязуючим інтерфейсом для компонування веб-сервісів.

```
<?php
  header("Content-type: text/html; charset=utf-8");
  include '../wa_wrapper/WolframAlphaEngine.php';
?>
<!doctype html>
<html>
<head>
 <!--Load the AJAX API-->
    <script type="text/javascript" src="https://www.google.com/jsapi"></script>

  <meta http-equiv=content-type content="text/html; charset=utf-8">
  <title>Mashup web-application</title>
```

```
  <link rel="stylesheet" href="styles.css" type="text/css">
    <link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Oswald:400,300" type="text/css">
</head>


<body>
<div id="wrapper">
<div id="heading">
    <h1>Mashup web service for sample analyse</h1>
</div>
<header>
<form method='POST' action='#'>
Enter your sample:
<input type="text" name="q" value="
<?php
  $queryIsSet = isset($_REQUEST['q']);
  if ($queryIsSet) {
    echo $_REQUEST['q'];
  };
?>"
>   <button type="submit">Analyse</button>
</form>
</header>
<br><br>
<hr>
<?php
#<input type="submit" name="Search" value="Analyse">
 # $appID = 'PLACE_YOUR_APP_ID_HERE';
  $appID = 'Y95RH5-VL2PXWAW96';

  if (!$queryIsSet) die();
  $qMas = explode (", ", $_REQUEST['q']);
/*          for ($i=0; $i<count($qMas)-1;$i++)

                                {
                                echo "[",".$qMas[$i]."],";
                        }
                                echo "[",".$qMas[count($qMas)-1]."]";*/
  $qArgs = array();
  if (isset($_REQUEST['assumption']))
    $qArgs['assumption'] = $_REQUEST['assumption'];
```

```
 // instantiate an engine object with your app id
 $engine = new WolframAlphaEngine( $appID );

 // we will construct a basic query to the api with the input 'pi'
 // only the bare minimum will be used
 $response = $engine->getResults( $_REQUEST['q'], $qArgs);

 // getResults will send back a WAResponse object
 // this object has a parsed version of the wolfram alpha response
 // as well as the raw xml ($response->rawXML)

 // we can check if there was an error from the response object
 if ( $response->isError() ) {
?>
 <h1>There was an error in the request</h1>
 </body>
 </html>
<?php
   die();
 }
?>

<h1>Results</h1>
<br>

<?php
 // if there are any assumptions, display them
 if ( count($response->getAssumptions()) > 0 ) {
?>
   <h2>Assumptions:</h2>
   <ul>
<?php
     // assumptions come as a hash of type as key and array of assumptions as value
     foreach ( $response->getAssumptions() as $type => $assumptions ) {
?>
       <li><?php echo $type; ?>:<br>
         <ol>
<?php
         foreach ( $assumptions as $assumption ) {
?>
```

```
          <li><?php echo $assumption->name ." - ". $assumption->description;?>, to change search to this assumption <a href="simpleRequest.php?q=<?php echo urlencode($_REQUEST['q']);?>&assumption=<?php echo $assumption->input;?>">click here</a></li>
<?php
        }
?>
        </ol>
      </li>
<?php
    }
?>

  </ul>
<?php
 }
?>

<hr>

<?php
 // if there are any pods, display them
 if ( count($response->getPods()) > 0 ) {
?>

  <table border=0px width="80%" align="center">
<?php
  foreach ( $response->getPods() as $pod ) {
?>
    <tr>
      <td>
        <h3><?php echo $pod->attributes['title']; ?></h3>
<?php
      // each pod can contain multiple sub pods but must have at least one
      foreach ( $pod->getSubpods() as $subpod ) {
        // if format is an image, the subpod will contain a WAImage object
?>
        <img src="<?php echo $subpod->image->attributes['src']; ?>">
        <hr>
<?php
      }
?>
```

```
      </td>
    </tr>
<?php
   }
?>
   </table>
<?php
  }
?>
   <script type="text/javascript">

    // Load the Visualization API and the piechart package.
    google.load('visualization', '1.0', {'packages':['corechart']});

    // Set a callback to run when the Google Visualization API is loaded.
    google.setOnLoadCallback(drawChart);

    // Callback that creates and populates a data table,
    // instantiates the pie chart, passes in the data and
    // draws it.
    function drawChart() {

    // Create the data table.

    var data = google.visualization.arrayToDataTable([
          ["",""],
          <?php

          for ($i=0; $i<count($qMas)-1;$i++)

                              {
                              echo "["."$qMas[$i]."],";
                    }
                              echo "["."$qMas[count($qMas)-1]."]";

          ?>
 ]);

    // Set chart options
    var options = {'title':'Histogram', legend: { position: 'none' },
                colors: ['blue'], 'width':600,
                 'height':450 };
```

```
    // Instantiate and draw our chart, passing in some options.
    var chart = new google.visualization.ColumnChart(document.getElementById('chart_div'));
    chart.draw(data, options);
  }
  </script>
<!--Div that will hold the pie chart-->
  <div id="chart_div" style="width:400; height:300"></div>
  <div>
  <footer>
  <div id="footer">

  </div>
</footer>
</body>
</html>
```