

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повна назва інституту/факультету)

Кафедра автоматика та управління в технічних системах

(повна назва кафедри)

«На правах рукопису»

УДК _____

«До захисту допущено»

Завідувач кафедри

_____ **О. І. Ролік**
(підпис) (ініціали, прізвище)

“ ____ ” _____ 2018 р.

Магістерська дисертація

зі спеціальності (спеціалізації) 126 «Інформаційні системи та технології»
(код і назва спеціальності)

на тему: Програмні засоби збереження стану об'єктів

Виконав: студент 6 курсу, групи ІА-73мп
(шифр групи)

Ульянич Демид Вячеславович

(прізвище, ім'я, по батькові)

_____ (підпис)

Науковий керівник доцент, к.тех.н. Дорогий Я.Ю.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Консультант _____

(назва розділу)

_____ (науковий ступінь, вчене звання, прізвище, ініціали)

_____ (підпис)

Рецензент _____

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

_____ (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2018 року

**Національний технічний університет України
“Київський політехнічний інститут
імені Ігоря Сікорського”**

Факультет інформатики та обчислювальної техніки
(повна назва)

Кафедра автоматики та управління в технічних системах
(повна назва)

Ступінь вищої освіти – другий (магістерський)
(код, назва)

Спеціальність 126 «Інформаційні системи та технології»
(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О. І. РОЛІК
(підпис) (ініціали, прізвище)

“ ___ ” _____ 2018_р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Ульяничу Демиду Вячеславовичу

(прізвище, ім'я, по батькові)

1. Тема дисертації Програмні засоби збереження стану об'єктів

Науковий керівник дисертації доцент, к.тех.н. Дорогий Я.Ю.

затверджені наказом по університету від “ 29 ” жовтня 2018 р. № _____

2. Строк подання студентом дисертації “ 4 ” грудня 2018 р.

3. Об'єкт дослідження: Програмні моделі та методи їх збереження

4. Зміст пояснювальної записки: а) огляд існуючих рішень і технологій; б) розробка програмного застосунку збереження стану об'єктів; в) дослідження та оптимізація методів серіалізації; г) розробка стартапу;

5. Перелік графічного (ілюстративного) матеріалу: Діаграма класів; Діаграма застосувань; Діаграма компонентів; Результат досліджень 1; Результат досліджень 2.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання “29” жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Огляд і аналіз існуючих рішень	02.10.2018	
2	Вивчення об'єкту дослідження	15.10.2018	
3	Розробка математичних моделей	29.10.2018	
4	Моделювання і дослідження методів серіалізації	10.11.2018	
5	Розробка програмної моделі	28.11.2018	
6	Оформлення текстової та графічної документації	2.12.2018	
7	Представлення до захисту	4.12.2018	

Студент

(підпис)

Д.В. Ульнич

(ініціали, прізвище)

Керівник проекту

(підпис)

Я.Ю. Дорогий

(ініціали, прізвище)

АНОТАЦІЯ

Магістерська дисертація освітньо-кваліфікаційного рівня “магістр” на тему «Програмні засоби збереження стану інформаційних об’єктів». 117 с., 31рис., 23 таб., 2 додатки, 17 джерел.

Об’єктом досліджень даної дисертації є передача даних між вузлами системи заснованої на клієнт-серверній архітектурі.

Предметом досліджень є певна виділена цілісність об’єкту, яка має найбільш істотні з точки зору мети даного дослідження ознаки або властивості. Тобто, в даному контексті предметом дослідження магістерської дисертації є алгоритми збереження інформаційних об’єктів побудованих з використанням серіалізації даних.

Метою даної магістерської дисертації являється створення програмного засобу збереження стану інформаційних об’єктів за допомогою різних форматів серіалізації даних, заснованому на дослідженні існуючих рішень та технологій, що наразі є самими популярними при розробці клієнт-серверної архітектури. Вибір того чи іншого формату серіалізації даних повинен ґрунтуватись на їх дослідженні.

Методи дослідження – метод абстрагування, метод індукції та дедукції, метод порівняння, метод моделювання, аналіз та синтез. Апаратура, яка використовувалась у дослідженні – персональний комп’ютер, зовнішні пристрої.

Практична цінність магістерської дисертації полягає у можливості застосування результатів дослідження різних форматів серіалізації даних для збереження різних даних у реальних проектах із клієнт-серверною архітектурою чи застосунках, у обміні даними між вузлами в яких може застосовуватися

бінарна серіалізація. Також практичною цінністю є результати порівняння існуючих широковідомих методів серіалізації.

СЕРІАЛІЗАЦІЯ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ІНФОРМАЦІЙНИЙ СТАН, ТУРИЗМ, МЕТОДИ СЕРІАЛІЗАЦІЇ

ABSTRACT

Master's dissertation of educational qualification level "Master" on the theme "Software tools for preservation of the status for information objects". 117 pp., 31 figures, 23 tables, 2 supplement, 17 sources.

The object of the research of this dissertation is the transfer of data between the nodes of the system based on client-server architecture.

The subject of research is a certain highlighted integrity of an object that has the most significant in terms of the purpose of this study of a sign or property. That is, in this context, the subject of the study of a master's thesis is the algorithms for storing information objects constructed using serialization of data.

The purpose of this master's thesis is to create a software tool for preserving the status of information objects using various data serialization formats, based on the study of existing solutions and technologies, which are currently the most popular in developing client-server architecture. The choice of this or that format of data serialization should be based on their research.

Research methods – abstraction method, method of induction and deduction, method of comparison, method of modeling, analysis and synthesis. Equipment used in the study – personal computer, external devices.

The practical value of a master's thesis is the ability to apply the results of research to different data serialization formats to store various data in real projects with client-server architecture or applications, in the exchange of data between nodes in which binary serialization can be applied. Also practical value is the comparison of existing well-known serialization methods.

SERIALIZATION, CLIENT-SERVER ARCHITECTURE, INFORMATION STATUS, TOURISM, METHODS OF SERIALIZATION

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

SQL (англ. Structured query language— мова структурованих запитів) — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних.

JSON (англ. JavaScript Object Notation, укр. об'єктний запис JavaScript, вимовляється джейсон) — це текстовий формат обміну даними між комп'ютерами.

XML – Розширювана мова розмітки (англ. Extensible Markup Language, скорочено XML) — запропонований консорціумом World Wide Web (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет.

HTTP – протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпертекстових документів

FTP – Протокол передачі файлів (англ. File Transfer Protocol, FTP) — дає можливість абоненту обмінюватися двійковими і текстовими файлами з будь-яким комп'ютером мережі, що підтримує протокол FTP

TCP/IP – це аббревіатура терміну Transmission Control Protocol / Internet Protocol (Протокол керування передачею / міжмережевий протокол). OSI

SNA – Systems Network Architecture (системна мережева архітектура) — розроблена компанією IBM в 1974 г., загальний опис структури, форматів, протоколів, що використовуються для передачі інформації між програмами IBM

та обладнанням GUI

DB – База даних — впорядкований набір логічно взаємопов'язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів

СУБД – Система управління базами даних (СУБД) — комп'ютерна програма чи комплекс програм, що забезпечує користувачам можливість створення, збереження, оновлення, пошук інформації та контролю доступу в базах даних.

ЛКМ – Локальна комп'ютерна мережа (англ. Local Area Network (LAN)) являє собою об'єднання певного числа комп'ютерів на відносно невеликій території

ОС – Операційна система, скорочено ОС (англ. operating system, OS) — це базовий комплекс програмного забезпечення, що виконує управління апаратним забезпеченням комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організовує взаємодію з користувачем.

SOAP – протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML.

RPC – Виклик віддалених процедур (англ. Remote procedure call, RPC) — протокол, що дозволяє програмі, запущеній на одному комп'ютері бути викликаною на іншому комп'ютері без написання безпосередньо коду для цієї операції.

API – Прикладний програмний інтерфейс (англ. Application Programming Interface, API) — набір визначень взаємодії різнотипного програмного забезпечення

ЕОМ – Електронна обчислювальна машина (скорочено ЕОМ) — загальна назва для обчислювальних машин, що є електронними (починаючи з перших лампових машин, включаючи напівпровідникові тощо) на відміну від електромеханічних (на електричних реле тощо) та механічних обчислювальних машин

XSD – мова схем XML документів, опублікована в травні 2001

консорціумом W3C як «рекомендація» (англ. Recommendation).

WSDL – (англ. Web Services Description Language) — мова опису зовнішніх інтерфейсів веб-служби.

UDDI – (англ. Universal Description Discovery & Integration) — платформо-незалежний інструмент для розміщення описів веб-сервісів (WSDL) для забезпечення можливості їх пошуку іншими організаціями і інтеграції в свої системи.

RESTful API чи REST – (скор. англ. Representational State Transfer, «передача стану подання») — підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів

HTTPS – схема URI, що синтаксично ідентична http: схемі, яка зазвичай використовується для доступу до ресурсів Інтернет.

COM – (Component Object Model) — платформа компонентно-орієнтованого програмування розроблена в 1993 році компанією Microsoft

CORBA (англ. Common Object Request Broker Architecture — загальна архітектура брокера об'єктних запитів) — це запропонований консорціумом OMG технологічний стандарт розробки розподілених застосунків.

CPAN (аббревіатура від англ. Comprehensive Perl Archive Network — «всеосяжна мережа архівів Perl») – архів документації та програмного забезпечення, написаного на мові програмування Perl.

CSV (від англ. comma-separated values ‘значення, розділені комою’, іноді character-separated values ‘значення, розділені символом’) – файловий формат, котрий є відмежовувальним форматом для представлення табличних даних, у якому поля відокремлюються символом коми та переходу на новий рядок.

ASN.1 (англ. Abstract Syntax Notation One) – в області телекомунікацій і комп'ютерних мереж мова для опису абстрактного синтаксису даних, що використовує OSI.

Консорціум Всесвітньої павутини (англ. World Wide Web Consortium, W3C) – головна міжнародна організація, що розробляє й впроваджує технологічні стандарти для всесвітньої павутини.

Ubuntu – операційна система для робочих станцій, лептопів і серверів, найпопулярніший у світі дистрибутив Linux

UTF-8 (від англ. Unicode Transformation Format — формат перетворення Юнікоду) – кодування, що реалізує представлення Юнікоду, сумісне з 8-бітовим кодуванням тексту.

Oracle – об'єктно-реляційна система керування базами даних від Oracle Corporation.

ASP.NET/.NET – технологія створення веб-застосунків і веб-сервісів від компанії Майкрософт.

UML – (англ. Unified Modeling Language) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування

Арифметико-логічний пристрій (АЛП) (англ. Arithmetic Logic Unit, ALU) – блок процесора, що служить для виконання арифметичних та логічних перетворень над даними, що іменуються операндами

ОЗУ – Комп'ютерна пам'ять (англ. memory, storage) — функціональна частина ЕОМ, призначена для прийому, зберігання та видачі даних.

Unix – це операційна система, яка спочатку розроблялася впродовж 1969-1970-х років групою співробітників підрозділу Bell Labs корпорації AT&T, яка включала Кена Томпсона, Денніса Рітчі та Дугласа Макілроя.

JVM – Віртуальна машина Java (англ. Java Virtual Machine; JVM) — набір комп'ютерних програм та структур даних, що використовують модель віртуальної машини для виконання інших комп'ютерних програм чи скриптів.

Just-in-time compilation (JIT) (також відома, як dynamic translation) — компіляція «на льоту» – це технологія збільшення продуктивності програмних систем, що використовують байт-код, шляхом трансляції байт-коду в машинний код безпосередньо під час роботи програми.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ	5
ВСТУП.....	12
1 ПОСТАНОВКА ЗАДАЧІ.....	16
2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ.....	22
2.1 Особливості клієнт-серверної архітектури.....	22
2.2 Застосунки з клієнт-серверною архітектурою	25
2.3 Бази даних	27
2.4 Клієнт-серверні технології.....	29
2.4.1 Дволанкова клієнт-серверна архітектура.....	32
2.4.2 Триланкова архітектура клієнт-сервер.....	35
2.5 Передача повідомлень	36
2.6 Веб-сервер та веб-сервіс	39
2.7 Серіалізація даних.....	43
2.8 Формати серіалізації	51
2.8.1 Бінарні формати.....	51
2.8.2 Текстові формати.....	56
3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ ЗБЕРЕЖЕННЯ СТАНУ ОБ'ЄКТІВ	66
3.1 Попереднє проектування системи.....	67
3.1.1 Розробка архітектури та структури програми.....	67
3.1.2 Модуль взаємодії з зовнішніми пристроями	68
3.1.3 Модуль обробки та збереження даних.....	69

3.1.4 Модуль взаємодії з користувачем	70
3.2 Вибір мови програмування	71
3.3 Вибір веб-фрейморку	75
3.4 Вибір БД та СУБД.....	77
3.5 Вибір інструментарію створення інтерфейсу користувача	80
3.6 Вихідні дані.....	81
3.6.1 Формування тестових даних	81
3.6.2 Синтаксис та простота реалізації.....	81
3.7 Розробка діаграми варіантів використання	81
3.8 Розробка діаграми компонентів.....	83
3.9 Розробка діаграми класів.....	84
3.10 Розробка моделі даних.....	85
3.11 Розробка основних модулів застосування	87
3.11.1 Модуль взаємодії з зовнішніми пристроями	87
3.11.2 Модуль обробки та збереження даних.....	90
3.11.3 Модуль взаємодії з користувачем	92
3.11.4 Інтерфейс користувача.....	92
3.12 Тестування програмного засобу	94
4 ДОСЛІДЖЕННЯ ТА ОПТИМІЗАЦІЯ МЕТОДІВ СЕРІАЛІЗАЦІЇ.....	95
4.1 Порівняльна характеристика форматів серіалізації	95
4.1.1 Швидкість обробки даних	95
4.1.2 Використання оперативної пам'яті	98
4.1.3 Розмір серіалізованих даних	101
4.2 Оптимізація методів серіалізації	102
5 РОЗРОБКА СТАРТАПУ.....	105

.....	5.1	Опис ідеї проекту	
.....			105
5.3.....			
Аналіз ринкових можливостей запуску стартап-проекту.....			106
5.4.....			
Розроблення ринкової стратегії проекту			109
5.5.....			
Розроблення маркетингової програми стартап-проекту.....			111
ВИСНОВКИ.....			2
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ			4
ДОДАТОК Б.....			8

ВСТУП

З часів появи клієнт-серверної технології у світі обчислювальної техніки у 80-х роках [1], фахівці не зупиняли розвиток та удосконалення застосунків з клієнт-серверною архітектурою. В результаті постійних досліджень та інвестувань, з'явилися застосунки, що підтримують спільну роботу безлічі користувачів за допомогою лише єдиного джерела даних у мережі. Подібний прорив перетворив архітектуру клієнт-сервер на загальнопоширену при спілкуванні з комп'ютером або з системою на його основі. Наразі клієнт-серверну архітектуру використовують навіть при під'єднанні до діалогової системи за допомогою телефонного зв'язку чи при користуванні касовим апаратом, зчитуючи штрих-коди товарів на спеціалізованому пристрої магазину або розплачуючись за них за допомогою кредитних карток. Дана архітектура присутня майже у всіх оточуючих нас системах, з якими ми стикаємось кожного дня, тож вона має здійснювати складні алгоритми обробки запитів мільйонів користувачів за невеликі проміжки часу. Крім того, дані системи зобов'язані володіти високим рівнем захисту інформації, що підтримується за допомогою використання середовища програмування SQL Server, а також зберігання всіх даних та прикладних засобів централізовано, тобто, зосереджених в одному місці.

У навантаженому проекті з клієнт-серверною архітектурою обмін повідомленнями між компонентами системи нараховує 100 тисяч в день і більше [2]. Саме тому передача даних є найвужчим місцем у реалізації клієнт-серверної архітектури. Абстрагуючись від транспортного рівня передачі даних, а саме, використання протоколу даних, за допомогою якого передається інформація, залишиться не менш важлива і суттєва фаза передачі даних - їх формування. Для цього використовується так звана серіалізація. Цей процес відповідає в першу чергу за структуру даних, а також їх обсяг по завершенню серіалізації. Іноді цей процес може займати багато машинного часу, навантажуючи застосунок, а іноді замість стиснення даних, використовувати надлишкові записи, залежно від

методу серіалізації. До того ж, сучасний об'єм інформації, що передається по каналу зв'язку досягає сотень мегабайт, з чим системі необхідно справлятися без зайвих затримок та помилок. Зменшення часу обробки та передачі даних дозволить покращити сервіс обслуговування людей машинами, розширити функціонал систем, збільшити капітал компанії-розробника, а також зменшити навантаження на сервери та клієнти.

Тож, беручи до уваги безсумнівне лідерство клієнт-серверної архітектури та важливість етапу серіалізації при обміні повідомленнями, оптимізація та ретельне дослідження форматів серіалізації являються актуальними фазами для будь-якого проекту. Вибір правильного формату серіалізації, що буде найкраще підходити для конкретної розроблюваної системи, а також виокремлення основних характеристик, на які слід покладатися при порівнянні форматів серіалізації, дозволить застосункам працювати ефективніше, надмірно не навантажуючи архітектуру і структуру даних.

Метою даної магістерської дисертації являється створення програмного засобу збереження стану інформаційних об'єктів за допомогою різних форматів серіалізації даних, заснованому на дослідженні існуючих рішень та технологій, що наразі є самими популярними при розробці клієнт-серверної архітектури. Вибір того чи іншого формату серіалізації даних повинен ґрунтуватись на їх дослідженні.

Для створення нового чи оптимізації існуючих форматів серіалізації, спочатку необхідно виконати етап дослідження існуючих технологій, знайти їх сильні та слабкі сторони, а також, зробити висновки, коли і де краще застосовувати окремі формати. Так як процес оптимізації потребує виділення характеристик, що необхідно оптимізувати, до задачі оптимізації входить також виокремлення найвужчих місць процесу серіалізації, що найбільше впливають на ефективність роботи застосунків із клієнт-серверною архітектурою, а також дослідження цих показників для окремих технологій. Перед виконанням самого етапу дослідження необхідно створити тестові дані, що матимуть якомога різноманітнішу структуру, яка дозволить прослідкувати за поведінкою різних

форматів при різних умовах та типах даних. Обсяг оброблюваних даних також має варіюватися у великому діапазоні із невеликим кроком дискретизації, тож до переліку задач входить генерація тестових даних таких, що б задовольняли усі характеристики та не змінювались із часом для отримання даних спостережень без урахування впливу зовнішніх факторів. Не менш важливою задачею є вибір мови, якою буде імплементуватися метод серіалізації.

Об'єкт досліджень у методології досліджень – це будь-яка частина або форма існування реальної дійсності, для вивчення яких суб'єкт (людина) змушена вдаватися до цілеспрямованих науково-пізнавальних дій [3]. Об'єктом досліджень даної дисертації є передача даних між вузлами системи заснованої на клієнт-серверній архітектурі.

Предметом досліджень є певна виділена цілісність об'єкту, яка має найбільш істотні з точки зору мети даного дослідження ознаки або властивості [3]. Тобто, в даному контексті предметом дослідження магістерської дисертації є алгоритми збереження інформаційних об'єктів побудованих з використанням серіалізації даних.

Згідно до [3], серед існуючих методів дослідження (методів наукового пізнання) виділяють загальні та спеціальні. Спеціальні методи мають специфічний характер та визначаються характером досліджуваного об'єкта. Загальні методи використовуються в дослідницьких процесах в різноманітних науках та поділяються на методи емпіричного дослідження, методи, що використовуються на теоретичному рівні дослідження та ті, що використовуються і на емпіричному, і на теоретичному рівнях. Емпіричні поділяються на спостереження, порівняння, вимірювання та експеримент. У даній роботі використовуються порівняння та експеримент. Експеримент полягає у подачі різних типів та обсягу даних на вхід форматів серіалізації і виявлення властивостей кожного з форматів залежно від умов роботи. Такий експеримент можна проводити декілька разів як для одних і тих самих даних, так і для даних з іншою складністю. Порівняння полягало у виявленні різниці отриманих даних у ході експерименту.

Практична цінність магістерської дисертації полягає у можливості застосування результатів дослідження різних форматів серіалізації даних для збереження різних даних у реальних проектах із клієнт-серверною архітектурою чи застосунках, у обміні даними між вузлами в яких може застосовуватися бінарна серіалізація. Також практичною цінністю є результати порівняння існуючих широковідомих методів серіалізації.

Науковою цінністю роботи є проміжні результати роботи та проведених досліджень, що були представлені та публікувались у наступних конференціях та опубліковані у відповідних збірниках:

- VII Міжнародної науково-практичної конференції «Фізико-технологічні проблеми передавання, оброблення та зберігання інформації в інфокомунікаційних системах (PREDT-2018)» (8-10 листопада 2018 року, м.Чернівці) [33].

1 ПОСТАНОВКА ЗАДАЧІ

Програмний засіб збереження інформаційних об'єктів призначений для обробки даних з навігаційних приладів та приладів фото-, відеофіксації та представлення їх у придатному для аналізу вигляді.

Будь-яка компанія, що володіє автопарком, зацікавлена в тому, щоб планувати оптимальні маршрути переміщення своїх автомобілів з метою планування та економії пального та інших ресурсів. Зазвичай GPS-навігатори, якими користуються водії, пропонують найкоротший маршрут, яких може виявитись найменш оптимальним через низьку якість доріг або високу завантаженість. Володіючи інформацією, зібраною за деякий час, можна робити прогнози та висновки щодо оптимального маршруту. Корисною також може бути інформація про час та місце зупинки водіїв.

Програмний засіб збереження інформаційних об'єктів може бути застосований при тренуваннях в таких видах спорту як біг, вело їзда, спортивний туризм, та в будь-яких інших, що пов'язані з переміщенням спортсмена на довгі дистанції. Також, такий засіб можна використовувати при побудові туристичних маршрутів. Маючи агреговані дані по пройдений дистанції (швидкості, зупинкам, набраній та втраченій висоті тощо), фото та відеоматеріали по туристичному маршруту можна робити висновки відносно ефективності прокладення туристичного маршруту та приймати міри для покращення майбутніх результатів. В туризмі і спортивних тренуваннях корисною є можливість ділитися власними маршрутами та переглядати маршрути інших користувачів системи.

Метою даної магістерської дисертації являється створення програмного засобу збереження стану інформаційних об'єктів за допомогою різних форматів серіалізації даних, заснованому на дослідженні існуючих рішень та технологій, що наразі є самими популярними при розробці клієнт-серверної архітектури. Вибір того чи іншого формату серіалізації даних повинен ґрунтуватись на їх дослідженні. Будь-який метод серіалізації повинен мати певні властивості.

По перше, дані мають бути представлені у форматі, який дозволяє опрацювати їх без втрати значень і точності на різних платформах. Якщо розглянути різні платформи і мови для розробки застосунків, ми побачимо наскільки сильно вони різняться. Є мови, такі як C++, що відносяться до сімейства скомпільованих, це значить, що вони залежні від типу процесора та системи на якій повинні виконуватися [4]. З іншого боку існує дуже великий клас мов програмування які виконуються в певному середовищі - одні з них - C#, Java. Також, частина цих мов інтерпретуються і код по суті являється повністю незалежний від платформи (крім ресурсів звичайно), але при цьому все одно виконується на основі специфічних процесорних команд.

Метод серіалізації повинен дозволяти представити дані в такому унікальному вигляді, який зможе бути легко використаний в подібних системах. Необхідно виділити ряд властивостей, які найбільш важливі для реалізації цих методів. Будь-якій із схем серіалізації властиве послідовне кодування даних за визначенням. Тож, отримання будь-якої частини серіалізованої структури даних вимагає, щоб опрацювання всього об'єкту від початку до кінця та його відтворення. Для багатьох застосунків така лінійність зручна, тому що вона дозволяє використовувати прості інтерфейси введення-виведення загального призначення для збереження і передачі стану об'єкта. Але, якщо для застосунку важлива висока продуктивність, може мати сенс використовувати складнішу, організацію обробки та зберігання даних.

Серіалізація також надає розробникам кілька корисних можливостей [5]:

- реалізація зберігання об'єктів, у більш зручному форматі, ніж послідовний запис їх властивостей у текстовий файл, що зберігається на диску, а після повторна збірка цих об'єктів через зчитування файлів із серіалізованими даними;
- метод виявлення змін даних, що змінюються з часом.

Для найбільш ефективного використання цих особливостей, необхідно забезпечувати незалежність від архітектури застосунку. Наприклад, мати можливість відтворити серіалізований потік даних, незалежно від порядку

байтів, що використовується в даній архітектурі. Це значить, що найпростіша та найшвидша процедура, що заключається у прямому копіюванні ділянки пам'яті із структурою даних, не може працювати надійно для всіх архітектур. Серіалізація структур даних в архітектурно-незалежному форматі має не допускати виникнення проблем через різний порядок проходження байт, відмінності представлення структур для різних мов програмування чи механізмів розподілу пам'яті для конкретної платформи.

Для будь-якої серіалізації важлива наявність залежності продуктивності від складності об'єкта, який може являти собою тип даних різних форматів. Наприклад, об'єкт String сам по собі вже несе дані в бітовому представленні і його перетворення далі не особливо має сенс, а от double вже має значно складнішу структуру, так як далеко не кожна мова дозволяє на пряму маніпулювати бітами double, перетворювати його в байти або зчитувати з байт. До складності об'єкта також можна віднести його розмір, кількість полів. Необхідно також враховувати структурну організацію об'єкту, а саме - різні зв'язки спадкування, адже об'єкт може бути успадкований від кількох інших об'єктів або інтерфейсів, і це повинно братися до уваги при серіалізації або контролюватися їх ідентичність під час зчитування об'єкта та його відтворення.

Дуже велику роль для деяких систем може грати можливість роботи з графами, особливо зацикленими, адже так чи інакше, серіалізація йде послідовно, і потрібно якимось чином (максимально ефективним) визначати, що вже серіалізувалось, а що ні, і при дуже великій структурі даних це може бути дуже витратною операцією. Також, важливе розуміння вкладених об'єктів, максимальної вкладеності і взагалі при серіалізації дуже складного об'єкта, який складається з незліченної кількості таких-же складних об'єктів, необхідно тримати результат в пам'яті до тих пір доки весь об'єкт не серіалізувався. При цьому, вихідний об'єкт може ще й динамічно довантажувати свої частини, що по суті призводить до неможливості спочатку вирахувати необхідну кількість цієї самої пам'яті.

У будь-якого методу серіалізації є багато переваг і недоліків. Саме вони стають наріжним каменем при виборі того чи іншого методу серіалізації. Дуже важливо скільки часу (процесорного часу) витрачається на серіалізацію, так як дані можуть бути і повинні бути (якщо система дійсно використовується на всі 100%) дуже великими. Це можуть бути тисячі і десятки тисяч об'єктів в секунду, і кожен з них повинен бути переведений в бітове представлення для подальшої передачі клієнту або серверу. Для кожної серіалізації витрачаються ресурси, машинний час. Також дуже важливо, щоб залежність за часом серіалізації від складності об'єкту була лінійною. Звичайно необхідно враховувати не тільки час, але й ресурси, одним з них може бути кількість використовуваних потоків.

Очевидно, що на даний момент процесори йдуть шляхом розвитку, при якому збільшення продуктивності досягається на основі створення нових і нових обчислювальних вузлів. При цьому у разі серверної системи кількість таких вузлів-ядер може доходити до сотні і більшої кількості. І на перший погляд може здатися що створення великої кількості потоків для серіалізації (наприклад, по одному на кожен об'єкт, або на рівень в ієрархії) хороший підхід, але на жаль це зовсім не так. У кожному разі перемикання між потоками вимагає час, і при досить великій їх кількості процесор буде зайнятий більшу частину часу саме перемиканнями а не обчисленнями як такими і абсолютна більшість буде очікувати і боротися за цей ресурс, що сильно знизить ефективність. Пам'ять, яка використовується під час серіалізації теж особлива характеристика, адже це самий обмежений ресурс в системі. Якщо недолік процесорного часу просто знижує ефективність і швидкість то недолік пам'яті може просто призвести до неможливості виконання тієї чи іншої операції. Як розглядалося раніше, складність об'єкта може бути кардинально різною, а отже використання пам'яті теж.

Слід розуміти, який з методів має яку характеристику для власне поліпшення цих самих характеристик. В результаті серіалізації виходить дзеркальне відображення об'єкта, яке несе в собі всю необхідну інформацію для його повторного відтворення. Це дуже важливий аспект, адже іноді ця

інформація зовсім не потрібна, або навпаки, вкрай важлива. Ця особливість алгоритму може бути плюсом чи мінусом в різних ситуаціях. Також потрібно враховувати кількість даних, необхідних для представлення того чи іншого об'єкта - а саме чи потрібно більше пам'яті (байт) для представлення об'єкта в тому чи іншому методі.

Методи серіалізації – досить складна частина будь-якої системи, перед якими стоїть завдання підготувати дані для їх відправки. Використовувані ресурси в ході цього процесу, такі як процесорний час, оперативна пам'ять, розмір результату готового для передачі в інший вузол розподіленої системи - дуже важливі показники, і вони сильно можуть вплинути на ефективність програми. Варто також розуміти, що передача даних в уніфікованому форматі повсюдно, рідко яка система може обійтися без цього прошарку, а це значить, що ці методи використовуються майже скрізь, де є більш-менш складна система.

На даний момент в клієнт-серверній архітектурі панують два найбільш важливих формати серіалізації JSON та XML [6], але вони володіють рядом недоліків, що призводять до занадто великої трати часу для відтворення об'єкта після його передачі. Поширеність їх пов'язана в першу чергу з популярністю і звичною простою реалізацією, але для високонавантажених систем ці критерії вже не так важливі як чиста продуктивність. Так як алгоритми серіалізації використовуються повсюдно, а найбільш популярні орієнтовані більше на зручність їх використання людиною ніж продуктивність, необхідно провести вивчення методів серіалізації і актуально об'єднати в одному методі переваги інших та позбутися від недоліків. Мета даної роботи – оптимізувати методи серіалізації, що дозволить з використанням тих же ресурсів досягати більшої ефективності при побудові клієнт-серверних архітектур, досягати найбільш ефективного спілкування клієнта і сервера. Як результат, без зміни ресурсів системи, але із вибором поліпшеного методу серіалізації вдасться досягти кращих результатів і показників, що само по собі економічно вигідно. Це дозволить використовувати незадіяні ресурси на виконання інших завдань у застосунку.

Оптимізація методів серіалізації повинна бути зосереджена на скорочення часу виконанні серіалізації, використання додаткової пам'яті та розміру отриманих даних, а також кількість використовуваних для цього обчислювальних потоків. Актуальність теми полягає в постійному збільшенні складності систем, а значить, і обсязі переданих даних і необхідності їх скорочення, економії часу на їх створення. З кожним роком, місяцем, тижнем і днем все більше ресурсів витрачається на серіалізацію. Розробка більш ефективного методу необхідна, тому що ціна витрачених ресурсів теж зростає, при цьому необхідність зростатиме в плині часу, так як системи розвиваються і їх складність невпинно зростає.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ

2.1 Особливості клієнт-серверної архітектури

Клієнт-сервер – це обчислювальна чи мережева архітектура, створена для розподілення мережевого навантаження між постачальниками послуг, які і називаються серверами та замовниками послуг, тобто клієнтами [7]. У фізичному контексті клієнт та сервер – це програмне забезпечення. Зазвичай їх взаємодія відбувається за допомогою комп'ютерної мережі, а отже використовуються мережеві протоколи. Не дивлячись на те, що зазвичай ці дві сутності розділяють на різні обчислювальні пристрої, вони можуть виконуватись і на одній машині. Програмою називають процес, коли після отримання запиту від клієнтів, сервери надають їм свої ресурси у вигляді даних (наприклад, завантаження файлів через HTTP, FTP, потокова мультимедія чи робота з базами даних) чи сервісних функцій (наприклад, робота з електронною поштою, комунікації через системи миттєвого обміну повідомленнями, перегляд web-сторінок).

Клієнт-серверний застосунок – це будь-який застосунок, у якому ініціатор дії знаходиться в одній системі, а виконавець – у іншій. Слід зазначити, що для більшості клієнт-серверних застосунків властиво обслуговування одним сервером декількох клієнтів. Як припускає термін, клієнт-серверне середовище складається з клієнтів і серверів. Клієнтські машини - це, як правило, персональні комп'ютери або спеціалізовані робочі станції для одного користувача, із дружнім інтерфейсом. Зазвичай клієнт має дуже зручний графічний інтерфейс користувача, який передбачає наявність вікон і миші. Одні з найвідоміших прикладів таких інтерфейсів – інтерфейси операційних систем Microsoft Windows або Macintosh. Клієнтські програми припускають простоту використання і знайомі інструментальні засоби, наприклад, електронні таблиці. Кожен сервер в клієнт-серверному середовищі надає клієнтам набір послуг. Найбільш поширений тип сервера – це сервер баз даних, що керує реляційною базою даних. Потужний та зазвичай високопродуктивний сервер забезпечує

колективний доступ декількох клієнтів до однієї і тієї ж бази даних. Також до складу клієнт-серверного середовища входить мережа.

Обчислювальна модель клієнт-сервер за визначенням є розподіленою. Це значить, що усі користувачі, програми та ресурси знаходяться на різних машинах і поєднані загальною глобальною, складовою або локальною мережею. Існує декілька характеристик, що відрізняють обчислювальну модель клієнт-сервер від інших схем розподілених обчислень. Надзвичайно велику увагу при розробці клієнт-серверних застосунків приділяють створенню дружнього для користувача інтерфейсу на клієнтському комп'ютері. Таким чином, користувач отримує повний контроль над розкладом і режимом роботи комп'ютера, а менеджери рівня відділів отримують можливість реагувати на локальні проблеми. Незважаючи на те, що застосунки є розподіленими, в архітектурі клієнт-сервер, як правило, використовуються централізовані корпоративні бази даних. Таке рішення дозволяє керівництву корпорації зберігати повний контроль над інвестиціями в інформаційні системи, а також забезпечувати повну зв'язність всіх систем. У той же час така конфігурація позбавляє різні відділи компанії від накладних витрат з управління складними обчислювальними системами та надає їм можливість обирати типи машин і інтерфейси, які їм необхідні для доступу до даних. І виробники, і корпоративні користувачі віддають перевагу відкритим та модульним системам. Це значить, що користувачеві надається більш широкий вибір продуктів і можливість об'єднувати обладнання від різних виробників. З одного боку, архітектура клієнт-сервер – це природне рішення з точки зору виробника, так як в ній використовуються все більш доступні мікрокомп'ютери та мережі. Також, архітектура клієнт-сервер, можливо, є ідеальним вибором для підтримки обраного організацією напрямку бізнесу. Ці та інші тенденції в світі бізнесу стали стимулом для збільшення інвестицій в технологію клієнт-сервер. Звичайно, як і будь-яка кардинальна зміна комп'ютерної конфігурації, перехід на архітектуру клієнт-сервер не є безпечним чи безболісним. У таблиці 2.1 показано, що користувачі на ряду із одержаними перевагами, повідомляють про безліч проблем при переході на архітектуру клієнт-сервер. Проте, архітектура

клієнт-сервер залишається домінуючою саме завдяки зниженню вартості та зросту популярності персональних комп'ютерів, а також завдяки зростаючій конкуренції у промисловості.

Таблиця 2.1 – Переваги та недоліки архітектури клієнт-сервер

Системна характеристика	Значення
Переваги	
Мережа невеликих потужних машин	Якщо одна машина вийде з ладу, компанія все одно зможе продовжувати роботу
Потужні системи комп'ютерів	Система надає потужність, що дозволяє виконувати роботу без монополізації ресурсів. У кінцевих користувачів виявляється достатньо потужності для локальних обчислень
Деякі робочі станції такі ж потужні, як і мейнфрейми, але їх вартість на порядок нижча	Надаючи обчислювальні потужності за меншу вартість, система дозволяє витратити зекономлені кошти на інші придбання чи на збільшення доходів
Відкриті системи	Апаратуру, програми та послуги можна придбати у різних виробників
Розширюваність системи	Систему не важко модернізувати як тільки виникне потреба

Продовження таблиці 2.1

Індивідуальна робоча область клієнта	Можливість об'єднувати комп'ютерні платформи, підбираючи їх під конкретні потреби підрозділів чи користувачів
Недоліки	
Слабка підтримка	Окремі частини системи не завжди коректно працюють разом. Буває доволі важко знайти причину несправності
Виокремлення основних обчислювальних процесів на серверну частину	Непрацездатність сервера може стати причиною непрацездатності всієї системи
Специфічність	Підтримка роботи системи потребує окремого спеціаліста – системного адміністратора

2.2 Застосунки з клієнт-серверною архітектурою

Найважливішою характеристикою обчислювальної моделі клієнт-сервер є розподіл прикладних задач між клієнтами та серверами. Загальний випадок представлений на рисунку 2.1. Операційна система – базове програмне забезпечення як для клієнта, так і для сервера, проте, вони, як і апаратні платформи можуть відрізнятися. В одному і тому самому середовищі можуть використовуватися різні типи клієнтських та серверних платформ і операційних систем. Ці відмінності не мають жодного значення, якщо сервер і клієнт використовують одні комунікаційні протоколи та підтримують однакові застосунки. Не дивлячись на різницю, саме комунікаційним програмним

забезпеченням підтримується взаємодія клієнта та сервера. На рисунку 2.1 представлена загальна архітектура клієнт-серверної системи. Прикладами такого програмного забезпечення є набір протоколів TCP / IP, - протоколи OSI, а також різні фірмові архітектури, як SNA [8]. Очевидно, це програмне забезпечення підтримки (протоколів та операційної системи надає базу для функціонування розподілених застосунків. В ідеальному випадку функція, що виконується застосунком повинна бути розподілена між клієнтом і сервером так, щоб оптимально використовувати обчислювальні та мережеві ресурси найбільш оптимально, а користувачі мали можливості для виконання різних завдань та виконання спільної роботи. Для деяких випадків необхідно, щоб більша частина програмного забезпечення виконувалася на сервері, тоді як в інших випадках більша частина логіки може бути реалізована на клієнті.

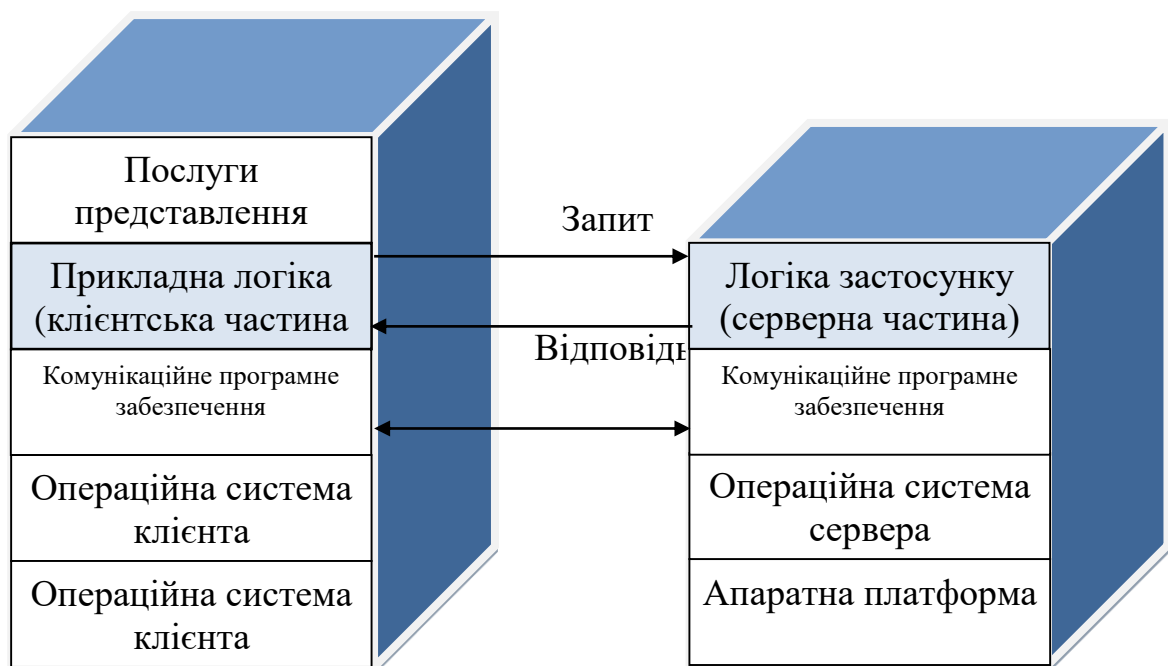


Рисунок 2.1 – Загальна архітектура клієнт-сервер

На завершення, головним впливовим фактором є спосіб взаємодії користувача із системою, тобто інтерфейс клієнтської машини. Для більшості клієнт-серверних систем приділяється дуже серйозна увага графічному інтерфейсу користувача (Graphical User Interface, GUI) - він повинен бути

простим і зручним, але і одночасно потужним та гнучким. Таким чином, можна сказати, що модуль подання послуг робочій станції вважається відповідальним за дружній інтерфейс із розподіленими застосунками.

2.3 Бази даних

Концепцію розподіленої між клієнтом та сервером логіки застосунку можна прослідкувати на прикладі реляційної бази даних. В даному випадку сервер називається сервером баз даних. Клієнт та сервер взаємодіє за допомогою транзакцій, в яких клієнт посилає серверу запит та отримує відповідь від нього. Архітектуру такої системи представлено на рисунку 2.2. За управління базою даних відповідальний сервер. На клієнтських машинах можуть розташовуватися різні застосунки, які користуються базою даних.

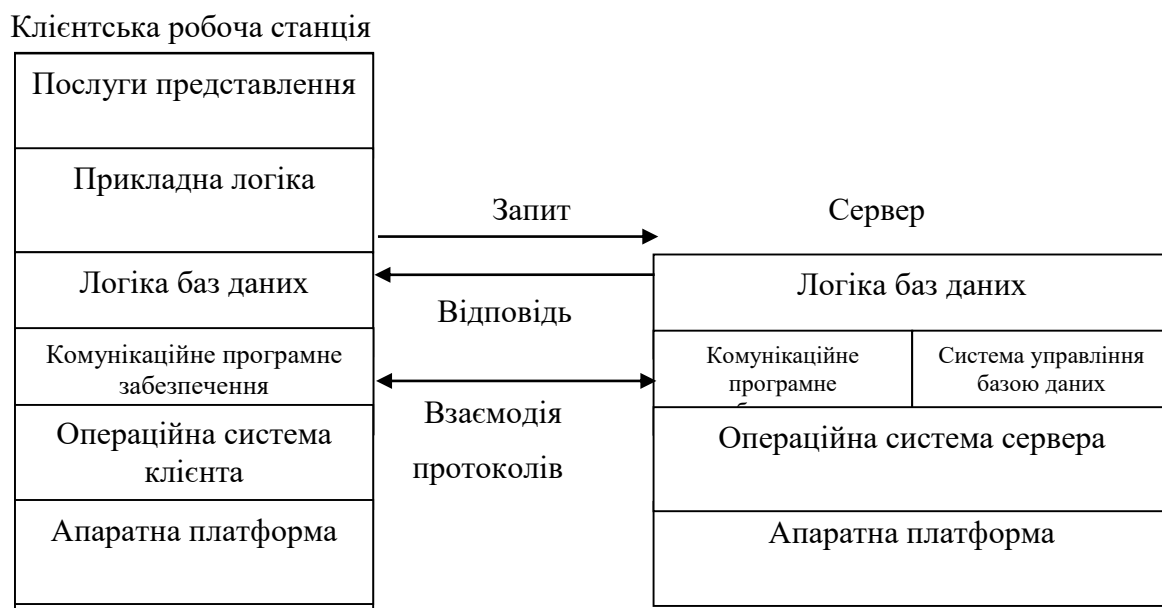


Рисунок 2.2 – Архітектура клієнт-сервер для баз даних

Існує спеціалізоване програмне забезпечення, що пов'язує клієнт та сервер, що дозволяє клієнту виконувати запити та на основі їх отримувати доступ до

бази даних. Популярним прикладом такої логіки є мова структурованих запитів (Structured Query Language, SQL). З рисунку 2.3 видно, що вся прикладна логіка, тобто програми для обробки та аналізу даних - розташовується на клієнтській стороні, тоді як сервер займається лише управлінням базою даних [9]. Застосування такої конфігурації залежить від стилю та завдання конкретного застосунку, наприклад, якщо головна мета програми забезпечити доступ для пошуку записів в режимі підключення (on-line). Приклад роботи такої схеми показаний на рисунку 2.3, а.

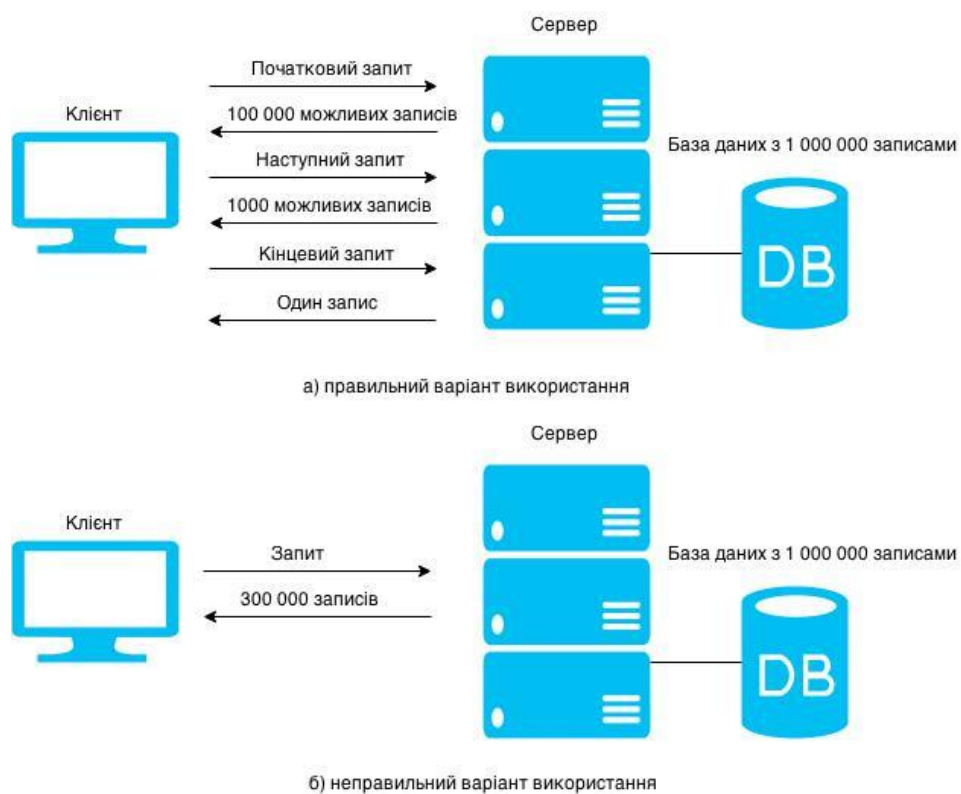


Рисунок 2.3 – Використання бази даних в архітектурі клієнт-сервер

Архітектура такого застосунку добре відповідає критеріям архітектури клієнт-сервер з двох причин.

По-перше, з базою даних проводиться великий обсяг робіт із сортування та пошуку даних. Для цього необхідно використовувати великий диск або систему дисків, разом із потужним процесором, що виступає в якості центрального та високошвидкісна архітектура вводу-виводу. Такі потужності не потрібні для робочої станції одному користувачі або для персонального комп'ютера.

Очевидно, що переміщення файлу з мільйоном записів для пошуку на клієнтську машину було б занадто важким тягарем для мережі [8]. Таким чином, окрім того, щоб просто отримувати доступ до записів від імені клієнта, сервер повинен також володіти логікою бази даних. Така властивість дозволить йому виконувати операції пошуку від імені клієнта.

Розглянемо тепер рисунок 2.3, б, який ілюструє сценарій роботи з тією ж самою базою даних, яка містить мільйон записів. Різниця полягає в тому, що в результаті одного запиту користувачу по мережі пересилається 300000 записів. Таке може статися, якщо, наприклад, користувач захоче визначити суму або середнє значення певного поля для великої множини записів або навіть для всієї бази даних. Зрозуміло, що подібні дії є неприпустимими. Одне з можливих рішень даної проблеми полягає в тому, щоб перемістити частину логіки застосунку на сервер. Тобто, окрім функцій доступу до записів бази даних і функцій пошуку записів, сервер може відповідати за прикладну логіку аналізу даних.

2.4 Клієнт-серверні технології

Архітектура клієнт-сервер застосовується для великого числа мережевих технологій, що використовуються для доступу до різних мережевих сервісів.

Розглянемо найпопулярніші та найпоширеніші типи серверів та сервісів, що повсякчас використовуються у застосунках із клієнт-серверною архітектурою.

Поштові сервери. Надають послуги з отримання електронних поштових повідомлень та їх відправки.

Файл-сервери. Зберігає інформацію у вигляді файлів та надає користувачам доступ до неї. Очевидно, що файл-сервер має забезпечувати необхідний рівень захисту інформації та має бути здатен захистити файли від несанкціонованого доступу. Зазвичай існує певна система привілеїв, яка на базі

отриманого ідентифікаційного номеру користувача надає йому відповідні привілеї.

Web-сервери. Спочатку використовувались для надання доступу до гіпертекстових документів по протоколу HTTP. Наразі використовуються для підтримки розширених можливостей, зокрема, роботи з бінарними файлами (зображеннями, мультимедіа тощо).

Сервери баз даних. Призначені для обробки запитів користувачів на мові SQL. СУБД має розташовуватись на сервері, до якого під'єднуються клієнтські програми.

Проксі-сервер. Найпоширеніше використання проксі-сервера – як посередника для того, щоб отримати інформацію з Інтернету чи будь-якої іншої мережі, при цьому забезпечуючи захист локальної мережі. Проте, такий сервер вміє зберігати часто запитувану інформацію в кеш-пам'яті на жорсткому диску, задля того, щоб швидко доставляти її користувачам без повторного звернення до серверу (мережі).

Сервери застосувань. Використовуються для централізованого вирішення прикладних задач у певній предметній області. Користувачі можуть запускати серверні програми на виконання. Серверні застосунки використовуються для того, щоб знизити вимоги до конфігурацій клієнтів та спростити загальне керування мережею.

Файрволи (брандмауери). Брандмауери – це міжмережеві екрани, які аналізують та фільтрують мережевий трафік, що проходить через них, для подальшого забезпечення безпеки мережі.

Сервери віддаленого доступу. Це системи, які створені для забезпечення зв'язку машини із мережею по комутованих лініях. Наприклад, співробітник, що працює віддалено, може звертатися до ресурсів корпоративної ЛКМ, за допомогою звичайного модему.

Наразі було наведено лише декілька типів з усього різноманіття клієнт-серверних технологій, що широко застосовуються як в глобальних, так і в локальних мережах. Зазвичай використовують клієнтів для доступу до тих чи

інших мережевих сервісів. Можливості таких клієнтів характеризуються поняттям «товщини». Воно визначає програмне забезпечення та конфігурацію устаткування, наявні у клієнта.

«Тонкий» клієнт. Цей термін визначає такого клієнта, який має достатньо обчислювальних ресурсів лише для запуску необхідної мережевої програми через web-інтерфейс. Зазвичай інтерфейс такого застосунку формується засобами статичного HTML (виконання JavaScript не передбачається), вся прикладна логіка виконується на сервері. Для роботи тонкого клієнта достатньо лише забезпечити можливість запуску web-браузера, у вікні якого і здійснюються всі дії. З цієї причини web-браузер часто називають "універсальним клієнтом".

«Товстий» клієнт. Під цим поняттям розуміють персональний комп'ютер або робочу станцію що має необхідний набір програмного забезпечення та працює під управлінням власної дискової операційної системи. За додатковими послугами (наприклад, доступ до web-серверу або корпоративній базі даних) «товсті» клієнти звертаються в основному до мережевих серверів. Під «товстим» клієнтом також мається на увазі і клієнтський мережевий застосунок, що працює під керуванням локальної ОС. Такий застосунок поєднує в собі компонент представлення даних (графічний користувальницький інтерфейс ОС) та прикладних компонент (обчислювальні потужності комп'ютера клієнта).

Останнім часом все частіше використовується ще один термін: «rich» клієнт. «Rich» клієнт – це свого роду компроміс між «товстим» і «тонким» клієнтом. Як і «тонкий» клієнт, «rich» клієнт також надає графічний інтерфейс, але вже створений засобами XML та реалізовує деяку функціональність товстих клієнтів (наприклад інтерфейс drag-and-drop, вкладки, множинні вікна, що випадають з меню і т.д.). Прикладна логіка «rich» клієнтів також зосереджена на сервері. Дані відправляються, застосовуючи стандартний формат обміну, створений також на основі XML (протоколи SOAP, XML-RPC) і інтерпретуються на протилежній стороні клієнтом. Надалі наведено основні протоколи «rich» клієнтів на базі XML:

- XUL – це стандарт, розроблений в рамках проекту Mozilla, що використовується, наприклад, в поштовому клієнті Mozilla Thunderbird або браузері Mozilla Firefox;
- XAML – розроблений Microsoft та використовується в застосунках на платформі .NET;
- Flex – мультимедійна технологія створена на основі XML, розроблена Macromedia/Adobe.2.4.1.

2.4.1 Дволанкова клієнт-серверна архітектура

Елементи клієнт-серверної взаємодії присутні у будь-якій мережі, побудованій на сучасних мережевих технологіях (навіть непостійній). Найчастіше такі елементи створені на основі дволанкової архітектури. Дволанковою вона називається через розподіл трьох базових компонентів між двома вузлами (клієнтом та сервером) [10]. Резонно використовувати дволанкову архітектуру в таких клієнт-серверних системах, де сервер безпосередньо і в повному обсязі відповідатиме на клієнтські запити, використовуючи лише власні ресурси. Тобто, сервер не викликає сторонні мережеві застосунки і не звертається до сторонніх ресурсів для виконання будь-якої частини запиту (рисунок 2.4).



Рисунок 2.4 – Дволанкова клієнт-серверна архітектура

Наступні основні моделі взаємодії клієнта та сервера в рамках дволанкової архітектури визначають розташування компонентів на тій чи іншій стороні відповідно [10]:

- сервер застосувань - віддалений програмний засіб.
- сервер терміналів - розподілене представлення даних;
- сервер БД - віддалене представлення даних;
- файл-сервер - доступ до файлових ресурсів та віддаленої бази даних.

Перераховані моделі з різними варіаціями представлені на рисунку 2.5.

Першою була розроблена модель розподіленого представлення даних (модель сервер терміналів). Реалізовувалась така модель на універсальній ЕОМ (мейнфрейми), що виступала в якості сервера, з підключеними до неї алфавітно-цифровими терміналами. Користувачі вводили вхідні дані з клавіатури або терміналу, які потім передавалися на мейнфрейм де виконувалась їх подальша обробка, включаючи генерацію «картинки» з результатами. Ця «картинка» далі відображалась користувачеві на екрані терміналу.

Після появи локальних мереж та персональних комп'ютерів була впроваджена модель файлового серверу, який вмів надавати доступ до файлових ресурсів, в тому числі і до віддаленої бази даних [11]. В цій моделі виділений вузол мережі є файловим сервером, на якому розміщені файли бази даних. Застосунки виконуються на клієнтах, які є поєднанням компонент подання та прикладних, що використовують під'єднану віддалену базу як локальний файл. Протоколи обміну при цьому функціонують як набір викликів операцій файлової системи низького рівня. Але при активній роботі з таблицями БД у цій системі виникало велике навантаження на мережу, що і стало першим показником неефективності такої моделі. Проте часткове рішення - це підтримка тиражування (реплікації) таблиць і запитів. У такому випадку при зміні даних, наприклад, оновлюється не вся таблиця, а тільки та частина, яка була модифікована.

Коли з'явилися спеціалізованих СУБД, можливість реалізації іншої моделі доступу до віддаленої бази даних - моделі серверу баз даних – стала реальною. Коротко кажучи, у такій моделі ядро СУБД розташовується на сервері, а прикладна програма – на клієнті. Мова SQL забезпечує протокол обміну між вузлами. У порівнянні із файловим сервером, такий підхід веде до зменшення

трафіку мережі й уніфікації інтерфейсу «клієнт-сервер». Хоча, завантаження мережі все ж залишається доволі високим. Окрім того, стало неможливе адміністрування застосунків, оскільки в одній програмі сполучаються різні функції.

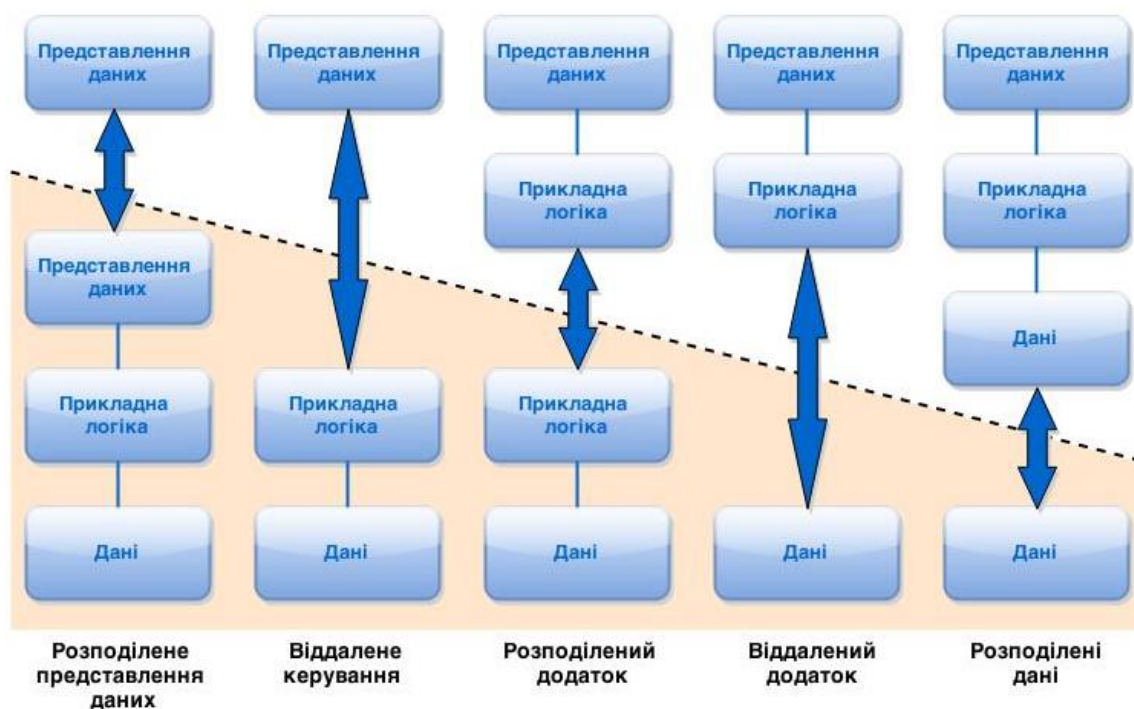


Рисунок 2.5 – Моделі клієнт-серверної взаємодії

Поряд із впровадженням і розробкою баз даних та механізму збережених процедур на рівні серверів активно застосовувалась концепція активного сервера БД. Для цього випадку частина функцій прикладного компонента реалізовані у вигляді збережених процедур, що виконуються на стороні сервера. Інша прикладна логіка функціонує на клієнтській стороні [12]. Замість протоколу взаємодії виступає специфічний діалект мови SQL.

Наразі помітна тенденція повернення до того, з чого починалася клієнт-серверна архітектура – використання моделі термінал-сервера для централізації досліджень. Згідно до [13] Різниця терміналів від їх алфавітно-цифрових предків у сучасному світі лише у тому, що маючи мінімум апаратних та програмних засобів, вони надають безліч мультимедійних можливостей (в тому числі

графічний інтерфейс). високопродуктивний сервер забезпечує роботу терміналів, куди винесено все, починаючи від віртуальних драйверів пристроїв та драйверів відеопідсистем.

2.4.2 Триланкова архітектура клієнт-сервер

Традиційна архітектура клієнт-сервер має два рівня, або ланки: клієнтський і серверний. За останні роки все більшого поширення набувала саме триланкова архітектура клієнт-сервер (рисунок 2.6). У даній архітектурі прикладне програмне забезпечення розподілено між машинами трьох типів: користувальницької машиною, проміжним сервером та сервером бази даних, про що наведено у статті [14]. Машина користувача – це простий клієнт, а в триланковій моделі, як правило, «тонкий» клієнт. Проміжні машини по суті являються шлюзами між «тонкими» клієнтами та різноманітними серверами, на яких функціонують бази даних. Проміжні машини повинні вміти перетворювати протоколи і проектувати одні типи запитів до баз даних на інші. Також проміжні машини мають вміти об'єднати результати запитів отримані з різних джерел даних. На кінець, ці машини мають відігравати роль шлюзів між десктопними застосунками і тими, що залишилися в організації систем управління базами даних за минулих часів, таким чином, існуючи як посередник між двома «світами».



Рисунок 2.6 – Триланкова архітектура клієнт-сервер

Взаємодія між проміжним сервером та сервером баз даних також відповідає моделі клієнт-сервер. Отже, проміжна система функціонує одночасно і як клієнт, і як сервер.

2.5 Передача повідомлень

Розподілену передачу повідомлень, що реалізує функціональність архітектури клієнт-сервер, ілюструє рисунок 2.7, а. Клієнтський процес запрошує певну послугу. Для цього він надсилає повідомлення із запитом даної послуги серверному процесу. Серверний процес приймає повідомлення, обробляє запит і відсилає повідомлення у відповідь. У найпростішому випадку необхідно реалізувати тільки дві функції: Receive (прийняти) та Send (надіслати). На вході функції Send необхідно вказати одержувача, а також зміст повідомлення. Для Функції Receive потрібно вказати, від кого очікується повідомлення (включаючи варіант «всі») та адресу буфера, в котрий необхідно помістити прийняте повідомлення.

На рисунку 2.8 представлена схема реалізації можливого механізму передачі повідомлень. Процеси користуються послугами, які надають модулі передачі повідомлень. Запити на ці послуги можуть приймати вигляд примітивів чи параметрів. Примітив відповідає функції, що виконується, а для контролю над інформацією чи для передачі даних використовуються параметри. Конкретна форма примітивів залежить від програмного забезпечення передачі повідомлень. Це може бути як виклик процедури, так і передача повідомлення процесу операційної системи [15].

Примітив Send використовується процесом, що бажає відправити повідомлення. Вхідними параметрами цього примітиву є ідентифікатор процесу, що отримує і зміст повідомлення. Модуль передачі повідомлень формує блок даних, які включають в себе ці два елементи. Цей блок даних надсилається на машину, на котрій працює отримуючий процес, за допомогою будь-яких

мережевих протоколів, наприклад, TCP / IP. Коли адресат отримує блок даних, цей блок пересилається модулю передачі повідомлень. Даний модуль досліджує поле ідентифікатора процесу і зберігає повідомлення в буфері цього процесу.

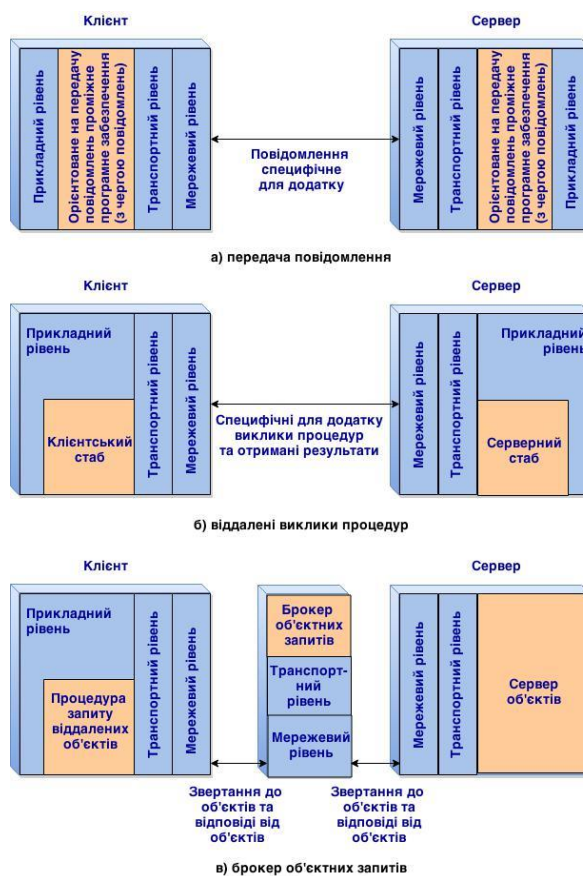


Рисунок 2.7 – Механізми реалізації проміжного програмного забезпечення



Рисунок 2.8 – Базові примітиви передачі повідомлень

У вищезазначеному сценарії отримуючий процес повинен оголосити про своє бажання отримувати повідомлення, виділити буфер для повідомлень та інформувати модуль передачі повідомлень за допомогою примітиву Receive про

свій стан. Альтернативний метод не вимагає подібного оголошення. Просто коли модуль передачі повідомлень отримує повідомлення, він сигналізує про це процесу, якому вона адресована, а потім розміщує повідомлення в загальний буфер.

Існують кілька варіантів реалізації механізму розподіленої передачі повідомлень, про які буде говоритися далі.

Надійна передача повідомлень. Надійною називається така система передачі повідомлень, яка гарантує доставку повідомлення, якщо така можлива. У такій системі використовується надійний транспортний протокол, а також забезпечується підтримка контролю над помилками, повторна передача і відновлення проходження повідомлень. Оскільки доставка гарантована, немає необхідності повідомляти передаючому процесу про те, що повідомлення було прийняте. Проте, може бути корисно надавати передаючому процесу підтвердження, яке б повідомило його про те, що транзакція пройшла успішно. Проте, якщо системі все ж не вдається доставити повідомлення (наприклад, через несправність мережі або виході з ладу одержувача), передаючий процес повідомляється про невдачу. Іншу крайність являє собою система передачі повідомлень, яка просто надсилає повідомлення в мережу, не повідомляючи ні про успіх, ні про невдачу. Така альтернатива дозволяє значно спростити систему передачі повідомлень і накладні витрати на обробку та взаємодію. Але ж якщо застосунку необхідне підтвердження доставки повідомлення, такий функціонал може бути реалізований на прикладному рівні.

Синхронні та асинхронні системи передачі повідомлень. При зверненні до асинхронного примітиву, процес не припиняється. Отже, після того як процес викликає примітив `Send`, операційна система повертає процесу управління відразу після постановки повідомлення в чергу на передачу або після створення копії повідомлення. Коли повідомлення надіслане або скопійовано в безпечне місце для подальшої передачі, передаючий процес переривається і інформується таким чином про те, що буфером повідомлення можна користуватися знову. Якщо копії повідомлення не створюється, то будь-які зміни повідомлення,

вироблені передавальним процесом вже після звернення до примітиву Send, але до відправки повідомлення, є ризикованими. Аналогічно, після звернення до асинхронного примітиву Receive процес продовжує роботу. Коли повідомлення доставлене, процес інформується про цю подію шляхом переривання або періодичного опитування. Асинхронні примітиви забезпечують ефективно і гнучке звернення процесів до системи передачі повідомлень. Недолік цього способу становить те, що програми, які використовують подібні примітиви, важко тестувати і налагоджувати. Події, що залежні від часу і які неможливо відтворити, можуть стати джерелом складних проблем. Альтернатива полягає у використанні синхронних або, як їх ще називають, блокуючих примітивів. При виклику синхронного примітиву Send управління не повертається передаючому процесу доки повідомлення не віддане (ненадійне обслуговування), або до тих пір, поки не отримано підтвердження про отримання повідомлення (надійне обслуговування). До блокуючого примітиву Receive не повернеться управління, доки сполучення не виявиться у виділеному для нього буфері.

2.6 Веб-сервер та веб-сервіс

Веб-сервіс – це спосіб зв'язку між двома електронними пристроями по мережі. Це набір зв'язних функцій, що виконують певні дії чи надають інформацію за певною мережевою адресою [16]. Доступ до сервісу надає інтерфейс описаний у машино-прийнятному форматі, що інкапсулює його реалізацію. Взаємозв'язок здійснюється за правилами системи, що відсилає запит із використанням SOAP [17] (Simple Object Access Protocol) повідомлення, що, як правило, передається за допомогою HTTP та XML серіалізації у поєднанні з іншими існуючими web-стандартами. Багато організацій використовують одразу декілька систем програмного забезпечення для здійснення управління і різним програмним системам часто необхідно обмінюватися даними один з одним. Веб-

сервіс – це спосіб зв'язку, що дозволяє двом системам програмного забезпечення обмінюватися даними через мережу. Програмне забезпечення, яке запитує дані називається замовник сервісу, в той час як програмне забезпечення, яке буде обробляти запит і надавати дані, називається постачальником послуг.

Різні програми можуть бути створені з використанням різних мов програмування, а, отже, існує необхідність у таких способах обміну даними, що не залежать від конкретної мови програмування чи платформи. Наприклад, більшість видів програмного забезпечення, здатні інтерпретувати XML теги, а значить, веб-сервіси зможуть використовувати XML-файли для обміну даними. Правила для зв'язку між різними системами мають бути визначені наступним чином:

- як система може створити запит на дані з іншої системи;
- які параметри необхідні для запиту даних;
- яка структура отримуваних даних. Як правило, обмін даними за допомогою XML-файла, а також структура XML-файлу звіряється по XSD-файлу;
- які повідомлення про помилку слід видавати для того щоб легше усувати несправності коли правила взаємодії не очевидні;

Всі ці правила для спілкування визначаються у файлі WSDL, який має розширення .wsdl.

Архітектура веб-сервісів. Постачальник послуг відправляє файл WSDL для UDDI. Замовник сервісу зв'язується з UDDI, щоб визначити, хто є постачальником для даних, а потім встановлює контакт із постачальником послуг за допомогою протоколу SOAP. Постачальник послуг перевіряє запит на обслуговування і передає структуровані дані в XML-файлі, також використовуючи протокол SOAP. Цей XML-файл буде підтверджено знову після звірення замовника отриманого файлу із файлом XSD. UDDI визначає, до якого саме програмного забезпечення системи слід звернутися та за яким типом даних. Тому, коли якійсь системі програмного забезпечення необхідно дістати дані, вона звернеться до UDDI і з'ясує, з якими системами можна зв'язатися для

отримання відповідних даних. Після того, як система ідентифікує систему, з якою необхідно зв'язатися, вона встановлює контакт із використанням спеціального протоколу SOAP. Система-постачальник послуг в першу чергу перевірить правильність запиту даних з посиланням на файл WSDL, а потім опрацює запит і відправить дані відповідно до протоколу SOAP.

Сервісна модель - це метод сприйняття застосунку як набору засобів або сервісів, які задовольняють запити клієнтів. Моделювання програми у вигляді набору окремих сервісів дозволяє повторно використовувати компоненти, надає доступ до них інших застосунків і допомагає розподіляти їх виконання між декількома комп'ютерними вузлами чи мережами.

У типових бізнес-застосунках можливі сервіси трьох категорій, представлені у таблиці 2.2

Таблиця 2.2 – Типи веб-сервісів

Тип сервісу	Розміщення	Призначення
Користувача	Клієнт	Представлення інформації та доступу до функцій програми, навігація, збереження цілісності і несуперечності користувача інтерфейсу

Бізнес-сервіс	Сервер	Реалізація основних стратегій, генерація інформації на основі даних і підтримка цілісності середовища прийняття бізнес-рішень
Сервіс даних	Сервер	Структуризація даних, їх зберігання, вилучення та підтримка цілісності

Веб-застосунки – це нова розробка у сфері веб-послуг, мета якої рухатися до більш простих репрезентативна способів передачі (REST) на основі комунікації. RESTful API не вимагають протоколи веб-служб на XML-основі (SOAP і WSDL), для підтримки своїх інтерфейсів. Критики веб-служб, заснованих не на RESTful часто виділяють складність таких застосунків, а також те, що вони орієнтуються на широковідомих постачальників програмного забезпечення більше ніж на звичні open-source імплементації. Також ведуться дискусії щодо швидкодії веб-служб через використання XML та SOAP/HTTP-технологій

Веб-сервер - сервер, що приймає HTTP - запити від клієнтів, та видає їм HTTP-відповіді, можливо, разом із медіа-поток, зображенням, HTML-сторінкою, файлом або іншими даними. Веб-сервером по суті - програмне забезпечення, яке виконує функції сервера, а також комп'ютер на якому це програмне забезпечення розгорнуте. На одному веб-сервері можуть працювати декілька веб-сервісів, які можуть виступати у ролі повноцінного веб-застосунку,

або як окремі функції чи служби. Веб-сервери можуть мати різні додаткові функції, наприклад:

- аутентифікація і авторизація користувачів;
- автоматизація роботи веб-сторінок;
- підтримка динамічно генеруються сторінок;
- ведення журналу звернень користувачів до ресурсів;
- підтримка протоколу HTTPS для з'єднань з клієнтами, що захищаються.

Часто разом із веб-сервером на комп'ютері встановлюється також і поштовий сервер.

При проведенні аналізу існуючих форматів серіалізації та проектуванні власного формату, постане необхідність спроектувати клієнт-серверну архітектуру. Нажаль, використання одразу декількох веб-сервісів на єдиному веб-сервері може провокувати певні непрогнозовані помилки у разі, наприклад, війни між сервісами за ресурс чи будь-яких інших конфліктах. Виконання серіалізації повідомлень, що будуть передаватися між клієнтом та сервером не потребує складної серверної архітектури, лише імплементація обраних форматів серіалізації. Таким чином, було прийняте рішення використовувати веб-сервіс або систему веб-сервісів в залежності від складності імплементації, у якості сервера.

2.7 Серіалізація даних

У комп'ютерній техніці, в контексті зберігання даних, серіалізація – це процес перетворення структур даних або стану об'єкта у формат, який може бути збережений (наприклад, у файлі або пам'яті буфера, або переданої по мережевому зв'язку) і реконструйований в тому ж або іншому комп'ютерному

середовищі. Коли результуюча послідовність бітів зчитується відповідно до формату серіалізації, вона може бути використана для створення семантично-ідентичного клону вихідного об'єкта. Для багатьох складних об'єктів, таких як ті, які містять посилання на інші об'єкти, цей процес не є простим. Серіалізація об'єктно-орієнтованих об'єктів не включає в себе серіалізацію будь-яких асоціативних методів, з якими вони нерозривно зв'язані. Процес серіалізації об'єкта також називають маршаллінгом, а зворотна операцію, формування структури даних з послідовності байтів – це десеріалізація (яку також називають демаршаллінг). Серіалізація використовується як:

- спосіб передачі даних по лініям (повідомлень);
- метод зберігання даних (у базах даних, на жорстких дисках);
- метод віддаленого виклику процедур, наприклад, як в SOAP.
- метод розподілу об'єктів, особливо в основі компонентів програмного забезпечення, таких як COM чи CORBA;
- спосіб виявлення змін в нестационарних даних.

Для деяких з цих функцій може бути корисно, коли архітектурна незалежність збережена. Наприклад, для максимального використання розподілу, комп'ютери, які працюють на іншій апаратній архітектурі, повинні бути в змозі надійно відновити серіалізований потік даних, незалежно від розташування байтів. Це означає, що простіша і швидша процедура прямого копіювання розподілу пам'яті структури даних може не працювати надійно для будь-якої з архітектур. Серіалізація структури даних у формат, незалежний від архітектури дає можливість запобігати проблемам упорядкування байтів, розподілу пам'яті, або просто різних способів представлення структур даних у різних мовах програмування. Властивістю будь-якої схеми серіалізації є те, що, оскільки кодування даних за визначенням послідовне, витяг однієї частини послідовної структури даних вимагає прочитати весь об'єкт повністю та реконструювати його. У багатьох застосунках ця лінійність присутня, тому що це дозволяє імплементувати прості, загальні інтерфейси введення / виводу, які будуть використовуватися для передачі та зберігання інформації про стан

об'єкта. У програмах, де необхідна висока продуктивність така лінійність є проблемою, і може мати сенс витратити більше зусиль, щоб створити нелінійну, більш складну організацію зберігання.

Зберігати покажчики на примітивні об'єкти ненадійно навіть на одній машині, тому що об'єкти, на які вони вказують, можуть бути переміщені в іншу область пам'яті. Для вирішення цієї проблеми, процес серіалізації включає в себе етап, що називається «unswizzling», який перетворює прямі посилання покажчика у посилання, засновані на імені чи позиції. Процес десеріалізації включає в себе зворотній крок під назвою «swizzling». З того часу, як стало можливо керувати процесом серіалізації та десеріалізації за допомогою коду (наприклад, функція серіалізації в Microsoft Foundation Classes), з'явилась можливість виконувати обидві операції одночасно, а значить і визначати відмінності між оригінальними та серіалізованими об'єктами та використовувати ці дані при повторній серіалізації цього об'єкту. При цьому, немає необхідності, формувати копію оригінального об'єкту, тому що відмінності можуть бути визначені на льоту. Така техніка називається диференційне виконання. Її корисно використовувати при програмуванні користувальницьких інтерфейсів, зміст яких змінюється в часі – створювати, видаляти, змінювати графічні об'єкти або обробляти подію введення без написання додаткового окремого функціоналу.

Проте, серіалізація має і недоліки. Серіалізація порушує скриті деталі реалізації абстрактного типу даних, потенційно викриваючи їх. Тривіальні реалізації, які серіалізують всі поля об'єктів можуть порушувати інкапсуляцію. Щоб перешкодити конкурентам копіювати чи створювати сумісні продукти, видавці широковідомих програмних забезпечень часто тримають деталі реалізації серіалізації у якості комерційної таємниці. Деякі навмисно приховують або навіть шифрують серіалізовані дані. Тим не менш, сумісність вимагає, щоб програми могли зрозуміти формати серіалізації один одного. Тому архітектури викликів віддалених методів, таких як CORBA зчитують такі формати серіалізації в деталях.

Багато установ, таких як бібліотеки намагаються зберегти свої резервні архіви на майбутнє, наприклад, дампи баз даних, зберігаючи їх у якомусь форматі, що буде легко прийнятний для людського сприйняття.

Існує багато об'єктно-орієнтованих мов програмування, що безпосередньо підтримують серіалізацію об'єктів (або архівацію об'єктів) або надання стандартного інтерфейсу для реалізації. Деякі з цих мов програмування – Ruby і Smalltalk, Python, PHP, Objective-C, Java, та сімейство .NET. Є також доступні бібліотеки, які додають функціонал серіалізації мовам, що не мають його у стандартному наборі бібліотек [18]

Java.

Java підтримує вбудовану серіалізацію, яка вимагає, щоб об'єкт був імплементацією інтерфейсу `java.io.Serializable`. Реалізація інтерфейсу визначає клас як сутність, що можна серіалізувати, а після Java виконує серіалізацію. Інтерфейс `Serializable` не має жодних визначених методів серіалізації, проте клас, що серіалізується може додатково визначити методи із певними спеціальними іменами та підписами, які в такому разі будуть викликатись як частина процесу серіалізації/десеріалізації. Мова також дозволяє розробнику перевизначити процес серіалізації для більш ретельної реалізації через імплементацію іншого інтерфейсу – `Externalizable`, який включає в себе два спеціальних метода, що використовуються для збереження та відновлення стану об'єкта.

Існує три основні причини, чому об'єкти не можуть бути серіалізовані за замовчуванням і повинні реалізувати `Serializable` інтерфейс для доступу до механізму серіалізації Java. По-перше, не будь-яку семантику можна використовувати для області серіалізації. Наприклад, об'єкт класу `Thread` пов'язаний зі станом поточного JVM. Не існує контексту, в якому десеріалізований об'єкт `Thread` підтримуватиме корисну семантику. По-друге, структура серіалізованого об'єкту формує структуру зв'язних із ним класів. Підтримка версій серіалізованих класів вимагає додаткових зусиль та уваги. Тому, виконання серіалізації класу – це скоріше свідоме рішення архітектора, аніж умова за замовчуванням. Нарешті, серіалізація дозволяє отримати доступ

до приватних членів класу, які інакше не доступні. Класи, що містять конфіденційну інформацію (наприклад, пароль) не повинні бути серіалізовані. Стандартний метод кодування використовує простий переклад полів в потік байтів. Примітиви, а також non-transient чи не non-static посилання на об'єкти кодуються в потік. Кожен об'єкт, на який посилається серіалізований об'єкт і не помічений як transient також повинен бути серіалізований. Розробник може впливати на таку поведінку, позначаючи об'єкти як transient або шляхом перевизначення серіалізації для об'єкта, так що частина структури посилань буде зменшена, а не серіалізована повністю. Можна також серіалізувати об'єкти Java через JDBC і зберігати їх в базі даних. І хоча компоненти Swing реалізують Serializable інтерфейс, вони не портативні між різними версіями Java Virtual Machine. В такому випадку, компонент Swing, або будь-який компонент, який його успадковує, можна серіалізувати в масив байтів, але це не гарантує, що збережений об'єкт зможе бути прочитаний на іншому комп'ютері.

.NET Framework.

.NET Framework має кілька серіалізаторів розроблених Microsoft. Є також багато серіалізаторів третіх осіб. Більше десятка серіалізаторів обговорені та протестовані і їх список постійно поповнюється.

CFML.

CFML дозволяє серіалізувати структури даних у WDDX з тегом <cfwddx> та JSON формати за допомогою функції SerializeJSON().

OCaml.

Стандартна бібліотека OCaml підтримує серіалізацію через Маршал модуль та функції output_value і input_value. У той час як в OCaml тип даних статично перевіряється, використання модулю Маршала може порушити гарантію типу, оскільки немає ніякого способу, щоб перевірити, чи є об'єкти потоку unmarshalled очікуваного типу. У OCaml важко серіалізувати функцію або структуру даних, яка містить функцію (наприклад, об'єкт, який містить метод), оскільки виконуваний код у функції не може бути виконаний через інші програми. (Існує прапор, щоб серіалізувати код позиції функції, але вона може

бути десеріалізована тільки через ту ж саму програму). Стандартні функції серіалізації можуть запобігти розповсюдженню та циклічній обробці даних, які можуть залежати від прапора.

Perl.

Кілька модулів Perl, доступних з CPAN забезпечують механізми серіалізації, включаючи Storable та FreezeThaw. Storable включає в себе функції серіалізації і десеріалізації структур даних Perl з та у файли (чи Perl-скаляри). На додаток до серіалізації безпосередньо до файлів, Storable включає в себе функцію заморожування щоб повернути серіалізовану копію даних, упакованих в скаляр і не десеріалізує такий скаляр. Це використовується для відправки складної структури даних через мережевий сокет або для зберігання в базі даних. При серіалізації структури з Storable, можна використовувати мережеві безпечні функції, які завжди зберігають свої дані у форматі, який доступний для читання на будь-якому комп'ютері за невелику плату швидкості. Ці функції називаються nstore, nfreeze і т.д. Не існує "n" функції для десеріалізації цих структур.

C і C++.

C і C++ не забезпечують прямої підтримки серіалізації. Однак, можна написати свої власні функції серіалізації, оскільки обидві мови підтримують запис двійкових даних. Крім того, рішення на основі компілятора, такі як система ODB ORM для C++, здатні автоматично підтримувати код серіалізації з декількома змінами в оголошенні класу. Інші популярні структури серіалізації – це Boost.Serialization з Boost бібліотеки, бібліотеки S11n та Cereal. Бібліотека MFC (Microsoft) також підтримує методологію серіалізації в рамках своєї Document-View архітектури.

Python.

Ядро загального механізму серіалізації – це стандартний модуль. Він являє собою крос-платформенний формат серіалізації, що легко налаштовується але не захищений від помилкових чи зловмисних даних. Стандартна бібліотека включає в себе модулі серіалізації до стандартних форматів даних: JSON і XML.

Plistlib.

Обмежується підтримкою типів Plist (числа, рядки, логічні значення, кортежі, списки, словники, дата і час та бінарні BLOB).

PHP.

PHP від початку підтримує серіалізацію за допомогою вбудованих функцій `serialize()` і `unserialize()`. PHP вміє серіалізувати будь-який з своїх типів даних, крім ресурсів (файлових покажчиків і т.д.). Вбудована функція `unserialize()` не захищена, коли використовується для абсолютно ненадійних даних. Для серіалізації об'єктів, існують два "магічні методи", які можуть бути імплементовані в класі – `__sleep()` та `__wakeup()` – , які викликаються з функцій `serialize()` і `unserialize()` відповідно, що можуть очистити та відновити об'єкт. Наприклад, при необхідності закрити з'єднання з базою даних за допомогою серіалізації і відновити з'єднання використовуючи десеріалізацію, ця функціональність буде оброблятися в цих двох методах. Вони також дозволяють підібрати такий об'єкт, у якого властивості можуть бути серіалізовані. Починаючи з PHP 5.1, включений об'єктно-орієнтований механізм серіалізації об'єктів: `Serializable` інтерфейс.

REBOL.

REBOL серіалізує дані, записуючи їх у файл (`save / all`) або в рядок (`mold/all`). Рядки та файли можуть бути десеріалізовані за допомогою функції поліморфного навантаження. `RProtoBuf` забезпечує серіалізацію крос-платформених даних в R, використовуючи `protocol buffers`.

Ruby.

Ruby включає в себе стандартний модуль `Marshal` з 2 методами `dump` і `load`, так як і стандартні утиліти Unix `dump` і `restore`. Ці методи серіалізують у стандартний клас `String`, тобто, вони ефективно перетворюються у послідовність байт. Деякі об'єкти не можуть бути серіалізовані (така спроба спровокує виникнення помилки `TypeError`): прив'язки, об'єкти, процедури екземплярів класу IO, об'єктів `singleton` та інтерфейсів. Якщо клас вимагає стандартної серіалізації (наприклад, вимагає певних дій для очищення перед операціями `dumping/restoring`), це може бути зроблено шляхом імплементування двох методів:

`_dump` і `_load`. Метод `_dump` має повернути об'єкт `String`, що містить всю інформацію, необхідну для відновлення об'єктів цього класу і всі пов'язані об'єкти до максимальної глибини даного в якості цілого значення параметру. Метод класу `_load` повинен приймати строку і повертати об'єкт цього класу.

Haskell.

У Haskell, серіалізація підтримується для типів, які є членами класів `Read` і `Show`. Кожен тип, який є членом класу типу `Read` визначає функцію, яка буде отримувати дані зі строкового представлення дампу даних. Клас `Show` у свою чергу, містить функцію `show`, яка може створити об'єкт зі строкового представлення. Програмісту не потрібно визначати функції явно – лише оголосити чи буде тип відправлений на опрацювання до функцій `Read` чи `Show`, або обом. Можна зробити генератор відповідних функцій для декількох сценаріїв (але не всіх: типи функцій, наприклад, не може автоматично передати `Show` або `Read`). Згенерована автоматично сутність для `Show` також дає на виході дійсний код, тому так само значення Haskell можуть бути отримані шляхом виконання коду `show`, як, наприклад, інтерпретатор Haskell. Для більш ефективної серіалізації, є Haskell бібліотеки, які дозволяють виконати високошвидкісну серіалізацію в двійковому форматі.

Windows PowerShell.

Windows PowerShell виконує серіалізацію через вбудований командлет `Export-CLIXML`. Експорт-CLIXML впорядковує об'єкти .NET і зберігає результуючий XML у файлі. Для відтворення об'єктів, використовується Імпорт-CLIXML командлет, який генерує десеріалізований об'єкт з XML в експортований файл. Десеріалізований об'єкти, що часто відомі як "носії конфігурацій" – не живі об'єкти. Це є знімки, які мають властивості, але не методи. Дві тривимірні структури даних також можуть бути (де) серіалізовані у форматі CSV, використовуючи вбудовані в командлети `Import-CSV` і `Export-CSV`.

Julia.

Julia реалізує серіалізацію через `serialize()` / `deserialize()` модулі, призначені для роботи в одній Джулії, та/або сутності одного і того ж образу системи. Пакет `HDF5.jl` пропонує більш стабільну альтернативу, використовуючи документований формат і загальну бібліотеку з обгортками для різних мов, а формати серіалізації за замовчуванням були розроблені для максимальної продуктивності та для роботи в мережі.

2.8 Формати серіалізації

Починаючи з 1987 року, коли компанія Sun Microsystems запропонувала світу ІТ перший формат передачі даних XDR із специфікацією, представленою у [19], нові ефективніші, компактніші та зрозуміліші формати серіалізації почали свій розвиток, який не закінчується по сьогоднішній день. Не дивлячись на те, що більшість розробників використовують стандартні вбудовані інструменти для серіалізації, або такі легко імплементовані формати як XML, JSON чи YAML, існує безліч інших маловідомих та майже не підтримуваних форматів, які мають свої вагомі переваги. Всі існуючі формати поділяються на бінарні, що формують та зберігають об'єкти як бітовий потік даних та текстові (сприйнятні для людини) формати, які легко зчитуються як машинами, так і людьми. Цей розділ має на меті представити короткі теоретичні відомості про кожен з форматів, які будуть досліджені в даній роботі.

2.8.1 Бінарні формати

BSON [bee · Sahn], скорочено від Binary JSON, є двійковим кодом серіалізації JSON-подібних документів. Специфікація BSON наведена у [20]. Як і JSON, BSON підтримує вкладені документи і масиви всередині інших документів і масивів. BSON також містить розширення, які дозволяють працювати з типами даних, що не є частиною специфікації JSON. Наприклад, BSON має тип `Date` і тип `BinData`. BSON можна порівняти з двійковими

форматами обміну, такими як Protocol Buffers. BSON більш зручніший ніж Protocol Buffers, який переважає його в гнучкості, але програє в економії простору (BSON має накладні витрати на імена полів в межах послідовних даних). Формат має такі три переважні характеристики:

- легкий;
- зведення просторових накладних витрати до мінімуму є важливим для будь-якого формату представлення даних, особливо при використанні в мережі. У BSON файлах легко здійснювати пошук по даним, ітеративно проходячи по файлу;
- ефективний. Кодування даних в BSON і декодування з BSON можуть бути виконані дуже швидко і в більшості мов, завдяки використанню типів даних C.

Документи BSON складаються із списків елементів, що відсортовані. Кожен з елементів має певне ім'я поля, значення та тип. Іменами полів є текстові елементи. До типів BSON належать:

- string – рядок;
- int – ціле число;
- double — число с плаваючою точкою подвійної точності;
- DateTime — дата;
- byte[] — масив байтів (бінарні данні);
- bool — булеві (True та False);
- null — «Null» (спеціальне значення);
- BsonObject — BSON-об'єкт;
- BsonObject[] — масив BSON-об'єктів.

Не всі ці типи доступні в JSON, в якому, наприклад, немає масиву з типом «бінарні дані», але через обмеження по довжині деякі дійсні значення JSON (наприклад, дуже довгі рядки) не можуть бути дійсними значеннями BSON. Порівнюючи із JSON, BSON є ефективнішим як у плані розміру зберігання даних, так і швидкості їх сканування.. BSON багато в чому аналогічний до Protocol Buffers - реалізація мовно- і платформи-незалежного формату для

обміну даними, але BSON є більш вільним від схеми даних. Тим самим, більша гнучкість BSON зменшує переваги в продуктивності у випадку, коли схема визначена.

Реалізації BSON існують для різних мов програмування. Деякі реалізації в даний час впроваджені на драйверах MongoDB, тож MongoDB був першим великим проектом, який використовував BSON. З часом ці бібліотеки будуть більш автономні, але вони мають бути доступними, незалежно від MongoDB в їх поточному стані. Більшість з цих бібліотек мають ліцензію Apache 2, але в більшості випадків все залежить від того, хто її написав, так що слід перевіряти деталі для окремого проекту. MongoDB – це база даних, що документо-орієнтована. Вона використовує BSON як представлення документів мережі та на диску. Bsontools - утиліти командного рядка для маніпулювання BSON файлами, і для переведення їх в інший формат, такий як CSV, JSON, XML.

FastInfoset.

FastInfoset - міжнародний стандарт, який визначає двійковий формат кодування для інформаційного набору XML (XML Infoset) як альтернатива XML-документу [21]. Він спрямований на забезпечення більш ефективної серіалізації, ніж текстовий формат XML. Можна вважати, що архівація FI виконується без втрат, як наприклад gzip для XML, в той час як оригінальне форматування має втрати, інформація не втрачається в процесі перетворення з XML в FI і назад, в XML. При цьому, стиснення даних використовується не лише для зменшення їх розміру, а і для оптимізації продуктивності їх обробки. Специфікація FastInfoset визначається МСЕ-Т та органами стандартизації ISO. FI офіційно названий ITU-T Rec. X.891 та ISO/IEC 24824-1 (FastInfoset), відповідно. Тим не менш, його зазвичай називають Fast Infoset. Стандарт був опублікований МСЕ-Т, 14 травня 2005 року, і ISO 4 травня 2007. Стандарт FastInfoset можна завантажити із веб-сайту МСЕ. На нього немає обмежень інтелектуальної власності при його реалізації та використанні. Існує поширена помилка, що FI необхідна підтримка ASN.1 інструменту. Хоча формальна специфікація використовує ASN.1, він використовує власні правила кодування через

кодування управління нотаціями (ECN). В якості альтернативи можна використовувати FleXPath

Основний формат файлу – це ASN.1, з тегом / довжиною / значенням блоків. Текстові значення атрибутів та елементів зберігаються з довжиною префіксів, а не кінцевих обмежувачів, так що немає необхідності уникати спеціальних символів. Еквівалент кінцевих тегів ("термінатори") потрібні тільки в кінці списку дочірніх-елементів, і бінарні дані не мають бути присутні в кодуванні base64. Імена елементів і атрибутів зберігаються в потоці октетів, на відміну від традиційного ASN.1. Це значить, що можна відновити звичайний XML файл із двійкового потоку, без необхідності посилатися на будь-яку XML-схеми. Це не спроба перетворити XML-схеми безпосередньо у визначенні ASN.1. ASN.1 разом з ECN використовується для визначення формату файлу. Індексна таблиця побудована для більшості рядків, який включає в себе імена елементів і атрибутів, а також їх значення. Це означає, що текст повторних тегів і значень з'являється тільки один раз в документі.

Реалізація Java специфікації FI доступна в рамках проекту GlassFish. Бібліотека з відкритим вихідним кодом розповсюджується під умовами ліцензії Apache 2.0. Кілька проектів використовує цю реалізацію, в тому числі еталонну реалізацію для JAX-WS, що використовується в GlassFish.

Під час обробки даних, деякі стискаються як частина процесу генерації XML і вони стають набагато оптимальніші, ніж ті, що отримуються при використанні алгоритмів стиснення індексування потоку XML: вони можуть генерувати кілька великих файлів. Продуктивність типу розбору SAX Fast Infoset також набагато швидша, ніж продуктивність розбору XML 1.0, навіть без врахування стиснення в архів. Кількість даних збільшується в залежності від швидкості розбору, результати досліджень реалізації алгоритму на Java є у 10 раз швидше в порівнянні з Java Xerces, і в 4 рази в порівнянні з драйвером Piccolo (один з найшвидших XML парсерів на Java-основі).

Портативні пристрої та мобільні пристрої, як правило, використовують лінії з низькою пропускною здатністю даних та мають більш повільні процесори.

Цей недолік можна обійти із Fast Infoset, так як він вміє знижувати швидкість передачі даних з обох сторін та обробку даних у декілька разів.

Як тільки застосунок, що використовує серіалізацію запускається, передача інформації через Інтернет стає одним із самих основних вузьких місць пропускну здатності. Якщо відсилаються достатньо великі шматки даних, ця процедура може серйозно відобразитися на продуктивності клієнтських застосунків і обмежити здатність сервера оброблювати запити. Тож, зменшення кількості даних, що передаються через Інтернет може суттєво знизити час, необхідний для того, щоб повідомлення було відправлено або отримано

MessagePack.

MessagePack — це формат обміну даними, призначений для двійкового представлення простих структур даних, таких як масиви і асоціативні масиви. MessagePack намагається бути настільки компактним та зрозумілим, як це можливо. Реалізація цього формату існує для багатьох мов програмування, таких як Python, C++, C# чи C, D, Java, JavaScript, Go, Lua, Haskell, Smalltalk, PHP, OCaml, Perl, Ruby, Scala та інші [22]. Структури даних, які обробляються MessagePack мало відповідають тим, які використовуються в форматі JSON. До підтримуваних типів даних належать наступні типи елементів:

- байтові масиви, що використовуються для представлення строкових або двійкових даних;
- відсутнє значення(null);
- булеві типи, можливі значення яких є false чи true;
- списки чи масиви;
- асоціативні масиви, мапи чи словники;
- цілі числа не більше 64 бітів;
- числа з плаваючою комою - IEEE одинарної і подвійної точності.

MessagePack більш компактний, ніж JSON, але має обмеження на розмір цілих чисел та масивів. З іншого боку, формат підтримує серіалізацію бінарних даних та строки, кодування яких відмінне від UTF-8. У порівнянні із BSON, MessagePack компактніше архівує дані. Наприклад BSON вимагає, щоб усі

строкові елементи закінчувались нульовим байтом, а також, запису строкових індексів для списків чи масивів, а MessagePack – ні. Також, MessagePack забезпечує щільніше представлення коротких списків, невеликих цілих чисел, та асоціативних масивів.

Бібліотека повністю сумісна з JSON, але все ж існують деякі обмеження:

- значення Integer обмежене від $-(2^{63})$ до $(2^{64}) - 1$;
- значення Float представлено стандартом IEEE 754 з одинарною або подвійною точністю;
- максимальна довжина бінарного об'єкта $(2^{32}) - 1$;
- максимальний розмір байтового рядка $(2^{32}) - 1$;
- рядок може містити неприпустиму послідовність байтів. Для такої ситуації поведінка десеріалізатора залежить від фактичної реалізації, коли він отримає неприпустиму послідовність байтів. З цього випливає, що в кожній з мов він матиме свою поведінку. Наприклад, у випадку, якщо ви розпакуєте дані на PHP буде автоматично згенеровано: Warning: [msgpack] (php_msgpack_unserialize)% s% d в ваш STDERR. Але цю функцію можна вимкнути використовуючи php.ini через прапор `msgpack.error_display = 0` (за замовчуванням 1);
- максимальна кількість елементів у масиві може бути не більше ніж $(2^{32}) - 1$;
- Максимальна кількість елементів в асоціативному масиві (ключ - значення) не більше ніж $(2^{32}) - 1$;

2.8.2 Текстові формати

XML.

XML – мова розмітки, що рекомендована Консорціумом Всесвітньої павутини (W3C) [23], яка з'явилася у зв'язку з необхідністю пристосувати SGML до мережевого середовища. Основна увага в XML зосереджено на даних. У XML проводиться суворі лінії розподілу між структурною розміткою даних і представленням даних. Специфікація XML частково описує поведінку XML-

процесорів (програм, які зчитують XML-документи та надають доступ до їх вмісту) та власне самі XML документи. XML розроблявся у якості мови з простим формальним синтаксисом, зручної для створення та обробки даних програмами та одночасного зчитування і створення документів людиною, з підкресленням на можливості використання в мережі. Мову називають розширюваною, оскільки вона не фіксує розмітку, що використовується при формуванні документа, а надає розробнику можливість вільно створювати розмітку відповідно до потреб конкретної області, хоча обмеження синтаксичними правилами мови залишаються. Поєднання простого синтаксису, що сприйнятний для людини, розширюваності, а також підтримка кодувань Юнікод-символів при поданні змісту документів призвело до дуже широкого використання як XML, так і безлічі похідних спеціалізованих мов на його основі.

З фізичної точки зору документ складається з сутностей, з яких кожна може посилатись на іншу. Єдиний кореневий елемент - документальна сутність. Зміст сутностей - символи. Всі вони розділені на два типи: символні дані і символи розмітки. До символів розмітки належать теги, що визначають межі елементів, інструкції та оголошення обробки, включаючи їх атрибути, посилання на сутності, коментарі, а також послідовності символів, що обрамляють секції «CDATA».

З логічної точки зору документ складається з коментарів, оголошень, елементів, посилань на сутності та інструкцій обробки. Всі ці сутності в документі структурували розміткою. Всі складові частини документа узагальнюються в пролог і кореневий елемент.

Кореневий елемент – головна частина документа, складова всієї його суті. Він може включати (а може не включати) інші елементи та символні дані, а також коментарі. Елементи, що є вкладеними в кореневий елемент, в свою чергу, можуть включати наступні елементи, символні дані і коментарі, і так далі. Існує так званий пролог, який може включати оголошення, інструкції обробки та коментарі. Починати його варто з оголошення XML, хоча в деяких випадках допускається відсутність цього оголошення. Символьні дані можуть

зустрічатися всередині елементів як безпосередньо так і в спеціальних секціях «CDATA». Елементи повинні бути вкладені правильно: будь-який елемент, що починається всередині іншого елемента (тобто будь-який елемент документа, крім кореневого), повинен закінчуватися всередині того елемента, в якому він почався. Атрибути використовуються для зв'язки з логічною одиницею текстових пар ім'я-значення. Розмітка завжди починається символом «<» і закінчується символом «>». Кутові дужки позначають кордони елементів, інструкцій обробки та деяких інших послідовностей. Поряд із цими символами, спеціальну роль відіграє також символ «&». Амперсанд дозволяє виконати заміну тексту за допомогою сутностей. Вживання розмічувальних символів в символічних даних може створити проблему неоднозначності структури. У XML ця проблема вирішується наступним чином: три згаданих символи не можуть бути присутніми в символічних даних і в значеннях атрибутів в їх безпосередньому вигляді, для їх подання в цих випадках використовують спеціальні сутності. Для вживання апострофів і лапок усередині значень атрибутів використовуються наступні сутності: '& Apos;' & Quot. За правилами заміни символів, що використовуються в розмітці, ними позначаються сутності, які не входять до символічних даних в секціях «CDATA», проте дозволено у всіх інших місцях документа. Як і мова розмітки Web-документів, XML володіє наступними перевагами:

- гнучкість. XML дозволяє обробляти унікальні дані в комерційній та медичній областях. Незалежно від характеру даних XML здатний забезпечити адекватними методами їх зберігання та обробки. Виняток становлять лише двійкові дані і впроваджені сценарії;
- можливість налаштування. XML успадкував синтаксичну цілісність і сувору структуру свого батька – SGML (рисунок 2.8).

Одним з найбільш потужних інтерфейсів доступу до вмісту XML документів є Document Object Model - DOM.

Об'єктна модель XML документів є поданням його внутрішньої структури у вигляді сукупності певних об'єктів. Для зручності ці об'єкти організуються в

деяку ієрархічну структуру даних у вигляді дерева - кожен елемент документа може бути віднесений до окремої гілки, а весь його вміст у вигляді набору вкладених елементів, коментарів, секцій CDATA представляється в цій структурі піддеревами. В будь-якому правильно складеному XML-документі обов'язково визначено головний елемент, тож весь вміст можна розглядати як піддерева цього основного елемента, званого в такому випадку коренем дерева документа. Для наступного фрагмента XML документа наведеному на рисунку 2.9.

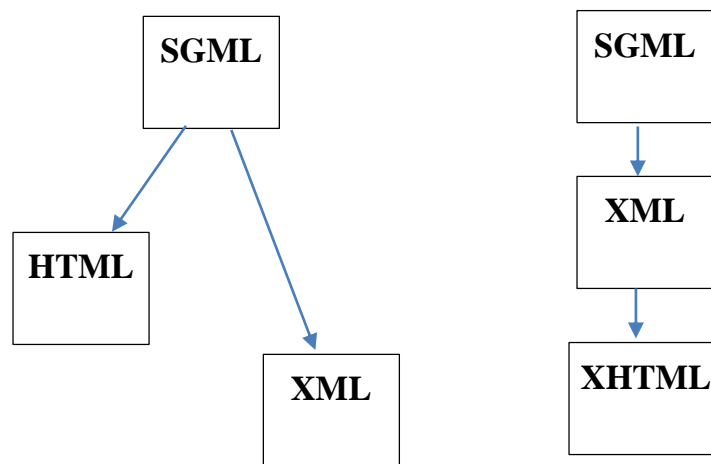


Рисунок 2.8 – Першочергове та поточне співвідношення між XML, HTML та SGML

```
<tree-node>
  <node-level1>
    <node-level2/>
    <node-level2>text</node-level2>
    <node-level2/>
  </node-level1>
  <node-level1>
    <node-level2>text</node-level2>
  <node-level1>
    <node-level2/>
    <node-level2><node-level3/></node-level2>
  </node-level1>
</tree-node>
```

Рисунок 2.9 – Фрагмент XML документа

Приклад дерева елементів наведено на рисунку 2.10.

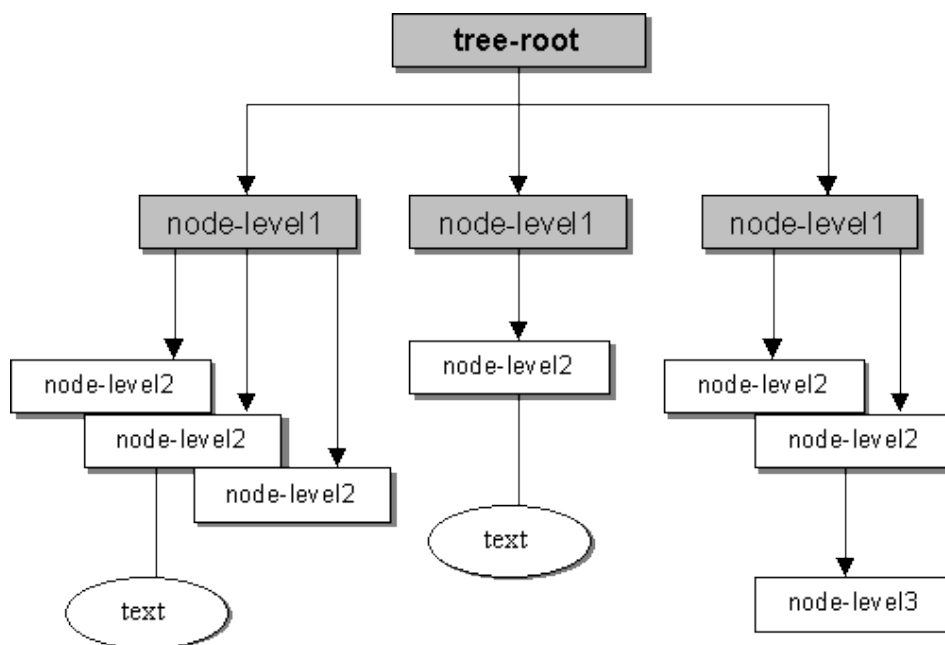


Рисунок 2.10 – Дерево елементів XML документу JSON.

JSON — формат обміну даними, що заснований на JavaScript та зазвичай використовується у парі саме з цією мовою. Як і інші текстові формати, JSON легко сприймається людьми. Формат JSON був розроблений Дугласом Крокфордом. [24] Не дивлячись на походження від JavaScript (точніше, від підмножини мови стандарту ECMA-262 1999 року), формат може використовуватися практично з будь-якою мовою програмування, так як вважається мовою незалежним. Для багатьох мов програмування існує готовий код для створення та обробки даних у форматі JSON. JSON підтримує серіалізацію наступних даних

- колекція пар ключ / значення. У різних мовах, ця концепція реалізована як об'єкт, запис, структура, словник, хеш, іменованний список або асоціативний масив.
- впорядкований перелік значень. У більшості мов такий перелік реалізовано як масив, вектор або послідовність..

Такі універсальні типи даних підтримують майже всі сучасні мови програмування у будь-якій формі. Логічно припустити, що формат даних,

незалежний від мови програмування, має бути заснований на цих структурах. У нотації JSON це виглядає наступним чином.

Об'єкт (рисунок 2.11) - неупорядкований набір пар ключ / значення. Об'єкт починається з «{» і закінчується «}». Кожне ім'я супроводжується символом «:», пари ключ / значення розділяються «,».

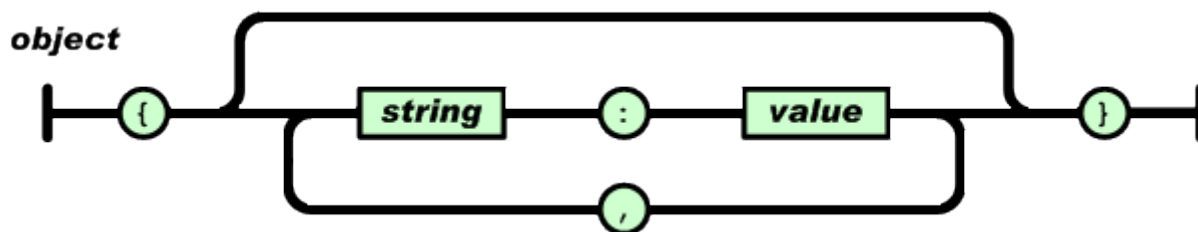


Рисунок 2.11 – Модель JSON об'єкту

Масив - упорядкована колекція значень, рисунок 2.12. Масив починається з «[» і закінчується «]». Значення розділені «,». Значення може бути рядком у подвійних лапках, числом, «true», «false», «null», об'єктом або масивом. Ці структури можуть бути вкладеними.

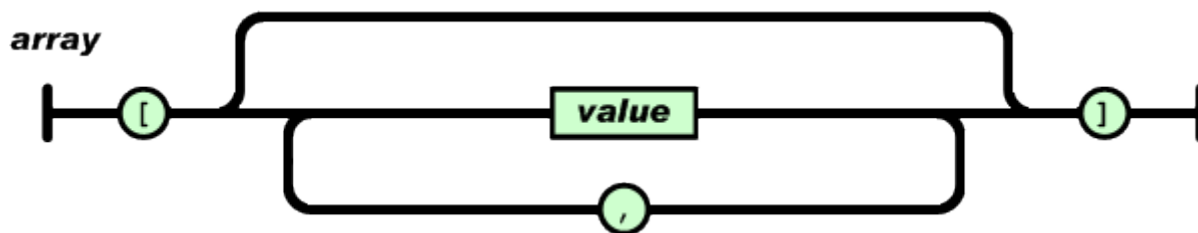


Рисунок 2.12 – Модель JSON масиву

Рядок – це колекція нуля або більше символів Unicode, укладена в подвійні лапки, використовуючи «\» в якості символу екранування. Символ представляється як односимвольний рядок. Схожий синтаксис використовується в C і Java. Модель JSON строки наведено на рисунку 2.13

Число представляється так само, як в С або Java, крім того, що використовується тільки десяткова система числення. Модель JSON числа наведено на рисунку 2.14

Пробіли можуть використовуватися між будь-якими лексемами.

Виключаючи деякі деталі кодування, вище сказане повністю описує мову.

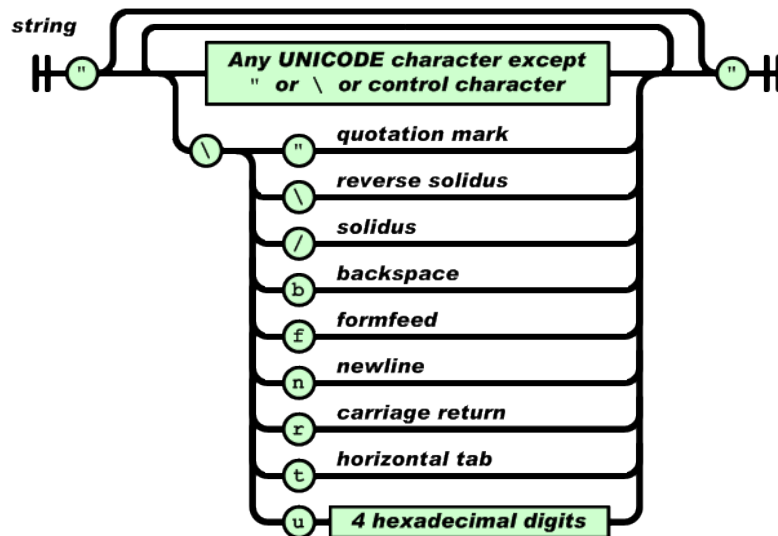


Рисунок 2.13 – Модель JSON строки

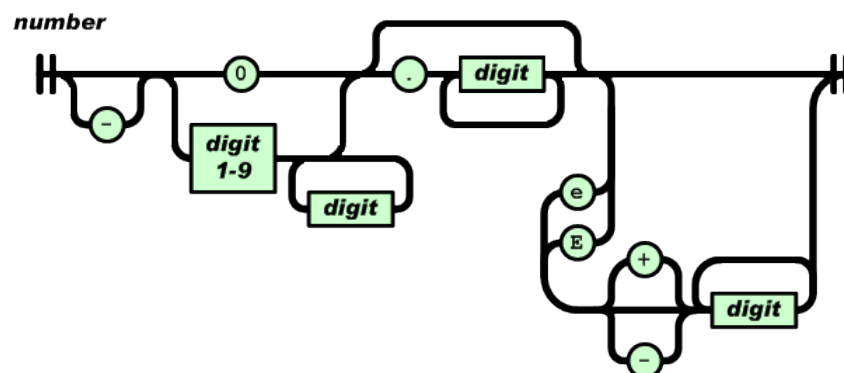


Рисунок 2.14 – Модель JSON числа

YAML.

YAML – сприйнятний для людини формат серіалізації даних, орієнтований на зручність введення-виведення типових структур даних багатьох мов

програмування, хоча залишається концептуально близьким до мов розмітки. Назва YAML створена як рекурсивний акронім `YAML Is not Markup Language` («YAML - не мова розмітки»). Раніше мова називалася `Yet Another Markup Language` («Ще одна мова розмітки») і навіть розглядався як конкурент XML, але пізніше був перейменований з метою акцентувати увагу на даних, а не на розмітці документів. Згідно меті розробки, озвученим Кларком Евансом, `YAML 1.0` покликаний :

- бути крос-платформеним між мовами програмування;
- бути легко зрозумілим людині;
- підтримувати потокову обробку;
- бути легким у реалізації та використанні.
- підтримувати структури даних, нативні для мов програмування;
- бути виразним і розширюваним;
- використовувати цільну модель даних для підтримки звичайного інструментарію.

У порівнянні із XML синтаксисом, синтаксис `YAML` мінімалістичний. У специфікації вказано, що великий вплив на прогрес `YAML` зробив стандарт RFC 822. `YAML` в більшості випадків використовується як формат для створення файлів конфігурації. Застосовується для налаштування веб-каркасів `Ruby on Rails`, `Dancer`, `Symfony`, `GAE framework`, `Google App Engine` і `Dart`. Також є основною мовою опису класів, ресурсів і маніфестів для пакетів застосунків `OpenStack Murano Project`.

Говорячи про відмінності `YAML` від XML, можна згадати, що XML-елементи використовуються для відображення довільних структур, проте `YAML` більш близький до відображення типових моделей даних з мов `Java`, `Python`, `Perl`. Він дозволяє описувати будь-які сполучення послідовностей, скалярних типів та зіставлень – тобто мова є ближчою до реальних структур даних мов програмування, і не вимагає дотримання різних специфікацій про DOM-відображення структур даних, як потрібно в XML. Надалі наведено невеликий список основних елементів `YAML`:

- потоки YAML використовують для друку Unicode-символи, як UTF-8, так і UTF-16
- відступи з пробілів (символи табуляції не допускаються) використовуються для позначення структури
- коментарі починаються з символу "#", можуть починатися в будь-якому місці рядка і тривають до кінця рядка
- списки позначаються початковим дефісом «-» з одним членом списку на рядок, або члени списку кладуться у квадратні дужки «[]» і розділяються комою і пробілом «,»
- асоціативні масиви представлені двокрапкою з пробілом «:» у вигляді ключ: значення, по одній парі ключ-значення на рядок, або у вигляді пар, укладених у фігурні дужки і розділених комою і пробілом «,»
- ключ в асоціативному масиві може мати як префікс знак питання «?», що дозволяє вказати складний ключ, наприклад представлений у вигляді списку
- рядки записуються без лапок, однак можуть бути укладені в одинарні або подвійні лапки
- всередині подвійних лапок можуть бути використані екрановані символи в C-стилі, що починаються зі зворотного слеша «\»
- YAML дозволяє задавати підстановки за допомогою амперсантів «&» і аліасів «*».
- явне завдання типу оформляється шляхом '!! [вказівку типу]'. Приклад, !! str 100 після парсинга видасть значення "100"
- значення типу Дата/Час задаються у форматі YYYY-MM-DD або YYYY-MM-DD HH: MM: SS. Якщо необхідно задати дату, як рядок, потрібно укласти її в лапки ("2012-12-21").

Не дивлячись на все, YAML має ряд недоліків. Один із найважливіших - проблеми з кодуваннями, відмінними від Unicode. Специфікація передбачає підтримку лише UTF-8, UTF-16 LE і UTF-16 BE. Як результат - робота з іншими кодуваннями повністю залежить від модуля, що використовується. Так, модуль

PyYAML, де використовується KOI8-R, і в Windows XP (CP1251) вперто не бажав обробляти файли, що містять хоча б один кириличний символ незалежно від його місця розташування. З модулем 'Yaml' в Ruby ніде проблем не виникло. В Ubuntu, де за замовчуванням використовується UTF8, всі протестовані модулі показали бездоганну роботу з кирилицею.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ ЗБЕРЕЖЕННЯ СТАНУ ОБ'ЄКТІВ

Як було сказано на самому початку роботи, метою власне самої магістерської дисертації є створення програмного засобу збереження інформаційних об'єктів побудованого з обґрунтованим використанням методів серіалізації даних. У попередньому розділі були наведені короткі теоретичні відомості по найпопулярнішим та найпоширенішим форматам серіалізації даних. Проте, для того, щоб проводити сам процес вибору та оптимізації, необхідно не тільки докладніше ознайомитись із кожним з форматів серіалізації, але й виділити основні характеристики, що являтимуться показниками ефективності процесу серіалізації, провести порівняльну характеристику, визначивши слабкі та сильні сторони кожного із форматів, визначити мету створення оптимізованого формату після спостереження та висновків коли і де варто використовувати конкретний формат серіалізації.

Не менш важливим та необхідним етапом дослідження є етап формування вихідних даних, на яких буде базуватись експеримент. Основними критеріями має бути незмінність даних для того, щоб правильно визначати різницю отриманих результатів в залежності від змін характеристик формату чи характеру вимірювання, а не зміни типу та обсягу даних. Також, тестові дані мають покривати більшу частину існуючих типів та можливих сценаріїв задля того, щоб прослідкувати особливості кожного з форматів на різних типах даних.

Один з перших та важливіших етапів є власне платформа, на якій відбуватиметься подальша розробка та проведення дослідження. Цьому етапу слід приділити чимало уваги, так як мова або платформа для програмування матиме вплив на вибір бібліотек, що використовуються для імплементації того чи іншого формату, а також продуктивність виконання тих чи інших методів, архітектуру проекту.

3.1 Попереднє проектування системи

В роботі під системою обробки даних мається на увазі веб-сервіс, який призначений для збору даних з навігаційних пристроїв та пристроїв фото та відеофіксації, що являються зовнішніми для системи, їх автоматичної обробки, побудови статистики та відображення користувачам у простому та зрозумілому вигляді.

В наступних підрозділах будуть розглянуті питання попереднього проектування сервісу, а саме: питання вибору архітектури, розробки структури, вибір та обґрунтування технологій і компонентів, які будуть використані при розробці сервісу.

3.1.1 Розробка архітектури та структури програми

Архітектура програмного забезпечення — це структура програми або обчислювальної системи, яка містить програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними.

Проектування архітектури програмного забезпечення — це процес розроблення, що виконується після етапу аналізу і формування вимог. Завдання такого проектування — перетворення вимог до системи у вимоги до програмного забезпечення і побудова на їхній основі архітектури системи.

Умовно система складається з трьох модулів:

- модуль взаємодії з зовнішніми (навігаційними, фото, відео) пристроями;
- модуль обробки та збереження даних;
- модуль взаємодії з користувачем.

Модуль взаємодії з зовнішніми пристроями приймає дані від навігаційних пристроїв, що зареєстровані в системі та передає їх далі до модулю обробки та збереження даних, який зберігає отримані дані в БД, будує статистику та передає на наступний рівень до модуля взаємодії з користувачем, який відповідає за вивід

даних в належному для користувача вигляді. Модуль взаємодії з користувачем також відсилає двом іншим модулям запити на виконання тих чи інших операцій. Взаємодія між модулями системи зображена на рисунку 3.1.

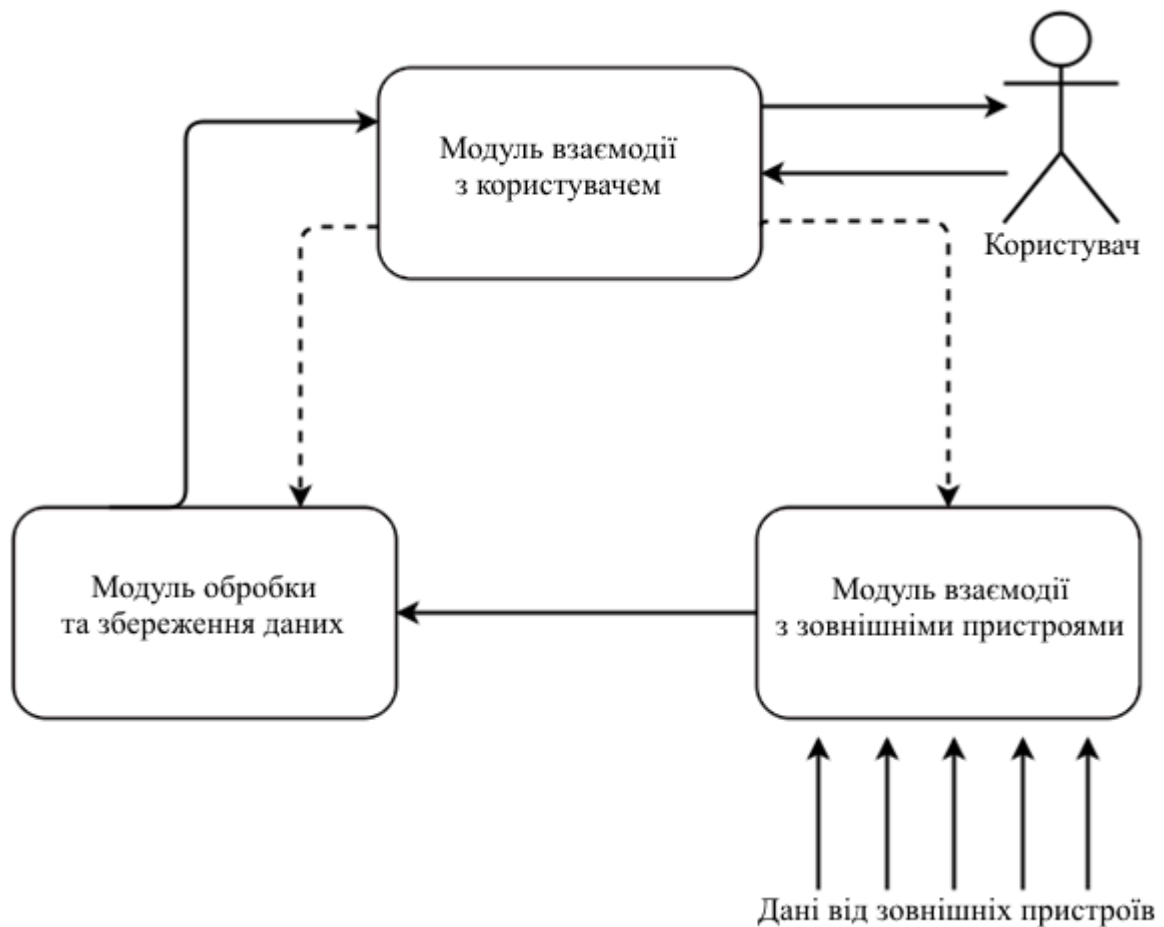


Рисунок 3.1 – Діаграма взаємодії між модулями системи

3.1.2 Модуль взаємодії з зовнішніми пристроями

Модуль має наступні функції:

- отримання геопозиційних даних від пристроїв;
- отримання фото- та відеоданих від пристроїв;
- перевірка отриманих даних на достовірність;
- передача отриманих і перевірених даних до модуля обробки та збереження.

Сучасні автомобільні GPS-навігатори мають встановлену ОС Android або Windows CE та обладнані GPRS-модемом, що дозволяє передавати дані через

глобальну мережу Internet. Сучасні смартфони обладнані GPS-приймачами та навігаційними програмами, мають встановлену операційну систему систему Android / iOS / Windows Mobile, і теж дозволяють передавати дані через мережу Internet. Отже, доцільно передавати геопозиційні дані з навігаційних пристроїв в систему через мережу Internet.

Передавати дані по мережі можна в бінарному або текстовому форматі. Значна перевага бінарних даних — малий розмір порівняно з текстовими. Перевага текстового формату перед бінарним - простота їх генерації, обробки та передачі. Враховуючи те, що текстові формати даних значно більш інтерперабельні, ніж бінарні, а також те, що з плином часу швидкість доступу до мережі Internet зростає, а вартість падає, для передачі геопозиційних даних доцільно використовувати текстові формати.

Оскільки система обробки геопозиційних даних розробляється як веб-застосування і дані від пристроїв передаються в JSON-форматі, найпростішим й одночасно безпечним та зручним протоколом для передачі даних буде протокол HTTP (HTTPS). Згідно з дослідженнями в розділі 4, формат JSON є оптимальним для збереження даних невеликого об'єму, і тому вибраний як формат, у якому будуть зберігатися геопозиційні дані в побудованій системі.

Що стосується фото- та відеоданих, то згідно з отриманими в розділі 4 дослідженнями, найкращим форматом зберігання серіалізованих даних виявився MessagePack.

Таким чином, розробка модуля взаємодії з зовнішніми пристроями зводиться до написання обробника HTTP-запитів, який буде приймати дані в форматі, визначеному відповідно до досліджень розділу 4, перевіряти правильність прийнятих даних та передавати їх далі модулю обробки та збереження.

3.1.3 Модуль обробки та збереження даних

Модуль має наступні функції:

- отримання даних від модуля взаємодії з зовнішніми пристроями;
- збереження даних в БД;
- побудова та збереження статистики.

Даний модуль не має архітектурних особливостей, його детальна розробка розглянута в наступних розділах.

3.1.4 Модуль взаємодії з користувачем

Даний модуль має наступний функціонал:

- авторизація та аутентифікація користувачів;
- надання інтерфейсу користувача;
- представлення оброблених даних користувачам у належному вигляді.

Модуль взаємодії з користувачем реалізовано як ВЕБ-застосування. При її побудові використано високорівневий архітектурний шаблон MVC (Модель-вид-контролер, англ. Model-view-controller).

Цей шаблон поділяє систему на три частини: модель даних, представлення даних та керування. Шаблон застосовується для відокремлення даних від представлення так, щоб модифікація одного з компонентів мінімально впливала на інші компоненти.

Основна мета застосування цієї концепції полягає в відділенні бізнес-логіки (моделі) від її візуалізації (уявлення, виду). За рахунок такого поділу підвищується можливість повторного використання. Найбільш корисне застосування даної концепції в тих випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах та / або з різних точок зору.

Концепція MVC дозволяє розділити дані, подання та обробку дій користувача на три окремих компонента:

- Модель (англ. Model). Модель надає знання: дані і методи роботи з цими даними, реагує на запити, змінюючи свій стан. Не містить інформації, як ці знання можна візуалізувати.

- Подання, вид (англ. View). Відповідає за відображення інформації (візуалізацію). Часто як уявлення виступає форма (вікно) з графічними елементами.
- Контролер (англ. Controller). Забезпечує зв'язок між користувачем і системою: контролює введення даних користувачем і використовує модель та подання для реалізації необхідної реакції.

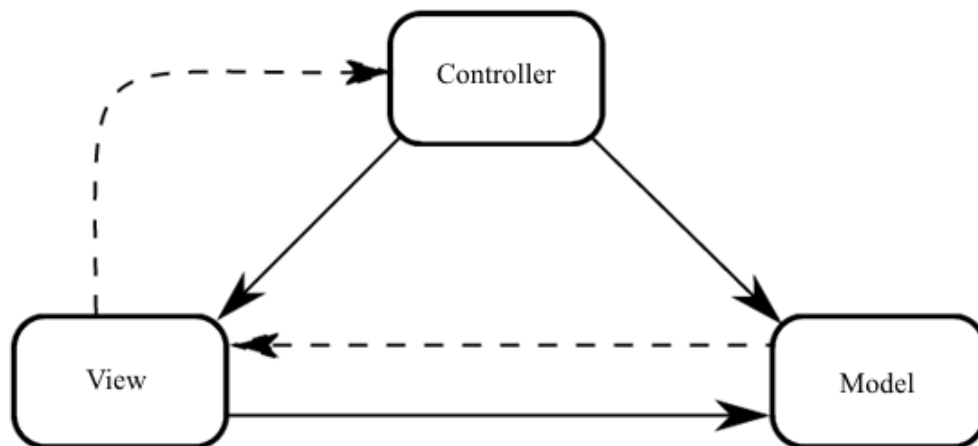


Рисунок 3.2 – Діаграма взаємодії між компонентами шаблону MVC

3.2 Вибір мови програмування

Java – це об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (в подальшому придбаній компанією Oracle). Головна особливість програм Java – це їх трансляція в спеціальний байт-код, тому вони можуть працювати на будь-якій віртуальній Java-машині незалежно від комп'ютерної архітектури, тобто являться крос-платформеними. Java являє собою мову програмування і платформу обчислень, яка була вперше випущена в 1995 році. Існує безліч програм і веб-сайтів, які не працюють за відсутності встановленої Java, і з кожним днем число таких веб-сайтів і застосунків збільшується. Java відрізняється швидкістю, високим рівнем захищеності та надійності. Від портативних комп'ютерів до центрів даних, від ігрових консолей до комп'ютерів, які використовуються для наукових розробок, від сотових телефонів до мережі Інтернет – Java є всюди. Перевагою способу виконання java

програм є повна незалежність байт-коду від платформної операційної системи і обладнання, що підтримує роботу Java-застосунків на будь-якому пристрої, для якого існує відповідна віртуальна машина. Наступною важливою особливістю Java є гнучкість системи безпеки, в контексті якої процес виконання програми повністю контролюється віртуальною машиною. Будь-які операції, що виходять за рамки встановлених повноважень програми (наприклад, спроба несанкціонованого доступу до даних або під'єднання до іншого комп'ютера), викликають негайне переривання виконання застосунку.

C # і Java - дві мови програмування, які багато в чому успадкували синтаксис C ++, і створені в постійних умовах конкуренції. Внаслідок цього мови володіють певною схожістю, але мають і ряд відмінностей. На цій арені, можливо, дві мови найбільш близькі до того, щоб вважатися конкурентами. Java з її платформою J2EE (Java (2) Enterprise Edition) і C# з його ASP.NET змагаються в області створення динамічного веб-контенту і застосунків. Порівняння найпопулярніших технологій за 2018 рік наведено на рисунку 3.3.

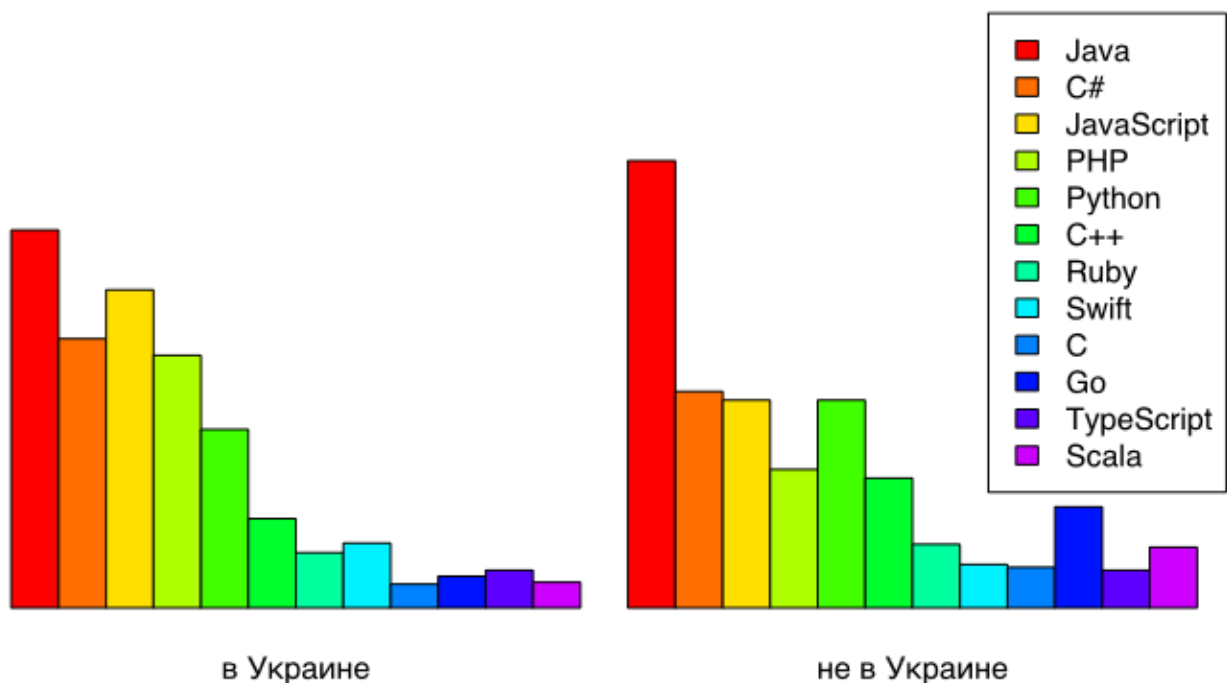


Рисунок 3.3 – Використання технологій за 2018 рік [26]

На цьому ринку широко використовуються і підтримуються обидві мови, разом з комплектом інструментів та супроводжуючих продуктів, наявних для JavaEE і .NET.

Рисунок 3.4 ілюструє процентне співвідношення використання java застосунків на різних платформах. З моменту появи C# він постійно порівнюється з Java. Неможливо заперечувати, що C# і його керована середа CLR багатьом зобов'язані Java і її JRE (Java Runtime Environment).

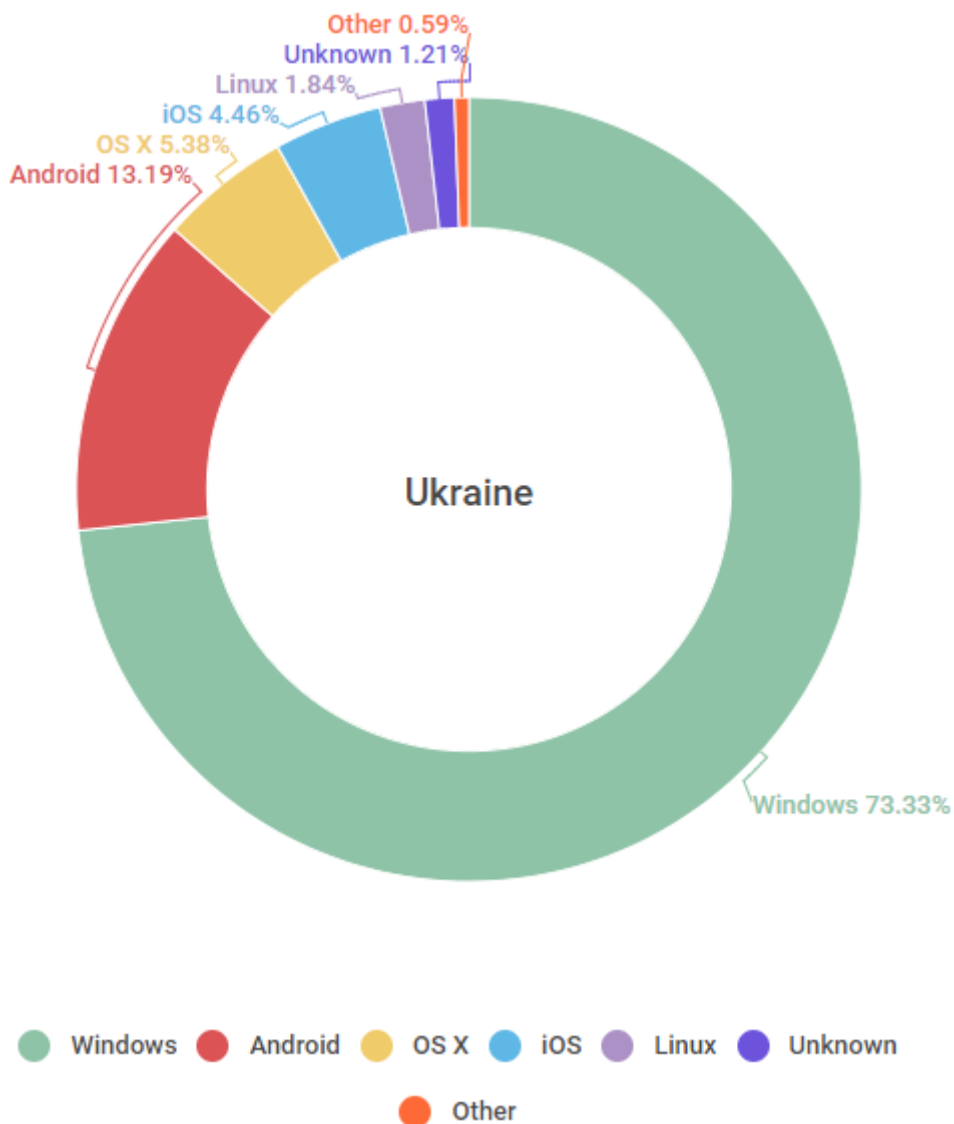


Рисунок 3.4 – Використання операційних систем [27]

Можна сперечатися, чи є розробка C# в якійсь мірі результатом визнання Майкрософтом того, що середовище керованого коду, де лідирує Java, має безліч переваг в зростаючому мережевому світі, особливо при появі Інтернету на

пристроях, відмінних від персональних комп'ютерів, і при зростаючій важливості мережевої безпеки. До створення C# Microsoft модифікувала Java (створивши J++), з тим щоб додати можливості, що працюють тільки на ОС Windows, порушивши таким чином ліцензійну угоду Sun Microsystems. Поки Microsoft перебувала на другій фазі своєї бізнес-стратегії, відомої як «Embrace, Extend, and Extinguish», розвиток J++ було зупинено позовом, поданим Sun. Будучи позбавленою можливості розробляти клон Java з потрібними їй властивостями, Microsoft створила альтернативу, яка більше відповідала їхнім потребам і баченню майбутнього. Незважаючи на такий початок, стає все більш очевидним, що дві мови рідко конкурують один з одним на ринку. Java домінує в мобільному секторі і має багато прихильників на ринку веб-застосунків. C# отримав гарну визнання на ринку настільних застосунків Windows і завдяки ASP.NET, C# також є гравцем і на ринку веб-застосунків.

Проте, головною особливістю java була і залишається наявність JVM середовища. Так як java – мова веб-застосунків, існує чимало так званих «движків» та драйверів, що вміють транслювати java код на клієнті у JavaScript, фреймворки, що дозволяють реалізувати гнучку та високошвидкісну клієнт-серверну архітектуру. Виходячи з власного досвіду, більшість застосунків, створені мовою Java мають клієнт-серверну архітектуру із доступом до мережі. В той час, як десктопні застосунки мовою C# зазвичай працюють із локальною мережею чи взагалі дампами баз розгорнутими на тому ж самому комп'ютері, де виконується програма.

Щодо серіалізації даних, C++ надає змогу проводити точні дослідження характеристик через наявність можливості контролювати роботу з пам'яттю, в той час як JVM такої можливості не дає. Проте, мовою java можна реалізувати більшість існуючих та всі згадані формати серіалізації за допомогою як рідних платформ бібліотек, так і розробленими третіми особами.

Ще одна мова програмування – інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою Python.

Серед її переваг можна назвати такі:

- чистий синтаксис;
- висока швидкість розробки;
- велика кількість бібліотек (в тому числі веб-фреймворків та бібліотек для роботи з даними різного типу та серіалізацією даних).

Python — високорівнева мова програмування з простим синтаксисом, орієнтована на швидке написання коду та підвищення продуктивності розробки.

Python підтримує кілька парадигм програмування, в тому числі структурну, об'єктно-орієнтовану, функціональну, імперативну і аспектно-орієнтовану. Основні архітектурні риси — динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточних обчислень і зручні високорівневі структури даних. Код в Пітоні організовується у функції та класи, які можуть об'єднуватися в модулі (вони в свою чергу можуть бути об'єднані в пакети) [28].

Окрім стандартної бібліотеки існує велика кількість бібліотек, які надають інтерфейс до всіх системних викликів на різних платформах. Для Пітона написано багато ORM (SQLObject, SQLAlchemy, Dejavu, Django), виконані програмні каркаси для розробки веб-додатків (Django, Pylons, Pyramid), а також існує інтерфейс шлюзу з веб-сервером - WSGI (Web Server Gateway Interface) [28].

Еталонною реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Він розповсюджується під вільною ліцензією Python Software Foundation License, що дозволяє використовувати його без обмежень у будь-яких застосуваннях, включаючи пропрієтарні [29].

Виходячи з того, що більшість форматів серіалізації, що імплементовані у Python, текстові і мають ряд недоліків, було прийняте рішення використовувати для досліджень та розробки саме платформу Python.

3.3 Вибір веб-фреймворку

Веб-фреймворк — програмне забезпечення, призначене для розробки динамічних веб-сайтів, застосування, сервісів. Головна мета веб-фреймворку — полегшити процес веб-розробки. Деякі веб-фреймворки для Python надають бібліотеки доступу до бази даних (ORM — object-relational mapping — об'єктно-реляційне відображення).

Власне, для мови Python існує велика кількість веб-фреймворків (Pylons, Pyramid, Bottle.py, Django, Zope та багато інших). Більшість з них реалізують MVC шаблон, та дозволяють користуватися сторонніми механізмами шаблонізації та об'єктно-реляційного відображення.

Архітектура Django подібна архітектурі MVC з тією відмінністю, що представлення називається шаблоном (Template), а контроллер — представленням (View). Таким чином, архітектуру прийнято називати MTV (Модель Шаблон Вид).

Початкова розробка Django, як засобу для роботи новинних ресурсів, досить сильно позначилася на його архітектурі: він надає ряд засобів, які допомагають у швидкій розробці веб-сайтів інформаційного характеру. Так, наприклад, розробнику не потрібно створювати контролери та сторінки для адміністративної частини сайту, в Django є вбудований модуль для керування вмістом, який можна включити в будь-який сайт, зроблений на Django, і який може керувати відразу декількома сайтами на одному сервері. Адміністративний модуль дозволяє створювати, змінювати і вилучати будь-які об'єкти наповнення сайту, протоколюючи всі дії, а також надає інтерфейс для управління користувачами і групами (з призначенням прав) [30].

У збірку Django включені модулі для підтримки коментарів, управління «статичними сторінками» (якими можна управляти без необхідності розробляти окремі відображення), редіректу URL та інші.

Також, на відміну від багатьох інших фреймворків, Django надає власний шаблонізатор та ORM, які погоджено працюють, хоч це і не виключає можливість користуватися сторонніми засобами. Це також прискорює процес розробки і при цьому не накладає додаткових обмежень на архітектуру.

3.4 Вибір БД та СУБД

База даних (скорочено — БД) — впорядкований набір логічно взаємопов'язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів. У технічному розумінні включно й система керування БД.

Головне завдання БД — гарантоване збереження значних обсягів інформації (так звані записи даних) та надання доступу до неї користувачеві або ж прикладній програмі. Таким чином, БД складається з двох частин: збереженої інформації та системи керування нею.

Система керування базами даних (СУБД) — комп'ютерна програма чи комплекс програм, що забезпечує користувачам можливість створення, збереження, оновлення, пошук інформації та контролю доступу в базах даних.

Найпоширенішими сьогодні є реляційні бази даних. Реляційна система керування базами даних (РСУБД; інакше Система керування реляційними базами даних, СКРБД) — СУБД, що керує реляційними базами даних. Поняття реляційний (англ. relation — відношення) пов'язане з розробками відомого англійського спеціаліста в області систем баз даних Едгара Кодда (Edgar Codd).

Ця модель характеризується простотою структури даних, зручним для користувача табличним представленням і можливістю використання формального апарату алгебри відношень і реляційного обчислення для обробки даних [31].

Реляційна модель орієнтована на організацію у вигляді двовимірних таблиць. Кожна реляційна таблиця являє собою двовимірний масив і має такі властивості:

- кожний елемент таблиці — один елемент даних;
- всі комірки в стовпці таблиці однорідні, тобто всі елементи в стовпці мають однаковий тип;
- кожний стовпець має унікальне ім'я;
- однакові рядки в таблиці відсутні;

– порядок наступності рядків і стовпців може бути довільним.

Основна перевага реляційних БД полягає в тому, що вони створені на основі розвинутого математичного апарату, який дозволяє досить лаконічно описати основні операції над даними. За допомогою операцій реляційної алгебри маніпуляція даними в реляційній БД досить гнучка і легка. Це дозволяє об'єднувати таблиці та здійснювати пошук даних по багатьом умовам та різним полям. Власне, для маніпуляцій даними в РБД існує мова SQL (англ. Structured query language — мова структурованих запитів) — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікації, системи контролю за доступом до бази даних. Отже реляційні СУБД разом з мовою SQL надають найширші можливості для маніпуляції даними, що представлені у вигляді таблиць.

Важливий недолік реляційних баз даних - мала швидкість доступу до даних.

В протипагу реляційним базам існують бази даних типу “Ключ - значення”. Такі бази даних в силу своєї простоти мають найвищу швидкість доступу до даних та високий рівень масштабованості, але в той же час, через свою простоту не дозволяють маніпулювати даними на стільки ж гнучко, як реляційні БД. Найчастіше пошук даних можна здійснювати лише по ключу, а більшість можливостей таких БД зводяться до того, щоб записати дані (ключ: значення) в базу та отримати значення, користуючись ключем.

Посередині між зручними та функціональними СУБД та швидкими і масштабованими базами даних типу “ключ-значення” знаходяться деякі документо-орієнтовані СУБД.

MongoDB — документо-орієнтована система керування базами даних з відкритим сирцевим кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СУБД, функціональними і зручними у формуванні запитів [32].

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій.

У MongoDB є вбудовані засоби із забезпечення шардінга (розподіл набору даних по серверах на основі певного ключа), комбінуючи який реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера в кластер проводиться без зупинки роботи БД простим додаванням нових машин.

Основні можливості MongoDB:

- документо-орієнтоване сховище (проста та потужна JSON -подібна схема даних);
- досить гнучка мова для формування запитів;
- динамічні запити;
- повна підтримка індексів;
- профілювання запитів;
- швидкі оновлення «на місці»;
- ефективне зберігання двійкових даних великих обсягів, наприклад, фото та відео;
- журналювання операцій, що модифікують дані в БД;
- підтримка відмовостійкості і масштабованості: асинхронна реплікація, набір реплік і шардінг;
- може працювати відповідно до парадигми MapReduce.

Враховуючи необхідність маніпулювати різноманітними даними, які зберігаються у ієрархічному вигляді у форматі XML, та необхідність проводити

часті операції запису, доцільно буде обрати саме MongoDB в якості СУБД для системи обробки даних.

3.5 Вибір інструментарію створення інтерфейсу користувача

Інтерфейс користувача подано як веб-застосування. Для розробки розмітки веб-сторінок використовується мова HTML (мова розмітки гіпертексту, англ. hyper text markup language), а для придання веб-сторінкам інтерактивності - мова Javascript, що активно використовується для написання сценаріїв у браузері та інтерпретується усіма сучасними браузерами. При розробці інтерактивних сторінок буде використано бібліотеку jQuery, що значно спрощує розробку на мові Javascript.

Важливим елементом інтерфейсу користувача, який заслуговує окремої уваги, є механізм відображення карт. Самостійна розробка такого механізму недоцільна, оскільки потребує забагато часу та ресурсів, отже слід обрати варіант з існуючих рішень.

Найбільш розвинутий та найпопулярніший сьогодні механізм відображення карт - Google Maps. До його переваг можна віднести наступні:

- карти та супутникові фотознімки усієї поверхні земного шару;
- можливість прокладати маршрути між двома точками та обирати маршрут, що задовольняє вказаним вимогам, враховуючи затори на дорогах;
- можливість побудови власних маршрутів та їх накладання на карту (також в режимі реального часу);
- наявність API, що дозволяє прикладній програмі легко взаємодіяти за сервісом.

Отже, для відображення маршрутів на карті, прокладання маршрутів та відображення місця положення пристроїв буде використано сервіс Google Maps. Взаємодія програми за сервісом буде налагоджена за допомогою Google Maps API.

3.6 Вихідні дані

3.6.1 Формування тестових даних

Для того, щоб дослідити різні параметри формату серіалізації, необхідно протестувати всі можливі сценарії. Основними особливостями форматів серіалізації є підтримка певних типів даних та підтримка складної ієрархії об'єктів. Дослідні дані формувалися виходячи з умови, що формати підтримує серіалізацію як примітивних типів мови java, так і об'єктних типів, а також їх масивів та колекцій. Також, необхідно формувати дані таким чином, щоб розмір об'єкта, який серіалізується міг калібруватися від 1 до 500 МБ з інтервалом 10МБ, щоб отримати чітку та зрозумілу залежність того чи іншого параметру від розміру оброблюваних даних. Як результат, було прийнято рішення формувати об'єкт із статичною вагою (100 КБ), який би мав у своєму контексті усі існуючі колекції і масиви об'єктів. Об'єкт, що серіалізується має список статичних об'єктів по 100КБ і його розмір визначається кількістю проініціалізованих об'єктів у колекції цього елемента.

3.6.2 Синтаксис та простота реалізації

Етап серіалізації даних не має бути ресурсозатратним чи використовувати складні технології. Цей етап необхідний у будь-якому клієнт-серверному застосунку і його імплементація – обов'язкова фаза проекту. Також, важливо, щоб серіалізація підтримувала будь-які існуючі типи даних та могла обробити будь-яку складну ієрархію даних. Код має залишатися не лише читабельним задля подальшої легкої підтримки та розвитку, але й легко реалізованим.

3.7 Розробка діаграми варіантів використання

Сервіс обробки даних дозволяє реєструватися як окремим користувачам, так і компаніям.

Користувач має можливість завантажувати маршрути з GPS-навігатора автоматично в режимі реального часу та вручну. Для ручного завантаження маршруту необхідно завантажити файл маршруту у систему. Щоб отримувати дані з пристроїв в режимі реального часу, потрібно зареєструвати у системі пристрій, який буде передавати геопозиційні дані в систему за допомогою деякого протоколу та з певною частотою.

Кожний маршрут асоціюється з пристроєм. При ручному завантаженні користувач власноруч обирає з яким пристроєм асоціювати маршрут.

Користувач має можливість переглядати завантажений (або завантажуваний) маршрут в режимі карти та в режимі перегляду статистики.

Доступна наступна статистика:

- загальний шлях;
- час простою;
- середня швидкість;
- графік швидкості;
- набрана та втрачена висота;
- графік висот;
- середня швидкість на відрізках маршруту;
- фото- та відеоматеріали для туристичних місць маршруту.

Статистику можна переглядати в розрізі певного маршруту, усіх маршрутів або деяких обраних маршрутів.

Маршрут може бути публічним (доступний для перегляду всім користувачам системи) або приватним (доступний лише тим, кому власник маршруту надав права).

Маршрут та статистику можна експортувати в одному з поширених форматів (маршрут — gpx, kml; статистику — csv, xls, xml, html).

Опираючись на вищенаведені вимоги, побудована діаграма варіантів використання, що наведена на рисунку 3.5.

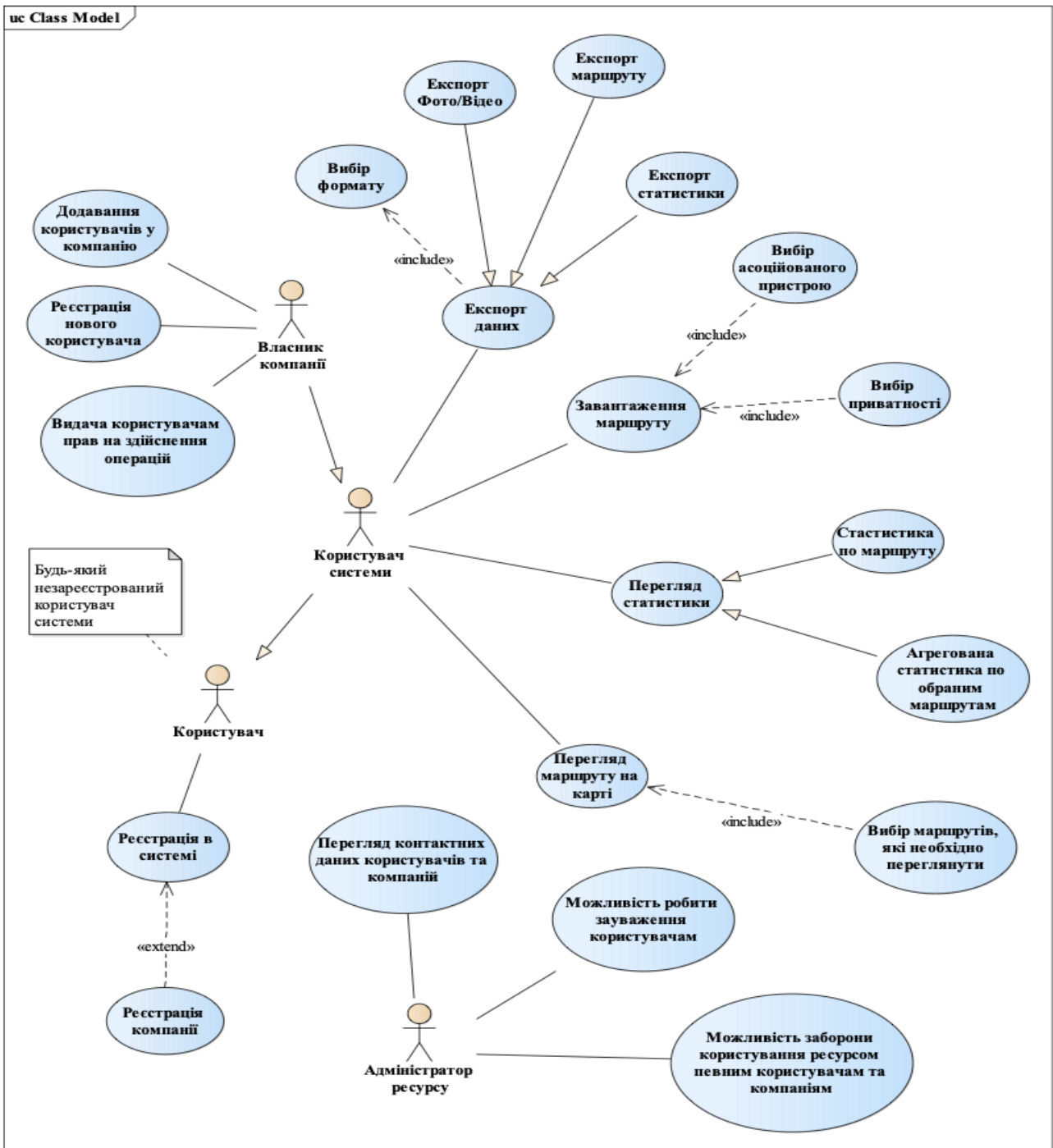


Рисунок 3.5 – Діаграма варіантів використання

3.8 Розробка діаграми компонентів

На основі обраних технологій та компонентів побудовано діаграму компонентів, що зображена на рисунку 3.6.

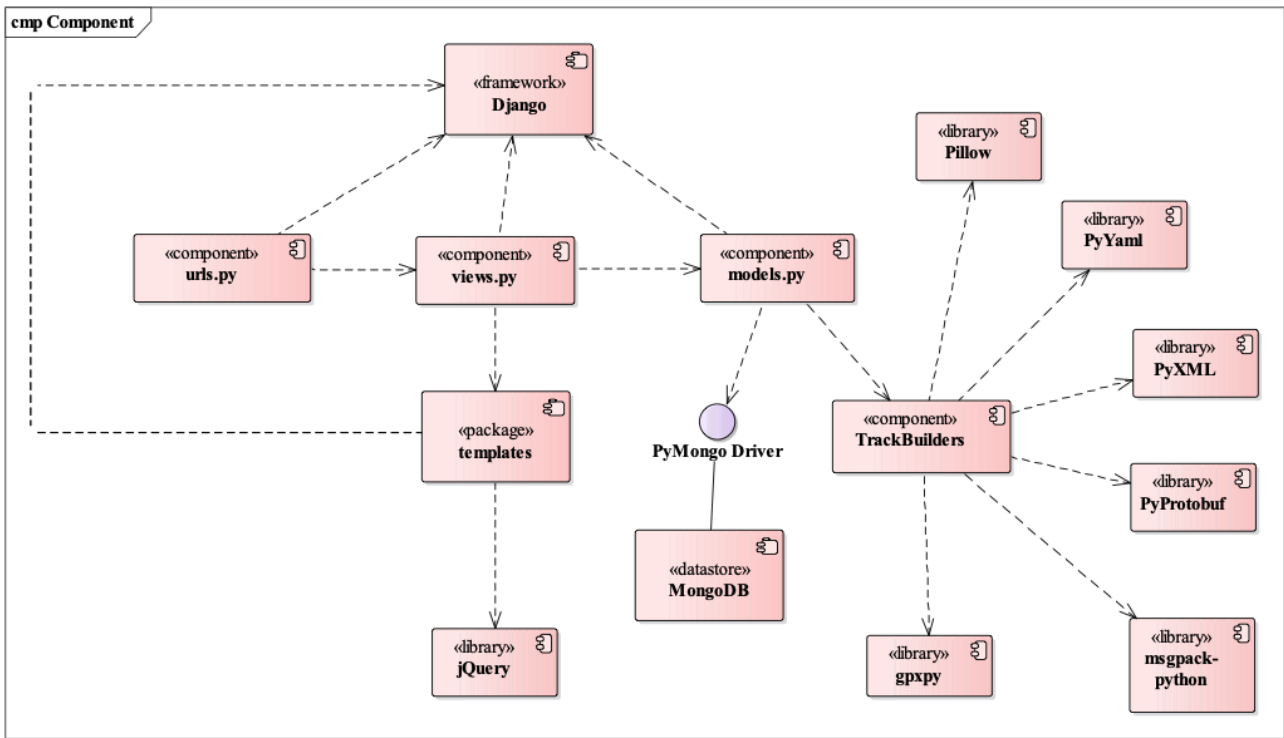


Рисунок 3.6 – Діаграма компонентів

3.9 Розробка діаграми класів

Діаграма класів – діаграма, що демонструє класи системи, їх атрибути та взаємозв'язки. Діаграма класів представлена на рисунку 3.7.

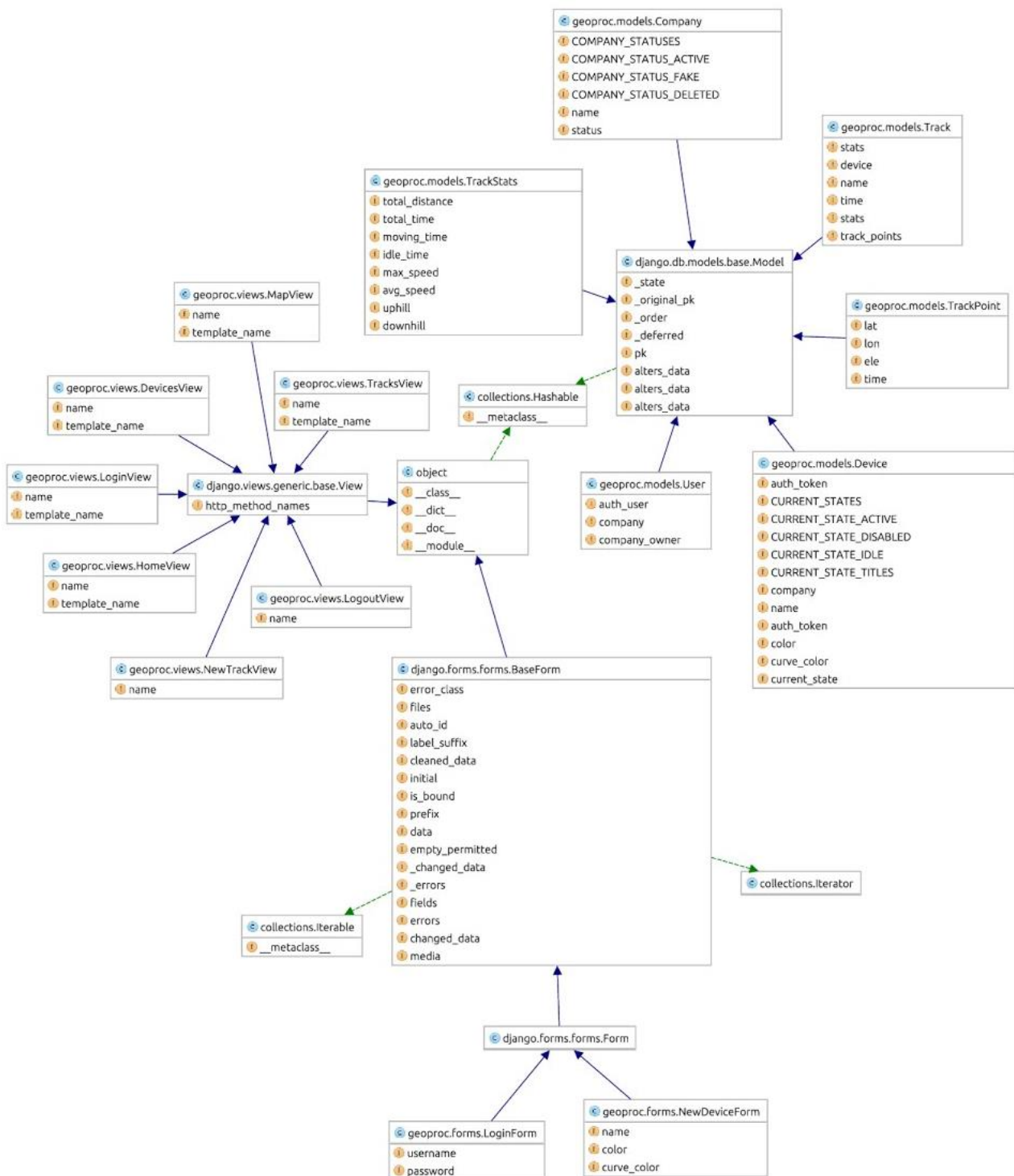


Рисунок 3.7 – Діаграма класів

3.10 Розробка моделі даних

В моделі описуються всі сутності системи, які будуть зберігатися в базі даних. В базі даних будуть присутні наступні моделі:

- компанія;

- користувач;
- пристрій;
- маршрут;
- статистика.

Для зручності разом з кожним користувачем, який самостійно зареєструвався, автоматично буде створюватись компанія, до якої користувач буде входити. Така компанія буде помічатися як несправжня до тих пір, поки користувач не вирішить зареєструвати компанію.

Одна компанія може мати безліч користувачів, а також безліч пристроїв. Кожен пристрій може містити безліч збережених маршрутів. Кожен маршрут має лише одну статистику. Маршрут може знаходитись в одному з двох станів: завершений чи незавершений. Один пристрій може мати одночасно лише один незавершений маршрут та безліч завершених. Незавершений маршрут — той, який записується в даний момент часу.

Варіант, при якому один пристрій намагається створити декілька маршрутів одночасно, наперед помилковий. Новий маршрут може бути створений лише в тому випадку, коли попередній вже був закритий. Схема зв'язків між елементами моделі зображена на рисунку 3.8.

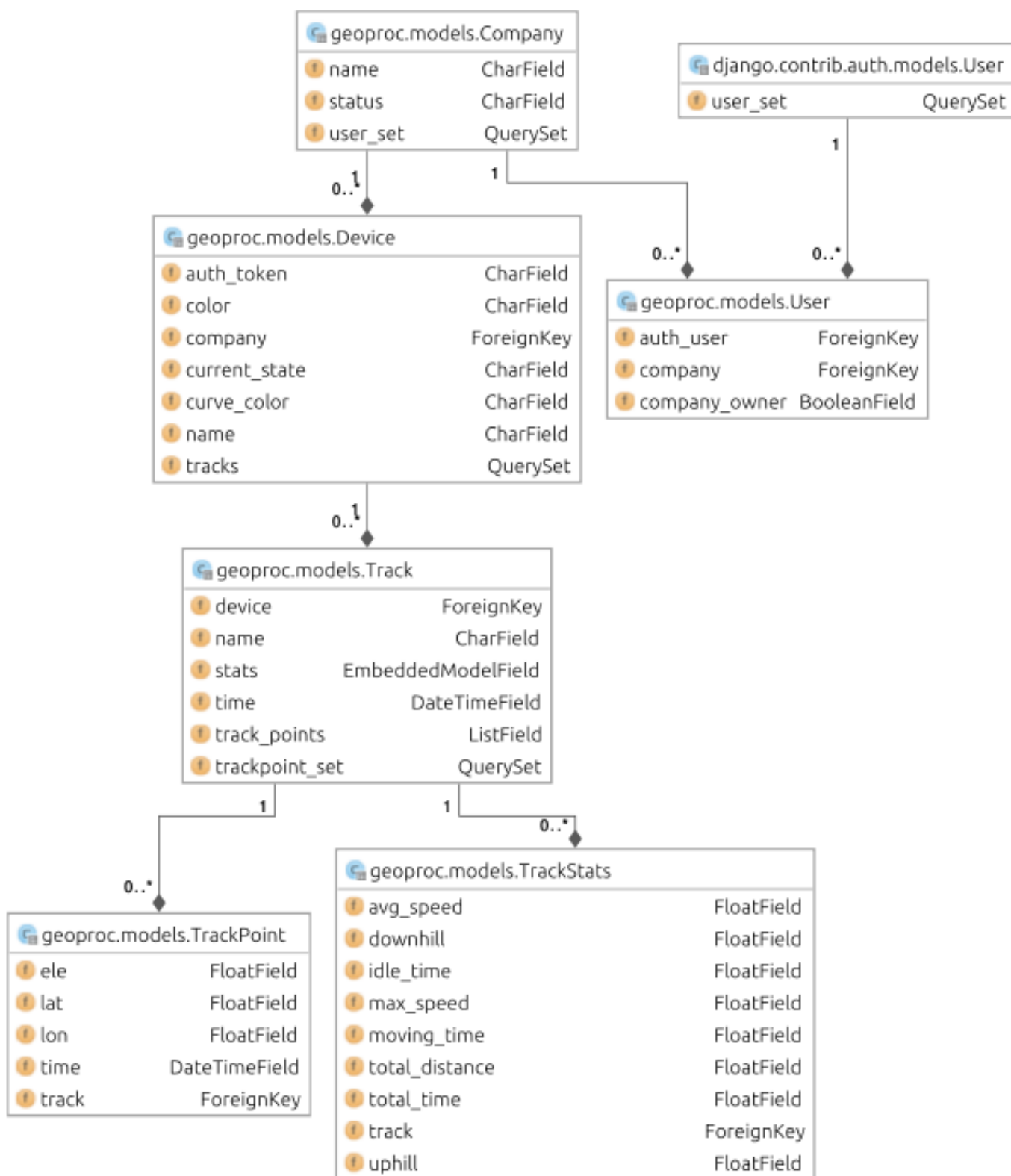


Рисунок 3.8 — Схема зв'язків між елементами моделі

3.11 Розробка основних модулів застосування

3.11.1 Модуль взаємодії з зовнішніми пристроями

Як було вирішено в попередніх розділах, модуль взаємодії з зовнішніми пристроями реалізується як обробник HTTP-запитів, які сервер отримує від клієнта в JSON-форматі. Отже необхідно розробити протокол обміну повідомленнями між навігаційними приладами (клієнт) та веб-сервісом (сервер).

Список заголовків, які клієнт має передавати серверу зображений у таблиці 3.1.

Таблиця 3.1 – Список заголовків

Заголовок запити	Опис	Необхідність
Authorization	Токен авторизації	обов'язково
Content-Length	Довжина запити (включно з заголовками)	обов'язково
Content-MD5	Хеш-сума запити (веб-сервіс може перевірити цілісність отриманого запити)	опціонально
Content-Type	MIME тип тіла запити (application/xml)	обов'язково
Date	Дата та час запити	обов'язкого
Host	URI запити	обов'язкого

В залежності від того, на який URL буде відправлено запит, сервер може виконувати різні дії:

/initialise;

/trackseg;

/pause;

/resume;

/finish;

Коли запит направлено на URL /initialise, тіло запиту повинно бути пустим. Сервер таке повідомлення сприймає як початок руху пристрою та ініціалізує всі процедури, які необхідні для подальшого запису треку.

Запит на URL /pause сигналізує про зупинку та вимкнення пристрою, що означає, що протягом якогось часу дані надходити не будуть, але трек ще не завершено (тіло запиту має бути пустим).

Запит на URL /resume означає продовження маршруту після паузи (тіло запиту залишається пустим).

Запит на URL /finish означає завершення маршруту (тіло запиту залишається пустим).

Запит на URL /trackseg робиться під час руху при відправці даних. Тіло запиту при цьому має бути в XML форматі та містити інформацію про пройдену ділянку маршруту.

Приклад такого запиту:

URL:

`https://example.com/trackseg`

Заголовки:

POST / HTTP/1.1

Host: example.com

Date: Mon, 09 Jun 2018 18:57:02 +0300

Content-Length: 513

Content-Type: application/json

Authorization: zVNpoQNsOSxZKqOZgckhpQ

Тіло запиту:

```
{
  "trackseg": {
    "trckpt": {
      "lat": 59.934721667,
```

```
        "lon": 30.310183333,  
        "ele": 103.54,  
        "time": "2018-09-06T18:56:51Z",  
    },  
    "trackpt": {  
        "lat": 59.934731667,  
        "lon": 30.310483333,  
        "ele": 103.52,  
        "time": "2018-09-06T18:57:51Z",  
    }  
}  
}
```

Сервер перевіряє отриманий JSON-документ на відповідність схемі (та у випадку, якщо вказаний заголовок Content-MD5, перевіряє достовірність отриманих даних), передає дані на наступний рівень та відправляє клієнту HTTP - відповідь зі статусом.

Клієнт надсилає дані за допомогою POST-запитів. Відповідь сервера не містить ніяких даних, крім стандартних HTTP заголовків. Про успіх або невдачу обробки клієнтського повідомлення свідчить заголовок коду HTTP стану. Якщо код стану містить помилковий статус, клієнт повинен повторити спробу передачі повідомлення.

3.11.2 Модуль обробки та збереження даних

Функціонал даного модуля виконується наступними класами:

- TrackBuilder — відповідає за створення та збереження маршрутів, що отримані від зовнішніх (навігаційних) пристроїв та вручну від користувача. Клас має наступні публічні методи:
 - initialize_new_track(device_id, track_name)

Метод створює в базі новий порожній маршрут з назвою, що вказана в

параметрі `track_name`, асоціює його з певним пристроєм та повертає ідентифікатор нового маршруту. Один пристрій одночасно може передавати один маршрут, отже, якщо вже існує маршрут, який асоційований з певним пристроєм, то ніякий інший маршрут не може бути асоційований з тим же пристроєм, аж доки попередній не буде переведено в стан “завершений”;

- `bulk_update_current_track(device_id, track_segment)`

Знаходить в базі відкритий маршрут, що асоційований з вказаним пристроєм, та доповнює його даними, вказаними у другому параметрі;

- `close_current_track(device_id)`

Завершує поточний маршрут вказаного пристрою та переводить його в стан “завершений”.

- `TrackStatsBuilder` — відповідає за створення та збереження статистики по маршруту. Має один публічний метод:

- `build_track_stats(track_id)`

Будує статистику по вказаному маршруту, зберігає її в БД та повертає ідентифікатор створеного об’єкту статистики.

Для маніпуляцій геопозиційними даними використовується бібліотека для python з відкритими вихідними кодами “`grxru`”. Бібліотека надає можливість зчитувати трек з файлу, а також виводити необхідну статистику.

Для маніпуляцій даними XML використовується бібліотека для python з відкритими вихідними кодами “`PyXML`”.

Для маніпуляцій даними YAML використовується бібліотека для python з відкритими вихідними кодами “`PyYAML`”.

Для маніпуляцій даними у форматі MessagePack використовується бібліотека для python з відкритими вихідними кодами “`msgpack-python`”.

Для маніпуляцій даними у форматі Protobuf використовується бібліотека для python з відкритими вихідними кодами “`PyProtobuf`”.

Для маніпуляцій фото у різних форматах використовується бібліотека для python з відкритими вихідними кодами “`Pillow`”.

3.11.3 Модуль взаємодії з користувачем

Фреймворк Django реалізує механізми авторизації та аутентифікації користувачів, підтримання сесій за допомогою cookies. Також фреймворк захищає від різного роду веб-вразливостей, таких як міжсайтовий скиптинг, SQL-ін'єкції та інші. Отже піклуватися про ці речі немає необхідності

Система аутентифікації Django складається з:

- Користувачів;
- Прав: Бінарні (так / ні) прапори, що визначають наявність у користувача права виконувати певні дії;
- Груп: Загальний спосіб призначення міток і прав на безліч користувачів;
- Настроюваної системи хешування паролів;
- Інструментів для форм і представлень для аутентифікації користувачів або для обмеження доступу до контенту;
- системи плагінів.

3.11.4 Інтерфейс користувача

Інтерфейс користувача побудовано як веб-застосування. Для розробки розмітки веб-сторінок використовується мова HTML (мова розмітки гіпертексту, англ. hyper text markup language), а для придання веб-сторінкам інтерактивності - мова Javascript, що активно використовується для написання сценаріїв у браузері та інтерпретується усіма сучасними браузерами. При розробці інтерактивних сторінок використано бібліотеку jQuery, що значно спрощує розробку на мові Javascript.

Інтерфейс користувача включає наступні сторінки:

- Реєстрація

Сторінка надає користувачам можливість зареєструватись у системі; містить форму реєстраційних даних, яку користувачу необхідно заповнити та

відправити на сервер.

- Авторизація

Сторінка авторизації містить поля вводу логіна та пароля, дозволяє авторизуватись у системі.

- Реєстрація компанії

Сторінка містить реєстраційну форму для створення компанії, надає користувачам можливість реєструвати у системі власну компанію.

- Управління користувачами компанії

Виводить список користувачів компанії, дозволяє додати або видалити користувача, надати або відібрати у користувача привілеї на здійснення певних дій. Сторінка доступна лише адміністратору компанії.

- Управління зовнішніми пристроями

Дозволяє реєструвати в системі новий пристрій.

- Управління маршрутами

Виводить список всіх маршрутів компанії або користувача. Дозволяє видалити, перейменувати або створити новий маршрут, завантаживши файл треку.

- Перегляд маршруту

Виводить карту з маршрутом та статистику по маршруту, а саме:

- загальний шлях;
- загальний час;
- час простою;
- час руху;
- середня швидкість;
- графік швидкості;
- набрана та втрачена висота;
- графік висот;
- середня швидкість на відрізках маршруту;
- туристичні місця та фото-, відеоматеріали.

– Перегляд сумарної статистики

Дозволяє переглядати сумарну статистику по певному пристрою або по всім пристроям, зареєстрованим в системі.

3.12 Тестування програмного засобу

Для моделювання роботи системи та тестування працездатності використано таку стратегію формування вибірки з теорії ймовірностей, як наближене моделювання.

Для підтвердження працездатності застосування створено сценарій, який емулює роботу зовнішнього пристрою. Сценарій працює наступним чином: Існуючий файл з реальним маршрутом у GPX-форматі зчитується з диску та відсилається фрагментами в систему, використовуючи внутрішній API, що розроблений для взаємодії з навігаційними пристроями.

Результати моделювання підтверджують працездатність розробленого застосування. Експеримент виконано 50 разів, результати підтверджують працездатність програми.

4 ДОСЛІДЖЕННЯ ТА ОПТИМІЗАЦІЯ МЕТОДІВ СЕРІАЛІЗАЦІЇ

4.1 Порівняльна характеристика форматів серіалізації

4.1.1 Швидкість обробки даних

При розробці будь-якої системи, швидкість обробки або передачі даних грає найвизначнішу роль. Більшість розроблюваних систем орієнтовані на користувача і мають надавати не тільки зручний інтерфейс, а й зручне користування застосунком, адже системи, у яких час обробки запиту перевищує хоча б 5 секунд залишаються не конкурентоспроможними на ряду із сучасними програмами. Саме тому замовники, купуючи програмний продукт платять не тільки за ефектний зовнішній вигляд, а й за швидкість роботи застосунку. Більшість розроблюваних систем працюють із даними, що витягуються із бази та постійно оновлюються, а значить, мають як мінімум дві розділені сутності – клієнт та сервер. При чому, процес обміну даними між цими сутностями виконується дуже часто. А якщо застосунок має мільйони користувачів одночасно, наприклад соціальні мережі чи онлайн ігри, то цінність параметру швидкості обробки та передачі даних лише збільшується. У контексті швидкості обробки даних в клієнт-серверних застосунках серіалізація даних являється одним із найслабкіших місць. Швидкість передачі інформації в основному залежить від протоколу, який використовується та об'єму даних, що передається. Про цей параметр докладніше буде описано в наступній частині. Отже, саме на швидкодію серіалізації/десеріалізації звертають увагу розробники при виборі формату серіалізації і при проведенні своїх досліджень та порівнянь ефективності різних найпопулярніших форматів серіалізації, цей параметр було першочергово включено до досліджень. Це та наступні дослідження проводились на тестових даних, про формування яких докладно описано у попередньому розділі, серіалізацію імплементовано мовою java. Графік на

рисунку 4.1 відображає залежність швидкості серіалізації та десеріалізації даних, які серіалізуються/десеріалізуються, на рисунку 4.2 – від об'єму даних.

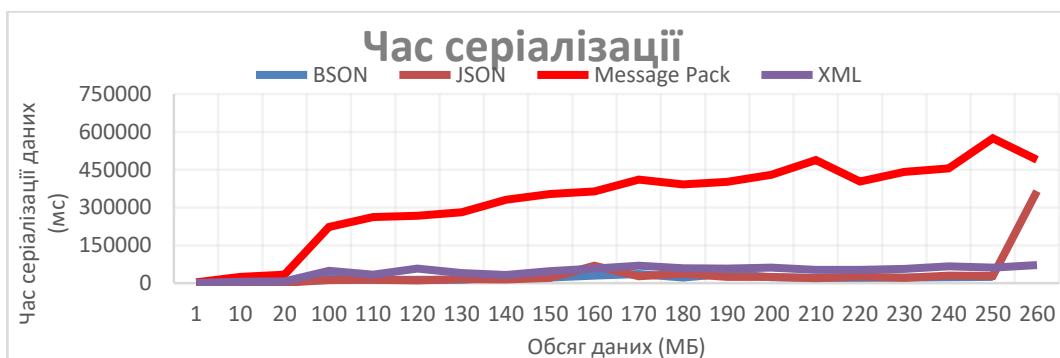


Рисунок 4.1 – Залежність часу серіалізації від обсягу даних

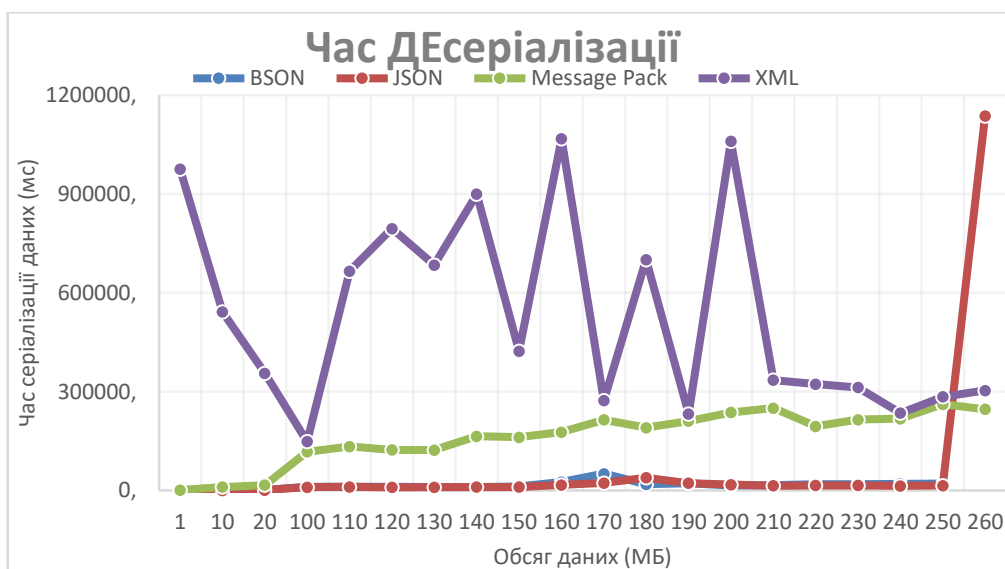


Рисунок 4.2 – Залежність часу десеріалізації від обсягу даних та формату десеріалізації

Як видно з графіку на рисунку 4.2, XML необхідний незрівнянно великий час по відношенню до інших розглянутих форматів для серіалізації даних. Тож, приведемо більш детальну порівняльну характеристику для інших форматів серіалізації на рисунку 4.3.

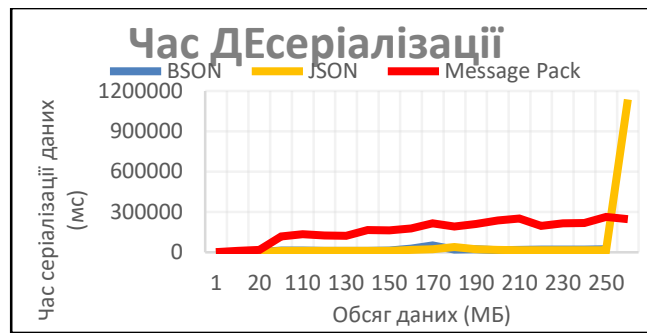


Рисунок 4.3– Залежність часу десеріалізації від обсягу даних та формату десеріалізації без представлення XML формату

Отже, з отриманих даних можна зробити висновок, що найшвидшим форматом серіалізації даних є binary json, що є оптимізованим форматом стандартної json-серіалізації. Мета створення формату bson полягала в створенні формату, що обробляв би дані швидше, ніж json. Тож, не дивно, що він виявився найефективнішим. JSON швидше опрацьовує дані доки їх розмір не перевищує 250 МБ, далі розрив необхідного для роботи часу надзвичайно великий. Хоча, можна помітити, що JSON та BSON мають практично однакові характеристики, аж до моменту, коли розмір даних сягає 260 МБ, де різниця стає помітною.

Час десеріалізації даних для Message Pack у 2 млн. разів перевищує час, затрачений BSON та JSON форматами, проте, він являється доволі стабільним форматом і час затрачений на обробку даних збільшується лінійно, на відміну від JSON. Але як видно з рисунку 3.6, Message Pack суттєво відрізняється від інших порівняних форматів серіалізації через витрату надзвичайно великого часу для обробки даних. Коефіцієнт збільшення досягає 6 млн., що робить даний формат неконкурентоспроможним на ряду із іншими методами серіалізації.

Щодо XML, то якщо час серіалізації порівняно із MessagePack близький до JSON та BSON, різниця лише приблизно у 5 тис. разів, то при десеріалізації даних цей формат не слід використовувати, якщо час обробки даних для системи важливий. По-перше, час змінюється не лінійно. Це значить, що прогнозувати час, що знадобиться для десеріалізації 1КБ чи сотень мегабайт неможливо. По-друге, доволі великий розмах – від 1,5 млн. до 12 млн., проте навіть найменша

вартість затраченого часу не менше ніж час, який витрачає MessagePack, тобто, жодної переваги при десеріалізації на фоні аналізованих форматів XML не має.

4.1.2 Використання оперативної пам'яті

Оперативна пам'ять – це пам'ять з довільним доступом, енергозалежна частина системи комп'ютерної пам'яті, в якій під час роботи комп'ютера зберігається виконуваний машинний код (програми), а також вхідні, вихідні та проміжні дані, що обробляються процесором. Обмін даними між процесором і оперативною пам'яттю проводиться: безпосередньо чи через нашвидку пам'ять 0-го рівня - реєстри в АЛП, або за наявності апаратного кешу процесора - через кеш. Розміщені дані в сучасній напівпровідниковій оперативній пам'яті доступні і зберігаються тільки тоді, коли на модулі пам'яті подається напруга. Вимкнення живлення оперативної пам'яті, навіть короточасне, призводить до спотворення або повного руйнування інформації, що зберігається. Енергозберігаючі режими роботи материнської плати комп'ютера дозволяють переводити його в режим сну, що значно скорочує рівень споживання комп'ютером електроенергії. В режимі глибокого сну харчування ОЗУ відключається. У цьому випадку для збереження вмісту ОЗУ операційна система (ОС) перед відключенням живлення записують вмісту ОЗУ на пристрій постійного зберігання даних (як правило, жорсткий диск). Наприклад, в ОС Windows XP вміст пам'яті зберігається в файл hiberfil.sys, в ОС сімейства Unix - на спеціальний swap-розділ жорсткого диска. У загальному випадку, ОЗП містить програми і дані ОС і запущені прикладні програми користувача і дані цих програм, тому від обсягу оперативної пам'яті залежить кількість завдань, які одночасно може виконувати комп'ютер під управлінням ОС. Оперативний пристрій, ОЗП - технічний пристрій, що реалізує функції оперативної пам'яті. ОЗП може виготовлятися як окремий зовнішній модуль або розташовуватися на одному кристалі з процесором, наприклад, в однокристальних ЕОМ або однокристальних мікроконтролерах. ОЗП більшості сучасних комп'ютерів являє собою модулі динамічної пам'яті, що містять

напівпровідникові ІС ЗП, організовані за принципом пристроїв з довільним доступом. Пам'ять динамічного типу дешевше, ніж статичного, і її щільність вище, що дозволяє на тій же площі кремнієвого кристала розмістити більше елементів пам'яті, але при цьому її швидкодію нижче. Статична пам'ять, навпаки, більш швидка пам'ять, але вона і дорожче. У зв'язку з цим основну оперативну пам'ять будують на модулях динамічної пам'яті, а пам'ять статичного типу використовується для побудови кеш-пам'яті усередині мікропроцесора.

При серіалізації даних залежно від використовуваного формату, можуть генеруватися різного виду допоміжні об'єкти, що збільшуватиме кількість споживаної оперативної пам'яті. Кількість займаної пам'яті теж є важливим параметром при використанні і виборі методу серіалізації. Якщо ж комп'ютер не володіє характеристиками сучасної обчислювальної техніки, то поліпшення його роботи може зазнати збитків. Саме тому при проектуванні програми варто прагнути до того, щоб застосунок зберігав в оперативній пам'яті якомога менше використовуваних об'єктів. Розглядаючи цей параметр слід згадати про особливості реалізації видалення з пам'яті не використовуваних об'єктів в Python. Python має так званий *garbage collector*, котрий і видаляє об'єкти, що більше не використовуються (наприклад, ті, на які відсутні посилання). На жаль, Python виконує цю дію за своїм непрогнозованим розсудом і виконати цей процес примусово, як наприклад це дозволяє C ++, неможливо. Завдання програміста полягає в тому, щоб створити таку програму, яка буде генерувати як можна менш важкі і меншу кількість тимчасових об'єктів з метою збереження вільної пам'яті. На рисунку 4.4 та рисунку 4.5 показана залежність обсягу займаних даних в пам'яті Python різних форматів серіалізації/десеріалізації в залежності від обсягу даних.

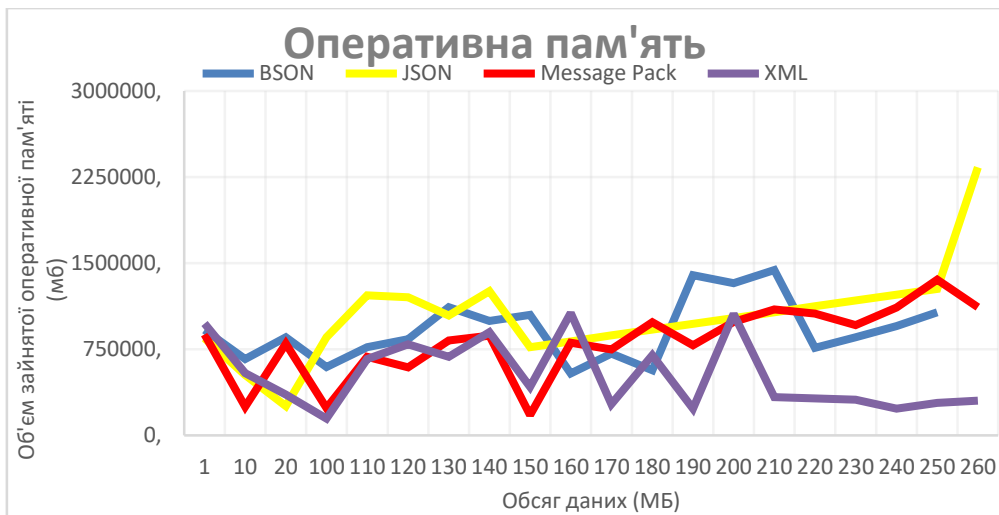


Рисунок 4.4 – Залежність зайнятої оперативної пам'яті під час процесу серіалізації

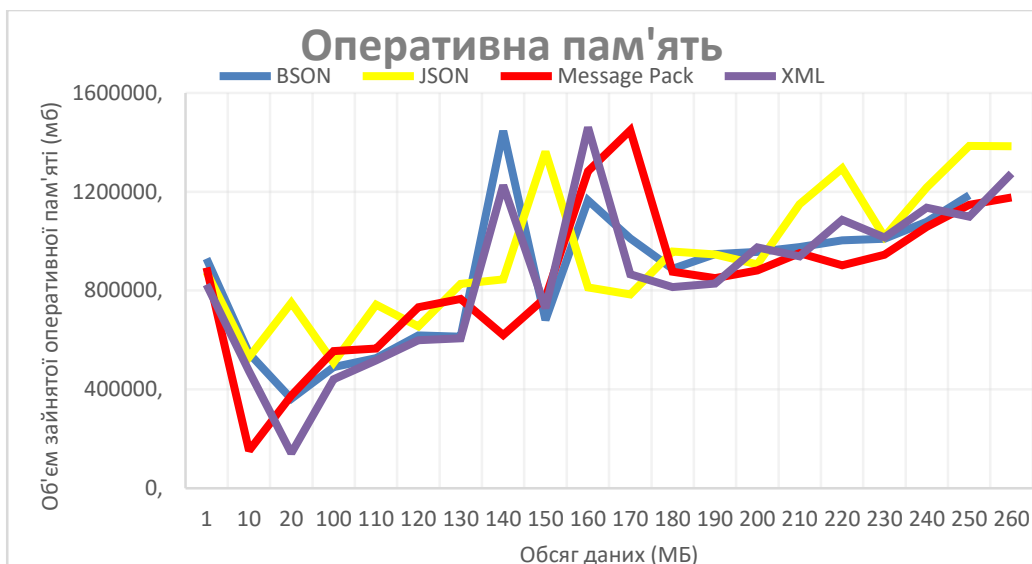


Рисунок 4.5 – Залежність зайнятої оперативної пам'яті під час процесу десеріалізації

При використанні платформи Python, на жаль, на досліди пов'язані із зайнятою оперативною пам'яттю великий вплив має так зване «сміття» - вже непотрібні програмі дані, що не використовуються, але все ще займають частину оперативної пам'яті. Справа в тому, що у мові Python неможливо проконтролювати збір цього самого «сміття». Задля зменшення цього зайвого впливу у вигляді перешкод в ході експерименту при розробці програми, були прийняті наступні кроки:

- об'єкти зберігались у файлі, а не оперативній пам'яті;
- при зчитуванні об'єкта з файлу змінні об'являлися локально так, що для серіалізації різними форматами використовувався лише один об'єкт;
- локальна змінна, якій присвоювалось посилання на об'єкт об'являлась всередині циклу, так що по завершенню ітерації об'єкт позначався як сміття і мав видалятися з пам'яті, коли цей крок виконувався Python.

Отже, як видно з рисунку 4.5, дані для кожного з експериментів для двох точок поряд доволі сильно відрізняються і зростають не лінійно, але приблизні результати експерименту показують, що найбільше пам'яті займає саме JSON, і, як і було прогнозовано, піку досягає при досягненні розміру даних 260МБ, як і час, за який відбувалась серіалізація. При цьому XML, як видно, займає найменше оперативної пам'яті серед інших форматів при серіалізації.

При десеріалізації використання ОЗП зростає більш-менш лінійно. Піки створені при обсязі даних 10, 20, 140-170 МБ зумовлені саме роботою Python. Тим не менш, JSON знову вирізняється серед інших форматів великою зайнятістю оперативної пам'яті даними.

4.1.3 Розмір серіалізованих даних

Як було зазначено в першій частині цього розділу, при передачі інформації важлива не тільки швидкість обробки інформації, а й швидкість її передачі. Цей параметр залежить в основному від протоколу даних, але й розмір даних, що передаються між вузлами застосунку має велике значення. Дослідження мало на меті порівняти об'єм даних, що зберігаються на диску після процесу серіалізації. Графік на рисунку 4.6 показує залежність об'єму серіалізованих даних від об'єму даних, що серіалізувались та використовуваного формату серіалізації.

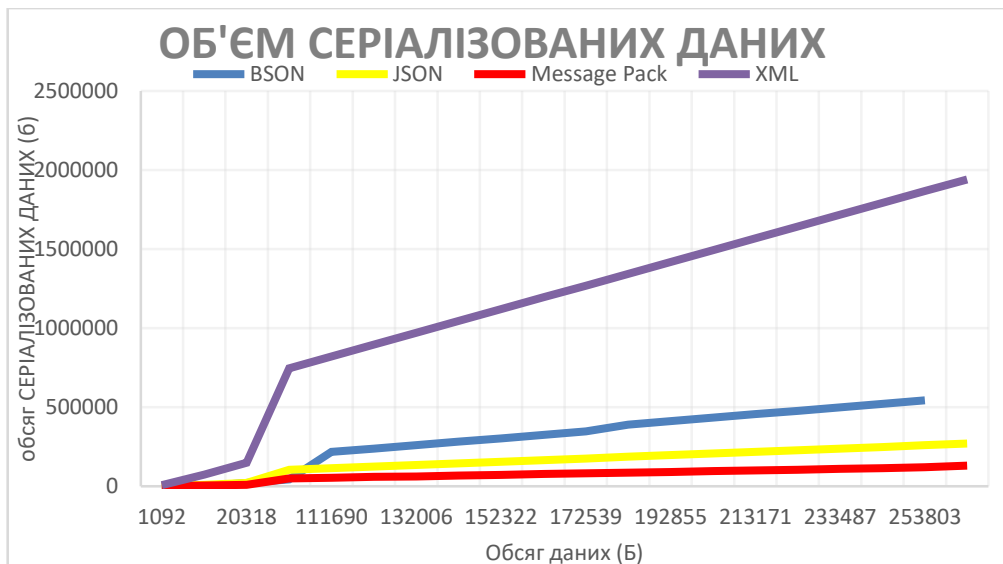


Рисунок 4.6 – Залежність розміру серіалізованих даних від даних, що серіалізуються

Як видно з графіку, XML не стискає дані, а лише збільшує їх обсяг за рахунок надлишкової інформації, в той час як Message Pack являється найкращим архіватором для повідомлень, що передаються між клієнтом та сервером.

На рівні із швидкодією BSON, як архіватор він показує себе не з найкращої сторони. Серед досліджуваних форматів він також не той формат, що стискає дані при серіалізації, а збільшує їх розмір приблизно у 2 рази. Для обох випадків це значить, що формат поміщає багато надлишкового коду у сформований серіалізований файл.

Окрім того, MessagePack – єдиний формат, характеристика якого завжди збільшується лінійно і не досягає значення більше 2 млн байт, в той час як для кожного формату властивий різкий скачок характеристики на проміжку від 100 до 120 МБ. Це значить, що MessagePack – найбільш прогнозований та ефективний формат.

4.2 Оптимізація методів серіалізації

Головною задачею магістерської дисертації було оптимізувати існуючі

формати серіалізації даних. У попередній частині розділу, були проведені дослідження роботи деяких форматів по найважливішим для роботи серіалізації параметрам: швидкість обробки даних, місце, що займає процес в оперативній пам'яті та місце, що займають дані на диску. У результаті досліджень було виявлено, що найефективнішим форматом серіалізації для невеликих даних є JSON, найкращим архіватором даних – MessagePack.

MessagePack – бінарний формат серіалізації даних. Насправді, BSON, який по показникам є кращим за JSON, відрізняється від нього лише перекладом JSON-об'єкту у послідовність байт. Тож, основним способом оптимізації форматів серіалізації є використання бінарного формату даних при записі.

Насправді, по-перше байти займатимуть менше місце на диску, ніж модель об'єкту, сформованого за допомогою великої строки. Це основний недолік форматів, що відносяться до форматів, прийнятних для людини – такий формат завжди буде надлишковим. Для того щоб серіалізувати подібний формат (XML, JSON) необхідно спочатку сформувати нову модель даних на основі оригінальної, конвертувати її в строкове представлення даних і лише після цього записувати у файл (по символу, кожен з яких також важить 16 біт). При десеріалізації відбувається зворотній процес, який займає більше часу через застосування рефлексії – витягування назви поля чи класу для перевірки на еквівалентність по назві, що записана в файлі.

Тож запис поля, значення якого «1» спочатку буде переведений в текст, а при десеріалізації текст із одиничкою буде приведений до типу int. Коли ж значення рівне «1.0», до цих операцій додається операція витягування типу поля за допомогою рефлексії. У бінарному форматі таких надлишкових операцій можна уникнути.

Проте, бінарні формати мають свої недоліки. Такий формат не крос-платформенний. Порядок запису байт знає лише серіалізатор та десеріалізатор конкретного формату, як власне і спосіб формування послідовності байтів в залежності від типу даних. А це значить, що серіалізовані дані не зможуть бути прочитані вручну будь яким іншим серіалізатором окрім нативного та правильно

розпарсені. Але, все ж таки, бінарні формати були та залишаються оптимальнішими форматами серіалізації, ніж текстові.

Одним з найбільших недоліків JSON являється не можливість підтримки формату відмінного від UTF-8. Підтримка іншого способу кодування не оптимізує час виконання роботи чи розмір даних, але додасть формату певну особливість та розширить функціонал.

Тож, головною задачею бінарного формату – запис якомога меншої кількості інформації (в байтах) та компактна обробка даних. При чому слід зосередитись на компактній формі таких структур даних як масиви, списки та числовому типі `int`.

Отже, враховуючи специфіку системи що розробляється було прийнято рішення використовувати для JSON даних про трек, та MessagePack для медіа даних.

5 РОЗРОБКА СТАРТАПУ

5.1 Опис ідеї проекту

Таблиця 5.1. Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Туристична трекінгова система	1. Продаж компаніям підписки на сервіс	Вихід на широку аудиторію для малого бізнесу Аналітика по послугам Доступність статистики Повторні залучення
	2. Просування компаній за плату	Розширення аудиторії для компаній
	3. Продаж аналітики по ринку	Ознайомлення з ринком та конкурентами що дозволить скоректувати позиціонування або пропозиції

5.2 Технологічний аудит ідеї проекту

Таблиця 5.2. Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Мова програмування	Компільована у машинний код (C, C++)	Наявні	Доступні
2.		Компільована у байткод (Java, C#)	Наявні	Доступні
3.		Скриптова/інтерпретована (Python, Perl, Lua, Ruby)	Наявні	Доступні
Обрана технологія реалізації ідеї проекту: Скриптова/інтерпретована (Python, Perl, Lua, Ruby)				

4.	Веб-фреймворк	Django	Наявні	Доступні
5.		Pyramid	Наявні	Доступні
6.		Zope	Наявні	Доступні
Обрана технологія реалізації ідеї проекту: Django				
7.	БД/СУБД	Реляційна (MySQL)	Наявні	Доступні
8.		Ключ-значення (BerkeleyDB)	Наявні	Доступні
9.		Документорієнтована (MongoDB)	Наявні	Доступні
Обрана технологія реалізації ідеї проекту: MongoDB				

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Таблиця 5.3. Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1.	Кількість головних гравців, од	0
2.	Загальний обсяг продаж, грн/ум.од	1372 грн/ум. од.
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5.	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6.	Середня норма рентабельності в галузі (або по ринку), %	50%

Таблиця 5.4. Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1.	Вихід на більшу аудиторію	1. Малий бізнес 2. Середній бізнес	Можливості по об'єму користувачів	Зручність та швидкість доступу Наявність користувачів
2.	Необхідність в аналітиці	1. Малий бізнес 2. Середній бізнес	Потреби в різних інструментах	Наявність іструментів аналітики та достатній бази даних
3.	Автоматизація бізнес процесів	1. Малий бізнес 2. Середній бізнес	Різні пристрої, що використовуються в аналітиці	Підтримка широкого кола пристроїв

Таблиця 5.5. Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Неподолання межі критичної маси користувачів	Недостатня кількість користувачів для забезпечення зацікавленості бізнесів та користувачів	Маркетингова компанія Партнерство с бізнесами
2.	Відсутність ринку	Відсутність шляху збуту товару внаслідок помилкового орієнтування	Ретельний розгляд проблем потенційних клієнтів Залучення експертів та менторів Консультації із спеціалістами
3.	Недостача капіталовкладень	Витрачені усі кошти до моменту виходу на ринок	Пошук нових джерел інвестицій

Таблиця 5.6. Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Отримання інвестицій	Отримання капіталу що необхідний для розширення можливостей продукту	Розширення функціоналу Маркетингова компанія
2.	Успішна маркетингова політика	В результаті проведеної маркетингової політики отримана висока зацікавленість користувачів	Підтримка стабільної роботи системи та проведення масштабування системи Збільшення цін на використання сервісу Використання подібної маркетингової стратегії надалі для залучення нових користувачів
3.	Поглинання конкурентами	Пропозиція купівлі проекту або розроблених технологій одним із конкурентів	Розвиток розроблених технологій Оцінка вартості розроблених технологій

Таблиця 5.7. Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1.	Унікальність сервісу	Розроблений продукт займає вільну нішу
2.	Зацікавленість аудиторії	Цільова аудиторія на даний момент не має аналогічних інструментів хоча на них є високий запит
3.	Модель “бізнес для бізнесу”	Бізнес модель ґрунтується на наданні загальної платформи для малого бізнесу із виходом на нових клієнтів. Малі бізнеси не мають конкуренції між собою тому будуть зацікавлені у тому щоби бути представленими на платформі

Таблиця 5.8. Порівняльний аналіз сильних та слабких сторін «Adimas»

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з Adimas							
			-3	-2	-1	0	+1	+2	+3	
1.	Унікальність сервісу	17								+

2.	Зацікавленість аудиторії	12						+	
3.	Модель “бізнес для бізнесу”	10						+	

Таблиця 5.9. SWOT- аналіз стартап-проекту

Сильні сторони: Вузька ніша Невисокі ціни	Слабкі сторони: Сильна залежність від зацікавленості аудиторії з самого початку
Можливості: Інвестиції Підвищення соціальних функцій Продаж маркетингових досліджень	Загрози: Низький інтерес аудиторії

Таблиця 5.10. Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Підвищення соціальних функцій	Дуже ймовірне	6 місяців
2.	Програма просунення у соціальних мережах	Ймовірне	2 місяці
3.	Продаж маркетингових досліджень	Малоймовірне	4 місяць
Обрана альтернатива: Підвищення соціальних функцій			

5.4 Розроблення ринкової стратегії проекту

Таблиця 5.11. Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
-------	--	---	---	--------------------------------------	--------------------------

1.	Малі компанії що організують трекові маршрути	Висока	90%	Низька	Низькі бар'єри входу
2.	Середні туристичні та великі туристичні компанії	Середня	40%	Низька	Середні бар'єри входу
Які цільові групи обрано: Малі компанії що організують туристичні маршрути					

Таблиця 5.12. Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	Надання платформи малому бізнесу	Максимальне охоплення	Високий запит аудиторії на подібну послугу	Стратегія голубого океану

Таблиця 5.13. Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Так	Нові споживачі	Ні	Стратегія першопрохідця у вузькій ніші

Таблиця 5.14. Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформулювати комплексну позицію власного проекту (три ключових)

1	Новий досвід Інструменти аналізу Багато компаній Багато користувачів	Стратегія голубого океану	Вузька аудиторія Повторні залучення Лояльність аудиторії Збір унікальної бази даних	Доступ до найцікавіших маршрутів Доступ до статистики Метрики для покращення вашого бізнесу
---	---	---------------------------	--	---

5.5 Розроблення маркетингової програми стартап-проекту

Таблиця 5.15. Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Створення лінійки маршрутів	Спрощення позиціонування та формування лінійки пропозицій	Унікальні метрики алгоритми вираховування яких є лише у нас
2	Аналіз маршрутів	Збір метрик по існуючим пропозиціям	Доступ до вибірки даних не доступних конкурентам
3	Просування маршрутів	Можливість за додаткову плату отримати вихід на більшу аудиторію	Вузька аудиторія що точно зацікавлена у послугах
4	Покращення зв'язку с ЦА	Підвищення Retention Rate	Данні що цікаві користувачам змушують їх повертатись саме на нашу площадку

Таблиця 5.16. Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Програмний продукт що надає можливість збирати інформацію пр туристичний маршрут		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	Кількість		1 шт.
	Якість: стандарти якості постачання програмних продуктів		

	Пакування: вебсервіс
III. Товар із підкріпленням	Програмний продукт
	Доступ по підписці
За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності	

Таблиця 5.17. Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Через загальнодоступні канали	Донесення інформації до аудиторії Забезпечення послуги	Один рівень	Комбінований канал збуту

Таблиця 5.18. Концепція маркетингових комунікацій

№ п / п	Специфіка поведінки цільових клієнтів	Канали комунікації, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Автоматизація бізнес-процесів	Прямі офіційні	Унікальність пропозиції Аналітика та спрощення моніторингу	Формування у цільовій аудиторії обізнаності про появу нового продукту Інформування користувачів про властивості та переваги продукту Інформування користувачів про нові способи використання відомого продукту Пояснення цільовій аудиторії принципу роботи платформи Виправити у користувачів неправильні представлення про продукт	Раціоналістична стратегія реклами

ВИСНОВКИ

Спроектований програмний засіб є веб-сервісом, який дозволяє приймати дані про місцезнаходження з зовнішніх пристроїв, формувати фото- та відеоматеріали, обробляти їх та представляти користувачам в зручному вигляді інформацію про шлях та положення об'єктів.

Перевагами розробки є мультиплатформенність та простота реалізації та інсталяції.

Модульність системи дозволяє змінювати одні компоненти без впливу на інші. Запропонована система може бути легко адаптована під потреби замовлювача та доповнена додатковим функціоналом.

В якості дослідної частини обрано аналіз алгоритмів збереження інформаційних об'єктів. Як результат аналізу алгоритмів були виділені найбільш вагомі показники продуктивності кожного з розглянутих алгоритмів. Мною були розглянуті і проаналізовані та порівняні бінарні алгоритми серіалізації, а саме BSON, MessagePack, а також алгоритми, що використовують текстовий формат XML, Json. Зіставлені різні переваги і недоліки, сформовані статистичні показники їх застосування. Це все допомогло сформуванню картини, як саме використовуються і якими саме мають властивості найбільш популярні формати, а також зіставити різні критерії продуктивності і виділити з них найбільш важливі.

Для порівняння і виділення найбільш вузьких і сильних сторін були зроблені великі дослідження продуктивності використання різних серіалізаційних форматів. Дані для цих алгоритмів використовувалися однакові, розмір тестових даних коливався від 1 кілобайт до сотень мегабайт, що дозволило отримати повну картину ефективності алгоритмів щодо серіалізації в системах з різним навантаженням, і структурою – складністю повідомлень.

Проведені дослідження дозволили обґрунтувати використання методів серіалізації для різних типів даних та використати саме їх при

побудові програмного засобу збереження інформаційних об'єктів на прикладі системи управління туристичними маршрутами.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Коржов В. Многоуровневые системы клиент-сервер / В. Кожов. – Москва: Издательство «Открытые системы», 1997. – 120 с.
2. Интуит Национальный Открытый Университет. Лекция 2: Взаимодействие компонент распределенной системы [Электронный ресурс]. – Режим доступа: <http://www.intuit.ru/studies/courses/1115/177/lecture/4780>. – Дата звернення: 28.09.2018 р.
3. Рузавин Г.И. Методология научного исследования: Учеб. пособие для вузов / Г.И. Рузавин. – Киев: ЮНИТ-ДАНА, 2012. – 287 с.
4. Шилдт Г. С++: базовый курс, 3-е издание / Г.Шилдт. — Москва: «Вильямс», 2012. — 624 с. — ISBN 978-5-8459-0768-4.
5. Wikipedia. Сериализация [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F>. – Дата звернення: 30.09.2018 р.
6. Wikipedia. Comparison of data serialization formats [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats. – Дата звернення: 02.10.2018 р.
7. Архитектура клиент-сервер [Электронный ресурс]. – Режим доступа: <http://fas.aics.ru/student/lectures/aiit/cli-se.pdf>. – Дата звернення: 02.10.2018 р.
8. Таненбаум Э., Уэзеролл Д. Компьютерные Сети, пятое издание / Э. Таненбаум, Д.Уэзеролл. – Питер: Классика Computer Science, 2012. – 960 с.

9. Дейт К. Введение в системы баз данных / К.Дж.Дейт. – Москва: ДМК, 2000. – 813 с.
10. Чернуха Д., Бевз И.А. Разноуровневая архитектура клиент-сервер [Электронный ресурс] / Д. Чернуха, И.А. Бевз. – Режим доступа:
http://www.rusnauka.com/12_KPSN_2013/Informatica/3_133118.doc.htm. – Дата звернення: 04.10.2018 р.
11. Wikipedia. Файловый сервер [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80. – Дата звернення: 10.10.2018 р.
12. Сервер баз даних (SQL) [Электронный ресурс]. – Режим доступа: http://www.stss.ru/solutions/database_server.html. – Дата звернення: 10.10.2018 р.
13. Терминальные сервера [Электронный ресурс]. – Режим доступа: <http://tonk.ru/catalog/servers/>. – Дата звернення: 10.10.2018 р.
14. Трехуровневая архитектура [Электронный ресурс]. Режим доступа: <http://dic.academic.ru/dic.nsf/ruwiki/198141>. – Дата звернення: 10.10.2018 р.
15. Системы обмена сообщениями [Электронный ресурс]. – Режим доступа: <http://www.williamspublishing.com/PDF/5-8459-1146-X/part.pdf>. – Дата звернення: 12.10.2018 р.
16. Машнин Т. Web-сервисы Java / Т. Машнин. – Петербург: БХВ-Петербург, 2012. – 253 с.
17. Дергачев А.М. Проблемы эффективного использования сетевых сервисов / А. М. Дергачев // Научно-технический вестник СПбГУ ИТМО. - 2011. № 1. – С. 83–87.
18. Wikipedia. Serialization [Электронный ресурс]. – Режим доступа: <https://en.wikipedia.org/wiki/Serialization>. – Дата звернення: 12.10.2018

19. XDR Protocol Specification [Электронный ресурс]. – Режим доступа: <http://pubs.opengroup.org/onlinepubs/9629799/chap3.htm>. – Дата звернення: 12.10.2018 р.
20. BSON [Электронный ресурс]. – Режим доступа: <http://bsonspec.org/>. – Дата звернення: 12.10.2018 р.
21. Russel J., Cohn R. Fast Infoset / J. Russel, R. Cohn. – Cimmeria:Bookvika publishing, 2012. – 322 с.
22. Радченко Г.И. Распределенные вычислительные системы [Электронный ресурс]. – Режим доступа: http://glebradchenko.ru/courses/master/dot/2014/DOT_SUSU_2_Serialization.pdf. – Дата звернення: 14.10.2018 р.
23. Wikipedia. XML [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/XML>. – Дата звернення: 14.10.2018 р.
24. Wikipedia. JSON [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/JSON>. – Дата звернення: 14.10.2018 р.
25. Russel J. JSON-RPC / J. Russel. – Cimmeria: Bookvika publishing, 2012. – 188 с.
26. Obie Fernandes. Web/Tech [Электронный ресурс]. – Режим доступа: <https://dou.ua/lenta/articles/language-rating-jan-2018/?from=doufp/>. – Дата звернення: 14.10.2018 р.
27. Самые популярные операционные системы [Электронный ресурс]. – Режим доступа: <https://marketer.ua/stats-operating-system-2017/>. – Дата звернення: 14.10.2018 р.
28. Python – Вікіпедія [Электронный ресурс]. – Режим доступа: <http://ru.wikipedia.org/wiki/Python>. – Дата звернення: 14.10.2018 р.
29. About Python [Электронный ресурс]. – Режим доступа: <https://www.python.org/about/>. – Дата звернення: 14.10.2018 р.
30. Django – Вікіпедія [Электронный ресурс]. – Режим доступа: <https://uk.wikipedia.org/wiki/Django>. – Дата звернення: 14.10.2018 р.

31. Реляційна система керування базами даних – Вікіпедія [Електронний ресурс]. – Режим доступу: http://uk.wikipedia.org/wiki/реляційна_система_керування_базами_даних. – Дата звернення: 17.10.2018 р.
32. MongoDB [Електронний ресурс]. – Режим доступу: <https://en.wikipedia.org/wiki/MongoDB>. – Дата звернення: 17.10.2018 р.
33. Улянич Д.В. Методи серіалізації даних / Д.В.Улянич, Я.Ю.Дорогий // Матеріали VII Міжнародної науково-практичної конференції «Фізико-технологічні проблеми передавання, оброблення та зберігання інформації в інфокомунікаційних системах (PREDT-2018)», 8-10 листопада. – м.Чернівці, 2018.

Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

VII Міжнародна науково-практична
конференція

Фізико-технологічні проблеми переда-
вання, оброблення та зберігання інфор-
мації в інфокомунікаційних системах

Програма

8-10 листопада 2018 року
м. Чернівці, Україна

ДОДАТОК Б



Дослідження оптичних мереж з хвильовим поділом каналів

Черкаський державний технологічний університет

Івченко О.В. / Лопушанська В.Р. / Булавін К.О

Тенденції розвитку телекомунікацій в 21 столітті показують, що людство рухається по шляху створення глобального інформаційного суспільства. Розвиток мережі Internet, в тому числі поява нових послуг зв'язку, сприяє зростанню переданих по мережі потоків даних і змушує операторів шукати шляхи збільшення пропускної здатності транспортних мереж та мереж доступу.



Комп'ютерна технологія архівації та інформування по телефонних каналах

Харківський національний університет радіоелектроніки

Загайнов В.І. / Кочкін М.І. / Ляховець В.О. / Сирцов С.Л. / Чеботарьова Д.В.

Розглянуто основні принципи створення автоматизованих інформаційно-довідкових комплексів з доступом через телефонні канали та програмний вибір режимів їх роботи. На основі комп'ютерної технології обробки інформації і апаратного забезпечення цифрової обробки сигналів розроблені структури апаратно-програмних комплексів архівування та інформування по телефонних каналах. Програмна частина комплексів забезпечує автоматизовану безперервну обробку інформації в режимі реального часу і побудована на принципах системи діалогу. Розглянуто основні компоненти програмного комплексу для створення систем інформування клієнтів муніципальних служб.



Методи серіалізації даних

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Дорогий Я.Ю. / Ульянич Д.В.

У роботі досліджено різні методи серіалізації даних за декількома параметрами, які мають найбільший вплив на їх вибір при реалізації програмних застосунків. Наведені результати дають можливість визначити основні напрямки оптимізації методів серіалізації при створенні нових методів серіалізації.



Структури із паралельних провідників в радіотехнічних пристроях

Чернівецький національний університет імені Юрія Федькович

Хобзей М.М. / Вовчук Д.А.

У роботі розглянуто основні можливі застосування структур із паралельних провідників (СПП), що включають передавання зображень, енергії електромагнітних (ЕМ) хвиль та пристрої ендоскопії. Показано широкий спектр задач, що можуть бути вирішені шляхом використання СПП від мікрохвильового діапазону частот до оптичного.



Properties of on-board navigation complex with high interference immunity with variable structure for remote-piloted vehicles

Lviv Polytechnic National University

Мукуйчук М.М. / Марків В.І

The article dwells upon the peculiarities of on-board navigation complex with high interference immunity with variable structure for remote-piloted vehicles. The general scheme of navigation control of terrestrial moving object is presented.