

ЗМІСТ

Вступ	8
1 Формулювання та постановка задачі	10
1.1 Аналіз предметної області та висновки про об'єкт реалізації	11
1.2 Моделювання словника сервісу	11
1.3 Вибір технологій реалізації веб-сервісу через дослідження рейтингів та статистичних даних	12
2 Розробка та тестування веб-сервісу	19
2.1 Підготовка до розробки RESTful сервісів	19
2.2 Розробка веб-сервісу при використанні обраних інформаційних технологій	20
2.2.1 Розробка RESTful сервісу мовою Java при допомозі Jax-RS	20
2.2.2 Розробка RESTful сервісу на PHP при допомозі Phalcon	26
2.2.3 Розробка RESTful сервісу на JavaScript за допомогою NodeJS	34
2.3 Тестування застосунку при використанні можливостей технології AngularJS	38
3 Безпека життєдіяльності, основи охорони праці	49
3.1. Загальні вимоги безпеки з охорони праці для користувачів ПК	49
3.2 Запобігання виникненню надзвичайних ситуацій	52
Висновки	56
Перелік джерел посилання	58
Додатки	61
Додаток А Диск	
Додаток Б Слайди презентації	
Додаток В Відгук та рецензія	

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота бакалавра містить: с., рис., табл., 26 джер.

ПРЕДМЕТНА ОБЛАСТЬ, МЕТА, ОБ'ЄКТ, ЗАВДАННЯ, ТЕХНОЛОГІЯ
РОЗРОБКИ, ВЕБ-СЕРВІС, РЕАЛІЗАЦІЯ, СЦЕНАРІЙ, ТЕСТУВАННЯ

За мету кваліфікаційної роботи взято розробку алгоритму визначення оптимальності сервісів за допомогою яких здійснюється передача даних в web-системах та сценаріїв їх тестування при використанні сучасних інформаційних технологій.

В роботі представлено аналіз предметної області, обґрунтовано вибір технології розробки веб-сервісів, виконано реалізацію програмного комплексу веб-сервісів та веб-додатку для тестування, розроблено сценарії тестування REST сервісів, проведено тестування веб-сервісів.

SUBJECT DOMAIN, PURPOSE, OBJECT, TASKS, DEVELOPMENT
TECHNOLOGY, WEB SERVICE, IMPLEMENTATION, SCENARIO, TESTING

The purpose of the qualification paper is to develop an algorithm for determination of the optimality of data transmission services in web systems and their testing scenarios with using modern information technologies.

In the paper presents the analysis of the subject domain, substantiated the choice of web services development technology, realized the software complex of web services and web application for testing, developed the scenarios for testing RESTservices, testing of web services was conduct.

ПЕРЕЛІК СКОРОЧЕНЬ

Java – об'єктно-орієнтована мова програмування

PHP – скриптова мова програмування

JavaScript – динамічна, об'єктно-орієнтована мова програмування

HTTP – hyper text transfer protocol (протокол передачі даних)

MIME – multipurpose internet mail extensions (комплексні розширення для інтернет-пошти)

XML – extensible markup language (розширювана мова розмітки)

JSON – JavaScript Object Notation (текстовий формат обміну даними між комп'ютерами)

Веб-сервер – сервер, що приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою, зображенням, файлом, медіа-потокком або іншими даними

Apache_CFX – відкритий каркас для веб-сервісів Service Oriented Architectures (SOA), що працює з декількома стандартними протоколами

Apache – відкритий веб-сервер Інтернет для UNIX-подібних, Microsoft Windows, Novell NetWare та інших операційних систем

NodeJS – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript

Веб-сервіс – програмна система, що ідентифікується URI, та публічні інтерфейси та прив'язки якої визначені та описані мовою XML

REST – Representational State Transfer (підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів)

URL – uniform resource locator (стандартизована адреса певного ресурсу)

БД – база даних

СКБД – система керування базами даних

ІС – інформаційна система

ПК – персональний комп'ютер

IP – internet protocol (Інтернет протокол)

W3C – world wide web consortium (головна міжнародна організація, що розробляє й впроваджує технологічні стандарти для всесвітньої павутини)

usability – зручність використання

EOM – електронна обчислювальна машина

Анотації – форма метаданих, яка не є частиною програми, але надають інформацію про програму

sURL – назва проекту і крос-платформового програмного засобу, що служить для передачі даних через Інтернет

Хостинг – послуга, що надає дисковий простір для розміщення фізичної інформації на сервері, що постійно перебуває в мережі

FTP – File Transfer Protocol (дає можливість абоненту обмінюватися двійковими і текстовими файлами з будь-яким комп'ютером мережі)

FTP-клієнт – програма для спрощення доступу до FTP серверу

3MCT

ВСТУП

Так склалось життя, що наше покоління зростає та встановлюється в буремний час, коли, без перебільшення, вершиться історія, коли два-три покоління кладуть свої голови за майбутнє і не лише своєї родини, а й цілої нації. Я цікавлюсь історією, і можу твердо говорити про те, що світ такої зухвалої жорстокості не бачив і навіть не чув ще напевно з часів середньовіччя. Сьогодні не може бути тих для кого немає значення, оскільки це питання виживання і в кожного свій фронт, хтось безпосередньо на полі бою дає відсіч ворогові, а нам випала доля воювати тут. Власним вкладом в наближення перемоги я вважаю своє навчання і отримання знань, які я зможу використати задля допомоги та втілення в життя та процвітання моєї країни.

Не дивлячись на ситуацію в Україні сьогодні, вважаю, що це тимчасово і потрібно і надалі йти вперед і розвиватись. З тієї допомоги, яку отримує моя країна, порівнюючи з тим, що нам було відомо про достатньо посередній стан речей очевидним є той факт, що темпи розвитку інструментів для розробників програмного забезпечення перевершують усі мислимі і немислимі прогнози. Тенденція така, що все те, що ще вчора могло здаватись якоюсь даниною моді, сьогодні вже можна вважати за невід'ємну частину повсякденного життя. І я глибоко та щиро переконаний у тому, що якщо хочеш залишатися бодай на місці потрібно завжди йти хоча би на один крок вперед. Заразом із тим в мене чомусь весь час якимсь таке відчуття, якоїсь може навіть і хаотичності настільки бурхливого розвитку, оскільки акценти та напрямки його весь час змінюються під впливом рандомних чинників.

Отже про мою роботу. Предметом мого дослідження я обрав технології створення REST, що означає собою ряд архітектурних принципів проектування web-сервісів, які є орієнтованими на системні ресурси, включаючи в себе способи обробки та передачі станів тих ресурсів по HTTP при використанні різноманітних клієнтських додатків, написаних на різних мовах програмування.

Актуальність ж розробки зумовлена значною увагою до прогресивних інформаційних веб-технологій на основі спеціалізованих програмних рішень з використанням середовищ програмування, СКБД та методики обміну даними за допомогою JSON-структур.

Мета дослідження – порівняти технології розробки веб-сервісів за певними критеріями.

Мета роботи визначила необхідність вирішення наступних задач:

- обрати набір використаних технологій;
- розробити набір проектних рішень для розробки програмного комплексу;
- реалізувати підходи до програмної побудови інформаційних веб-сервісів;
- провести тестування реалізованих програмних продуктів.

Методи дослідження. Сучасні методи проектування систем інформаційного типу, технології веб-програмування та технології обміну даними з використанням СКБД.

Новизною проведених досліджень стало те, що розроблено та реалізовано REST сервіси для веб-систем та веб-додаток для тестування.

1 ФОРМУЛЮВАННЯ ТА ПОСТАНОВКА ЗАДАЧІ

Представлені в кваліфікаційній бакалаврській роботі [1-7] матеріали дослідження показують обрані оптимальні, на мою думку, варіанти використання RESTful сервісів для різних веб-систем. Для цього реалізовано сервіси, які мають управління інформацією про клієнтів системи.

1.1 Аналіз предметної області та висновки про об'єкт реалізації

Завжди спочатку, перед тим, як приступати до виконання аналізується предметна область, в результаті чого я і зробив висновки про необхідність реалізації наступних функціональних можливостей розроблюваного веб-сервісу:

- отримання інформації про клієнта;
- отримання інформації про всіх клієнтів;
- створення нового клієнта;
- зміна інформації про клієнта;
- видалення інформації про клієнта;
- отримання повідомлення про статус дії сервісу.

Згідно правил побудови представлення предметної області, в сутності я виділив наступні основні характеристики, зокрема: ля клієнта виділив наступні атрибути:

- прізвище;
- ім'я;
- номер телефону;
- електронна адреса;
- фізична адреса;
- ідентифікатор контракту;
- термін придатності контракту.

Вибір технологій реалізації RESTful сервісів для порівняльного дослідження винесено в окремий пункт підрозділу 1.3.

1.2 Моделювання словника сервісу

Для розробки будь-якого програмного забезпечення завжди потрібно створити словник з основними термінами. В рамках представленої роботи дослідження оптимальності REST сервісів для веб-систем створених на різних мовах програмування будуть використані основні терміни для розробки програм, використання сервера та використання середовища розробки, зокрема:

для розробки програм потрібні наступні терміни:

- Customer – модель, яка представляє запис у таблиці Customers бази даних;
- REST path – відносне URI (запит), яке у відповідь, в залежності від типу запиту, поверне дані, створить, оновить або видалить конкретний запис в конкретній таблиці бази даних;
- RESTful сервіс – REST веб-сервіс.

для використання сервера потрібні наступні терміни:

- Запит – формулювання інформаційної потреби;
- Відповідь – результат, який повертає сервер, виконаний на конкретний запит.

для використання середовища розробки:

- Плагін – є додатком, незалежно скомпільованим програмним модулем, який динамічно підключається до середовища розробки, та який призначено для розширення чи використання можливостей.

1.3 Вибір технологій реалізації веб-сервісу через дослідження рейтингів та статистичних даних

Для вибору технологій для розробки було проведено пошук статистичної інформації, зокрема, які мови програмування зазвичай використовуються розробниками і вже відповідно до цих мов програмування знайдено популярні двигуни, які мають можливість реалізації REST сервісів [8].

За основу я взяв у вільному доступі результати статистичні дані результатів опитування. Я навмисне обрав дані не останньої «свіжості», оскільки цікавим для мене було дослідити динаміку, і для роботи таким чином можна спостерігати розвиток та тенденцію.

В опитуванні погодились прийняти участь 5905 осіб, із яких близько 93% проживали в Україні (станом на довійськовий стан). На рисунку 1.1 показано графік відсоток використання в своїй роботі розробниками мов програмування.

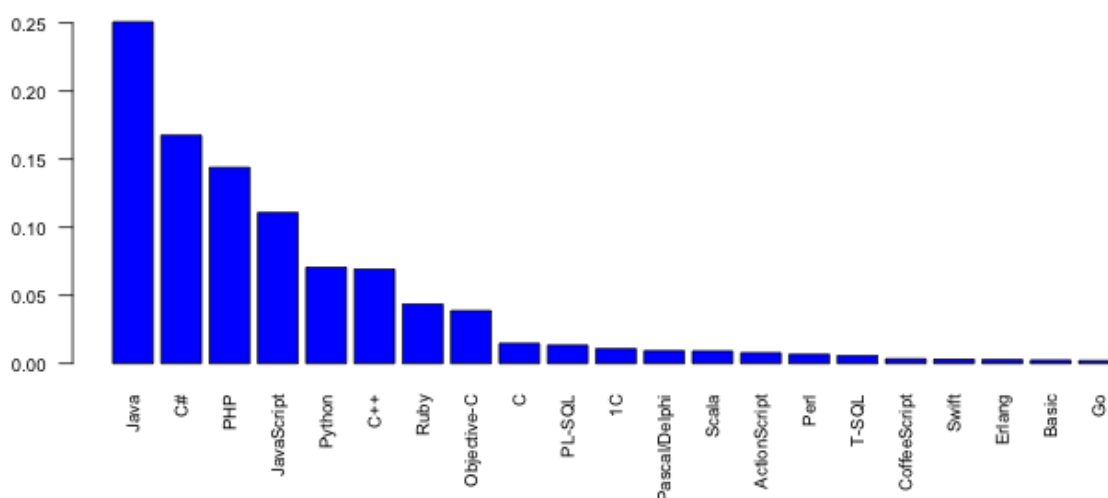


Рисунок 1.1 – Графік використання розробниками в роботі мов програмування [8]

З графіку чітко видно, що українські програмісти найчастіше віддають перевагу мові Java, відрив від C# досить великий, хоча синтаксисом ці мови програмування досить подібні між собою [9].

За підсумками статистики попередніх років (див. рисунок 1.2), стає зрозуміло, що застосування мови Java підвищилося зовсім незначно, а основним «винуватцем» у збільшенні розриву є власне продовження пониження використання мови C#. Кількість користувачів JavaScript, стрімко зростає, виводячи її на 4-е місце.

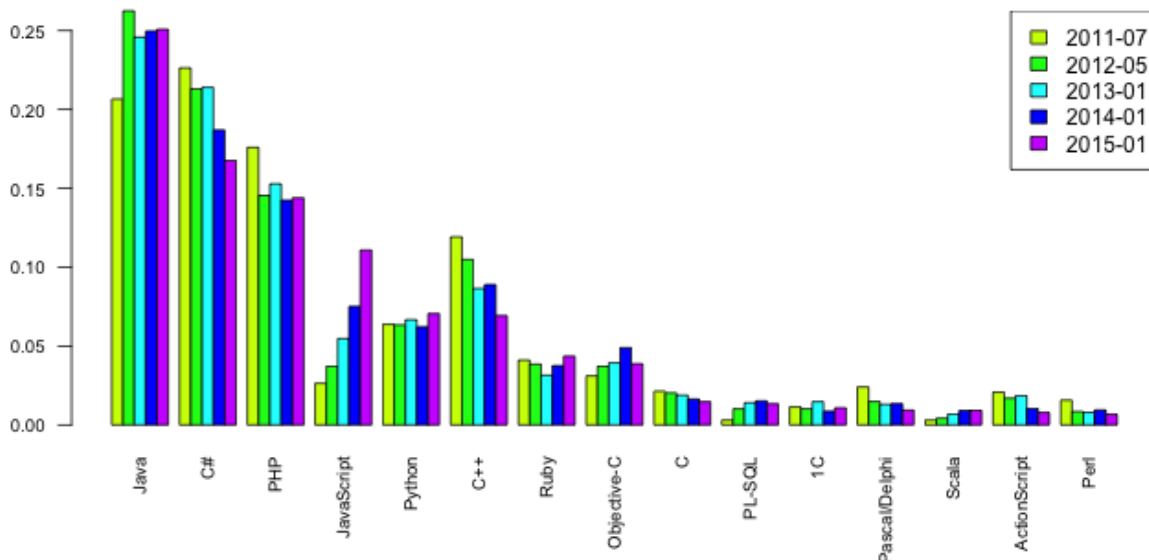


Рисунок 1.2 – Порівняльний графік історії використання розробниками в роботі мов програмування [8]

Однак, буде несправедливим використовувати лише цю статистику. Я знайшов і свіжі статистичні дані, і щоб бути об'єктивним наведу їх у роботі. Після вивчення, я виявив цікаві нюанси, зокрема відповідно до рейтингу від ТЮВЕ найпопулярнішою мовою виявилась мова Python. А тенденція до зростання популярності через частоту запитів, яка збільшилась на 4,62% у C++ (див.рис.1.3).

Дещо відрізняються статистичні дані, наведені за рейтингом PYPL. Зокрема тут Python є найпопулярнішою мовою у світі. А за останні п'ять років вона проявила один із найбільших приростів популярності (до 7,8%), найбільше ж втрат зазнала мова Java (близько -5,2 %) (див.рисунок 1.4).

Та ще цікавіше стало, коли я почав вивчати рейтинг від Stack Overflow. Тут, відповідно до проведеного опитування вияснилось, що все таки вже 10 років, як

вважається, найпопулярнішою мовою програмування все таки залишається Java Script, а все тому, що її найбільше використовують, а отже і вивчають (див.рисунок 1.5).


Індекс TIOBE					
Січ 2023	Січ 2022	Змінення	Мова програмування	Рейтинг	Змінення
1	1		 Python	16.36%	+2.78%
2	2		 C	16.26%	+3.82%
3	4	▲	 C++	12.91%	+4.62%
4	3	▼	 Java	12.21%	+1.55%
5	5		 C#	5.73%	+0.05%
6	6		 Visual Basic	4.64%	-0.10%
7	7		 JavaScript	2.87%	+0.78%
8	9	▲	 SQL	2.50%	+0.70%
9	8	▼	 Мова асемблера	1.60%	-0.25%
10	11	▲	 PHP	1.39%	-0.00%

Рис. 1.3 – Статистика популярності мов програмування відповідно до рейтингу від TIOBE [9]

Індекс PYPL				
Місце	Змінення	Мова програмування	Частка	Тенденція
1		 Python	27.93 %	-0.9 %
2		 Java	16.78 %	-1.3 %
3		 JavaScript	9.63 %	+0.5 %
4	▲	 C#	6.99 %	-0.3 %
5	▼	 C/C++	6.9 %	-0.5 %
6		 PHP	5.29 %	-0.8 %
7		 R	4.03 %	-0.2 %
8	▲▲▲	 TypeScript	2.79 %	+1.0 %
9		 Swift	2.23 %	+0.3 %
10	▼▼	 Objective-C	2.2 %	-0.1 %

Рис. 1.4 – Статистика популярності мов програмування відповідно до рейтингу від PYPL [9]

Мова	Частка
JavaScript	65,36%
HTML/CSS	55,08%
SQL	49,43%
Python	48,07%
TypeScript	34,83%
Java	33,27%
Bash/Shell	29,07%
C#	27,98%
C++	22,55%
PHP	20,87%

Рис. 1.5 – Статистика популярності мов програмування відповідно до рейтингу від Stack Overflow [9]

Таким чином для порівняльного дослідження я вибрав три різно-типових мови програмування: Java, PHP та JavaScript [10].

Java – вважається об’єктно-орієнтованою мовою програмування. Для вихідного продукту код потрібно компілювати.

Мова програмування Java в першу чергу призначена для написання програм, які «вміють» безперебійно функціонувати на будь-яких платформах та при будь-якому навантаженні. Основна увага в мові програмування Java була приділена саме ранньому виявленню усіх можливих помилок, динамічній перевірці (безпосередньо під час роботи програми), мінімізації чи навіть виключенню ситуацій, які можуть призвести до помилок.

Єдиною, на мою думку, значною відмінністю між мовами Java та C++ є в модель вказівників, прийнята в мові програмування Java, якою виключається можливість перезаписування ділянки пам’яті, так само як і пошкодження даних.

Ця властивість дуже важлива. Компілятор мови програмування Java дає змогу виявляти такі помилки, які в інших мовах програмування можна виявити лише виключно на етапі виконання програми. Окрім того, програмісти, витративши багато часу на пошук помилки, що викликає пошкодження пам'яті через неправильний вказівник, позбудуться подібної проблеми в Java, оскільки з цією мовою програмування такі проблеми не виникають [11, 12].

Програміст, пишучи на Java, може не переживати про витік даних. За це відповідає так званий «Garbage collector».

Популярною технологією розробки веб-сервісів для Java є JAX-RS.

JAX-RS визначає анотації, за допомогою яких можна отримати URI ресурсу на сервері, який метод HTTP використовуватиметься. Кожен клас, який є ресурсом, повинен мати, принаймні, одну з цих.

PHP – як скриптова мова програмування зі своїми особливостями, зокрема, особливістю мови програмування PHP є те, що для виконання PHP-скриптів потрібен інтерпретатор. Інтерпретатор складається, в загальному, з ядра та модулів, так званих «розширень», які представляють собою бібліотеки динамічного типу. Розширення дають змогу доповнювати базові можливості мови, надають можливість роботи як і з базами даних, так і з сокетом, графікою динамічного типу, криптографічними бібліотеками, документами в форматі .pdf і таким подібним. Існує велика кількість розширень: стандартні, створені третьою стороною (компаніями) та небайдужими ентузіастами, проте до стандартної поставки входять лише декілька десятків, при чому які добре себе зарекомендували [13].

Популярним двигуном розробки веб-сервісів для PHP є Phalcon [14].

Phalcon має наступні характеристики:

- Всі компоненти повністю написані на мові програмування C;
- Існують версії для різних популярних операційних систем: Linux, Windows, Mac;
- Висока швидкість роботи, малі витрати серверних ресурсів. Судячи з тестів, один з найпродуктивніших фреймворків для PHP [15];

- Компоненти фреймворку слабо пов'язані між собою;
- Взаємодія з базами даних реалізовано на С за технологією ORM.

JavaScript – є динамічною та прикладом об'єктно-орієнтованої мови програмування [16-18].

У JavaScript будь-який об'єкт представлений хеш-таблицею, а кожна властивість об'єкта, включаючи ім'я методу, це ключ хеш-таблиці, тому доступ до властивостей відбувається порівняно швидко. Але якщо необхідно зберігати послідовний набір даних не по ключах, то використовуються масиви, оскільки швидкість послідовного перебору елементів у ньому значно вище, якщо ж необхідні два методи доступу до об'єктів (по ключу і послідовно), то ніхто не забороняє додати один і той самий об'єкт у хеш-таблицю і масив. Практично для всіх операцій роботи з масивами можна використовувати метод. Масиви в JavaScript мають вбудовану реалізацію стека і черги.

Крім іншого в JavaScript досить незвична модель наслідування – прототипна. Кожен об'єкт в JavaScript має прототип (який теж є об'єктом і знаходиться по властивості `__proto__`), і якщо в об'єкта спробувати яку-небудь властивість або метод і об'єкт його не має, ця властивість буде запитана у його прототипу, далі при необхідності у його прототипу і т.д. У підсумку ієрархію спадкування можна представити як Linked List складається з хеш-таблиць, по якому проходить інтерпретатор, перевіряючи, чи є заданий ключ в черговий хеш-таблиці і якщо він його знайшов, то повертає його, якщо немає – повертає `undefined`. Наприкінці ланцюгу прототипів завжди виявляється порожній об'єкт, який буде означати кінець пошуку.

В рамках даного дослідження мною було використано сервер NodeJS [19].

NodeJS в основному призначено для відокремленого виконання мовою JavaScript високопродуктивних мережних застосунків. Функціонально платформа не обмежується лише створенням серверних скриптів для web, платформу можна використовувати і для створення звичайних клієнтоорієнтованих так і серверноорієнтованих мережних програм. З метою забезпечення реалізації JavaScript коду кристуються розроблени компанією Google рушієм типу V8.

Для забезпечення обробки паралельних запитів у великій кількості за допомогою NodeJS виконується асинхронна модель запуску кода, суттю якої є обробка подій в «не блокуючому» режимі і визначення обробників зворотних викликів (англ. call back). В якості способів для мультиплексування з'єднань `epoll`, `kqueue`, `/dev/poll`, а також `select`. Зазвичай з метою мультиплексації з'єднань використовують бібліотеку `libev`, а для створення `thread pool` (українською пул ниток) задіюють бібліотеку `libeio`, а для виконання запитів DNS у «не блокуючому» режимі інтегрований `c-ares`. Разом із тим, усі системні виклики, які можуть спричинити блокування, виконують всередині `thread pool` і вже після цього, як і в обробниках сигналів, підсумок всієї своєї роботи повертається назад через найменовані канали (англ. `pipe`).

По своїй суті NodeJS подібний до рушіїв типу Perl AnyEvent, RubyEvent Machine та PythonTwisted, проте з відмінністю: цикл по обробці подій (англ. `event loop`) у NodeJS приховано від розробника і він нагадує опрацювання подій у веб-застосунку, якій працює у браузері. Також, при створенні програм для NodeJS варта враховувати і специфіку подієво-орієнтованого програмування.

Таким чином в першому розділі я висвітлив та описав призначення REST сервісу. В першому пункті наведено можливості, які повинен виконувати веб-сервіс. Описано модель представлення інформації в базі даних та детально описано її атрибути.

У другому пункті складено основні терміни: для реалізації програми, для використання сервера та використання середовища розробки.

У третьому проведений вибір технологій реалізації веб-сервісу. Вибір оснований на статистичному опитуванні. Інформація використана із Інтернет джерела. До обраних мов програмування підібрані технології, які можуть реалізувати REST сервіс. Описано технології розробки та їх особливості.

2 РОЗРОБКА ТА ТЕСТУВАННЯ ВЕБ-СЕРВІСУ

2.1 Підготовка до розробки RESTful сервісів [20, 21]

Сервіс буде працювати з однією таблицею в БД (як на рисунку 2. 1 та лістинг 2. 1), додавати, змінювати, видаляти, та отримувати записи списком або по Id.

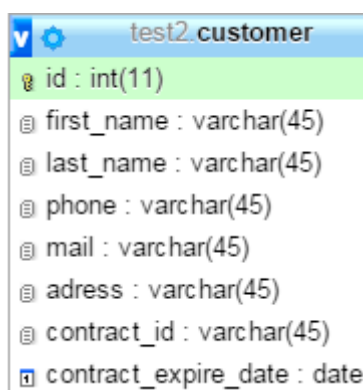


Рисунок 2.1 – Схема таблиці «Customer»

Лістинг 2.1 – Код MySQL для створення таблиці «Customer»

```
CREATE TABLE `customer` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) DEFAULT NULL,
  `last_name` varchar(45) DEFAULT NULL,
  `phone` varchar(45) DEFAULT NULL,
  `mail` varchar(45) DEFAULT NULL,
  `adress` varchar(45) DEFAULT NULL,
  `contract_id` varchar(45) DEFAULT NULL,
  `contract_expire_date` date DEFAULT NULL,
  PRIMARY KEY (`id`)
);
```

Сервіс повинен мати можливість обслуговування параметризованих запитів на отримання списку записів, налаштування маршрутів із «читабельними» URL адресами та керувати кодами помилок у відповідях від сервера.

Для REST сервісу запроектовано можливі запити, які обслуговуються при його використанні (див. таблицю 2. 1)

Таблиця 2. 1 – Представлення запитів, обслуговуваних REST сервісом

Тип запиту	Відносна адреса	Опис
GET	/service/customer	отримання списку записів
GET	/service/customer/{id}	отримання інформації про клієнта по його ідентифікатору
POST	/service/customer	додавання нової інформації по клієнту до таблиці у базі даних
PUT	/service/customer/{id}	зміна інформації про клієнта в базі даних по його ідентифікатору
DELETE	/service/customer/{id}	видалення інформації стосовно клієнта з таблиці у базі даних по його ідентифікатору

Основні коди статусів, які буде повертати сервіс у відповідь на запит:

2** – Успішно;

3** – Перенаправлення;

4** – Помилка клієнта;

5** – Помилка сервера.

Прикладом успішного завершення запиту є статус-коди: 200 – Success, 201 – Created.

2.2 Розробка веб-сервісу при використанні обраних інформаційних технологій

2.2.1 Розробка RESTful сервісу мовою Java при допомозі Jax-RS [22]

Для реалізації RESTful сервісу на Java необхідні наступне програмне забезпечення:

- JDK 1.6 – комплект розробника застосунків на мові Java, до якого входить компілятор Java(javac), а також стандартизовані бібліотеки класів типу Java, приклади, документація, різні утиліти та виконавча система Java(JRE);

- Apache_CXF – відкритий каркас для веб-сервісів Service Oriented Architectures (SOA), що працює з декількома стандартними протоколами, включаючи SOAP, WSDL 1.1 і 2.0, WS-адресацію, WS- Policy, WS- Reliable Messaging і WS- Security;
- Spring 3 Framework JDBC – це програмний двигун з відкритим кодом та контейнером з підтримкою інверсії керування під платформу Java;
- Apache Tomcat 7.0 – є контейнером сервлетів, який розроблено Apache SoftwareFoundation. Даний контейнер є повністю написаний мовою програмування Java та яким реалізується специфікація сервлетів та Java ServerPages від SunMicrosystems, що вважається стандартами власне для розробки web-застосунків на Java мові;
- MySQL 5.1 – є вільною системою керування базами даних реляційного типу;
- Eclipse 4.5 Mars – є вільним модульним інтегрованим середовищем що використовується при розробці програмного забезпечення;
- Maven 3.0 – є засобом що забезпечує автоматизацію роботи з програмними проектами для керування та безпосереднього складання програм.

ПЗ вибиралося за дуже простим принципом – чим простіше, тим краще.

Для розробки сервісу, спочатку створено клас, який буде обслуговувати запити (див. лістинг 2.1).

Лістинг 2.1 – Веб-сервіс із основними властивостями:

```
public class CustomersServiceJSON implements ICustomersService
{
    private ICustomersDAO customersDAO;
    public ICustomersDAO getCustomersDAO() {
        return customersDAO;
    }
    public void setCustomersDAO(ICustomersDAO customersDAO) {
        this.customersDAO = customersDAO;
    }
}
```

```

    @Context
    private HttpHeaders requestHeaders;
    private String getHeaderVersion() {
        return requestHeaders.getRequestHeader("version").get(0);
    }
    ...
}

```

Для обслуговування запитів, використовуються спеціальні анотації для методів. Для виконання дії на запит типу GET, до метода сервісу дописується анотація @GET. Для налаштування шляху, до метода сервісу дописується анотація @Path(routeName). Щоб використовувати параметри запиту, в методі окрім вказування типу формальної змінної, дописується анотація @PathParam(paramName). У лістингу 2.2 представлена реалізація обслуговування GET запитів.

Лістинг 2.2 – Обслуговування GET запитів

```

...
@GET
@Path("/{id}")
public Response getCustomer(@PathParam("id") String id) {
    Customer customer = customersDAO.getCustomer(id);
    if (customer != null) {
        return ResponseCreator.success(getHeaderVersion(),
customer);
    } else {
        return ResponseCreator.error(
            404, Error.NOT_FOUND.getCode(), getHeaderVersion()
        );
    }
}

@GET
public Response getCustomers(
    @QueryParam("keyword") String keyword,
    @QueryParam("orderBy") String orderBy,
    @QueryParam("order") String order,
    @QueryParam("pagenum") Integer pageNum,
    @QueryParam("pagesize") Integer pageSize
) {
    CustomerListParameters parameters = new
CustomerListParameters();
    parameters.setKeyword(keyword);
    parameters.setPageNum(pageNum);
    parameters.setPageSize(pageSize);
    parameters.setOrderBy(orderBy);
}

```

```

        parameters.setOrder(Order.fromString(order));
        List<Customer> listCust =
customersDAO.getCustomersList(parameters);
        if (listCust != null) {
            GenericEntity<List<Customer>> entity = new
GenericEntity<List<Customer>>(listCust) {};
            return ResponseCreator.success(getHeaderVersion(), entity);
        } else {
            return ResponseCreator.error(
                404, Error.NOT_FOUND.getCode(), getHeaderVersion()
            );
        }
    }
    ...

```

Для обслуговування запитів типу DELETE (видалення кортежів із таблиці БД) використовується анотація `@DELETE` (див. лістинг 2.3).

Лістинг 2.3– Реалізація обслуговування DELETE запиту

```

...
@DELETE
@Path("/{id}")
public Response removeCustomer(@PathParam("id") String id) {
    if (customersDAO.removeCustomer(id)) {
        return ResponseCreator.success(getHeaderVersion(),
"removed");
    } else {
        return ResponseCreator.success(getHeaderVersion(), "no
such id");
    }
}
...

```

Для обслуговування запитів типу POST (додавання нового кортежу в таблицю БД) використовується анотація `@POST`. Для POST запиту також потрібно вказати, що разом із запитом надходять дані. Для цього використовується анотація `@Consumes(dataType)` (див. лістинг 2.4).

Лістинг 2.4 – Реалізація обслуговування POST запиту

```

...
@POST
@Consumes(MediaType.APPLICATION_JSON)
public Response createCustomer(Customer customer) {
    Customer creCustomer = customersDAO.createCustomer(customer);
    if (creCustomer != null) {

```

```

return ResponseCreator.success(getHeaderVersion(), creCustomer);
    } else {
        return ResponseCreator.error(
            500, Error.SERVER_ERROR.getCode(), getHeaderVersion()
        );
    }
}
...

```

Для обслуговування запитів типу PUT (зміни кортежу в таблиці БД) використовується анотація `@PUT` (див. лістинг 2.5).

Лістинг 2.5 – Реалізація обслуговування PUT запиту

```

...
@PUT
@Consumes(MediaType.APPLICATION_JSON)
public Response updateCustomer(Customer customer) {
    Customer updCustomer = customersDAO.updateCustomer(customer);
    if (updCustomer != null) {
        return ResponseCreator.success(getHeaderVersion(),
updCustomer);
    } else {
        return ResponseCreator.error(
            500, Error.SERVER_ERROR.getCode(), getHeaderVersion()
        );
    }
}
...

```

У коді можуть виникнути помилки, спричинені декількома винятками: користувач надсилає невірні дані, конфлікт виконання завдання, немає прав або помилка сервера. Для цього реалізовано обробник помилок (див. лістинг 2.6).

Лістинг 2.6 – Реалізація обробника помилок

```

public class CustomExceptionHandler implements
ExceptionHandler<Exception> {
    @Context
    private HttpHeaders requestHeaders;

    private String getHeaderVersion() {
        return requestHeaders.getRequestHeader("version").get(0);
    }

    public Response toResponse(Exception ex) {
        return ResponseCreator.error(
            500, Error.SERVER_ERROR.getCode(), getHeaderVersion()
        );
    }
}

```

Основний інтерес представляє підключення action-servlet від Apache-CXFServlet і стандартного Spring ContextLoaderListener. Для цього потрібно налаштувати конфігураційний файл web.xml (див. лістинг 2.7).

Лістинг 2.7 – Реалізація конфігурації web.xml

```
...
<web-app>
  <display-name>service</display-name>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/beans.xml</param-value>
  </context-param>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <servlet>
    <servlet-name>CXFServlet</servlet-name>
    <display-name>CXF Servlet</display-name>
    <servlet-class>
      org.apache.cxf.transport.servlet.CXFServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CXFServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Для підключення CFX конфігураційних файлів, DAO об'єктів, обробника помилок і сам Java-bean з сервісами використовується файл beans.xml (див. лістинг 2.8).

Лістинг 2.8 – Beans конфігурація

```

...
<!-- Imported resources for cxf -->
<import resource="classpath:META-INF/cxf/cxf.xml" />
<import resource="classpath:META-INF/cxf/cxf-extension-jaxrs-
binding.xml" />
<import resource="classpath:META-INF/cxf/cxf-servlet.xml" />

<!-- Imported bean for dao -->
<import resource="classpath:META-INF/spring/dao.xml"/>

<bean id="customersService"

class="com.test.services.customers.rest.CustomersServiceJSON">
    <property name="customersDAO" ref="customersDAO"/>
</bean>

<bean id="preInvokeHandler"
    class="com.test.services.rest.PreInvokeHandler" />
<bean id="customExceptionHandler"
    class="com.test.services.rest.CustomExceptionHandler" />
<jaxrs:server id="restContainer" address="/customer">
    <jaxrs:serviceBeans>
        <ref bean="customersService" />
    </jaxrs:serviceBeans>
    <jaxrs:providers>
        <ref bean="preInvokeHandler" />
        <ref bean="customExceptionHandler" />
    </jaxrs:providers>
</jaxrs:server>
...

```

Тестування зазначеного сервісу описано в пункті 2.3.

2.2.2 Розробка RESTful сервісу на PHP при допомозі Phalcon

Для розробки простого обслуговування сервісу не потрібно включати ціле середовище MVC Phalcon. В такому випадку використовується мікро-додаток.

Такої структури файлів буде достатньо:


```
my-rest-api/
  models/
    Customers.php
    index.php
  .htaccess
```

Для початку потрібно прописати конфігурації перенаправлення URI на файл index.php (див. лістинг 2.9).

Лістинг 2.9 – Конфігурація перенаправлення URI на файл index.php

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^((?s).*)$ index.php?url=/$1 [QSA,L]
</IfModule>
```

Після цього створюється index.php із референцією на мікро-додаток (див. лістинг 2.10).

Лістинг 2.10 – Реалізація мікро-додатку

```
<?php
use Phalcon\Mvc\Micro; \\ розширення області пам'яті
$app = new Micro();

// тут будуть описані маршрути
$app->handle();
```

Відповідно до вимог, прописуються маршрути (див. лістинг 2.11).

Лістинг 2.11 – Конфігурація маршрутів RESTful сервісу

```
<?php
...
// Отримання списку всіх кортежів таблиці Customers
$app->get( '/api/customers', function () { ... });

// Отримання кортежу таблиці Customers по вказаному ключеві
$app->get( '/api/customers/{id:[0-9]+}', function ($id) { ... });

// Додавання нового кортежу в таблицю Customers
$app->post( '/api/customers', function () { ... });

// Зміна кортежу по вказаному ключу в таблиці Customers
$app->put( '/api/customers/{id:[0-9]+}', function () { ... });

// видалення кортежу по вказаному ключу із таблиці Customers
```

```
$app->delete( '/api/customers/{id:[0-9]+}', function () { ... });
$app -> handle();
```

Кожен маршрут заданий за допомогою методу з такою ж назвою як і HTTP метод. В якості першого параметру передається шаблон маршруту, другим – обробник, який, в нашому випадку є анонімною функцією. Такий маршрут як «/api/customers/ {id: [0-9] +}» однозначно встановлює за допомогою регулярного виразу, що параметр id повинен бути числом.

API представляє інформацію про «клієнтів», яка зберігається в базі даних. Описана нижче модель (див. лістинг 2.12) дозволяє отримати доступ до таблиці об'єктно-орієнтованим шляхом.

Лістинг 2.12 – Реалізація моделі Customer

```
<?php
use Phalcon\Mvc\Model;
use Phalcon\Mvc\Model\Message;
use Phalcon\Mvc\Model\Validator\Uniqueness;
use Phalcon\Mvc\Model\Validator\InclusionIn;

class Customers extends Model
{
    public function validation()
    {
        // тут пишуться обробники
        ...

        // перевіряє, були отриманні якісь повідомлення
        при валідації

        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

Реалізовано небагато бізнес-правил, використовуючи валідатори Phalcon з найпростішими перевітками. Виконано це для того, щоб бути впевненим, що дані, які зберігаються в БД відповідають вимогам додатку.

Тепер потрібно налаштувати підключення з базою даних, щоб використовувати його в цій моделі (див. лістинг 2.13).

Лістинг 2.13 – Конфігурація драйверу підключення до БД

```

<?php
use Phalcon\Loader;
use Phalcon\Mvc\Micro;
use Phalcon\DI\FactoryDefault;
use Phalcon\Db\Adapter\Pdo\Mysql as PdoMysql;

// Використовується Loader() для автозагрузки моделі
$loader = new Loader();

$loader->registerDirs(
    array(
        __DIR__ . '/models/'
    )
)->register();

$di = new FactoryDefault();

// Налаштування сервісу бази даних
$di->set('db', function () {
    return new PdoMysql(
        array(
            "host"      => "localhost",
            "username" => "root",
            "password" => "surprise",
            "dbname"   => "test2"
        )
    );
});

// Створюється і прив'язується DependencyInjection до додатку
$app = new Micro($di);

```

Для отримання даних спочатку реалізовується обробник, який відповідає на GET-запит і повертає інформацію про всіх клієнтів. Для виконання цієї задачі використовується RSQL, який повертає результат виконання простого запиту в форматі (див. лістинг 2.14).

Лістинг 2.14 – Реалізація обробника маршруту на отримання всіх клієнтів

```

<?php
...
// Отримання всіх клієнтів

```

```

$app->get('/api/customers', function () use ($app) {
    $sql = "SELECT * FROM Customers ORDER BY name";
    $robots = $app->modelsManager->executeQuery($sql);
    $data = array();
    foreach ($customers as $customer) {
        $data[] = array(
            'id' => $customer->id,
            'name' => $customer->first_name
        );
    }
    echo json_encode($data);
});

```

RHQL дозволяє писати запити за допомогою високорівневого, об'єктно-орієнтованого SQL-діалекту, які всередині нього будуть переведені в правильні SQL-оператори в залежності від використаної СУБД. «use» дозволяє передати деякі змінні із глобальної області видимості в локальну.

Обробник пошуку по ідентифікатору крім того, що вертає сам кортеж, так ще й повідомляє знайдений він чи ні (як представлено в лістингу 2.15).

Лістинг 2.15 – Реалізація обробника маршруту на отримання клієнта по ідентифікатору

```

<?php
use Phalcon\Http\Response;

// Отримання інформації про клієнта по ідентифікатору
$app->get('/api/customers/{id:[0-9]+}', function ($id) use ($app) {
    $sql = "SELECT * FROM Customers WHERE id = :id:";
    $customer = $app->modelsManager->executeQuery($sql, array(
        'id' => $id
    ))->getFirst();

    // Формуємо відповідь

```

```

$response = new Response();
if ($customer == false) {
    $response->setJsonContent(
        array(
            'status' => 'NOT-FOUND'
        )
    );
} else {
    $response->setJsonContent(
        array(
            'status' => 'FOUND',
            'data'    => array(
                'id'    => $customer->id,
                'name' => $customer->name
            )
        )
    );
}
return $response;
});

```

Для додавання даних використовується теж PHQL діалект (як представлено в лістингу 2.16).

Лістинг 2.16 – Реалізація обслуговування маршруту для додавання нового клієнта

```

<?php
use Phalcon\Http\Response;

// Додавання нового клієнту
$app->post('/api/customers', function () use ($app) {
    $cusomer = $app->request->getJsonRawBody();

    $phql = "INSERT INTO Customers (first_name, last_name, phone,
email, address, contract_id, contract_expire_date) VALUES
(:first_name:, :last_name:, :phone:, :email:, :address:,
:contract_id:, :contract_expire_date:)";
    $status = $app->modelsManager->executeQuery($phql, array(
'first_name' => $cusomer-> first_name,
    'last_name' => $cusomer-> last_name,
    'phone' => $cusomer->phone,
    'email' => $cusomer->email,
    'address' => $cusomer-> address,
    'contract_id' => $cusomer->contract_id,
    'contract_expire_date' => $cusomer->contract_expire_date
    ));

    // Формуєм відповідь
    $response = new Response();

```

```

// Перевіряємо, чи додавання пройшло успішно
    if ($status->success() == true) {
// обробник успішного додавання клієнта
    } else {
        // обробник невдалого додавання клієнта
    }
    return $response;
});

```

При додаванні нового об'єкту можуть бути 2 варіанти: або успішно додається, або конфлікт даних. Якщо успішно, то формується HTTP статус 201 (дивись лістинг 2.17), інакше формується HTTP статус 409 і повідомлення про помилку (дивись лістинг 2.18).

Лістинг 2.17 – Реалізація обробника успішного додавання нового клієнту

```

...
// Міняємо HTTP статус
$response->setStatusCode(201, "Created");
$customer->id = $status->getModel()->id;
$response->setJsonContent(
    array(
        'status' => 'OK',
        'data' => $customer
    )
);
...

```

Лістинг 2.18 – Реалізація обробника невдалого додавання нового клієнту

```

...
// Міняємо HTTP статус
$response->setStatusCode(409, "Conflict");
// Відправляємо повідомлення про помилку користувачу
$errors = array();
foreach ($status->getMessages() as $message) {
    $errors[] = $message->getMessage();
}
$response->setJsonContent(
    array('status' => 'ERROR', 'messages' => $errors)
);
...

```

Зміна клієнта аналогічна його додаванню. Отриманий параметр `id` повідомляє про те, яку інформацію про клієнта треба змінити (див. лістинг 2.19).

Лістинг 2.19 – Реалізація обслуговування маршруту для зміни клієнта

```
<?php
use Phalcon\Http\Response;

// Зміна кортежу по вказаному ключу в таблиці Customers
$app->put('/api/customers/{id:[0-9]+}', function ($id) use
($app) {
    $customer = $app->request->getJsonRawBody();
    $sql = "UPDATE Customers SET first_name = :first_name:,
last_name = :last_name:, phone = :phone:, email = :email:, address =
:address:, contract_id = :contract_id:, contract_expire_date =
:contract_expire_date: WHERE id = :id:";
    $status = $app->modelsManager->executeQuery($sql, array(

        'first_name' => $customer->first_name,
        'last_name' => $customer->last_name,
        'phone' => $customer->phone,
        'email' => $customer->email,
        'address' => $customer->address,
        'contract_id' => $customer->contract_id,
        'contract_expire_date' => $customer->contract_expire_date
    ));

    // Формуємо відповідь
    $response = new Response();

    // Перевіряємо, що оновлення пройшло успішно
    if ($status->success() == true) {
        $response->setJsonContent(
            array('status' => 'OK')
        );
    } else {

    // обробник невдалої зміни клієнта
    }
    return $response;
});
```

Видалення даних схоже до зміни. Отриманий параметр `id` повідомляє про те, яку інформацію про клієнта потрібно видалити (див. лістинг 2.19).

Лістинг 2.20 – Реалізація обслуговування маршруту для видалення клієнта

```

<?php
use Phalcon\Http\Response;

// видалення кортежу по вказаному ключу із таблиці Customers
$app->delete('/api/customers/{id:[0-9]+}', function ($id) use
($app) {
    $sql = "DELETE FROM Customers WHERE id = :id:";
    $status = $app->modelsManager->executeQuery($sql, array(
        'id' => $id
    ));

    // Формуємо відповідь
    $response = new Response();

    if ($status->success() == true) {
        $response->setJsonContent(
            array(
                'status' => 'OK'
            )
        );
    } else {

// обробник невдалого видалення
    }
    return $response;
});

```

Тестування зазначеного сервісу описано в пункті 2.3.

2.2.3 Розробка RESTful сервісу на JavaScript за допомогою NodeJS

Для реалізації RESTful сервісу на JavaScript необхідне наступну програмне забезпечення:

- NodeJS – є платформою, що має відкритий код, призначеною для здійснення високопродуктивних мережових застосунків, які написані на мові JavaScript;
- NodeJS модуль Mongoose – є крос-платформним веб-сервером від CesantaSoftware;

- NodeJS модуль Express – є рушієм, за допомогою якого можна організувати код у архітектурі MVC зі сторони сервера;
- MongoDB – є документо-орієнтованою системою управління базами даних із сирцевим кодом відкритого типу, для якої не потрібний опис схеми таблиць;
- WebStorm – є інтегрованим середовищем розробки для Java Script-a, HTML та CSS розробленим Jet Brains на основі платформи IntelliJ IDEA.

Для розробки сервісу, як говорилося вище, потрібні модулі NodeJS. Крім того потрібна модель, маршрути та основний файл, в кому буде бізнес-логіка. Для такого сервісу потрібна наступна структура файлів:

```

/models
  Customer.js
/node_modules
  /express
  /mongoose
/routes
  Customer.js
app.js

```

Для початку потрібно додати модулі до проекту. Для цього з каталогу сервісу відкривається консоль і вписуються наступні команди:

- `npm install express --save;`
- `npm install mongoose --save.`

Далі створюється основний файл, який буде запускатися як сервіс – `app.js` (див. лістинг 2.21).

Лістинг 2.21 – Реалізація основного файлу-двигуна сервісу із підключенням до бази даних

```

var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/service', function(err) {
  if(err) { console.log('Помилка з\'єднання', err); }
  else { console.log('З\'єднання успішне'); }
});

```

Тепер, при запуску команди із консолі `npm start app.js` побачимо повідомлення «Помилка зі з'єднанням» або «З'єднання пройшло успішно», в залежності від можливості підключення до бази даних.

Все вищеописане – це підготовка. Тепер потрібно створити модель `Customer` (див. лістинг 2.22). При імпортуванні файлу до основного файлу, створиться схема (не в базі даних). Це робить `mongoose`, щоб зберегти послідовну структуру моделі.

Лістинг 2.22 – Реалізація моделі `Customer`

```
var mongoose = require('mongoose');
var CustomerSchema = new mongoose.Schema({
  first_name: String,
  last_name: String,
  phone: String,
  email: String,
  address: String,
  contract_id: String,
  contract_expire_date: Date
});
module.exports = mongoose.model('Customer', CustomerSchema);
```

Після створення моделі створюються маршрути за допомогою `express` модуля (див. лістинг 2.23).

Лістинг 2.23 – Реалізація маршрутів `Customer` за допомогою `express` модуля

```
var express = require('express');
var router = express.Router();
var mongoose = require('mongoose');
var Customer = require('../models/Customer.js');

/* GET /customers */
router.get('/', function(req, res, next) {
  Customer.find(function (err, customers) {
    if (err) return next(err);
    res.json(customers);
  });
});
module.exports = router;
```

Для створення нового клієнта використовується метод `post` (див. лістинг 2.24).

Лістинг 2.24 – Реалізація маршруту для додавання нового клієнта

```
...
/* POST /customers */
router.post('/', function (req,res,next) {
  Customer.create(req.body, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
...
```

Для реалізація пошуку клієнта по ідентифікатору використовується метод моделі `findById` (див. лістинг 2.25).

Лістинг 2.25 – Реалізація маршруту для пошуку клієнта по його ідентифікатору

```
...
/* GET /customers/id */
router.get('/:id', function (req,res,next) {
  Customer.findById(req.params.id, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
...
```

Для реалізації оновлення даних клієнта по його ідентифікатору використовується метод моделі `findByIdAndUpdate` (див. лістинг 2.26).

Лістинг 2.26 – Реалізація маршруту для оновлення інформації про клієнта

```
...
/* PUT /customers/:id */
router.put('/:id', function (req,res,next) {
  Customer.findByIdAndUpdate(req.params.id, req.body, function
(err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
...
```

Для реалізації маршруту видалення інформації про клієнта по його ідентифікатору використовується метод моделі `findByIdAndRemove` (див. лістинг 2.27).

Лістинг 2.27 – Реалізація маршруту для видалення інформації про клієнта по його ідентифікатору

```
...
/* DELETE /customers/:id */
router.delete('/:id', function (req,res,next) {
  Customer.findByIdAndRemove(req.params.id, req.body, function
(err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
...

```

Тепер потрібно підключити вищеописані маршрути до основного файлу-двигуна (див. лістинг 2.28).

Лістинг 2.28 – Використання маршрутів в основному файлі-двигуні

```
...
var customersRoutes = require('./routes/customers');
app.use('/customers', customersRoutes);
...

```

2.3 Тестування застосунку при використанні можливостей технології AngularJS

Тести написані на JavaScript із використанням технології AngularJS. Тестування будуть включати в себе перевірку правильності функціонування REST сервісу та швидкодію надання відповіді. Для виводу інформації використано Chrome console.

Оскільки використовується технологія AngularJS, щоб швидко побудувати «скелет» аплікації, використовується `bower`, в якості загрузчика необхідних бібліотек та компонентів.

Для додатку достатньо мати наступну структуру файлів:

```

/bower_components
  /angular
    angular.min.js
  app.js
  index.html

```

Для початку мені потрібно інсталиувати AngularJS. Це я зробив за допомогою команди із командної стрічки `bower install angular --save`. Після цього в директорії `bower_components` з'явиться директорія `angular`, в якій є усі необхідні файли.

Для запуску тестів використано браузер, для якого в свою чергу потрібен HTML документ для висвітлення (див. лістинг 2.29).

Лістинг 2.29 – Реалізація основного HTML-документу для тестування

```

<!DOCTYPE html>
<html ng-app="mainApp">
<head>
  <meta charset="UTF-8" author="Ihor Staryk"/>
  <title>Тестування</title>
  <script
src="bower_components/angular/angular.min.js"></script>
  <script src="app.js"></script>
</head>
<body>
<div ng-controller="Tests"></div>
</body>
</html>

```

У лістингу 2.29 наведено приклад використання AngularJS, а саме використано:

- `ng-app="mainApp"` – директиву, за допомогою якої автоматично завантажує модуль `mainApp`;
- `ng-controller="Tests"` – для запуску контролера `Tests`.

У файлі `app.js` буде знаходитися «мозок» програми. Для початку потрібно оголосити модуль `mainApp`. Це я зробив за допомогою однієї лінії коду (див. лістинг 2.30). Далі потрібно створити налаштування, яким відповідно будуть здійснюватися запити до створених REST сервісів. Для цього потрібна мапа

(ключ-значення) шляхів. Використав AngularJS метод, за допомогою якого створив константу як `DependencyInjector` (див. лістинг 2.30).

Лістинг 2.30 – Створення налаштування запитів

```
angular.module("mainApp", [])

.constant("rest_config", {
  "urls": {
    "getAll": "http://localhost/java-rest/customers",
    "getById": "http://localhost/java-rest/customers/jr12-ffgh-CZ-22",
    "update": "http://localhost/java-rest/customers",
    "delete": "http://localhost/java-rest/customers/qxty-ffrz-CX-18"
  }
})
```

Окрім маршрутів, мені ще потрібно налаштувати тестові об'єкти, які будуть в якості моделі для додавання інформації про нового клієнта та оновлення інформації про вже існуючого клієнта (див. лістинг 2.31).

Лістинг 2.31 – Створення моделей для додавання та оновлення інформації про клієнтів

```
angular.module("mainApp", [])

.constant("rest_config", {
  "urls": {...},
  "data": {
    "create": {
      "first_name": "Герасим",
      "last_name": "Віталій",
      "phone": "0679505236",
      "mail": "vitaliygerasym@gmail.com",
      "address": "м. Тернопіль, вул. Руська буд. 56, кв. 104"
    },
    "update": {
      "id": "jr12-ffgh-CZ-22",
      "_id": "jr12-ffgh-CZ-22", // MongoDB identity
      "first_name": "Герасим",
      "last_name": "Анна",
      "phone": "0679505237",
      "mail": "annaherasym@gmail.com",
      "address": "м. Тернопіль, вул. Руська буд. 56, кв. 104"
    }
  }
})
```

```
    }
  })
```

Для тестування швидкодії використано технологію логування часу за допомогою об'єкта `console`. Для використання логування придатна наступна схема:

- `console.time(назваЛогера);`
- виконання певної дії;
- `console.timeEnd(назваЛогера);`

Оскільки запити в `AngularJS` є повністю асинхронним, тому мені потрібно створити константу для мапи назв логерів (як представлено див. лістинг 2.32).

Лістинг 2.32 – Створення константи із конфігураціями логерів

```
angular.module("mainApp", [])
...
.constant("console_time_test_map", {
  "all": "getAll",
  "one": "getOne",
  "create": "post",
  "update": "put",
  "delete": "delete"
})
```

Завершальною стадією розробки є створення контролера `Tests`. Слід обов'язково звернути увагу на те, що `AngularJS` усі запити робить асинхронно, тому буде використано алгоритм «неасинхронна-асинхронність», щоб не створювати конфліктів у базі даних. Суть такого алгоритму полягає в тригеруванні подій (на закінчення події запускати нову подію). Реалізувати такий алгоритм можна за допомогою провайдера `$Scope` та його методами `$on` і `$broadcast`.

Для створення контролера існує метод `controller`, першим параметром якого є назва контролера, а другим параметром – налаштування усіх `DependencyInjector`'ів (див. лістинг 2.33).

Лістинг 2.33 – Створення контролера Tests

```
angular.module("mainApp", [])
...
.controller("Tests", ["$rootScope", "$scope", "$http",
"rest_config", "console_time_test_map", function ($rootScope,
$scope, $http, rest_config, console_time_test_map) { }
])
```

Для запитів використано провайдер `$http`. Щоб протестувати отримання всіх клієнтів, використано логування часу, запит з типом GET та тригерування події GET_ALL_END (див. лістинг 2.34).

Лістинг 2.34 – Реалізація тесту отримання всіх клієнтів

```
angular.module("mainApp", [])
...
.controller("Tests", [..., function (...) {
  console.time(console_time_test_map.all);
  $http({
    "method": "GET",
    "url": rest_config.getAll
  }).then(function (response) {
    console.log("getAll end");
    console.timeEnd(console_time_test_map.all);
    console.log("get all response", response);
    $rootScope.$broadcast("GET_ALL_END");
  });
}])
```

Щоб протестувати отримання клієнта по його ідентифікатору, використано логування часу, запит з типом GET та тригерування події GET_ONE_END (як представлено див. лістинг 2.35).

Лістинг 2.35 – Реалізація тесту отримання клієнта по його ідентифікатору

```
angular.module("mainApp", [])
...
.controller("Tests", [..., function (...) {
  ...
  $scope.$on("GET_ALL_END", function () {
    console.time(console_time_test_map.one);
    $http({
      "method": "GET",
      "url": rest_config.getById
    }).then(function (response) {
      console.log("getById end");
    });
  });
}])
```



```

        console.timeEnd(console_time_test_map.one);
        console.log("get one response", response);
        $rootScope.$broadcast("GET_ONE_END");
    });
});
})

```

Щоб протестувати створення нового клієнта, використано логування часу, запит з типом POST, тестові дані та тригерування події CREATE_END (див. лістинг 2.36).

Лістинг 2.36 – Реалізація тесту створення нового клієнта

```

angular.module("mainApp", [])
...
.controller("Tests", [..., function (...) {
...
    $scope.$on("GET_ONE_END", function () {
        console.time(console_time_test_map.create);
        $http({
            "method": "POST",
            "url": rest_config.update,
            "data": rest_config.data.create
        }).then(function (response) {
            console.log("create end");
            console.timeEnd(console_time_test_map.create);
            console.log("create response", response);
            $rootScope.$broadcast("CREATE_END");
        });
    });
});
})

```

Щоб протестувати оновлення інформації клієнта по його ідентифікатору, використано логування часу, запит з типом PUT, тестові дані та тригерування події UPDATE_END (див. лістинг 2.37).

Лістинг 2.37 – Реалізація тесту оновлення інформації про клієнта по його ідентифікатору

```

angular.module("mainApp", [])
...
.controller("Tests", [..., function (...) {
...
    $scope.$on("CREATE_END", function () {
        console.time(console_time_test_map.update);
        $http({

```

```

        "method": "PUT",
        "url": rest_config.update,
        "data": rest_config.data.update
    }).then(function (response) {
        console.log("update end");
        console.timeEnd(console_time_test_map.update);
        console.log("update response", response);
        $rootScope.$broadcast("UPDATE_END");
    });
});
})();

```

Щоб протестувати видалення інформації клієнта по його ідентифікатору, використано логування часу та запит з типом DELETE (див. лістинг 2.38).

Лістинг 2.38 – Реалізація тесту видалення інформації про клієнта по його ідентифікатору

```

angular.module("mainApp", [])
...
.controller("Tests", [..., function (...) {
    ...
    $scope.$on("UPDATE_END", function () {
        console.time(console_time_test_map.delete);
        $http({
            "method": "DELETE",
            "url": rest_config.delete
        }).then(function (response) {
            console.log("delete end");
            console.timeEnd(console_time_test_map.delete);
            console.log("update response", response);
            $rootScope.$broadcast("DELETE_END");
        });
    });
});
})();

```

Для кожного із REST-сервісів виділено своя директорія на локальному сервері:

- Java – /java-rest;
- PHP – /php-rest;
- JavaScript – /node-rest.

Щоб протестувати кожен із них, потрібно міняти конфігурації. Тому тести проведено 3 рази. Перший тест проведено із Java REST сервісом (як представлено див. рисунок 2.2).

```

getAll end app.js:
getAll: 1143.176ms app.js:
get all response app.js:
▶ [Object, Object, Object, Object, Object, Object, Object, Object,
Object, Object, Object, Object, Object, Object]
getById end app.js:
getOne: 345.576ms app.js:
get one response app.js:
▶ Object {id: "jr12-ffgh-CZ-22", first_name: "Герасим", last_name: "Віманію",
phone: "000 000 00 1", mail: herasymvitaliy@gmail.com}
update end app.js:
post: 450.557ms app.js:
create response app.js:
▶ Object {first_name: "Герасим", last_name: "Анна", phone: "000 000 00 2", mail:
gerasymanna@gmail.com", address: "м.Тернопіль, вул. Руська буд. 56, кв. 104"}
...}
update end app.js:
put: 621.773ms app.js:
update response app.js:
▶ Object {id: "jr12-ffgh-CZ-22", first_name: "Герасим", last_name: "Антін",
phone: "000 000 00 3", mail: "herasymantyn@gmail.com"}
...}
delete end app.js:
delete: 380.456ms app.js:
update response Object {message: "SUCCESS"} app.js:

```

Рисунок 2.2 – Результат тестування Java REST сервісу

Другий тест проведено із PHP REST сервісом (див. рисунок 2.3). По результатам видно, що PHP REST сервіс дещо уступає по часі виконання операцій.

```

getAll end app.js:
getAll: 1290.718ms app.js:
get all response app.js:
▶ [Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, Object]
getById end app.js:
getOne: 389.550ms app.js:
get one response app.js:
▶ Object {id: "jr12-ffgh-CZ-22", first_name: "Герасим", last_name: "Віманіў", phone: "000 000 00 1", mail: "herasymvitaliy@gmail.com" ...}
update end app.js:
post: 508.558ms app.js:
create response app.js:
▶ Object {first_name: "Герасим", last_name: "Анна", phone: "000 000 00 2", mail: "gerasymanna@gmail.com", address: "м.Тернопіль, вул. Руська буд. 56, кв. 104" ...}
update end app.js:
put: 701.518ms app.js:
update response app.js:
▶ Object {id: "jr12-ffgh-CZ-22", first_name: "Герасим", last_name: "Антін", phone: "000 000 00 3", mail: "herasymantin@gmail.com" ...}
delete end app.js:
delete: 429.490ms app.js:
update response Object {message: "SUCCESS"} app.js:

```

Рисунок 2.3 – Результат тестування PHP REST сервісу

Третій тест проведено із JavaScript REST сервісом (див. рисунок 2.4). Видно, що такі операція як отримання списку всіх клієнтів, додавання та видалення працює набагато швидше, ніж у Java чи PHP, але з пошуком елементів з'являються затримки. Причиною цього є використання іншої бази даних.

Усі результати зведені в таблиці 2.1.

```

getAll end app.js:
getAll: 643.429ms app.js:
get all response app.js:
▶ [Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, Object, Object]
getById end app.js:
getOne: 328.167ms app.js:
get one response app.js:
▶ Object {_id: "jr12-ffgh-CZ-22", first_name: "Герасим", last_name: "Віманій", phone: "000 000 00 1", mail: "gerasymvitaliy@gmail.com" ...}
update end app.js:
post: 252.497ms app.js:
create response app.js:
▶ Object {first_name: "Герасим", last_name: "Анна", phone: "000 000 00 2", mail: "gerasymanna@gmail.com", address: "м.Тернопіль, вул. Руська буд. 56, кв. 104" ...}
update end app.js:
put: 481.491ms app.js:
update response app.js:
▶ Object {_id: "jr12-ffgh-CZ-22", first_name: "Герасим", last_name: "Антін", phone: "000 000 00 3", mail: "gerasymantin@gmail.com" ...}
delete end app.js:
delete: 346.447ms app.js:
update response Object {message: "SUCCESS"} app.js:
|

```

Рисунок 2.4 – Результат тестування JavaScript REST сервісу

Таблиця 2.1 – Результати тестування

Запит	Час виконання (мс)		
	Java	PHP	JavaScript
GET /customers	1143.176	1290.718	643.429
GET /customers/jr12-ffgh-CZ-22	345.576	389.550	328.167
POST /customers	450.557	508.558	252.497
UPDATE /customers	621.773	701.518	481.491
DELETE /customers/qxty-ffrz-CX-18	380.456	428.490	346.447

По результатам тестування, JavaScript в поєднанні з NodeJS та MongoDB є найоптимальнішим рішенням для розробки REST сервісу для веб-систем. Якщо вибирати між Java та PHP, то кращим варіантом буде Java.

В ході написання розділу було підготовлено базу даних для роботи REST сервісами. Описано детально можливі маршрути (URLs), які надають певну

можливість веб-сервісу. Описано, які конкретно статус-коди повинен повертати при будь-якому запиті до REST сервісу. Розроблено REST сервіс на мові програмування Java за допомогою технологій Jax-RS із використанням сервера Apache CXF та MySQL. Розроблено REST сервіс на мові програмування PHP за допомогою двигуна Phalcon із використанням сервера Apache та MySQL. Розроблено REST сервіс на мові програмування JavaScript за допомогою двигуна NodeJS із модулями Express та Mongoose із використанням сервера бази даних MongoDB.

В ході проведення тестування було визначено по яким критеріям тестувати розроблені REST сервіси, а саме: правильність функціонування та швидкодія. Розроблено веб-додаток за допомогою двигуна AngularJS для тестування розроблених веб-сервісів. Створено сценарії виконання тестів із тестовими даними. Проведено тестування для кожного із REST сервісів. Результати тестування зведено до таблиці.

По результатам тестування виявилось, що JavaScript REST сервіс в поєднанні з NodeJS та MongoDB є найоптимальнішим рішенням для розробки REST сервісу для веб-систем. Якщо вибирати між Java та PHP, то кращим варіантом буде Java.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Загальні вимоги безпеки з охорони праці для користувачів ПК. [23,24]

Темою кваліфікаційної роботи є дослідження оптимальності REST сервісів для веб-систем створених на різних мовах програмування. Дослідження проводиться з урахуванням розробки тестових продуктів та їх тестування на всіх етапів життєвого циклу ПЗ. При використанні ПЗ, яке є результатом даної розробки, як і при використанні будь-якого іншого ПЗ, необхідно дотримуватися вимог з охорони праці при роботі з ПК. Розглянемо основні нормативні документи, в яких зазначені вимоги до робочих місць та приміщень при використанні ПК.

При розробці системи потрібно запобігти негативному впливу виробничих факторів, а саме дотримуватися правил та норм щодо приміщень де знаходитиметься робоче місце, відповідно до «Правил охорони праці під час експлуатації електронно-обчислювальних машин» та ДСанПіН «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Відповідно до ДСанПіН робоче місце працівника не можна розміщувати у підвалах та цокольних поверхах. Площа робочого місця для виконання розробки повинна становити не менше 6 м², а об'єм 20 м³. Приміщення, в якому працюють на ПК повинно бути забезпечене природнім та штучним освітленням згідно з нормами ДБН. Природне освітлення здійснюється крізь світлові отвори, направлені на північний схід і вони забезпечують коефіцієнт природньої освітленості (КПО) не менший ніж 1,5%. Розраховується КПО за методикою, що міститься у ДБН. В приміщеннях де проводиться розробка та міститься ПК, штучне освітлення здійснюється системою загального рівномірного освітлення. Джерелом штучного освітлення слугують люмінесцентні лампи. Віконні отвори приміщення обладнані вертикальними жалюзіями та зовнішніми козирками. У даному приміщенні функціонує система опалення. Відповідно до санітарних норм

та правил у даному приміщенні знаходиться аптечка першої медичної допомоги. Також, у даному приміщенні щоденно проводиться вологе прибирання.

Згідно правил НПАОП обладнання та організація робочого місця працюючого з ПК має відповідати ергономічним вимогам, а саме:

- оптимальна робоча поза користувача ПК забезпечується конструкцією робочого місця;
- робоче місце розташоване так відносно світлових отворів, що природне світло падає збоку зліва;
- відповідно вимогам, робочий стілець є підйомно-поворотним з регульованою висотою;
- будова робочого стола відповідає вимогам ергономіки та дозволяє оптимально розміщувати на робочій поверхні ПК, допоміжне обладнання та документи.

Нормовані параметри мікроклімату, іонного складу повітря, вмісту шкідливих речовин мають відповідати вимогам Санітарних норм;

Мікрокліматичні умови приміщення повинні відповідати нормальним значенням таких показників:

- температура повітря;
- відносна вологість повітря;
- швидкість руху повітря;
- інтенсивність теплового (інфрачервоного) опромінення.

За ступенем впливу на тепловий стан людини мікрокліматичної умови поділяють на оптимальні та допустимі.

Для робочої зони приміщення встановлюються оптимальні та допустимі мікрокліматичні умови з урахуванням складності виконуваної роботи та періоду року. При виконанні роботи на ПК, що пов'язано з нервово-емоційним напруженням, у приміщенні потрібно підтримувати температура повітря $+22^{\circ}\text{C}$ – $+24^{\circ}\text{C}$, відносну вологість 60-40%. Дані вимоги до параметрів мікроклімату містяться у санітарних нормах ДСН.

Оскільки розробка системи комплексної обробки подій здійснюється за ПК, необхідно проводити перерви по 15 хвилин через кожну годину. Якщо, виробничі обставини не дозволяють здійснювати часті перерви, тривалість роботи з ПК не повинна бути більшою чотирьох годин.

Відповідно до основних правил електробезпеки під час використання ПК потрібно дотримуватися таких вимог:

- при використанні ліній електромережі необхідно запобігти виникненню електричного джерела займання внаслідок короткого замикання чи перевантаження мережі;
- до електромережі ПК підключається тільки з а допомогою штепсельних з'єднань і електророзеток;
- спеціальні контакти для підключення нульового захисного провідника повинні міститися у штепсельних з'єднаннях та електророзетках. При цьому, під'єднання нульового захисного провідника відбувається швидше, ніж під'єднання фазового та нульового робочого провідника;
- підключення живлення ПК до двопровідної електромережі є недопустимим, навіть з використанням перехідних пристроїв. Тому здійснюється підключення по трьохпровідній мережі.

Згідно основних вимог до пожежної безпеки приміщення, у яких знаходиться ПК, мають бути оснащені переносними вуглекислотними або аерозольно-водопінними вогнегасниками. Підходи до засобів пожежогасіння повинні бути вільними. Також, згідно вимог НАПБ «Перелік однотипних за призначенням об'єктів, які підлягають обладнанню автоматичними установками пожежогасіння та пожежної сигналізації», у приміщенні де здійснюється робота з ПК, робочі місця повинні бути обладнанні системою автоматичної пожежної сигналізації з димовим пожежним сповіщувачем.

Таким чином, в даному підрозділі виконано огляд основних законодавчих актів та нормативів при роботі з ПК. Так як, дипломна робота магістра спрямована на розробку системи комплексної обробки подій та її реалізація для сфери алгоритмічної торгівлі, то було наведено основні стандарти і правила щодо

влаштування робочих місць де використовують комп'ютерну техніку. Також було наведено вимоги, які стосуються приміщення де знаходиться робоче місце працівника. Крім того, розглянуті санітарні норми та правила щодо мікроклімату у даному приміщенні. Наведенні правила електробезпеки під час роботи з ПК. Також поданні основні вимоги до пожежної безпеки приміщення. При дотриманні даних правил гарантуються безпечні умови праці на ПК та запобігання шкідливих виробничих факторів.

3.2 Запобігання виникненню надзвичайних ситуацій [25,26]

Запобігання виникненню надзвичайних ситуацій— це підготовка та реалізація комплексу правових, соціально-економічних, політичних, організаційно-технічних, санітарно-гігієнічних та інших заходів, спрямованих на регулювання безпеки, проведення оцінки рівнів ризику, завчасне реагування на загрозу виникнення надзвичайної ситуації на основі даних моніторингу (спостережень), експертизи, досліджень та прогнозів щодо можливого перебігу подій з метою недопущення їх переростання у надзвичайну ситуацію або пом'якшення її можливих наслідків.

Зазначені функції запобігання надзвичайним ситуаціям техногенного та природного характеру в нашій країні виконує Єдина державна система запобігання надзвичайним ситуаціям техногенного і природного характеру і реагування на них, затверджена Постановою Кабінету Міністрів України.

Єдина державна система запобігання надзвичайним ситуаціям техногенного і природного характеру і реагування на них (ЄДСЗР) включає в себе центральні та місцеві органи виконавчої влади, виконавчі органи рад, державні підприємства, установи та організації з відповідними силами і засобами, які здійснюють нагляд за забезпеченням техногенної та природної безпеки, організують проведення

роботи із запобігання надзвичайним ситуаціям техногенного та природного походження і реагування у разі їх виникнення з метою захисту населення і довкілля, зменшення матеріальних втрат.

Основною метою створення ЄДСЗР є забезпечення реалізації державної політики у сфері запобігання і реагування на надзвичайні ситуації, забезпечення цивільного захисту населення.

ЄДСЗР складається з постійно діючих функціональних і територіальних підсистем і має чотири рівні: загальнодержавний, регіональний, місцевий та об'єктовий.

Функціональні підсистеми створюються міністерствами та іншими центральними органами виконавчої влади для організації роботи, пов'язаної із запобіганням надзвичайним ситуаціям та захистом населення і територій від їх наслідків.

Кожний рівень ЄДСЗР має координуючі та постійні органи управління щодо розв'язання завдань у сфері запобігання надзвичайним ситуаціям, захисту населення і територій від їх наслідків, систему повсякденного управління, сили і засоби, резерви матеріальних та фінансових ресурсів, системи зв'язку та інформаційного забезпечення.

Координуючі органи ЄДСЗР:

- державна комісія з питань техногенно-екологічної безпеки та надзвичайних ситуацій;
- національна рада з питань безпечної життєдіяльності населення;
- комісії Ради міністрів Автономної Республіки Крим, обласних, Київської та Севастопольської міських державних адміністрацій з питань техногенно-екологічної безпеки та надзвичайних ситуацій;
- комісії районних державних адміністрацій і виконавчих органів рад з питань техногенно-екологічної безпеки та надзвичайних ситуацій;
- комісії з питань надзвичайних ситуацій об'єкта.

До складу ЄДСЗР входять відповідні сили і засоби функціональних і територіальних підсистем, а також недержавні (добровільні) рятувальні формування, які залучаються для виконання відповідних робіт.

У виняткових випадках, коли стихійне лихо, епідемія, епізоотія, аварія чи катастрофа ставить під загрозу життя і здоров'я населення і потребує термінового проведення великих обсягів аварійно-рятувальних і відновлювальних робіт, Президент України може залучати до виконання цих робіт у порядку, визначеному Законом України «Про надзвичайний стан», спеціально підготовлені сили і засоби Міноборони.

На базі існуючих спеціалізованих служб і підрозділів (будівельних, медичних, хімічних, ремонтних та інших) в областях, районах, населених пунктах, підприємствах, установах та організаціях утворюються позаштатні спеціалізовані формування, призначені для проведення конкретних видів невідкладних робіт у процесі реагування на надзвичайні ситуації. Ці формування проходять спеціальне навчання, періодично залучаються до участі у практичному відпрацюванні дій з ліквідації надзвичайних ситуацій.

У виконанні робіт, пов'язаних із запобіганням надзвичайним ситуаціям і реагуванням на них, можуть брати участь також добровільні громадські об'єднання за наявності у представників цих об'єднань відповідного рівня підготовки, підтвердженого в атестаційному порядку. Свої дії вони повинні узгоджувати з територіальними органами та уповноваженими з питань надзвичайних ситуацій та цивільного захисту населення, а роботи виконувати під їх керівництвом.

Надзвичайний стан по всій території України або в окремих її місцевостях вводиться постановою Верховної Ради України з негайним повідомленням Президента України або Указом Президента України, який підлягає затвердженню Верховною Радою України.

Під час надзвичайного стану держава може вживати заходів, передбачених Законом «Про надзвичайний стан», відступаючи від своїх зобов'язань за Конституцією лише настільки, наскільки це вимагається гостротою стану, за

умови, що такі заходи не є несумісними з іншими зобов'язаннями за міжнародним правом і не тягнуть за собою дискримінації на основі національності, мови, статі, релігії чи соціального походження.

З метою ліквідації наслідків надзвичайної ситуації у мирний час може проводитись цільова мобілізація. У виняткових випадках допускається залучення працездатного населення і транспортних засобів громадян для виконання невідкладних аварійно-рятувальних робіт за умови обов'язкового забезпечення безпеки праці. При Цьому забороняється залучення неповнолітніх, а також вагітних жінок до робіт, які можуть негативно вплинути на стан їхнього здоров'я.

ВИСНОВКИ

Було представлено кваліфікаційну роботу бакалавра, основою для розробки якої є завдання на дипломну роботу, затверджене наказом від кафедри програмної інженерії, факультету комп'ютерно-інформаційних систем та програмної інженерії при Тернопільському національному технічному університеті ім.Ів.Пуллюя.

В роботі представлено виконані задачі для реалізації програмної розробки, до яких було віднесено:

1. Аналізування предметної області з метою вибору існуючих або розробки нових (удосконалення існуючих) моделей, що описують предметну область, математична постановка задач та вибір моделей їх розв'язку.

2. Вибір та обґрунтування вибору архітектури, вибір безпосередньо програмного середовища та необхідних додаткових бібліотек для реалізації програмної системи, вибір зовнішніх програмних, а також апаратних засобів для забезпечення роботи програмної системи.

3. Аналізування та уточнення вимог технічного завдання з точки зору обраних моделей, методів, алгоритмів та середовища розробки.

4. Проєктування, реалізація, а також тестування окремих компонент програмної системи, розробка принципів взаємозв'язка поміж ними.

5. Тестування програмного комплексу на навантаження після написання тест плану.

В процесі виконання представленої роботи на кваліфікаційний рівень «бакалавр» було:

- висвітлено призначення REST сервісу;
- описано модель представлення інформації в базі даних та детально описано її атрибути;
- складено словник термінів розробки програмного комплексу веб-сервісів;
- проведено вибір технологій реалізації веб-сервісів.

Крім вище переліченого виконано розробку веб-сервісів та веб-додатку для тестування:

- підготовлено базу даних для роботи REST сервісами;
- описано функціональні можливості веб-сервісів;
- розроблено REST сервіс на мові програмування Java;
- розроблено REST сервіс на мові програмування PHP;
- розроблено REST сервіс на мові програмування JavaScript;
- визначено по яким критеріям тестувати розроблені REST сервіси;
- розроблено веб-додаток для тестування розроблених веб-сервісів;
- створено сценарії виконання тестів із тестовими даними;
- проведено тестування для кожного із REST сервісів.

По результатам тестування виявилось, що JavaScript REST сервіс в поєднанні з NodeJS та MongoDB є найоптимальнішим рішенням для розробки REST сервісу для веб-систем. Якщо вибирати між Java та PHP, то кращим варіантом буде Java.

Також було дотримано, висвітлено та описано заходи по охороні праці та з безпеки при надзвичайних ситуаціях за отриманим від консультанта завданням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки до виконання дипломної роботи освітнього рівня —бакалавр студентами усіх форм навчання для напряму підготовки 121 — —Інженерія програмного забезпечення// Укладачі : Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладь С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

2. М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – “Інженерія програмного забезпечення” для усіх форм навчання [Текст] – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.

3. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, Я.І. Кінах, Г.Б. Цуприк - Тернопіль: ТНТУ, 2017-38 с

4. Бойко І.В., М.Р. Петрик, Г.Б. Цуприк. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

5. Бойко І.В., М.Р. Петрик, Г.Б. Цуприк. Моделювання та видобуток даних (високопродуктивні обчислення у великих алгебраїчних та числових системах, комбінаторному аналізі): навчальний посібник. Тернопіль: : ТНТУ 2019 – 62 с.

6. Нога В.В., Цуприк Г.Б. Інформаційні технології при створенні автоматизованих систем для задач збору та збереження інформації. Збірник тез доповідей VII Міжнародної науково-технічної конференції молодих учених та студентів. Актуальні задачі сучасних технологій – Тернопіль 28-29 листопада 2018. — Т. : ТНТУ, 2018. — Том 2. — С. 131.

7. І.В. Бойко, М.Р. Петрик, Г.Б. Цуприк. Дискретні структури (Алгебраїчні та числові системи, комбінаторний аналіз). Навчально-методичний посібник для

студентів спеціальності 121 «Інженерія програмного забезпечення», аспірантів та викладачів вищих навчальних закладів. Конспект лекцій : Навчальний посібник / І.В. Бойко, М.Р. Петрик, Г.Б. Цуприк – Тернопіль : ТНТУ , 2017. – 64 с.

8. Рейтинг мов програмування на комерційному ринку [Електронний ресурс] – Режим доступу: URL: <http://dou.ua/lenta/articles/language-rating-jan-2015/>.

9. [Електронний ресурс] – Режим доступу: URL: <https://highload.today/uk/top-10-najpopulyarnishih-mov-programuvannya/>

10. [Електронний ресурс] – Режим доступу: URL: https://uk.wikipedia.org/wiki/Історія_мов_програмування

11. [Електронний ресурс] – Режим доступу: URL: <https://www.programming.in.ua/programming/java.html>

12. [Електронний ресурс] – Режим доступу: URL: <https://uk.wikipedia.org/wiki/Компілятор>

13. [Електронний ресурс] – Режим доступу: URL: HTTP://WIKI.TNTU.EDU.UA/INDEX.PHP?TITLE=4_ОСНОВИ_PHP

14. High Performance PHP Framework – Phalcon Framework [Електронний ресурс] – Режим доступу : URL : <HTTPS://PHALCON.IO/EN-US>

15. [Електронний ресурс] – Режим доступу: URL: <https://web.archive.org/web/20160328224926/http://www.cse.wustl.edu/~schmidt/PDF/HPL.pdf>

16. Вейтман, Віктор. Програмування для Web [Текст] : рук. розробника : [навч. посібник] / В. Вейтман; СПб. ; Київ : Діалектика : Вільямс, 2000. - 364 с. : рис., табл. - 7000 прим.

17. Java Hibernate. Частина 1 - Введення [Електронний ресурс] – Режим доступу: URL: <http://javaxblog.ru/article/java-hibernate-1/>

18. HIBERNATE-Relational Persistence for Idiomatic Java [Електронний ресурс] – Режим доступу: URL: <http://samsonych.com/lib/hibernate/index.html>

19. [Електронний ресурс] – Режим доступу: URL: <https://nodejs.org/en>

20. Fielding, Roy (2000). Architectural Styles and the Design of Network-based Software Architectures (Ph.D.) (англійською). Каліфорнійський університет в Ірвайні.

21. Richardson, Leonard; Amundsen, Mike; Ruby, Sam (2013). RESTful Web APIs (вид. First edition). O'Reilly. ISBN 978-1-4493-5806-8.

22. [Електронний ресурс] – Режим доступу: URL: <https://docs.oracle.com/javaee/6/tutorial/doc/giepu.html>

23. Дистанційний курс «Основи охорони праці» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: <http://dl.tntu.edu.ua/index.php>

24. Охорона праці : Навчальний посібник [Текст] Геврик Є.О. - Ельга: Ніка-Центр. - 2003. – 279 с.

25. Запобігання виникненню надзвичайних ситуацій [Електронний ресурс] / Wikipedia – URL: https://uk.wikipedia.org/wiki/Запобігання_виникненню_надзвичайних_ситуацій_характеру.

26. Ураження сильнодіючими отруйними речовинами [Електронний ресурс] / URL: <http://www.wikidocs.ru/preview/44941>.