Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня бакалавра

(бакалавра, спеціаліста, магістра)

Студента Атамаса Іллі Олександровича
ПІБ)

академічної групи 126-17-1
(шифр)

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

за освітньо-професійною програмою

«Інформаційні системи та технології»
(офіційна назва)

на тему Створення хмарної розподіленої високопродуктивної системи обробки зображень і відео

(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
Кваліфікавційної роботи	проф. Гнатушенко В.В.			
Розділів				

Рецензент	доц. Коротенко Л.М.			

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2021

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)

_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)

«_____»_____2021року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня _____бакалавр_____
(бакалавра, спеціаліста, магістра)

студента Атамаса Іллі Олександровича
академічної групи 126-17-1
(прізвище та ініціали) (шифр)

спеціальності 126 « Інформаційні системи та технології »

за освітньо-професійною програмою Інформаційні системи та технології

на тему Комплексна кваліфікаційна робота: Створення хмарної розподіленої високопродуктивної системи обробки зображень і відео

затверджену наказом ректора НТУ «Дніпровська політехніка» від 27.04.2021 р. № 317-с

Розділ	Зміст	Термін виконання
Розділ 1		01.02.2021 – 28.02.2021
Розділ 2		01.03.2021 – 30.03.2021
Розділ 3		01.04.2021 – 10.06.2021

Завдання видано _____ Гнатушенко В.В.
(підпис керівника) (прізвище, ініціали)

Дата видачі 01.02.2021 р.

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Атамас І.О.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 98 сторінок, 2 малюнка, 23 джерела, 5 додатків.

Об'єкт розробки: хмарна розподілена високопродуктивна система обробки зображень і відео.

Мета дипломної роботи: Створення хмарної розподіленої високопродуктивної системи обробки зображень і відео.

У вступі вказана мета даної дипломної роботи, обгрунтована її актуальність, конкретизовані завдання, які необхідно вирішити.

У першому розділі пояснювальної записки проведено аналіз проблемної області і визначення цілей дослідження, описано сутність та принципи функціонування веб-серверів, та визначені можливі напрямки рішення поставленої задачі.

У другому розділі описано інструменти та методи обробки зображень і відео.

У третьому розділі описано розгортання та інтеграцію розробленої системи системи для обробки зображень і відео.

Практична значимість даного проекту полягає в можливості його використання для обробки та оптимізації зображень та відео файлів у будь-якій системі, яка працює із такими файлами.

У третьому розділі було виконано розгортання і інтеграція системи для обробки і оптимізації зображень і відео.

ХМАРНА РОЗПОДІЛЕНА ВИСОКОПРОДУКТИВНА СИСТЕМА ОБРОБКИ ЗОБРАЖЕНЬ І ВІДЕО, КОМПІЛЬОВАНА МОВА ПРОГРАМУВАННЯ GO.

ABSTRACT

The explanatory note contains 98 pages, 2 figures, 23 sources, 4 appendices.

Development object: a cloud-distributed, highly loaded, high-availability image and video processing and optimization system.

The purpose of the thesis: Development of a cloud distributed high-load high-availability system for processing and optimization of images and video.

The introduction indicates the purpose of this thesis, substantiates its relevance, specifies the tasks to be solved.

The first section of the explanatory note analyzes the problem area and defines the objectives of the study, describes the essence and principles of operation of web servers, and identifies possible areas for solving the problem.

The second section describes the tools and methods of image and video processing.

The third section describes the deployment and integration of developed system systems for image and video processing.

The practical significance of this project lies in the possibility of its use for processing and optimization of images and video files in any system that works with such files.

CLOUD DISTRIBUTED HIGHLY LOADED AVAILABLE SYSTEM OF IMAGE AND VIDEO PROCESSING AND OPTIMIZATION SYSTEM, COMPILED GO PROGRAMMING LANGUAGE.

ЗМІСТ

ВСТУП

Метою даного проекту є створення рішення для пришвидшення завантаження фото і відео файлів задля покращення користувацького досвіду, створення рішення для зменшення об’єму споживання оперативної пам’яті мобільними пристроями і пристроями з невеликим об’ємом оперативної пам’яті, створення рішення для оптимізації об’єму займаної файлами пам’яті у системі, в якій розгорнуто застосунок.

Для вирішення поставленої мети вирішуються наступні завдання:

- Огляд методів та засобів, що використовуються для пришвидшення завантаження фото і відео файлів у веб-застосунках;
- Огляд методів та засобів, що використовуються для пришвидшення завантаження фото і відео файлів у мобільних додатках;
- Огляд методів для зменшення об’єму споживання оперативної пам’яті мобільними пристроями і пристроями з невеликим об’ємом оперативної пам’яті;
- Дослідження можливості оптимізації процесу розробки вищеперелічених систем за допомогою об’єднання цих систем в одну систему для оптимізації та обробки фото і відео файлів;
- Огляд методів зменшення розміру файлів для подальшого їх зберігання;
- Огляд інструментів для оптимізації розміру файлів;
- Огляд засобів для забезпечення ефективної обробки фото і відео файлів;
- Розробити систему для ефективної обробки фото і відео файлів;
- Розробити систему для зменшення об’єму споживання оперативної пам’яті мобільними пристроями і пристроями з невеликим об’ємом оперативної пам’яті.

Реалізація вищезазначених завдань дозволить покращити користувацький досвід у веб і мобільних додатках, заменить навантаження на операційні системи користувачів, що використовують ОС iOS або Android, або використовують настільні комп'ютери з невеликим об'ємом оперативної пам'яті.

Постановка задачі

Необхідно продумати систему, яка забезпечувала би швидкий показ медіафайлів і систему, яка забезпечувала би зменшення навантаження на оперативну пам'ять веб і мобільних додатків.

Перш за все, необхідно визначити, чи можливе об'єднання цих двох систем в одну, задля оптимізації процесу розробки (його швидкості і якості) за рахунок зменшення об'єму розроблюваних систем. Згідно дослідженням, операційна система споживає тим більше оперативної пам'яті, чим більшого розширення зображення або відео вона відображає [8] [9].

З цього випливає, що можна зменшити об'єм зайнятої застосунком оперативної пам'яті за рахунок зменшення розширення зображення або відео. Це дозволить відображати зображення або відео із меншим навантаженням на оперативну пам'ять. Тож можна забезпечити зменшення навантаження на ОС за рахунок обробки і оптимізації фото і відео файлів, що дозволяє об'єднати дві необхідні системи в одну. Також система не повинна використовувати багато обчислювальних ресурсів для того, щоб її впровадження забезпечувало високу продуктивність 8 і не впливало на роботу базової системи (сервісу до впровадження розроблюваної системи). Вона повинна працювати швидко для того, аби обробка фото і відео файлів не займала багато часу і не впливала на швидкість роботи базової системи.

Також розроблювана система повинна працювати окремо від базової системи та, за можливості, не залежати від клієнтської частини базової системи. Це необхідно для того, аби систему, що розробляється, можна було

легко пристосувати до різних застосунків - щоб вона була модулем базової системи, а не частиною її ядра. Також дуже важливо мати змогу оптимізувати вже існуючі у базовій системі зображення та відео, а не тільки ті, що будуть завантажуватись після впровадження системи, що розробляється.

Отже, задача практики полягає у розробленні хмарної розподіленої високонавантаженої високодоступної системи обробки і оптимізації зображень і відео.

Задачі для розробки системи наступні:

- Дослідити можливості оптимізації зображень і відео з мінімальною втратою якості вхідних файлів;
- Дослідити інструменти та методи для обробки зображень і відео;
- Розробити систему, яка не залежить від клієнтської частини веб або мобільного застосунку, працює швидко, не споживає багато обчислювальних ресурсів;
- Система, що розробляється повинна працювати однаково як для мобільний пристроїв (iOS і Android), так і для веб-застосунків;
- Система, що розробляється, повинна бути конфігуруємою, тобто необхідно мати змогу налаштовувати силу оптимізації (стиснення) зображень або відео;
- Система повинна мати змогу обробляти вже існуючі (завантажені) зображення і відео.

РОЗДІЛ 1. ВЕБ-СЕРВЕР ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ: СУТНІСТЬ, ПРИНЦИПИ ФУНКЦІОНУВАННЯ ТА ЕФЕКТИВНІСТЬ

1.1. Аналіз проблемної області і визначення цілей дослідження

Інформаційні технології є невід'ємною частиною сучасного світу, вони значною мірою визначають подальший економічний та суспільний розвиток людства. У цих умовах революційних змін вимагає й система навчання. Звідси можна сказати, що актуальність даного питання має місце у сучасному освітньому середовищі, адже нині якісне викладання дисциплін не може здійснюватися без використання засобів і можливостей, які надають комп'ютерні технології та Інтернет. Інформаційні технології, ІТ – сукупність методів, виробничих процесів і програмно-технічних засобів, інтегрованих з метою збирання, опрацювання, зберігання, розповсюдження, показу і використання інформації в інтересах її користувачів. Технології, що забезпечують та підтримують інформаційні процеси, тобто процеси пошуку, збору, передачі, збереження, накопичення, тиражування інформації та процедури доступу до неї. Інформаційно-комунікаційні технології (ІКТ, від англ. Information and communications technology, ICT) – часто використовується як синонім до інформаційних технологій (ІТ), хоча ІКТ це загальніший термін, який підкреслює роль уніфікованих технологій та інтеграцію телекомунікацій (телефонних ліній та бездротових з'єднань), комп'ютерів, підпрограмного забезпечення, програмного забезпечення, накопичувальних та аудіовізуальних систем, які дозволяють користувачам створювати, одержувати доступ, зберігати, передавати та змінювати інформацію. Іншими словами, ІКТ складається з ІТ, а також телекомунікацій, медіа-трансляцій, усіх видів аудіо і відеообробки, передачі, мережевих функцій управління та моніторингу.

Обчислювальні та комунікаційні системи використовуються все частіше і з кожним днем все глибше входять в наше повсякденне життя. Компанії та окремі користувачі все більше залежать у своїй роботі від web-додатків. Web-додатки з'єднують різні відділи всередині компаній, різні

компанії і простих користувачів. Web-додатки дуже динамічні, а їх функціональні можливості безперервно зростають. Безперервно зростає потоковий трафік засобів інформації та запити, що формуються переносними і вбудованими пристроями[21]. Внаслідок цього зростає складність систем такого роду. Очевидно, що для розуміння, аналізу, розробки та управління такими системами потрібні кількісні методи та моделі, які допомагають оцінити різні сценарії функціонування, досліджувати структуру і стан великих систем. Спостерігаються тенденції до постійного зростання попиту на web-служби. Таким чином, проблеми, пов'язані з недостатньою продуктивністю виникатимуть і в майбутньому, і, врешті-решт, вони стануть переважати при плануванні та введенні в експлуатацію нових web-служб і збільшення користувачів Internet. Щоб уникнути багатьох проблем, пов'язаних з невідповідністю реальної та очікуваної користувачами продуктивності, необхідні спеціальні методики планування продуктивності.

Часи, коли Web-сайти складалися із статичного контенту і декількох cgi або javaскриптів для обробки форм, пішли безповоротно. Зараз від Web-додатків потрібно значно більше інтерактивності.

Кількість різноманітних технологій і підходів для Web-розробок нині здається просто гігантською, а напрям їх розвитку – непередбаченим. Як вибрати перспективну робочу платформу для Web-розробок, що дійсно задовольняє сучасним вимогам? Для такого вибору необхідно порівнювати безліч підходів і реалізацій, треба мати про них представлення. Спочатку перед Web-серверами стояло просте завдання: знайти і відправити клієнтові файл, вказаний в отриманому від клієнта запиті. Запит складався теж дуже просто в адресі URL. Прикладне програмування для Web починалося з обробки запитів користувача, що передаються через форми і динамічній генерації сторінок на стороні сервера. За цим же принципом працюють мови програмування вставок (SSI) в HTML документи.[Дронов В.HTML 5, CSS 3 и Web 2.0 РазработкасовременныхWeb-сайтов / В. Дронов // Профессиональноепрограммирование. – 2013. – 414 с.].

Розвиток WWW привів до створення мов програмування елементів HTML документів на стороні клієнта (наприклад, JavaScript). Це привело до необхідності створення на сервері систему попередньої обробки файлу, що відправлявся. Web-сервер ускладнився, що з'явилися різні прийоми динамічної генерації сторінок HTML, що зажадало виконувати на сервері процедури. У запиті URL вставили виклик процедур, а на сервері реалізували технологію CGI (CommonGatewayInterface). Тепер в запиті URL вказується процедура, яку треба виконати на сервері. Процедуру CGI можна написати на будь-якій мові, аби вона сприймала стандартне введення і стандартний вивід. У технології Java для цього створюються аплети, сервлети, використовується мова JSP(Java Server Pages). Причини зростання ролі Web-додатків зрозумілі - вони не вимагають установки програмних засобів у користувача і їх набагато простіше "підлаштовувати під цього самого користувача", такі застосування більше керовані з обох боків, менше вимог до клієнтського пристрою. Багато застосувань вже використовують для взаємодії з користувачем Web-інтерфейс, в Web-додатки закладається функціональність, порівнянна з традиційними настільними застосуваннями. У основі реалізації корпоративних інформаційних систем на базі архітектури Інтернет/ Інтранет лежить принцип "відкритої архітектури", що багато в чому визначає незалежність реалізації корпоративної системи від конкретного виробника. Усепрограмне забезпечення таких систем реалізуєтьсяу вигляді аплетів або сервлетів (програм написаних на мові JAVA) або у вигляді cgi модулів (програм написаних, як правило, на Perl або C++) [Буди Курняван - Создание WEB-приложений на языке Java с помощью сервлетов, JSP и EJB / Буди Курняван // NewRiders. – 2013. – 880 с.]. Під клієнтською платформою доцільно розуміти не лише системне оточення на клієнтській стороні, але і спосіб організації призначеного для користувача інтерфейсу і його взаємодії з бізнес-логікою, розділеною у рамках додатка на клієнтську і серверну частину. У додатку здійснюється взаємодія між клієнтською і серверною частиною, і це є визначальний для клієнтської платформи. При забезпеченні

Web-доступа до існуючих баз даних (БД), можливий ряд технологічних і організаційних рішень.

Практика використання Web-технологій для доступу до існуючих БД надає широкий спектр технологічних рішень, по різному пов'язаних між собою, що перекривають, взаємодіють і доповнюють. Вибір конкретних рішень при забезпеченні доступу, що залежить від специфіки конкретної СУБД і від ряду інших чинників, як платформа, сервер, наявність фахівців, здатних з мінімальними витратами освоїти певну гілку технологічних рішень, існування інших БД. У загальному випадку інформаційна система, реалізована з використанням архітектури клієнт-сервер, включає Web-вузли з інтерактивним інформаційним наповненням, реалізованих за допомогою технологій Java, JavaBeans, JavaScript, PHP, ASP, Perl, що взаємодіють з базою даних, з одного боку, і з клієнтським місцем з іншого. База даних, у свою чергу, є джерелом інформації для інтерактивних додатків реального часу [Фролов А. Практика застосування Perl, PHP, Apache, MySQL для активних Web -сайтов / А. Фролов, Г. Фролов // Інтернет-технології. – 2002. – 576 с.].

За останні кілька років в технології управління інформаційними системами змінилася точка зору на проблему ефективності функціонування web-додатків. Ще недавно вважалося, що основними критеріями якості роботи інформаційної системи є характеристики роботи обладнання. Вважалося, що якщо значення цих характеристик не перевищують встановлені межі, то інформаційна система працює добре. Тому основна увага приділялася питанням управління активним обладнанням мережі.

Відповідно до сучасної концепції управління інформаційними системами[Куликовский Л.Ф., Мотов В.В.,Теоретическиеосновыинформационныхпроцессов. М: Высшая школа, 1987, 248с] вважається, що висновки про їх якість роботи слід робити на підставі якості роботи web-додатків. Якщо якість роботи web-додатків висока, то інформаційна система працює добре, інакше - погано. При цьому характеристики роботи обладнання також важливі, але використовуються, в

першу чергу, для з'ясування того, чому web-додатки працюють погано і що треба зробити, щоб поліпшити їх роботу.

На сьогоднішній день існує досить велика кількість компаній [Олифер В.Г., Олифер Н.А., Компьютерные сети. Принципы, технологии, протоколы. СПб. Питер, 2003, 672 стр.], які займаються дослідженням ефективності функціонування web-додатків. Причому ці компанії пропонують на ринку як власні послуги по оцінці ефективності функціонування додатків замовників, так і програмні продукти для самостійного використання. Був проведений аналіз наявних коштів для оцінки ефективності функціонування, який показав, що: на даний момент немає чітко визначеного визначення того, у чому полягає оцінка ефективності функціонування web-додатки. Трактування визначення, як правило, диктується замовником, або пропонується виконавцем, виходячи з наявних технічних засобів для реалізації тестування web-додатки замовника.

Практично будь-який web-додаток працює в унікальному середовищі функціонування. Унікальність визначається поєднанням широкого набору параметрів (конфігурація обладнання, програмне забезпечення, архітектура web-додатки, кількість і склад цільової аудиторії, режим роботи тощо). Це призводить до розробки програмних засобів тестування саме конкретної розглянутої середовища, в якій функціонує web-додаток.

Як наслідок цього - відсутність єдиної технології оцінки ефективності функціонування web-додатки.

Підводячи підсумок, можна сказати, що проблема ефективності функціонування є актуальною, і пріоритетність цієї проблеми безперервно зростає з розвитком Internet-технологій. Детальне дослідження режимів роботи web-серверів і віддалених користувачів дозволило б систематизувати наявну інформацію і розробити методику оцінки ефективності функціонування високонавантеженихweb-серверів на практиці.

1.2. Сутність та принципи функціонування веб-серверів

Веб-сервер або просто сервер – це не обладнання, на якому працює служба HTTP, а комп'ютерна програма, що працює під управлінням однієї операційної системи (ОС). Головним призначенням ОС, як відомо [Назаров С.В. Современныеоперационныесистемы [Электронный ресурс] / С.В.Назаров, А.И.Широков // Открытыесистемы. – Режим доступа : http://www.intuit.ru/ department/os/modernos/1/1.html.], є виокремлення і розподіл ресурсів оперативної пам'яті, процесора та дискового простору між різними програмами. Для багатьох користувачів і навіть фахівців була традиційно сформована аксіома, що на одному комп'ютері у певний момент часу працює лише одна ОС. Ця аксіома втратила чинність. Сучасні технології віртуалізації [Виртуализациясерверов [Электронный ресурс] // TrinityGroup. – Режим доступа:http://www.trinitygroup.ru/solution/infrastucture/virtualization/server/.]

дають змогу одночасно працювати на одно-му потужному комп'ютері практично необмеженій кількості різнорідних ОС, що повністю еквівалентно роботі відповідної кіль-кості окремих комп'ютерів. Кожна віртуальна ОС налаштовується незалежно від інших, має власного адміністратора, власний набір IP-адрес [Стек протоколов TCP/IP [Электронный ресурс] // CIT-Forum. – Режим доступа : http://citforum.ru/nets/ip/glava_2.shtml.]. Наприклад, на одному фізичному комп'ютері з однією фізичною мережевою платою може працювати п'ять ОС, кожна із трьома віртуальними мережевими платами, з окремою IP-адресою на кожну плату. При цьому інтернет-користувачі завжди сприйматимуть такий комп'ютер як п'ятнадцять окремих незалежних серверів з окремими IP-адресами. Проте, що насправді працює один комп'ютер із п'ятьма ОС, користувачі ніколи не дізнаються. Очевидно, що вимкнення такого комп'ютера призведе до припинення роботи всіх п'яти ОС, всіх п'ятнадцяти IP-адрес та до відповідних наслідків.

Розглянемо одну ОС та визначимо кількість веб-серверів, які можуть працювати під її управлінням. Веб-сервер приймає від веб-клієнтів команди протокола HTTP, який за стандартною конфігурацією [Полубояров В.В. Введение в технологиисозданияИнтернет-узлов [Электронный ресурс] / В.В.Полубояров // Открытыесистемы. – Режим доступа : http://www.intuit.ru/department/internet/ inwwwtech/1/.] налаштований на використання порта транспортного протоколу TransmissionControlProtocol (TCP) [8], що дорівнює значенню 80. Це дозволяє веб-клієнту за відомою IP-адресою веб-сервера, наприклад 212.109.32.5, відкрити його головну сторінку за такою URL-адресою [Полубояров В.В. Введение в технологиисозданияИнтернет-узлов [Электронный ресурс] / В.В.Полубояров // Открытыесистемы. – Режим доступа : http://www.intuit.ru/department/internet/ inwwwtech/1/.]:

http://212.109.32.5

Якщо адміністратор веб-сервера змінює стандартне значення TCP-порту на інше значення, наприклад 81, то в такому разі URL-адреса ускладнюється:

http://212.109.32.5:81

Досвід свідчить, що саме через ускладнення URL-адреси, яку повинен зберігати і щоразу вводити під час відвідування сайту користувач, веб-сервери використовують лише TCP-порт 80. Особливо це стосується веб-серверів, що обслуговують комерційні проекти. Отже, якщо для звернення до веб-сервера використовувати його IP-адресу і не використовувати TCP-порт, то максимальна кількість веб-серверів у межах однієї ОС дорівнює кількості IP-адрес цієї ОС. Наприклад, якщо ОС має три IP-адреси, то кожна із них може бути прив’язана до окремого веб-сервера або навпаки, до одного спільно-го веб-сервера.

Використання IP-адрес для адресації веб-серверів є доволі незручним через цілу низку причин [Основныепонятия о службе DNS [Элек-тронный ресурс] // Microsoft Technet. – Режим доступа :

http://technet.microsoft.com/ru-ru/library/cc779489(v=ws.10).] і на практиці використовується рідко. Замість цього в URL-адресах використовуються доменні імена [Основныепонятия о службе DNS [Элек-тронный ресурс] // Microsoft Technet. – Режим доступа : http://technet.microsoft.com/ru-ru/library/cc779489(v=ws.10).], напри-клад, http://rada.gov.ua.

Для того, щоб веб-клієнт міг завжди використовувати доменні імена замість IP-адрес, він зобов'язаний бути також клієнтом служби DNS [Основныепонятия о службе DNS [Элек-тронный ресурс] // Microsoft Technet. – Режим доступа : http://technet.microsoft.com/ru-ru/library/cc779489(v=ws.10).]. Винятком є випадок, коли веб-клієнт використовує для інтернет-доступу службу веб-проксі [Веб-прокси и SOCKS-прокси [Электронный ресурс] // FlossManuals. – Режим доступа : http://booki.flossmanuals.net/bypassing-ru/_draft/ _v/1.0/ВЕБ ПРОКСИ/.]. У цьому разі роль DNS-клієнта перекладається на цю службу. Служба DNS є спільною загальносвітовою розподіленою базою даних і має складну структуру, щоб розглядати її у цій статті. Отже, наведемо лише основні функції служби DNS, що притаманні веб-серверам.

Служба DNS ототожнює доменні імена та IP-адреси веб-серверів. Це дає змогу зіставляти довільну кількість доменних імен із однією IP-адресою і навпаки, одне доменне ім'я із довільною кількістю IP-адрес.

Зазначена вище властивість служби DNS дозволяє розгорнути в одній ОС практично необмежену кількість веб-серверів і у такий спосіб обійти наведене вище обмеження за кількістю IP-адрес цієї ОС. При цьому, залежно від налаштувань ОС, в одному граничному випадку кожне доменне ім'я або IP-адреса можуть ідентифікувати окремий незалежний від інших веб-сервер. В іншому граничному випадку всі доменні імена та IP-адреси можуть бути псевдонімами [Основныепонятия о службе DNS [Элек-тронный ресурс] // Microsoft Technet. – Режим доступа :

http://technet.microsoft.com/ru-ru/library/cc779489(v=ws.10).] одного єдиного веб-сервера, як це справедливо, зокрема для сайту Верховної Ради України. Існують також змішані довільні комбінації доменних імен і веб-серверів. Головною умовою при зверненні до кожного веб-сервера є використання в URL-посиланні саме доменного ім'я, а не IP-адреси, що є вимогою протоколу HTTP, на рівні якого і відбувається ідентифікація веб-серверів у межах однієї ОС. Очевидно, що серед кількох веб-серверів однієї ОС лише деякі із них можуть бути нелегальними і заслуговувати на силове втручання, а отже, завершення роботи усієї ОС є далеко не найкращим способом вирішення проблеми.

Розглянемо один веб-сервер. Як незалежна серверна програма він має доволі складну деревоподібну структуру, що складається з кореневого веб-каталогу та ієрархії віртуальних каталогів. Кожен із них пов'язаний із конкретним фізичним каталогом цієї ОС, зі спільним каталогом іншого комп'ютера локальної мережі або віртуальним катало-гом іншого веб-сервера у межах усієї мережі Інтернет. Кожен віртуальний каталог має цілу низку власних налаштувань, зокрема анонімний або авторизований доступ, можливість або неможливість для веб-клієнта перегляду змісту каталогу, файл, що відкривається за замовчуванням. Звернення до віртуального каталогу, що має, наприклад, ім'я "documents" і розташований в іншому віртуальному каталозі "download", здійснюється за таким URL-посиланням:

http://rada.gov.ua/download/documents.

Сукупність віртуальних каталогів веб-сервера та пов'язаних із ним веб-серверів дає змогу увести поняття "інтернет-сайту". Сайт – целогічно завершений, самодостатній проект, що складається із довільної кількості статичних або динамічних веб-сторінок, які відкриває веб-клієнт у результаті своєї активної роботи із сайтом, зокрема при використанні гіперпосилань сайту.

Статичні веб-сторінки не містять програмного коду і становлять собою текстові файли із HTML-розміткою [Полубояров В.В. Введение в

технологиисозданияИнтернет-узлов [Электронный ресурс] / В.В.Полубояров // Открытыесистемы. – Режим доступа : http://www.intuit.ru/department/internet/ inwwwtech/1/.], що зберігаються у віртуальних каталогах веб-сервера і мають розширення .htm або .html. Для того щоб завантажити статичну сторінку з ім'ям, наприклад, "start-page.html", у наведеному вище каталозі, потрібно відкрити URL-посилання http://rada.gov.ua/download/ documents/start-page.html.

Статичні веб-сторінки, про що говорить їх назва, виглядають для всіх веб-клієнтів однаково, незалежно від того, яким чином клієнти відкривають ці сторінки. Веб-сервер зі статичними сторінками фактично становить собою файловий сервер протоколу HTTP і за функціональністю майже не відрізняється від сервера мережевого протоколу FileTransferProtocol (FTP) [Полученийефайлов через FTP [Элект-ронный ресурс] // CIT-Forum. – Режим доступа : http://citforum.ru/internet/ftp/ftpusage.shtml.].

Протилежним випадком до статичних веб-сторінок є динамічні веб-сторінки, які також називають веб-програмами [Яковлев С. Веб-программирование (Обзорнаястатья) [Электронный ресурс] / С.Яковлев // IBM. – Режим доступа : http://wseweb.ru/diz/obzor3.htm.]. Переважна більшість усіх інтернет-сайтів використовує саме динамічні сторінки. При кожному HTTP-запиті клієнта такі сторінки своєчасно формуються постійно діючою на веб-сервері програмою, яка може працювати у різний спосіб залежно від багатьох факто-рів, навіть від версії браузера клієнта. Таким чином, динамічно сформована веб-сторінка з HTML-розміткою фактично становить собою інтерфейс взаємодії користувача з серверною програмою. Сторінки формуються персонально для кожного індивідуального користувача і можуть виглядати зовсім по-різному. Програма працює як чорна скринь-ка, і логіка її роботи є повністю прихованою не лише від користувачів веб-програми, але і від інших адміністраторів і програмістів веб-сервера. Змінювати програму може тільки її розробник-програміст.

Код веб-програми та HTML-розмітка можуть розміщуватись на сервері у вигляді одного файлу, але внаслідок поширеного зараз принципу роздільного коду веб-програми організовують у вигляді мінімум двох файлів – файлу, що містить тільки HTML-розмітку і називається веб-формою, і файлу, що є самим "тілом" веб-програми. Останній, у свою чергу, має або текст із вихідним кодом програми (який може прочитати фахівець) або відкомпільований бінарний масив (який не може бути прочитаний людиною).

Розглянемо, наприклад, два найбільш популярні веб-сервери – Appach [Яковлев С. У истоковApache: История и обзорархитектуры [Электронный ресурс] / С.Яковлев // IBM. – Режим доступа : http://www.ibm.com/developerworks/ru/library/os-apache_3/], що працює на ОС сімейства UNIX [Назаров С.В. Современныеоперационныесистемы [Электронный ресурс] / С.В.Назаров, А.И.Широков // Открытыесистемы. – Режим доступа : http://www.intuit.ru/ department/os/modernos/1/1.html.], і Microsoft InternetInformation Server (Microsoft IIS) [13], який працює на ОС Microsoft Windows Server [Назаров С.В. Современныеоперационныесистемы [Электронный ресурс] / С.В.Назаров, А.И.Широков // Открытыесисте-мы. – Режим доступа : http://www.intuit.ru/ department/os/modernos/1/1.html.]. На веб-сервері Appach обидва файли зазвичай мають роз-ширення .php, програма, як правило, має вигляд вихідного коду, який лише під час клієнтського запиту компілюється і виконується в оперативній пам'яті сервера. Після формування сторінки і направлення її до клієнта відкомпільовані дані відразу вилучаються. Однак програміст-порушник може на-вмисно відкомпілювати програму "вручну", а вихідний код на сервері вилучити. Внаслідок цього логіка роботи його програми за-лишиться невідомою. Прикладом веб-сервера, що працює на програмі Appach, є сайт документації мовою програмування PHP: http://php.net/manual/ru/index.php.

На веб-сервері Microsoft IIS файл веб-форми має розширення .aspx, а програма завжди зберігається у вигляді бінарного масиву з розширенням .dll, що створюється програмістом у результаті компіляції його програми.

Прикладом веб-сервера, що працює на програмі Microsoft IIS, є сайт Посольства Канади в Україні: http://canadainternational.gc.ca/ukraine/visas/index.aspx.

Характерною властивістю усіх веб-програм є можливість приймати від веб-клієнта довільну кількість додаткових пара-метрів, які користувач може явно передати в URL-посиланні до веб-сторінки. Ці параметри є початковими значеннями змінних або констант веб-програми, залежно від яких програма може працювати по-іншому. У наведеному вище прикладі від першого параметра, зокрема, залежить мова сайту Посольства. Якщо деякі параметри користувач не задає, веб-програма використовує для цих змінних значення, закладені розробником. Очевидно, що залежність ро-боти веб-програми від додаткових параметрів є для зловмисника дуже зручним способом приховати справжню логіку роботи не-легального інтернет-сайта і замаскувати такий сайт, наприклад, під соціальну мережу, сайт новин або прогнозу погоди. Сайт може тривалий час надавати користувачам закон-ну інформацію, аж поки до нього не направити HTTP-запит із конкретним значенням потрібного параметра.

Крім явно вказаних додаткових параметрів існують також і неявні, які не передаються в URL-посиланні, але входять до структури HTTP-запита веб-програми. Прикладом такого параметра є попередня веб-сторінка-референт (англ. URL Referrer [Полубояров В.В. Введение в технологиисозданияИнтернет-узлов [Электронный ресурс] / В.В.Полубояров // Открытыесистемы. – Режим доступа : http://www.intuit.ru/department/internet/ inwwwtech/1/.]), на якій здійснено перенаправлення клієнта на цю веб-програму.

Якщо користувач працював із веб-сторінкою А і у результаті його дій відбувся перехід на веб-сторінку Б, то сторінка А є референтом для сторінки Б. Якщо користувач відкрив сторінку Б відразу, а не перейшов до неї зі сторінки А, то у цьому випадку сторінка Б не має референта. Можна дійти висновку: якщо перший користувач перейшов на сторінку Б зі сторінки А,

другий зі сторінки В, а третій відкрив сторінку Б відразу, набравши в браузері її точну URL-адресу, то для трьох користувачів сторінка Б може працювати абсолютно по-різному.

Найбільш серйозну проблему для компетентних служб становить широке використання веб-адміністраторами згаданої вище служби DNS. Користувачі інтернет-сайтів майже завжди ідентифікують і запам'ятовують сайти саме за їх доменними іменами, а не за IP-адресами. По-перше, це означає, що веб-адміністратор має змогу змінювати IP-адресу, а отже, фізичне місце-знаходження веб-сервера, на свій власний розсуд. По-друге, з міркувань розподілу на-вантаження або забезпечення відмовостій-кості, зокрема від втручання сторонніх осіб, веб-адміністратор може ввести до експлуатації довільну кількість веб-серверів із різ-ними IP-адресами, причому в службі DNS може бути зареєстрована лише частина цих веб-серверів.

Веб-адміністратор може обрати будь-яку всесвітньо відому інтернет-компанію на зразок американської "Go!Daddy" для реєстрації власного DNS-домена другого рівня [Основныепонятия о службе DNS [Электронный ресурс] // Microsoft Technet. – Режим доступа : http://technet.microsoft.com/ru-ru/library/cc779489(v=ws.10).], наприклад, PINOKKIO.COM у домені першого рівня COM [Основныепонятия о службе DNS [Элек-тронный ресурс] // Microsoft Technet. – Режим доступа : http://technet.microsoft.com/ru-ru/library/cc779489(v=ws.10).]. Потім адміністратор реєструє доменне ім'я інтернет-сайту WWW.PINOKKIO.COM , розгортає три веб-сервери, що розташовані у різних підприємствах або домашніх мережах України або іншої держави. IP-адреси першого і другого веб-серверів адміністратор прив'язує до доменного ім'я WWW.PINOKKIO.COM, IP-адреса третього сервера не прив'язується, а залишається "на потім", якщо за певної при-чини припинить працювати перший або другий сервер. За такої конфігурації одна половина користувачів інтернет-сайту з'єднається з першим сервером, друга поло-вина із другим сервером, а третій сервер використовуватись не буде і ніхто з допитливих

користувачів ніколи не дізнається про його існування. Відповідальний адміністратор, зацікавлений в безперервній роботі інтернет-сайту, обов'язково налаштує спеціальний програмний монітор, який у випадку припинення нормальної роботи одного із веб-серверів відразу його сповістить про це за допомогою sms-повідомлення. Коли один із серверів припиняє роботу, адміністратор відразу вносить відповідні зміни у конфігурацію доменного ім'я WWW.PINOKKIO.COM, у результаті чого до цього доменного ім'я залишаються прив'язаними лише IP-адреси працюючих серверів. Адміністратор може також налаштувати автоматичне виконання цієї операції.

При використанні певної кількості веб-серверів адміністратор інтернет-сайту повинен врахувати те, що майже 90% користувачів не знають, що таке IP-адреса і не бажають це знати; їм не цікаво, яка кількість веб-серверів забезпечує роботу сайту, але для користувача сайт повинен працювати постійно, надійно і в єдиний спосіб, незалежно від того, з яким саме сервером з'єднається користувач. Групу однаково налаштованих серверів умовно називають фермою серверів. Інакше кажучи, адміністратор повинен забезпечити синхронізацію даних між усіма веб-серверами ферми. Виявляється, що за допомогою сучасних інформаційних технологій це завдання вирішити абсолютно не складно.

Головний принцип ферми серверів поля-гає у територіальному розподілі програми і даних, з якими ця програма працює. Розроблену і відлагоджену веб-програму розміщують на кількох веб-серверах, на яких, крім самої веб-програми, ніяких даних не зберігається. Отже, якщо один із таких веб-серверів потрапить до рук правоохоронців, то не зможе бути використаний як речовий доказ. Для даних, які отримує і зберігає кожна копія веб-програми, виділяють окремий сервер, який спільно використовується усіма веб-серверами – так зване спільне джерело даних. На ньому розгортають систему управління базами даних (СУБД) на основі такого програмного забезпечення, як FoxPro, SQL, LDAP або Oracle. Кожен

веб-сервер має авторизований (захищений) мережевий доступ до серверів СУБД. Інформація, яку записує до бази даних один веб-сервер, є доступною для всіх інших веб-серверів, а отже, незалежно від того, з яким саме сервером з'єднався веб-користувач, інтернет-сайт працюватиме для нього абсолютно однаково і проблеми синхронізації да-них між веб-серверами ферми не виникне.

Альтернативою сервера СУБД може бу-ти так звана веб-служба [Ньюкомер Э. Веб-сервисы: XML, WSDL, SOAP и UDDI / Э.Ньюкомер. – СПб. : Питер, 2003. – 256 с]. Це постійно діюча спеціалізована веб-програма, яка не має інтерфейсу користувача, через що користувачі працювати безпосередньо з цією веб-програмою не можуть. Замість цього веб-служба має певний набір функцій спеціального формату і синтаксису, які можуть бути викликані іншими веб-програмами, зокрема тими, що працюють на веб-серверах ферми. Коли веб-серверу, з яким працює користувач, потрібно отримати або зберегти дані, він з'єднується з веб-службою і викликає відповідну функцію. Веб-служба може використовувати для накопичення даних сервер СУБД, файловий сервер, іншу веб-службу або будь-яке інше джерело даних. Уся внутрішня інфраструктура інтернет-сайту повністю прихована від користувача. Про факт з'єднання веб-серверів із спільними джерелами даних можна дізнатися лише аналізуючи мережевий трафік на тих підприємствах, де встановлені і налаштовані веб-сервери.

Нарешті, ферму можуть формувати взагалі не веб-сервери, а так звані проксі-сервери (англ. proxy – представник). Проксі-сервер не містить ні даних, ні програмного коду. Він є лише низькорівневим мережевим ретранслятором, що приймає мережеві запити від клієнта і передає їх на інший сервер, IP-адреса та місцезнаходження якого залишаються для клієнта невідомими. Цей інший сервер, у свою чергу, може бути або веб-сервером, або іншим проксі-сервером, що утворює з першим проксі ланцюжок мережевих з'єднань. Служба проксі має два типи – SOCKS-проксі і веб-проксі [Веб-прокси и SOCKS-прокси [Электрон-ный ресурс] //

FlossManuals. – Режим доступа : http://booki.flossmanuals.net/bypassing-ru/_draft/ _v/1.0/ВЕБ ПРОКСИ/.], які ретранслюють мережеві запити клієнтів за інформацією протоколу транспортного рів-ня TCP та програмного рівня HTTP відповідно. Для уточнення цього факту службу SOCKS-проксі також називають TCP-проксі, а веб-проксі – HTTP-проксі.

З огляду на зазначене доходимо висновку, що мережа Інтернет має логічну (глобальну) та фізичну (регіональну) інфраструктури, взаємно незалежні одна від одної. Інтернет-сайт є елементом логічної інфраструктури, тісно пов'язаної з реєстрацією доменного ім'я в службі DNS. Натомість веб-сервер є елементом регіональної інфраструктури, який по суті є лише одним екземпляром інтернет-сайту, що працює у конкретному регіоні.

1.3.Визначення можлив напрямків рішення поставленої задачі

Сучасні веб і мобільні застосунки зазвичай складаються з двох частин: клієнтська частина і серверна. У якості клієнтської частини виступає веб або мобільний додаток, доступ до якого має користувач. Якщо це веб-застосунок, у якості клієнта маємо веб-сайт, яким може скористатись користувач (перейти на необхідний URL [10]). Якщо це мобільний застосунок, у якості клієнта виступає додаток, який користувачу необхідно заздалегідь встановити на свій пристрій або з магазину додатків (App Store на iOS [11] і Play Market [12] на Android). Розподілення логіки усієї системи на клієнтську та серверну частини допомагає: ● розділити логіку запитів у базу даних та у інші необхідні сервіси і логіку відображення користувацього інтерфейсу (з англійської user interface або UI [13]); ● зробити логику і відображення незалежними одне від одного, що дозволяє змінювати, наприклад, технології відображення (клієнт), не змінюючи логику (сервер); ● інкапсулювати логіку - користувач не зможе взаємодіяти із серверною частиною напряму - лише через клієнтську частину, у спосіб, визначений

розробниками цієї клієнтської частини. У клієнт-серверній системі користувач має доступ лише до клієнтської частини, з якою і взаємодіє за допомогою користувацького інтерфейсу.

А вже клієнтська частина може “спілкуватись” із серверною за допомогою API (з 10 англійської Application Programming Interface) [14] - інтерфейсу, визначеному розробниками серверної частини системи. Виконувати обробку і оптимізацію зображень і відео можна виконувати на стороні клієнта (веб або мобільний додаток) або на стороні сервера. Спочатку розглянемо обробку на стороні клієнта. У першу чергу слід зазначити, що при обробці на стороні клієнта реалізація обробки зображень і відео буде різнитись в залежності від технологій, використовуваних для розробки клієнтської частини. Наприклад, якщо клієнтська частина системи працює на операційній системі iOS і написана на мові програмування Swift [15], реалізація буде одною, а якщо вона працює на ОС Android і написана на мові Java [16], то іншою. Ситуація ускладнюється тим, що технології для розробки під різні операційні системи не обмежуються двома мовами програмування - зараз існує багато фреймворків для розробки під обидві операційні системи.

Наприклад, фреймворк React Native [17] дозволяє створювати кросплатформені мобільні додатки на iOS і Android, або ж додатки для обох ОС можуть бути написані на мові програмування Kotlin [18], або ж вони можуть бути створені трохи згодом, використовуючи якусь нову технологію, зараз навіть не існує. Тобто, реалізовуючи обробку і оптимізацію зображень і відео на різних ОС з використанням різних технологій, неможливо зробити цю оптимізацію універсальною - її необхідно буде розробляти окремо для кожного застосунку, написаному на якійсь окремій мові програмування. Вже на цьому етапі стає зрозумілим, що реалізація оптимізації на клієнтській частині результує у складний процес розробки не універсальної системи, яку необхідно буде підтримувати (оновлювати і дороблятиі) разом із всією клієнтською частиною. Але може бути таке, що оптимізація зображень і відео на клієнті буде дуже ефективною, тому поки ще розглянемо цей варіант.

Для дослідження оптимізації у мобільних додатках візьмемо стандартні інструменти розробки під кожну з найпопулярніших мобільних ОС: мову Swift для iOS і мову Java для Android. Для оптимізації зображень на iOS у мові програмування Swift існує об’єкт UIImage [19], за допомогою якого можна зменшити розмір зображення, отримуючи на виході оптимізоване зображення. Але мінус полягає у тому, що при обробці зображення за допомогою UIImage iOS все одно декодує оригінальне зображення, загружаючи його в оперативну пам’ять пристрою. Це результує у велике навантаження на систему у момент відгрузки зображення.

Окрім того, силу стиснення зображення не можна конфігурувати. Що стосується оптимізації відео, то iOS оптимізує його автоматично при виборі відео з галереї, але силу стиснення також не можна конфігурувати. Для оптимізації на ОС Android в мові програмування Java існує клас BitmapFactory [20], за допомогою якого можна декодувати зображення у файл із меншим розширенням. Але за допомогою цього класу не можна налаштовувати силу стиснення зображень. Автоматичної оптимізації відео в Android, на відміну від iOS, немає. У веб-застосунках оптимізація зображень або відео взагалі не підтримується. Це означає, що оптимізація на стороні клієнта неможлива через ряд факторів: ● оптимізація зовсім не підтримується у веб-застосунках; ● оптимізація зображень не може налаштовуватись на iOS і Android; ● оптимізація відео на Android відсутня; ● оптимізація зображень на iOS і Android не може обробляти вже відвантажені, існуючі, файли. 12 Розглянемо обробку і оптимізацію на стороні сервера. Оскільки сервер розроблюваної системи може працювати на будь-якій операційній системі, незалежно від усіх інших частин існуючої системи, ми можемо обрати ту операційну систему, яку забажаємо.

Обробка зображень і відео на стороні сервера не лімітує нас вищеперерахованими інструментами, які існують в мобільних ОС, бо ми можемо реалізувати оптимізацію на будь-якій ОС і на будь-якій мові програмування. Окрім того, ця логіка на серверній частині дозволяє

виконувати оптимізацію зображень і відео навіть для веб-застосунків. Логіка оптимізації виглядає наступним чином:

1. Клієнт надсилає зображення або відео на сервер;

2. Сервер надсилає зображення або відео системі для оптимізації, що розробляється;

3. Система оптимізує зображення або відео, надсилає його назад на сервер;

4. Сервер повідомляє клієнт, що файл успішно відвантажений та оптимізований. Щоправда, згідно із цією логікою клієнт все одно буде чекати на оптимізацію файлу (на сервері) і так система не зможе обробляти попередньо завантажені файли. Окрім того, при досить великій кількості одночасно обробляємих файлів система може вийти з ладу через нестачу обчислювальних ресурсів. Щоб вирішити ці проблеми, потрібно використовувати черги. Це такі спеціальні системи, що приймають якісь дані на вхід, і по черзі обробляють їх. Відмінною рисою черг є те, що вони дозволяють обробляти не більше 13 якоїсь встановленої кількості об’єктів одночасно. Це допоможе не перегрузити систему великою кількістю файлів - всі вони просто будуть у черзі чекати на обробку. Також з використанням черг можна не чекати на закінчення обробки на сервері перед посиланням відповіді на клієнт, бо у цьому більше немає необхідності. Ще один великий плюс черг у тому, що у них можна вручну або автоматично завантажити попередньо завантажені файли, і обробник черг оптимізує їх за якийсь кінцевий проміжок часу.

Тож з використанням черг логіка буде наступною:

1. Клієнт надсилає зображення або відео на сервер;

2. Сервер додає зображення або відео у чергу;

3. Сервер швидко відповідає клієнту, що файл був завантажений;

4. Клієнт працює з поки не обробленим файлом;

5. Із черги файл потрапляє в обробник системи, що розробляється, який оптимізує файл;

6. Після обробки файл замінює раніше відвантажений файл;

7. При наступному запиті на сервер клієнт отримує вже оптимізований файл.

Таких підхід задовольняє всі вимоги до системи, що розробляється:

- оптимізація зображень або відео не залежить ні від клієнтської, ні від серверної частини;
- оптимізація буде працювати як із мобільними, так і з веб-застосунками;
- оптимізація файлів не створить великого навантаження на систему через використання черг;
- клієнтська частина зможе працювати із файлом одразу, не чекаючи на його оптимізацію (що створює гарний UX);
- система зможе оптимізувати вже існуючі фото або відео файли;
- оптимізація буде конфігуруємою
- ми зможемо наташтовувати силу стиснення фото або відео

РОЗДІЛ 2. ІНСТРУМЕНТИ ТА МЕТОДИ ДЛЯ ОБРОБКИ ЗОБРАЖЕНЬ І ВІДЕО

2.1. Методи оптимізації зображень

Оптимізація графіки починається з вибору формату, в якому вона буде поміщена на Web-сторінку. В даний момент в Мережі використовується три 122 графічні формати: GIF, JPEG і PNG. Кожен з них має сенс застосовувати в певних ситуаціях. Формат GIF. Абревіатура GIF розшифровується як Graphics Interchange Format - формат для обміну графікою. Це растровий формат, колірний діапазон якого обмежений 256 кольорами, т. К. Для зберігання інформації про колір використовуються тільки 8 бітів. Для зменшення розміру графічних файлів можливо скоротити кількість використовуваних квітів до 2. Переваги формату GIF: підтримує прозорість. Крізь пікселі, яким призначено прозорий− колір, будуть видні нижележащие об'єкти, або фон. Однак в зображеннях формату GIF може використовуватися тільки один рівень прозорості - прозорість 100%, на відміну від формату PNG, який підтримує 256 рівнів прозорості. черезрядковий. При включенні цієї опції зображення буде поступово− збільшувати чіткість у міру його завантаження. Спочатку будуть відображена кожна 8 рядок, потім кожна 4, кожна 2 і, нарешті, буде виведено повне зображення. Таким чином, зображення з'являється на екрані майже відразу після початку завантаження сторінки і, не чекаючи повного завантаження, можна зрозуміти, що представлено на зображенні; підтримка анімації. Анімація підтримується у версії формату GIF89a.− У цьому випадку зображення представляється у вигляді змінюють один одного кадрів Стиснення в форматі GIF здійснюється по рядках, тобто якщо рядок має однорідний колір, то при збереженні в GIF до неї буде застосовано стиснення. Якщо однорідний колір використовується в шпальтах, то стиснення немає. Формат GIF слід використовувати в тому випадку, якщо колірний діапазон вихідних зображень не перевищує 256 кольорів, або

кількість кольорів може бути зменшено за рахунок значного зменшення якості. Це, як правило, зображення з великими площами однорідних одноколірних областей, зображення, отримані конвертацією з векторних форматів, зображення з текстом. 123 Для повнокольорових зображень, в тому числі для фотографій, формат GIF малоприйнятний. У цьому випадку слід використовувати інші формати стиснення. Формат JPEG JPEG розшифровується як Joint Photographic Experts Group - об'єднана група експертів в області фотографії. Зображення в форматі JPEG підтримують 24-бітові кольори, і внаслідок цього їм добре користуватися для збереження повнокольорових зображень. Формат передбачає стиснення з втратами. JPEG-стиснення засноване на розкладанні зображень на складові, близькі до тих, які використовуються в людському зорі при відкиданні інформації, що не позначається на зоровому сприйнятті образу. За рахунок цього досягається висока стиснення зображень при незначному погіршенні якості. Ступінь стиснення і якість зображень знаходяться в зворотній залежності: чим сильніше стисло зображення, тим нижче його якість.

Зазвичай ці параметри визначаються у відсотках в діапазоні від 0 до 100. JPEG добре підходить для зображень з багатою колірною гамою, плавним переходом кольорів, для фотографій і зображень з градієнтними областями. Не слід використовувати JPEG для стиснення зображень, колірна гамма яких обмежена кількома кольорами, зображень з дрібним текстом, зображень, які повинні зберегти чіткі межі або містять дрібні деталі. Формат PNG Формат PNG розшифровується як Portable Network Graphics - переноситься мережева графіка. Це відносно новий формат, покликаний замінити собою формат GIF. Формат PNG існує в двох варіантах PNG-8 і PNG-24. PNG-8 практично повністю аналогічний формату GIF, за винятком поліпшеного стиснення і відсутності можливості анімації. PNG-24 має низку додаткових переваг, таких як: наявність альфа-прозорості - методу визначення прозорих областей,− який на відміну від формату GIF забезпечує

256 рівнів прозорості; існування гамма-корекції - автоматичної корекції яскравості− зображення при відтворенні на різних системах; застосування поліпшеного стиснення.− Поширення формату PNG стримується старими версіями браузерів, які не підтримують даний формат, а також недостатньою і неповною 124 підтримкою можливостей PNG в нових версіях. Так, наприклад, йде справа з альфа-прозорістю, підтримка якої відсутній в браузерах. При використанні формату PNG-24 для стиснення повнокольорових зображень він програє формату JPEG в розмірі створеного файлу, т. К. Використовує стиснення без втрат. PNG-24 рекомендується вибирати для повнокольорових зображень з чіткими краями і дрібними деталями, зображень з дрібним текстом, а також для зображень з прозорими областями. Ще один важливий спосіб оптимізації графіки - це зменшення її розмірів по ширині і висоті і подальше розтягування до вихідних розмірів засобами браузера. Цей спосіб застосовується, для елементів оформлення сайту: фонових малюнків таблиць, розділових смуг тощо

Заснований він на наступних особливостях відображення HTML: якщо малюнок, вставлений за допомогою тега по ширині або висоті менше, ніж зазначено у властивостях height і width, то він буде розтягнутий (або навпаки, стислий) до потрібного розміру, а якщо малюнок описаний як фон таблиці, комірки або іншого об'єкта у властивості «background», то він буде повторюватися до тих пір, поки не заповнить весь об'єкт. В результаті замість того, щоб зберігати в файл весь елемент цілком, досить вирізати його невеликий шматок, а далі розтягнути або розмножити безпосередньо в браузері за допомогою властивостей, описаних вище. Наступний спосіб оптимізації графіки - нарізування великих зображень на «скибочки». Сам по собі цей метод майже неефективний, так як кожен «скибочку» займає приблизно такий же обсяг, як і в оригінальному документі, крім того, до нього додається заголовок файлу і необхідність виконати додатковий HTTP-запит для отримання ще одного файлу (що додає ще приблизно 1 Kb

переданої / прийнятої інформації). Виняток становлять лише ті випадки, коли зображення містить різкі переходи між різними областями, які збігаються з межами «скибочок», тому що в цьому випадку можливе збільшення ефективності стиснення. Однак якщо в розрізати зображенні є однорідні області, то можна застосовувати поєднання цього методу з методом розтягування в браузері, описаним вище, і тоді його ефективність значно підвищується. 125 Говорячи про програмне забезпечення для оптимізації графічних зображень необхідно перерахувати найбільш популярні безкоштовні програми: Smush.it! - Використовує методи оптимізації для конкретного− формату зображення, щоб видалити непотрібні байти з файлів зображень; RIOT - абревіатура від Інструмент Радикальної Оптимізації− Зображень. Оптимізатор картинок для користувачів Windows, який доступна як окремий додаток або як розширення IrfanView; PNGOUT - є загальнодоступним, безкоштовним, без надмірностей,− інструментом для оптимізації ваших зображень; Online Image Optimizer від Dynamic Drive являє собою вебінструмент для стиснення зображень;− uperGIF - це безкоштовна утиліта для Windows і Mac OS, яка− допомагає оптимізувати GIF; PNGGauntlet - це .NET додаток для PNGOUT. Воно може бути− використано для перетворення JPG, GIF, TGA, PCX, BMP і оптимізації вмісту PNG файлів. SuperPNG - безкоштовний плагін для Photoshop, що дозволяє− сохраненять PNG в значно більш компактному вигляді. Оптимізація графіки сайту - досить трудомістка робота, що вимагає як певних знань, так і вміння підбирати оптимальні параметри експериментально, але ретельне виконання цієї роботи дасть свій результат: чи не буде ситуацій, коли не дочекалися закінчення завантаження користувачі будуть закривати браузер, так і не побачивши всього того , що ви хотіли їм запропонувати. Крім цього, якщо

ваш сайт стоїть на хостингу з оплатою по трафіку, ваші витрати на оплату хостингу зменшаться, так як на кожного відвідувача доведеться менший обсяг трафіку при тій же самій його інформативності.

Оптимізувати зображення можна декількома способами:

- Без зміни вихідних файлів - файли перетворюються сервером на льоту.
- Зі зміною вихідних файлів - файли перетворюються заздалегідь за допомогою спеціалізованих утиліт.

Без зміни вихідних файлів

Оптимізація без зміни вихідних файлів можлива завдяки модулю PageSpeed. Для цього в його налаштуваннях можна включити такі опції:

- **inline_images** - дозволяє вбудовувати невеликі зображення безпосередньо в HTML у вигляді data: URL, Що скорочує число запитів до сервера, необхідне для побудови сторінки.
- **recompress_images** - прискорює завантаження зображень, на льоту видаляючи з них надлишкову інформацію (метадані та колірні профілі ICC) і перетворюючи в найбільш підходящий формат (включає в себе конвертацію в формат WebP, якщо браузер клієнта його підтримує).
- **resize_images** - прискорює завантаження великих зображень, стискаючи їх до того розміру, в якому вони відображаються на сторінці.

Зі зміною вихідних файлів

На серверах хостингу присутній ряд утиліт, За допомогою яких можна оптимізувати файли зображень.

- PNG:
  - pngcrush
  - optipng
- GIF:
  - gifsicle

- JPG/JPEG:
  - jpegoptim
  - jpegtran

Ними можна скористатися безпосередньо через консоль, підключившись до хостингу по SSH. Також можна створити власний скрипт, який буде звертатися до них і виконувати необхідні операції. Існує безліч вже готових скриптів для оптимізації зображень, Ці скрипти зазвичай включають в себе лише набір команд для зазначених утиліт і по черзі запуск для відповідних зображень.

Використання скриптів

Наприклад, для оптимізації файлів JPG / JPEG і PNG можна використовувати такий bash-скрипт.

Після оптимізації все оптимізовані файли будуть поміщені в один з каталогів:

- Або в каталог, вказаний після ключа -o.
- Або в підкаталог output поточного каталогу, якщо ключ -o зазначений не був.

Щоб завантажити і запустити скрипт оптимізації, виконайте команди:

```
wget https://gist.githubusercontent.com/lgiraudel/6065155/raw/24f667559eee61dd00a99a9940e06b46a125d3ec/optimize.sh
sh optimize.sh -i ~/example.com/images/input -o ~/example.com/images/output
```

Параметри:

- -i або --input - визначає каталог, в якому розташовані оптимізуються зображення.
- -o або --output - визначає каталог, куди будуть зберігатися всі оптимізовані зображення.
- -q або --quiet - відключає виведення всіх дій.

- -s або --no-stats - відключає висновок статистики розміру.
- -h або --help - висновок довідки за вказаними ключам.

Скрипт використовує утиліти:

- pngcrush
- optipng
- jpegtran

Робота з утилітами pngcrush

Утиліта для оптимізації зображень без втрати якості. синтаксис:

```
pngcrush опції вихідний_файл.png оптимізований_файл.png
```

Варіанти використання утиліти:

- Максимальне стиснення без втрати якості:

```
pngcrush -rem alla -rem text -reduce -brute початковий_файл.png оптимізований_файл.png
```

- Оптимізація файлів в каталозі без переміщення:

```
for file in ~/example.com/www/images/*.png ; do pngcrush -reduce -brute -rem alla -rem gAMA -rem cHRM -rem iCCP -rem sRGB "$file" "${file%.png}-crushed.png" && mv "${file%.png}-crushed.png" "$file" ; done
```

o Замість example.com/www/images вкажіть шлях до каталогу з зображеннями. Зверніть увагу, оптимізація не проводиться рекурсивно по всіх каталогах, а обмежується тільки зазначеним.

o Кінцевий оптимізований файл замінить вихідний.

Більш детальну інформацію по роботі з утилітою можна отримати за допомогою команди:

```
pngcrush --help
```

optipng

Утиліта стиснення та оптимізації зображення, створена на основі pngcrush. Формати зображень, які можуть бути перетворені в оптимізований PNG:

- PNG
- BMP
- GIF
- PNM
- TIFF

Для оптимізації одиничного файлу потрібно виконати команду (Файл автоматично буде перетворений і замінений оптимізованою версією):

```
optipng ~/example.com/www/images/image.png
```

Для оптимізації всіх файлів в одному каталозі потрібно виконати команду:

```
find ~/example.com/www/images/ -iname *.png -print0 | xargs -0 optipng -o7
```

- Замість example.com/www/images/ вкажіть шлях до каталогу зображень.
- Замість *.png вкажіть потрібний формат файлів у вигляді *.bmp або інші.

Більш детальну інформацію по роботі з утилітою можна отримати, виконавши команду:

```
optipng --help
```

gifsicle

Утиліта для роботи з анімованими GIF-файлами. З її допомогою можна проводити безліч дій: оптимізувати, масштабувати, обрізати.

Оптимізація GIF-анімації з втратою якості:

```
gifsicle -03 --lossy=80 -o оптимізований_файл.gif вихідний_файл.gif
```

- Для вказівки ступеня стиснення потрібно змінити значення параметра –lossy=XX.

Більш детальну інформацію по роботі з утилітою можна отримати, виконавши команду:

```
gifsicle --help
```

jpegoptim

Утиліта для оптимізації зображень JPEG і JPG.

Популярні способи використання утиліти:

- Оптимізація одиничного файлу:

```
jpegoptim image.jpg
```

o Для оптимізації всіх файлів .jpg в каталозі потрібно виконати команду так:

```
jpegoptim ~/example.com/www/images/*.jpg
```

▪ Замість example.com/www/images/ вкажіть шлях до каталогу зображень.

- Видалення всіх метаданих зображення:

```
jpegoptim image.jpg --strip-all
```

- Перетворення в прогресивний JPEG:

```
jpegoptim image.jpg --all-progressive
```

- Оптимізація зображення з втратою якості:

```
jpegoptim -m85 image.jpg
```

o Для управління рівнем якості оптимізованого зображення потрібно змінити значення ключа -mXX.

Більш детальну інформацію по роботі з утилітою можна отримати, виконавши команду:

```
jpegoptim --help
```

jpegtran

Ще одна утиліта для оптимізації зображень JPEG і JPG без втрати якості.

Популярні способи використання утиліти:

- Оптимізація одиничного файлу:

```
jpegtran -copy none -optimize -outfile оптімізірофанний_файл.jpg вихідний_файл.jpg
```

o Видалення метаданих зображення виробляються за допомогою ключа -copy none.

- Оптимізація і перетворення в прогресивний JPEG:

```
jpegtran -copy none -optimize -progressive -outfile оптімізірофанний_файл.jpg вихідний_файл.jpg
```

- Оптимізація зі зміною якості:

```
jpegtran -quality 80 -copy none -optimize -outfile оптімізірофанний_файл.jpg вихідний_файл.jpg
```

o Для управління рівнем погіршення якості потрібно змінити значення ключа -quality XX.

Більш детальну інформацію по роботі з утилітою можна отримати, виконавши команду:

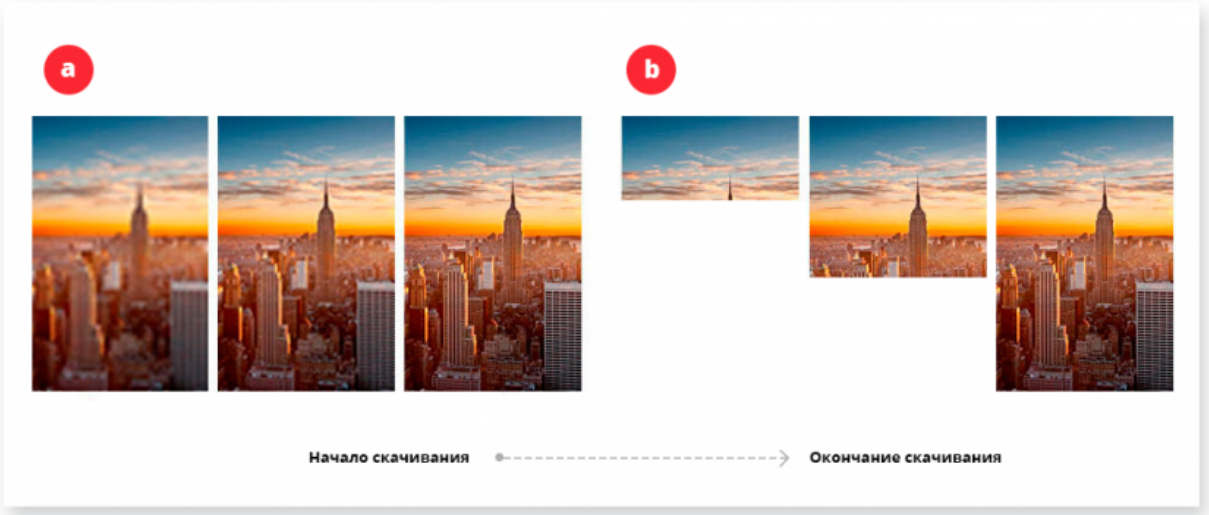
```
jpegtran --help
```

Вибір оптимального формату зображення дозволяє помітно прискорити завантаження веб-сторінки без шкоди для якості картинки. У більшості користувачів сформувалися високі вимоги до цих чинників. Щоб їх задовольнити, необхідно дотримуватися балансу між часом завантаження і прийнятними візуальними характеристиками зображень.

Насправді не існує поняття "кращий формат", але для конкретного типу зображення можна вибрати найбільш підходящий в поєднанні з ефективними методами стиснення. Розглянемо кілька популярних графічних форматів:

- GraphicsInterchangeFormat (GIF) - цей формат зображення найкраще підходить для зображень, в яких використовується 1-2 кольори, наприклад, логотипи або анімація. Особливості: прозорий фон, відсутня змішання кольорів.
- PortableNetworkGraphics (PNG) - забезпечує високу якість, але при цьому генерує файли великого розміру, які часто не підходять для використання в мережі. Існують винятки, наприклад, коли потрібно надзвичайно високу якість: твори мистецтва (картини).
- JointPhotographicExpertsGroup (JPEG) - це бажаний формат для зображень в мережі Інтернет. Показує велику різноманітність кольорів і генерує файли меншого розміру, ніж PNG. Цей формат чудово підходить для фотографій, а також для ілюстрацій, де присутній градієнт, тіні і т. п.

Зберігати зображення краще із застосуванням прогресивної схеми стиснення (progressive JPEG, див. мал.3a) замість базового режиму кодування (baseline JPEG, див. мал.3b). У такому випадку малюнок буде завантажуватися швидше, поступово збільшуючи ступінь деталізації. (див. *мал.3a*).



Мал. 3 - a (прогресивне стиснення JPEG); b (базове стиснення JPEG)

Розмір одного і того ж зображення при використанні різних форматів: GIF (256 кольорів) - 42K; PNG-8 (256 кольорів) - 37K; PNG-24 - 146K; JPG (60 quality) - 32K[22].

2.2. Стиск мультимедійних даних

Застосовується для більш раціонального використання пристроїв зберігання і передачі даних. Синоніми — упакування даних, компресія, стискаюче кодування джерела. Зворотна процедура називається відновленням даних (розпакуванням, декомпресією). Стиск заснований на усуненні надмірності, що міститься у вихідних даних. Найпростішим прикладом надмірності є повторення в тексті фрагментів (наприклад, слів природньої або машинної мови). Подібна надмірність зазвичай усувається заміною повторюваної послідовності посиланням на вже закодований фрагмент із зазначенням його довжини. Інший вид надмірності пов’язаний з тим, що деякі значення в даних, які стискаються, зустрічаються частіше за інші. Скорочення об’єму даних досягається за рахунок заміни, даних, що часто зустрічаються, короткими кодовими словами, а рідкісних — довгими (ентропійне кодування). Стиск даних, які не мають властивості надмірності (наприклад, випадковий сигнал або білий шум, зашифровані повідомлення),

принципово неможливий без втрат. Принципи стиску даних В основі будь-якого способу стиску лежить модель надмірності вихідних даних. Іншими словами, для стиску даних використовуються деякі апріорні відомості про те, якого роду дані стискаються. Не володіючи такими відомостями про джерело, неможливо зробити ніяких припущень про перетворення, яке дозволило б зменшити об’єм повідомлення.

Модель надмірності може бути статичною, незмінною для всього повідомлення, що стискається, або будуватися чи параметризуватися на етапі стиску (і відновлення). Методи, що дозволяють на основі вхідних даних змінювати модель надмірності інформації, називаються адаптивними. Неадаптивними є зазвичай вузькоспеціалізовані алгоритми, застосовувані для роботи з даними, що мають добре відомі і незмінні характеристики. Більшість досить універсальних алгоритмів є тією чи іншою мірою адаптивними. Види стиску даних Усі методи стиску даних діляться на два основні класи: · стиск без втрат (lossless) — при якому можливо повне відновлення вихідних даних; · стиск із втратами (lossy) — дозволяє відновити дані з викривленнями, зазвичай несуттєвими з погляду сприймання користувачем і подальшого використання відновлених даних.

Стиск без втрат зазвичай використовується для передачі і зберігання текстових даних, комп’ютерних програм, рідше — для скорочення об’єму аудіо- і відеоданих, цифрових фотографій і т. п., у випадках, коли викривлення неприпустимі або небажані. Стиск із втратами відіграє більш важливу роль у мультимедіа, оскільки, як правило, має значно більшу ефективність, дозволяючи, таким чином, сильніше упаковувати мультимедійний контент, зберігаючи при цьому показники його якості на необхідному рівні. Стиск із втратами зазвичай застосовується для скорочення об’єму аудіо- і відеоданих і цифрових фотографій у тих випадках, коли таке скорочення є пріоритетним, а повна відповідність вихідних і відновлених даних не потрібна. Коефіцієнт стиску Коефіцієнт стиску — основна

характеристика будь-якого алгоритму стиску. Вона зазвичай визначається як відношення об’єму вихідних незжатих даних до обсягу стислих:

c o S S k = , де k — коефіцієнт стиску, So — об’єм вихідних даних, а Sc — об’єм стиснутих. При такому визначенні коефіцієнт стиску показує число разів, у яке вдалося стиснути дані з використанням певного алгоритму. Чим вищій коефіцієнт стиску, тем алгоритм ефективніший.

Слід зазначити: якщо k = 1, то алгоритм не робить стиску, тобто вихідне повідомлення виявляється за обсягом рівним вхідному; якщо k < 1, то алгоритм породжує повідомлення більшого розміру, ніж незжате, тобто, робить «шкідливу» роботу. Ситуація з k < 1 цілком можлива при стиску. Але навіть коли алгоритм стиску збільшує розмір вихідних даних, легко домогтися того, щоб їхній об’єм гарантоване не міг збільшитися більш, ніж на 1 біт. Тобто зробити так, щоб навіть у самому гіршому випадку мала місце нерівність: +1 ³ o o S S k , Робиться це в такий спосіб: якщо об’єм стиснутих даних менший, ніж об’єм вихідних, потрібно повернути стиснуті дані, додавши до них «1», інакше повернути вихідні дані, додавши до них «0». Коефіцієнт стиску може бути постійним або змінним. У другому випадку він може бути визначений або для кожного конкретного повідомлення, або оцінений за деякими критеріями: · середній (звичайно по деякому тестовому набору даних); · максимальний (випадок найкращого стиску); · мінімальний (випадок найгіршого стиску); або яким-небудь іншим. Коефіцієнт стиску із втратами при цьому сильно залежить від припустимої погрішності стиску або якості, яка зазвичай виступає як параметр алгоритму.

У загальному випадку постійний коефіцієнт стиски здатні забезпечити тільки методи стиску даних із втратами. Допустимість втрат при стиску Основним критерієм відмінності між алгоритмами стиску є описана вище наявність або відсутність втрат. У загальному випадку алгоритми стиску без втрат універсальні в тому розумінні, що їхнє застосування безумовно можливе для даних будь-якого типу, у той час як можливість застосування стиску із втратами повинна бути обґрунтована. Для деяких

типів даних викривлення не припустимі в принципі. У їхньому числі: · символічні дані, зміна яких неминуче приводить до зміни їхньої семантики: програми та їхні вихідні тексти, двійкові масиви і т. п.; · життєво важливі дані, зміни в яких можуть призвести до критичних помилок: наприклад, ті, що отримуються з медичної вимірювальної апаратури або контрольних приладів літальних, космічних апаратів і т. п.; · проміжні дані, що багаторазово зазнають стиску і відновлення при багатоетапній обробці (графічні, звукові, відеодані) (наявність втрат, несуттєвих у випадку однократної обробки, при багаторазовій обробці приводить до нагромадження втрат і перетворенню їх в істотні).

Системні вимоги алгоритмів Різні алгоритми стиску можуть вимагати різної кількості ресурсів обчислювальної системи, на яких вони реалізовані: · оперативної пам’яті (під проміжні дані); · постійної пам’яті (під код програми і константи); · процесорного часу. У цілому, ці вимоги залежать від складності і «інтелектуальності» алгоритму. Загальна тенденція така: чим ефективніший і універсальніший алгоритм, тем більші вимоги до обчислювальних ресурсів він пред’являє. Проте, у специфічних випадках прості і компактні алгоритми можуть працювати не гірше складних і універсальних. Системні вимоги визначають їхні споживчі якості: чим менш вимогливий алгоритм, тим на більш простій, а отже, компактній, надійній і дешевій системі він може бути реалізований. Оскільки алгоритми стиску і відновлення працюють у парі, має значення співвідношення системних вимог до них. Нерідко можна, ускладнивши один алгоритм, значно спростити інший. Таким чином, можливі три варіанти: 1) Алгоритм стиску вимагає більших обчислювальних ресурсів, ніж алгоритм відновлення — це найпоширеніше співвідношення, характерне для випадків, коли однократно стиснуті дані (звук, відео) будуть використовуватися багаторазово. 2) Алгоритми стиску і відновлення вимагають приблизно рівних обчислювальних ресурсів — найбільш прийнятний варіант для ліній зв’язку, коли стиск і відновлення відбувається однократно на двох її кінцях

(наприклад, у цифровій телефонії). 3) Алгоритм стиску суттєво менш вимогливий, ніж алгоритм відновлення — така ситуація характерна для випадків, коли процедура стиску реалізується простим, часто портативним обладнанням, для якого обсяг доступних ресурсів досить критичний, наприклад, космічний апарат або велика розподілена мережа датчиків. Це можуть бути також дані, розпакування яких потрібно в дуже малому відсотку випадків, наприклад запис камер відеоспостереження.

2.3. Проблеми роботи з медіафайлами

Медіафайли сьогодні - невід'ємна частина будь-яких веб-застосунків, незалежно від напрямку, для якого був створений застосунок, та від розміру (що залежить від кількості постійних користувачів).

Сьогодні близько 70% всіх компаній інвестують в контент-маркетинг, який включає в себе візуальні маркетинг-стратегії. Основна форма контент-маркетингу - це відео [4]. Тож зараз більша частина бізнесу перейшла в Інформаційні технології і використовує візуальний контент для маркетингу. Це означає, що робота з медіафайлами - дуже важлива складова успішного продукту, оскільки фото і відео присутні майже на всіх інформаційних ресурсах: від ілюстрацій в особистому блозі до відео на YouTube (популярний відеохостинг) [5]. Однак існують проблеми при перегляді медіафайлів користувачами, будь-де - і на веб-сторінках, і у мобільних додатках. 3 Перша проблема полягає у тому, що великі фото і відео файли завантажуються дуже повільно. Наприклад, файл розміром 10 мегабайт буде завантажуватись 8 секунд при швидкості 10 мегабіт в секунду [6]. Це дуже довго, враховуючи, що подібних файлів на веб-сторінці чи у мобільному додатку може бути багато.

Користувачі звикли до швидкого перегляду інформації: згідно зі статистикою, більшість людей не буде чекати завантаження веб-сторінки більше 4 секунд [7]. Якщо ж інформаційний ресурс спеціалізується на показі

медіафайлів, така затримка у їх відображенні просто недопустима. Наприклад, популярні соціальні мережі, такі як Facebook, Instagram, Twitter, практикують техніку “нескінченної прокрутки” на головних екранах за списком записів, створених користувачами (з англійської “feed”). За цією технікою, при прокрутці автоматично формуються користувацькі записи, які будуть показані користувачу під час наступної прокрутки.

Таким чином, користувач може переглядати записи інших користувачів нескінченно довго. Що найважливіше, медіафайли під час такої “нескінченної прокрутки” повинні завантажуватись дуже швидко - інакше користувач буде бачити просто пустий список записів інших користувачів до тих пір, поки фото і відео файли не будуть завантажені [рис. 1.1].

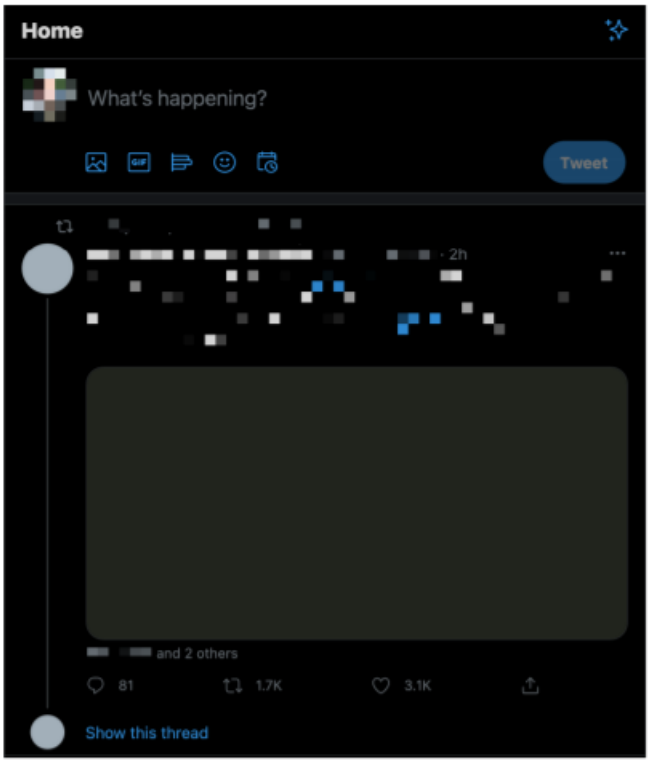


Рисунок 1.1 - Довге завантаження зображень у соціальній мережі Twitter.

Таке довге очікування результує у поганий користувацький досвід (UX), тому що користувачі замість того, щоб переглядати цікавий для них контент, переглядають пустий екран. Як наслідок, збільшується вірогідність того, що користувач не буде користуватись цим сервісом і він стане менш

популярним, що, у свою чергу, зменшить прибуток бізнесу, якому цей сервіс належить. Із відеофайлами така ж проблема - довге їх завантаження результує у поганий UX і зменшення популярності сервісу.

Інша проблема великих медіафайлів стосується мобільних пристроїв та комп'ютерів з невеликим об'ємом оперативної пам'яті. Полягає вона у тому, що при показі великого фото або відео файлу, використовується багато оперативної пам'яті пристрою. Коли застосунок використовує більше пам'яті, ніж операційна система (ОС) може під нього виділити, ОС просто закриває цей застосунок. Це результує у екстрене закриття веб-браузерів і мобільних додатків із запущеним сервісом, що споживає багато оперативної пам'яті. Таким чином, якщо на пристрої вільної оперативної пам'яті 500 мегабайт (наприклад, на мобільних пристроях), то технологія "нескінченного завантаження" спричинить недостачу оперативної пам'яті вже за кілька прокруток: нові великі файли будуть додаватися в оперативну пам'ять, а операційна система (якщо вона очищає невикористовувану оперативну пам'ять) не буде встигати очищати оперативну пам'ять від старих невикористовуваних файлів.

У такому випадку операційна система просто закриє запущений застосунок, причому, якщо це мобільний пристрій, що працює на ОС iOS або Android, користувач навіть не зрозуміє, чому застосунок був закритий. Це також породжує поганий UX. Окрім того, на мобільних девайсах зображення використовуються ще у системних сповіщеннях і у системному плеєрі. Якщо вони будуть довго завантажуватись, користувач не буде бачити зображення, прикріплене до повідомлення або зображення-обкладинку альбому у системному плеєрі, що не настільки критично, як екстрене закриття застосунку, але сильно погіршує користувацький досвід. Інша проблема великих фото і відео файлів у тому, що у оригінальному розмірі вони займають багато постійної пам'яті у системі, на якій розгорнуто застосунок. Якщо кожен користувач буде відвантажувати по 2 файли розміром 10

мегабайт кожен, то тільки 100 користувачів за місяць 6 відвантажать файлів на 8000 мегабайт. Хранiння всіх цих файлів буде досить дорогим для бізнесу.

РОЗДІЛ 3. ОГЛЯД ЗАСОБІВ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЕФЕКТИВНОЇ ОБРОБКИ ФОТО І ВІДЕО ФАЙЛІВ

3.1 Основні підходи до хмаркових технологій

Хмаркова технологія охоплює, окрім самих обчислень, такі області, як зберігання даних, інформаційні послуги, інтеграцію, безпеку, організація певних процесів і управління. Причому один і той же провайдер може пропонувати будь-який перелік послуг. Користувач же може отримувати весь спектр послуг у одного провайдера, а може скористатися і різними провайдерами для певних різновидів сервісів. Наявне різноманіття провайдерів і сервісів істотно ускладнює процес вибору для користувача. До переваг концепції хмаркових сервісів для провайдерів можна віднести:

- Розвиток додатків і програмних засобів проміжного рівня (middleware) привів до можливості використання віртуальних ресурсів.
- Зниження вартості і підвищення масштабованості обчислювальних ресурсів дозволяє створювати потужні віртуальні машини.
- Це звільняє замовників від необхідності купувати і підтримувати апаратні засоби, а можуть користуватися віртуальними машинами, що працюють в «мережевій хмарці».
- Послуги, доступ до яких надається через «мережеву хмарку», мають вели-кий набір конкурентних переваг в порівнянні з іншими типами керованих послуг.
- Cloud Computing створює платформу для стандартних керованих послуг, які можуть пропонуватися на нішевих вертикальних ринках, що включають ринок малих підприємств. Серед переваг Cloud Computing для замовників можна виділити:

- Можливість зниження накладних витрат, пов’язаних з підтримкою апаратних і програмних компонентів (залежить від типа пропонованих послуг).
- Зниження сукупної вартості володіння у випадку оплати за фактом використання дає можливість замовникові починати роботу з системою з невеликих об’ємів і збільшувати об’єм використовуваних ресурсів в міру необхідності. Такий підхід дозволяє запобігти великим капітальним витратам на початковому етапі проекту.
- Швидке і просте придбання нових послуг і прискорення процесів виходу на ринок. Приведений огляд показав усі різноманіття використання хмаркових технологій з точки зору користувача. Але при підході до проектування таких сервісів необхідно враховувати й іншу точку зору – точку зору розробника. Тому потрібно детально проаналізувати архітектурні вирішення хмарних технологій[23].

Швидке збільшення пропускної здатності комп'ютерних мереж дозволило розробляти численні додатки, які передбачають інтенсивну обробку даних. Ці нові додатки можуть виконувати задачі, починаючи від масової передачі даних (SDSS [104] та e-VLBI [87]), до інтерактивних систем високої пропускної здатності (GeoWall [70]). Однак, різні програми ставлять різні вимоги до послуг передачі даних. Наприклад, додаток GeoWall може віддати перевагу плавній зміні швидкості передачі даних, тоді як для додатку SDSS бажано, щоб дані передавалися з максимально можливою швидкістю в приватних мережах. Проте, нинішня система Інтернет призначена для забезпечення підтримки цілої множини різного типу додатків. Ця філософія дизайну Інтернету має великий вплив на розвиток транспортних протоколів. Більшість трафіку в Інтернеті формується за рахунок потоків TCP, але існують додатки для яких протокол TCP не забезпечує достатнього рівня ефективності. У контексті високопродуктивних обчислень, TCP добре

відомий своєю низькою ефективністю та справедливим поділом ресурсів в мережах з високою затримкою пропускної здатності [44]. За останні декілька років дослідники комп'ютерних мереж запропонували багато нових алгоритмів керування перевантаженням для загальної та специфічної 86 передачі даних [46, 73, 75]. Тим не менше, більшість з цих запропонованих алгоритмів завершуються моделюванням та обмежуються умовами лабораторних експериментів; мало з них пройшли практичне тестування і випробування в реальних високопродуктивних мережах. З іншого боку, модифікації мережевого стеку ядра протоколу (наприклад, нові варіанти TCP) зазвичай вимагають кількох років для стандартизації, впровадження та широкого розгортання. Справді, з часів появи протоколу TCP, близько трьох десятиліть тому, тільки його чотири версії були широко розгорнуті, а саме Tahoe, Reno, NewReno, і SACK [44, 73, 108]. Хоча на сьогоднішній день все більше мереж отримують швидкість передачі даних 1 Гбіт/с і вище, але як і раніше актуальною проблемою для гріддодатків залишається використання широкої смуги пропускної здатності, через обмеження існуючих транспортних протоколів мережі [108].

Обмеження впроваджених мережевих транспортних протоколів є однією з головних причин, через яку так важко масштабувати додатки з інтенсивним використанням мережевих з'єднань від місцевих кластерів до глобальних мереж [36, 41, 49, 108]. Протокол управління передачею (TCP) успішно використовується протягом десятиліть, як основний протокол транспортного рівня стеку мережних протоколів. Проте останнім часом було показано, що TCP має деякі втрати продуктивності при його використанні для високошвидкісних мереж Wide Area, особливо для географічно віддалених мереж. Алгоритм управління перевантаженням AIMD, який використовується протоколом TCP, є досить «бідним» у розкритті доступної смуги пропускної здатності і у випадку великої втрати пакетів у високопродуктивних мережах з досить великими часами затримки [44]. Дослідники комп'ютерних мереж працюють над новими транспортними

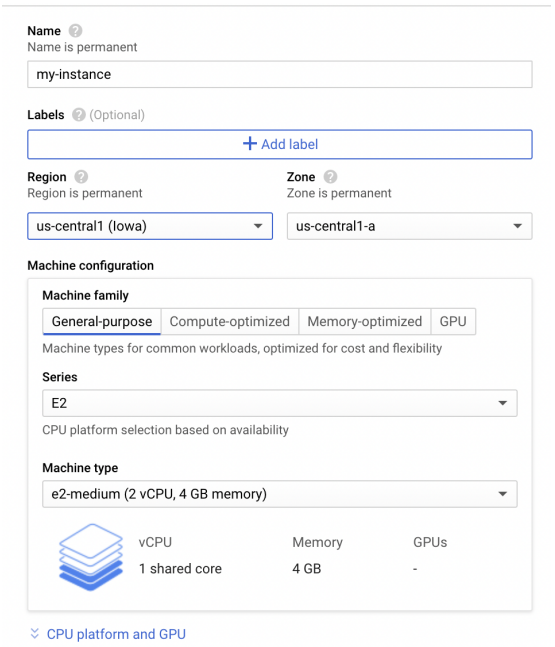
протоколами і алгоритмами контролю насичення для підтримки високошвидкісних мереж наступного покоління. Багато робіт, у тому числі варіантів TCP (FAST [43], BiC [46], Scalable [73], і HighSpeed [115]) і XCP [71] показали більш високу продуктивність при їх моделюванні. Однак практичне використання в реальних додатках цих протоколів все ще дуже обмежена через 87 труднощі їх реалізації, встановлення та обмеження технічного рівня. Користувачі мережі, яким потрібно передавати великі масиви даних зазвичай звертаються до рішень рівня додатків, серед яких дуже популярні протоколи на основі UDP, наприклад SABUL [57], UDT [108], Tsunami [77], RBUDP [86], FOBS [38] і GTP [114]. Протоколи на основі UDP забезпечують набагато кращу переносимість і прості в при їх встановленні. Однак, незважаючи на простоту впровадження протоколів на рівні користувача, досить важко налаштувати їх в ядрі, щоб зробити їх максимально ефективними. Оскільки реалізації рівня користувач не можуть змінити код ядра, можуть бути додаткові перемикання контексту і копіюванням ділянок пам'яті між рівнем користувача та рівнем ядра. На високих швидкостях передачі даних, ці операції дуже чутливі до завантаження процесора і продуктивності протоколу. Враховуючи перераховані утруднення та недоліки існуючого стану передачі даних великого обсягу через високопродуктивні регіонально-розподілені мережі видається доцільним з практичної точки зору розробка протоколу сеансового рівня для таких мереж. Новий протокол сеансового рівня повинен бути сумісним з стандартними протоколами стеку протоколів OSI – TCP та UDP. Крім того, він повинен бути адаптований для сучасних мереж, тобто ефективно використовувати пропускну здатність мережі, не залежно від її характеристик. При використанні такого протоколу не повинні збільшуватися затримки при переході передачі між різними типами мереж. Тим самим протокол повинен забезпечити ефективне використання ресурсів крайніх вузлів мережі. У питанні захищеності --- він повинен підтримувати шифрування даних, тобто він повинен дозволяти підключення протоколів

шифрування [20]. Доцільно провести оптимізацію ефективності реалізації протоколу на базі UDP і показати, що за цими ідеями можна реалізувати ефективні та практичні додатки на базі протоколів UDP. Наприклад, використання середовища протоколу UDT (базований на UDP) [108], може легко підтримувати різні алгоритми 88 управління перевантаженням, наприклад, високошвидкісні TCP [36, 46, 73, 115] або вибуховий RBUDP [115]. Перевагами даного середовища протоколу є: • нові протоколи на базі UDP можуть бути швидко прототиповані і випробувані; • різноманітні алгоритми управління перевантаженням можна легко порівняти експериментально; • конкретні прикладні протоколи, що використовують UDP, можуть бути розроблені відносно легко. Також, з використанням даного середовища можуть розроблятися конкретні протоколи для розподілених даних або потокових медіа-додатків. Основним недоліком даного середовища протоколу є додаткове навантаження самого середовища, які накладають додаткові витрати протоколу прикладного рівня у порівнянні з рівнем протоколу ядра. Для досягнення високої продуктивності, керуються двома принципами: 1. Не повинно бути піків завантаження процесора, особливо на стороні отримання даних. Сплески завантаження CPU можуть призвести до невчасної обробки прийнятих пакетів, що у свою чергу призведе до втрати даних передачі. Це може призвести до серйозних проблем, які спостерігаються в TCP з AIMD, у якому при втраті пакету, зазвичай, витрачається багато часу для відновлення передачі даних при з’єднаннях на великих відстанях. 2. Загальне завантаження процесора повинно бути якомога меншим. Проста реалізація додатку може запобігти надмірному використанню центрального процесора. Крім того, інші операції з обробки даних, які також потребують процесорного часу, часто виконуються паралельно з передачею даних. Для перевірки завантаженості центрального процесора (CPU), системних затримок та впливу розмірів пакетів та розмірів буферів на швидкість передачі даних для протоколу UDP було проведено декілька практичних експериментів.

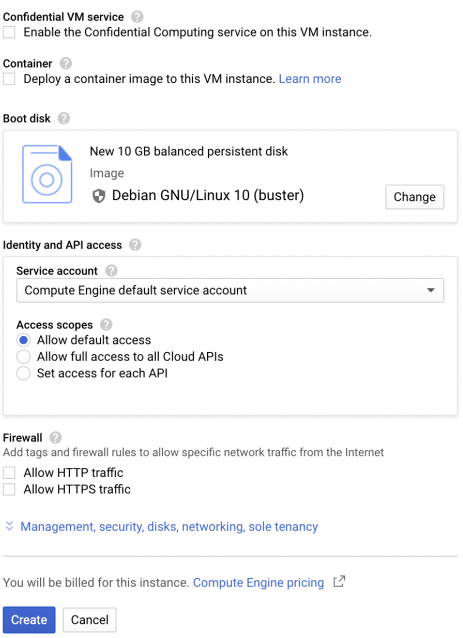
3.2. Розгортання та інтеграція розробленої хмарної розподіленої високонавантаженої високодоступної системи обробки і оптимізації зображень і відео

Наступним кроком після розробки системи є її розгортання та інтеграція у цільове середовище. Це потрібно для того, щоб застосувати розроблену систему та зробити її доступною для постійного та стабільного використання.

Для розгортання створеної системи було вирішено використовувати хмарну платформу Google Cloud Platform та споріднені сервіси Google Compute Engine (GCE), Google Kubernetes Engine (GKE) та Google Cloud Storage (GCS). Google Cloud Platform це IaaS (Infrastructure as a Service, Інфраструктура як Сервіс) платформа, що надає користувачам можливість розгортання широкого асортименту інфраструктурних рішень у одному або декількох датацентрах на вибір. На сьогоднішній день GCP налічує 76 датацентрів, 25 регіонів, 144 мережевих вихідних точок, та є доступною у 200 країнах світу. Google Compute Engine являється підсервісом Google Cloud Platform, що надає доступ до розгортання віртуальних машин та комп’ютерних мереж. GCP дозволяє створювати віртуальні машини з різними ресурсами, конфігураціями, та у різніх регіонах (рисунки 3.1, 3.2).



(Рисунок 3.1)

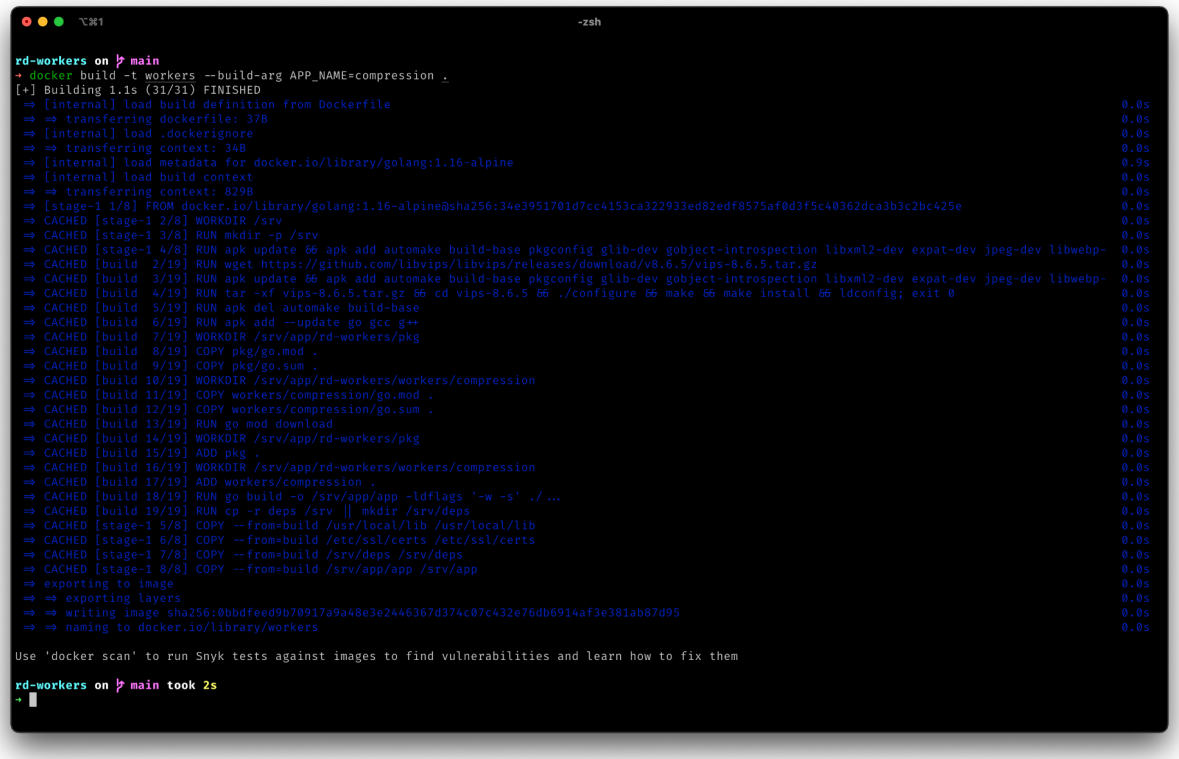


(Рисунок 3.2)

Як платформу для розміщення системи було обрано Kubernetes та Google Kubernetes Engine. Kubernetes - це портативна, розширювана платформа з відкритим кодом для управління робочими навантаженнями та послугами в контейнерах, що сприяє як декларативній конфігурації, так і простій автоматизації.

Для розгортання застосунку у Kubernetes потрібно цей застосунок контейнеризувати. Контейнери схожі на віртуальні машини, але вони мають розслаблені властивості ізоляції для спільного використання операційної системи (ОС) серед програм, тому вважаються легкими. Подібно до віртуальної машини, контейнер має власну файлову систему, частку центрального процесора, пам’ять, простір процесів тощо. Оскільки контейнери відокремлені від базової інфраструктури, вони легко переносяться між хмарами та дистрибутивами ОС. Для контейнеризації застосунку було використано Docker - інструмент для контейнеризації застосунків. Для конфігурації Docker у кореневій директорії застосунку було створено файл “Dockerfile”, у якому був прописаний скрипт, який задає властивості контейнеризації застосунку (див. Додаток Б). Збірка контейнеру

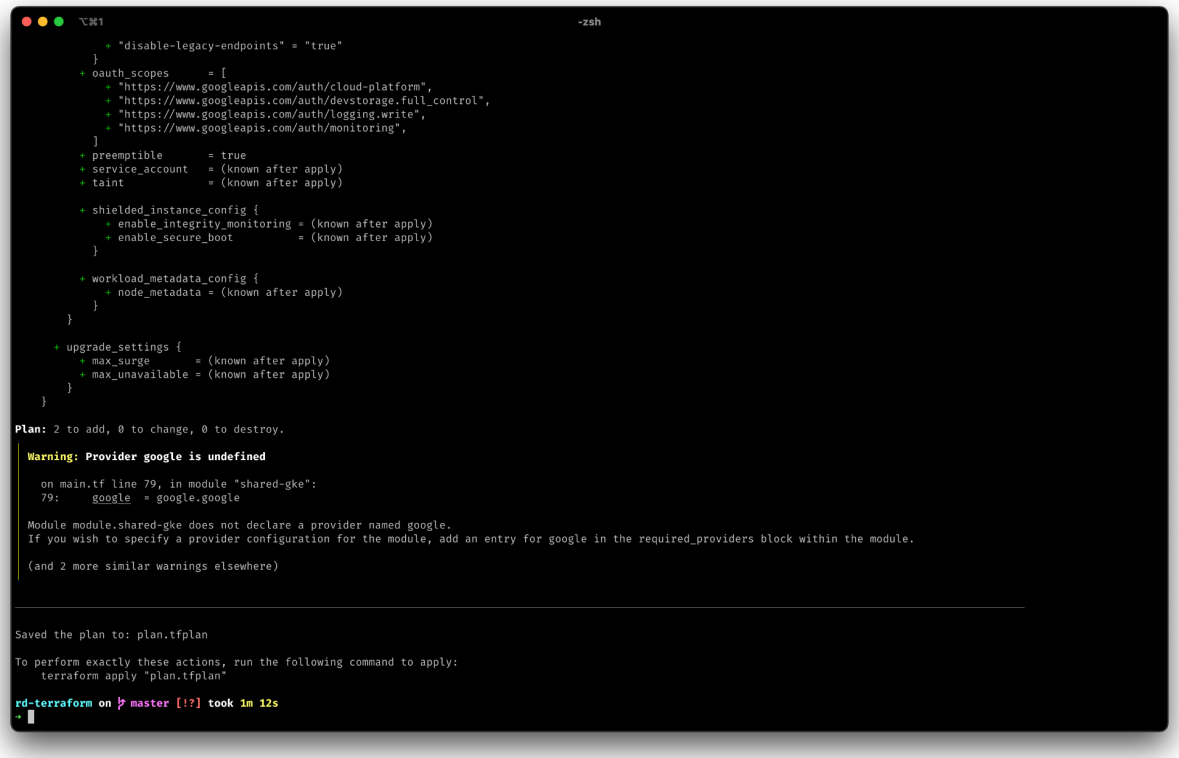
була протестована за допомогою команди “docker build -t workers --build-arg APP_NAME=compression .” (рисунок 3.3).



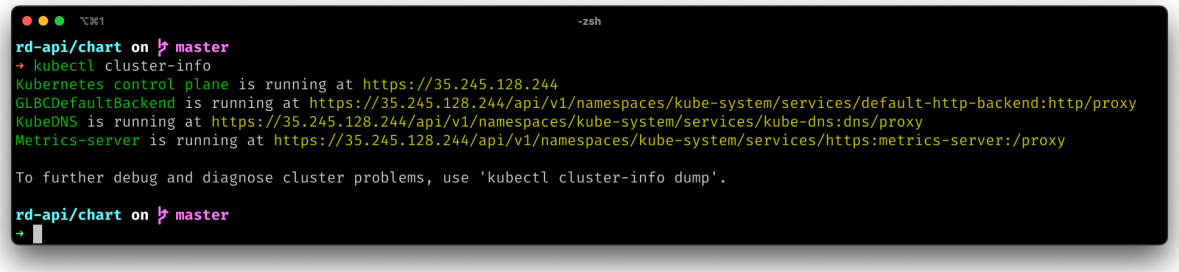
(рисунок 3.3)

Для створення та конфігурації кластера Kubernetes було вирішено використати інструмент Terraform від компанії Hashicorp. Terraform - це інструмент для безпечного та ефективного створення, зміни та редагування інфраструктури. Terraform може керувати існуючими та популярними постачальниками послуг, а також власними власними рішеннями. Файли конфігурації Terraform описують компоненти, необхідні для запуску однієї програми або всього датацентру. Terraform формує план виконання, що описує, що він буде робити, щоб досягти бажаного стану, а потім виконує цей план для побудови описаної інфраструктури. У міру зміни конфігурації Terraform може визначити, що змінилося, і створити додаткові плани виконання, які можна застосувати. Інфраструктура, якою може керувати Terraform, включає компоненти низького рівня, такі як віртуальні машини, хмарні сховища та мережі, а також компоненти високого рівня, такі як записи

DNS, функції SaaS тощо. Конфігураційні файли Terraform для створення цільового кластеру Kubernetes приведені у Додатку Б. Були використані такі об’єкти Terraform, як “google_container_cluster” (розгортує та конфігурує кластер Kubernetes в Google Kubernetes Engine) та “google_container_node_pool” (розгортує та конфігурує пул віртуальних машин в Google Compute Engine). Цільова інфраструктура була запланована та розгорнута за допомогою команд “terraform plan -out plan.tfplan” (рисунок 3.4) та “terraform apply plan.tfplan”. Корректне фунціонування новоствореного кластеру Kubernetes було перевірено за допомогою команди “kubectl cluster-info” (рисунок 3.5).

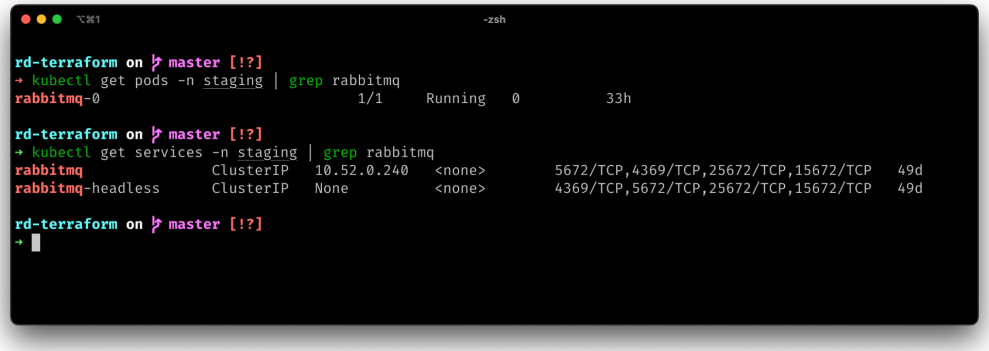


(рисунок 3.4)



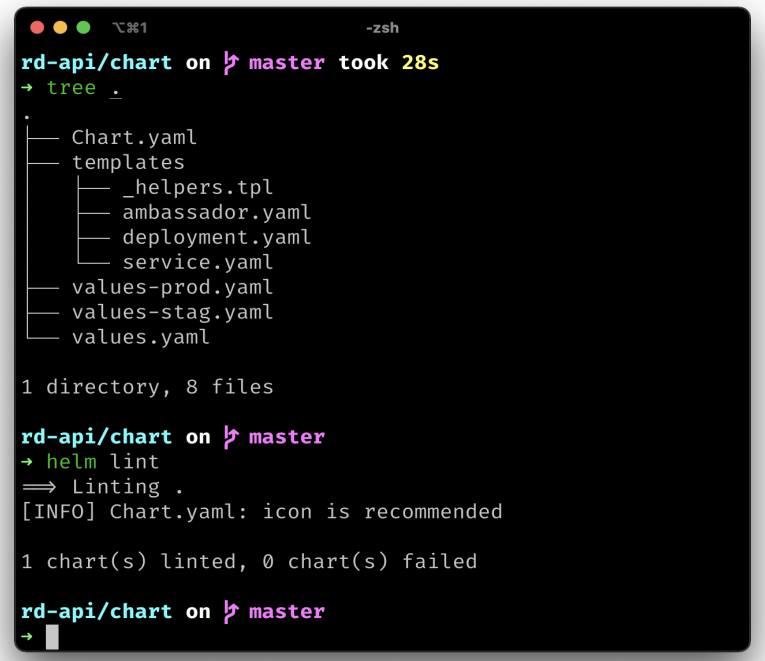
(рисунок 3.5)

Додатково до кластеру Kubernetes було розгорнуто чергу RabbitMQ. RabbitMQ - це програмне забезпечення для передачі повідомлень з відкритим кодом, яке оригінально реалізовувало протокол AMQP (Advanced Messaging Queue Protocol, розширений протокол черги повідомлень), але з тих пір було розширено для підтримки протоколів Streaming Text Oriented Messaging Protocol, MQ Telemetry Transport та інших. Функціонуючий сервер RabbitMQ є обовязковим для розгортання застосунку, так як він виступає інтерфесом для коммунцікації з системою оптимізації зображень і відео. Для розгортання RabbitMQ було використано ресурси Terraform “helm_release” та “kubectl_manifest”. Для перевірки коректного розгортання серверу RabbitMQ було використано команди “kubectl -n staging get services” та “kubectl -n staging get services” (рисунок 3.6). Після розгортання серверу RabbitMQ, стало можливим підключення до нього за адресою “rabbitmq.staging.svc.cluster.local”. Варто зазначити, що ця адреса доступна для використання тільки з внутрішньої мережі кластера.



(рисунок 3.6)

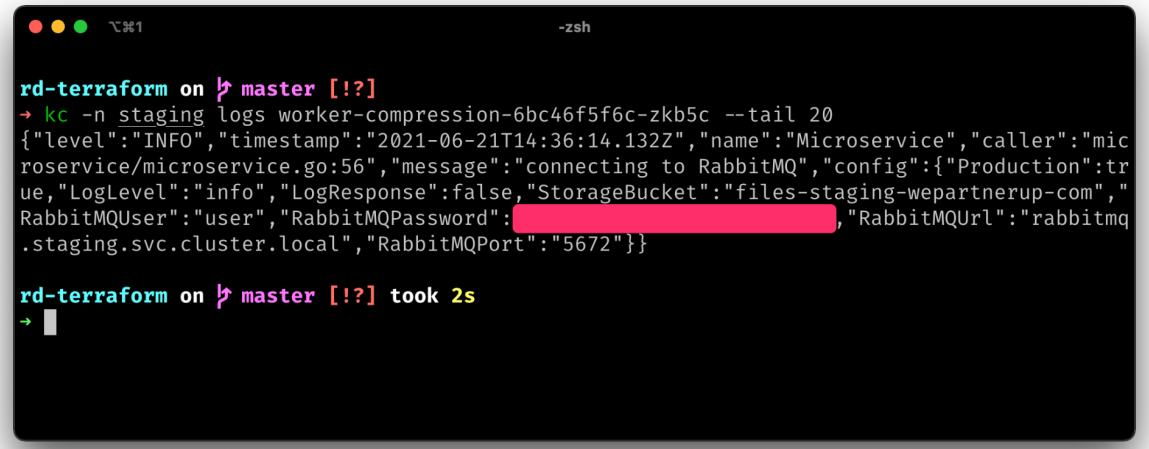
Для розгортання застосунку у кластер Kubernetes було вирішено використовувати Helm, менеджер застосунків для Kubernetes. Helm дозволяє описати структуру програми за допомогою зручних схем управління та керувати нею за допомогою простих команд. Helm використовує формат упаковки, який називається чарт (Chart, діаграма). Chart - це сукупність файлів, що описують відповідний набір ресурсів Kubernetes. Один чарт можна використовувати для розгортання чогось простого, як сервер memcached, або чогось складного, як повний стек веб-додатків із серверами HTTP, базами даних, кешами тощо. Для розгортання застосунку за допомогою Helm було створено Helm Chart (див. Додаток Б). Перевірка роботоспроміжності та валідація створеного чарту була зроблена за допомогою команди “helm lint” (рисунок 3.7). Розгортання застосунку було виконано за допомогою команди “helm install”. Успішна встановка на корректне функціонування застосунку було перевірено за допомогою команд “kubectl -n staging get pods”, “kubectl -n staging get services” та “kubectl -n staging get pods” (рисунки 3.8, 3.9).


⌥⌘1
-zsh
rd-api/chart on master took 28s
→ tree .
.
├── Chart.yaml
├── templates
│   ├── _helpers.tpl
│   ├── ambassador.yaml
│   ├── deployment.yaml
│   └── service.yaml
├── values-prod.yaml
├── values-stag.yaml
└── values.yaml
1 directory, 8 files
rd-api/chart on master
→ helm lint
==> Linting .
[INFO] Chart.yaml: icon is recommended
1 chart(s) linted, 0 chart(s) failed
rd-api/chart on master
→

(рисунок 3.7)

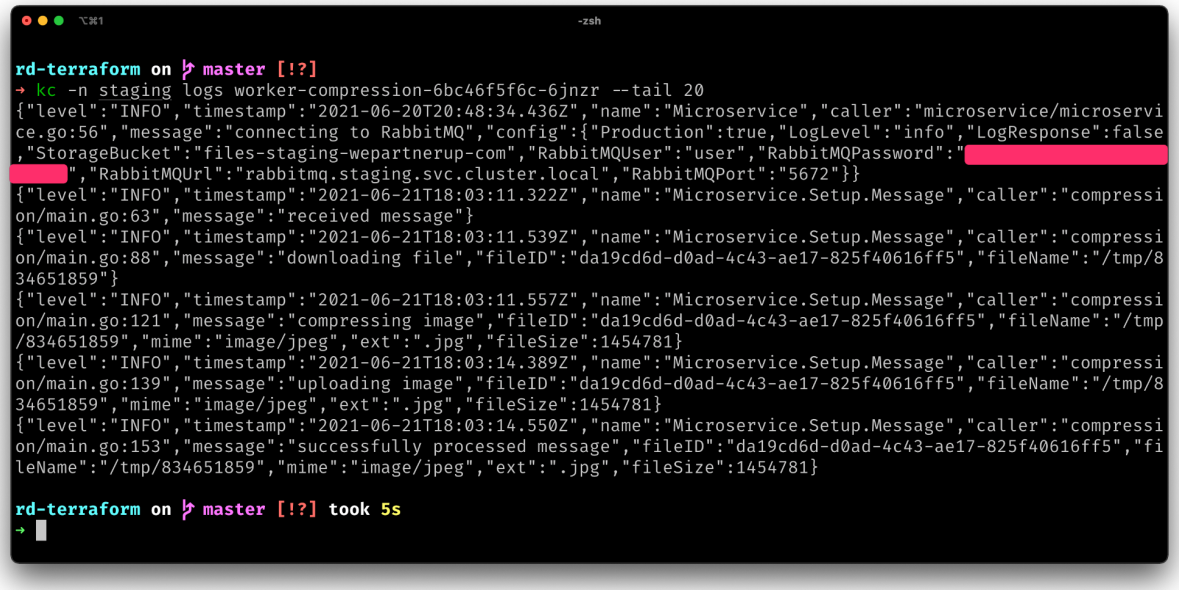

⌥⌘1
-zsh
rd-terraform on master [!?]
→ kc get pods -n staging | grep worker-compression
worker-compression-5796c554bf-gw6vc 1/1 Running 4 126m
worker-compression-5796c554bf-qlcgc 0/1 CrashLoopBackOff 13 126m
rd-terraform on master [!?]
→ kc get services -n staging | grep worker-compression
worker-compression ClusterIP 10.52.0.84 <none> 80/TCP 47d
rd-terraform on master [!?]
→

(рисунок 3.8)

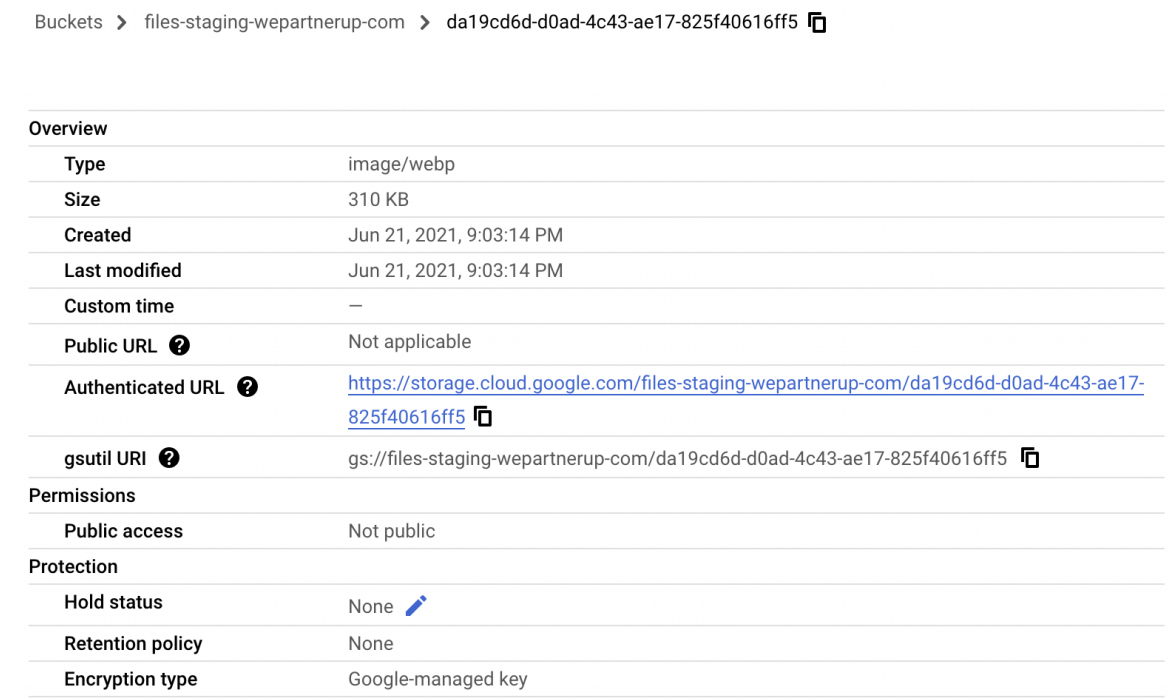


(рисунок 3.9)

Система була протестована за допомогою надсилання декількох зображень у чергу RabbitMQ з іншої частини системи, перевіркою логів застосунку та знайдення оптимізованих зображень на цільовому хмарному сховищі. З логів можна зрозуміти, що до застосунку було надіслано зображення розміром в 1454781 байтів, або1454 кілобайтів (рисунок 3.10). Знайшовши це зображення у сховищі можна побачити, що його розмір став 310 кілобайтів, тобто був зменшений на 1144 кілобайтів, або 78.6 відсотків (рисунок 3.11). При цьому, якість зображення зосталась на високому рівні (рисунок 3.12).



(рисунок 3.10)



(рисунок 3.11)

(рисунок 3.12)

ВИСНОВКИ

Сучасні веб і мобільні застосунки зазвичай складаються з двох частин: клієнтська частина і серверна. У якості клієнтської частини виступає веб або мобільний додаток, доступ до якого має користувач. Якщо це веб-застосунок, у якості клієнта маємо веб-сайт, яким може скористатись користувач (перейти на необхідний URL [10]). Якщо це мобільний застосунок, у якості клієнта виступає додаток, який користувачу необхідно заздалегідь встановити на свій пристрій або з магазину додатків (App Store на iOS [11] і Play Market [12] на Android). Розподілення логіки усієї системи на клієнтську та серверну частини допомагає:

- розділити логіку запитів у базу даних та у інші необхідні сервіси і логіку відображення користувацього інтерфейсу (з англійської user interface або UI [13]);
- зробити логику і відображення незалежними одне від одного, що дозволяє змінювати, наприклад, технології відображення (клієнт), не змінюючи логику (сервер);
- інкапсулювати логіку - користувач не зможе взаємодіяти із серверною частиною напряму - лише через клієнтську частину, у спосіб, визначений розробниками цієї клієнтської частини. У клієнт-серверній системі користувач має доступ лише до клієнтської частини, з якою і взаємодіє за допомогою користувацького інтерфейсу.

А вже клієнтська частина може “спілкуватись” із серверною за допомогою API (занглійської Application Programming Interface) [14] - інтерфейсу, визначеному розробниками серверної частини системи. Виконувати обробку і оптимізацію зображень і відео можна виконувати на стороні клієнта (веб або мобільний додаток) або на стороні сервера. Спочатку розглянемо обробку на стороні клієнта. У першу чергу слід зазначити, що при обробці на стороні клієнта реалізація обробки зображень і відео буде різнитись в залежності від технологій, використовуваних для розробки клієнтської частини. Наприклад, якщо клієнтська частина системи працює на

операційній системі iOS і написана на мові програмування Swift [15], реалізація буде одною, а якщо вона працює на ОС Android і написана на мові Java [16], то іншою. Ситуація ускладнюється тим, що технології для розробки під різні операційні системи не обмежуються двома мовами програмування - зараз існує багато фреймворків для розробки під обидві операційні системи.

Практично будь-який web-додаток працює в унікальному середовищі функціонування. Унікальність визначається поєднанням широкого набору параметрів (конфігурація обладнання, програмне забезпечення, архітектура web-додатки, кількість і склад цільової аудиторії, режим роботи тощо). Це призводить до розробки програмних засобів тестування саме конкретної розглянутої середовища, в якій функціонує web-додаток.

Була розроблена система обробки і оптимізації зображень і відео. Реалізована ця система була за допомогою мови програмування Go. Оптимізація зображень відбувається за допомогою утиліти bimg, що дозволяє конвертувати зображення у різні формати і стискати зображення. Оптимізація відео відбувається за допомогою утиліти командного рядку FFmpeg, що дозволяє конвертувати відеофайли у різні формати а також стискати їх.

Розроблена система була розгорнута у хмарній платформі Google Cloud Platform за допомогою Google Kubernetes Engine - це портативна, розширювана платформа з відкритим кодом для управління робочими навантаженнями та послугами в контейнерах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Топорков В.В. Моделі розподілених обчислень. — М.: ФІЗМАТЛІТ, 2004. - 320 с. - ISBN 5-9221-0495-0.

2. Петренко А.И., Свистунов С.Я., Киселев Г.Д. Практикум по грідтехнологиях. / Київ: НТУУ "КПІ", 2011. - 448 с.

3. Хьюз К., Хьюз Т. Паралельне і розподілене програмування з використанням C++. Пер. з англ. – М.: Видавничий будинок "Вильямс", 2004. – 672 с.: ил.

4. Эндрюс Г. Р. Основи многопоточного паралельного і розподіленого програмування. Пер. з англ. – М.: Видавничий будинок "Вильямс", 2003. – 512 с.: ил.

5. Розподілені системи. Принципи та парадигми / Э. Таненбаум, М. ван Стеен. – СПб.: Питер, 2003. – 877 с. 10

6. Шпаковский Г. И., Стецюренко В. И., Верхотуров А. Е., Серикова Н. В. Застосування технології MPI в Грід (лекції) // Минск.: БГУ. 2008. – 137 с.

7. Косяков М.С. Введення в розподілені обчислення. – СПб: НИУ ИТМО, 2014. – 455 с. 6.2. Допоміжна література

9. Конспект лекцій з дисципліни «Розподілені комп'ютерні системи і мережі».

10. Хорстманн, К. Java 2. Бібліотека професіонала. Т. 2: Тонкощі програмування / К. Хорстманн, Г. Корнелл. – М. : Видавничий будинок «Вильямс», 2010.

11. Java : основи Web-служб / Г. Беккет [и др.] ; пер. з англ. – М. : КУДИЦОБРАЗ, 2004.

12. Шумаков, П. ADO.NET і створення додатків в середовищі Microsoft Visual Studio.Net / П. Шумаков. – М. : Диалог-МиФи, 2003.

13. Лабор, В. В. Си Шарп – створення додатків для Windows / В. В. Лабор. – Минск : Харвест, 2003.

14. Рихтер, Дж. Програмування на платформі .NET Framework / Дж. Рихтер. – М. : Русская редакция, 2003.

15. Мак-Дональд, М. Microsoft ASP.NET 3.5 з прикладами на C# / М. МакДональд, М. Шпушта. – М. : Видавничий будинок «Вильямс», 2010.

16. Рейли, Д. Створення додатків Microsoft ASP.NET / Д. Рейли. – М. : Українська редакція, 2002. 6.3. Web-ресурси

17. Архітектура розподілених обчислень: від хмари до туману і росі / Прес-центр компанії Cloud4Y / Хабр

18. Бізнес-модель розподілених обчислень і p2p / Хабр

19. Розподілені обчислення на платформі .NET / Хабр

20. SimGrid Home

21. Джерело: https://nmetau.edu.ua/file/ikt_tutor.pdf

22. Джерело: https://promodo.ua/ua/blog/metody-optimizatsii-izobrazhenij.html

23. Джерело: https://lpnu.ua/sites/default/files/2020/dissertation/1551/disstrubyckyy2.pdf

ДОДАТОК А

ВІДОМОСТІ МАТЕРІАЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

		Позначення	Найменування	Кільк. аркушів	Примітки
1					
2			Документація		
3					
4	А4	**ІТКІ.КР 19.03.ДА.ПЗ**	Пояснювальна записка		
5					
6			Презентація		
7					
8					

Зм	Лист	№ докум	Підпис	Дата	**ІТКІ.КР 21.01.ДА.ПЗ**

Розроб.		Єфремов В.О.			**Матеріали кваліфікаційної роботи**	Літ.	Лист	Аркушів
						Н	1	1
Рецензент		Коротенко Л.М.				**НТУ «ДП», 126-17-1**		
Керівник		Гнатушенко В.В.						
Н.контр.								
Зав.каф.		Гнатушенко В.В.						

ДОДАТОК Б

microservices/compression/main.go

```go
package main

import (
	"context"
	"io"
	"io/ioutil"
	"strings"

	"github.com/gabriel-vasile/mimetype"
	"github.com/softcery/rd-workers/pkg/microservice"
)

// CompressionMicroservice is responsible for
// compressing photos and videos uploaded to ryddm by
users.
type CompressionMicroservice struct {
	microservice.Microservice
}

// Setup connects to rabbitmq queue and consumes file ids
from it.
func (ms *CompressionMicroservice) Setup() {
	ms.Initialize(&microservice.InitializeOptions{
		Storage:  true,
		RabbitMQ: true,
	})
	logger := ms.Logger.Named("Setup")

	// declare rabbitmq for accepting image ids
	logger.Debugw("declaring queue")
	queue, err := ms.RabbitMQChannel.QueueDeclare(
		"files", // name
		true,    // durable
		false,   // delete when unused
		false,   // exclusive
```

```
            false,   // no-wait
            nil,     // arguments
        )
        if err != nil {
            logger.Fatalw("failed to declare queue", "err",
err)
        }

        // consume messages from queue
        logger.Debugw("cunsuming messages")
        messages, err := ms.RabbitMQChannel.Consume(
            queue.Name, // queue
            "",         // consumer
            true,       // auto-ack
            false,      // exclusive
            false,      // no-local
            false,      // no-wait
            nil,        // args
        )
        if err != nil {
            logger.Fatalw("failed to start consuming
messages", "err", err)
        }

        // process each message
        // 1. download file from gcs to disk
        // 2. get mime type and size
        // 3. compress photo or video
        // 4. upload back to gcs
        for message := range messages {
            logger = logger.Named("Message")
            logger.Infow("received message")

            // get file id
            fileID := string(message.Body)
            logger = logger.With("fileID", fileID)
```

```
          // create file in temp directory
          file, err := ioutil.TempFile("", "*")
          if err != nil {
               logger.Fatalw("failed to create temp file",
"err", err)
          }
          logger = logger.With("fileName", file.Name())
          defer file.Close()

          // get file from gcs
          obj :=
ms.Storage.Bucket(ms.Config.StorageBucket).Object(fileID)

          // get file reader from gcs
          fileReader, err :=
obj.NewReader(context.Background())
          if err != nil {
               logger.Fatalw("failed to get file reader",
"err", err)
          }
          defer fileReader.Close()

          // download file from gcs to local storage
          logger.Infow("downloading file")
          _, err = io.Copy(file, fileReader)
          if err != nil {
               logger.Fatalw("failed to download file",
"err", err)
          }

          // get file mime type
          mime, err := mimetype.DetectFile(file.Name())
          if err != nil {
               logger.Fatalw("failed to detect file mime
type", "err", err)
          }
          logger = logger.With("mime", mime, "ext",
```

```
mime.Extension())

          // get file size
          stat, err := file.Stat()
          if err != nil {
               logger.Fatalw("failed to stat file", "err",
err)
          }
          fileSize := stat.Size()
          logger = logger.With("fileSize", fileSize)

          // compressed file
          var output *CompressOutput

          // compress image if larger than 128 kb
          if strings.Contains(mime.String(), "image") {
               // don't compress if less than 128 KB
               if stat.Size() < 128<<10*1 {
                    logger.Infow("file to small, skipping")
                    continue
               }

               // compress image
               logger.Infow("compressing image")
               output, err =
ms.CompressImage(&CompressOptions{File: file})
               if err != nil {
                    logger.Fatalw("failed to compress
image", "err", err)
               }
          } else if strings.Contains(mime.String(), "video")
{
               // compress video
               logger.Infow("compressing video")
               output, err =
ms.CompressVideo(&CompressOptions{File: file})
               if err != nil {
```

```
                    logger.Fatalw("failed to compress
video", "err", err)
                }
            } else {
                logger.Infow("unknown file type, skipping")
                continue
            }

            // upload compressed file
            logger.Infow("uploading image")
            writer := obj.NewWriter(context.Background())
            _, err = io.Copy(writer, output.Data)
            if err != nil {
                logger.Fatalw("failed to write file to cloud
storage", "err", err)
            }

            // finish writing
            err = writer.Close()
            if err != nil {
                logger.Fatalw("failed to write file to cloud
storage", "err", err)
            }

            // yay
            logger.Infow("successfully processed message")
        }
}

func main() {
        ms := CompressionMicroservice{}
        ms.Setup()
}
```

microservices/compression/compression.go

```
package main

import (
	"bytes"
	"fmt"
	"os"
	"os/exec"
	"runtime"

	"gopkg.in/h2non/bimg.v1"
)

// CompressImageOptions is used to parameterize
CompressImage.
type CompressOptions struct {
	File *os.File
}

// CompressImageOptions represents output of CompressImage
and CompressVideo functions.
type CompressOutput struct {
	Data *bytes.Buffer
	Type string
}

// CompressImage is used to compress image files.
func (ms *CompressionMicroservice) CompressImage(options
*CompressOptions) (*CompressOutput, error) {
	buffer, err := bimg.Read(options.File.Name())
	if err != nil {
		return nil, fmt.Errorf("failed to read file: %s",
err)
	}

	// create new image in Webp format
	newImage, err :=
bimg.NewImage(buffer).Convert(bimg.WEBP)
```

```
	if err != nil {
		return nil, fmt.Errorf("failed to convert image to
webp: %s", err)
	}

	// decrease quality
	newImage, err =
bimg.NewImage(newImage).Process(bimg.Options{
		Quality:       15,
		StripMetadata: true,
	})
	if err != nil {
		return nil, fmt.Errorf("failed to set image
quality: %s", err)
	}

	return &CompressOutput{
		Data: bytes.NewBuffer(newImage),
		Type: "image/webp",
	}, nil
}

// CompressVideo is used to compress video files.
func (ms *CompressionMicroservice) CompressVideo(options
*CompressOptions) (*CompressOutput, error) {
	logger := ms.Logger.Named("CompressVideo")

	// determine binary path
	var binary string
	if runtime.GOOS == "linux" {
		binary = "./deps/ffmpeg"
	} else if runtime.GOOS == "darwin" {
		binary = "./deps/ffmpeg-darwin"
	} else {
		return nil, fmt.Errorf("unsupported runtime: %s",
runtime.GOOS)
	}
```

```
    // create command
    logger.Info("creating command")
    var cmd = exec.Command(
        binary,
        // disable verbose messages
        "-hide_banner",
        "-nostats",
        "-loglevel", "error",
        // input jpg from file
        "-y",
        // "-loop", "1",
        // "-r", "1",
        // input video from stdin
        "-i", options.File.Name(),
        // output video
        "-shortest",
        // options
        "-c:v", "libx264", // video codec
        "-pix_fmt", "yuv420p", // most supported pixel
format
        "-vf", `scale='min(1280,iw)':'min(720,ih)'`, //
reduce resolution to 720p
        "-crf", "30", // qualioty, bigger = worse
        "-preset", "fast", // encoding preset
        "-movflags", "frag_keyframe+faststart", // args,
google them
        "-level", "3", //
http://blog.mediacoderhq.com/h264-profiles-and-levels/
        // output mp4 to stdout
        "-f", "mp4",
        "pipe:",
    )

    // alternative: webm
    // // options
    // "-c:v", "libvpx-vp9", // video codec
```

```
	// "-c:a", "libopus", // audio codec
	// "-crf", "30", // quality, bigger = lower
	// "-preset", "ultrafast",
	// "-movflags", "+faststart", // starts faster
	// // output mp4 to stdout
	// "-f", "webm",

	// get standard outputs
	var stdout bytes.Buffer
	cmd.Stdout = &stdout
	var stderr bytes.Buffer
	cmd.Stderr = &stderr

	// run command
	logger.Info("running command")
	err := cmd.Run()
	if err != nil {
		return nil, fmt.Errorf("failed to run command: %s, %s", err, stderr.String())
	}

	// check stderr
	if stderr.Len() > 0 {
		return nil, fmt.Errorf("received stderr: %s", stderr.String())
	}

	return &CompressOutput{
		Data: &stdout,
		Type: "video/mp4",
	}, nil
}
```

pkg/microservice/main.go

```
package microservice
```

```
import (
	"context"

	"cloud.google.com/go/storage"
	"github.com/streadway/amqp"
	"go.uber.org/zap"

	"github.com/softcery/rd-workers/pkg/config"
	"github.com/softcery/rd-workers/pkg/logger"
)

// Microservice provides base struct for all microservices.
// Microservice provides access to shared resources.
type Microservice struct {
	Config *config.Config
	Logger *zap.SugaredLogger

	// optional
	Storage         *storage.Client
	RabbitMQChannel *amqp.Channel
}

// InitializeOptions is used to parametrize Initialize
// and allows to specify services that need to be enabled.
type InitializeOptions struct {
	Storage  bool
	RabbitMQ bool
}

// Initialize is used to read config files
// and establish connections to databases.
func (ms *Microservice) Initialize(options
*InitializeOptions) {
	var err error

	// read config
```

```
	ms.Config = config.New()

	// create logger
	ms.Logger = logger.
		New(ms.Config.LogLevel, ms.Config.Production).
		Named("Microservice")

	// configure storage
	if options.Storage {
		ms.Storage, err =
storage.NewClient(context.Background())
		if err != nil {
			ms.Logger.Fatalw("failed to connect to
storage", "config", ms.Config, "err", err)
		}
		ms.Logger.Debugw("connected to storage")
	}

	// create RabbitMQ connection and open channel
	if options.RabbitMQ {
		ms.Logger.Infow("connecting to RabbitMQ",
"config", ms.Config)
		ms.RabbitMQChannel, err = ms.NewRabbitMQ()
		if err != nil {
			ms.Logger.Fatalw("failed to connect to
RabbitMQ", "config", ms.Config, "err", err)
		}
		ms.Logger.Debugw("connected to RabbitMQ channel")
	}
}
```

pkg/microservice/rabbitmq.go

```
package microservice

import (
	"fmt"
```

```
    "github.com/streadway/amqp"
)

// NewRabbitMQ is used to create a new NewRabbitMQ
connection and open channel.
func (ms *Microservice) NewRabbitMQ() (*amqp.Channel,
error) {
    // connect to RabbitMQ
    conn, err :=
amqp.Dial(fmt.Sprintf("amqp://%s:%s@%s:%s/",
ms.Config.RabbitMQUser, ms.Config.RabbitMQPassword,
ms.Config.RabbitMQUrl, ms.Config.RabbitMQPort))
    if err != nil {
        ms.Logger.Fatalw("failed to connect to RabbitMQ",
"err", err)
    }

    // open channel
    channel, err := conn.Channel()
    if err != nil {
        ms.Logger.Fatalw("failed to open RabbitMQ
channel", "err", err)
    }

    return channel, nil
}
```

pkg/logger/main.go

```
package logger

import (
    "go.uber.org/zap"
    "go.uber.org/zap/zapcore"
)
```

```
// New is used to create a new logger instance.
func New(logLevel string, production bool)
*zap.SugaredLogger {
     // create .og level
     var level zapcore.Level
     level.Set(logLevel)

     // logger config
     config := zap.Config{
          Development:      !production,
          Encoding:         "json",
          Level:            zap.NewAtomicLevelAt(level),
          OutputPaths:      []string{"stderr"},
          ErrorOutputPaths: []string{"stderr"},
          EncoderConfig: zapcore.EncoderConfig{
               EncodeDuration:
zapcore.SecondsDurationEncoder,
               LevelKey:       "level",
               EncodeLevel:    zapcore.CapitalLevelEncoder,
// e.g. "Info"
               CallerKey:      "caller",
               EncodeCaller:   zapcore.ShortCallerEncoder,
// e.g. package/file:line
               TimeKey:        "timestamp",
               EncodeTime:     zapcore.ISO8601TimeEncoder,
// e.g. 2020-05-05T03:24:36.903+0300
               NameKey:        "name",
               EncodeName:     zapcore.FullNameEncoder, //
e.g. GetSiteGeneralHandler
               MessageKey:     "message",
               StacktraceKey:  "",
               LineEnding:     "\n",
          },
     }

     if !production {
          config.Encoding = "console"
```

```
        // config.EncoderConfig.LevelKey = ""
        // config.EncoderConfig.CallerKey = ""
        config.EncoderConfig.TimeKey = ""
        config.EncoderConfig.NameKey = ""
        config.EncoderConfig.EncodeLevel =
zapcore.CapitalColorLevelEncoder
    }

    // build logger from config
    logger, _ := config.Build()

    // configure and create logger
    return logger.Sugar()
}
```

pkg/config/main.go

```
package config

import (
    "fmt"
    "strings"

    "github.com/iamolegga/enviper"
    "github.com/spf13/viper"
)

// Config represents shared configuration options for all
rd-workers microservices and packages.
type Config struct {
    Production bool `mapstructure:"production"`

    // logger
    LogLevel    string `mapstructure:"log_level"`
    LogResponse bool   `mapstructure:"log_response"`

    // Google Storage
```

```
    StorageBucket string `mapstructure:"storage_bucket"`

    // RabbitMQ
    RabbitMQUser     string `mapstructure:"rabbitmq_user"`
    RabbitMQPassword string
`mapstructure:"rabbitmq_password"`
    RabbitMQUrl      string `mapstructure:"rabbitmq_url"`
    RabbitMQPort     string `mapstructure:"rabbitmq_port"`

    // Apple Push Notification service
    APNSToken            string
`mapstructure:"apns_token"`
    APNSKeyID            string
`mapstructure:"apns_key_id"`
    APNSTeamID           string
`mapstructure:"apns_team_id"`
    APNSBundleIdentifier string
`mapstructure:"apns_bundle_identifier"`
}

// New is used to create and configure config.
// Default config name is .workers_cfg.
// Default config locations are current dir, repo root and
/vault/secrets.
func New() *Config {
    viper := enviper.New(viper.New())

    // initialize config
    viper.SetConfigName(".workers_cfg")
    viper.AddConfigPath("../..") // root of the git repo
    viper.AddConfigPath(".")
    viper.AddConfigPath("/vault/secrets/")
    viper.SetEnvPrefix("RD") // env vars have higher
precedence than config file
    viper.SetEnvKeyReplacer(strings.NewReplacer(".", "_"))
    viper.AutomaticEnv()
    if err := viper.ReadInConfig(); err == nil {
```

```
            viper.WatchConfig()
        }
        // We tolerate missed config file cuz params may come
from env
        // config defaults
        viper.SetDefault("log_level", "info")

        viper.SetDefault("rabbitmq_user", "guest")
        viper.SetDefault("rabbitmq_password", "guest")
        viper.SetDefault("rabbitmq_url", "localhost")
        viper.SetDefault("rabbitmq_port", "5672")
        viper.SetDefault("storage_bucket",
"user-content-staging-ryddm-co")

        // read config
        var config Config
        err := viper.Unmarshal(&config)
        if err != nil {
            panic(fmt.Sprintf("failed to read config: %s",
err.Error()))
        }

        return &config
}
```

chart/Chart.yaml

```
apiVersion: v2
name: service
description: A Helm chart for Kubernetes

# A chart can be either an 'application' or a 'library'
chart.
#
# Application charts are a collection of templates that can
be packaged into versioned archives
# to be deployed.
```

```
#
# Library charts provide useful utilities or functions for
the chart developer. They're included as
# a dependency of application charts to inject those
utilities and functions into the rendering
# pipeline. Library charts do not define any templates and
therefore cannot be deployed.
type: application

# This is the chart version. This version number should be
incremented each time you make changes
# to the chart and its templates, including the app
version.
# Versions are expected to follow Semantic Versioning
(https://semver.org/)
version: 0.1.2

# This is the version number of the application being
deployed. This version number should be
# incremented each time you make changes to the
application. Versions are not expected to
# follow Semantic Versioning. They should reflect the
version the application is using.
appVersion: 1.16.0
```

chart/templates/ambassador.yaml

```
---
apiVersion: getambassador.io/v1
kind: Mapping
metadata:
  name: {{ .Release.Name }}
spec:
  host: {{ .Values.host }}
  prefix: {{ .Values.prefix }}
  rewrite: {{ .Values.prefix }}
  service: {{ .Release.Name }}.{{ .Release.Namespace }}.svc
```

```
  add_response_headers:
    Vary: Origin, Access-Control-Request-Headers,
Access-Control-Request-Method
  cors:
    origins: "*"
    methods: GET, POST, PUT, PATCH, DELETE, OPTIONS
    headers:
      - Content-Type
      - Authorization
    exposed_headers: "*"
```

chart/templates/deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}
  labels:
    app: {{ .Release.Name }}
    service: {{ .Values.host }}
  annotations:
    config.linkerd.io/skip-inbound-ports: 80, 443, 8080
spec:
  replicas: {{ .Values.replicas }}
  strategy:
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
    type: RollingUpdate
  selector:
    matchLabels:
      app: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app: {{ .Release.Name }}
        service: {{ .Values.host }}
```

```
    spec:
      affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - topologyKey: kubernetes.io/hostname
                labelSelector:
                  matchLabels:
                    app: {{ .Release.Name }}
      containers:
        - name: {{ .Release.Name }}
          image: "{{ .Values.image.repository }}:{{
.Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: 8080
              protocol: TCP
          resources:
            {{- toYaml .Values.resources | nindent 12 }}
          env:
            {{- toYaml .Values.env | nindent 12 }}
```

chart/templates/service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}
  labels:
    app: {{ .Release.Name }}
    service: {{ .Values.host }}
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: http
      protocol: TCP
```

```
      name: http
  selector:
    app: {{ .Release.Name }}
```

Dockerfile

```
FROM golang:1.16-alpine AS build

# install vips
# Version latest is 8.7.x but go-bimg crashes because of
that
# Issue: https://github.com/h2non/bimg/issues/244
# So using 8.6.5 for now
ARG VIPS_VERSION="8.6.5"

RUN wget
https://github.com/libvips/libvips/releases/download/v${VIP
S_VERSION}/vips-${VIPS_VERSION}.tar.gz
RUN apk update && apk add automake build-base pkgconfig
glib-dev gobject-introspection libxml2-dev expat-dev
jpeg-dev libwebp-dev libpng-dev
# Exit 0 added because warnings tend to exit the build at a
non-zero status
RUN tar -xf vips-${VIPS_VERSION}.tar.gz && cd
vips-${VIPS_VERSION} && ./configure && make && make install
&& ldconfig; exit 0
RUN apk del automake build-base
RUN apk add --update go gcc g++

# name of the endpoint to build
ARG APP_NAME

# copy pkg mods
WORKDIR /srv/app/rd-workers/pkg
COPY pkg/go.mod .
```

```
COPY pkg/go.sum .

# copy endpoint mods
WORKDIR /srv/app/rd-workers/workers/${APP_NAME}
COPY workers/${APP_NAME}/go.mod .
COPY workers/${APP_NAME}/go.sum .

# download mods
RUN go mod download

# copy pkg code
WORKDIR /srv/app/rd-workers/pkg
ADD pkg .

# copy endpoint code
WORKDIR /srv/app/rd-workers/workers/${APP_NAME}
ADD workers/${APP_NAME} .

# build code
# -ldflags -w -s disables binary debugging and profiling
# https://stackoverflow.com/a/22276273/760726
RUN go build -o /srv/app/app -ldflags '-w -s' ./...

# copy deps
RUN cp -r deps /srv || mkdir /srv/deps

# run built binary
FROM golang:1.16-alpine
ENV GIN_MODE release
EXPOSE 8080

WORKDIR /srv
RUN mkdir -p /srv

# install libvips
RUN apk update && apk add automake build-base pkgconfig
glib-dev gobject-introspection libxml2-dev expat-dev
```

```
jpeg-dev libwebp-dev libpng-dev

# copy built binary
COPY --from=build /usr/local/lib /usr/local/lib
COPY --from=build /etc/ssl/certs /etc/ssl/certs
COPY --from=build /srv/deps /srv/deps
COPY --from=build /srv/app/app /srv/app

CMD ["/srv/app"]
```

gke.tf

```
resource "google_container_cluster" "primary" {
  name     = "primary-${var.postfix}"
  location = var.zone

  # We can't create a cluster with no node pool defined,
but we want to only use
  # separately managed node pools. So we create the
smallest possible default
  # node pool and immediately delete it.
  remove_default_node_pool = true
  initial_node_count       = 1


  # set kubernetes version
  min_master_version = "1.18.16-gke.302"
  release_channel {
    channel = "REGULAR"
  }

  master_auth {
    username = ""
    password = ""

    client_certificate_config {
```

```
      issue_client_certificate = false
    }
  }

  # make the cluster vpc-native
  network    = "default"
  subnetwork = "default"
  ip_allocation_policy {
    cluster_ipv4_cidr_block  = "/16"
    services_ipv4_cidr_block = "/22"
  }
}

resource "google_container_node_pool" "primary" {
  name       = var.postfix
  cluster    = google_container_cluster.primary.name
  node_count = var.gke_node_count
  location   = var.zone

  node_config {
    preemptible  = true
    machine_type = var.gke_node_type
    disk_size_gb = 20
    image_type   = "COS_CONTAINERD"

    metadata = {
      disable-legacy-endpoints = "true"
    }

    oauth_scopes = [
      "https://www.googleapis.com/auth/cloud-platform",
      "https://www.googleapis.com/auth/logging.write",
      "https://www.googleapis.com/auth/monitoring",

"https://www.googleapis.com/auth/devstorage.full_control",
# full acceess to cloud storage
    ]
```

```
  }
}

# get cluster data
data "google_container_cluster" "primary" {
  name     = "primary-${var.postfix}"
  location = var.zone
}

# output cluster data to parent module
output "gke_cluster" {
  value = google_container_cluster.primary
}
```

rabbitmq.hcl

```
# rabbitmq
resource "helm_release" "rabbitmq" {
  name      = "rabbitmq"
  namespace = kubernetes_namespace.module.metadata[0].name

  repository = "https://charts.bitnami.com/bitnami"
  chart      = "rabbitmq"
  version    = "8.12.2"

  set {
    name  = "auth.password"
    value = "PweABgWs_fMJuA@f!EANjs-G"
  }

  depends_on = [kubernetes_namespace.module]
}

# ambassador config
resource "kubectl_manifest" "rabbitmq-mapping-admin" {
  yaml_body = <<YAML
```

```
apiVersion: getambassador.io/v1
kind: Mapping
metadata:
  name: rabbitmq-admin-${var.postfix}
  namespace:
${kubernetes_namespace.module.metadata[0].name}
spec:
  host: rabbitmq.${var.domain}
  prefix: /
  service:
rabbitmq.${kubernetes_namespace.module.metadata[0].name}:15
672
  timeout_ms: 20000
  cors:
    origins: "*"
    methods: "*"
    headers: "*"
    exposed_headers: "*"
  allow_upgrade:
    - websocket
YAML

  depends_on = [kubernetes_namespace.module]
}
```

ДОДАТОК В

ВІДГУК

на дипломный проект бакалавра на тему:

«Разработка сайта для работы с участниками объединений совладельцев многоквартирных домов (ОСМД)» студента группы КНгр-12-1 Гордиенко Виталия Александровича.

ДОДАТОК Г

РЕЦЕНЗІЯ

на комплексну кваліфікаційну роботу бакалавра на тему:

«Створення хмарної розподіленої високопродуктивної системи обробки зображень і відео»

студента групи 126-17-1 Атамаса Іллі Олександровича.

Зростання обсягів медіа та соціальних мереж ставить перед бізнесом нові завдання. Реалізація сучасних веб і мобільних застосунків неминуче вимагає побудови і розробки систем для оптимізації медіа файлів (зображень та відео) з метою їх подальшого показу користувачам. Така система повинна забезпечувати швидке завантаження зображень і відеофайлів, забезпечуючи тим самим гарний користувацький досвід користувачам будь-яких соціальних мереж, веб-застосунків, мобільних застосунків. Також така система має забезпечувати зниження навантаження на інфраструктуру описаних мереж.

В рецензованій кваліфікаційній роботі створено хмарну розподілену високонавантажену вискодоступну систему обробки і оптимізації зображень і відео.

Використовувані технології розробки даних програмних засобів обліку та контролю різноманітних процесів є евристичною компонентою і безпосередньо пов'язане з об'єктом діяльності фахівця спеціальності 126 «Інформаційні системи та технології» галузі знань 12 «Інформаційні технології».

Студент І.О. Атамас досить добре розібрався в специфіці застосування засобів системи розробки застосунків на мові програмування Go та системі Google Cloud Platform для побудови відповідної інформаційної системи.

До недоліків роботи слід віднести:

– недостатньо повно розкрита тема хмарних сховищ.

Однак розроблена система - повноцінний продукт для оптимізації фото і відео, тож його можна вважати закінченим етапом виконаної кваліфікаційної роботи.

На підставі вищевикладеного, можна зробити висновок, що представлені матеріали цілком відповідають вимогам, що пред'являються до кваліфікаційних робіт першого рівня вищої освіти, тобто ступеню бакалавра.

З огляду на весь спектр створених компонентів, що забезпечують формування даної кваліфікаційної роботи, в цілому вона заслуговує на оцінку «добре», а її виконавець, студент Атамас І.О., присвоєння йому відповідної кваліфікації.

Рецензент, доцент кафедри

ПЗКС, канд. техн. наук **Л.М. Коротенко**