Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка web-сервісу з бізнес-логікою з використанням хмарних технологій
(тема)

Виконав:
студент 2 курсу, групи СШІм-22-3
Хомякова О.В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)
Керівник Кудрявцева М.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Хомяковій Ользі Валентинівні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка web-сервісу з бізнес-логікою з використанням хмарних технологій

затверджена наказом університету від 1 квітня 20 24 р. № 260Ст
2. Термін подання студентом роботи до екзаменаційної комісії 7 червня 20 24 р.
3. Вихідні дані до роботи Науково-технічні публікації, дані статей, результати експериментальних досліджень по технологіям, методам, моделям, алгоритмам ройового інтелекту та агентних систем

4. Перелік питань, що потрібно опрацювати в роботі
1) Вступ, мета роботи та постановка задачі, визначення бізнес-логіки
2) Огляд технології хмарних обчислень,
3) порівняння Google App Engine та Amazon Cloud
4) Аналіз платформи Google App
5) Вибір хмарної платформи, розробка додатку під платформу App Engine

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	01.04.2024	виконано
2	Анализ предметної області та постановка задачі	10.04.2024	виконано
3	Теоретичні дослідження бізнес-логіки у	20.04.2024	виконано
	програмному додатку		
4	Теоретичні дослідження хмарних обчислень	30.04.2024	виконано
5	Розробка web-сервісу з бізнес-логікою з	10.05.2024	виконано
	використанням хмарних технологій		
6	Підготовка пояснювальної записки	31.05.2024	виконано
7	Попередній захист кваліфікаційної роботи	05.06.2024	виконано
8	Захист перед ЕК	07.06.2024	

Дата видачі завдання 1 квітня 20 24 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Кудрявцева М.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 79 с., 20 рис., 2 табл., 4 дод., 20 джерел.

БІЗНЕС-ЛОГІКА, БІЗНЕС-ПРАВИЛА, ХМАРНА ПЛАТФОРМА, ХМАРНІ ТЕХНОЛОГІЇ, BRMS, GOOGLE APP ENGINE, PAAS.

Метою кваліфікаційної роботи є дослідження процесу розробки web-додатків із бізнес-логікою замовлень на хмарній платформі.

Об'єктом дослідження є бізнес-логіка та хмарні технології.

Методом дослідження є аналіз відкритих джерел на тему роботи та реалізація додатку на базі отриманих теоретичних даних.

Результатом роботи є розгорнення на хмарній платформі web-додаток по роботі із замовленнями та видалення фону з фотографій.

ABSTRACT

Master’s thesis contains: 79 pp, 20 fig., 2 tabl., 4 ann., 20 references.

BUSINESS LOGIC, BUSINESS-RULES, BRMS, CLOUD PLATFORM, CLOUD TECHNOLOGIES, GOOGLE APP ENGINE, PAAS.

The purpose of the qualification work is to study the process of developing web applications with the business logic of orders on a cloud platform.

The object of research is business logic and cloud technologies.

The research method is the analysis of open sources on the topic of the work and the implementation of the application based on the obtained theoretical data.

The result of the work is the deployment of a web application for working with orders and removing the background from photos on a cloud platform.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface – інтерфейс програмування програм;

BRMS – Business Rule Management System – система управління бізнес-правилами;

GAE – Google App Engine – хмарна платформа для розробки та розгортання веб-додатків в інфраструктурі Google;

GFS – Google File System – файлова система Google;

IaaS – Infrastructure-as-a-Service – інфраструктура як сервіс;

MVC – Model-View-Controller – шаблон (патерн) програмування;

PaaS – Platform-as-a-Service – платформа як сервіс;

SaaS – Software-as-a-Service – програмне забезпечення як сервіс.

ВСТУП

Програмне забезпечення може розглядатися як процес, що розвивається спільно з предметною областю, тому його гарна адаптованість до швидкого розвитку та зміни бізнес-параметрів дуже важлива.

Розглядаючи розробку звичайного web-програми, може виникнути питання: «Чи можна ізолювати контроль та управління бізнес-логікою від низькорівневих архітектурних понять під час процесу розробки?». В даний час для вирішення цієї проблеми є як техніка програмування, так і програмні засоби.

Однак web-додаток, що розробляється, необхідно десь запускати, а також брати необхідну обчислювальну потужність для розв'язуваних додатком завдань. У цій проблемі можуть допомогти хмарні технології, які не тільки надають хостинг, але також надають додаткові інструменти для ефективної розробки та контролю за програмою.

Хмарні обчислення як обчислювальна парадигма останнім часом стали темою високого дослідницького інтересу. Це стало привабливою альтернативою традиційним обчислювальним середовищам. Більшість обчислень у хмарі, однак, фокусується на постачальниках хмарної інфраструктури IaaS. Однак існують альтернативні сервіси, що все більше набирають популярності PaaS, які збільшують ефективність розробки додатків, а також спрощують процес їх менеджменту.

Таким чином, розробка web-додатку під хмарну платформу, яка була б здатна швидко та ефективно змінювати свої правила та параметри у бізнес-логіці, викликає дослідницький інтерес.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Мета роботи та постановка задачі

У цій роботі розглядається питання про дослідження процесу розробки додатків з бізнес-логікою на хмарній платформі та реалізації конкретного сервісу з бізнес-логікою замовлень з обробки фотографій в автоматичному та напівавтоматичному режимі.

Задача потребує чимало ресурсів процесора та оперативної пам'яті. Тут на допомогу приходять хмарні технології. Вони позбавляють розробника подібних питань і надають всю необхідну інфраструктуру. Підтримка подібних систем самотужки була б дуже дорогою.

Для розміщення Web-додатку з доступом до нього користувачів необхідно вибрати місце його хостингу та засобів розробки. У роботі розглядається поняття хмарних технологій, здійснюється аналіз сучасних провайдерів та вибирається конкретна платформа. Оскільки існує кілька видів хмарних сервісів, варто зазначити, що в цій роботі докладніше розглядається саме PaaS, оскільки саме цей вид служб дозволяє підвищити якість та швидкість реалізації програми для її розробників. Як така платформа вибирається Google App Engine і наводяться її характеристики, які дозволяють віддати перевагу перед іншими подібними сервісами.

Google App Engine є хмарною платформою PaaS, яка сильно спрощує розробку масштабованих веб-додатків. У роботі досліджуються можливості App Engine з погляду розгортання повноцінного додатка з бізнес-логікою замовлень та можливості платформи, які полегшують цей процес.

Розроблений web-сервіс для видалення фону з фотографій надає можливість користувачам організовувати фотографії на замовлення. Замовлення відображаються на веб-інтерфейсі у певних статусах. Далі користувачі переглядають фотографії у своїх замовленнях і мають можливість скористатися інструментами для коригування фотографій, які

система обробила недостатньо добре.

В роботі досліджується бізнес-логіка у всіх шарах програми, від базових концепцій проекту до високорівневих архітектурних патернів. У роботі також вказуються деякі техніки програмування та програмні інструменти для успішного відокремлення бізнес-логіки від її реалізації та технічних деталей.

Виходячи з розгляду способів відділення бізнес-логіки та легкої адаптованості бізнес-правил, досліджується такий спосіб як використання окремого, спеціалізованого програмного забезпечення для управління бізнес-логікою.

1.2 Визначення бізнес-логіки

Бізнес-логіка – сукупність правил, принципів, залежностей поведінки об'єктів предметної галузі (галузі людської діяльності, яку реалізує система) [1]. Інакше можна сказати, що бізнес-логіка – це реалізація правил та обмежень операцій, що автоматизуються. Є синонімом терміну «логіка предметної області» (англ. domain logic).

Простіше кажучи, бізнес-логіка – це реалізація предметної сфери в інформаційній системі. До неї відносяться, наприклад, формули розрахунку щомісячних виплат з позик (у фінансовій індустрії), автоматизована відправка повідомлень електронної пошти керівнику проекту після закінчення частин завдання всіма підлеглими (у системах управління проектами), відмова від готелю при скасуванні рейсу авіакомпанією (у туристичному бізнесі)).

У фазі бізнес-моделювання та розробки вимог бізнес-логіка може описуватися у вигляді:

– тексту;

– концептуальних аналітичних моделей предметної галузі (онтології);

– бізнес-правил;
– різноманітних алгоритмів;
– діаграм діяльності;
– графів та діаграм переходу станів;
– моделей бізнес-процесів.

У фазі аналізу та проектування системи бізнес-логіка втілюється в різних діаграмах мови UML або подібних до неї. У фазі програмування бізнес-логіка втілюється в коді класів та їх методів, у разі використання об'єктно-орієнтованих мов програмування, або процедур та функцій у разі застосування процедурних мов.

На мові розробників програмного забезпечення бізнес-логікою також називаються програмні модулі, що її реалізують, та рівень системи, на якому ці модулі знаходяться (англ. business logic layer, domain logic layer).

У багаторівневих (багатошарових) інформаційних системах цей рівень взаємодіє з нижчим рівнем інфраструктурних сервісів (англ. infrastructure layer), наприклад, інтерфейсом доступу до бази даних або файлової системи та вищим рівнем сервісів програми (англ. application services layer), який, у свою чергу, взаємодіє з рівнем інтерфейсу користувача (англ. user interface layer) або зовнішніми системами.

1.3 Поняття web-додатку

Розробка web-додатків нині займає один із провідних напрямів IT-сфери. Web-додатки можуть бути як простими майданчиками для надання інформації користувачам, так і повноцінними програмами з багатою функціональністю (прикладом можуть бути SAAS додатки).

Дослідження в роботі зроблено в контексті web-додатку, тому слід пояснити, як традиційно розробляється корпоративний web-додаток та як організовується реалізація бізнес-логіки.

Бізнес-логіка оперує бізнес-даними. У цій роботі бізнес-даними

називатиметься підмножина даних, яку обробляє програма, і яку представляє бізнес-об'єкти предметної області (користувач, замовлення), опускаючи такі технічні поняття як функція, XML-файл.

По суті, розробка web-додатків не сильно відрізняється від традиційної розробки програмного забезпечення і зазвичай залучає такі завдання, як: взаємодія з базою даних та з іншими додатками, надання інтерфейсу для взаємодії з додатком. Проте web-додатки є клієнт-серверними. На стороні клієнта, web-додаток зазвичай є Document Object Model (об'єктну модель документа), яка керується сценаріями (зазвичай Javascript), завдяки яким користувач може взаємодіяти зі сторінкою.

У роботі не розглядається розробка клієнта, оскільки більшість бізнес-логіки зосереджено на сервері. На сервері відбувається обробка більшості запитів користувача, зв'язок між додатком і зовнішніми сервісами, доступ до бази даних. Таким чином, розгляд проблем масштабування програми залежно від кількості запитів та гнучка адаптованість програми до змінних бізнес-параметрів є дуже важливим завданням.

1.4 Шаблон Model-View-Controller

Розробка web-програми може бути реалізована через три стадії:

– створення низькорівневого обробника бізнес-логіки, який обробляв би і зберігав бізнес-дані в базу даних, або передавав їх зовнішнім web-сервісам;

– створення високорівневого інтерфейсу користувача;

– з'єднання цих рівнів, обробляючи вхідні запити, передаючи бізнес-дані до подання та реагуючи на введення даних користувачем.

Доволі поширеною проблемою є змішування цих відповідальності. Це призводить до проблем управління бізнес-даними, які розкидані з різних аспектів коду. Без спільної угоди про те, як і де реалізовувати бізнес-код, кожен із розробників може реалізувати маніпулювання бізнес-даними на

будь-якому рівні. Як наслідок, будь-яка зміна вимагала б повну ретельну перевірку коду всієї програми. З цих проблем свого часу виник шаблон (паттерн) програмування, названий Model-View-Controller (MVC) [2].

Взаємодія між цими рівнями представлена на рисунку 1.1.

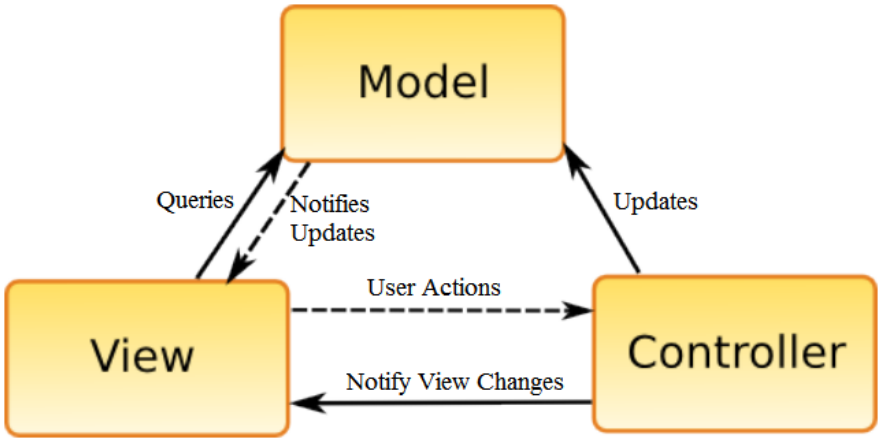

Рисунок 1.1 – Взаємодія компонентів у патерні MVC

При використанні патерну MVC розробка програми стає формалізованою, і код поділяється на кілька рівнів:

– модель містить бізнес-логіку та код рівня програми (application logic). Це основна частина програми, яка відповідає реалізації та подання предметної області у додатку;

– уявлення має справу з інтерфейсом. У web-додатках зазвичай це шар, що містить прості шаблони, куди вбудовується інформація користувача і далі відображається як HTML сторінок;

– контролер з'єднує ці елементи разом, перехоплюючи вхідні запити, обробляючи дані та направляючи їх до подання.

1.5 Бізнес-логіка та логіка програми

Раніше в роботі бізнес-логіка була визначена як підмножина коду

програми, яка управляє бізнес-об'єктами та даними. У підході MVC, описаному в попередньому підрозділі, маніпулювання бізнес-логікою має місце в моделі (рівень model) [3].

Логіка програми, з іншого боку, є підмножиною коду, яке управляє внутрішньою частиною програми: інтерфейсами, протоколами, абстрактними об'єктами. Таким чином, логіка програми не важлива як кінцевого користувача, так і людям, що пишуть специфікацію програми.

Поділ бізнес-логіки та логіки програми є гарною практикою в розробці програми, і патерн MVC був розроблений, щоб спростити це завдання. Проте такого поділу важко досягти практично [4]. Саме тому є інші способи вирішення цієї проблеми, які суттєво спрощують маніпулювання бізнес-логікою.

2 СИСТЕМА УПРАВЛІННЯ БІЗНЕС-ПРАВИЛАМИ

Rule Engines – категорія програм, розроблених для управління бізнес-логікою як окремим об'єктом. Система управління бізнес-правилами (BRMS) – програма, що складається з Rule Engine та додаткового програмного забезпечення для управління та маніпулювання правилами.

2.1 Формалізація бізнес-логіки

Як відомо, штучний інтелект – це сфера інформатики, яка покликана зробити комп'ютери здатними приймати рішення як люди. Його застосування в корпоративних додатках дозволяє здійснити кращу обробку предметної області програми, делегуючи деякі ключові рішення в програмне забезпечення, що розробляється. В автоматизації деяких рішень додатком є багато переваг.

Типовим прикладом може бути дисконтна політика. Торговий представник певної компанії може керувати базою клієнтів, використовуючи систему CRM (система управління взаємовідносинами з клієнтами), і застосовувати знижки згідно з директивами, наданими йому відділом продажів. Або безпосередньо відділ продажів міг би встановити параметри дисконтної політики у додаток через інтерфейс користувача. Таким параметром може бути така пропозиція, виражена природною мовою: кожному клієнту компанії, який зареєстрований у ній більше одного року, має бути запропонована 20% знижка на відкриття нового облікового запису.

Одна з галузей прикладного штучного інтелекту – експертні системи. Експертні системи використовують дані, які зберігаються в базі знань, і застосовують їх до даних додатка, щоб зробити висновки та висновки, використовуючи парадигми штучного інтелекту. Ранні експертні системи були впроваджені в код програми, але подальший розвиток призвів до

використання окремого експертного модуля, що містить базу знань та код прийняття рішень, що взаємодіє через програмну оболонку. Цей експертний модуль має назву Rule Engine.

Системи управління бізнес-правилами (BRMS) – це розширена версія Rule Engine, спеціально розроблена для використання додатками з різною бізнес-логікою, що потенційно змінюється. Зазвичай така система включає програмне забезпечення, що дозволяє редагувати правила і тестувати їх.

2.2 Бізнес-правила

Функціональність програми може бути розділена окремі дії, які можуть бути індивідуально протестовані. Це називають юніт-тестуванням (unit testing). Те саме істинно для бізнес-логіки: вона може бути формалізована в окремі висловлювання, пов'язані з бізнес-об'єктами.

Бізнес-правило – це певне висловлювання чи твердження, яке визначає чи обмежує певний аспект бізнесу [5]. Бізнес-правила призначені для того, щоб побудувати структуру бізнес-логіки програми або керувати та впливати на його поведінку.

Граматика та синтаксис вираження бізнес-правил залежить від реалізації. Вони можуть бути дуже близькими до природної мови. Наприклад: «Постачальник електроенергії не повинен поставити одному клієнту більше 100 кВт». Така мова виразів має слідувати таким властивостям: атомарність, незалежність, декларативність, приховування технічних деталей.

Правила повинні бути впорядковані та задіяні в певний момент виконання програми. Тому правила групуються в структуру, яка називається потік правил (англ. ruleflow). Під час виконання програми Rule Engine застосовує правила до бізнес-даних у порядку, визначеному ruleflow [6].

2.3 Обробка бізнес-правил

Однією з частин Rule Engine є механізм логічного висновку, який відповідає за виконання правил, коли відбувається збіг вхідних даних до певних шаблонів. Rule Engine об'єднує правила в ланцюжок згідно з цим ruleflow [7].

Як стратегії вирішення завдань використовуються два загальні методи логічного висновку: прямий логічний висновок і зворотний логічний висновок.

Прямий логічний висновок є метод формування міркувань від фактів до висновків, які випливають із цих фактів. Наприклад, якщо перед виходом з дому людина виявить, що йде дощ (факт), то він має взяти з собою парасольку (висновок).

Зворотний логічний висновок передбачає формування міркувань у зворотному напрямку – від гіпотези (потенційного висновку, що має бути доведено) до фактів, що підтверджують гіпотезу. Наприклад, якщо хтось не виглядає назовні, але побачив, як хтось увійшов до будинку з вологими черевиками та парасолькою, то можна прийняти гіпотезу, що йде дощ.

Щоб підтвердити цю гіпотезу, достатньо запитати дану людину, чи дощ. У разі позитивної відповіді буде доведено, що гіпотеза є істинною, тому вона стає фактом. Гіпотеза може розглядатися як факт, істинність якого викликає сумнів і має бути встановлена. У такому разі гіпотеза може інтерпретуватися як мета, яка має бути доведена.

2.4 Управління бізнес-правилами

Rule Engine – ядро BRMS, яке також включає інструменти для його інтегрування до певної архітектури та платформи для командної роботи з управління правилами [8].

Життєвий цикл програми з використанням BRMS представлений рисунку 2.1.

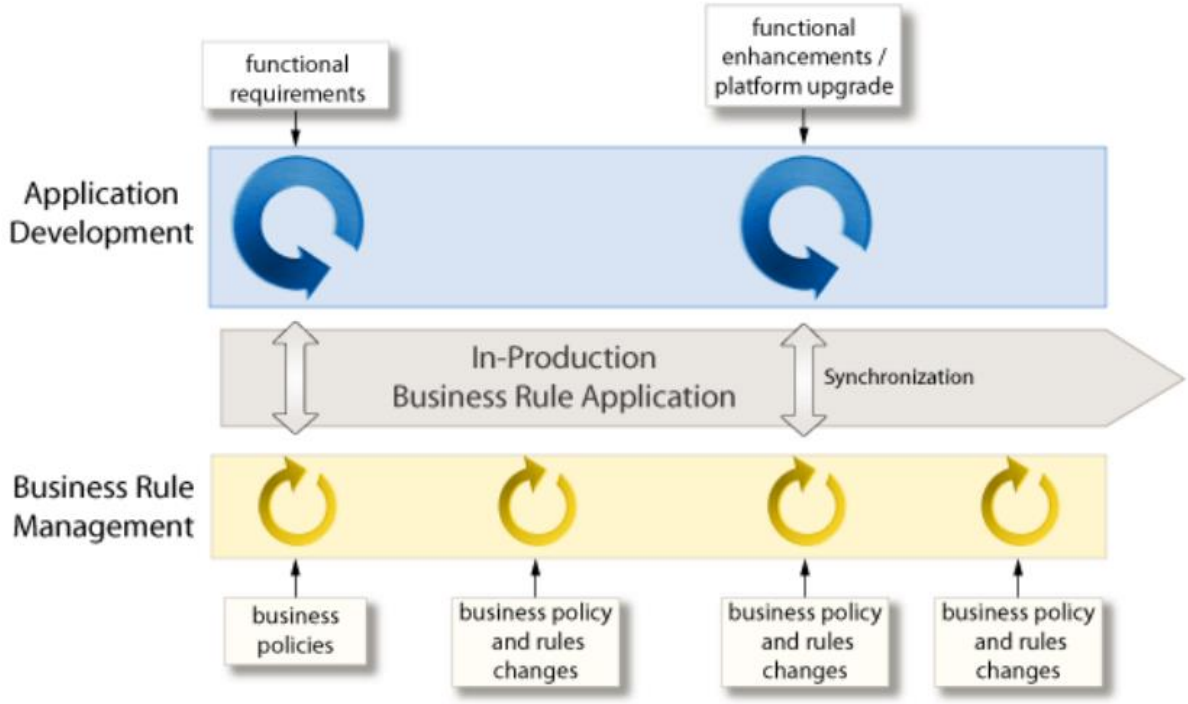

Рисунок 2.1 – Життєвий цикл програми з використанням BRMS

Є кілька причин, які роблять використання BRMS у бізнес-додатку корисним:

– досягається повний поділ між бізнес-логікою та логікою програми;

– виділення бізнес-логіки в окремі правила покращує легкість підтримки програмного забезпечення та його гнучкість;

– бізнес-правила можуть часто змінюватися без впливу на сам додаток (не доводиться змінювати код програми).

В результаті застосування BRMS діаграма взаємодії шарів програми перетворюється на зображену на рисунку 2.2.

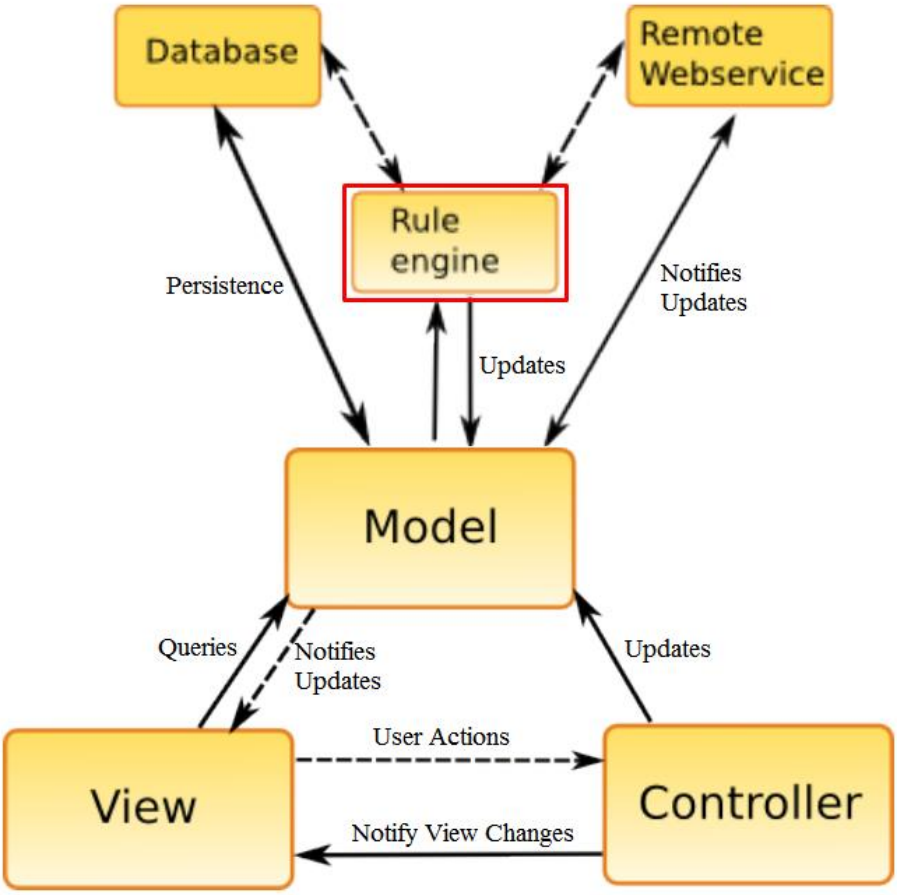

Рисунок 2.2 – Застосування BRMS до патерну MVC

Часто технічний розрив між розробниками та людьми, що приймають рішення про бізнес-правила додатка, заважає своєчасно здійснювати бізнес-операції на програмному забезпеченні [9].

3 ОГЛЯД ТЕХНОЛОГІЇ ХМАРНИХ ВИРАХУВАНЬ

3.1 Архітектури

Хмарні обчислення базується на визначенні послуг [10]. Усього розрізняють 3 види робіт.

Software as a Service (SaaS) – це програми в Інтернеті [11]. Зазвичай користувач може запустити ці програми, використовуючи веб-браузер. Користувач нічого не знає про апаратне та програмне забезпечення, що використовується, яке використовується додатком, а просто має доступ до інтерфейсу через web-браузер, витягуючи потрібну інформацію і задіявши необхідну функціональність. Прикладом цього виду служб може бути Google Docs.

Platform as a Service (PaaS) – це сервіси, які фокусуються на розгортанні програм у повністю підготовленому оточенні, дозволяючи розробнику вибирати апаратні засоби або необхідне програмне забезпечення, надаючи повноцінний стек програмних рішень [12]. Такі послуги надають кошти всього життєвого циклу докладання: проектування товару, реалізація, тестування, розгортання, цілісність з базами даних тощо. У такого виду служб є три основні параметри:

– можливість розгортання, тестування та обслуговування додатків;

– розрахована на багато користувачів архітектура (масштабованість);

– спільні інструменти (кооперація розробників).

Прикладом такого виду служб може бути Google App Engine.

Infrastructure as a Service (IaaS) – це сервіси, які фокусуються на тому, щоб запропонувати комп'ютерну інфраструктуру [13]. Провайдери такого виду надають сервери, з'єднання, програмне забезпечення моніторингу комп'ютерної інфраструктури та інші ресурси.

3.2 Види хмари

Відкрита хмара – традиційний вид хмари, де послуги надані деяким провайдером через Інтернет, і вони помітні для всіх.

Закрита хмара – хмара, яка хостить програми в межах приватної компанії та емулює хмару в Інтернеті, але розрахована лише для особистого користування (приватної мережі). Обслуговування інфраструктури таких хмар проводиться у межах цієї приватної організації.

Гібридна хмара – комбінація відкритої та закритої хмари. В організації може бути частина служб у її власній інфраструктурі, а частина у відкритій хмарі. Такі хмари часто є вибором тих організацій, які хочуть мати свої важливі дані та програми локально з можливістю їх контролювати, але також не хочуть вкладати занадто багато капіталу на підтримку всієї інфраструктури.

3.3 Переваги та недоліки хмарних обчислень

Переваги хмарних обчислень:

– менше початкових інвестицій. Якщо організація вирішує запускати якийсь потужний проект, то спочатку вона має купити та налаштувати всю необхідну інфраструктуру. Це означає багато інвестицій. Якщо навіть у цієї організації вже є якась інфраструктура для обслуговування проекту, то це означає, що всі сервери та персональні комп'ютери повинні задовольняти вимоги продуктивності та надійності, що важко контролювати з часом. Якщо ця організація починає використовувати деякі сервіси у хмарі, вона може інвестувати менше грошей у цю інфраструктуру та інвестувати їх в інші галузі проекту. Відбувається скорочення витрат через можливість не платити за фактори, що супроводжують (такі як інфраструктура). Організації необхідно заплатити лише за те, що використовується, і не потрібно фокусуватися на

обслуговуванні та відповідності інфраструктури чи програмного забезпечення вимогам часу, що використовується хмарними платформами;

– нова функціональність та актуальність програмного забезпечення. Постачальник хмарних послуг буде зацікавлений у тому, щоб інфраструктура та програмне забезпечення, що надається, працювало швидше і відповідало запитам сучасних вимог. У цьому полягає конкуренція провайдерів хмарних технологій;

– доступ до даних. Оскільки хмари обслуговуються через Інтернет, це означає, що доступ до всіх даних клієнтської організації або її інформації можливий через будь-який пристрій з доступом до Інтернету.

Хмарні обчислення також мають потенційні недоліки:

– доступність хмарної послуги. Велика навантаженість користувачами хмарних провайдерів погіршує якість сервісу. Провайдери не можуть гарантувати 100% зв'язку, але все ж таки рівень доступності таких служб досить високий;

– низька надійність у безпеці інформації. Не всі можуть нормально сприймати той факт, що їхня цінна інформація зберігається за межами організації. Зазвичай такі організації вирішують винести на хмару лише некритичні дані, а конфіденційну інформацію зберігати всередині організації;

– низька ефективність. Моменти відмови, затримка та швидкість мережі може бути вузьким місцем для продуктивності. Продуктивність системи може бути особливо сильно порушена у сервісах IaaS, де зазвичай потрібна передача великих обсягів даних;

– проблема спеціалізованого налаштування програми. Послуги, запропоновані у відкритій хмарі, фокусуються на тисячі користувачів, не надаючи рішення у спеціалізованих проблемах, а надаючи загальні рішення, що не допускають багато персоналізації.

4 ПОРІВНЯННЯ GOOGLE APP ENGINE І AMAZON CLOUD

4.1 Google App Engine

Google App Engine (GAE) – платформа для розробки та розгортання веб-застосунків в інфраструктурі Google [14]. GAE можна безкоштовно використовувати до певного рівня споживання ресурсів додатком, після перевищення зазначених безкоштовних лімітів усі ресурси оплачуються відповідно до тарифікацій на кожний конкретний тип ресурсу. Розробники можуть використовувати Python і Java як дві основні мови розробки, а також поступово впроваджується підтримка інших мов.

4.1.1 База даних NoSQL

Бази NoSQL відносяться до нереляційних баз даних. Подібні бази даних не підтримують повну реляційну модель SQL, а також мають проблеми з узгодженістю даних. Подібні бази даних фокусуються на продуктивності та масштабованості, але як наслідок мають недостатню узгодженість та цілісність даних.

Зниження продуктивності реляційних баз даних відбувається зі збільшенням даних, і коли розмір даних досягає певного значення, стає складно керувати таким обсягом даних. З такими проблемами стикалися багато компаній, таких як Facebook та YouTube. Facebook перемістив свої дані на базу даних Cassandra, а YouTube почав використовувати Bigtable, обидва з яких є рішеннями NoSQL.

GAE надає базу даних Datastore, яка є оболонкою над BigTable та має високі показники надійності, швидкості вилучення даних та їх обробки у великих обсягах.

4.1.2 Файлова система Google

Файлова система Google (Google Fle System – GFS) – це масштабована, розподілена та інформаційно-ємна файлова система, як і багато традиційних розподілених файлових систем, вона забезпечує надійність, доступність, відмовостійкість, масштабованість і продуктивність [15]. Вона була розроблена інженерами Google, щоб задовольнити потреби зберігання цих програм. Є багато реалізацій GFS з відкритим вихідним кодом, насамперед розподілена файлова система Hadoop (HDFS), яку використовує Yahoo. Дизайн GFS ґрунтувався на кількох фактах та міркуваннях:

– відмови окремих компонентів, поширених у великих розподілених системах;

– файли великих розмірів;

– додавання великих файлів замість зміни наявних.

Проект GFS має такі особливості:

– складений із багатьох недорогих товарних серверів;

– розроблений для великих потокових читань або маленьких випадкових читань та великого потоку запису;

– паралельне додавання файлів, гарантуючи атомарність;

– висока пропускна здатність передачі даних.

4.1.3 Система зберігання даних BigTable

BigTable – це розподілена система зберігання управління структурованими даними на тисячах вузлів [16]. Вона була розроблена, щоб підтримувати масштабованість, високу доступність та надійність. Bigtable – це компроміс між повною підтримкою реляційної моделі даних та масштабованістю, вона надає просту модель даних з динамічним керуванням за форматом та лейаутом даних.

У Bigtable дані зберігаються у таблицях як у реляційних базах даних,

але ці таблиці – це багатовимірні сортовані карти, організовані як рядок, стовпець та мітка часу. Дані, на які посилається рядок, стовпець та мітка часу, називається клітиною (cell). Послідовні рядки з подібними ключами зберігаються поруч і відомі як tablets, а подібні ключі стовпців, які групуються поруч, називаються column families. Третя розмірність (timestamp) дозволяє зберігати кілька версій даних.

4.2 Amazon Cloud

4.2.1 Amazon Elastic Compute Cloud

Web-сервіси Amazon – набір хмарних сервісів, що дозволяють зберігання даних, networking, доставку контенту, обчислень і баз даних. Amazon Elastic Compute Cloud (EC2) – гнучкий хмарний сервіс, який є інфраструктурою як сервіс (IaaS) [17]. EC 2 – віртуальне обчислювальне оточення, яке імітує традиційну обчислювальну систему; користувачі можуть керувати обчислювальною потужністю, основною пам'яттю, нетворкінгом, операційною системою, прикладним програмним забезпеченням і т.д.

Характерні особливості Amazon EC2:

– дозволяє повну конфігурацію інфраструктури користувачем (можливість використовувати різні інструменти програмування, мови та програми);

– персистентне зберігання даних через Amazon Elastic Block Storage (користувачі можуть використовувати обсяги даних від 1 Гб до 1 Тб);

– дозволяє користувачеві вибирати обчислювальні вузли у різних зонах (locations);

– надає гнучкі IP-адреси інстансів, які прив'язуються до користувача і залишаються у користувача навіть у разі відмови інстансу (у

такому разі IP-адреса переймається іншою працюючою інстансом);

– забезпечує віртуальну приватну мережу (VPN) між Amazon EC2 та клієнтом, яка називається «Amazon virtual private cloud»;

– дозволяє додавати або видаляти інстанс згідно з правилами, визначеними користувачем (дана можливість називається «auto scaling»);

– гнучке вирівнювання навантаження автоматично розподіляє навантаження серед усіх інстансів;

– дозволяє використання образів віртуальних машин, які попередньо налаштовані установкою операційної системи та прикладного програмного забезпечення.

4.2.2 Amazon EC2 Storage

Amazon EC 2 не забезпечує постійне зберігання, тому коли інстанс видаляється – всі його дані губляться. Щоб додати постійне зберігання до Amazon EC 2, є ще один сервіс відомий як Amazon Elastic Block Store (EBS). EBS це просто сервіс зберігання, і користувач може сам відформатувати його будь-якою файловою системою [18]. EBS автоматично реплікуються на службу Amazon Simple Storage Service (S3), яка є високодоступною, масштабованою, розподіленою та надійною системою зберігання, заснованою на системі зберігання Amazon Dynamo.

4.2.3 Система зберігання даних Amazon Dynamo

Amazon Storage System заснована на системі зберігання високої доступності, надійності, масштабованості та розподіленості, відомої як Dynamo [19]. Dynamo забезпечує властивості ACID(Atomicity, Consistency, Isolation, Durability), водночас забезпечуючи високу доступність даних.

Dynamo – розподілене key-value сховище, децентралізована система зберігання, що складається із тисячі вузлів. Дані зберігаються серед усіх

вузлів, при цьому використовується узгоджений механізм хешування даних. Усі вузли спілкуються друг з одним через таблицю маршрутизації, що дозволяє їм знати дані, розміщені кожному вузлі. Дані реплекуються на багато вузлів, щоб забезпечити високу доступність та надійність. Dynamo надає функцію версіонування даних, що гарантує, що дані ніколи не загубляться.

5 АНАЛІЗ ПЛАТФОРМИ GOOGLE APP ENGINE

Основна ідея хмарних обчислень полягає в тому, що ресурси, програмне забезпечення та дані забезпечуються на вимогу користувача через Інтернет. Фактична технологія, яка використовується у хмарі, абстрагована від користувача, і всі завдання адміністрування покладені на постачальника послуг. Крім того, провайдер має справу з такими проблемами, як вирівнювання навантаження та масштабованість.

Зазвичай на вирішення таких проблем використовується віртуалізація ресурсу. Хмарні обчислення забезпечують гнучку та економічно ефективну альтернативу локальному управлінню обчислювальними ресурсами. Оплата зазвичай потрібна лише за фактичні ресурси, що були використані. Як було зазначено в попередніх розділах, хмарні послуги можуть бути класифіковані трьома категоріями рівнів абстракції послуги.

IaaS забезпечує лише звичайне зберігання даних, надає мережеві та обчислювальні ресурси. Користувач не керує або керує базовою хмарною інфраструктурою, але може розгорнути та запустити довільне програмне забезпечення, включаючи операційну систему та звичайні програми.

PaaS надає платформу для виконання створюваної програми, надає використання різних мов програмування та програмного забезпечення інструментів. Користувач не керує базовою хмарною інфраструктурою, зберіганням даних або операційною системою, але керує розгорнутими програмами.

SaaS забезпечує використання програм, розроблених виробником, що працює на хмарній інфраструктурі через клієнт, зазвичай веб-браузер. Користувач не має жодного контролю над інфраструктурою, операційною системою чи можливостями програмного забезпечення.

В даний час багато програм, що вимагають великих обчислювальних ресурсів у хмарі, фокусується на хмарах IaaS, таких як Amazon EC2, тому що під такими платформами може бути виконане будь-яке програмне

забезпечення, також немає жодних обмежень з точки зору операційної системи або мови програмування.

Google App Engine – хмарний сервіс PaaS, який фокусується на розгортанні веб-додатків, що масштабуються. Головним чином, ця платформа призначається для невеликих компаній, які не можуть дозволити собі дорогу інфраструктуру, щоб обробити велику кількість запитів або раптових піків трафіку. App Engine є основою для розробки базованого web-додатку, використовуючи Python або Java як мову програмування.

Програми розгорнуті на інфраструктурі серверів Google. Кожна програма може використовувати ресурси, такі як процесорний час, число запитів або пропускна здатність, що використовується [20]. Google надає певну кількість безкоштовних щоденних ресурсів кожній програмі. Якщо тарифікації включено, то нараховуються платежі за використання ресурсів, які перевищують безкоштовний ліміт, інакше web-додаток стає недоступним, якщо виснажено критичні ресурси.

Кожен член групи розробників може надати власний обліковий запис окремі ресурси. Проблема полягає в тому, що платформа дуже строго заточена на конкретні мови програмування, бібліотеки та багато іншого, що робить використання платформи менш зручним, ніж на звичайних платформах IaaS.

5.1 Архітектура

Загальна архітектура хмарної платформи App Engine може бути представлена як архітектура обробки окремого запиту до програми, розгорнутої на цій платформі. Усі головні елементи інфраструктури платформи беруть участь у обробці запиту, схема якої зображена на рисунку 5.1.

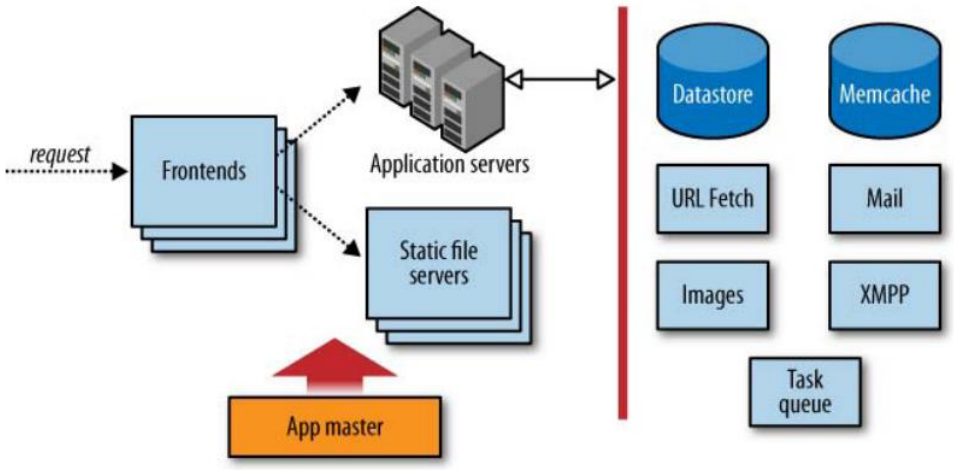

Рисунок 5.1 – Схема архітектури обробки запиту App Engine

Кожен запит спочатку приймається фронтендом App Engine. Він являє собою кілька машин фронтенду і балансувальник навантаження, який управляє належним розподілом запитів до фактичних машин. Фронтенд визначає, до якої програми адресується запит, оглядаючи доменне ім'я запиту. На наступному етапі фронтенд читає конфігурацію відповідної програми. Конфігурація програми визначає, як фронтенд обробляє запит, залежно від URL (статичні файли, зображення, сценарії Java або файли). Обробник запиту динамічно генерує відповідь на основі коду програми. Запити до статичних файлів передаються статичним файловим серверам.

Статичні файлові сервери оптимізовані для швидкої доставки ресурсів, які часто не змінюються. Чи статичний файл і повинен зберігатися на статичних файлових серверах, вирішується при розгортанні програми. Якщо запит порівнюється з обробником запиту, він передається серверам додатків. Вибирається один певний сервер і запускається екземпляр програми. Якщо вже є ініціалізований екземпляр програми, то він може бути перевикористаний балансувальником навантаження для обробки запиту. Далі викликається відповідний обробник запиту у додатку. Стратегії вирівнювання навантаження та розподільчих запитів до серверів додатків

все ще перебувають у стадії активної розробки в Google App Engine і дуже оптимізуються. Основна мета балансувальника – це швидко передавати запит обробнику, щоб гарантувати високу пропускну спроможність.

Те, скільки екземплярів програми запущено за один раз і як розподіляються запити, залежить від трафіку програми та зразків використання ресурсів. Сам код програми працює в середовищі виконання, яке абстрагує в собі операційну систему. Тобто немає можливості безпосередньо звернутися до операційної системи. Цей механізм дозволяє серверам керувати ресурсами, такими як цикли процесора та пам'ять для різних програм, що працюють на тому самому сервері.

Сервер програм чекає, поки обробник запиту завершить роботу і повертає відповідь на фронтенд, таким чином завершуючи обробку запиту. Оброблювачі запиту повинні завершитись перед поверненням даних, тому потокова передача даних не можлива. Якщо клієнт вказує, що дані потрібні в закодованому вигляді, надаючи відповідний заголовок запиту, дані автоматично стискаються, використовуючи формат zip.

App Engine складається з трьох основних частин: середовище виконання, сховище даних та сервіси, що масштабуються. Середовище виконання виконує код програми. Сховище даних дозволяє розробникам зберегти дані поза часом життя запитів. App Engine надає кілька послуг, що масштабуються, корисних для web-додатків.

5.2 Рантайм оточення

Як вже було згадано, код програми працює в середовищі виконання, яке абстраговано від базової операційної системи. Це ізольоване середовище, що запускає програми, називають sandbox. App Engine програми можуть бути запрограмовані у кількох мовах, таких як Java, Python та інші. Як наслідок, у кожної мови програмування є своє власне середовище виконання. Середовище виконання Java слідує за стандартами

Java сервлетів, забезпечуючи відповідне середовище для програми. Звичайні технології веб-застосунків Java, такі як JSP, також підтримуються. Підтримуються Java SDK у версії 6 та вище. Хоча програми зазвичай розробляються на Java, можуть використовуватися будь-які мови, що компілюються в байт-код Java (JavaScript, Ruby або Scala). «Sandbox» вводить кілька обмежень до додатків:

– розробники мають обмежений доступ до файлової системи. Файли, розгорнуті разом із додатком, можуть бути прочитані, однак немає ніякого доступу для запису;

– у програм немає прямого доступу до мережі, хоча запити HTTP можуть бути виконані через сервіси API;

– немає ніякого доступу до обладнання або операційної системи, що використовується;

– App Engine не підтримує домени без «www», такі як http://example.com, через записи канонічного імені, що використовуються для вирівнювання навантаження;

– використання потоків не дозволено;

– Java-програми можуть використовувати лише обмежений набір класів від стандартного середовища виконання Java.

Виконання програм у «sandbox», з одного боку, перешкоджає тому, щоб програми виконували шкідливі операції, які могли б нашкодити стійкості інфраструктури сервера або втрутитися в інші програми, що працюють на тій самій фізичній машині. З іншого боку, це дозволяє App Engine виконати автоматичне вирівнювання навантаження, тому що не має значення те, на якому обладнанні чи операційній системі виконується програма. Немає гарантії, що два запити будуть виконуватися на тій же машині, навіть якщо запити прийдуть один за одним і від того ж клієнта. Примірники того ж чи навіть різних програм можуть працювати на тій же машині, не впливаючи один на одного. «Sandbox» також обмежує ресурси, такі як ЦП або використання пам'яті і може відрегулювати додатки, які

використовують певну велику кількість ресурсів, щоб захистити програми, що виконуються на тій же машині. Один запит має максимум 60 секунд, щоб завершитися і повернути відповідь на клієнт.

5.3 База даних Datastore

Web-додаткам потрібен спосіб зберегти дані між запитами до програми. Традиційний підхід – реляційна база даних, що є єдиному сервері бази даних. Перевага такої системи полягає в тому, що кожен web-сервер завжди має свіжу версію даних. Однак, як тільки межа для обробки великої кількості паралельних запитів до бази даних досягнуто, стає складним масштабувати систему. Поряд із системами реляційних баз даних є різні інші підходи, такі як бази даних XML або об'єктні бази даних.

Datastore – власна розподілена служба зберігання даних Google App Engine. Основна ідея полягає в тому, щоб забезпечити високорівневий API та приховати від розробника деталі того, як дані зберігаються фактично. Це економить розробникам додатків трудовитрати вирішення завдання масштабованості даних. Парадигма бази даних Datastore найсильніше нагадує об'єктну базу даних. Дані називаються об'єктами та мають ряд властивостей. Значення властивостей можуть бути вибрані з ряду типів даних, що підтримуються.

Об'єкти – іменований вид, щоб забезпечити механізм категоризації даних. Така структура може здатися подібною до реляційної бази даних. Об'єкти нагадують рядки, та властивості нагадують стовпці у таблиці. Однак є деякі основні відмінності до реляційної бази даних. В першу чергу, в Datastore об'єкти того ж виду не вимагають мати одні й самі властивості. Крім того, двом об'єктам дозволяється мати властивість з тим самим ім'ям, але різними типами значення. Інша важлива відмінність – те, що властивості можуть містити колекцію значень. Об'єкти ідентифіковані ключем, який може бути згенерований автоматично App Engine, або вручну програмістом.

На відміну від первинного ключа в реляційній базі даних ключ об'єкта є не полем, а окремим аспектом об'єкта. App Engine використовує ключ у поєднанні з видом об'єкта, щоб визначити, де зберегти об'єкт у розподіленій інфраструктурі сервера.

Як наслідок, ключ та вид об'єкта не може бути змінено після створення. Індекси, що використовуються у Datastore, визначені у конфігураційному файлі. При тестуванні програми на локальному сервері розробки індекси автоматично додаються до конфігурації. Платформа розпізнає типові запити, виконані додатком, та генерує відповідні індекси.

Запит до Datastore складається з виду об'єкта з низкою умов та порядком сортування. Виконання запиту призводить до пошуку серед усіх об'єктів даного виду, що задовольняють даним умовам та повертаються у порядку, що відповідає сортуванню. Крім того, що запит може повернути повноцінні об'єкти, є також опція, яка дозволяє йому повернути лише значення ключів об'єктів. Це допомагає мінімізувати передачу даних від Datastore до програми, якщо лише деякі із запитаних об'єктів фактично будуть використовуватися. До даних web-програми зазвичай отримують доступ багато користувачів одночасно, таким чином, роблячи поняття транзакційності дуже важливим.

App Engine гарантує атомарність: кожне оновлення об'єкта, що включає багато властивостей або успішно виконується повністю або повністю скасовують запитану роботу, залишаючи об'єкт у його вихідному стані. Інші користувачі можуть бачити об'єкт тільки повністю оновленим або в його вихідному стані, але не в проміжному.

App Engine використовує механізм керування оптимістичного паралелізму. Передбачається, що транзакції відбуваються без конфліктів. Як тільки конфлікт відбувається (кілька користувачів намагаються оновити той самий об'єкт одночасно), об'єкт відкочується до його вихідного стану, і всі користувачі, які намагаються виконати оновлення, отримують помилку. Такий підхід є найефективнішим для системи, де конфлікти трапляються

досить рідко. Читання завжди успішно виконуватимуться, і користувач буде просто бачити нову версію об'єкта. Є можливість вважати багато об'єктів у групі, щоб гарантувати несуперечність даних. Є також можливість здійснювати транзакції вручну, пов'язуючи багаторазові операції бази даних транзакцію. Наприклад, програма може вважати об'єкт, оновити властивість, записати об'єкт назад і закоміті транзакцію. Якщо транзакція не відпрацювала, усі операції бази даних мають бути повторені.

Datastore забезпечує два стандартні інтерфейси Java для доступу до даних: Java Data Objects (JDO) і Java Persistence API (JPA). Реалізація двох інтерфейсів використовує платформу DataNucleus. App Engine також надає низькорівневий API, який може використовуватися для програмування подальших інтерфейсів бази даних. Низькорівневий API може також використовуватися безпосередньо з програми, що в деяких випадках може бути ефективнішим.

6 ВИБІР ХМАРНОЇ ПЛАТФОРМИ

Розробка корпоративної програми вимагає, щоб цільова платформа була в змозі забезпечити деякі вимоги. Ці вимоги варіюються від одного додатка до іншого, але можна узагальнити і виділити ряд особливостей додатків, відштовхуючись від яких можна зробити вибір хмарної платформи. Рисунок 6.1 показує групування корпоративних додатків у 4 категорії, такі як: Transactional або Non-Transactional, Data Intensive або Data Non-Intensive.

Platform	Application Requirements	
Amazon EC 2	Data Analysis Application (e.g. Banking, Finance)	Transactional Application
Google App Engine	Data Management Applications (e.g. Customer Support, Feedback, Issue Tracker)	Non-Transactional Application

Рисунок 6.1 – Матриця вибору хмарної платформи

Виходячи з таблиці GAE рекомендується як придатний для нетранзакційних додатків та тих корпоративних додатків, які не виконують важкий аналіз великого набору даних, оскільки GAE має погану глобальну підтримку транзакцій. Однак інші типи корпоративних додатків можуть бути реалізовані швидше та ефективно саме в GAE через велику кількість надійних інструментів розробки, які пропонує платформа. Корпоративні програми, які вимагають багато обчислювальних ресурсів, гірше підходять для GAE, оскільки GAE не дозволяє виділення ресурсів вручну. Amazon EC2, навпаки, дозволяє виділяти велику кількість ресурсів, щоб забезпечити високу продуктивність обчислення.

GAE забезпечує більш швидке встановлення середовища розробки та production середовища. У випадку з Amazon EC 2 потрібна конфігурація

ресурсів, таких як networking, встановлення операційної системи та програмного забезпечення, яке може бути складним та важким для підтримки системи. GAE усуває проблеми обслуговування операційної системи, її оновлення, встановлення патчів, усунення несправностей тощо, які є в Amazon EC 2.

Розробка для GAE може вимагати додаткового часу через деякі обмеження, а також підтримку лише деяких мов програмування, бібліотек та фреймворків. Однак сервіси для зберігання даних, їх обробки, автоматична масштабованість додатків та інші сервіси, що надаються, істотно полегшує процес розробки і робить платформу GAE дуже привабливою для розробки додатків у хмарі.

7 РОЗРОБКА ПРОГРАМИ ПІД APP ENGINE

7.1 Предметна галузь програми

У цій роботі як практичне застосування розробляється сервіс з бізнес-логікою замовлень з обробки фотографій в автоматичному режимі. Обробка фотографій є видаленням з них фону і можливим коригуванням результатів обробки за допомогою пропонованих інструментів. Під бізнес-логікою замовлень мається на увазі сукупність статусів, у яких може бути замовлення, і навіть дії, які проводяться над замовленням під час переходу до одного з таких статусів.

Автоматичне видалення фону – одна з класичних проблем комп'ютерного зору та обробки зображень, відома ще з 80-х років 20 століття. На даний момент розроблено безліч алгоритмів для обробки зображень, які спрощують це завдання, але не вирішують її повністю. Вирішення завдання видалення фону з абсолютно різних картинок в автоматичному режимі поки що не є можливим, однак його можна вирішувати вузькоспрямованим підходом для картинок із заданими обмеженнями. Такі системи мають практичний інтерес і можуть використовуватися вже зараз.

Актуальність такої системи підтверджується у сфері продажу товарів в інтернет-магазинах. Відомо, що товар на білому тлі продається щонайменше на 10–15% краще, ніж без нього. Проста автоматична процедура одразу приносить матеріальний ефект.

Данийweb-сервіс надає можливість користувачам організовувати фотографії на замовлення. Замовлення відображаються на веб-інтерфейсі у певних статусах. Далі користувачі переглядають фотографії у своїх замовленнях і мають можливість скористатися інструментами для коригування фотографій, які система обробила недостатньо добре.

7.2 Розробка web-додатку

При розробці використовуються такі технологи та інструменти:

– платформа App Engine;

– мова програмування Java сьомої версії;

– бібліотека Objectify для зручної роботи з базою даних Datastore;

– Spring framework як IoC (Inversion of Control) та DI (Dependency Injection) контейнер;

– послуги хмарної платформи.

У додатку є кілька головних сутностей, що відображають модельпредметної галузі:

– замовлення (Order);

– зображення (Image);

– користувач (User).

Замовлення – це набір картинок, що підлягає обробці, а також інформація про те, коли оброблялося замовлення, хто його переглядав та інша системна інформація. Замовлення має посилання на користувача, який створив його, а також поточний статус.

Структура замовлення, виражена засобами мови Java, представлена на рисунку 7.1. На ньому видно Java інструкції, які необхідні для роботи з базою даних Datastore. Є багато різних анотацій, доступних розробнику, що дозволяють гнучко налаштувати роботу з базою даних. Наприклад, інструкція «@Index» показує, що це властивість можна використовувати у запитах до бази даних.

Зображення має бути фотографією людей на однорідному тлі. Вона зберігає в собі інформацію про те, якому замовленню належить, шляхи до допоміжних файлів, які генеруються при обробці зображення та можуть бути використані для її покращення.

```
@Entity
class OrderImpl extends PersistentObject implements Order {

    @Index private String customerId;
    @Index private String customerName;
    @Index private String customerReference;
    @Index private Date receivedDate;
    private Boolean priority;
    @Index private Date dueDate;
    @Index private Date completedDate;
    @Index private OrderStatus status;
    @Index private String reviewer;
    @Index private Integer numberOfImages;
    @Unindex private int version;
    @Ignore
    private Map<ImageNumberCriteria, Integer> numberOfImagesOfCriteria;
    private Set<String> operators = Sets.newHashSet();
    private String notes;
    private String txtFile;
```

Рисунок 7.1 – Структура замовлення, виражена засобами Java

Життєвий цикл замовлення може бути виражений проходом за визначеними статусами – від «in progress» до «completed», і далі в архів. Статуси, в яких може бути замовлення, при проходженні через систему, а також дозволи на дії, які можуть виконуватися над замовленням у певному статусі, представлені на рисунку 7.2.

Замовлення потрапляє в систему, коли за FTP завантажені всі зображення та текстовий файл, що містить короткий опис замовлення та його пріоритет. Завантажені по FTP зображення насправді відправляються на зберігання в хмарне сховище файлів Google Cloud Storage (GCS), що надається платформою App Engine. Таким чином розробникам додатків під App Engine не потрібно турбуватися про те, де зберегти їх дані зі швидким доступом до них, до того ж GCS тарифікується тільки за обсяг даних, що використовується, зберігаючи гроші власникам додатків.

Як видно із рисунку 7.2 над замовленням можна виконувати деякі дії, які залежать від статусу, такі як: зміна деталей замовлення, пріоритету обробки, рев'юїра, архівування ордера.

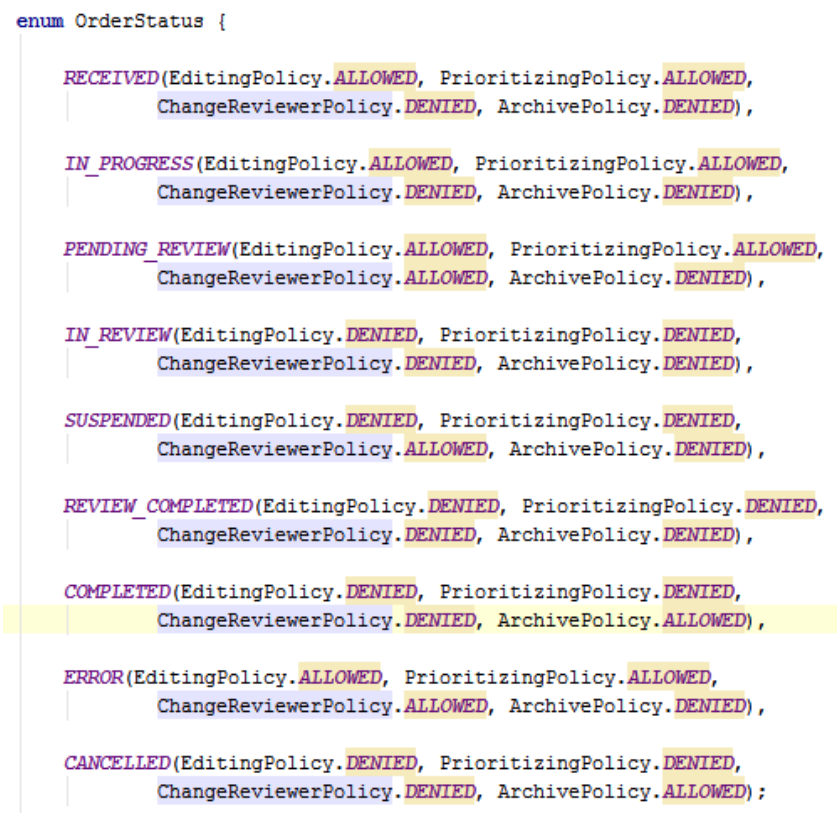

Рисунок 7.2 – Перелік статусів замовлення, виражений мовою Java

Потрапляючи в систему, замовлення отримує статус «in progress». Кожна з картинок потрапляє у чергу для обробки інтансом Compute Engine.

Compute Engine – IaaS сервіс, що входить до платформи App Engine. Він надає можливість піднімати віртуальні машини із довільним програмним забезпеченням. Можна вибрати кількість оперативної пам'яті та частоту процесора, яку номінально використовуватиме один інстанс. Інстанс – це екземпляр віртуальної машини.

Кожен інстанс Compute Engine оплачується за той час, який він працює. Щоб замовлення з великою кількістю картинок відпрацьовували швидко, потрібно запускати кілька інстансів пропорційно кількості картинок. Так як Compute Engine не має вбудованого механізму, що регулює кількість інстансів необхідних певного завдання, був написаний код, який допомагає це робити. Такий підхід називають оркестрацією. Оркестратор піднімає та завершує інстанси залежно від поточного стану завдання та

використовує технологію «cron job» – механізм, який допомагає призначити завдання, які виконуватимуться з проміжком у якийсь час. Опис завдань для «cron job» наводиться на рисунку 7.3.

Інстанси Compute Engine повідомляють web-сервіс про те, що зображення оброблені, і після того, як була оброблена остання зображення, замовлення переходить у стан «pending review», який дозволяє одному з користувачів, який називається оператором у термінах предметної області, розпочати роботу над замовленням перемістивши його в статус «review».

```
<?xml version="1.0" encoding="UTF-8"?>
<cronentries>
    <cron>
        <url>/admin/cron/orchestrator/orchestrate</url>
        <description>Check status regularly</description>
        <schedule>every 3 minutes</schedule>
        <target>orchestrator-backend</target>
    </cron>
    <cron>
        <url>/admin/cron/orders/archive</url>
        <description>Archive all completed or cancelled orders older than [archive period]</description>
        <schedule>every day 00:00</schedule>
    </cron>
    <cron>
        <url>/admin/cron/orders/suspend</url>
        <description>Suspend orders that are not being reviewed or repaired for more than 10 minutes</description>
        <schedule>every 2 minutes</schedule>
    </cron>
    <cron>
        <url>/admin/cron/channels/clear</url>
        <description>Deletes expired channels</description>
        <schedule>every 12 hours</schedule>
    </cron>
    <cron>
        <url>/admin/cron/holidays/update</url>
        <description>Update holidays</description>
        <schedule>1 of jan 00:00</schedule>
    </cron>
</cronentries>
```

Рисунок 7.3 – Опис cron job задач

«Review» – це процес перегляду картинок. Користувач може вказати, як виглядає зображення після обробки:

– «Looks Good», що означає, що зображення виглядає добре і не вимагає подальшої обробки;

– «Needs Replacement» – дає можливість користувачеві залити іншу картинку замість поточної;

– «Needs Repair», що означає, що зображення має бути

відправлено в ремонт, де користувач може відкоригувати її пропонованими інструментами.

Екран з діями, які можна робити над картинкою в режимі review показані на рисунку 7.4.

Логіка програми передбачає, що над замовленням може працювати кілька операторів. Для цього є статус замовлення suspended. «Suspended» – означає, що роботу над замовленням призупинено і її може продовжити інший оператор. Замовлення потрапляє у статус «suspended» у двох варіантах: коли замовлення не переглядали протягом 2 хвилин (це завдання виконується за допомогою «cron job», показаного на рисунку 8.3), а також коли користувач явно натиснув кнопку «Suspend».

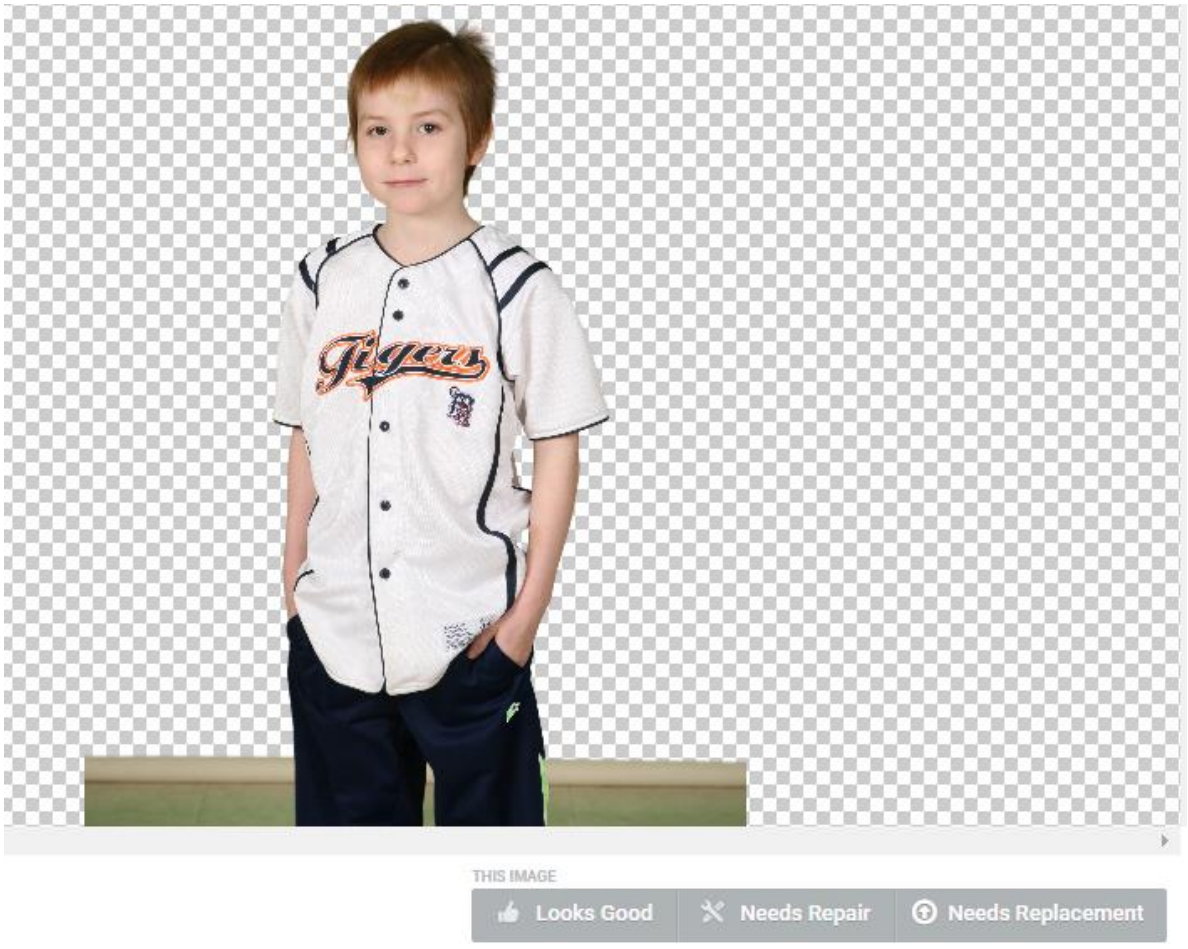

Рисунок 7.4 – Екран «Review»

Коли всі зображення переглянуті і оператор помітив їх як Looks Good, він натискає кнопку Complete Review і замовлення переходить у статус review completed. На цьому етапі система вирішує, що всі результуючі зображення добре оброблені і починає копіювати їх в результуючий бакет (GCS bucket). Коли всі зображення скопійовані, користувачі, що завантажили свої замовлення, можуть забрати оброблені зображення таким же способом, як і при завантаженні замовлення в систему.

Через деякий час, замовлення потрапляє в архів. Архів замовлень – це системна інформація про замовлення, що пройшли через систему. Таким чином, користувачі можуть бачити дату завантаження, час обробки замовлення та іншу корисну інформацію навіть після того, як отримали результати обробки картинок.

7.3 Демонстрація роботи програми

Початок роботи з програмою починається із завантаження картинок по FTP протоколу. Формується ордер та завдання на обробку, що використовують механізм платформи «task queue», зображений на рисунку 7.5. Кожне завдання обробляється інстансом Compute Engine (рисунок 7.6).

Рисунок 7.5 – Інтерфейс сервісу «Task Queue», що показує поточні завдання у системі

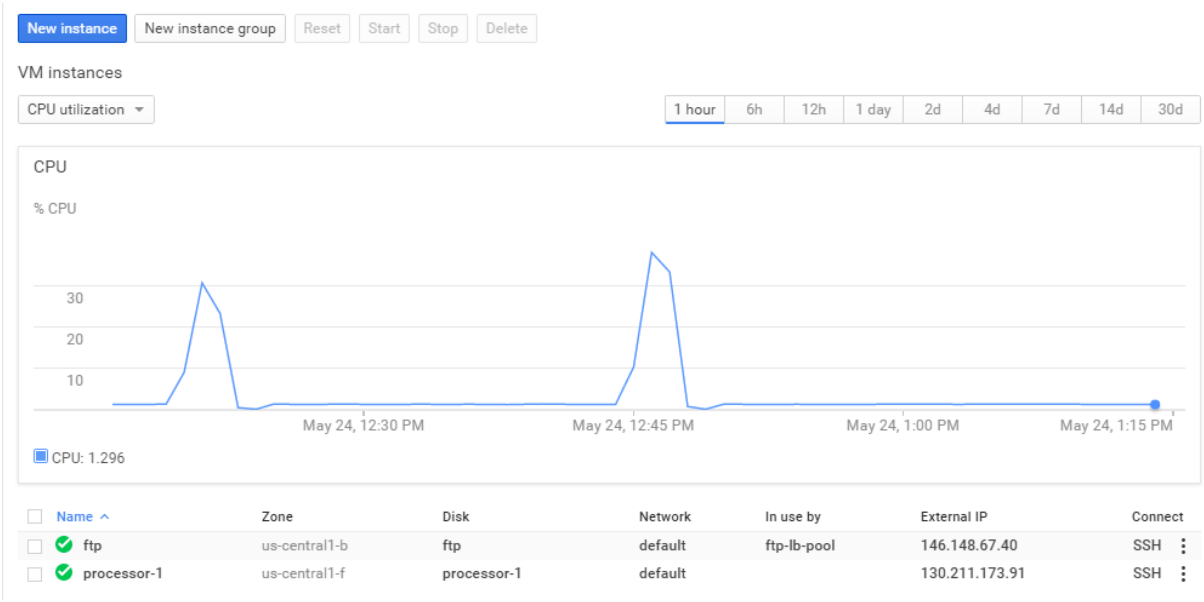

Рисунок 7.6 – Інтерфейс Compute Engine

Замовлення потрапляє в систему і його можна побачити у стані In Progress, що продемонстровано на рисунку 7.7.

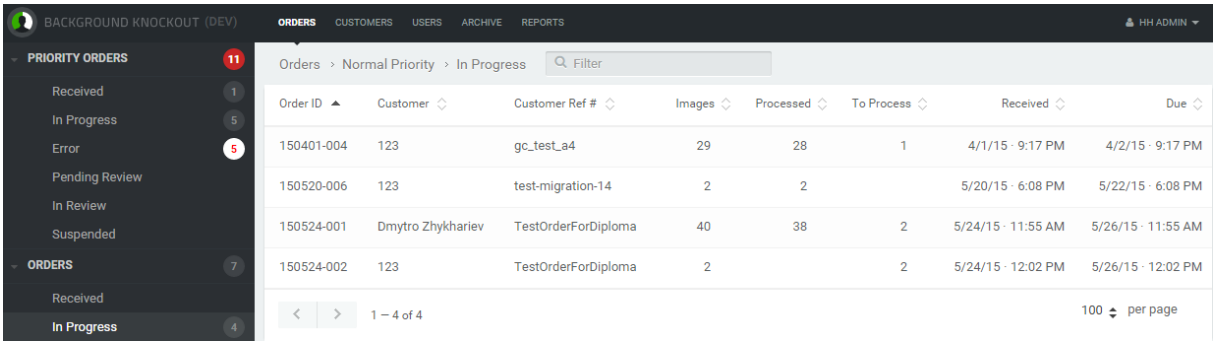

Рисунок 7.7 – Екран програми зі списком замовлень «In Progress»

Після того, як всі завдання з Task Queue оброблені в Compute Engine, замовлення може бути переглянуте оператором. Можливість перегляду замовлення у статусі «Review» зображена на рисунку 7.8.

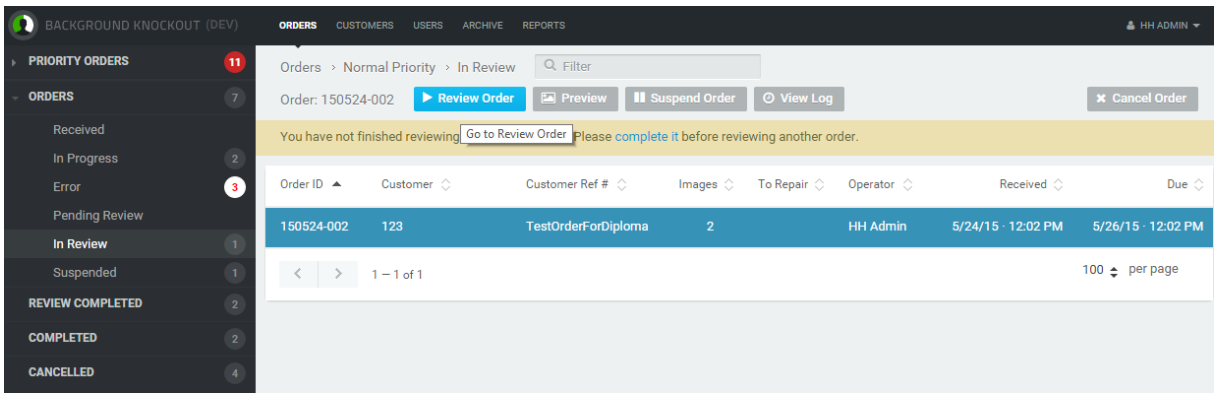

Рисунок 7.8 – Екран програми з можливістю почати «review» замовлення

«Review» замовлення є екраном, на якому користувач може вказати, чи добре зображення або вимагає ремонту. Такий екран зображено рисунку 7.9.

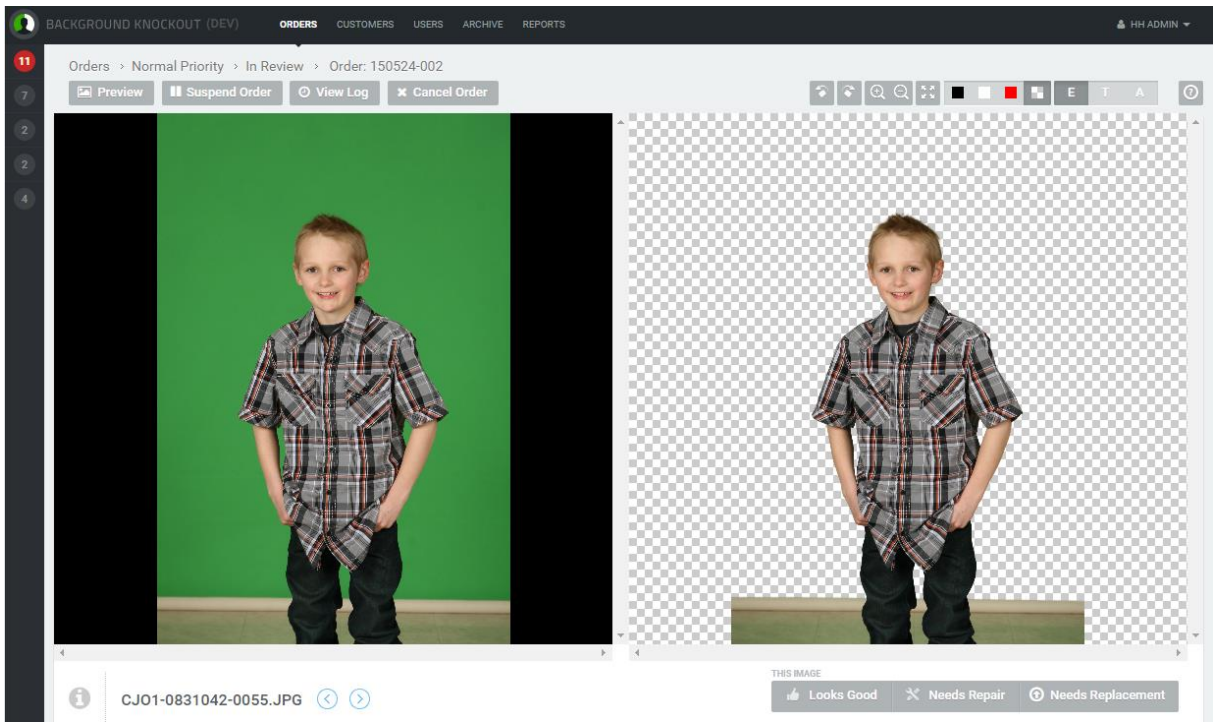

Рисунок 7.9 – Екран «Review»

Якщо під час перегляду замовлення було відзначено зображення, яким необхідний ремонт, після закінчення перегляду оператор потрапляє на

екран «Repair», який представлений рисунку 7.10. На цьому екрані оператор може підказати підказку алгоритму, що прибирає фон з фотографії, за допомогою різних інструментів.

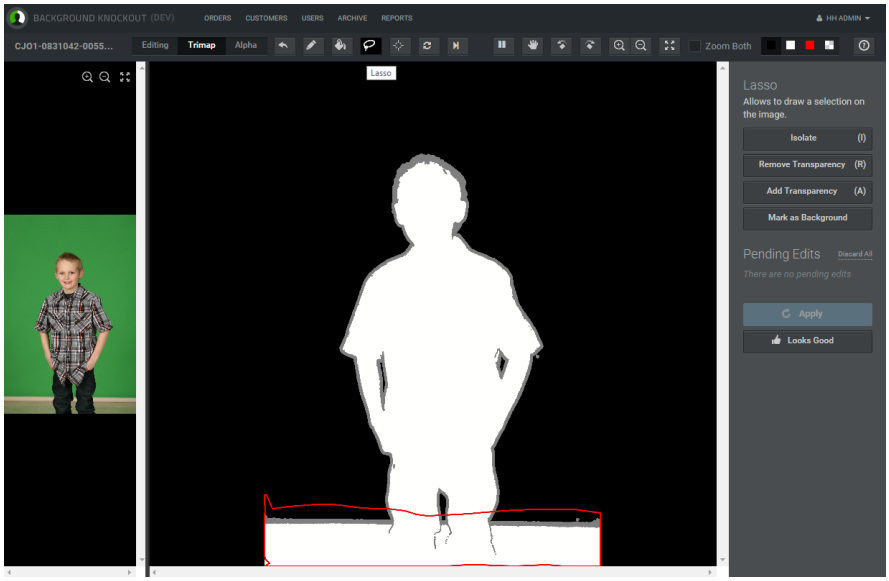

Рисунок 7.10 – Екран «Repair»

Коли всі зображення замовлення були позначені оператором як хороші, замовлення переходить у стан «Completed» (рисунок 7.11). У цьому статусі зображення копіюються в результуючу папку Google Cloud Storage, що зображено на рисунку 7.12.

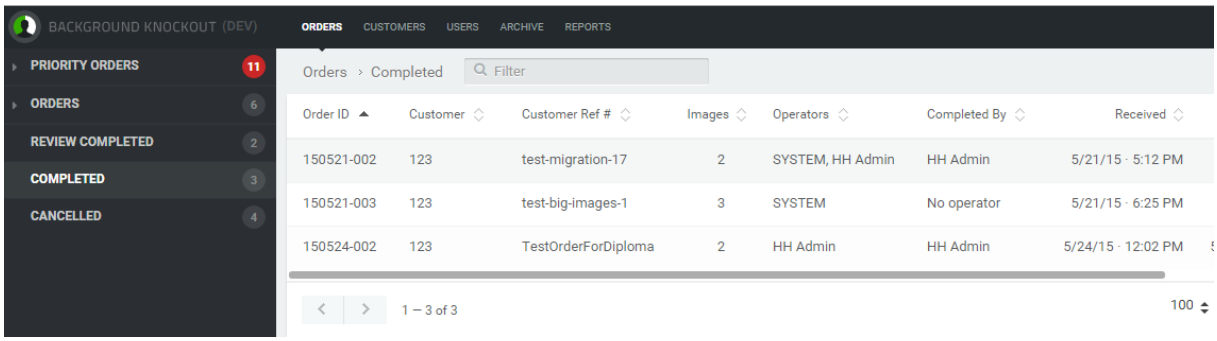

Рисунок 7.11 – Екран замовлень у статусі «Completed»

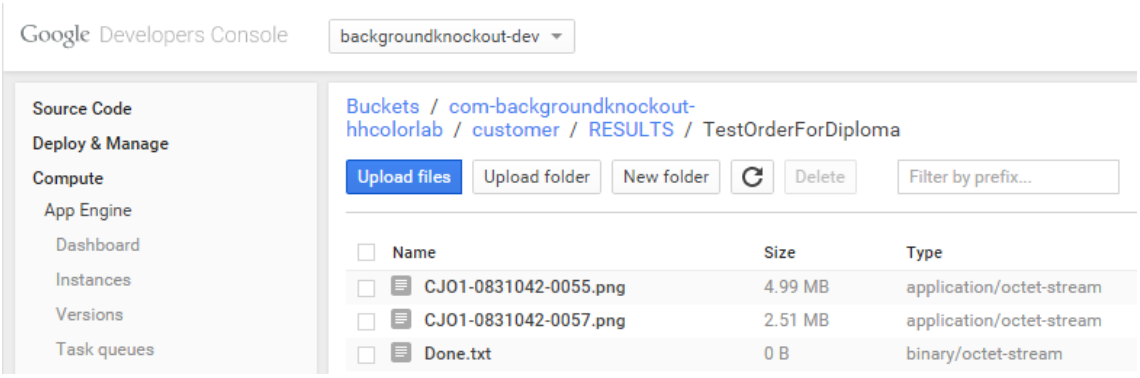

Рисунок 7.12 – Екран Google Cloud Storage із зображеннями результатів

Зображення має бути фотографією людей на однорідному тлі. Вона зберігає в собі інформацію про те, якому замовленню належить, шляхи до допоміжних файлів, які генеруються при обробці зображення та можуть бути використані для її покращення.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи докладно вивчено поняття бізнес-логіки програми, можливості гнучкої адаптованості програми до змінних бізнес-параметрів. Також було розглянуто питання розгорнення програми на хмарній платформі з метою спрощення розробки та збільшення її ефективності.

В роботі досліджувалося поняття бізнес-логіки у всіх шарах програми, від базових концепцій проекту до високорівневих архітектурних патернів. У роботі були вказані деякі техніки програмування та програмні інструменти для успішного відокремлення бізнес-логіки від її реалізації та технічних деталей. Більш детально було описано спосіб відокремлення бізнес-логіки за допомогою системи управління бізнес-правилами, а також показано відмінність такого підходу від стандартної розробки з використанням патерна програмування MVC.

Постановка практичного завдання включала розробку web-додатка з бізнес-логікою замовлень з обробки картинок на одній з хмарних платформ. Перед тим як вибрати хмарну платформу, що найбільш підходить для програми, зроблено твердження щодо гнучкості та економічної ефективності хмарних обчислень у порівнянні з локальним управлінням обчислювальними ресурсами. У роботі було проведено порівняльний аналіз найбільш поширених хмарних платформ та зроблено рекомендації щодо вибору хмарної платформи залежно від типу програми.

У роботі докладніше вивчений PaaS сервіс, оскільки саме цей шар хмарних технологій дозволяє набагато підвищити якість та швидкість реалізації програми для її розробників. Як така платформа була обрана Google App Engine. Наводяться характеристики даної платформи, які дозволяють віддати перевагу перед іншими подібними сервісами як Amazon EC 2. У роботі детально розглянута архітектура App Engine, а також сервіси платформи, які пропонуються розробникам і роблять розробку додатків

легшою, ефективнішою та надійнішою.

У практичній частині роботи описано деякі головні моменти розробки під платформу App Engine, наведено приклади використання сервісів та інструментів платформи для розробників, описано головні сутності доменної моделі розробленого додатка.

На основі теоретичних даних, отриманих при докладному аналізі хмарної платформи, реалізовано web-додаток для роботи із зображеннями. Розгорнутий на App Engine додаток надає можливість користувачам організовувати зображення на замовлення. Зображення є фотографіями та підлягають видаленню з них фону. Замовлення обробляються частиною платформи App Engine, яка називається Compute Engine, і є IaaS-подібним сервісом. Завдяки даній можливості алгоритм обробки картинок задіює віртуальні машини обраної потужності з необхідною кількістю оперативної пам'яті і ресурсами процесора. Замовлення відображаються на веб-інтерфейсі у певних статусах. Користувачі мають можливість переглядати фотографії у своїх замовленнях та скористатися інструментами для коригування фотографій, які система обробила недостатньо добре.

Перехід замовлень із статусу до статусу супроводжується різними діями, що містять у собі складну бізнес-логіку. У ході роботи було наведено рекомендації, як винести таку логіку та зробити її керованою, з можливістю швидкої зміни.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Учасники проектів Вікімедіа. Бізнес-логіка – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Бізнес-логіка (дата звернення: 15.05.2024).

2. How to use Model-View-Controller. URL: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html (дата звернення: 15.05.2024).

3. Framework для автоматичної генерації електронних ресурсів URL: http://www.ti5.tu-harburg.de/Staff/turau /pubs/sac02.pdf (дата звернення: 15.05.2024).

4. Pro Apache Struts with AJAX. URL: http://www.apress.com/book /view/1590597389 (дата звернення: 15.05.2024).

5. Agent-Oriented Enterprise Modeling Based on Business Rules? URL: http://dl.acm.org/citation.cfm?id=728342 (дата звернення: 15.05.2024).

6. Business Rules Integration in BPEL – a Services Oriented Approach URL: http://www.infosys.tuwien.ac.at/staff/sd/papers/TUV-1841-2005-30.pdf (дата звернення: 15.05.2024).

7. Business rules manipulation model. URL: http//itc.ktu.lt/itc363/Motiejun363.pdf (дата звернення: 15.05.2024).

8. Performance Analysis and Capacity Planning. URL: http://www.ilog.com/brms/media/jrules_cap_wp.pdf (дата звернення: 15.05.2024).

9. Business rules. URL: http://en.wikipedia.org/wiki/Business_Rules (дата звернення: 15.05.2024).

10. Web Services Activity. URL: http://www.w3.org/2002/ws/ (дата звернення: 15.05.2024).

11. Software as a Service. URL: http://www.wikinvest.com/concept/Software_as_a_Service (дата звернення: 15.05.2024).

12. PaaS primer: What is platform as service and why does it matter?

URL: http://www.infoworld.com/article/2613027/paas/paas-primer--what-is-platform-as-a-service-and-why-does-it-matter -.html (дата звернення: 15.05.2024).

13. What is IaaS? URL: http://www.interoute.com/what-iaas (дата звернення: 15.05.2024).

14. Google App Engine Framework. URL: http://code.google.com/intl/en-EN/appengine/ (дата звернення: 15.05.2024)..

15. The Google File System. URL: http://static.googleusercontent.com/media /research.google.com/en//archive/gfs-sosp2003.pdf (дата звернення: 15.05.2024).

16. Bigtable: A Distributed Storage System for Structured Data. URL: http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf (дата звернення: 15.05.2024).

17. Amazon Elastic Compute Cloud (Amazon EC2). URL: http://aws.amazon.com/en/ec2/ (дата звернення: 15.05.2024)..

18. Amazon EBS. URL: http://aws.amazon.com/en/ebs/ (дата звернення: 15.05.2024).

19. Dynamo: Amazon's Highly Available Key-value Store. URL: http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf (дата звернення: 15.05.2024).

20. App Engine Developers Guide. URL: http://code.google.com/intl/en-EN/appengine/docs/ (дата звернення: 15.05.2024).