МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2023 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

АВТОМАТИЗОВАНА СИСТЕМА ФОРМУВАННЯ ОБРАЗУ БАЖАНОГО МАЙБУТТЯ

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.1910214

Виконав студент 4-го курсу, групи 402
_____ А. К. Лі-Злотін
«___» _____ 2023_ р
Керівник: д-р пед. наук, проф.
_____ О. П. Мещанінов
«___» _____ 2023_ р

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Лі-Злотіну Анатолію Кімовичу.

1. Тема кваліфікаційної роботи «Автоматизована система формування образу бажаного майбуття».

Керівник роботи Мещанінов Олександр Павлович, д-р пед. наук, професор.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «___» _____ 20___ р. № ____

2. Строк представлення кваліфікаційної роботи студентом «___» ____ 20__ р.

3. Вхідні дані до роботи: вимоги до функціональності, користувацького інтерфейсу та безпеки веб-застосунку автоматизованої системи формування образу бажаного майбуття.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

– аналіз актуальності застосунку для формування образу бажаного майбуття; – огляд існуючих технологій та методологій для розробки веб-застосунків;

– розробка веб-застосунку для формування образу бажаного майбуття для профорієнтація з використанням технології Java Spring;

– проведення тестування розробленого веб-застосунку, включаючи його адміністративну панель, функціонал користувача;

– оцінка роботи веб-застосунку та визначення потенційних напрямків для подальшого розвитку та оптимізації.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Охорона праці при роботі з екранними пристроями».

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Боженко А. Л. викладач	

Керівник роботи_____д-р пед. наук, проф. Мещанінов О.П.
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання _____Лі-Злотін А.К.
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «__» _____ ____ 20__ р.

КАЛЕНДАРНИЙ ПЛАН
виконання бакалаврської кваліфікаційної роботи

Тема: Автоматизована система формування образу бажаного майбуття

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівника БКР	27.10.2022	30.10.2022	Виконано
2	Отримання завдання на виконання БКР	08.11.2022	10.11.2022	Виконано
3	Складання календарного плану роботи на весь період виконання БКР	01.12.2022	01.12.2022	Виконано
4	Отримання завдання на переддипломну практику	12.03.2023	12.03.202	Виконано
5	Аналіз ринку, визначення основних потреб користувачів та розробка вимог	15.03.2023	25.03.2023	Виконано
6	Вивчення та вибір технологій для розробки веб-застосунку	30.03.2023	11.04.2023	Виконано
7	Розробка архітектури, планування та дизайн веб-застосунку	12.04.2023	21.04.2023	Виконано
8	Розробка фронтенду та бекенду, налаштування бази даних і інтеграція її	25.04.2023	19.05.2023	Виконано
9	Фінальне злиття і налаштування коду веб-застосунку	22.05.2023	26.05.2023	Виконано
10	Розробка тестових сценаріїв і тестування веб-застосунку	27.05.2023	28.05.2023	Виконано
11	Попередній захист БКР на засіданні комісії кафедри	29.05.2023	30.05.2023	Виконано
12	Доробка та остаточне оформлення БКР	02.06.2023	08.06.2023	Виконано
13	Подання БКР рецензенту	13.06.2023	16.06.2023	Виконано
14	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	20.06.2023	22.06.2023	Виконано
15	Захист БКР перед екзаменаційною комісією (ЕК)	26.06.2023	29.06.2023	Виконано

Розробив студент __Лі-Злотін А. К. _____
(прізвище та ініціали) *(підпис)*

Керівник _____д-р пед. наук Мещанінов О. П.___ _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 202__ р.

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили

Лі-Злотіна Анатолія Кімовича

Тема: «Автоматизована система формування образу бажаного майбуття»

Дана робота являє собою розробку системи профорієнтації, заснованої на інтерактивній взаємодії між адміністратором і користувачами сайту. Адміністратор має повні права на редагування контенту, спеціалізацій, запитань і рекомендацій, а користувачі можуть обирати бажану спеціалізацію і відповідати на запитання, щоб отримати персоналізовані рекомендації.

Метою даної роботи є створення ефективної системи профорієнтації, яка допоможе користувачам визначити найбільш підходящі для них професійні напрямки та надасть їм конкретні рекомендації на основі їхніх уподобань і параметрів.

Актуальність теми полягає в необхідності допомоги людям у виборі професійного шляху в умовах різноманітності спеціалізацій і можливостей. Система профорієнтації дає змогу об'єктивно оцінити інтереси, знання і бажання користувача та надати йому цінні рекомендації, скорочуючи часові та інформаційні витрати на пошук відповідних професій.

Об'єктом роботи є процес профорієнтації, що ґрунтується на взаємодії адміністратора і користувачів сайту, а предметом роботи є розробка і реалізація системи профорієнтації з використанням прихованих інтерфейсів для редагування контенту, спеціалізацій, запитань і рекомендацій.

Результатом цієї роботи буде функціональна система, здатна надати персоналізовані рекомендації користувачеві на основі його відповідей на запитання, які обирає та редагує адміністратор. Це допоможе користувачам ухвалити поінформоване рішення щодо свого професійного шляху та підвищить ефективність процесу профорієнтації.

Метою цієї програми є надання користувачеві персоналізованих рекомендацій щодо вибору професійного шляху на основі їхніх вподобань, навичок і параметрів. Програма має допомогти користувачам прийняти поінформоване рішення щодо своєї професійної орієнтації та полегшити процес вибору професії.

Завдання для досягнення мети:

Розробка системи адміністрування: необхідно створити інтерфейс, через який адміністратор зможе додавати та редагувати спеціалізації, запитання та рекомендації. Це дасть змогу адміністратору актуалізувати інформацію й адаптувати її під мінливі вимоги та реалії ринку праці.

Створення призначеного для користувача інтерфейсу: потрібно розробити інтерфейс для користувачів, через який вони зможуть вибрати бажану спеціалізацію і відповісти на запитання, запропоновані системою. Інтерфейс має бути інтуїтивно зрозумілим і зручним для використання.

Розробка алгоритму збору даних: необхідно визначити, які параметри і питання будуть враховані для формування персоналізованих рекомендацій. Алгоритм має ефективно збирати дані від користувачів і використовувати їх для подальшого аналізу та генерації рекомендацій.

Реалізація алгоритму формування рекомендацій: потрібно розробити алгоритм, який аналізуватиме дані користувача і на основі попередньо заданих рекомендацій адміністратора формуватиме список рекомендацій, що відповідають інтересам і вимогам користувача.

Тестування та оптимізація: після реалізації системи необхідно провести тестування, щоб переконатися в її коректній роботі та ефективності. У процесі тестування можуть бути виявлені помилки або поліпшення, які вимагають внесення відповідних змін.

Розвиток і підтримка: у міру розвитку програми може знадобитися внесення змін і оновлень, щоб система залишалася актуальною і відповідала вимогам користувачів. Також потрібно забезпечити підтримку користувачів і розв'язання проблем, що виникають. Робота складається з теоретичного огляду, частини

розробки, тестування, аналізу та спеціальної частини з охорони праці. Пояснювальна записка включає вступ, чотири розділи та висновки.

У першому розділі проводиться аналіз предметної області і сутності інформаційної системи. У другому розділі представлений огляд яким чином буде здійснено проєктування системи. Третій розділ описує процес розробки веб-застосунку на основі технології Java Spring. Спеціальний розділ присвячено питанням охорони праці під час розробки та використання веб-застосунку.

Бакалаврська кваліфікаційна робота містить 81 сторінку, 9 рисунків, 30 використаних джерел, 1 додаток, 1 таблиця.

Ключові слова: *веб-застосунок, бажане майбуття, профорієнтація.*

ABSTRACT

for bachelor's qualification work of a student of 402 group at Petro Mohyla Black Sea National University

Li-Zlotin Anatolii Kymovych

Theme: «Automated system of forming the image of the desired future»

This work is the development of a career guidance system based on interactive interaction between the administrator and users of the site. The administrator has full rights to edit content, specializations, questions and recommendations, and users can choose the desired specialization and answer questions to get personalized recommendations.

The aim of this work is to create an effective career guidance system that will help users identify the most suitable professional areas for them and provide them with specific recommendations based on their preferences and parameters.

The relevance of the topic lies in the need to help people choose a professional path in the face of a variety of specializations and opportunities. The career guidance system allows for an objective assessment of the user's interests, knowledge and desires and provides valuable recommendations, reducing the time and information costs of finding suitable professions.

The object of the work is the process of career guidance based on the interaction between the administrator and users of the site, and the subject of the work is the development and implementation of a career guidance system using hidden interfaces for editing content, specializations, questions and recommendations.

The result of this work will be a functional system capable of providing personalized recommendations to users based on their answers to questions selected and edited by the administrator. This will help users make an informed decision about their career path and increase the efficiency of the career guidance process.

The purpose of this program is to provide the user with personalized recommendations for choosing a career path based on their preferences, skills, and

parameters. The program should help users make an informed decision about their career guidance and facilitate the process of choosing a profession.

Tasks to achieve the goal:

Development of an administration system: an interface should be created through which the administrator can add and edit specializations, questions, and recommendations. This will allow the administrator to update the information and adapt it to the changing requirements and realities of the labor market.

Create a user interface: you need to develop an interface for users to select the desired specialization and answer the questions offered by the system. The interface should be intuitive and easy to use.

Development of a data collection algorithm: it is necessary to determine which parameters and questions will be taken into account to generate personalized recommendations. The algorithm should efficiently collect data from users and use it for further analysis and recommendation generation.

Implementation of the recommendation algorithm: you need to develop an algorithm that will analyze user data and, based on the administrator's predefined recommendations, generate a list of recommendations that meet the user's interests and requirements.

Testing and optimization: after the system is implemented, it is necessary to conduct testing to make sure that it works correctly and efficiently. The testing process may reveal errors or improvements that require appropriate changes.

Development and support: as the program evolves, changes and updates may need to be made to keep the system relevant and meet user requirements. It is also necessary to provide support for users and solve problems that arise. The work consists of a theoretical overview, a development part, testing, analysis and a special part on occupational safety and health. The explanatory note includes an introduction, four chapters and conclusions.

The first section analyzes the subject area and the essence of the information system. The second section provides an overview of how the system will be designed. The third section describes the process of developing a web application based on Java

Spring technology. A special section is devoted to the issues of labor protection during the development and use of a web application.

The bachelor's qualification work contains 81 pages, 9 figures, 30 references, 1 appendix, 1 table.

Key words: *web application, desired future, career guidance*.

ЗМІСТ

ВСТУП

У сучасному світі, де інформація швидко змінюється і вирішення проблем вимагає комплексного підходу, виникає потреба в автоматизованих системах, що спрощують процес прийняття рішень та надають швидкий доступ до актуальної інформації. Однією з таких систем є система формування образу бажаного майбутнього.

Актуальність розробки автоматизованої системи формування образу бажаного майбутнього полягає в потребі сучасного світу у швидкому доступі до актуальної інформації та оптимізації процесу прийняття рішень у різних сферах діяльності. Ця система має потенціал покращити ефективність робочих процесів, забезпечити швидкий доступ до необхідних відповідей та сприяти прийняттю обґрунтованих рішень.

Мета цієї дипломної роботи полягає в тому, щоб розробити автоматизовану систему формування бажаного майбутнього, яка може вирішувати поширені питання в різних сферах роботи. Система дозволяє адміністратору вносити інформацію та налаштовувати її відповідно до потреб користувачів. Вона допомагає усунути необхідність вручну шукати відповіді на питання та забезпечує швидкий та точний доступ до актуальної інформації.

Предметом роботи є розробка та реалізація автоматизованої системи формування образу бажаного майбутнього. Об'єктом роботи є процес формування образу бажаного майбутнього та надання відповідей на питання в різних областях роботи.

Застосування системи може бути широким: від бізнесу та управління проєктами до особистого використання. Вона може бути цінним інструментом для підвищення продуктивності, сприяти поліпшенню прийняття рішень та оптимізації робочих процесів. Важливою особливістю системи є її гнучкість та адаптованість

до різних сфер діяльності, завдяки можливості внесення та налаштування інформації адміністратором.

Розробка такої системи має великий потенціал у поліпшенні якісного та швидкого доступу до інформації, що сприятиме підвищенню ефективності робочих процесів та прийняттю обґрунтованих рішень. Дана робота може бути використана як основа для подальших досліджень та розвитку подібних систем з метою оптимізації різних видів діяльності.

У процесі розробки системи будуть використовуватися сучасні технології програмування та бази даних для забезпечення швидкості та надійності системи. Буде створено зручний інтерфейс користувача, який дозволяє з легкістю знаходити відповіді на питання в різних областях, таких як фінанси, маркетинг, управління проєктами та інші.

Основними завданнями дипломної роботи є:

– аналіз потреб користувачів та визначення основних питань, на які система повинна надавати відповіді;

– розробка архітектури системи та вибір необхідних технологій;

– реалізація системи формування образу бажаного майбутнього з можливістю внесення та налаштування інформації адміністратором;

– тестування та оцінка ефективності системи з використанням реальних даних та сценаріїв використання;

– висновки та рекомендації щодо подальшого розвитку та використання системи формування образу бажаного майбутнього.

Результати даної дипломної роботи сприятимуть поліпшенню процесу прийняття рішень та надаватимуть користувачам зручний інструмент для швидкого доступу до актуальної інформації в різних областях роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Прикладні програми

Прикладне програмне забезпечення (ППЗ) є спеціальним видом комп'ютерних програм, призначених для виконання конкретних завдань. Вони забезпечують функціональні можливості для задоволення особистих, бізнесових та освітніх потреб користувачів. Таке програмне забезпечення часто використовується як інструмент для підвищення продуктивності. Кожна програма розробляється з метою допомогти користувачам виконувати певні процеси, пов'язані з продуктивністю, ефективністю та комунікацією. У відмінність від системного програмного забезпечення, ППЗ має специфічну функціональність і виконує завдання, для яких воно призначене. Наприклад, більшість програм, що використовуються на смартфонах, є прикладним програмним забезпеченням.

Ролі та функції прикладного програмного забезпечення

Залежно від потреб користувача програмне забезпечення може виконувати різні ролі та функції. Серед найпоширеніших функцій:

– дозволяє користувачам зберігати, оновлювати та обробляти дані в зручний спосіб;

– допомагає організовувати, зберігати та отримувати доступ до інформації швидко та ефективно;

– надає можливість виконувати різні обчислення і операції для досягнення певних результатів;

– дозволяє створювати графічні представлення даних та інформації для зручного сприйняття користувачем;

– допомагає керувати та координувати використання ресурсів, таких як час, гроші або матеріали, для досягнення певних цілей;

– дозволяє генерувати звіти і документи з оброблених даних для подальшого використання або аналізу.

Робота з електронними таблицями: ППЗ надає засоби для створення, редагування та аналізу електронних таблиць з числовими даними та формулами.

Маніпулювання зображеннями: ППЗ дозволяє обробляти, редагувати та маніпулювати зображеннями для досягнення певних візуальних ефектів.

Ведення документації: ППЗ надає засоби для створення, оформлення та організації документів та текстової інформації.

Розробка веб-сайтів: ППЗ допомагає веб-розробникам створювати, редагувати та публікувати веб-сайти з використанням різноманітних інструментів і технологій.

Розрахунок витрат: ППЗ надає засоби для розрахунку, аналізу та контролю витрат в рамках бізнесової або фінансової діяльності.

Існує багато типів прикладного програмного забезпечення, які можуть використовуватися в залежності від ваших потреб. Ось кілька прикладів різних типів програмного забезпечення:

- графічні редактори: програми, які дозволяють створювати та редагувати графіку і зображення;
- комунікаційні засоби: програми для спілкування через Інтернет, такі як месенджери та програми для відеозв'язку;
- фінансове програмне забезпечення: програми для ведення бухгалтерії, управління фінансами та розрахунків;
- освітнє програмне забезпечення: програми для навчання та освіти, включаючи інтерактивні підручники та мовні курси;
- медіа-плеєри: програми для відтворення аудіо та відеофайлів;
- мобільні програми: програми, призначені для використання на смартфонах і планшетах;

– системи управління відносинами з клієнтами (crm): програми для організації та керування взаємодією з клієнтами;

– текстові редактори: програми для створення та редагування текстових документів;

– програмне забезпечення для управління проєктами: програми для планування, виконання та контролю проєктів;

– електронна пошта та календарі: програми для управління електронною поштою та організації робочого графіка.

Це лише деякі з багатьох типів прикладного програмного забезпечення, доступних для різних цілей та потреб користувачів.

Програмне забезпечення для обробки тексту використовується з метою форматування, оформлення та опрацювання текстової інформації. Воно має можливість використовувати синоніми та антоніми для словесного виразу. За допомогою функції Word Art можна змінювати шрифти, кольори та стиль тексту відповідно до власних уподобань. Крім того, таке програмне забезпечення надає можливість перевірки помилок, граматики та орфографії. Microsoft Word є відмінним прикладом програмного забезпечення для обробки тексту.

Електронні таблиці в основному використовуються для організації та аналізу даних за допомогою таблиць і виконання широкого спектру обчислень. Ці програмні інструкції складаються з клітинок, які мають здатність зберігати числа, дати, час і текст. Користувачі можуть виконувати складні обчислення за допомогою формул і функцій. Microsoft Excel є яскравим прикладом комплексного програмного забезпечення для роботи з електронними таблицями.

Програмне забезпечення для створення презентацій дозволяє користувачам передавати свої думки та ідеї у візуально привабливій формі за допомогою презентацій. Цей тип програмного забезпечення полегшує створення інтерактивних та інформативних слайдів шляхом включення таких елементів, як відео, текст, діаграми, графіки та зображення. Microsoft PowerPoint слугує

яскравою ілюстрацією надійного програмного забезпечення для створення презентацій.

Мультимедійне програмне забезпечення відіграє ключову роль у створенні та маніпулюванні відео, аудіо та зображеннями. Його функціональні можливості охоплюють редагування відео, графічний дизайн та анімацію. До популярних прикладів мультимедійного програмного забезпечення належать VLC, MX Player та Windows Media Player.

Браузери - це незамінні програми, що використовуються для навігації в безмежному просторі Інтернету. Вони надають користувачам можливість шукати інформацію та отримувати дані з онлайн-джерел. Chrome і Firefox стали одними з найпоширеніших веб-браузерів.

Освітнє програмне забезпечення, також відоме як академічне програмне забезпечення, спеціально розроблене для полегшення процесу навчання з широкого спектру предметів і дисциплін. Воно охоплює широкий спектр додатків, пристосованих для освітніх цілей. Яскравими прикладами освітнього програмного забезпечення є EDX, MindPlay та Kid Pix.

Графічне програмне забезпечення дозволяє користувачам маніпулювати та створювати візуальні ефекти, зображення та анімацію. Ці програми включають різні інструменти для графічного редагування та дизайну. Adobe Photoshop, Unity 3D та PaintShop є одними з найвідоміших прикладів графічного програмного забезпечення.

Безкоштовне програмне забезпечення - це програмне забезпечення, яке можна завантажити та встановити безкоштовно. Хоча користувачі можуть використовувати безплатне програмне забезпечення, вони, як правило, не мають дозволу на зміну вихідного коду. Skype є прикладом вільно поширюваного програмного забезпечення.

Умовно-безкоштовні програми розповсюджуються серед користувачів протягом пробного періоду, що дозволяє їм оцінити функціональність і можливості

програми. Після закінчення пробного періоду користувачі, які бажають продовжувати користуватися програмою, як правило, повинні придбати ліцензію. WinZip є яскравим прикладом умовно-безкоштовного програмного забезпечення.

Програмне забезпечення для моделювання дозволяє користувачам спостерігати та аналізувати операції, не виконуючи їх фізично. Воно виявляється особливо цінним, коли має справу з системами, які є неточними, непередбачуваними або потенційно небезпечними. Програмне забезпечення для моделювання знаходить широке застосування в інженерії, робототехніці, авіаційних системах, прогнозуванні погоди, тестуванні, освіті та відеоіграх. MATLAB є яскравим прикладом програмного забезпечення для моделювання, яке широко використовується для імітаційних цілей.

Програмне забезпечення з відкритим вихідним кодом (open source) має відкритий вихідний код і дозволяє користувачам перевіряти, змінювати та покращувати його. Більшість програм з відкритим вихідним кодом доступна безкоштовно, хоча є й платні варіанти.

Антиподом програмного забезпечення із відкритим кодом є програмне забезпечення з закритим кодом. Його купують, і споживачі не мають доступу до його вихідного коду. Зазвичай власник програмного забезпечення накладає на нього обмеження та права інтелектуальної власності.

Бізнес-прикладне програмне забезпечення є підмножиною прикладного програмного забезпечення, яке організації використовують для виконання бізнес-цілей. Це програмне забезпечення спеціально розроблено для полегшення певних бізнес-функцій. Воно допомагає підвищити продуктивність, ефективність, точність та забезпечує регулярну звітність для бізнес-аналізу. Програмне забезпечення для бізнесу використовується багатьма сучасними компаніями, які швидко розвиваються.

Програма, відома як управління взаємовідносинами з клієнтами (CRM), регулює, як компанія взаємодіє з клієнтами, як існуючими, так і потенційними.

Великі обсяги даних про клієнтів можна збирати, вивчати та керувати ними, щоб сприяти розвитку бізнесу.

Планування ресурсів підприємства (ERP) є програмним забезпеченням та системою, яка керує всіма основними аспектами діяльності та бізнес-процесами організації. Воно автоматизує та спрощує такі діяльності, як закупівлі, бухгалтерський облік, управління проєктами, управління ризиками та ланцюгом поставок.

Програмне забезпечення для керування проєктами використовується для планування, керування змінами, розподілу ресурсів та виконання проєктів. Воно допомагає керувати бюджетом і витратами, документувати прогрес проєкту, звітувати про результати та призначати завдання.

База даних - це структурована колекція даних, яка зберігається та управляється за допомогою програмного забезпечення бази даних (СУБД). СУБД дозволяє створювати, зберігати, оновлювати та використовувати дані, забезпечуючи швидкий доступ до них та забезпечуючи цілісність та безпеку даних.

Управління бізнес-процесами (Business Process Management, BPM) - це підхід до управління організацією, який використовує програмне забезпечення для автоматизації, аналізу та вдосконалення бізнес-процесів. BPM дозволяє компаніям моделювати, оптимізувати та автоматизувати свої бізнес-процеси з метою підвищення ефективності, зниження витрат та покращення якості.

Програмне забезпечення для керування ресурсами (Resource Management Software) - це тип програмного забезпечення, яке допомагає планувати, використовувати та відстежувати ресурси (такі як персонал, матеріали, обладнання) для успішного виконання проєктів. Воно допомагає забезпечити ефективне використання ресурсів, оптимізувати планування та виконання проєктів у режимі реального часу.

Навчальне програмне забезпечення (Educational Software) - це програмне забезпечення, спеціально розроблене для освітніх цілей. Воно може включати

інтерактивні навчальні матеріали, електронні підручники, вправи та інші інструменти для полегшення навчання та викладання.

Програмне забезпечення для продуктивності (Productivity Software) - це набір програм, які допомагають користувачам збільшити їх продуктивність та ефективність у роботі. Це можуть бути програми для створення документів, управління проєктами, спільної роботи над файлами, організації робочого часу тощо. Вони дозволяють зосередитися на виконанні завдань і покращити робочий процес.

Програмне забезпечення, розроблене на замовлення (Custom Software) - це прикладне програмне забезпечення, створене на замовлення для конкретної організації або користувача з метою задоволення їхніх унікальних потреб і вимог. Це програмне забезпечення розробляється спеціально під конкретний бізнес-процес або завдання і може бути повністю налаштованим під вимоги та потреби замовника.

1.2 Поняття і сутність інформаційної системи

Інформаційна система (ІС) представляє собою формальну, соціально-технічну та організаційну структуру, призначену для збору, обробки, зберігання та поширення інформації. З перспективи соціальних технологій, ІС складається з чотирьох основних компонентів: завдань, персоналу, структури (або ролей) і технологій. Інформаційна система може бути визначена як інтеграція компонентів, що використовуються для збору, зберігання та обробки даних, які використовуються для надання інформації, просування знань і цифрових продуктів.

Комп'ютерна інформаційна система є системою, що об'єднує людей і комп'ютери для обробки або інтерпретації інформації. Іноді цей термін

використовується для простого опису комп'ютерної системи, на якій встановлено відповідне програмне забезпечення.

Інформаційна система є предметом академічного вивчення, включаючи конкретну базу знань, довідкову інформацію, а також додаткову мережу апаратного та програмного забезпечення, які використовуються людьми та організаціями для збору, фільтрації, обробки, створення та поширення даних. У центрі уваги знаходяться інформаційні системи з чітко визначеними межами, користувачами, процесорами, бібліотеками зберігання, входами, виходами та комунікаційними мережами, які були згадані раніше.

Багато організацій мають відділи або підрозділи, відповідальні за інформаційні системи та обробку даних, які називаються "інформаційними службами".

Кожна конкретна інформаційна система призначена для підтримки операцій, управління та прийняття рішень. Інформаційна система включає в себе інформаційно-комунікаційні технології (ІКТ), які використовуються організацією, а також способи взаємодії людей з цими технологіями для підтримки бізнес-процесів.

Деякі автори проводять межу між бізнес-процесами, комп'ютерними системами та інформаційними системами. ІКТ-компоненти складають більшість інформаційних систем, але вони зосереджуються не лише на ІКТ, а й на тому, як інформаційні технології будуть використовуватися в майбутньому. Хоча інформаційні системи та бізнес-процеси відрізняються один від одного, вони обидва сприяють управлінню ефективністю останніх.

Переваги розгляду інформаційних систем як особливого виду системи праці були проілюстровані компанією Alter. Функціонуюча система використовується для здійснення процесів і діяльності з виробництва певних товарів або послуг для клієнтів, незалежно від того, чи керують нею люди, чи машини.

Робоча система, яка збирає, передає, зберігає, витягує, обробляє та відображає інформацію, відома як інформаційна система.

Таким чином, інформаційні системи взаємодіють з системами даних і бізнес-системами. Вони можуть бути розглянуті як комунікаційні системи, де дані представлені і обробляються у формі соціальної пам'яті. Інформаційні системи також можна розглядати як напівформальну мову, яка підтримує прийняття людських рішень та дій.

Дослідження організаційної інформатики зосереджені на вивченні інформаційних систем. Сільвер (1995 рік) запропонував два погляди на інтелектуальну власність, включаючи програмне забезпечення, апаратне забезпечення, дані, персонал і процедури. Чжен вніс різноманітні системні уявлення про інформаційну систему, додавши процес і ключові елементи, такі як середовище, межі, ціль та взаємодія.

Згідно з визначенням, наданим Комп'ютерною асоціацією, фахівці в галузі інформаційних систем зосереджуються на інтеграції рішень інформаційних технологій з бізнес процесами для задоволення інформаційних потреб підприємств та інших організацій.

Системи обробки транзакцій, системи підтримки прийняття рішень, системи управління знаннями, системи управління навчанням, системи управління базами даних, а також офісні інформаційні системи - це лише частина прикладів різних типів інформаційних систем. Кожна категорія виконує певну функцію і допомагає організації працювати як єдине ціле. Ключовим компонентом більшості інформаційних систем є інформаційні технології, які дозволяють виконувати завдання, які перевищують можливості людського мозку, такі як обробка великого обсягу інформації, складні обчислення та керування багатьма паралельними процесами.

Інформаційні технології є надзвичайно значущим та гнучким ресурсом, яким менеджери можуть скористатися. Багато компаній вже утворили посаду головного

директора інформаційних технологій (CIO), який входить до складу виконавчого комітету, разом з генеральним директором (CEO), фінансовим директором (CFO), головним оперативним директором (COO) та головним технологом. Головний технолог також може виконувати роль головного директора інформаційних технологій і навпаки.

Управління інформаційною безпекою зосереджується на роботі генерального директора з інформаційної безпеки (CISO).

Для створення інформаційної системи необхідно поєднати чотири компоненти.

У контексті інформаційних систем обладнання - це механічні та технічні ресурси, необхідні для використання та обслуговування. Сюди входить не лише сам комп'ютер, але й усі його численні частини у сучасних інформаційних системах. Пристрої вводу/виводу, зберігання та комунікації підпадають під категорію допоміжного обладнання. Важливо пам'ятати, що апаратне забезпечення в ранніх комп'ютерних інформаційних системах могло включати навіть матеріальні ресурси, такі як книги та чорнила.

Програмне забезпечення охоплює комп'ютерні програми та будь-які пов'язані з ними документи, що підтримують їхню функціональність. Ці програми складаються з машинозчитувальних інструкцій, які керують роботою апаратних компонентів, дозволяючи витягувати цінну інформацію з даних. Зазвичай програми зберігаються на носіях введення/виведення, таких як жорсткі диски або касети. У попередніх інформаційних системах програмне забезпечення також охоплювало такі завдання, як підготовка обладнання до використання (наприклад, форматування стовпчиків у книзі) та надання інструкцій щодо його використання (наприклад, керівництво для каталожних карток).

З іншого боку, дані - це фактична інформація, яка використовується системою для отримання корисних висновків. У сучасних інформаційних системах дані зазвичай зберігаються у машинозчитувальному форматі на пристроях

зберігання, таких як жорсткі диски або стрічки, до того часу, поки вони не будуть потрібні комп'ютеру. У більш ранніх інформаційних системах дані часто зберігалися у форматі, який можна було безпосередньо прочитати і зрозуміти без необхідності комп'ютерної обробки.

Процедура: Процедура визначає політику функціонування інформаційної системи управління. "Програми для людей, програмне забезпечення для апаратних засобів" - це загальноприйнята метафора, яка демонструє роль програм у системі.

Люди: Кожній системі потрібні люди для функціонування. Зазвичай найменш помітним компонентом системи є людина, яка може мати найбільший вплив на успіх або невдачу інформаційної системи. Це охоплює не тільки користувачів, але й тих, хто керує та обслуговує комп'ютери, хто забезпечує підтримку даних та мереж комп'ютерів.

Зворотній зв'язок: Цей компонент необов'язковий для функціонування інформаційної системи, але визначає його здатність надавати зворотний зв'язок.

1.3 Поняття веб-додатку

Веб-додаток, відомий як програмне забезпечення, що працює на веб-сервері, відмінно від локальних комп'ютерних програм, які запускаються на пристрої в операційній системі (ОС). Веб-додатки доступні користувачам через веб-браузер з активним інтернет-з'єднанням. Вони програмуються за допомогою клієнт-серверної архітектури, де послуги надаються користувачеві (клієнту) через віддалений сервер. Приклади широко використовуваних веб-додатків включають веб-пошту, онлайн-роздрібні продажі, онлайн-банкінг і онлайн-аукціони.

Відмінність між динамічними веб-сторінками будь-якого типу і "веб-додатком" не завжди очевидна. Веб-сайти, які часто називаються "веб-додатками", мають функціональні можливості, схожі на програмне забезпечення для настільних комп'ютерів або мобільних додатків. HTML5 дозволить створювати програми, як

веб-сторінки, але вони також можуть працювати автономно та зберігати дані локально.

Оскільки вони відмовляються від традиційної веб-парадигми переміщення між різними сторінками за допомогою різних URL-адрес, односторінкові додати більше схожі на звичайні програми. Це означає, що окремі компоненти можуть бути замінені або оновлені без необхідності оновлювати всю веб-сторінку. Односторінкові фреймворки можуть прискорити розробку таких мобільних веб-додатків, зменшуючи пропускну здатність і припиняючи потребу у завантаженні зовнішніх файлів.

В історії попередніх моделей обчислень, таких як клієнт-сервер, програмне навантаження розподілялося між кодом, розміщеним на сервері, та кодом, встановленим локально на кожному клієнті. Іншими словами, програма мала власну попередньо скомпільовану клієнтську програму, що слугувала інтерфейсом користувача і повинна була бути встановлена на кожному комп'ютері користувача окремо. Оновлення серверного коду програми зазвичай вимагало оновлення коду на клієнтській стороні, що встановлюється на кожному робочому місці користувача, що призводить до витрат на підтримку та зниження продуктивності. Крім того, як клієнтські, так і серверні компоненти програми зазвичай були тісно пов'язані з певною комп'ютерною архітектурою та операційною системою, тому перенесення їх на інші платформи часто було дорогим. Ці обмеження також стосуються нативних програм для мобільних пристроїв сьогодні.

Веб-додатки використовують веб-документи, такі як HTML і JavaScript, для створення інтерактивного досвіду для користувачів. Коли клієнт відвідує веб-сторінку, клієнтське програмне забезпечення (наприклад, веб-браузер) завантажується на його машину за допомогою стандартних процедур, таких як HTTP.

На початку Інтернету окремі веб-сторінки були статичними документами, але вони все одно могли забезпечувати інтерактивність за допомогою елементів

веб-форм, які вбудовувалися в розмітку сторінки. Однак, коли були потрібні значні зміни на веб-сторінці, користувачеві доводилося повертатися на сервер для оновлення всієї сторінки.

JavaScript, мова програмування на стороні клієнта, була вперше представлена компанією Netscape у 1995 році. За допомогою JavaScript до клієнтського інтерфейсу можна додавати динамічні функції. Сьогодні вбудовані скрипти виконують різноманітні дії, такі як перевірка введених даних або відображення чи приховування частин сторінки, замість того, щоб надсилати всі дані на сервер для побудови цілої веб-сторінки.

У 1996 році Macromedia випустила Flash - програвач векторної анімації, який надав можливості щодо встановлення у вигляді плагіну у браузери з метою вбудовування анімації на веб-сайти. Це надало можливість писати інтерактив на стороні клієнта за допомогою мови сценаріїв без необхідності підключення до сервера.

Ці розробки дозволили веб-дизайнерам створювати динамічні, інтерактивні веб-додатки, які працюють повністю на стороні клієнта, покращуючи користувацький досвід.

Ідея "веб-додатку", побудованого на мові Java, вперше була описана в специфікації сервлетів версії 2.2 1999 року [2]. На той час вже використовувалися JavaScript і XML, однак технологію Ajax ще не було розроблено, а об'єкт XMLHttpRequest тільки нещодавно став доступний як об'єкт ActiveX в Internet Explorer 5.

Впровадження концепції Ajax у 2005 році дозволило таким додаткам, як Gmail, забезпечити більш динамічну роботу на стороні клієнта. Веб-сторінки могли взаємодіяти з сервером за допомогою скриптів для збереження або отримання даних без необхідності повного перезавантаження сторінки [3].

У 2007 році Стів Джобс зазначив, що оптимальним форматом додатків для iPhone будуть веб-додатки, побудовані на основі HTML5 і з використанням

технології Ajax. Вони повністю інтегрувалися в пристрій через браузер Safari і не потребували створення SDK. З часом ця концепція була додана в App Store, що задовільнило розробників та обмежило поширення джейлбрейку.

Створення HTML5 у 2014 році дозволило надавати графічні та мультимедійні можливості без встановлення клієнтських плагінів. Також завдяки HTML5 було покращено семантичне наповнення документів. Основними частинами специфікації HTML5 були API та об'єктна модель документа (DOM), а API WebGL уможливив потужну 3D-графіку на основі HTML5 та JavaScript. Ці технології мають вирішальне значення для створення крос-платформних, крос-браузерних веб-додатків з багатьма функціями.

Ерік Бідельман (старший інженер-програміст), представив прогресивні веб-додатки (PWA) як новий стандарт у веб-розробці на щорічній конференції Google IO, яка відбулася у 2016 році. Джефф Бертофт (програмний директор Microsoft), визначив, що компанія вирішила повністю прийняти стандарт PWA, оскільки Google встановлює планку для створення прогресивних веб-додатків.

Веб-додатки використовують JavaScript і CSS для побудови користувацького інтерфейсу. Раніше також використовувалися такі технології, як Silverlight, Flash і Java. Ці технології уможливлюють виконання різноманітних завдань, зокрема передача звуку, малювання та взаємодію з клавіатурою і мишею. Для інтеграції цих розрізнених технологій в уніфікований і впізнаваний інтерфейс, подібний до інтерфейсу операційних систем, було розроблено різні сервіси. Крім того, підтримуються такі широко використовувані методи, як перетягування і падіння. Скріптинг на стороні клієнта часто використовується розробниками веб-додатків для поліпшення функціональності, особливо при наданні інтерактивного досвіду без необхідності повного перезавантаження сторінки. В результаті останніх розробок було створено такі технології, як Perl/Plack , J2EE, ASP.NET і PHP, що інтегрують клієнтські сценарії з технологіями на стороні сервера. Технологією, яка забезпечує більш інтерактивний досвід, є Ajax, яка інтегрує кілька інструментів

веб-розробки. Веб-додатки мають багаторівневу архітектуру. На клієнтській системі традиційні додатки зазвичай складаються лише з одного рівня. Однак багаторівнева стратегія природно підходить і для веб-додатків.

Трирівнева архітектура є найпоширенішим типом побудови. Веб-браузер представлений першим рівнем в цій архітектурі, який називається рівнем представлення. Python, JSP/Java, Node.js, ColdFusion, Ruby on Rails, CGI, Dart, ASP, PHP - ось деякі приклади серверних технологій, які використовуються другим рівнем, або рівнем додатку, для передачі логіки програми. База даних, де зберігаються дані, представлена третім рівнем, який також називають рівнем зберігання. Другий рівень отримує запити від веб-браузера, обробляє їх, виконує необхідні операції з базою даних і утворює користувацький інтерфейс, який використовується для взаємодії з програмою. Третій рівень, або зберігання, представляє базу даних, де зберігаються дані. Веб-браузер надсилає запити на другий рівень, який обробляє їх, виконує необхідні операції з базою даних й створює інтерфейс користувача для взаємодії з додатком.

Для складних додатків, 3-рівневе рішення може бути недостатнім, і використання n-рівневого підходу може бути корисним. Основна перевага полягає у розбитті бізнес-логіки на більш детальну модель на рівні програми. Додатковою перевагою є використання рівня інтеграції, що відокремлює рівень даних від інших рівнів і забезпечує простий інтерфейс доступу до даних. Наприклад, для отримання даних клієнта можна використовувати функцію "list_clients()" замість безпосереднього SQL-запиту до таблиці клієнтів в базі даних. Це дозволяє замінити базу даних без внесення змін на інших рівнях.

Деякі люди вважають, що веб-додаток складається з двох рівнів: або «розумний» клієнт зв’язується з «тупим» сервером і виконує всю роботу, або «тупий» клієнт залежить від «розумного» сервера. Клієнт керує прикладним рівнем, сервер обробляє операції з базою даних (рівень зберігання), і один або обидва з них розміщують рівень відображення та бізнес-логіку.

Хоча ця парадигма відокремлює дисплей від бази даних і збільшує масштабованість додатків, вона все ще не повністю спеціалізує рівні, тому більшість програм виходять за її межі [4].

1.4 Основні принципи веб-розробки

Веб-сайт - це сукупність веб-сторінок і пов'язаної з ними інформації, яка публікується на веб-сервері під єдиним доменним ім'ям. Прикладами є такі веб-сайти, як Google.com, Amazon.com, Wikipedia.org. Всесвітня павутина складається з усіх сайтів, які є відкритими для спільноти. Крім того, існують приватні веб-сайти, які захищені паролем і доступні лише через приватну мережу, наприклад, внутрішній веб-сайт компанії для співробітників.

Веб-сайти, як правило, зосереджені на одній темі або меті, наприклад, новини, бізнес, освіта, медицина, розваги або соціальні мережі. Навігація по веб-сайту здійснюється за допомогою гіперпосилань між веб-сторінками, починаючи з головної сторінки.

Користувачі можуть відвідувати веб-сайти на різних пристроях, включаючи ПК, ноутбуки, планшети та смартфони.

Для цього вони використовують веб-браузери - програми, які дозволяють користувачам отримувати доступ до веб-сторінок.

У 1990 році британський фізик з ЦЕРНу Тім Бернерс-Лі розробив Всесвітню павутину (WWW) [1]. Всесвітня павутина стала загальнодоступною і безкоштовною у квітні 1993 року, як повідомляє ЦЕРН [2]. До появи протоколу передачі гіпертексту (HTTP) використовувалися інші протоколи, такі як FTP і Gopher. За допомогою HTTP можна отримати певні файли з сервера. Для представлення документів часто використовувався формат текстових процесорів або простого тексту.

Персональний веб-сайт, корпоративний веб-сайт компанії, урядовий веб-сайт, веб-сайт організації тощо - це лише кілька прикладів багатьох видів використання веб-сайтів. Вони можуть створюватися людьми, компаніями або іншими групами і, як правило, мають певну тему або мету. Будь-який веб-сайт може мати посилання на інші веб-сайти, тому дуже важливо розрізняти їх, щоб користувачі могли їх розуміти. Динамічний веб-сайт відрізняється від статичного тим, що вміст на сторінках генерується за допомогою програмного забезпечення на сервері перед його відправкою клієнтському браузеру. Це дозволяє створювати більш інтерактивні та змінювані змістом веб-сторінки. Динамічні веб-сайти часто використовують бази даних для зберігання та управління інформацією, що відображається на сторінках.

На динамічному веб-сайті можуть бути функції, які дозволяють користувачам реєструватися, увійти в систему, надсилати коментарі, заповнювати форми, виконувати пошукові запити та взаємодіяти з іншими користувачами. Такі веб-сайти можуть надавати персоналізований вміст, який залежить від введених користувачем даних або попередніх дій.

Підписка на веб-сайт зазвичай вимагає від користувача платити певну суму грошей на певний період часу, щоб мати доступ до платного вмісту або послуг, наданих на сайті. Це може бути використано багатьма різними видами веб-сайтів, такими як новинні портали, стрімінгові платформи, академічні ресурси, ігрові сервіси та інші.

Веб-сайти соціальних мереж також можуть вимагати реєстрації користувача та підписки для отримання певних привілеїв або функцій, але багато з них пропонують базовий доступ безкоштовно.

Звідси виходить, що веб-сайти можуть мати різні моделі доступу до вмісту, включаючи безкоштовний доступ, підписку, плату за вміст або комбінацію цих підходів, залежно від мети та бізнес-моделі конкретного веб-сайту.

Для зручності редагування, такого як спільне використання панелі навігації на кількох сторінках, статичні веб-застосунки як і раніше, можуть використовувати серверні компоненти (SSI). Веб-сайти не вважаються динамічними, оскільки їх поведінка перед користувачем є статичною. Оскільки поведінка веб-сайту перед користувачем є статичною, він не вважається динамічним веб-сайтом. Динамічний веб-сайт - це сайт, який часто і автоматично змінює або персоналізує себе. За допомогою комп'ютерного коду, що генерує HTML, динамічні сайти створюються сервером "на льоту" (CSS відповідає за зовнішній вигляд і тому залишається статичним). CGI, Java-сервлети і Java Server Pages (JSP), Active Server Pages і ColdFusion (CFML) є одними з багатьох програмних систем, які дозволяють створювати динамічні веб-системи і веб-сайти. Для популярних мов програмування, таких як Perl, PHP, Python і Ruby, існують різноманітні фреймворки веб-додатків і системи веб-шаблонів, що спрощують створення складних динамічних веб-сайтів. Динамічні веб-сайти можуть відображати поточний стан розмови між користувачами, відстежувати мінливу ситуацію або адаптувати інформацію до потреб кожного користувача. Наприклад, коли запитується домашня сторінка новинного веб-сайту, веб-сервер може об'єднати фрагменти HTML з новинами, отриманими з бази даних або іншого веб-сайту через RSS, щоб створити сторінку з найсвіжішою інформацією. Динамічні веб-сайти можна зробити інтерактивними завдяки використанню HTML-форм, збереженню та зчитуванню файлів cookie браузера або створенню послідовних сторінок, на яких відображається історія попередніх кліків. Другий приклад динамічного контенту - це коли веб-сайт роздрібної торгівлі з базою даних медіапродукції дозволяє користувачам вводити пошукові терміни, наприклад, "Бітлз". У відповідь веб-сайт змінює свій вигляд і показує добірку товарів "Бітлз", включаючи компакт-диски, DVD та книги. JavaScript використовується інтерактивним HTML для того, щоб дати веб-браузеру вказівки, як динамічно змінювати вміст сторінки. Періодична автоматична регенерація великої кількості статичних сторінок є одним з методів

імітації певного типу динамічного веб-сайту без втрат продуктивності через запуск динамічного механізму для кожного користувача або з'єднання [4].

Історично склалося так, що веб-сайти складалися переважно з тексту, а зображення додавалися згодом. Плагіни для веб-браузерів використовувалися як додаткові компоненти для додавання аудіо, відео та інтерактивності. Наприклад, плагіни дозволили розробляти багатофункціональні додатки, які можна порівняти з настільними програмами, такими як текстовий процесор. Microsoft Silverlight, Adobe Flash, Adobe Shockwave та аплети, написані на Java, є прикладами відомих модулів цієї категорії. Однією з основних тенденцій у веб-дизайні 2010 року стала "адаптивний дизайн" веб-сайтів, який забезпечує оптимальний досвід перегляду для користувачів шляхом адаптації макету до пристрою, на якому відбувається перегляд. Це дозволяє користувачам отримати зручну і збалансовану інтеракцію з сайтом незалежно від пристрою чи платформи, на якій він відкритий, забезпечуючи високу якість користування [6].

Веб-сайти можна розділити на дві основні категорії: статичні та інтерактивні. Інтерактивні сайти входять до спільноти веб-сайтів Web 2.0 і дозволяють взаємодіяти між власниками сайту та відвідувачами чи користувачами. Статичні сайти, натомість, надають або збирають інформацію, але не дозволяють прямо взаємодіяти з аудиторією чи користувачами. Деякі веб-сайти є інформативними або створюються ентузіастами для особистих цілей чи розваги.

Багато веб-сайтів спрямовані на заробіток, використовуючи одну або кілька бізнес-моделей [4]:

– розміщення цікавого матеріалу та здійснення продажу контекстної реклами, як прямими продажами, так і через рекламні мережі;

– електронна комерція: покупка товарів або послуг безпосередньо через веб-сайт;

– реклама товарів або послуг, що пропонуються у будівельному та цегельному бізнесі;

– базовий зміст, доступний безкоштовно, але для преміум-змісту потрібно сплатити (наприклад, платформа з відкритим кодом WordPress для створення блогів та веб-сайтів).

Деякі веб-сайти можуть відноситись до однієї або кількох з цих категорій. Наприклад, веб-сайт бізнесу може просувати свою продукцію, а також містити інформативні документи, такі як технічні специфікації. Крім того, існує безліч підкатегорій, які випливають із вказаних вище. Наприклад, порносайт може відноситись до електронної комерції або бізнес-сайтів (якщо пропонує платне членство для доступу до свого вмісту), або мати функціонал соціальної мережі. Фан-сайт може бути присвячений відповідній знаменитості. Веб-сайти обмежені архітектурними обмеженнями, такими як обчислювальна потужність, приділена самому сайту. Facebook, Yahoo!, Microsoft і Google – це приклади великих веб-сайтів, які використовують численні сервери та обладнання для розподілу навантаження. Комутатори Cisco Content Services Switches, наприклад, використовуються для розподілу навантаження від відвідувачів між різними комп'ютерами в різних місцях.

Так, починаючи з 2011 року, Facebook мав дев'ять центрів обробки даних з приблизно 63 000 серверів. Компанія Netcraft, яка відстежує розвиток Інтернету з 1995 року, повідомила, що станом на лютий 2009 року існувало 215 675 903 веб-сайтів із доменними іменами та контентом, тоді як у серпні 1995 року було лише 19 733 веб-сайти [8]. Після перевищення 1 мільярду веб-сайтів у вересні 2014 року, згідно з дослідженням веб-серверів Netcraft, кількість веб-сайтів зменшилася і впала нижче 1 мільярда у жовтні 2014 року. Це сталося через щомісячні коливання кількості неактивних доменів. Після березня 2016 року кількість веб-сайтів продовжила зростати і перевищила 1 мільярд [9].

Веб-розробка - це процес створення веб-сайтів для Інтернету (Всесвітньої павутини) або інтрамережі (приватної мережі) [1]. Вона охоплює широкий спектр завдань, починаючи від створення базових статичних веб-сторінок і закінчуючи

розробкою складних інтернет-додатків, платформ електронної комерції та соціальних мереж. Веб-інженерія, веб-дизайн, розробка контенту, комунікація з клієнтами, написання клієнт-серверних скриптів, налаштування веб-сервера та мережевої безпеки, а також впровадження електронної комерції є найважливішими аспектами веб-розробки.

Цей термін зазвичай відноситься до технічних аспектів, а не до дизайну у веб-розробці. Він включає в себе написання розмітки та коду для розробки веб-сайтів [2]. CMS часто використовуються для спрощення управління контентом і роблять його більш доступним для користувачів з елементарними технічними навичками.

В процесі розробки веб-сайтів команди веб-розробників у великих організаціях і корпораціях можуть складатися з сотень професіоналів, які дотримуються стандартних методологій, таких як Agile.

У менших організаціях може бути один спеціаліст, який працює за контрактом, або особи, які виконують другорядні ролі, наприклад, графічні дизайнери чи технічні спеціалісти з інформаційних систем. Веб-розробка часто передбачає співпрацю між різними відділами, а не обмежується конкретним відділом. Спеціалізації у веб-розробці включають фронтенд-розробників, бек-енд-розробників та повностекових розробників. Фронтенд-розробники зосереджуються на користувацькому досвіді та візуальних ефектах, тоді як бек-енд-розробники займаються операціями на стороні сервера.

Комерціалізація Інтернету призвела до зростання індустрії веб-розробки. Багато компаній визнають важливість використання веб-сайтів для реклами та продажу товарів і послуг споживачам [3].

Сфера веб-розробки пропонує безліч інструментів з відкритим вихідним кодом, які сприяють її розвитку та мінімізують витрати на навчання. Прикладами є BerkeleyDB, GlassFish, стек LAMP (Linux, Apache, MySQL, PHP) та Perl/Plack. Ці інструменти сприяли економічно ефективному навчанню веб-розробці. Більше того, галузь стала свідком розвитку зручного програмного забезпечення для веб-

розробки WYSIWYG, такого як Adobe Dreamweaver, BlueGriffon та Microsoft Visual Studio, що полегшило створення веб-сайтів для людей без глибоких знань мов програмування, таких як HTML.

Розробники можуть створювати динамічні та інтерактивні веб-сайти завдяки постійному розвитку інструментів і технологій. Крім того, веб-розробники тепер пропонують свої програми як веб-сервіси, розширюючи сферу їх використання за межі традиційних настільних додатків. Поява хмарних сервісів, таких як Adobe Creative Cloud, Dropbox і Google Drive, є результатом цього зміни, яка дозволила спільне використання медіа та децентралізоване розповсюдження даних. Користувачі тепер можуть використовувати програми та взаємодіяти з ними з різних місць, а не обмежуватися однією робочою станцією.

Інтернет бере свій початок із розробок, спрямованих на створення та взаємозв’язок комп’ютерних мереж, що з’явилися внаслідок досліджень США та передбачали міжнародну співпрацю, зокрема з дослідниками з Великої Британії та Франції 1. Історія Інтернету починається 29 жовтня 1969 року, коли Каліфорнійський університет у Лос-Анджелесі (UCLA), а незабаром і три інші університети прийняли участь в експерименті, який мав дослідити можливість побудови комп’ютерної мережі без виділеної центральної точки 1.

Згідно з дослідженням соціологічного центру Pew Research, до 2025 року більшість людей почне сприймати інтернет як необхідний елемент їх життя 2. Це означає, що Інтернет продовжить впливати на багато аспектів нашого життя, включаючи бізнес та комунікації. Це також означає, що заходи безпеки повинні продовжуватись розвиватися, щоб захистити користувачів від шкоди. Це може включати в себе розробку нових технологій та методик для запобігання шкодливим практикам, таким як SQL-ин’єкція та іншим видам атак. Також це може означати поступове збереження конфиденційних даних та застосування криптографічних методів для захисту передачі даних через Інтернет.

Розвиток Інтернету також змінив бізнес і комунікації, зокрема електронну комерцію. Такі сайти онлайн-аукціонів, як eBay, змінили те, як клієнти знаходять і купують товари та послуги. Незліченна кількість людей помітила, як Amazon.com і Buy.com змінили їхній спосіб купівлі. Крім того, з’явлення платформ для ведення блогів, таких як Movable Type і WordPress, дозволило людям використовувати форуми для обміну ідеями та думками. Впровадження корпоративних і з відкритим кодом систем управління контентом посилило вплив веб-розробки на спілкування та взаємодію в Інтернеті. Веб-сайти змінили маркетинг і особисте спілкування. Веб-сайти зараз не просто інструменти для продажу чи бізнесу; вони також використовуються як канали спілкування та соціальні мережі. Такі платформи, як Facebook і Twitter, надають людям платформу для спілкування та організаціям більш персоналізований і інтерактивний спосіб залучення громадськості.

Заходи безпеки, які використовуються в веб-розробці, включають перевірку помилок при введенні даних через форми, фільтрацію вихідних даних і шифрування. Зловмисники можуть використовувати шкідливі практики, такі як SQL-ін’єкції, але тільки якщо вони мають базове розуміння веб-розробки. Зловмисники, які намагаються зібрати конфіденційні дані, такі як адреси електронної пошти, паролі та номери кредитних карток, можуть використовувати скрипти для експлуатації веб-сайтів.

Дещо залежить від серверного середовища, в якому виконується мова сценаріїв, наприклад, ASP, JSP, PHP, Python, Perl або Ruby, і тому не обов'язково підтримується розробником. Однак рекомендується, щоб веб-додатки проходили ретельне тестування перед їх розповсюдженням серед широкої публіки, щоб запобігти таким експлоітам. Якщо веб-сайт містить контактну форму, вона повинна містити поле captcha, яке не дозволяє комп'ютерним програмам автоматично заповнювати форми та надсилати електронну пошту.

Зміцнення портів сервера — це термін, який використовується для опису захисту веб-сервера від атак. Для захисту передачі даних через Інтернет з одного

місця в інше застосовуються різні заходи. Наприклад, центри сертифікації виробляють сертифікати TLS, також відомі як «SSL-сертифікати», щоб запобігти шахрайству в Інтернеті. Розробники часто використовують кілька різних методів шифрування, коли передають і зберігають конфіденційні дані. Часто веб-розробники знають основи безпеки інформаційних технологій.

Навіть після тестування та випуску нових веб-додатків виявляються нові вразливості в системі безпеки, що вимагає регулярного оновлення популярних програмних патчів безпеки. Веб-розробники часто відповідають за підтримку програм в актуальному стані, коли випускаються патчі безпеки та виявляються нові проблеми з безпекою.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Вибір архітектури

Model-View-Controller (MVC) – це патерн проєктування програмного забезпечення [1] для розробки користувацьких інтерфейсів, який ізолює програмну логіку на три взаємопов'язані компоненти: модель, представлення та контролер. Це робиться для того, щоб відрізнити внутрішнє представлення інформації від способу, за допомогою якого користувач може отримати до неї доступ [2, 3].

Цей патерн, який традиційно використовується для настільних графічних інтерфейсів користувача (GUI), набув популярності в розробці веб-додатків [4]. Фреймворки MVC для популярних мов програмування полегшують реалізацію цих патернів.

MVC став одним з перших методів опису та реалізації програмних структур відповідно до їхніх обов'язків [5], як одна із загальних концепцій ранньої розробки графічних інтерфейсів користувача.

Наприкінці 1970-х років Трігве Ренскауг, під час роботи над Smalltalk-79, як запрошений науковець у дослідницькому центрі Xerox Palo Alto Research Center (PARC), розробив MVC [6, 7, 8]. Він намагався створити систему, яку можна було б впровадити в будь-яку програму, в якій користувачі взаємодіють з великими й складними колекціями даних. Початкова конструкція включала модель, подання, предмет і редактор. Він із іншими членами команди визначилися з моделлю, представленням і контролером [6] після консультацій з іншими розробниками Smalltalk.

У своїй кінцевій формі модель чітко та інтуїтивно представляє програмні компоненти. Представлення - це візуальне відображення моделі, яке показує дані моделі для користувача та відповідає на запити. Контролер організовує і координує декілька представлень на екрані, отримує вхідні дані від користувача і надсилає

відповідні повідомлення своїм базовим представленням. Ця архітектура також включає редактор, як спеціалізований тип контролера для модифікації створеного за допомогою нього подання [6].

Smalltalk-80 підтримує розвинений варіант MVC [6]. Він надає абстрактні класи для представлення та контролера, а також конкретні підкласи для кожного абстрактного класу, які представляють різні загальні об'єкти. У цьому сценарії представлення представляє конкретний метод відображення інформації користувачеві, тоді як контролер представляє конкретний метод взаємодії з представленням. Представлення також пов'язане з об'єктом моделі, структура якого визначається програмістом. Smalltalk-80 також включає в себе MVC Inspector, інструмент розробки для вивчення структури моделі, представлення та контролера даного додатку [9].

У Journal of Object Technology (JOT) двоє колишніх співробітників PARC в 1988 році представили MVC як загальну "парадигму та метод програмування" для розробників Smalltalk-80. Однак, вказана схема відрізняється від тієї, що була представлена в буклеті Smalltalk-80 Ренскаугом та іншими. Вигляд, згідно з їх визначенням, включає в себе усі графічні питання, тоді як контролер є більш абстрактним, зазвичай невидимим об'єктом. Він отримує дані користувача і взаємодіє з одним або декількома представленнями даних, а також володіє єдиною моделлю [10].

Згодом патерн MVC еволюціонував [11], в результаті чого з'явилися такі моделі, як: ієрархічна модель-подання-контролер (HMVC), модель-подання-адаптер (MVA), модель-подання-презентатор (MVP) й модель-подання-модель (MVVM), які реалізують MVC в різних контекстах.

Після впровадження NeXT WebObjects у 1996 році, який спочатку був написаний мовою Objective-C (значною мірою похідною від Smalltalk) і допоміг популяризувати принципи MVC, патерн MVC став більш поширеним у веб-додатках. Патерн MVC набув популярності серед Java-розробників після

перенесення WebObjects на Java. Тісний зв'язок між Java та MVC підтримувався наступними фреймворками Java, такими як Spring (вийшов у жовтні 2002 року).

У книзі Мартіна Фаулера "Шаблони архітектури корпоративних додатків", опублікованій у 2003 році, MVC пропонується як парадигма, в якій "вхідні контролери" приймають запити, надсилають відповідні повідомлення об'єктам моделі, отримують відповіді від об'єктів моделі та передають відповіді [8]. Це схоже на техніку фреймворку Ruby on Rails (серпень 2004 року), в якому клієнт надсилає запит до сервера через браузерне представлення, який потім обробляє його. Після цього контролер спілкується з асоційованим об'єктом моделі [12].

Фреймворк Django (для Python, липень 2005 року) має схожу парадигму, названу "MTV" (Model Template View), в якій представлення бере дані з моделі і передає їх шаблону для відображення [13]. Поява Rails та Django, які роблять акцент на швидкому розгортанні, розширила популярність MVC за межі його багаторічного успіху в традиційних корпоративних середовищах.

Будь-яке представлення інформації, наприклад, діаграма, графік або таблиця, мають можливі різні точки зору на одну й ту ж інформацію, наприклад, гістограма для керівництва та таблиця для бухгалтерів.

Контролер.

Команди для перетворення вхідних даних у модель або подання [15].

Конструкція модель-подання-контролер не тільки розділяє програмне забезпечення на ці компоненти, але й визначає їхню взаємодію [16]. Модель відповідає за управління даними програми. Вона отримує дані користувача від контролера, демонструє представлення моделі у певному форматі, реагує на вхідні дані користувача та взаємодіє з об'єктами моделі даних. Контролер отримує вхідні дані, перевіряє їх, якщо це необхідно, а потім передає їх моделі.

MVC, як і інші патерни програмного забезпечення, визначає "базове рішення" проблеми, що дозволяє застосовувати його до будь-якої системи [17].

Конкретний дизайн MVC може значно відрізнятися від традиційних описів, наведених тут [18].

Використання у веб-додатках.

Хоча MVC був розроблений для настільних комп'ютерів, основні мови програмування масово застосовують його для розробки веб-додатків. Ця парадигма була реалізована в ряді веб-фреймворків. Ці програмні архітектури інтерпретуються по-різному, насамперед у тому, як обов'язки MVC розподіляються між клієнтом та сервером [20].

Деякі веб-системи MVC використовують архітектуру тонкого клієнта, де практично вся функціональність моделі, представлення та контролера виконується на сервері. Клієнт надсилає гіперпосилання або запит на форму до контролера, який отримує повну та оновлену веб-сторінку (або інший документ) з представлення, модель повністю знаходиться на сервері [20].

2.2 Аналіз варіантів діяльності

Першим кроком має бути проєктування майбутніх функцій системи.

За допомогою діаграм варіантів використання UML можна ефективно охопити системні вимоги, моделюючи поведінку системи.

Діаграми варіантів використання пропонують огляд верхнього рівня, щоб показати функції та межі системи. Крім того, вони описують, як працює інтерфейс між акторами та системою. Діаграми варіантів використання та акторів показують дії системи, а також те, як її використовують користувачі, однак вони не показують внутрішню роботу системи.

Діаграми варіантів використання демонструють контекстуальні та основні вимоги всієї системи або її важливих компонентів. Можна використовувати ці діаграми для моделювання складної системи в цілому або для створення кількох діаграм, щоб показати різні частини системи. Діаграми варіантів використання

зазвичай створюються на ранніх стадіях проєкту та служать орієнтиром під час усього процесу розробки.

Діаграми варіантів використання корисні в таких ситуаціях: вони можуть бути використані для моделювання бізнесу перед початком проєкту, щоб усі учасники проєкту могли краще зрозуміти, що відбувається з працівниками, клієнтами та бізнесом; вони також можуть бути використані щодо зображення системних вимог, для пояснення, що має робити система, коли збираються вимоги.

Випадки використання описують елементи моделі в діаграмах варіантів використання. Варіант використання описує функцію, яку система виконує для досягнення мети користувача. Щоб користувач системи був задоволений, варіант використання повинен мати чіткий результат.

Актори: Актор виконує роль користувача, який використовує систему, яку ви моделюєте. Користувач може бути індивідом, компанією, машиною або іншою зовнішньою системою.

Діаграми варіантів використання показують зв’язки між елементами моделі. Тип елемента моделі, відомий як відношення UML, визначає структуру та поведінку між елементами моделі, щоб додати семантику до моделі.

Рисунок 2.1 демонструє діаграму варіантів використання.

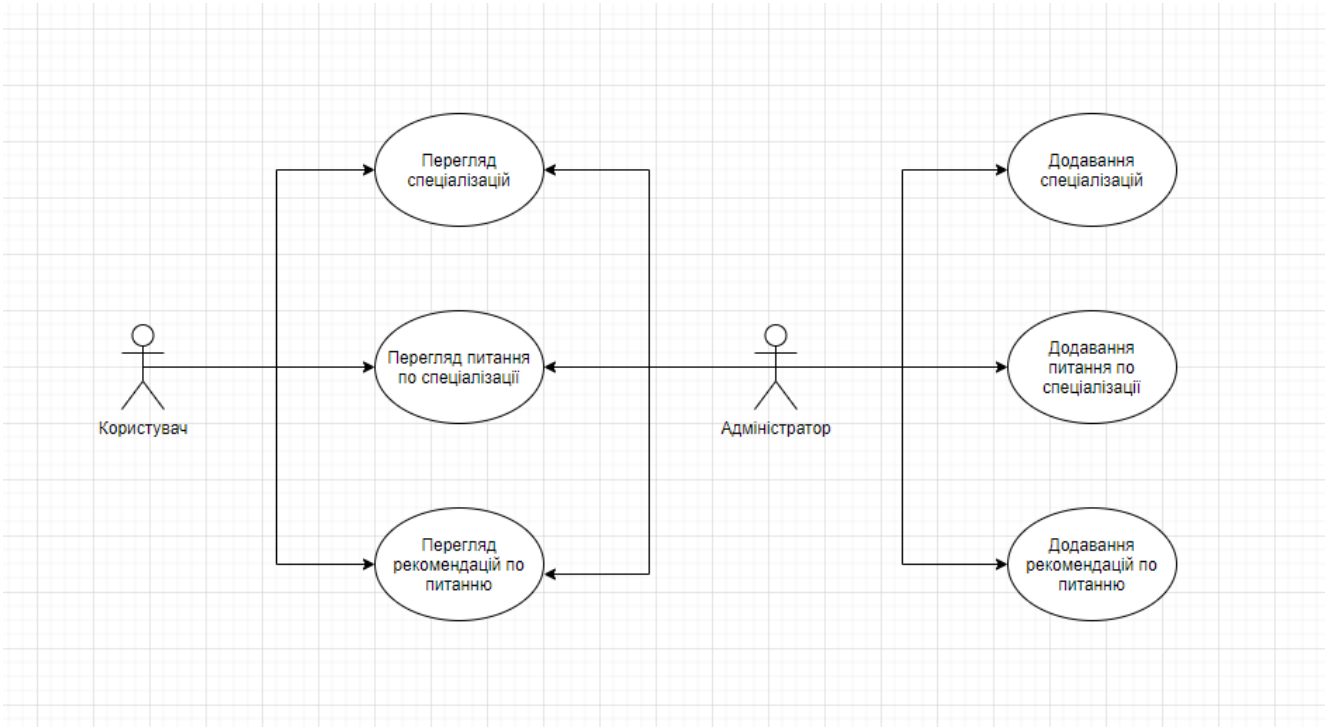

Рисунок 2.1 – Діаграма варіантів використання

2.3 Проєктування внутрішньої будови

Для проєктування внутрішньої будови системи було розроблено UML діаграму класів, яка ілюструє внутрішню структуру проєкту.

В UML діаграми класів є одним із шести типів структурних діаграм. Діаграми класів є фундаментальними для процесу моделювання об’єктів і моделюють статичну структуру системи. Залежно від складності системи ви можете використовувати одну діаграму класів для моделювання всієї системи або кілька діаграм класів для моделювання компонентів системи.

Діаграми класів — це креслення вашої системи або підсистеми. Ви можете використовувати діаграми класів, щоб моделювати об’єкти, які складають систему, відображати зв’язки між об’єктами та описувати, що ці об’єкти роблять, і послуги, які вони надають.

Діаграми класів корисні на багатьох етапах проєктування системи. На етапі аналізу діаграма класів може допомогти вам зрозуміти вимоги вашої проблемної

області та визначити її компоненти. У проєкті об’єктно-орієнтованого програмного забезпечення діаграми класів, які ви створюєте на ранніх стадіях проєкту, містять класи, які часто перетворюються на фактичні класи та об’єкти програмного забезпечення під час написання коду. Пізніше ви можете уточнити свій попередній аналіз і концептуальні моделі в діаграми класів, які показують конкретні частини вашої системи, інтерфейси користувача, логічні реалізації тощо. Тоді ваші діаграми класів стають знімком, який точно описує, як працює ваша система, зв’язки між системними компонентами на багатьох рівнях і те, як ви плануєте реалізувати ці компоненти.

Ви можете використовувати діаграми класів для візуалізації, визначення та документування структурних особливостей у своїх моделях. Наприклад, на етапах аналізу та проєктування циклу розробки ви можете створювати діаграми класів для виконання таких функцій:

– зберіть і визначте структуру класів та інших класифікаторів;

– визначте зв’язки між класами та класифікаторами;

– проілюструйте структуру моделі за допомогою атрибутів, операцій і сигналів;

– покажіть загальні ролі та обов’язки класифікатора, які визначають поведінку системи;

– показати класи реалізації в пакеті;

– покажіть структуру та поведінку одного чи кількох класів;

– показати ієрархію успадкування серед класів і класифікаторів;

– покажіть працівників і сутності як моделі бізнес-об’єктів.

На етапі впровадження циклу розробки програмного забезпечення ви можете використовувати діаграми класів для перетворення ваших моделей у код і для перетворення вашого коду в моделі.

У наступних темах описуються елементи моделі в діаграмах класів:

Заняття.

В UML клас представляє об’єкт або набір об’єктів, які мають спільну структуру та поведінку. Класи або екземпляри класів є звичайними елементами моделі в діаграмах UML.

Об'єкти.

У моделях UML об’єкти — це елементи моделі, які представляють екземпляри класу або класів. Ви можете додавати об’єкти до своєї моделі для представлення конкретних та прототипних екземплярів. Конкретний екземпляр представляє реальну особу чи річ у реальному світі. Наприклад, конкретний екземпляр класу Customer представляє реального клієнта. Прототипний екземпляр класу Customer містить дані, які представляють типового клієнта.

Пакети.

Пакети групують пов’язані елементи моделі всіх типів, включаючи інші пакунки.

Сигнали.

У моделях UML сигнали є елементами моделі, які не залежать від класифікаторів, які їх обробляють. Сигнали визначають односторонній асинхронний зв’язок між активними об’єктами.

Перерахування.

У моделях UML перерахування - це елементи моделі в діаграмах класів, які представляють визначені користувачем типи даних. Переліки містять набори іменованих ідентифікаторів, які представляють значення переліку. Ці значення називаються літералами перерахування.

Типи даних.

У діаграмах UML типи даних - це елементи моделі, які визначають значення даних. Типи даних зазвичай використовуються для представлення примітивних типів, таких як цілі чи рядкові типи, і перерахувань, таких як типи даних, визначені користувачем.

Артефакти.

У UML-моделях артефакти - це елементи моделі, які представляють фізичні сутності в програмній системі. Артефакти являють собою фізичні одиниці реалізації, такі як виконувані файли, бібліотеки, програмні компоненти, документи та бази даних.

На рисунку 2.2 зображено діаграму класі системи.

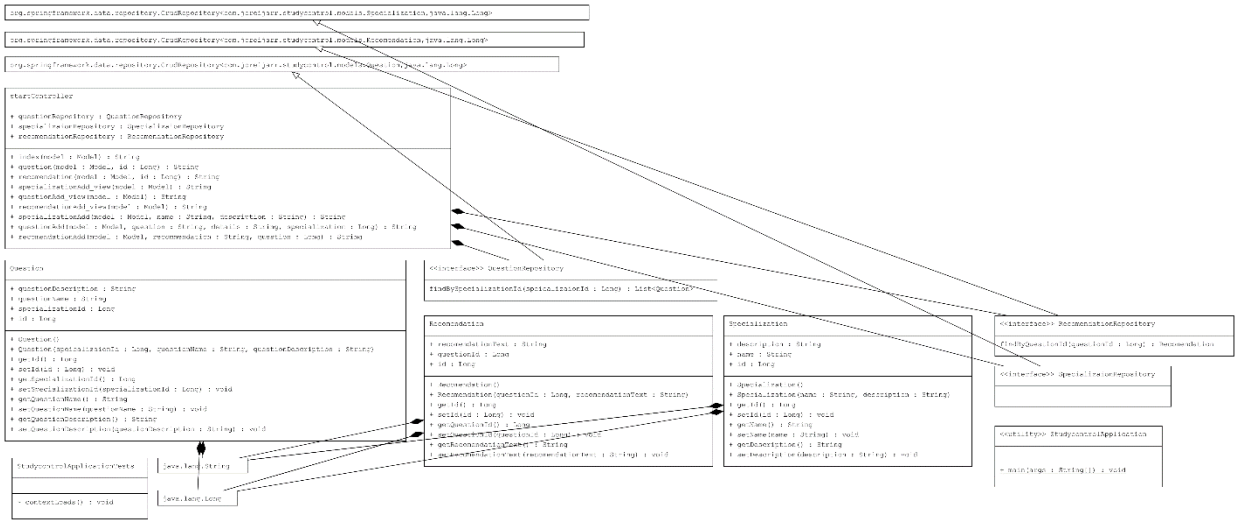

Рисунок 2.2 – Діаграма класів системи

3 РОЗРОБКА СИСТЕМИ

3.1 Вибір інструментальних засобів розробки

3.1.1 Узагальнення мови програмування

Java є високорівневою, об’єктно-орієнтованою мовою програмування, що базується на класах, з метою мінімізації залежностей при реалізації. Java є мовою програмування загального призначення, тому скомпільований код може працювати на будь-якій платформі, яка підтримує Java, без перекомпіляції [18]. «Написати один раз, запустити де завгодно» (WORA) є терміном, який стосується цього [17]. Програми Java зазвичай упаковуються в байт-код, який дозволяє їх виконувати на будь-якій віртуальній машині Java (JVM), незалежно від базової архітектури комп’ютера. Хоча Java не має так багато низькорівневих функцій, як C++ або C, вона схожа за синтаксисом на ці мови. Динамічні можливості середовища виконання Java відрізняються від звичайних скомпільованих мов, такі як можливість переглядати та змінювати код у режимі реального часу. Станом на 2019 рік GitHub повідомив, що Java мала 9 мільйонів розробників і була однією з найпоширеніших мов програмування [19], [20], особливо щодо клієнт-серверних веб-додатків [21].

На початку Sun Microsystems Java була розроблена Джеймсом Гослінгом, а вже в травні 1995 року вона стала основним компонентом Sun Microsystems Java Platform. Спочатку Sun пропонувала віртуальні машини, бібліотеки класів і компілятори Java під власними ліцензіями. На травень 2007 року Sun переліцензувала більшість своїх технологій Java під ліцензією GPL-2.0, вимагаючи, щоб вони виконували вимоги Java Community Process. Незважаючи на те, що Oracle має власну віртуальну машину Java HotSpot, OpenJDK JVM, який є вільним програмним забезпеченням з відкритим кодом і використовується більшістю розробників, є офіційною еталонною реалізацією і JVM за замовчуванням практично для всіх дистрибутивів Linux.

Незважаючи на те, що Java 17, 11 і 8 все ще перебувають на довгостроковій підтримці (LTS), остання версія Java 20 вже доступна з 21 березня 2023 року. Незважаючи на те, що Oracle продовжуватиме надавати загальнодоступні оновлення Java 8 для особистого використання, останнє безкоштовне оновлення для застарілої Java 8 LTS було доступне для комерційного використання в січні 2019 року. У вільних збірках OpenJDK 8 і 11 все ще є оновлення безпеки та додаткові оновлення від інших постачальників.

Oracle та інші наполегливо рекомендують не встановлювати застарілі та непідтримувані версії Java через невирішені проблеми з безпекою попередніх версій [22]. Користувачам надається можливість негайно оновитися до підтримуваної версії, наприклад, до однієї з версій LTS (8, 11 або 17), від Oracle.

Java Virtual Machine (JVM) і байт-код для Java

Одним із цілей архітектури Java є переносимість, що означає, що програми, створені для платформи Java, повинні працювати однаково на будь-якому апаратному забезпеченні та операційній системі, за умови підтримки під час виконання. Байт-код Java є проміжною формою коду Java, а не безпосередньо створенням машинного коду для конкретної архітектури. Інструкції байткоду Java, як і машинний код, розроблені для роботи на віртуальних машинах (ВМ). Кінцеві користувачі часто використовують веб-браузер, щоб переглядати програми Java, які працюють автономно, або середовище виконання Java (JRE), встановлене на їхньому пристрої.

Щоб отримати доступ до функцій хоста, таких як графіка, потоки та мережа, звичайно використовується стандартна бібліотека.

Універсальний байт-код полегшує перенесення. Через додаткові витрати, пов’язані з перетворенням байт-коду в машинні інструкції, інтерпретовані програми, однак, майже завжди, працювали повільніше, ніж нативні виконувані файли. На початку були розроблені компілятори «точно вчасно» (JIT), які можна було використовувати для перетворення байт-коду в машинний код, коли це було

необхідно. Компілятор Hotspot Java працює з GraalVM, який дозволяє багаторівневу компіляцію і включений у Java 11, але вилучений з Java 16 [17]. Він також об’єднує два компілятори в один. Віртуальна машина Java (JVM), яка перетворює байт-код Java на машинну мову платформи, дозволяє Java, яка не залежить від платформи, працювати на певній платформі [18].

Історично склалося так, що програми, розроблені на Java, працюють повільніше і використовують більше пам'яті, ніж ті, що створені на C++. Однак швидкість роботи програм на Java значно зросла після впровадження компіляції just-in-time у 1997-1998 роках. З введенням внутрішніх класів, класу StringBuilder та необов'язкових операторів, Java 1.1 покращила аналіз та виконання коду. Також було оптимізовано віртуальний рушій Java. У 2000 році Java стала стандартною JVM від Sun.

Продуктивність було ще більше покращено у наступних версіях Java, таких як Java 1.5 та Java 1.6. Покращення продуктивності стало можливим завдяки появі пакету java.util.concurrent в Java 1.5, включаючи безблокові ConcurrentMaps та інші реалізації багатоядерних колекцій.

Продуктивність Java також значно підвищується завдяки автоматизованому управлінню пам'яттю. Java автоматично збирає сміття для керування пам'яттю. Час виконання Java відповідає за відновлення пам'яті, коли об'єкти більше не використовуються, а програмісти вирішують, коли слід створювати об'єкти. Завдяки такій парадигмі автоматичного керування пам'яттю, програмістам більше не доведеться мати справу з ручним керуванням пам'яттю, що є типовим для таких мов, як C++. Щоб запобігти витоку пам'яті, коли на речі, які більше не потрібні, все ще посилаються і займають пам'ять, дуже важливо, щоб програмісти ефективно керували посиланнями на об'єкти.

Завдяки автоматизованому управлінню пам'яттю у Java немає необхідності у явному виділенні та звільненні пам'яті. Щоб відновити невикористану пам'ять, збирач сміття запускається через різні проміжки часу, в ідеалі, коли програма не

виконується активно. Java не пропонує явного виділення та звільнення пам'яті, як це робить C++, що дає C++ повний контроль над управлінням пам'яттю. Крім того, функції безпеки та захисту Java забороняють використання арифметики з вказівниками в парадигмі C/C++, забезпечуючи безпеку типів та уникаючи вразливостей у безпеці.

Примітивні типи даних в Java зберігаються не в купі, як не примітивні типи даних, а безпосередньо в полях (для об'єктів) або в стеку (для методів). Таке дизайнерське рішення було прийнято з метою максимізації продуктивності.

Garbage First Garbage Collector (G1GC) - це збірник сміття за замовчуванням, доступний в Java починаючи з версії 9. Купою можуть керувати різні збирачі сміття, хоча G1GC є достатнім для більшості програм на Java. Паралельний збирач купи використовувався у Java 8.

Програмістам також потрібно забезпечити належне поводження з іншими видами ресурсів, такими як дескриптори файлів, мережеві з'єднання та з'єднання з базами даних, особливо коли йдеться про винятки, навіть якщо автоматичне керування пам'яттю в Java спрощує управління пам'яттю.

Сервлет Джакарти - це програмний елемент Java, який збільшує функціональність сервера. Сервлети є серверними веб-інтерфейсами API, оскільки вони часто використовуються у веб-контейнерах для розміщення веб-додатків на веб-серверах. За своєю функцією у веб-розробці сервлети еквівалентні іншим технологіям, таким як PHP та ASP.NET, і дозволяють генерувати динамічний веб-контент.

Вступ до Java класів

Клас Java, який відповідає API сервлетів Джакарти, відомий як сервлет Джакарти, і він дозволяє програмістам створювати класи Java, які відповідають на запити. Сервлети дозволяють вставляти динамічний контент на веб-сервер, що працює на платформі Java, і часто використовуються з HTTP як клієнт-серверний протокол. Використовуючи HTTP-кукі або відображення URL, сервлети можуть

зберігати стан сеансу протягом численних серверних транзакцій, генеруючи при цьому вміст, який найчастіше є HTML, але також може бути XML і JSON.

Дві широко використовувані технології веб-сервісів Java, Jakarta RESTful Web Services (JAX-RS 2.0) для AJAX, JSON і REST-сервісів і Jakarta XML Web Services (JAX-WS) для SOAP-сервісів, частково замінили Jakarta Servlet API.

Веб-контейнер (або контейнер сервлетів) необхідний для розгортання і запуску сервлетів. Веб-контейнер - це частина веб-сервера, яка взаємодіє з сервлетами, керує їхнім життєвим циклом, пов'язує URL-адреси з конкретними сервлетами і гарантує, що запитувачі URL-адрес мають відповідні привілеї доступу. Сервлет API, який описує очікувану взаємодію між веб-контейнером і сервлетом, визначається ієрархією пакетів javax.servlet.

Отримавши запит, сервлет використовує цей запит для створення відповіді. Сервлет API визначає об'єкти Java, які представляють середовище виконання, налаштування конфігурації, запити та відповіді сервлетів. Ці об'єкти мають специфічні для HTTP підкласи, доступні в пакеті javax.servlet.http, включаючи об'єкти управління сеансами, які відстежують численні запити і відповіді, що надсилаються між веб-сервером і клієнтом. Сервлети можуть міститися у файлі WAR (Web ARchive) веб-додатку.

Компілятор Jakarta Server Pages дозволяє автоматично створювати сервлети з JSP. Сервлети і JSP можна порівняти, але JSP пропонують вищий рівень абстракції. JSP перетворюються в сервлети під час виконання і кешуються для подальшого використання. JSP можна використовувати незалежно або як компонент серверної архітектури модель-вид-контролер (MVC), де сервлети Java (або фреймворки, такі як Apache Struts) виступають в ролі контролера, а JavaBeans - в ролі моделі.

Для розгортання та роботи Jakarta Server Pages, таких як Apache Tomcat або Jetty, вам потрібен веб-сервер, сумісний з контейнерами сервлетів. Згенерована сторінка потім створюється і виконується на сервері, щоб надати документ. JSP

дозволяють змішувати код Java і задані дії зі статичним матеріалом веб-розмітки (наприклад, HTML). JSP компілюються в байт-код Java разом із залежними бібліотеками Java, і їм потрібна віртуальна машина Java (JVM) для виконання в операційній системі хоста сервера, щоб забезпечити нейтральне до платформи середовище.

Неявні об'єкти JSP, такі як запит, відповідь, сеанс, додаток, конфігурація, сторінка, pageContext, виняток і вихід, створюються веб-контейнером. Рушій JSP генерує ці об'єкти на етапі трансляції.

Компілятор JavaServer Page (JSP) сервера додатків - це програма, яка аналізує JSP-файли і перетворює їх на Java-сервлети, які можна виконувати. Для підвищення ефективності сторінки можуть бути попередньо скомпільовані або скомпільовані під час створення програми. Цей процес автоматично виконується при першому відвідуванні JSP.

Деякі контейнери JSP дозволяють налаштовувати частоту, з якою вони перевіряють мітки часу JSP-файлів, щоб дізнатися, чи змінилася сторінка. Інтервал перевірки міток часу для розгорнутих веб-програм можна встановити довшим (наприклад, хвилини), хоча для розробки програмного забезпечення зазвичай встановлюють короткий інтервал (наприклад, секунди).

3.1.2 Огляд середовища розробки

Java-додатки можна створювати за допомогою інтегрованого середовища розробки (IDE), відомого як IntelliJ IDEA. Воно було створене компанією JetBrains і доступне як у приватній комерційній версії, яка підходить для комерційної розробки, так і у версії Apache 2, ліцензованій спільнотою.

Одним з перших Java IDE, яке запропонувало складну навігацію та можливості рефакторингу коду, було IntelliJ IDEA, яке дебютувало в січні 2001 року. У звіті InfoWorld за 2010 рік IntelliJ, яка конкурувала з Eclipse, NetBeans і

JDeveloper, отримала найвищий рейтинг серед найкращих інструментів розробки на Java.

На основі відкритої версії IntelliJ IDEA компанія Google у грудні 2014 року випустила Android Studio як IDE для Android-додатків. AppCode, CLion, DataGrip, GoLand, PhpStorm, PyCharm, Rider, RubyMine, WebStorm та MPS - це інші середовища програмування на основі фреймворку IntelliJ.

Доповнення коду на основі аналізу контексту, навігація по коду для швидкого доступу до класів та оголошень, реструктуризація коду, налагодження коду та пропозиції щодо усунення проблем - це лише деякі з можливостей, які пропонує IntelliJ IDEA для допомоги програмістам.

Поряд з підтримкою систем контролю версій, таких як Git, Mercurial, Perforce та SVN, IDE також пропонує інтеграцію з інструментами збірки та пакування, такими як grunt, bower, gradle та SBT. Крім того, IntelliJ IDEA Ultimate має вбудований DataGrip для швидкого доступу до баз даних, включаючи Microsoft SQL Server, Oracle, PostgreSQL, SQLite і MySQL.

Екосистема плагінів, що підтримується IntelliJ, дозволяє користувачам розширювати функціональність IDE. Плагіни можна знайти та встановити за допомогою вбудованого інструменту пошуку та встановлення плагінів або відвідавши веб-сайт репозиторію плагінів IntelliJ Plugin Repository. Існують окремі репозиторії плагінів для версій Community та Ultimate, і кожен з них має великий вибір плагінів (понад 3000 станом на 2019 рік).

3.1.3 Огляд додаткових інструментів

Рекомендована мова розмітки для сторінок, які відображаються у веб-браузері, називається мовою розмітки гіпертексту, або HTML. У цьому можуть допомогти мови сценаріїв, такі як JavaScript, і технології, такі як каскадні таблиці стилів (CSS).

Файли HTML завантажуються з веб-сервера або локального сховища за допомогою веб-браузера, який потім перетворює їх на мультимедійні веб-сторінки. HTML спочатку містить підказки про те, як має виглядати документ, і семантично пояснює структуру веб-сторінки.

Основою HTML-сторінок є компоненти HTML. У дубльовані сторінки зображення та інші об’єкти (наприклад, інтерактивні форми) можна вставляти за допомогою HTML-фреймворку. Щоб описати структурну семантику тексту, таку як заголовки, абзаци, списки, посилання та інші елементи, HTML надає інструменти для створення структурованих документів. Теги, укладені в кутові дужки, використовуються для позначення елементів HTML. Наприклад, теги введення та зображення додають вміст безпосередньо на сторінку. Інші теги, наприклад p>, охоплюють і описують текст документа, і вони можуть мати інші теги як дочірні компоненти. Браузери використовують теги HTML для розшифровки вмісту сторінки, а не для його відображення.

HTML дозволяє вставляти програми, такі як JavaScript, які змінюють функціональність і вміст веб-сторінок. Вигляд і організація матеріалу визначаються активацією CSS. З 1997 року Консорціум Всесвітньої павутини (W3C), який раніше наглядав за підтримкою HTML, а тепер відповідає за стандарти CSS, надає перевагу використанню CSS, а не явному представленню HTML [2]. Відео та аудіо відображаються за допомогою форм HTML або HTML5, переважно з використанням елемента canvas і JavaScript.

Символьні типи даних, посилання на символи, посилання на сутності та теги (і їхні атрибути) є одними з основних частин синтаксису HTML. Хоча деякі теги, як-от img>, позначають порожні елементи і тому є незвичайними, теги HTML найчастіше зустрічаються парами, наприклад h1> і /h1>. Початковий тег є першим тегом у такій парі, тоді як кінцевий тег є другим (також відомий як початковий і кінцевий теги).

Оголошення типу документа HTML, яке запускає рендеринг у стандартному режимі, є ще одним важливим елементом.

Ось ілюстрація традиційного HTML сценарію.

```
<!DOCTYPE html>
<html>
<head>
<title>Це ім'я</title>
</head>
<body>
<h1>Заголовок</h1>
<p>Це є параграф тексту.</p>
<ul>
  <li>Перший елемент списку</li>
  <li>Другий елемент списку</li>
</ul>
</body>
</html>
```

Рисунок 3.1 – HTML сценарій

Веб-сторінка описується в тексті між <html> і </html>, а її видимий вміст знаходиться в тексті між <body> і </body>. Тег визначає поділ сторінки, який використовується для спрощення дизайну. Текст тегу <title> <title> визначає назву сторінки браузера, яка відображається у вкладках і заголовках вікна браузера. Щоб указати метадані сторінки, розмістіть елемент «meta» між «head» і «/head».

Оголошення doctype HTML5 є<!DOCTYPE html>. Якщо рекламу не ввімкнено, різні веб-переглядачі повертаються до «відмінного» режиму.

Елемент

Публікації HTML посилаються на ієрархію елементів HTML. Вони представлені в документі за допомогою тегів HTML <p>, які взяті в кутові дужки.

«Відкриваючий тег» (p) і «закриваючий тег» (/p) описують область елемента в основному загальному випадку. Текстовий вміст елемента, якщо такий є, розміщується між цими тегами.

Між початком і кінцем міток можуть бути додаткові теги міток, які можуть поєднувати мітки та текст. Таким чином наступні (вкладені) елементи позначаються як дочірні елементи батьківського елемента.

Атрибути елемента також можуть міститися в початковому тегу. Вони також вказують додаткові деталі, такі як ідентифікатор розділу, ідентифікатор, який пов’язує інформацію про стиль із представленням документа, а для деяких тегів, як-от img> для вбудовування зображень, — посилання на ресурси зображень у таких документах.

Наприклад: img src="example.com/example.jpg">.

Ви не можете вставляти вміст, текст або інші теги в деякі компоненти, такі як розриви рядків або
. Вони не використовують закриваючий тег і просто потребують порожнього тегу (як відкриваючого). Багато тегів є необов’язковими, зокрема широко використовуваний заключний тег елемента абзацу p>. Відповідно до вимог до контексту та структури, визначених стандартом HTML, браузер HTML або інший проксі-сервер може визначити кінець елемента. Більшість програмістів HTML не знають про ці обмеження, оскільки вони складні.

Як наслідок, звичайна форма елемента HTML: tag attribute1="value1" attribute2="value2">. " вміст " /тег>. Деякі елементи HTML мають тег форми attribute1 = "value1" attribute2 = "value2"> і визначаються як порожні елементи. Такі компоненти, як вбудовані теги «img» і теги br, можуть бути порожніми та не мати вмісту. Імена, які використовуються в тегах, є іменами елементів HTML. Пам’ятайте, що закриваючі теги не є необов’язковими та дозволеними в порожніх елементах, і що їхні назви починаються з скісної риски (/). Значення за замовчуванням застосовується в кожному випадку, якщо атрибут не вказано.

Мета тексту показана за допомогою структурних підказок.

Наприклад, «Гольф» позначається як заголовок другого рівня за допомогою h2> Golf /h2>. Хоча більшість веб-браузерів пропонують стилі за замовчуванням для форматування елементів, структурована розмітка не потребує особливого рендерингу. Каскадні таблиці стилів (CSS) використовуються для вдосконалення стилю вмісту. Незалежно від своєї функції, теги презентації визначають, як має виглядати вміст.

Наприклад, «жирний текст» насправді вказує лише на те, які пристрої виводу мають відображати текст жирним шрифтом (наприклад, пристрої візуального виведення), а які - ні (наприклад, пристрої виведення аудіо, які читають текст вголос). Існують інші елементи, такі як strong>strong text/i> strong >> і em>underlined text/em>, які можуть мати однакові візуальні, але більш семантичні сигнали, ніж b>жирний текст/b> та i>курсивний текст/i >. Останні два аспекти простіше зрозуміти з огляду того, як слухові агенти користувача повинні їх сприймати. Програми зчитування з екрана не передбачають, що назви книг будуть підкреслені, але на екрані такі назви виділяються курсивом, тому вони не збігаються з аналогами для презентації. Відповідно до специфікації HTML 4.0 більшість елементів розмітки презентації було скасовано на користь стилів, створених за допомогою CSS.

Частини документа можна посилати на інші документи за допомогою гіпертекстової розмітки.

Цільова URL-адреса гіперпосилання, створеного елементом прив’язки в документі, визначається властивістю href. HTML-тег a href="https://www.google.com/">, наприклад, слово "Вікіпедія" з’явиться як гіперпосилання після кінцевого оголошення. Вставте елемент img як вміст в елемент an, щоб зображення відображалося як гіперпосилання. Img — це порожній елемент з атрибутами, але без вмісту чи закриваючого тегу, подібний до br. a href="https://example.org"> Зображення src="image.gif" alt="текст опису" width="50" height="50" border="0"> . зробити цей текст характерним.

Атрибути.

Початкова розмітка елемента записується після імені елемента, і більшість атрибутів елемента є парами ім’я-значення, розділеними символом =. Хоча значення в HTML (але не в XHTML) можна залишити без лапок, вони повинні бути взяті в одинарні або подвійні лапки. Значення атрибутів без лапок вважаються небезпечними. На відміну від атрибутів пари ім’я-значення, деякі атрибути впливають на елемент лише тому, що вони включені в його початкову розмітку [6], наприклад, атрибут ismap елемента зображення.

Численні елементи можуть мати такі характеристики:

- атрибут id надає кожному елементу в усьому документі окремий ідентифікатор. Це використовується для ідентифікації елемента, щоб сценарії та таблиці стилів могли змінювати, анімувати або видаляти його вміст чи вигляд. Доданий до URL-адреси сторінки, він надає елементу, як правило, частині сторінки, глобальну унікальну ідентифікацію;

- атрибут class дає вам механізм групування пов’язаних елементів.

Для цього можливе як презентаційне, так і семантичне використання. Наприклад, семантичне використання нотації HTML class="notation"> може бути використано, щоб показати, що будь-які елементи з цим значенням класу підпорядковуються тілу документа. Такі елементи можна об’єднати та представити як виноски на сторінці під час презентації, а не в вихідному коді HTML. Мікроформати семантично використовують атрибути класу. Можна надати декілька значень класу; наприклад, class="notation important" розміщує елемент як у нотації, так і в важливому класі. Властивість стилю може бути використана автором для надання певним елементам властивостей представлення. Використання ідентифікатора елемента або атрибутів класу для вибору елемента з CSS зазвичай вважається найкращою практикою, хоча це може бути надто складним для простого, спеціалізованого або індивідуального стилю. Атрибут title використовується для надання підтекстового опису для елемента. Цей атрибут

зазвичай відображається як спливаюча підказка в браузерах, він визначає вміст природної мови елемента, який може відрізнятися від мови, що використовується в решті частини документа. Як ілюстрація, у тексті, написаному англійською: Anyway, як кажуть у Франції, "c'est la vie," або span lang="fr" .

Значення закодованої інформації над її поданням (виглядом) підкреслюється в семантичному HTML, стилі створення HTML. На додаток до презентаційної розмітки, як-от теги font>, i> і center>, HTML завжди мав семантичну розмітку. Крім того, теги span і div є семантично нейтральними. Використання розмітки презентації HTML для відділення презентації від вмісту не рекомендується з кінця 1990-х років, коли каскадні таблиці стилів почали функціонувати в більшості браузерів.

Мова, відома як CSS, іноді відома як каскадні таблиці стилів, була створена, щоб полегшити створення веб-сторінок, які здаються професійними. Ви можете додавати стилі до веб-сторінок за допомогою CSS. Що ще більш важливо, CSS дає вам змогу це робити, окрім HTML-коду, з якого складається кожна веб-сторінка. Він визначає правильні кольори, шрифти, інтервали та інші елементи дизайну для веб-сторінки. Простіше кажучи, ви можете створити свій веб-сайт як завгодно. CSS дає дизайнерам і розробникам можливість контролювати поведінку елемента, зокрема те, де він відображатиметься у браузері.

Тоді як CSS використовує набори правил, html використовує теги. CSS простий для вивчення та розуміння, але він пропонує ефективний контроль над тим, як представлено HTML-документ.

Чому CSS?

CSS економить час, оскільки документ можна написати один раз і повторно використовувати на кількох HTML-сторінках.

Щоб внести глобальні зміни, просто змініть стиль, і всі елементи на всіх веб-сторінках оновляться автоматично.

CSS вважається чистою технікою кодування, що означає, що пошуковим системам не потрібно «читати» його вміст.

Стилі перевершують HTML: CSS має набагато ширший набір атрибутів, ніж HTML, тому ви можете надати своїй HTML-сторінці значно кращого вигляду, ніж за допомогою атрибутів HTML.

Офлайн-перегляд: використовуючи офлайн-кеш, CSS може кешувати веб-програми локально. Це дозволяє нам переглядати веб-сайти офлайн.

Spring спрощує розробку корпоративних програм Java. Він надає все необхідне для використання мови програмування Java у корпоративному середовищі, включаючи підтримку Groovy і Kotlin як альтернативних мов у JVM і можливість створювати різноманітні архітектури для окремих програм. Починаючи з Spring Framework версії 6.0, потрібна Java 17+.

Spring сумісний із різноманітними сценаріями застосування. У великій організації програми часто існують протягом тривалого періоду часу і повинні працювати на JDK і сервері програм, цикл оновлення яких знаходиться поза контролем розробника. Інші можуть працювати як окремий банк із вбудованим сервером, можливо, у хмарі. Інші можуть бути автономними програмами (наприклад, інтеграція або пакетні операції), які не потребують сервера.

Spring - це проєкт із відкритим вихідним кодом. Він має велику й активну спільноту, яка постійно надає відгуки на основі різноманітних випадків використання в реальному світі. Це дуже довго сприяло розвитку Spring.

Термін «весна» має різні значення залежно від контексту. Його можна використовувати для позначення проєкту Spring Framework, який започаткував всю ініціативу. Згодом на основі Spring Framework було створено додаткові ініціативи Spring. Як правило, кажучи «Весна», мають на увазі всю родину ініціатив.

Цей Framework складається з модулів. Програми можуть вибирати потрібні модулі. Модулі основного контейнера, включаючи модель конфігурації та механізм впровадження залежностей, складають ядро. Крім того, Spring забезпечує

фундаментальну підтримку різноманітних архітектур додатків, таких як обмін повідомленнями, транзакційні дані та постійність, а також Інтернет. Крім того, він складається з веб-фреймворку Spring MVC на основі Servlet і реактивного веб-фреймворку Spring WebFlux.

Примітка щодо модулів: Spring дозволяє розгортати шлях модуля JDK 9 («Jigsaw»). Spring Framework 5 jar містить записи маніфесту «Automatic-Module-Name», які визначають стабільні назви модулів на рівні мови («spring.core», «spring.context» тощо) незалежно від імен артефактів jar (jar мають однакові шаблон іменування з «-» замість «.», наприклад «spring-core» і «spring-context»). Очевидно, що структура Spring продовжує добре працювати на шляху до класів у JDK 8 і пізніших версіях.

3.2 Розробка графічного інтерфейсу користувача

GUI, або графічний інтерфейс користувача (Graphical User Interface), є формою інтерфейсу, що дозволяє користувачам взаємодіяти з комп'ютерною системою за допомогою графічних елементів, таких як кнопки, поля введення, меню, вікна та інші. Він надає користувачам зручний та інтуїтивно зрозумілий спосіб комунікації з програмним забезпеченням, що дозволяє виконувати завдання без необхідності запам'ятовувати складні команди або коди.

GUI стало ключовим компонентом більшості сучасних операційних систем, програм та мобільних пристроїв, оскільки воно полегшує використання та зрозуміння функцій системи для широкого кола користувачів. Воно використовує візуальні елементи, які представляють структуру та поведінку програми, і дозволяє користувачам взаємодіяти з цими елементами за допомогою миші, клавіатури або дотику.

Основні складові GUI включають:

– вікна (windows): вони представляють основні області, в яких відображаються вміст, елементи управління та інформація, вікна можуть бути переміщені, змінювати розмір, мінімізуватися, максимізуватися або закриватися;

– елементи управління (controls): це графічні об'єкти, такі як кнопки, поле введення, перемикачі, списки та інші, які дозволяють користувачам здійснювати взаємодію з програмою, кожен елемент управління має свою функцію та може відрізнятися візуальною поведінкою;

– меню (menus): вони містять список команд або опцій, які користувач може вибрати, меню можуть бути розгорнуті або згорнуті, і вони надають шлях для доступу до різних функцій програми;

– діалогові вікна (dialog boxes): вони використовуються для виведення спеціальних повідомлень, запитів або отримання даних від користувача. діалогові вікна можуть мати поля введення, кнопки підтвердження або відміни, та інші елементи для збору інформації або виконання певних дій;

– взаємодія зі значками (icons): значки є графічними піктограмами, які представляють програми, файли або папки. користувачі можуть взаємодіяти зі значками, натискаючи їх або перетягуючи їх на певні області.

GUI має кілька переваг:

– інтуїтивність та зручність використання: gui дозволяє користувачам взаємодіяти з програмним забезпеченням за допомогою візуальних елементів, що робить процес використання програми більш зрозумілим та легким;

– швидкість навчання: gui зменшує необхідність в запам'ятовуванні складних команд або кодів, що дозволяє користувачам швидко навчитися використовувати програмне забезпечення;

– можливість візуалізації інформації: gui надає можливість відображення складних даних та інформації за допомогою графічних елементів, що полегшує їх сприйняття та аналіз;

– створення привабливого дизайну: gui дозволяє розробникам створювати естетично привабливий та професійний дизайн інтерфейсу, що покращує враження користувачів від використання програми.

Проте, GUI також має свої недоліки:

– потреба в високих обчислювальних ресурсах: gui вимагає більшої кількості ресурсів комп'ютера для відображення графічних елементів, що може впливати на продуктивність системи;

– вимоги до екрану: gui може бути більш вимогливим до роздільної здатності та розміру екрану, що може ускладнювати використання на пристроях з меншими дисплеями;

– велика кількість функцій: іноді gui може надавати занадто багато функцій та опцій, що може призводити до перенавантаження та плутанини для користувачів.

Узагальнюючи, GUI є потужним інструментом для взаємодії з програмним забезпеченням, надаючи зручний та інтуїтивно зрозумілий спосіб комунікації. Воно полегшує роботу з програмами, забезпечує швидке навчання та сприяє візуалізації інформації.

Розроблений графічний інтерфейс користувача представлено на рисунках 3.1-3.6.

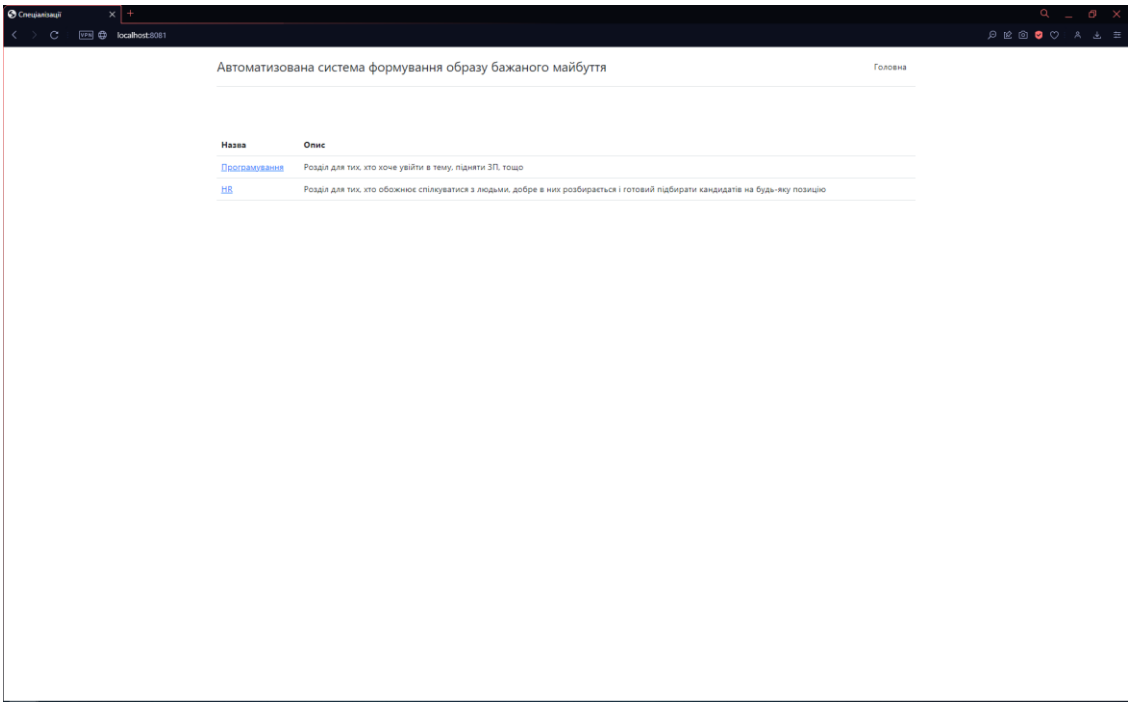

Рисунок 3.2 – Головна сторінка

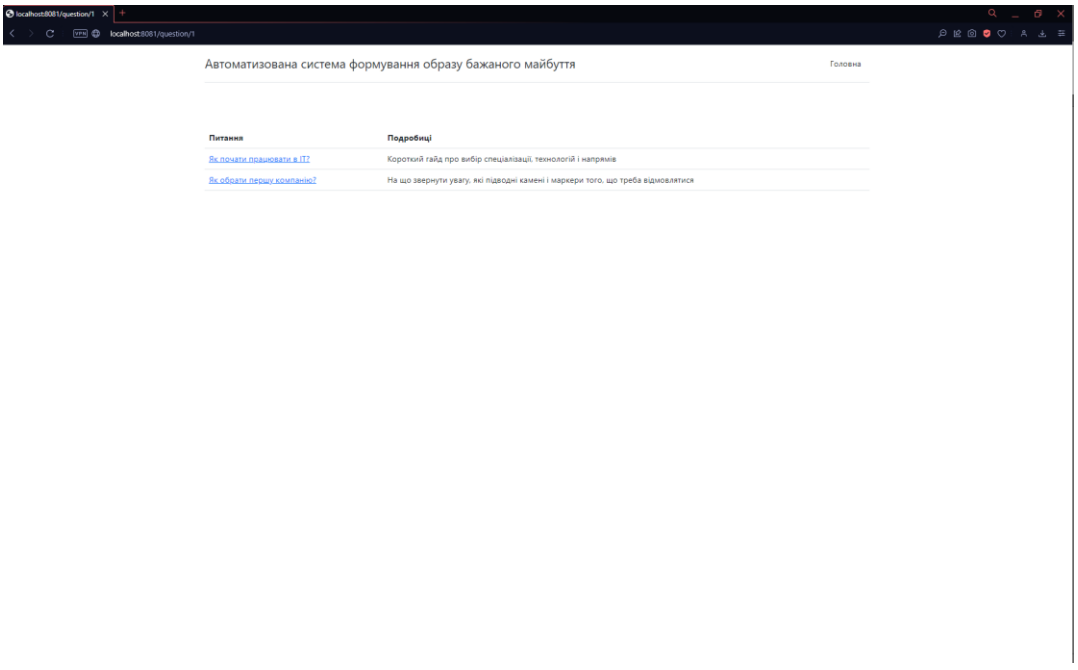

Рисунок 3.3 – Сторінка запитань

Рисунок 3.4 – Сторінка рекомендацій

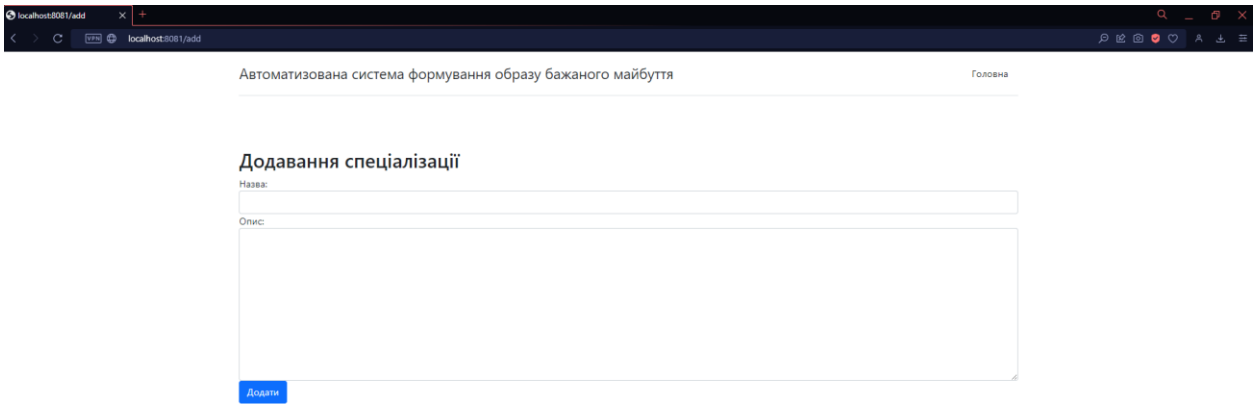

Рисунок 3.5 – Сторінка додавання спеціалізацій

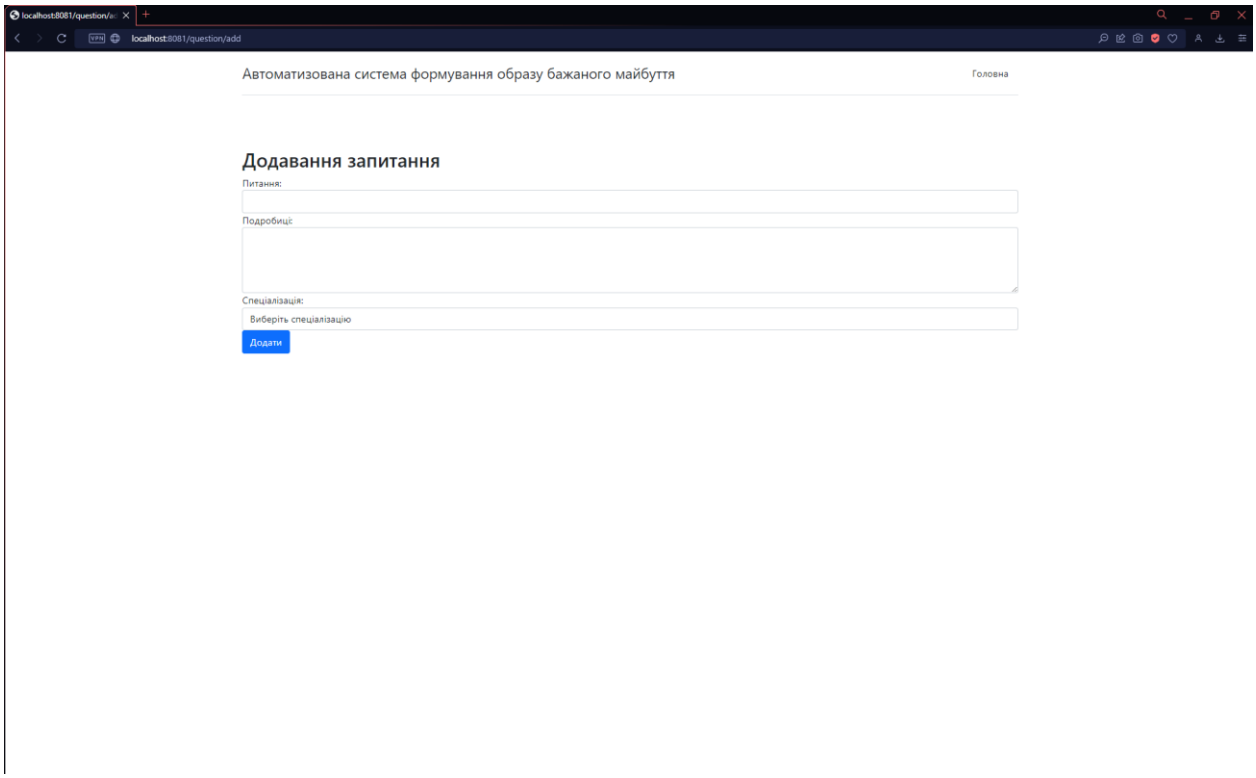

Рисунок 3.6 – Сторінка додавання питань

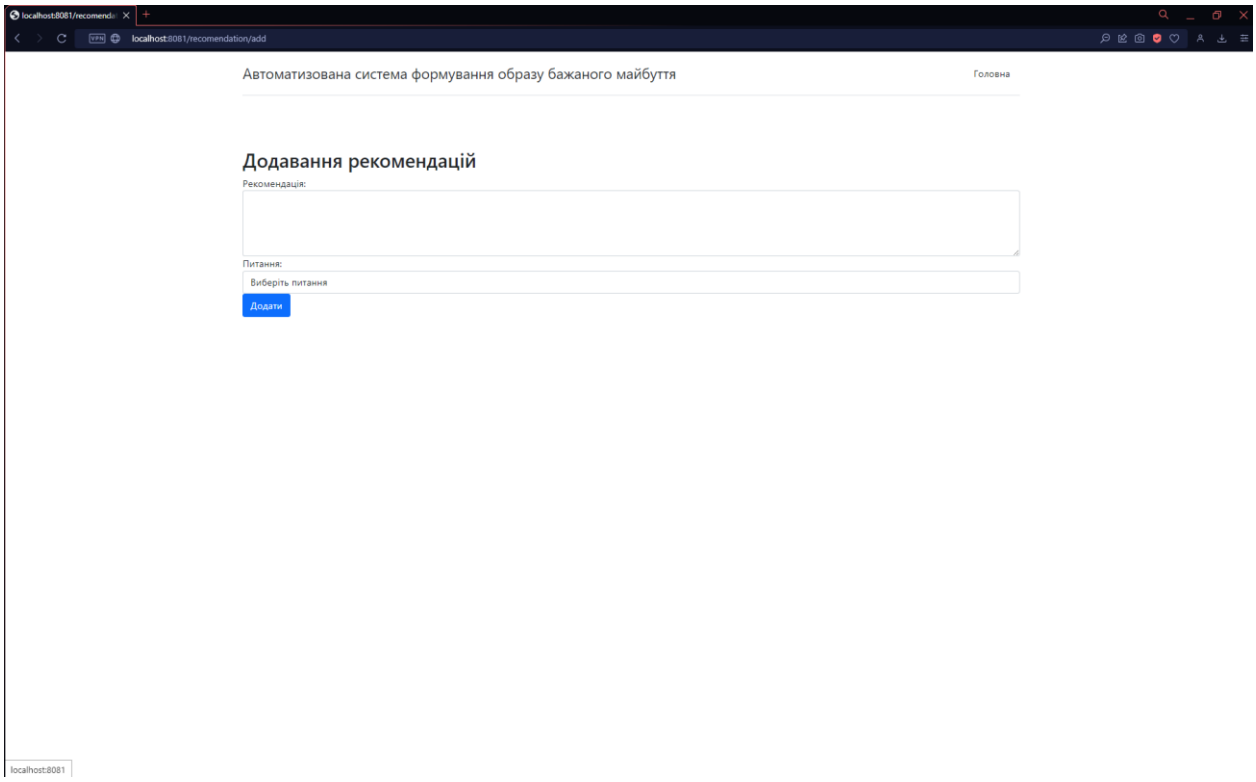

Рисунок 3.7 – Сторінка додавання рекомендацій

3.3 Тестування системи

Тестування програмного забезпечення (software testing) - це процес перевірки програмного продукту з метою виявлення помилок, дефектів та недоліків з метою покращення якості програми. Воно виконується шляхом запуску програми з різними тестовими вхідними даними, виконання послідовності дій і перевірки отриманих результатів.

Тестування програмного забезпечення може бути розділене на декілька типів та методів, включаючи:

– функціональне тестування: перевірка функціональних можливостей програми, тестувальник переконується, що програма працює згідно з вимогами та очікуваннями користувача;

– нефункціональне тестування: перевірка якості програми, що не пов'язана з її функціональністю, наприклад, тестування продуктивності, навантаження, безпеки та інтерфейсу користувача;

– модульне тестування: перевірка окремих модулів або компонентів програми для виявлення дефектів на ранніх етапах розробки. використовуються малий набір тестів;

– інтеграційне тестування: перевірка взаємодії між різними модулями або компонентами програми, мета - виявити дефекти, що можуть виникнути при їх злитті;

– системне тестування: перевірка програмного забезпечення як єдиного цілого для виявлення проблем на рівні системи або поведінки системи в цілому;

– регресійне тестування: повторне тестування програми після внесення змін з метою перевірки, чи не виникли нові дефекти або чи не були пошкоджені існуючі функції.

Одним з важливих видів тестування є тестування "чорного ящика" (black-box testing). Цей підхід зосереджений на перевірці зовнішнього поведінки програми, не

звертаючи увагу на внутрішню структуру чи реалізацію коду. Тестувальник працює з програмою як з "чорним ящиком", не знаючи деталей її реалізації.

Під час тестування чорного ящика використовуються такі методи:

– валідація: перевірка, чи відповідає програма вимогам та очікуванням користувача;

– тестування еквівалентних значень: вибір тестових вхідних даних з кожного класу еквівалентних значень, що представляють однакові стани системи;

– граничний аналіз: вибір тестових вхідних даних, що знаходяться на межі класів еквівалентних значень або в околі межі;

– тестування структури даних: перевірка правильності обробки вхідних даних, таких як масиви, списки, бази даних;

– тестування помилок: перевірка реакції програми на некоректні або недопустимі вхідні дані.

Тестування програмного забезпечення має наступні цілі:

– виявлення дефектів: ідентифікація помилок та проблем, які можуть призвести до некоректної роботи програми;

– підтвердження вимог: перевірка, чи відповідає програма встановленим вимогам та специфікаціям;

– підвищення якості: забезпечення надійності, ефективності та зручності використання програми;

– забезпечення безпеки: виявлення потенційних вразливостей та помилок, що можуть бути використані для злому або зловживання програмою.

Узагальнюючи, тестування програмного забезпечення є важливою складовою процесу розробки програм та допомагає забезпечити їхню якість та надійність. Тестування чорного ящика дозволяє перевіряти зовнішнє поведінку програми без залучення внутрішніх деталей реалізації, використовуючи різні методи, такі як валідація, тестування еквівалентних значень, граничний аналіз та інші.

Основні тест-кейси системи проілюстровано в таблиці 3.1.

Таблиця 3.1 — Основні тест-кейси

№	Тест-кейс	Очікуваний результат	Отриманий результат
1	Додавання нової спеціалізації	Спеціалізація додана, користувач переходить на головну сторінку	Спеціалізація додана, користувач переходить на головну сторінку
2	Додавання нового питання	Питання додається до списку, користувач переходить на головну сторінку	Питання додається до списку, користувач переходить на головну сторінку
3	Додавання нової рекомендації	Рекомендація додана, користувач переходить на головну сторінку	Рекомендація додана, користувач переходить на головну сторінку

Результати тестування свідчать про те, що система повністю працює і готова до використання в реальному світі.

ВИСНОВКИ

У рамках дипломної роботи була розроблена автоматизована система формування образу бажаного майбутнього, яка надає відповіді на часті питання в різних областях роботи. Система реалізована з використанням сучасних технологій програмування та баз даних, що забезпечило швидкість та надійність її роботи. Інтерфейс користувача розроблений зручним та інтуїтивно зрозумілим способом, що дозволяє з легкістю знаходити відповіді на питання.

Результати тестування системи показали, що вона ефективно виконує свої завдання та надає користувачам швидкий та точний доступ до актуальної інформації. Система дозволяє адміністратору вносити та налаштовувати інформацію відповідно до потреб користувачів, що забезпечує її гнучкість та адаптованість до різних сфер діяльності.

Процес розробки системи виявився складним і вимагав глибокого аналізу потреб користувачів, вибору оптимальної архітектури та технологій, а також детального тестування і валідації системи. Однак, завдяки систематичному підходу та використанню сучасних розробницьких методологій, було досягнуто успішного результату.

Застосування розробленої системи може мати значний вплив на підвищення ефективності робочих процесів та прийняття рішень. Вона може бути використана в різних сферах діяльності, від бізнесу до особистого використання, і сприяти автоматизації та оптимізації різних завдань.

Протягом роботи над дипломною роботою виявлено певні обмеження та можливості подальшого розвитку системи. Одним з можливих напрямків розвитку є розширення функціональності системи та додавання нових модулів для покриття більш широкого спектру запитань. Також, вдосконалення інтерфейсу користувача та забезпечення більшої зручності використання можуть підвищити ефективність системи.

Загалом, розроблена система формування образу бажаного майбутнього є корисним інструментом для швидкого та зручного доступу до інформації в різних сферах діяльності. Її впровадження може сприяти підвищенню продуктивності та поліпшенню прийняття рішень. Результати дипломної роботи підтверджують, що розроблена автоматизована система формування образу бажаного майбутнього є потужним інструментом з широким спектром застосування. Вона забезпечує користувачам швидкий та точний доступ до актуальної інформації, сприяє підвищенню ефективності робочих процесів та прийняттю рішень.

Процес розробки системи вимагав значних зусиль і ретельного аналізу потреб користувачів. Використання сучасних технологій програмування та баз даних дало змогу забезпечити швидкість та надійність роботи системи. Інтуїтивно зрозумілий інтерфейс користувача дозволяє з легкістю знаходити відповіді на питання.

Тестування системи підтвердило її ефективність і здатність до швидкого та точного реагування на запити користувачів. Адміністратор системи має можливість налаштовувати та оновлювати інформацію залежно від потреб користувачів, що забезпечує гнучкість та адаптованість системи до різних сфер діяльності.

Однак, розроблена система має певні обмеження. Для подальшого розвитку можна розглядати розширення функціональності та додавання нових модулів, які охоплюватимуть ще більший спектр запитань. Покращення інтерфейсу користувача для забезпечення більшої зручності використання також є потенційним напрямком розвитку, що сприятиме підвищенню ефективності системи.

Загалом, розроблена система формування образу бажаного майбутнього є цінним ресурсом для швидкого та зручного доступу до інформації в різних сферах діяльності. Впровадження системи може сприяти підвищенню продуктивності та

поліпшенню прийняття рішень, що в свою чергу має потенціал позитивно вплинути на результативність та успішність в різних областях роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Tim Berners-Lee". www.w3.org. Archived from the original on 27 September 2021. Retrieved 17 November 2021.
2. "The website of the world's first-ever web server". Archived from the original on 10 June 2017. Retrieved 30 August 2008.
3. Cailliau, Robert. "A Little History of the World Wide Web". Archived from the original on 6 May 2013. Retrieved 16 February 2007.
4. "Internet, Web, and Other Post-Watergate Concerns". University of Chicago. Archived from the original on 20 February 2010. Retrieved 18 September 2010.
5. AP Stylebook [@APStylebook] (16 April 2010). "Responding to reader input, we are changing Web site to website. This appears on Stylebook Online today and in the 2010 book next month" (Tweet). Retrieved 18 March 2019 – via Twitter.
6. "Web Server Survey". Netcraft. Archived from the original on 20 August 2011. Retrieved 13 March 2017.
7. A total number of Websites | Internet live stats Archived 20 July 2017 at the Wayback Machine. internetlivestats.com. Retrieved on 14 April 2015.
8. "Web Server Survey". Netcraft News. Archived from the original on 24 July 2018. Retrieved 17 May 2021.
9. Deon (26 May 2020). "How Many Websites Are There Around the World? [2021]". Siteefy. Archived from the original on 17 May 2021. Retrieved 17 May 2021.
10. "OpenGL ES for the Web". khronos.org. 19 July 2011. Archived from the original on 15 December 2009. Retrieved 1 April 2019.
11. Pete LePage. "Responsive Web Design Basics | Web". Google Developers. Archived from the original on 5 March 2017. Retrieved 13 March 2017.

12. "What Is A Web Application?". stackpath.com. Stack Path. Retrieved 2022-08-15. A web application is a computer program that utilizes web browsers and web technology to perform tasks over the Internet.
13. Liam Tung (2020-06-15). "JavaScript creator Eich: My take on 20 years of the world's top programming language". ZDNet.
14. Davidson, James Duncan; Coward, Danny (1999-12-17). Java Servlet Specification ("Specification") Version: 2.2 Final Release. Sun Microsystems. pp. 43–46. Retrieved 2008-07-27.
15. Jay Hoffmann (2019-03-04). "What Does AJAX Even Stand For?". Retrieved 2021-10-18.
16. Russell, Alex. "Progressive Web Apps: Escaping Tabs Without Losing Our Soul". Retrieved June 15, 2015.
17. Petersen, Jeremy (4 September 2008). "Benefits of using the n-tiered approach for web applications".
18. "Top Tips for Secure App Development". Dell.com. Archived from the original on 2012-05-22. Retrieved 2012-06-22.
19. Multiple (wiki). "Web application framework". Docforge. Archived from the original on 2020-06-20. Retrieved 2010-03-06.
20. "What is Web Development? - Definition from Techopedia". Techopedia.com. Retrieved 2018-12-07.
21. Campbell, Jennifer (2017). Web Design: Introductory. Cengage Learning. p. 27.
22. Northwood, Chris (2018-11-19). The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer. Apress. ISBN 978-1-4842-4152-3.
23. "Longer Bio for Tim Berners-Lee". www.w3.org. Retrieved 2022-04-27.
24. Rainer, R. Kelly and Cegielski, Casey G. (2009). "Introduction to Information Systems: Enabling and Transforming Business, 3rd Edition" Archived 2010-06-28 at the Wayback Machine.

25. Kroenke, David (2008). Using MIS – 2nd Edition.
26. Lindsay, John (2000). Information Systems – Fundamentals and Issues. Kingston University, School of Information Systems.
27. Dostal, J. School information systems (Skolni informacni systemy). In Infotech 2007 – modern information and communication technology in education. Olomouc, EU: Votobia, 2007. s. 540 – 546. ISBN 978-80-7220-301-7.
28. O'Leary, Timothy and Linda. (2008). Computing Essentials Introductory 2008. McGraw-Hill on Computing2008.com.
29. Imperial College London – Information Systems Engineering degree – Information Systems Engineering.
30. Sage, S.M. "Information Systems: A brief look into history", Datamation, 63–69, Nov. 1968. – Overview of the early history of IS.

ДОДАТОК А

Лістінг програмного коду

```
package com.joreijarr.studycontrol.controllers;

import com.joreijarr.studycontrol.models.Question;
import com.joreijarr.studycontrol.models.Specialization;
import com.joreijarr.studycontrol.models.Recomendation;
import com.joreijarr.studycontrol.repo.SpecializaionRepository;
import com.joreijarr.studycontrol.repo.QuestionRepository;
import com.joreijarr.studycontrol.repo.RecomendationRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.ArrayList;
import java.util.List;

@Controller
public class startController {
```

```
@Autowired
public RecomendationRepository recomendationRepository;

@Autowired
public SpecializaionRepository specializaionRepository;

@Autowired
public QuestionRepository questionRepository;




@GetMapping("/")
public String index(Model model)
{
   model.addAttribute("title", "Спеціалізації");
   model.addAttribute("specializations", specializaionRepository.findAll());
   return "index";

}

@GetMapping("/question/{id}")
public String question(Model model, @PathVariable Long id)
{
   model.addAttribute("questions",
questionRepository.findBySpecializationId(id));
   return "questions";
```

```
    }

    @GetMapping("/recomendation/{id}")
    public String recomendation(Model model, @PathVariable Long id)
    {
       model.addAttribute("recomendation",
recomendationRepository.findByQuestionId(id));
       return "recomendation";
    }



    @GetMapping("/add")
    public String specializationAdd_view(Model model)
    {
       return "specialization_add";
    }


    @GetMapping("/question/add")
    public String questionAdd_view(Model model)
    {
       model.addAttribute("specializations", specializaionRepository.findAll());
       return "questions_add";
    }

    @GetMapping("/recomendation/add")
    public String recomendationAdd_view(Model model)
```

```
{
   model.addAttribute("questions", questionRepository.findAll());
   return "recomendation_add";
}


@PostMapping("/add")
public String specializationAdd(Model model, @RequestParam String name,
                                @RequestParam String description)
{
   Specialization specialization = new Specialization(name, description);
   specializaionRepository.save(specialization);
   return "redirect:/";
}


@PostMapping("/question/add")
public String questionAdd(Model model, @RequestParam String question,
                                @RequestParam String details,
                                @RequestParam Long specialization)
{
   Question questionObj = new Question(specialization, question,details);
   questionRepository.save(questionObj);
   return "redirect:/";
}


@PostMapping("/recomendation/add")
```

```
    public String recomendationAdd(Model model, @RequestParam String recommendation,
                                   @RequestParam Long question)
    {
      Recomendation recomendationObj = new Recomendation(question, recommendation);
      recomendationRepository.save(recomendationObj);
      return "redirect:/";
    }




}
package com.joreijarr.studycontrol.models;

import javax.persistence.*;

@Entity
public class Question {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long id;
  public Long specializationId;
  public String questionName;
  public String questionDescription;
```

```
public Question() {
}

public Question(Long speicalizaionId, String questionName, String questionDescription) {
    this.specializationId = speicalizaionId;
    this.questionName = questionName;
    this.questionDescription = questionDescription;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public Long getSpecializationId() {
    return specializationId;
}

public void setSpecializationId(Long specializationId) {
    this.specializationId = specializationId;
}
```

```
    public String getQuestionName() {
        return questionName;
    }

    public void setQuestionName(String questionName) {
        this.questionName = questionName;
    }

    public String getQuestionDescription() {
        return questionDescription;
    }

    public void setQuestionDescription(String questionDescription) {
        this.questionDescription = questionDescription;
    }
}
package com.joreijarr.studycontrol.models;


import javax.persistence.*;

@Entity
public class Recomendation {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long id;

    public Long questionId;
```

```
@Column(length = 65555)
public String recomendationText;

public Recomendation() {
}


public Recomendation(Long questionId, String recomendationText) {
   this.questionId = questionId;
   this.recomendationText = recomendationText;
}

public Long getId() {
   return id;
}

public void setId(Long id) {
   this.id = id;
}

public Long getQuestionId() {
   return questionId;
}

public void setQuestionId(Long questionId) {
   this.questionId = questionId;
}
```

```
    public String getRecomendationText() {
        return recomendationText;
    }

    public void setRecomendationText(String recomendationText) {
        this.recomendationText = recomendationText;
    }
}
package com.joreijarr.studycontrol.models;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Specialization {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long id;
    public String name;
    public String description;

    public Specialization() {
    }
```

```
public Specialization(String name, String description) {
    this.name = name;
    this.description = description;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}
```

}