Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“ Дослідження та програмна реалізація Web-додатка для збереження спогадів з застосуванням фреймворка Spring ”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-23М
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Бондаревський С.О.
«____» _____ 2024 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Кислун О.А.
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *магістр*
Галузь знань _12 _____*"*Інформаційні технології*"*_____
Спеціальність 123 *"*Комп'ютерна інженерія*"*
Освітньо-професійна (освітньо-наукова) програма _*"*Комп'ютерна інженерія*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« » 2024 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Бондаревський Сергій Олександрович
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація Web-додатка збереження спогадів з застосуванням фреймв Spring*

2. Керівник роботи *Кислун Олег Андрійович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 19-13 від 07.08.2024 року

3. Строк подання студентом роботи до захисту *02.12.2024р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробк дослідження та програмна реалізація Web-додатка для збереження спога застосуванням фреймворка Spring*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Маркетингове та економічне обґрунтування ІТ проекту.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та тех безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Доренська А.О.	05.10.2024	14.11.2
Охорона праці	Марченко К.М.	06.10.2024	16.11.2

7. Дата видачі завдання « » 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Прим
1.	Аналіз існуючих систем	10.10.2024р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2024р.	
3.	Розробка моделі компонента	20.10.2024р.	
4.	Розробка структур даних	25.10.2024р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2024р.	
6.	Програмування алгоритмів	10.11.2024р.	
7.	Розрахунок економічної ефективності	13.11.2024р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2024р.	
9.	Оформлення ПЗ	17.11.2024р.	
10.	Попередній захист роботи	02.12.2024р.	

Дата видачі завдання
« » 2024р.

Підпис керівника
_____ Кислун О.А.
(прізвище та ініціали)
Завдання прийнято до виконання
« » 2024 р.

Підпис здобувача
_____ Бондаревський С.О.
(прізвище та ініціали)

АНОТАЦІЯ

Бондаревський С.О. Дослідження та програмна реалізація Web-додатка для збереження спогадів з застосуванням фреймворка Spring. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для Web-додатка для збереження спогадів з застосуванням фреймворка Spring.

Метою роботи є дослідження та висвітлення методик правильного проектування та поетапного створення Web систем за допомогою Spring Framework на прикладі додатку для збереження спогадів (надалі Memories).

Об'єктом дослідження є правильні практики створення веб-систем на базі Spring Framework та існуючі методи взаємодій з ними.

Результат роботи – розроблено програмне забезпечення WEB системи збереження спогадів людини на базі Spring Framework. Отриманні результаті можна також використовувати як довідник при створенні веб-систем з використанням Spring, для висвітлення можливостей окремих модулів Spring для роботи з веб-оточенням.

В ході виконання даної магістерської роботи було: визначено правильні методи проектування веб-систем на базі Spring Framework та на прикладі додатку Memories; висвітлено кожний етап при створені web-систем з використанням Spring Framework у послідовності нарощування функціональності без значних редагувань існуючого проекту; описано можливості Spring Framework при використанні одразу декількох модулів Spring Framework - MVC, Rest, Data, Security на прикладі додатку Memories.

Ключові слова: SPRING FRAMEWORK, база даних, JAVA, Memories, Android, HTTP

.

ABSTRACT

Bondarevsky S.O. Research and software implementation of a Web application for saving memories using the Spring framework. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for a Web application for saving memories using the Spring framework.

The purpose of the work is to research and highlight the methods of proper design and step-by-step creation of Web systems using the Spring Framework using the example of an application for saving memories (hereinafter Memories).

The object of the study is the correct practices of creating web systems based on the Spring Framework and the existing methods of interaction with them.

The result of the work is the development of WEB software for the preservation of human memories based on the Spring Framework. The results obtained can also be used as a reference when creating web systems using Spring, to highlight the capabilities of individual Spring modules for working with the web environment.

In the course of this master's thesis: the correct methods of designing web systems based on the Spring Framework and on the example of the Memories application were determined; each stage of creating web systems using the Spring Framework in the sequence of increasing functionality without significant editing of the existing project is covered; described the capabilities of the Spring Framework when using several Spring Framework modules - MVC, Rest, Data, Security using the Memories application as an example.

Keywords: SPRING FRAMEWORK, database, JAVA, Memories, Android, HTTP

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

API	–	Application Programming Interface
MVC	–	Model-view-controller
JDBC	–	Java Database Connectivity
АОП	–	Аспектно-орієнтоване програмування
БД	–	База даних
CMS	–	Content Managment System

ВСТУП

Актуальність теми. Spring — один із найсучасніших фреймворків у розробці програмного забезпечення на Java. Це не просто фреймворк, а ціла екосистема, що включає в себе кілька різних фреймворків, таких як Spring MVC, Spring Security, Spring Data, Spring Integration та інші. Кожен із цих фреймворків має своє спеціалізоване застосування. Наприклад, Spring MVC використовується для розробки веб-додатків, Spring Data спрощує роботу з базами даних і обробку даних, а Spring Security відповідає за захист додатків.

Завдяки використанню Spring Framework програміст може більшу увагу приділяти розробці логіки додатку, замість кодування низькорівневих процесів, як зв'язок з базою даних через JDBC API, чи використання Servlet API для створення(генерації) веб-сторінок та інших засобів. Але вони все ще актуальні – вони використовуються при розробці, але у вузьких місцях, де, можливо, потрібна гнучка обробка ситуацій, не передбачених у Spring Framework. Але навіть у таких ситуаціях Spring пропонує своє рішення цієї проблеми, наприклад, у випадку з JDBC - це JDBC Template.

Взагалі – Spring Framework переважно застосовується для розробки складних бізнес-додатків і спрощення створення корпоративних (J2EE) рішень. Він орієнтований на використання простих JavaBean-об'єктів — звичайних Java-класів, які не успадковуються від інших класів і, в кращому випадку, лише реалізують певний інтерфейс. Spring Framework – це модульний фреймворк, а це означає, що кожен із модулів можна використовувати незалежно від інших(окрім Spring Core).

Ще одна перевага Spring Framework - це велика спільнота. По-перше, це означає, що баги виправляються за досить короткий час, а по-друге – можна задавати питання до інших програмістів, які

використовують Spring, і тобі буде надана відповідь. На даний момент їх навіть не потрібно задавати, тільки гарно пошукати, більша кількість відповідей вже є у інтернеті, тому ще розробка фреймворка почалася у 2003 році. А по-третє – завдяки тому, що Spring з відкритим кодом, то розвивати Spring Framework може кожен бажаючий і нерідко це бувають крупні компанії, які використовуюсь Spring для створення свого програмного забезпечення.

Тема роботи у більшій частині стосується веб-оточення, тобто більша увага буде приділена Spring MVC, тому що цей модуль дає змогу зручно програмувати Web додатки.

Мета й завдання дослідження. Метою роботи є дослідження та висвітлення методик правильного проектування та поетапного створення Web систем за допомогою Spring Framework на прикладі додатку для збереження спогадів (надалі Memories)

Для досягнення мети роботи поставлені наступні **задачі**:

– визначити правильні методи проектування веб-систем на базі Spring Framework та на прикладі додатку Memories;

– висвітлити кожний етап при створені web-систем з використанням Spring Framework у послідовності нарощування функціональності без значних редагувань існуючого проекту;

– описати можливості Spring Framework при використанні одразу декількох модулів Spring Framework - MVC, Rest, Data, Security на прикладі додатку Memories.

Об'єктом дослідження є правильні практики створення веб-систем на базі Spring Framework та існуючі методи взаємодій з ними.

Предметом дослідження є Spring Framework як засіб створення веб-систем.

Методи дослідження базуються на методах інтеграційного тестування, тестування безпеки, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати: досліджено основні методи та можливості використання Spring Framework, як потужного інструменту для створення гнучких і безпечних додатків у різних галузях, включаючи розподілені системи та високонавантажені корпоративні рішення;

- із використанням JPA та Spring Data JPA реалізовано рівень даних додатку;

- створена взаємодія з веб системою по HTTP протоколу через розробку контролерів Spring MVC;

- розроблена система безпеки додатку із використанням Spring Security;

- створено клієнт для відображення даних від веб системи на базі платформи Android.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно реалізувати клієнт-застосунок з веб-системою на базі Spring Framework.

Достовірність наукових результатів підтверджена теоретичними дослідження та математичними розрахунками, даними отриманими при тестуванні.

Робота апробована на Всеукраїнській науково-практичній on-line конференції “Проблеми енергоефективності та автоматизації в промисловості та сільському господарстві”, яка відбулася 13-14 листопада 2024 року у ЦНТУ м. Кропивницький.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи збереження спогадів є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Розробка веб-додатку – це процес створення веб-сторінок або сайтів. Веб-сторінки створюються з використанням HTML, CSS і JavaScript і т.д. Сторінки можуть статичними (текст або інформація не змінюються протягом перегляду) або динамічними (дані можуть оновлюватися після відправки форми, довантажувати інформацію по типу пагінації; це здійснюється за допомогою request з JavaScript).

Перед створенням конкретного веб-продукту відбувається комплексний аналіз, що визначає критерії, яким він має відповідати. Він включає такі етапи:

- визначення цілей та задач;
- розробка структури;
- HTML-верстка;
- програмування та контроль якості;
- тестування окремих частин і в цілому;
- запуск та оптимізація.

Продукт надає можливість занесення зображень (спогадів) до бази даних (для зберігання та подальшого відображення), виділяючи їхнє місцерозташування. Для кожного зображення потрібно ввести назву і прикріпити файл в інтерфейсі. Файл потім можна завантажити чи замінити.

Усі об’єкти будуть зберігатися в одній базі. У середині буде маркер, при натисканні на який показуватимуться назва і можливість вибору дій над файлами.

1.2 Область застосування

Щодо вебу, то ця галузь стає дедалі більш схожа на реалізацію операційної системи з багатою кількістю додатків, а браузер – це графічний клієнт для взаємодії с операційною системою, але не тільки він. У процесі розробки ми будемо використовувати андроїд платформу як клієнт, але який буде отримувати усю необхідну інформацію через протоколи мережі, тобто веб.

Отже, веб не прив'язує нас до окремої тематики, список проблем, які можна вирішити з його використанням не обмежений. Це, по-перше, означає, що Spring Framework можна використовувати для створення рішень для різноманітних галузей, а по-друге – розробка додатку Memories вирішує тільки поставленні задачі, але якщо прибрати логіку додатку, то залишиться каркас, модифікація якого дасть смогу створювати додатки для будь-яких галузей, та, авжеж, завдяки зручній взаємодії модулів Spring Framework можна легко інтегрувати інші модулі для підвищення функціональності створюємого додатку.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Веб-системи - це потужні веб-додатки, які складаються з декількох (або чималої кількості) веб-програм, котрі мають систему розмежування прав доступу та призначенні для надання персоналізованого доступу до веб-ресурсу великій кількості користувачів.

Види веб-систем

Веб додатки та веб системи розподіляються на наступні види:

– рахівники;
– рейтинги;
– форми відправки повідомлень;
– форми реєстрації;
– гостьові книги;
– форуми;
– форми завантаження (upload);
– системи голосування;
– пошукові системи (по сайту чи по Інтернет);
– движки сайту;
– Content Managment System (CMS);
– банерні движки та системи (локальні та глобальні);
– веб-пошта;
– персоналізовані веб-системи різного спрямування, які надають комплекс послуг свої користувачам.

Основні вразливості

До основних вразливостей вищенаведеної класифікації веб додатків та веб-систем відносяться:

– повна відсутність перевірки вхідних даних (у веб-формах будь-яких систем) або тільки часткова перевірка даних;

– некоректна обробка вхідних даних (нульовий байт, символи рівня директорій);

– переповнення буферу;

– необережна робота програми з файлами, у випадку коли ім’я файлу передається програмі ззовні (GET або POST);

– не врахування особливостей GET та POST запитів;

– некоректна робота з паролями (під час зберігання, передачі та обробки);

– неправильні права доступу;

– неправильні права програм на сервері;

– не врахування особливостей роботи програм завантаження файлів на сервер;

– некоректна логіка роботи веб-програми, яка при деяких допустимих вхідних даних приводить до непередбачуваних наслідків;

– виведення інформації при помилках програми або доступу до бази даних, коли виводиться додаткова службова інформація, не призначена для сторонніх очей;

– некоректна робота за базами даних (паролі, виведення службової інформації, завищена кількість запитів до бд);

– вразливості недостатньої обробки вхідних даних при роботі з бд (SQL-injections);

– неоптимізований програмний код, котрий приводить до значних навантажень на веб-сервер (при своїй звичайній роботі та особливо у випадку збою при передачі некоректних вхідних даних);

– вразливість веб додатків та систем до DoS та DDoS атак.

За 12 років свого існування Spring Framework досяг значних

результатів. Сучасні розробки Spring Framework ведуться для різноманітних напрямків: обробка даних у великому обсязі; сучасні методи взаємодії з веб-середовищем; зручне створення додатків для мобільних телефонів та дистанційна взаємодія з ними; створення сучасних інструментів для проектування, кодування, відлагодження додатків; зручна та гнучка взаємодія з середовищем, де додаток буде працювати, наприклад, з різноманітними операційними системами; зручна взаємодія із соціальними мережами для впровадження у додатки; інструменти для створення розподілених систем (наприклад, управління конфігурацією, інтелектуальна маршрутизація, мікро-проксі, глобальне блокування, розподіленні сесії); створення рівня безпеки для додатку завдяки наявності великої кількості готових рішень для використання у реальних проектах; та інші.

На даний момент Spring Framework використовується з таких компаніях як Hulu, CoreLogic, Philips, BMW, Warner Music Group, Cisco, Tinkoff Bank. Виходячи з назв компаній, можна зробити висновок, що модулі Spring Framework використовуються у різних галузях. [1]

Розглянемо сучасну структуру Spring Framework [2]:

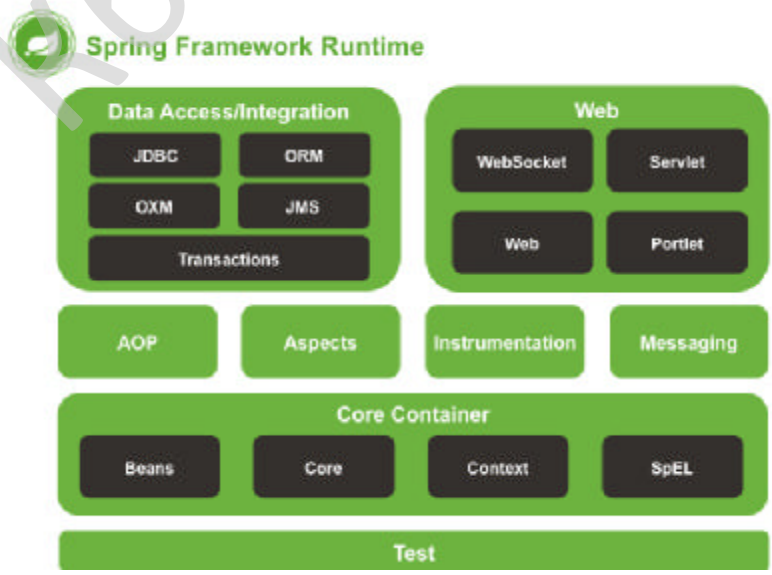

Рисунок 2.1 – Модулі Spring Framework

У ядрі Spring Framework лежать такі основні модулі [2]:

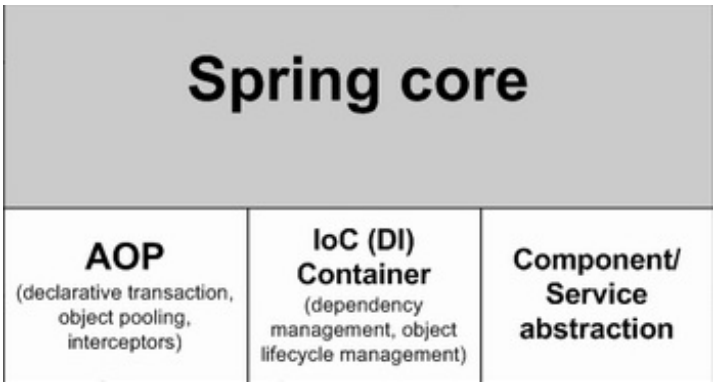

Рисунок 2.2– Ядро Spring Framework

Розглянемо основні модулі, які будуть використані при розробці додатку Memories (деякі з них будуть використовуватись побічно).

Інверсія управління(Inversion of control, IoC) – це принцип організації програми, згідно з яким керування потоком виконання програми здійснюється окремим модулем (як правило це частина програмного каркасу), що дозволяє зменшити зв'язність між іншими модулями програми. IoC входить у п'ятірку найважливіших принципів SOLID. Найпопулярнішою реалізацією IoC є **Dependency Injection Principle** (принцип впровадження залежностей). Dependency Injection використовується в багатьох фреймворках, вони називаються IoC контейнери [3]. У Spring Framework конфігурація компонентів додатків і управління життєвим циклом об’єктів Java, здійснюється головним чином через інверсію управління.

Spring IoC дає можливість не тільки налаштовувати компоненти додатків, але також створювати залежності у бізнес логіці. Тобто залежності створює не бізнес сутність, а конфігураційний файл, або за допомогою Java анотацій. Конфігурацію Spring Framework також можна виконати за допомогою анотацій Java.

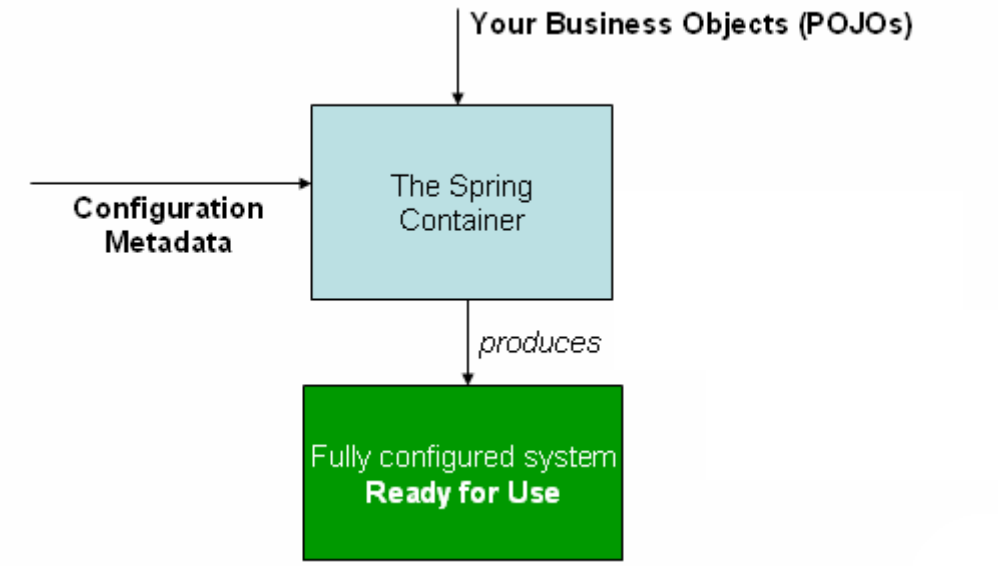

Рисунок 2.3 - Робота контейнеру Spring IoC

Стосовно до легковажних контейнерів, основна ідея цього паттерну полягає в усуненні залежності компонентів або класів додатку від конкретних реалізацій допоміжних інтерфейсів і делегування повноважень з управління створенням потрібних реалізацій IoC контейнеру. IoC контейнер відповідає за створення потрібної реалізації проблеми Постачальника для Споживача [4].

Головні переваги використання IoC контейнерів:

- можливість управління залежностями між бізнес сутностями;
- спрощення повторного використання класів або компонентів;
- спрощення процесу модульного тестування.

Одним з ключових компонентів Spring є AOP. Навіть не використовуючи AOP безпосередньо ви, швидше за все, зіткнетеся з ним опосередковано.

Основні завдання, які вирішуються за допомогою AOP в Spring:

- декларативне управління транзакціями;
- організація пулів об'єктів;
- написання власних аспектів.

Найбільш зручний і гнучкий спосіб управління транзакціями це декларативне управління транзакціями. Декларативне управління

транзакціями позбавляє код від залежності від фреймворку або конкретного механізму управління транзакціями [5].

AOP допомагає усунути дублювання коду. Java - це об'єктно-орієнтована мова програмування, яка дозволяє створювати ієрархії взаємозалежних об'єктів. Але як бути, якщо один і той же код використовується в різних ієрархіях об'єктів? Будь-який додаток можна розглядати з двох позицій. З точки зору функціональності окремих класів (core concerns) та функціоналу, що охоплює увесь додаток (crosscutting concerns). Модулі, що керують функціональністю усього додатку - називаються аспектами. В Spring аспекти реалізуються через Advisors і Interceptors.

Аспекти можуть вирішувати такі завдання: логування, кешування, управління безпекою, транзакції і т.д. [5]. Введення додаткового шару інтерцепторів(перехоплючів) розмежовує відповідальність між основним кодом і допоміжними завданнями. В результаті спрощується супровід коду і його повторне використання. Інтерцептори конфігуруються незалежно, вони можуть використовуватися в інших проектах, в свою чергу об'єкти можуть конфігуруватися з різними інтерцепторами.

Модель-вид-контролер (англ. Model-view-controller, MVC) - архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Цей шаблон поділяє систему на три частини: *модель даних*, *вигляд даних* та *керування*. Застосовується для відокремлення даних (модель) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача [6].

Мета шаблону - гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах

призводить до певної впорядкованості їх структури і робить їх зрозумілішими завдяки зменшенню складності.

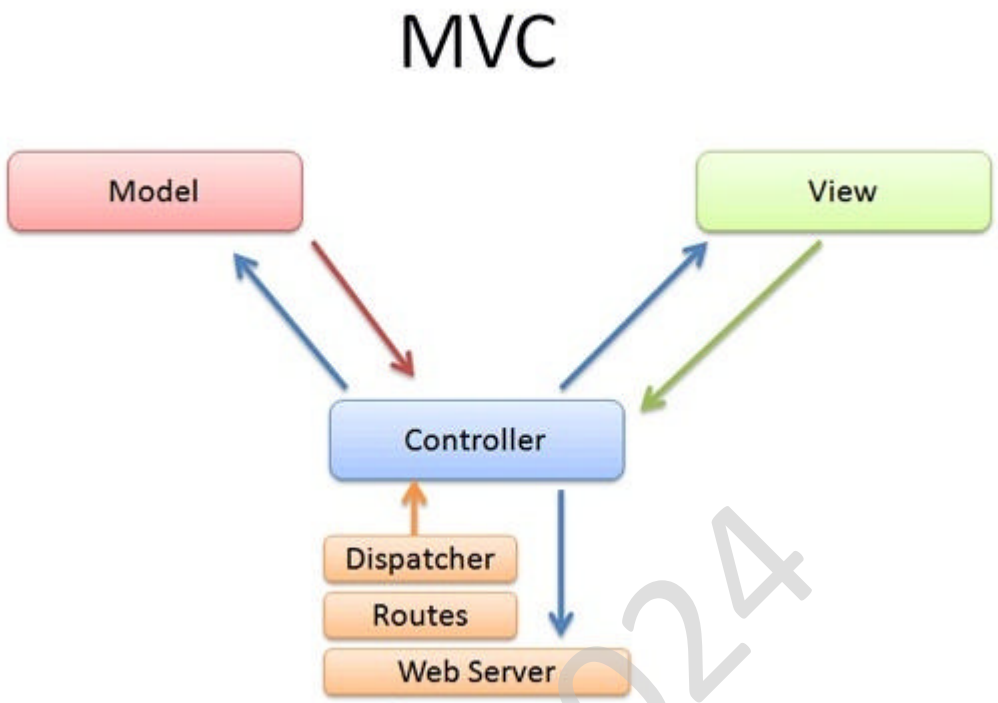

Рисунок 2.4 - Діаграма взаємодії між компонентами шаблону

«Відкритий для розширення ...» ключовий принцип дизайну в Spring MVC і в основі усього Spring лежить «відкритий для розширення, закритий для модифікації» принцип [6].

Деякі методи в основних класів Spring MVC відзначені як final. Як розробник, ви не можете перевизначити ці методи для створення своєї поведінки.

Ви не можете додати свою поведінку до final методів, коли ви використовуєте Spring MVC. Наприклад, ви не можете перевизначити поведінку методу AbstractController.setSynchronizeOnSession.

В Spring MVC ви можете використовувати будь-який об'єкт в якості команди або форм-бек (при заповненні будь-якої форми буде лежати об’єкт з необхідною бізнес логікою) об'єкта. Вам не потрібно реалізовувати спеціальний інтерфейс або базовий клас. Зв'язування даних Spring є дуже гнучким: наприклад, невідповідність даних до деякого шаблону або до

будь-якого правилу буде вважатися як помилка валідації, які можна виправити, а не як системні помилки. Таким чином, вам не потрібно дублювати властивості бізнес-об'єктів при обробці даних від користувачів у веб-обробниках, або писати код для обробки не типізованих полів (class field) у вашому об'єкті, обробляти неприпустимі вхідні дані чи конвертувати їх правильно. Замість цього, потрібну поведінку можливо прив’язати до кожного класу бізнес логіки за допомогою анотацій, наприклад.

Система подачі видів (views) Spring є надзвичайно гнучкою. Контролер, як правило, відповідає за підготовку мапи(Map) моделі з даними і вибравши ім'я виду(view), але він також може написати безпосередньо в потік відповіді (response stream) і завершити запит. Система подачі видів(views) досить гнучко налаштовується через розширення файлу або через заголовок Accept типу контенту, через імена бінів, файл властивостей, або навіть через власну реалізацію ViewResolver. Модель (M у MVC) є інтерфейс мапи, яка дозволяє повністю абстрагуватися від вибраної технології для відображення видів(views). Існує можливість інтегрувати безпосередньо з шаблонів на основі технологій відтворення (rendering), таких як JSP, Velocity і Freemarker або безпосередньо генерувати XML, JSON, Atom, і багато інших типів контенту. Мапа моделі просто перетворюється у відповідному форматі, наприклад у атрибути запиту при використанні JSP, або у модель шаблону при використанні Velocity [6].

Spring MVC(модель-вид-контролер) фреймворк розроблено навколо сервлету *DispatcherServlet*, який відсилає запити обробникам, які містять у конфігураційних файлах розпізнавачів видів (view resolvers), локалі, часові пояси та також підтримку завантаження файлів. Оброблювач за замовчуванням основується на *@Controller* та *@RequestMapping* анотаціях, пропонуючи широкий спектр гнучких методів для обробки. З введенням Spring 3.0, механізм *@Controller* також дозволяє створювати RESTful веб-

сайти і додатки, через *@PathVariable* анотації та інші можливості.

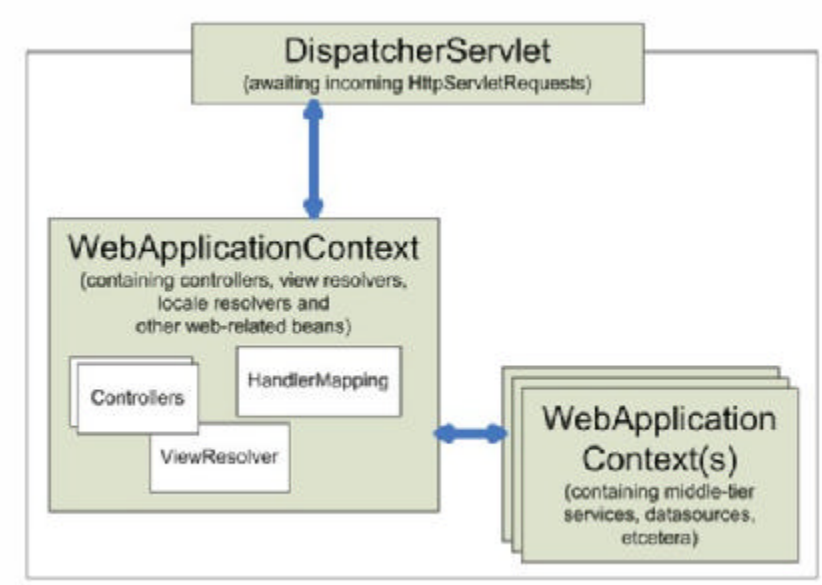

Рисунок 2.5 – Взаємодія DispatcherServlet із іншими компонентами системи

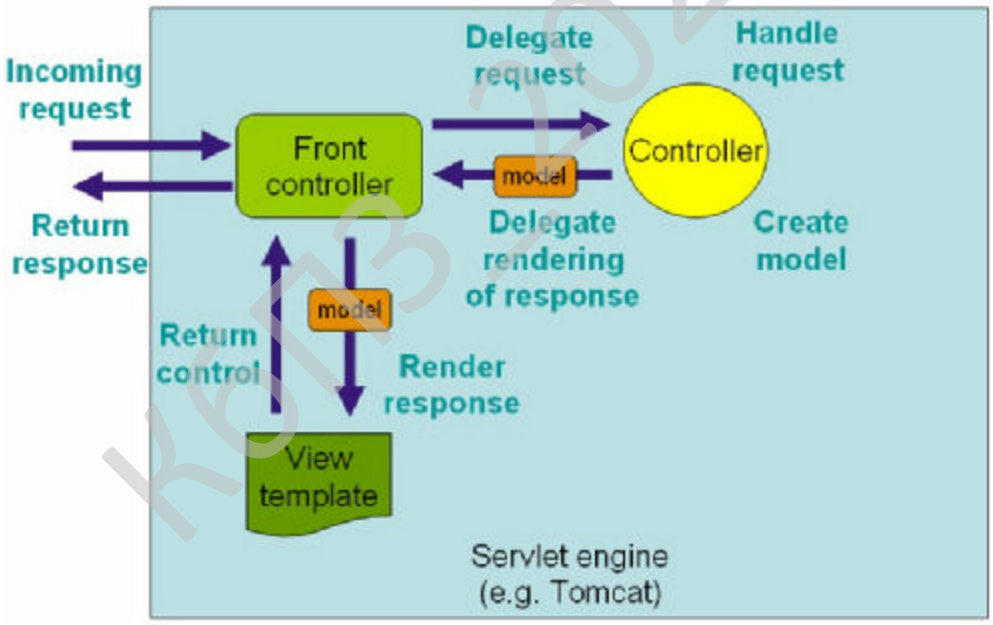

Рисунок 2.6 – Як працює шаблон MVC у Spring Framework

Розглянемо рисунок 2.6. Усі вхідні запити та вихідні відповіді проходять через *DispatcherServlet.* Потім усі запити передаються до *Front controller*, який виконую пошук відповідного до запиту контролера. У контролері відбувається отримання даних з бази даних через сервіси доступу до бази даних – що і буде складати нашу модель, але зазвичай це клас *Model* або звичайна Map мапа, куди і потрібно вставляти отримані

дані. Також в моделі потрібно вказати який вид потрібно використовувати для показу даних – це можна зробити: повернути із методу контролера строку із іменем виду, або, якщо використовується для моделі клас *Model*, то визвати метод *setViewName*. Потім *Front controller* виконує пошук *View resolver* – це спеціальний клас, який і вирішує, якому із видів передавати модель з даними. Потім дані передаються до потрібного виду і відповідь повертається до *Front controller'y*, а він у свою чергу повертає відповідь до *DispatcherServler*, а потім до клієнту. *DispatcherServlet* у свою чергу дає змогу використовувати фільтри(перехоплювачі) для вхідних запитів та вихідних відповідей для додаткової обробки.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Spring Security надає комплексні послуги безпеки для корпоративного програмного забезпечення додатків JavaEE-додатків. Існує особливий акцент на підтримці проектів, створених з використанням Spring Framework, яка наводить рішення JavaEE для розробки корпоративного програмного забезпечення.

Люди використовують Spring Security з багатьох причин, але більшість з них звертається до проекту, знайшовши функції безпеки специфікації сервлетів JavaEE або EJB специфікації, у яких не вистачає глибини, необхідної для типових сценаріїв корпоративних додатків. Незважаючи на те, згадуючи ці стандарти, важливо визнати, що вони не є портативними на WAR або EAR рівні. Тому, якщо ви змінюєте серверне середовище, це, як правило, багато роботи, щоб переналаштувати безпеку вашого додатку в новому цільовому середовищі. Використання Spring Security долає ці проблеми, а також приносить вам десятки інших корисних настроюються функцій безпеки.

Існує дві основні галузі безпеки додатків "аутентифікація" і

"авторизація" (або "контроль доступу"). Ці два основних напрямки, на які націлений Spring Security модуль. "Аутентифікація" це процес встановлення провідника (зазвичай це користувач, пристрій або яка-небудь інша система, яка може виконати дію в додатку). "Авторизація" відноситься до процесу прийняття рішення, чи дозволено провіднику виконувати дії у вашому додатку . Щоб прийти до точки, де необхідна авторизація рішення, інформація про провідника вже була встановлена у процесі перевірки аутентифікації. Ці поняття є загальними, а зовсім не специфічні для Spring Security [7].

На рівні аутентифікації, Spring Security підтримує широкий спектр моделей аутентифікації. Більшість з цих моделей аутентифікації або надаються третіми сторонами, або розробляються відповідними органами стандартизації, такими як Internet Engineering Task Force. Крім того, Spring Security надає свій власний набір функцій аутентифікації. Зокрема, на даний момент Spring Security підтримує інтеграцію аутентифікація із цими технологіями [7]:

– HTTP Basic authentication заголовки (IETF RFC стандарт);
– HTTP Digest authentication заголовки (IETF RFC стандарт);
– HTTP X.509 обмін сертифікатами із клієнтом (IETF RFC стандарт);
– LDAP;
– аутентифікація через форми;
– аутентифікація OpenID;
– аутентифікація на основі попередньо встановлених заголовків (Computer Associates Siteminder);
– JA-SIG центральна служба аутентифікації (CAS);
– RMI та HttpInvoker;
– автоматична «запам’ятайте мене» аутентифікація;
– анонімна аутентифікація;
– запустити-як аутентифікація(як адміністратор, користувач …);

– Java аутентифікація та сервіс авторизації (JAAS);
– аутентифікація JEE контейнеру;
– Kerberos;
– Java Open Source Single Sign On (JOSSO);
– OpenNMS мережева платформа керування;
– AppFuse;
– AndroMDA;
– Mule ESB;
– Direct Web Request (DWR);
– Grails;
– Tapestry;
– JTrac;
– Jasypt;
– Roller;
– Elastic Path;
– Atlassian Crowd.

Незалежно від механізму аутентифікації, Spring Security забезпечує широкий набір можливостей авторизації. Є три основні області інтересів – авторизація веб-запитів, чи можна посилатися на методи і дозвіл до окремих екземплярів об'єктів домену. Spring Security забезпечує глибокі можливості у всіх цих важливих областях.

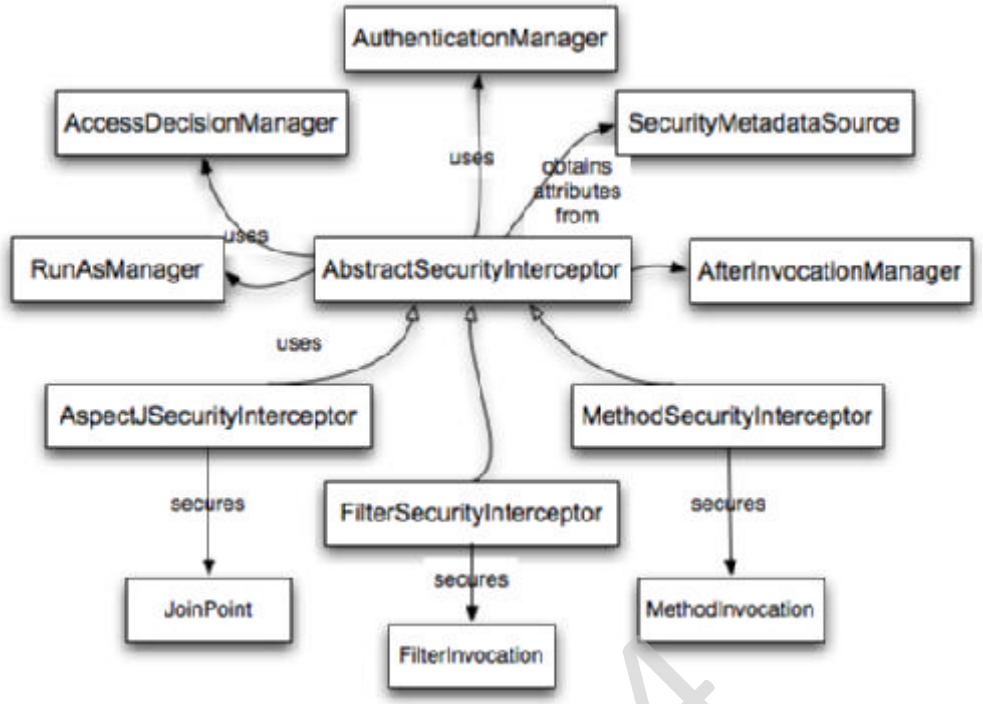

Рисунок 2.7– Модель перехоплювачів Spring Security

У проекті Memories Spring Security буде використовуватись разом із Spring MVC і для цього потрібно пояснити ще деякі деталі.

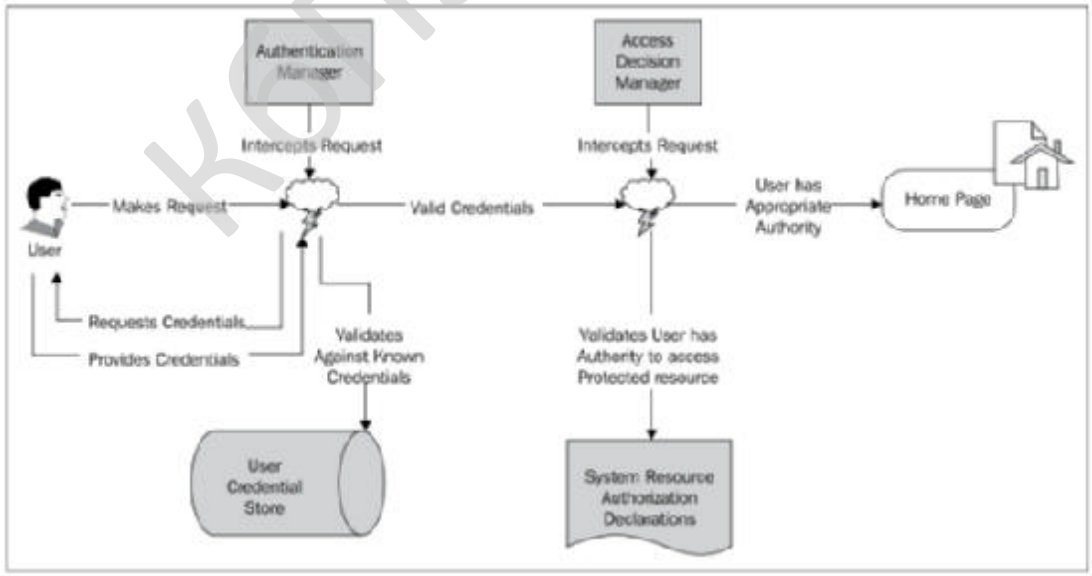

Рисунок 2.8 – Аутентифікація користувача

Коли клієнт робить запит, то він пересилається до *DispatcherServlet*, як раніше було згадано. Для *DispatcherServlet* можна встановити

інтерцептори (перехоплювачі) для запитів, що і робить Spring Security. Він (Spring Security) встановлює свої фільтри безпеки. На рисунку 1.8 клієнт робить запит, запит перехоплюється і передається до *AuthenticationManager*, який звертається до *UserCredentialStore* для пошуку аутентифікаційних даних переданих користувачем у сховищі. Зазвичай, сховище налаштовується і це може бути база даних, указані у конфігураційному файлі дані користувачів, або навіть збереження авторизаційних даних в оперативній пам'яті. В залежності від результату пошуку *AuthenticationManager* відбувається наступне: або запит клієнта проходить далі, або повертається відповідь з поясненням, що неправильні аутентифікаційні дані. Далі *AccessDecisionManager* перевіряє, чи достатньо у даного користувача прав для доступу к потрібному ресурсу і *AccessDecisionManager* звертається до отриманих даних від *UserCredentialStore* та порівнює їх із *SystemResourceAuthorizationDeclarations*, тобто із необхідними правами доступу до цього ресурсу. І знову в залежності від результату на цьому етапі можливо наступні ситуації: або запит клієнта проходить далі, або повертається відповідь з поясненням, що недостатньо прав для доступу до даного ресурсу. Потім запит передається далі до *DispatcherServler*, а цей процес вже описано у розділі Spring MVC.

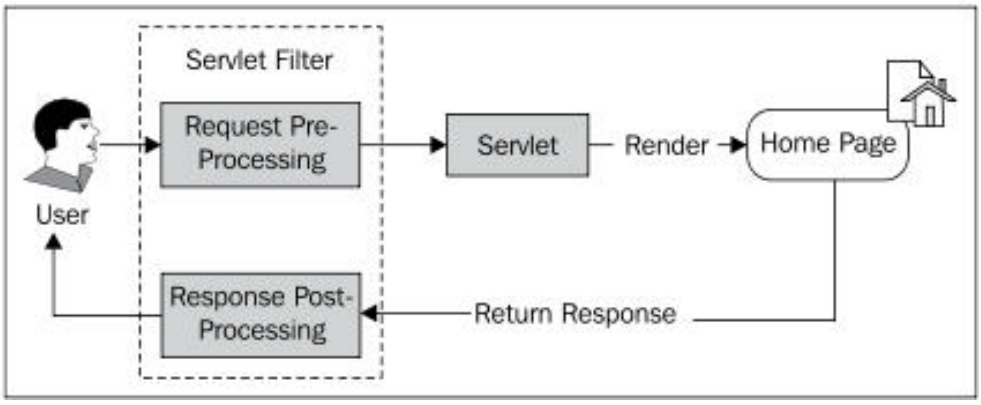

Рисунок 2.9 – Захоплення фільтром сервлету запиту клієнта

На цьому рисунку ілюструється ще один важливий момент.

Інтерцептор Spring Security також виконує пост-обробку відповіді. Тобто рівень безпеки містить також і перехоплення відповіді і в залежності від ситуації виконується потрібна обробка. Наприклад, при поверненні відповіді можна модифікувати її в залежності від пристрою клієнта, навіть блокувати. Але це також можна зробити і на обробці запиту.

На рисунку 2.10 зображено ланцюжок фільтрів сервлету, які у свою чергу викликають методи інтерцепторів у Spring MVC, на яких основана веб безпека Spring Security.

Spring Data високорівневий модуль Spring Framework, мета якого полягає в об'єднанні і полегшенні доступу до різних видів сховищ, як реляційні системи управління базами даних і сховищ даних NoSQL [8].

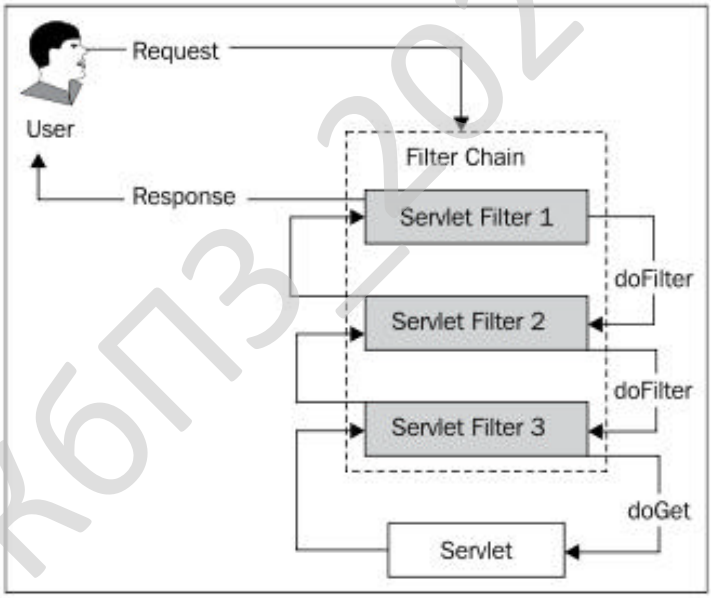

Рисунок 2.10 – Ланцюжок фільтрів сервлету

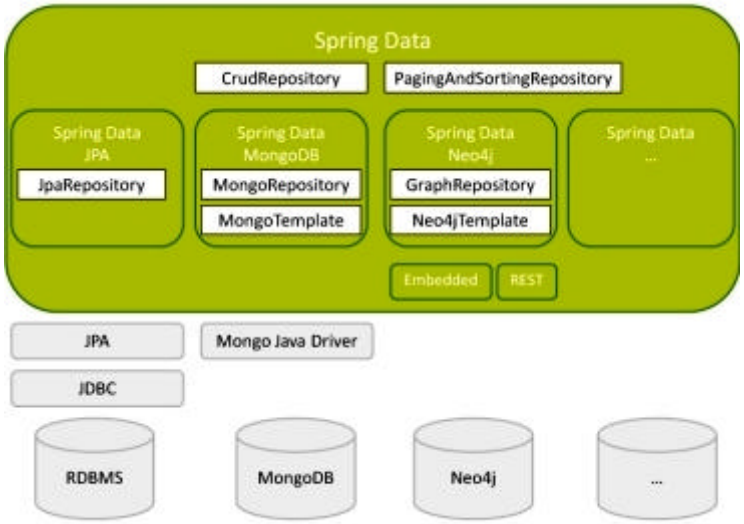

Рисунок 2.11 – Підтримка Spring Data різного виду сховищ та базові інтерфейси доступу до даних

Із кожним видом постійного сховища, ваші репозиторії (об'єкти DAO, або Data Access Objects) як правило, пропонують CRUD (Create-Read-Update-Delete) операції над одним із доменних об'єктів, пошук, сортування та розбиття на сторінки. Spring Data забезпечує загальні інтерфейси для цих аспектів (*CrudRepository*, *PagingAndSortingRepository*), а також збереження у сховище використовуючи свої конкретні реалізації інтерфейсів Spring Data [9].

Об’єкти шаблону є потужним інструментом, але можна зробити краще. Використовуючи репозиторії Spring Data, ви тільки повинні написати інтерфейс з методами пошуку, визначеними відповідно до заданих набором конвенцій (які можуть змінюватися в залежності від виду сховища, який використовується). Spring Data забезпечить належну реалізацію цього інтерфейсу під час виконання. Як приклад:

```
public interface UserRepository extends JpaRepository<User, String> {

        @Transactional(timeout = 2,
                propagation = Propagation.REQUIRED)
        @Query("SELECT u FROM User u WHERE u.name = 'User 3'")
        List<User> findAllUsers();

}
```

Виклик методу *findAllUsers()* виконає запит до бази даних згідно з запитом, який записано у анотації *@Query*, також для виконання цього

запита потрібна власна транзакція, і при необхідності вона буде створена.

Spring Data була створена на заміну DAO, тому що, при проектуванні DAO класів майже завжди відбувалося дублювання коду. Існує принцип DRY (Don't repeat yourself, Не повторюй самого себе). Тобто потрібно уникати дублювання коду. Spring Data дає змогу цього досягти великою кількістю інтерфейсів для різних ситуацій. У проекті Memories ми будемо використовувати модуль Spring Data JPA, для створення ORM між даними у базі даних та об'єктами у програмі. При написанні методів інтерфейсів можна використовувати ще одну перевагу – це ключові слова у назвах методів [8].

Keyword	Sample
And	findByLastnameAndFirstname
Or	findByLastnameOrFirstname
Between	findByStartDateBetween
LessThan	findByAgeLessThan
GreaterThan	findByAgeGreaterThan
IsNull	findByAgeIsNull
IsNotNull,NotNull	findByAge(Is)NotNull
Like	findByFirstnameLike
NotLike	findByFirstnameNotLike
OrderBy	findByAgeOrderByLastnameDesc
Not	findByLastnameNot
In	findByAgeIn(Collection<Age> ages)
NotIn	findByAgeNotIn(Collection<Age> age)

Рисунок 2.12 – Ключові слова для використання при написанні назв методів інтерфейсу та приклади використання

Тобто достатньо лише вказати потрібні ключові слова для створення деякої фільтрації результатів та результат, який повинен повернути Spring Data. І все. А завдяки використанню анотацій – можливо інтегрувати використання транзакцій, специфічних запитів та ін.

В якості мови програмування було обрано Java — одна з найпопулярніших мов для веб-розробки, особливо з використанням таких фреймворків, як Spring. Інтерфейс на платформі android зручно описувати використовуючи XML.

2.3 Розгорнута постановка завдання

Отже, сучасний стан модулів Spring Framework відмінний, доводом цього є не тільки актуальні на даний час версії фреймворку, а також і використання фреймворку компаніями по розробці програмного забезпечення. У проекті Memories ми будемо використовувати всі описані вище модулі Spring Framework

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для Web-додатка для збереження спогадів з застосуванням фреймворка Spring.

У процесі підготовки роботи другого (магістерського) ступеня вищої освіти необхідно виконати такі завдання:

- проектування архітектури веб системи;
- розробка структури бази даних;
- реалізація рівня даних додатку з використанням JPA та Spring Data JPA;
- інтегрування у додаток Spring MVC та розробка контролерів;
- інтегрування Spring Security та розробка системи безпеки додатку;
- створення клієнту для відображення даних від веб системи.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Клієнт буде створено на платформі Android 8.1, для підтримання максимальної кількості пристроїв.

З Android API буде використано:

– шаблони інтерфейсу додатку;

– класи багатопотоковості android, *AsyncTask*;

– базове API для створення додатків на android;

Для роботі із мережею буде використано бібліотека Retrofit. Вона має переваги над стандартними бібліотеками роботи із мережею самого android. Вона (Retrofit) – швидка та дає зручне API для безпечної роботи із REST.

Із Retrofit буде використано анотації:

– *@GET* – для роботи через Http get метод;

– *@POST* – для роботи через Http post метод;

– @FormUrlEncoded – для кодування даних як при роботі через html форми;

– *@Field* – додаткова анотація до *@FormUrlEncoded*, поле у html формі;

– *@Multipart* – для роботи із файлами;

– *@Part* – додаткова анотація до @Multipart, частина тіла відповіді;

– *@Headers* – для роботи із заголовками відповідей;

– Та класи:

– *RestAdapter* - для генерації реалізації інтерфейсів із Retrofit анотаціями;

– *Callback* – власний клас реалізації функції зворотного виклику;

– *Response* – для роботи із відповіді від веб системи;

– *RequestInterceptor* – для перехоплення запита до сервера та проведення деяких операцій, у нашому випадку – додавання куки сесії, щоб постійно не відсилати до серверу дані для авторизації;

– OkHttpClient – у ядрі бібліотеки Retrofit лежить бібліотека для роботи із мережею, але на більш низькому рівню (потрібно не ього класу ми додаємо перехоплював(RequestInterceptor) для запитів і кожен забувати, зо Retrofit зроблена полегшувати роботу із REST). Для ц раз додаємо куку із ідентифікаційним номер сесії.

При розробці клієнта використовувалася бібліотека Picasso. Вона пропонує зручне API для роботи зі зображеннями у android. У цілому, головними використаними класами є:

– *Picasso* – головний клас для роботи із видом ImageView у android;

– *LruCache* – клас, який реалізує кеш LRU(Least recently used, Витіснення давно не використовуваних). Дуже корисний клас, мобільні платформи мають один значний недолік – коли активно використовується оперативна пам'ять, то значно зменшується заряд батареї, але також і відчуваються значні лаги при користуванням приладу. Цей клас дає змогу створити кеш розміром 1/7 розміру доступної оперативної пам’яті та зберігати там зображення. Це досить сильно прискорює прилад та зберігає трафік у нашому випадку;

– *Transformation* – для створення необхідних операцій над зображеннями, обрізання гострих кутів та ін.;

Переховане буде складати основу клієнта веб системи.

3.2 Розробка структурної схеми

Розпочнемо з розробки структурної схеми, на якій зображена екосистема додатку для збереження спогадів.

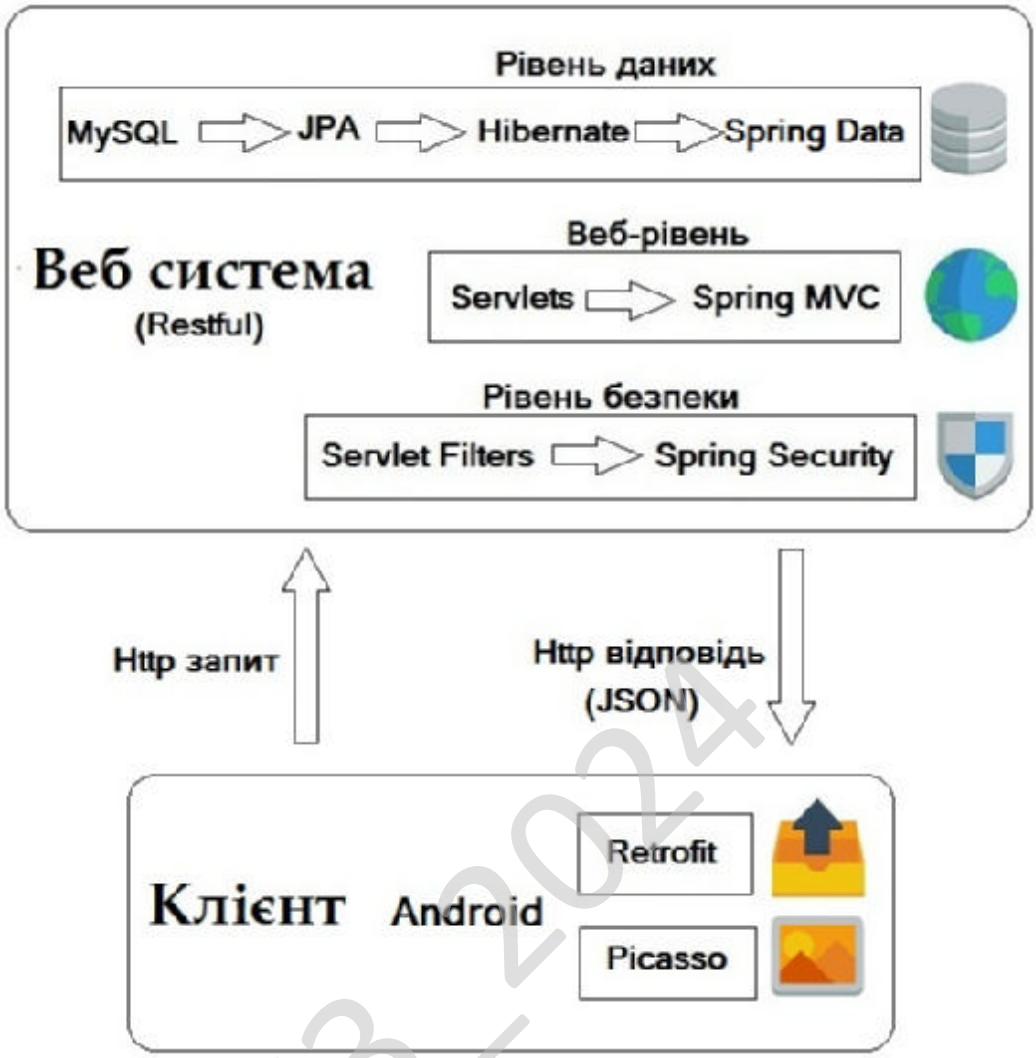

Рисунок 3.1 – Структурна схема додатку для збереження спогадів

Рівень даних

Отже, почнемо з огляду рівня даних веб-системи. При розробці рівня даних буде використовуватись база даних MySQL. MySQL була вибрана по наступним критеріям [10]:

– найбільш популярна повноцінна серверна СУБД;

– вільно поширювана;

– більша частина функціоналу SQL підтримується MySQL;

– велику кількість функцій, які забезпечують безпеку та які підтримуються за замовчуванням;

– масштабованість - MySQL легко працює з великими обсягами даних і легко масштабується;

– швидкість - спрощення деяких стандартів дозволяє MySQL

значно збільшити продуктивність.

Далі використовується JPA(Java Persistent API) – спеціальна бібліотека для збереження Java об’єктів у базу даних. Дуже корисний інструмент який прискорює процес розробки додатків у рази. В якості реалізації JPA використовується Hibernate та деякий додатковий функціонал Hibernate. У проекті Memories безпосередньо SQL запити використовуватись не будуть, за нас це зробить Hibernate, він згенерує необхідні SQL запити до бази даних. Але над Hibernate також налаштований Spring Data модуль, що дає також свій функціонал, який буде використано при створенні додатку. Конкретно у Spring Data модуля буде використовуватись його анотації:

– *@EnableJpaRepositories* – включення можливості використання JPA репозиторіїв з Sping Data JPA;

– *@EnableTransactionManagment* – включення можливості контролю транзакціями при запитах;

– *@Query* – для створення запитів із складною логікою;

– *@Transaction* – для створення запитів із налаштуванням транзакцій для них;

– *@Modifying* – для дозволу модифікації даних;

Та інтерфейси:

– *PagingAndSortingRepository* – для виконання зручних запитів із можливістю сортування даних, та подальшої зручної адаптації відтворювання даних посторінково вже на клієнті;

– *CrudRepository* – інтерфейс для виконання базових операцій створення, читання, модифікації та видалення даних.

На цьому етапі проектування архітектури рівня даних завершилось.

Веб-рівень

Веб-рівень починається із сервлетів, але безпосередньо їх ми використовувати не будемо. Буде використано Spring MVC, який є

надбудовою над Servlet API (сервлети). Отже, із Spring MVC ми вже познайомились, тепер потрібно вказати саме що буде використано із Spring MVC API.

Анотації:

– *@Controller* – для створення контролерів, які будуть обслуговувати веб-запити та повертати *DispatcherServlet*'у модель із даними;

– *@RequestParam* – для того, щоб значення параметру запиту, ім'я якого вказано у дужках та лапках(*@RequestParam("name of request param")*), прив'язати до значення змінної;

– *@RestController* – для того, щоб вказати Spring MVC, що реалізується REST API, яке обслуговує JSON, XML, або спеціально вказаний тип *MediaType* контенту;

– *@ModelAttribute* – коли запит переадресовується до іншого контролеру, то запит поки що не передається до *View Resolver*, але модель вже можна створювати. Тобто існує можливість в одному контролері додати дані до моделі, переадресувати до іншого контролеру та прочитати їх там. Або можливо навіть передати модель до View Resolver, а потім дані, які повертаються назад з відповіддю від клієнта Spring MVC може розпарсити, або вони можуть зберігатися у об'єктах сесії, або ін.;

– *@RequestMethod* – для зазначення, по якому Http-методу потрібно обробляти запити цьому контролеру;

– @RequestHeader – для отримання або для встановлення заголовків у запитах або відповідях.

Це базові речі, які будуть використовуватись. Проектування веб-рівня закінчилось.

На даному етапі буде розглянуте покрокове проектування структури бази даних. Почнемо із сутності користувача.

Сутність «Користувач». Сутність користувача мусить містить дані для авторизації, логін та пароль, та роль його користувача – щоб визначити права користувача. При подальшій розробці додатку це поле знадобиться для створення розширеного інтерфейсу для модераторів та адміністраторів.

Для зв’язку із користувачам потрібна електронна пошта та як до нього звертатись, тобто ім’я. Та id для створення зв’язків між таблицями у СУБД.

Таблиця 3.1 - Структура таблиці «Користувачі»

Користувачі
Ідентифікаційний номер
Логін
Пароль
Ім’я
Електронна пошта
Роль користувача

Сутність «Спогад». Спогад буде містити таку інформацію: ідентифікаційний номер, текст спогаду, ідентифікаційний номер типу настрою спогаду, опис настрою спогаду, ідентифікаційний номер користувача-власника його спогаду, чи можуть бачити цей спогад інші користувачі.

Тепер пояснення. Щодо ідентифікаційного номеру – все очевидно, для подальших зв’язків з іншими таблицями. Текст спогаду – це невеликий опис того моменту, в який було створено цей запис. Тип настрою – також очевидне значення, але різниця у реалізації цієї можливості буде різнитися залежно від клієнта веб системи, тобто кожен із клієнтів додатків може реалізувати це по своєму. Опис настрою спогаду – це також повинно різнитися залежно від клієнта додатка веб системи. Але, наприклад, можна вказати емоції, які відчуває людина у той момент. Ідентифікаційний номер користувача-власника – це звичайний зв'язок із таблицею «користувачі», це поле вказує на власника спогаду.

Таблиця 3.2 - Структура таблиці «Спогади»

Спогади
Ідентифікаційним номер
Текст спогаду
Ідентифікаційний номер типу настрою
Опис настрою
Ідентифікаційний номер власника спогаду

Сутність «Настрої». Ця сутність має просту структуру і не подрібнює пояснення. Це довідкова таблиця.

Таблиця 3.3 - Структура таблиці «Настрої»

Настрої
Ідентифікаційний номер
Назва

Сутність «Зображення». Ця сутність зберігає шлях до файлу(URL) на сервері та ідентифікаційний номер спогаду, до якого воно належить.

Таблиця 3.4 - Структура таблиці «Зображення»

Зображення
Ідентифікаційний номер
Ідентифікаційний номер спогаду
Шлях до файлу(URL)

Сутність «Відносини». Ця таблиця має ключове значення для подальшого розвитку веб системи. У перспективі додаток буде підтримувати можливість користувачам поділитися власними спогадами, та вказати якими саме. Також, завдяки відносинам між користувача буде реалізована система коментування спогадів. З одного погляду ці речі досить прості, але сама можливість поділитися спогадом із іншими користувача створює щось подібне із стрічкою новин(news feed), а це зовсім не тривіальна задача і виходить далеко за межі цієї роботи. Але, авжеж, у розвиток веб системи це входить обов'язково.

Отже, відносини бувають, зазвичай, між двома людьми (як одиниця відносин). Цю ідею і реалізує таблиця «Відноси». Також вона містить стан відносин: підписка на новини, друзі, заблоковано; і дату створення зв'язку(оновлення). Якщо тип відносин буде змінено – то і дата, коли буде виконане оновлення зв'язку, перезапишеться.

Таблиця 3.5 - Структура таблиці «Відносини»

Відносини
Ідентифікаційний номер
Ідентифікаційний номер 1-шого користувача
Ідентифікаційний номер 2-гого користувача
Ідентифікаційний номер типу зв'язку
Остання дата оновлення зв'язку

Сутність «Тип відносин». Ця сутність також відноситься до типу довідкових сутностей та має тривіальну структуру: ідентифікаційний номер та назва зв'язку.

Таблиця 3.6 - Сутність «Тип відносин»

Тип відносин
Ідентифікаційний номер
Тип зв'язку

Фінальний результат

Тепер ми реалізуємо ці сутності у нашій базі даних MySQL та завдяки використанню додатку «Дизайнер», ми зробили наступну діаграму.

Тут видно і типи полів у таблиці та зв’язки між ними.

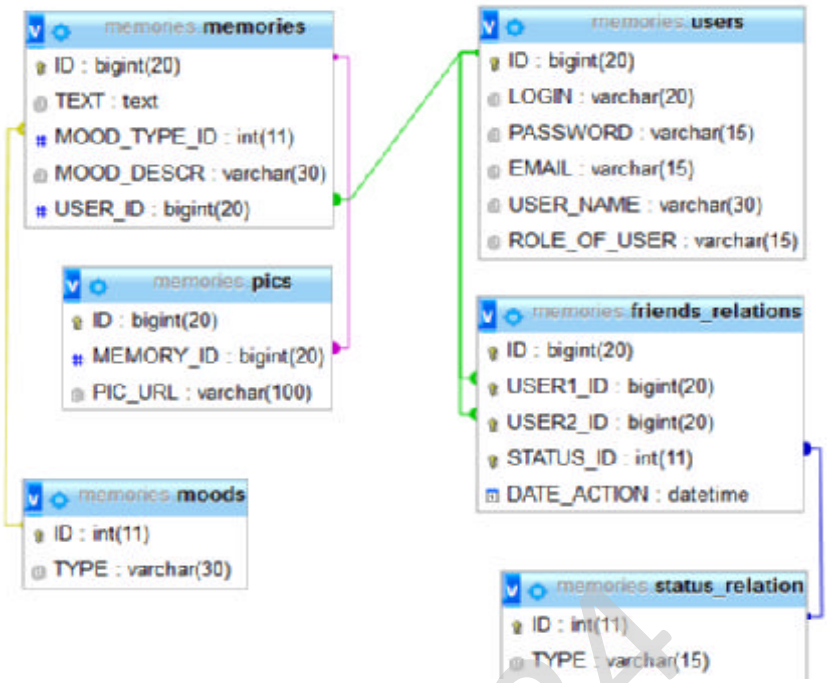

Рисунок 3.2 – Структурна схема бази даних веб-додатку

3.3 Розробка функціональної схеми

Рисунок 3.3 – Функціональна схема схема додатку для збереження спогадів

Функціональна схема відображає взаємодії компонентів програмного забезпечення з описом інформаційних потоків, склад даних в потоках і вказівкою частин, що використовуються.

Автентифікація виконує функцію, спрямовану на забезпечення

безпеки інформації. За допомогою неї ідентифікується користувач та визначаються його права доступу.

Автентифікації користувача включає в себе введення логіну та паролю; також забезпечує безпомилкове визначення користувача під час отримання доступу.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання магістерської роботи, наведена на рисунку 3.4. З нього видно, що процеси взаємодіють наступним чином.

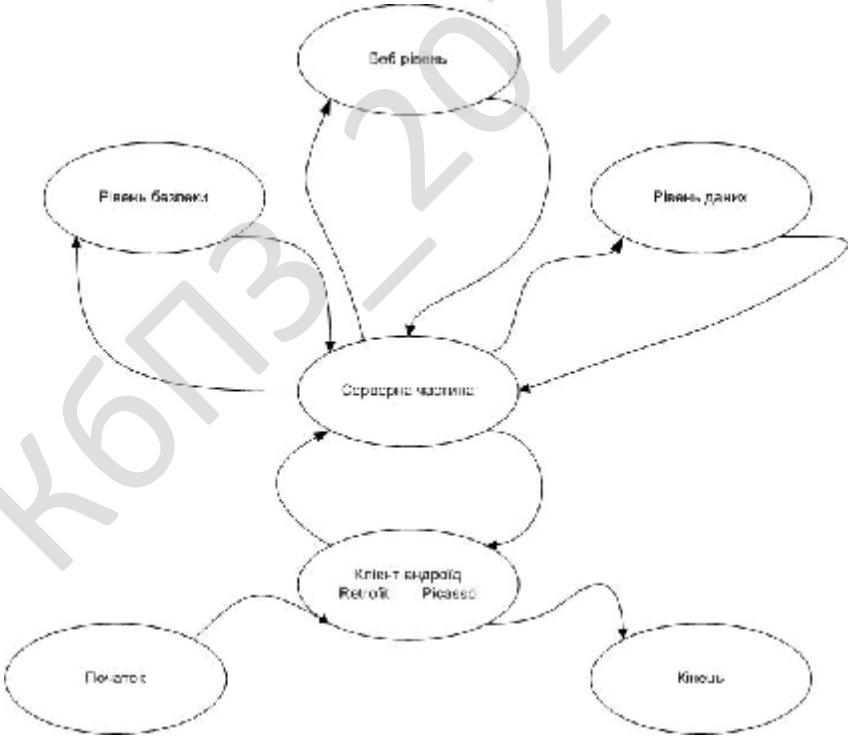

Рисунок 3.4 – Діаграма взаємодії процесів

Android додаток надсилає HTTP запит на сервер, наприклад, через бібліотеки Retrofit для взаємодії з REST API. Retrofit дозволяє працювати з JSON відповідями та зручно інтегрувати HTTP запити в додаток. Для завантаження зображень використовуєте Picasso, що дозволяє легко

відображати зображення у вашому інтерфейсі.

Веб-рівень**:** Запит надходить на сервер, де його обробляють сервлети та Spring MVC. Spring MVC відповідає за маршрутизацію запиту на відповідний контролер. Контролери (Controller) обробляють логіку запиту і можуть взаємодіяти з іншими частинами системи для виконання необхідних операцій.

Рівень безпеки**:** Використовується Spring Security, який додає фільтри для контролю доступу. Це дозволяє захищати певні ресурси та перевіряти автентифікацію та авторизацію користувачів.

Рівень даних**:** Сервер взаємодіє з базою даних MySQL через JPA та Hibernate. Spring Data полегшує роботу з базою даних, забезпечуючи простіший доступ до даних через репозиторії (Repository).

Після виконання необхідних операцій сервер формує відповідь у форматі JSON та відправляє її назад клієнту через HTTP.

Android додаток приймає JSON відповідь та обробляє її за допомогою Retrofit, наприклад, для відображення даних в інтерфейсі користувача або виконання подальших дій.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

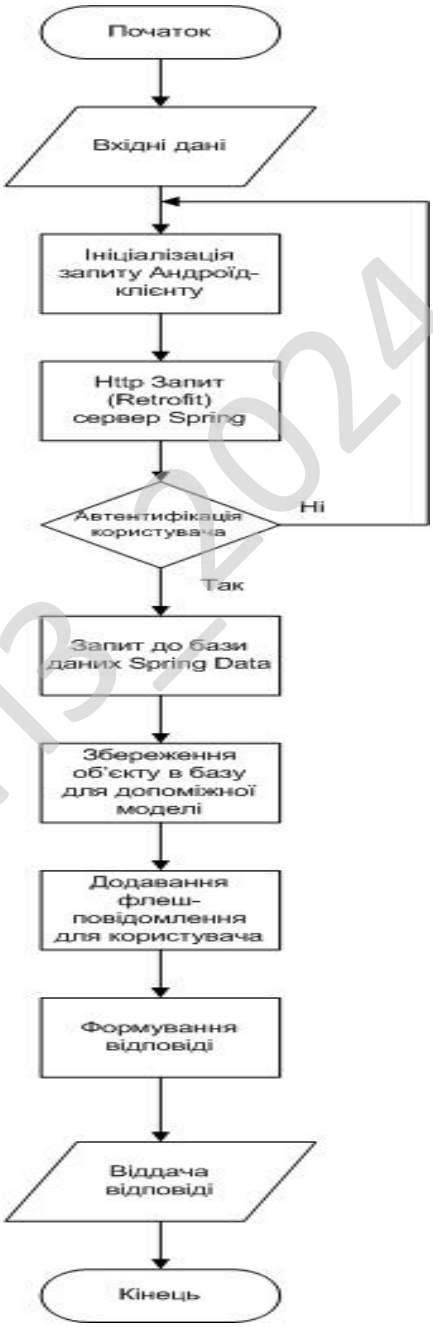

Рисунок 4.1 – Блок-схема алгоритму роботи головної програми

Головний алгоритм роботи системи працює наступним чином:

1. Спочатку відбувається ініціація запиту з боку клієнта (Android):

- Користувач виконує певну дію (наприклад, натискає кнопку), яка запускає процес HTTP-запиту;

- Android-додаток використовує Retrofit для створення та відправки HTTP-запиту на сервер.

2. Надсилання запиту на сервер:

- Запит відправляється на веб-сервер через HTTP. Це може бути запит типу GET, POST, PUT, або DELETE, залежно від потреби.

3. Обробка запиту на сервері (Spring):

- Фільтри безпеки Spring Security: Запит проходить через фільтри безпеки, де перевіряється автентифікація (наприклад, токен доступу) і авторизація (доступ до конкретних ресурсів);

- Маршрутизація запиту (Spring MVC): Після проходження фільтрів запит передається на відповідний контролер (Controller), який відповідає за обробку цього типу запиту;

- Логіка обробки: Контролер виконує необхідну бізнес-логіку. Якщо потрібні дані з бази, контролер звертається до сервісного рівня.

4. Доступ до даних (Spring Data + Hibernate + MySQL):

- Контролер звертається до рівня роботи з даними (Repository або Service), де виконується запит до бази даних через JPA та Hibernate;

- Hibernate перетворює SQL запити в об'єкти Java та навпаки;

- Дані з MySQL бази отримуються або записуються відповідно до запиту.

5. Формування та відправка відповіді:

- Після обробки запиту сервер формує відповідь у форматі JSON;

- Відповідь повертається через HTTP клієнту.

6. Обробка відповіді на клієнті (Android):

- Android-додаток отримує HTTP відповідь у форматі JSON;

- Retrofit перетворює JSON відповідь у об'єкти Java, з якими легко

працювати в додатку;

- За допомогою Picasso (якщо відповідь містить зображення) зображення можуть бути завантажені та відображені в інтерфейсі.

7. Відображення даних користувачу:

- Клієнт обробляє дані та відображає їх у інтерфейсі (UI) для користувача, або виконує інші дії на основі отриманої відповіді.

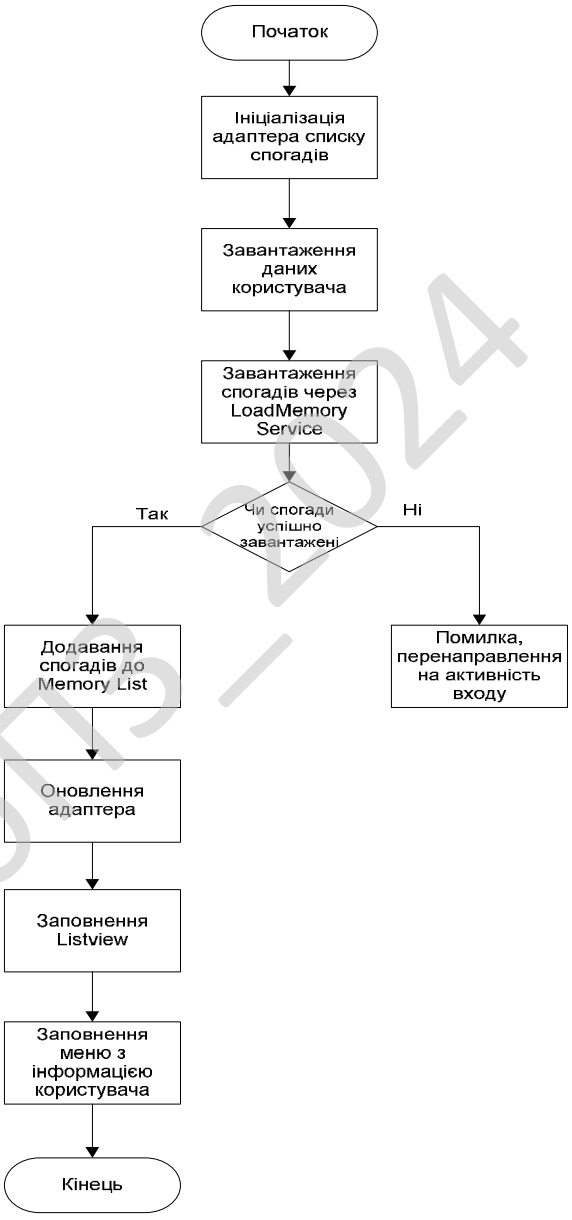

Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми завантаження спогадів

Опис алгоритму:

1. Ініціалізація адаптера: Після запуску адаптера класу

MemoriesListAdapter, ініціалізується об'єкт UserInfoStorage, який завантажує дані користувача.

2. Завантаження спогадів: Викликається сервіс LoadMemoriesService, який завантажує спогади користувача.

3. Перевірка успішного завантаження:

- Якщо спогади успішно завантажені, вони додаються до списку memoryList, і адаптер оновлюється;
- Якщо завантаження не вдалося, викликається помилка, і користувач перенаправляється на активність входу LoginActivity.

4. Відображення даних:

- Оновлений список спогадів заповнюється в ListView;
- Також заповнюється бокове меню (Side Menu) з інформацією про користувача.

5. Кінець процесу.

Ця схема відображає логіку завантаження і відображення спогадів у додатку, з перевіркою успішності завантаження і відповідною обробкою помилок.

Реалізація рівня даних додатку з використанням JPA та Spring Data JPA. Створення сутностей із використанням стандарту JPA. При створенні сутностей, які відображають сутності із бази даних, використовувались анотації із пакету javax.persistence. Розглянемо приклад створення сутності «Відносини»:

```
@Entity
@Table(name = "friend_relations")
public class FriendRelation {

    @Id
    @GeneratedValue
    @Column(name = "id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user1_id")
    private User user1;

    @ManyToOne
    @JoinColumn(name = "user2_id")
    private User user2;

    @ManyToOne
    @JoinColumn(name = "status_id")
    private StatusRelation status;

    @Temporal(value = TemporalType.TIMESTAMP)
```

```
    @Column(name = "date_action")
    private Date dateAction;

    public Date getDateAction() {
        return dateAction;
    }
    public void setDateAction(Date dateAction) {
        this.dateAction = dateAction;
    }
    public StatusRelation getStatus() {
        return status;
    }
    public void setStatus(StatusRelation status) {
        this.status = status;
    }
    public User getUser2() {
        return user2;
    }
    public void setUser2(User user2) {
        this.user2 = user2;
    }
    public User getUser1() {
        return user1;
    }
    public void setUser1(User user1) {
        this.user1 = user1;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
}
```


Почнемо з анотацій *@Entity* та *@Table*. Анотація *@Entity* повідомляє інструменту, який реалізує стандарт JPA, що цей клас – це сутність, яка відповідає сутності у базі даних. Анотація *@Table* вказує, яку таблицю із бази даних відображає ця сутність у Java коді. В якості параметру передається ім’я таблиці у базі даних.

Наступні анотації для розгляду – це *@Id*, *@GeneratedValue* та *@Column*.

Анотація *@Id* вказує, що дане поле є ідентифікатором строк у таблиці.

Анотація *@GeneratedValue* виконує автоматичне присвоєння значення до колонки ідентифікатора у нових записах в таблиці. Також можна вказати стратегію генерації значень.

Анотація *@Column* використовується для прив’язання поля класу до колонки у

таблиці в базі даних.

Отже, код:

```
@Id
@GeneratedValue
@Column(name = "id")
private Long id;
```

означає, що це поле (*Long id*) є ідентифікатором строк у таблиці, воно генерується автоматично (у MySQL для цієї колонки вказано автоінкремент значення) та у базі даних відповідає колонці з назвою «id».

Як видно із коду, використовується також анотація *@Temporal*. Вона відповідає за потрібне відображення часу у Java, а саме конвертування у потрібний формат та присвоєння значення класу *java.util.Date*. Бази даних мають більш широкий вибір типів для зберігання часу, тому ця анотація часто стає у пригоді при виникненні проблем із конвертацією часу.

В нашому випадку, поле *dateAction* зберігає *TIMESTAMP*, тобто час у форматі '2038-01-19 03:14:07'.

Відношення сутностей у JPA. Стандарт JPA включає в себе анотації для створення відношень між сутностями між Java бінами. Ці анотації:

- @OneToMany – один-до-багатьох;
- @ManyToOne – багато-до-одного;
- @OneToOne – один-до-одного;
- @ManyToMany – багато-до-багатьох.

У нашій сутності FriendRelation існує зв'язок із сутністю User та StatusRelation із типом зв’язку багато-до-одного.

У коді:

```
@ManyToOne
@JoinColumn(name = "user1_id")
private User user1;
```

Це означає, що між сутністю, у якій це поле об’явлено (*FriendRelation*), та сутністю *User* існує зв'язок багато-до-одного. Але це не все. Також у сутності *User* повинно міститись поле такого виду:

```
@OneToMany(mappedBy = "user1")
private Collection<FriendRelation> friendsRelations;
```

Це означає, що між сутністю *User* та *FriendRelation* тип зв’язку один-до-багатьох. Тобто стандарт JPA потребує, щоб типу зв’язків із сутностями

вказувалися з обох боків цих зв'язків. Теж саме і для зв'язку із сутністю *StatusRelation*.

```
@ManyToOne
@JoinColumn(name = "status_id")
private StatusRelation status;
```

Та в сутності *StatusRelation*:

```
@OneToMany(mappedBy = "status")
private Collection<FriendRelation> friendsRelations;
```

Це досить зручно, інколи буває необхідно із однієї сутності отримати іншу, або колекцію інших. Але також це буває і проблемою, наприклад, наша веб система буде віддавати усі дані у форматі JSON, а виконувати конвертування буде модуль Spring MVC. І він буде конвертувати усі поля сутності у поля об'єкту JSON. По-перше, це може викликати безкінечну рекурсію (при відтворенні сутності *FriendRelation* у JSON буде відтворене її поле status, а це означає, що буде спроба конвертації сутності *StatusRelation*, яка у свою чергу має поле *friendsRelations* – на цьому моменті і виникне безкінечна рекурсія, яка і приведе до падіння серверу). По-друге, виникає можливість генерації виключення (exception) *LazyInitException*, яке повідомляє, що для отримування даних не відкрита сесія, а не відкрита вона для того, щоб не перевантажувати базу даних зайвими запитами (тому що, наприклад, поле із колекцією може містити велику кількість записів і це може привести до перевантаження запитами бази даних та до генерації виключення).

Створення репозиторія та сервісу

При конфігурації додатку використовувались анотації. Для доступу до даних використовувалась *@EnableJpaRepositories.* Spring при деплої(установки) на сервер додатків виконає пошук у пакеті, вказаному у параметрі анотації *@EnableJpaRepositories*. Після того, як Spring знайде анотації, дочірні до анотації *@Component*, а в даному випадку йде мова о рівні даних, то використовується *@Repository*. Потім буде виконана генерація класу на основі нашого інтерфейсу для доступу до даних. Але, так

як у нас потрібен базовий функціонал для доступу до даних ми будемо використовувати інтерфейс *CrudRepository<Entity,Id>*, який вже помічений анотацією *@Repository* за замовченням. Цей інтерфейс дає змогу виконувати запити всіх типів: створення, читання, оновлення та видалення даних із бази.

Розглянемо на прикладі реалізації сутності «Відносини».

```
public interface FriendRelationsRepository
      extends CrudRepository<FriendRelation, Long> {
}
```

Це все, що потрібно для реалізації репозиторія до даних таблиці *FRIEND_RELATIONS* із бази даних, останню роботу за нас виконає Spring Data JPA.

Тепер детально розглянемо можливості, які Spring дає нам по замовчуванням при такому виді інтерфейсу. На прикладі сервісу *FriendRelationsService*:

```
@Service
public class FriendRelationsService
extends DataService<FriendRelation> {

    @Autowired
    private FriendRelationsRepository friendRelationsRepository;

    @Override
    public void persist(FriendRelation entity) {
        friendRelationsRepository.save(entity);
    }
    @Override
    public void update(FriendRelation entity) {
        friendRelationsRepository.save(entity);
    }
    @Override
    public FriendRelation findById(Long id) {
        return friendRelationsRepository.findOne(id);
    }
    @Override
    public void delete(Long id) {
        friendRelationsRepository.delete(id);
    }
    @Override
    public List<FriendRelation> findAll() {
        return Util.makeCollection(
             friendRelationsRepository.findAll());
    }

    @Override
    public void deleteAll() {
        friendRelationsRepository.deleteAll();
    }
    @Override
```

```
    protected AbstractDao<FriendRelation> getDao() {
        return null;
    }
}
```

Перше, що потрібно відмітити – це анотацію *@Service*. Так прийнято помічати клас(бін), який виконує функцію мосту між репозиторіями та контролерами. Тобто клас-сервіс також маніпулює даними, як і репозиторії, але вже на рівні логіки додатку, та результатом роботи методів класу сервісу є дані, які можна використовувати у контролерах (тобто і в видах).

Наступний момент - це анотація *@Autowired*. Ця анотація належить до модулю Spring Core, частина DI, яку реалізує Spring Framework. Як вище зазначено, інтерфейс *CrudRepository* вже позначений анотацією *@Repository*. Це означає, що модуль Spring Data JPA згенерує бін(клас), а анотація *@Autowired* виконає пошук створених об'єктів типу *FriendRelationsRepository* та виконає прив'язку адресу об'єкту у пам'яті до посилання, яке позначено анотацією *@Autowired*.

Тепер щодо реалізованих Spring Framework'ом методів доступу до даних. Як видно із коду – Spring Data JPA реалізував методи: *findAll()*, *findById()*, *save()*, *delete()*, *deleteAll()*. Їх значення відповідає їх назвам, але відмітимо, що метод *save()* – виконує функції вставки і редагування даних.

Із коду видно, що наш клас-сервіс виконує роль оболонки над репозиторієм. Майже усі методи мають лише одну строку коду – це виклик відповідного методу із інтерфейсу репозиторія. Але так буває не завжди, особливо коли складна бізнес логіка додатку.

Інтегрування у додаток Spring MVC та розробка контролерів. Як було зазначено– Spring MVC дає можливість зручно працювати із веб середовищем через HTTP протокол. Наведемо приклад одного із контролерів та пояснимо принцип його роботи.

```
@Controller
@RequestMapping("/memory")
public class MemoriesController extends AbstractController {

    @Autowired
    private MemoriesService memoriesService;

    @ResponseBody
```

```
    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public Memory getMemoryById(@PathVariable("id") Long id,
                                HttpServletResponse response)
                    throws ServletException, IOException {
        Memory memory = memoriesService.findById(id);
        if (memory == null) {
            response.sendRedirect("/memory/error/"
                    + "?errorText=" + "Memory with id:" + id +
                        " doesn't exists.");
            return null;
        }
        return memory;
    }

    @ResponseBody
    @RequestMapping(value = "/user", method = RequestMethod.GET)
    public List<MemoryVO> getMemoriesByUser(
            @RequestParam(value = "id",required = false) Long id,
                  @RequestParam(value = "login",required = false)
                        String login,
            HttpServletResponse response)
                  throws ServletException, IOException {
        List<MemoryVO> list;
        if (id != null)
            list = memoriesService.findVOByUserId(id);
        else
            list = memoriesService.findVOByUserLogin(login);
        if (list.isEmpty()) {
            response.sendRedirect("/memory/error/"
                    + "?errorText=" +
                        "Memories not found, id: " + id);
            return null;
        }
        return list;
    }
}
```

Почнемо з головного - це анотація *@Controller*. Завдяки цій анотація Spring і розуміє, що даний клас може обробляти веб-запити. Як видно з назви контролеру – цей контролер відповідає за обробку запитів які стосуються спогадів. Анотація *@RequestMapping("/memory")* означає, що усі url запитів, які здатен обробляти цей контролер, повинні мати після кореневого url серверу приставку */memory*.

У контролері спогадів є сервіс для роботі із даними сподівань, *MemoriesService*, об'єкт класу якого буде створений після деплою додатку до веб-серверу (контейнеру сервлетів).

Контролер здатен обробляти запити до двох url:

– server_url/memory/id, де *id* – це ідентифікатор спогаду;

– server_url/memory/user?id=value_id&login=value_login, де *id* – це

ідентифікатор користувача, а *login* – логін користувача.

Потрібно зазначити, що контролер обслуговує лише *GET* запити, це можна побачити виходячи із анотації *@RequestMapping*, яка містить параметр *value* – це url для прив'язки запитів до методу контролеру, та *method* – тип HTTP методу, по якому виконується запит.

Так як наш додаток повинен слідкувати RESTful принципам, то над методами контролеру використовується анотація *@ResponseBody*. Вона значення, яке повертає метод, конвертує у JSON формат. Для цього використовується стороння бібліотека org.codehaus.jackson.

Наприклад, якщо виконати наступній запит:

```
http://localhost:9821/memory/user?login=admin
```

То сервер поверне значення такого виду:

```
[
	{
		"id":1,
		"text":"Hello world!",
		"moodId":1,
		"moodDescription":"Have a nice day!",
		"pictureUrls":[
			"https://i1.sndcdn.com/artworks-000031744302-0730nk-
			t500x500.jpg",
			"http://uploads-airport.rhcloud.com/img/lg-g3-linii-
			abstrakcii.jpg"]
	},
	{
		"id":2,
		"text":"This is second memory. Love it.",
		"moodId":2,
		"moodDescription":"That's how Ruff Ryders",
		"pictureUrls":["123"]
	}
]
```

Із відповіді видно, що сервер повернув два спогади, які містить користувач.

Потрібно відмітити також те, що для повернення результатів використовується клас MemoryVO (Value Object – малий об'єкт, який представляє простий об'єкт (сутність), які мають однакові значення, але які можуть бути не однаковими об'єктами). В даному випадку цей клас потрібен для уникнення передачі зайвих даних до користувачів веб системи.

```
public class MemoryVO {

	private Long id;
```

```
private String text;
private Long moodId;
private String moodDescription;
private List<String> picturesUrls;

public MemoryVO(Long id, String text,
                    Long moodId, String moodDescr,
                    List<String> urls) {
    this.id = id;
    this.text = text;
    this.moodId = moodId;
    this.moodDescription = moodDescr;
    this.picturesUrls = urls;
}

public Long getId() {
    return id;
}

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

public String getText() {
    return text;
}

public void setText(String text) {
    this.text = text;
}

public Long getMoodId() {
    return moodId;
}

public void setMoodId(Long moodId) {
    this.moodId = moodId;
}

public String getMoodDescription() {
    return moodDescription;
}

public void setMoodDescription(String moodDescription) {
    this.moodDescription = moodDescription;
}

public List<String> getPictureUrls() {
    return picturesUrls;
}

public void setPictureUrls(List<String> pictureUrls) {
    this.picturesUrls = pictureUrls;
}

@Override
public String toString() {
    StringBuilder builder = new StringBuilder();
    builder.append("id: " + id + "\n");
    builder.append("text: " + text + "\n");
    builder.append("moodId: " + moodId + "\n");
    builder.append("moodDescr: " + moodDescription + "\n");
    builder.append("picturesUrls: ");
   for (String str : picturesUrls)
        builder.append(str);
```

```
            return  builder.toString();
        }
    }
```

Із коду видно, що це звичайний клас, але він потрібний для передачі даних користувачам веб системи, або при створенні відповіді на запит, у якому приймає участь декілька сутностей. У випадку із *MemoryVO*, то оригінальна сутність має зв'язок із користувачем, настроєм та малюнками. Але у нашому VO лише ідентифікатор настрою та колекція малюнків.

Для повернення результатів можна було використовувати і клас сутності *Memory*, але тоді у користувача веб системи могло б з'явитись враження помилкових запитів, тому що поле користувача у спогаді завжди було б *NULL* і назва настрою також *NULL* - смислове перенавантаження користувачів веб-системи та можливе введення в оману можливим API.

Тіло методу *getMemoriesByUser()* досить просте. Якщо параметр запиту з іменем *id* пустий, тоді обробляється другий параметр запиту *login*, який вже повинен бути присутній. Обробляється лише один. Потім виконується запит до сховища даних через клас-сервіс *MemoriesService*, який у свою чергу звертається до репозиторію спогадів. Якщо у результаті буде повернено пустий список, то запит переадресується до контролеру помилок та у якості відповіді на запит посилається текст помилки. Якщо список не пустий – то він конвертується бібліотекою jackson у JSON та посилається користувачу у якості відповіді на запит.

Подібно працює і метод для отримання спогадів по їх ідентифікаційному номеру.

Інтегрування у додаток Spring Security та розробка системи безпеки додатку. Інтегрування Spring Security починається зі створення конфігураційного класу.

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter
{

    @Autowired
    private DataSource dataSource;

    private final String userQuery =
            "SELECT login, password, TRUE "
                    + "FROM users "
                    + "WHERE login = ?";
```

```
    private final String authoritiesQuery =
            "select login, role_of_user " +
                  " from users where login = ?;";


    @Autowired
    public void configureGlobal(
      AuthenticationManagerBuilder auth)
            throws Exception {
        auth
                .jdbcAuthentication()
                .dataSource(dataSource)
                .usersByUsernameQuery(userQuery)
                .authoritiesByUsernameQuery(authoritiesQuery);
    }

    protected void configure(HttpSecurity http)
            throws Exception {
        http
                    .csrf().disable()
                    .authorizeRequests()
                    .antMatchers("/css/**",
                        "/fonts/**",
                        "/image/**",
                        "/js/**")
                                .permitAll()
                    .anyRequest().authenticated()
                .and()
                    .formLogin()
                    .loginProcessingUrl("/login").permitAll()
                    .usernameParameter("login")
                    .passwordParameter("password")
                            .successHandler(
      new MemoriesRestAuthenticationSuccessHandler())
                    .failureHandler(
new SimpleUrlAuthenticationFailureHandler())
                .and()
                    .logout()
                    .logoutUrl("/logout");
    }
}
```

Клас відмічено двома анотаціями – *@Configuration*, *@EnableWebSecurity*. При використанні першої анотації – Spring Framework генерує автоматично усі необхідні конфігураційні моменти. Друга анотація дія таким же чином, але згенерована конфігурація має відношення до безпеки веб рівня додатку.

Поле типу *DataSource* із анотацією *@Autowired* буде автоматично прив'язано Spring Framework. Цей клас реалізує доступ до бази даних у контейнері сервлетів. Так як Spring MVC базується на основі сервлетів, то деплой (установка) відбувається до контейнеру сервлетів.

Далі йдуть два поля, значення яких відповідають двом SQL запитам.

Перший – це запит логіну, паролю та поля «чи активовано користувача» (в нашій системі ця можливість не потрібна – тому *TRUE*). Другий – отримання прав користувача по логіну (авторізаційних даних).

Метод *configureGlobal()* має в якості параметру клас *AuthenticationManagerBuilder*. В цьому класі ми виставляємо тип аутентифікація (через JDBC), об'єкт *DataSource*, та попередні два запити. Потрібно відмітити, що метод помічен анотацією *@Autowired*, це зроблено для того, щоб Spring Framework автоматично зробив прив'язку біна, який буде створений модулем Spring Security.

Метод *configure()* приймає параметр *HttpSecurity*, та за допомогою нього і налаштовується потрібна поведінка рівня безпеки. Вимикаємо CRRF (Cross Site Request Forgery – міжсайтова підробка запитів). Цей рівень безпеки потрібно вимкнути, так як запроси будуть приходити від клієнта. Якщо не вимкнути – то будуть оброблятися запити лише з того серверу, на якому і розташовано додаток. Потім дозволяємо отримувати усім файли із статичним вмістом. Але для створення усіх інших запитів потрібно бути аутентифікованим.

В якості шляху аутентифікація буде використовуватися звичайна форма для аутентифікація, до речі, Spring Security сам згенерує html сторінку для аутентифікації.

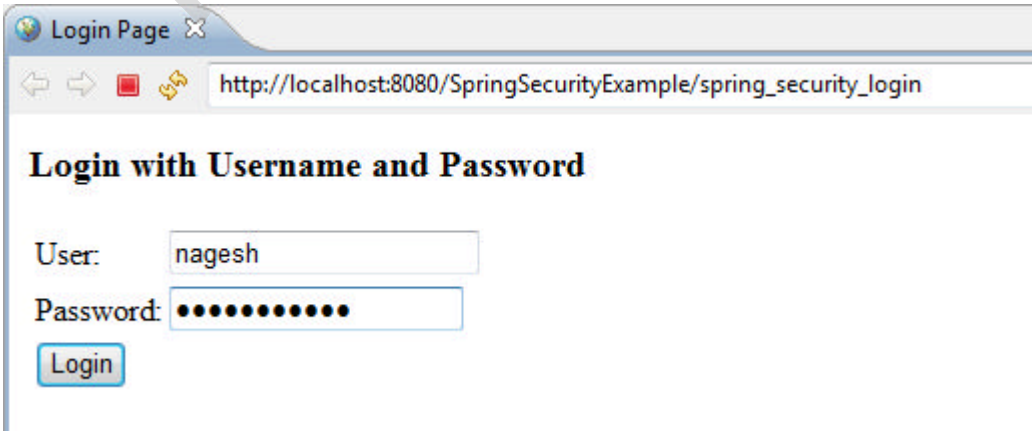

Рисунок 4.3 – Автоматично згенерована сторінка аутентифікації

Далі установлюється путь відносно url серверу для аутентифікації –

«*/login*», імена полів ім’я – користувача та пароля користувача (атрибут name у html тегу <*input*>). Та класи, які налаштовують модель поведінки при успішний та безуспішній аутентифікації.

Останнє – це встановлення url для завершення сесії (*logoutUrl()*).

Сесія та кукі. При успішній аутентифікації сервер відправляє нам таку відповідь.

```
HTTP/1.1 200 OK

Server: Apache-Coyote/1.1
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Set-Cookie: JSESSIONID=11C983E2A0EB79231EFA034E21BB5BE3;
                  Path=/; HttpOnly
Content-Length: 0
Date: Sun, 07 Oct 2024 21:59:34 GMT
```

Особливу увагу потрібно приділити заголовку *Set-Cookie*, який містить куку із назвою *JSESSIONID*. Як видно із назви куки – вона відповідає за сесію. Ця кука постійно буде зберігатися та оновлюватись у android клієнті, щоб клієнт мав змогу аутентифікуватись та відсилати запити до серверу.

Розробка клієнта на базі Android

Розробку клієнту умовно можна розділити на три етапи:

– створення інтерфейсу користувача;

– створення сервісів для завантаження даних від веб системи;

– створення активностей та адаптерів для маніпуляції отриманими даними.

Створення інтерфейсу користувача

Інтерфейс на платформі android зручно описувати використовуючи XML. Наприклад, цей файл шаблону відповідає за вид кожного із елементів списку спогадів:

Виділимо елементи, які присутні на рисунку:

– ім’я користувача та його фотографія;

– іконка настрою та його опис;

– текст спогаду;

– фотографія спогаду.

Створення сервісів для завантаження даних від веб системи. При описі архітектури на рівні клієнту було вказано, що використовується бібліотека Retrofit. За допомогою неї і виконується зручне отримання даних з мережі від серверу. Головний клас – це генератор служб. Завдяки Retrofit класу *RestAdapter.Builder* ми на основі інтерфейсу маємо змогу отримувати дані. Є інтерфейс, який відповідає за отримування даних о користувачі:

```
public interface UpdateUserInfoService {
    @GET("/ulogin")
    User loadUser(@Query("login") String login);
}
```

Основні моменти в цьому коді – це анотації *@GET* та *@Query*. Перша повідомляє, що запит буде виконуватись через HTTP метод GET, а друга вказує, що запит буде містити параметр запиту. Тобто url такого виду:

http://localhost:9821/ulogin?login=username

На основі цього Retrofit згенерує самостійно код для відправки запиту та отримання результату від серверу. Але потрібно вказати наступне:

```
public static <S> S createService(Class<S> serviceClass,
      Context context) {
        OkHttpClient okHttpClient = new OkHttpClient();
        okHttpClient.interceptors().add(
            new ReceivedCookieInterceptor(context));

        RestAdapter.Builder builder = new RestAdapter.Builder()
                .setEndpoint(BASE_URL)
                .setClient(new OkClient(okHttpClient));

        RestAdapter adapter = builder.build();

        return adapter.create(serviceClass);
    }
```

Для *RestAdapter.Builder* встановлюються параметри, які вказують:

– куди відправляти запити;

– реалізація HTTP клієнту.

OkHttpClient використовується для перехвачування від відповіді куки *JSESSIONID* та збереження її у файлі захищених властивостей.

```
public class ReceivedCookieInterceptor
                  implements Interceptor {

    private final Context context;
```

```
    public ReceivedCookieInterceptor(Context context) {
        this.context = context;
    }

    @Override
    public Response intercept(Chain chain) throws IOException {
        Response originalResponse =
                  chain.proceed(chain.request());

        if (!originalResponse.headers("Set-Cookie").isEmpty()) {
            SecurePreferences securePreferences =
                    new SecurePreferences();

            String[] cookies;
            String[] cookieParts;

            for (String header :
                  originalResponse.headers("Set-Cookie")) {
                cookies = header.split("; ");
                for (String cookie : cookies) {
                    cookieParts = cookie.split("=");
                    if (cookieParts[0].equals("JSESSIONID")) {
                        securePreferences.put("JSESSIONID",
                                    cookieParts[1]);
                        break;
                    }
                }
            }
        }
        return originalResponse;
    }

}
```

Для цього йому потрібно вказати вище наведений клас. Цей клас виконує пошук куки *JSESSIONID* у заголовку *Set-Cookie* та зберігає її у файлі захищених властивостей.

Так як для отримання усіх даних від серверу нам потрібно бути аутентифікованим у системі, то потрібно до кожного запити додавати куку *JSESSIONID*. Наступний клас виконує цю роль.

```
public class AddCookieInterceptor
            implements Interceptor {

    private final Context context;

    public AddCookieInterceptor(Context context) {
        this.context = context;
    }

    @Override
    public Response intercept(Chain chain) throws IOException {
        Request.Builder builder = chain.request().newBuilder();
        SecurePreferences securePreferences =
                  new SecurePreferences();
        String cookie = "JSESSIONID=" +
                  securePreferences.getString("JSESSIONID");
        builder.addHeader("Cookie", cookie);
```

```
            return chain.proceed(builder.build());
        }
    }
```

Достатньо замінити клас перехоплювач при створенні об’єкту *OkHttpClient* на цей і кожен запит буде аутентифікований.

Створення активностей та адаптерів для маніпуляції над отриманими даними

Активності у платформі android – це базові сутності на яких будуються додатки. Наступний код реалізує активність для аутентифікації користувача у веб системі:

```
public class LoginActivity extends Activity {

    private UserInfoStorage userInfoStorage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        userInfoStorage = new UserInfoStorage(this);
        if (userInfoStorage.hasUserInfo())
            authorize();

        setContentView(R.layout.activity_login);

        Button submit = (Button) findViewById(R.id.submit);

        submit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String login =
                        ((EditText)
                findViewById(R.id.login)).getText().toString();
                String password =
                        ((EditText)
                findViewById(R.id.password)).getText().toString();
                if (login.isEmpty() || password.isEmpty()) {
                    showAlertDialog("Hey",
                        Please fill fields login and password!");
                    return;
                }
                authorize(login, password);
            }
        });
    }

    private void authorize() {
        User user = userInfoStorage.loadUserInfo();
        new LoginTask(user.getLogin(), user.getPassword(),
      LoginActivity.this).execute();
    }

    private void authorize(String login, String password) {
        new LoginTask(login, password,
                  LoginActivity.this).execute();
    }
}
```

На екрані є два поля для вводу та кнопка. У цій активності отримується на них посилання та отримується введені дані. Після натискання запускається завдання аутентифікації користувача.

Даний код виконується у власному потоці. У ньому створюється та запускається служба аутентифікації. Тут присутній один важливий момент. Після отримання відповіді запускається механізм callback (зворотній виклик) і, також у власному потоці, виконується функція оновлення даних о користувачі (пошта, ім’я, адрес фотокартки). Якщо все пройшло успішно, то відбувається перехід до екрану зі спогадами, якщо ні – то користувач повинен повторити спробу аутентифікації у додатку.

Перед відтворенням списку спогадів, спочатку активність списку спогадів створює наступний адаптер. Цей адаптер створює кожен елемент списку, отримує посилання на елементи інтерфейсу та заповнює них інформацією спогаду. Спогади отримується шляхом створення нового сервісу та його запуску. Потім спогади відображаються на екрані активності списку спогадів.

```
public class MemoriesListActivity extends ActionBarActivity {

    private ListView listView;
    private MemoriesListAdapter adapter;
    private Picasso picasso;
    private ListView navigList;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_memories_list);

        this.picasso = new Picasso.Builder(this)
                .memoryCache(new LruCache(this))
                .build();

        listView = (ListView) findViewById(R.id.listMemories);
        adapter = new MemoriesListAdapter(this,picasso);
        listView.setAdapter(adapter);
        setSideMenu();
    }

    private void setSideMenu() {
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        loadProfileSideMenu();
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setHomeButtonEnabled(true);
    }
```

```
    private void loadProfileSideMenu() {
        ImageView userPhotoSM = (ImageView)
findViewById(R.id.userPhotoSideMenu);
        TextView userNameSM = (TextView)
findViewById(R.id.userNameSideMenu);

        picasso.load(adapter.getUser().getPhotoUrl())
                .resize(100,100)
                .transform(new CropCircleTransformation())
                .into(userPhotoSM);
        userNameSM.setText(adapter.getUser().getUserName());
    }
}
```

На цьому реалізація рівня клієнту закінчується.

4.2 Захист розробленого програмного забезпечення

Для захисту системи (програми) від санкціонованого копіювання пропонується один із загальнорозповсюджених методів - прив’язка до параметрів комп’ютера [39].

У нашому випадку доцільно організувати прив’язку до жорсткого диска. Технічно прив’язка виконується до серійного номера вінчестера. В Delphi безпосередньо серійний номер можна визначати [40].

```
function TProtect.GetHDDSerial:dword;
var
a,b,SerialNum:dword;
VolumeName : array [0..255] of char;
begin
Result := 0;
if GetVolumeInformation(PChar('c:\'), VolumeName, SizeOf(VolumeName),
@SerialNum, a, b, nil, 0)
then Result := SerialNum;
end;
```

Автентифікацію системи пропонується визначати шляхом порівняння зчитаного номера диска з номером, який було попередньо зареєстровано при встановленні програми, записано до системного реєстру. З метою отримання таких даних може бути використано зчитування з реєстру за допомогою функції [41]:

```
function RegSerialNamder:dword;
uses Registry;
var reg:TRegistry;
begin
reg:=TRegistry.Create;
reg.RootKey:=HKEY_LOCAL_MACHINE;
reg.OpenKey('HardWareSN}', false);
Result:=RegKeyGetDw(HKEY_LOCAL_MACHINE;'HardWareSN');
reg.CloseKey;
reg.Free;
end;
```

Для автентифікації проведення процедури прив'язки (активації) системи до обладнання може проводитися записом номера диска до системного реєстру при встановленні програми за допомогою:

```
procedure Sn_HDD;
uses  Registry;
var reg:TRegistry;
SerialNum,a,b:dword;
VolumeName:array [0..255] of char;
begin
if  GetVolumeInformation(PChar('c:\'),  VolumeName,  SizeOf(VolumeName),
@SerialNum, a, b, nil, 0)
then
begin
reg:=TRegistry.Create;
reg.RootKey:=HKEY_LOCAL_MACHINE;
reg.OpenKey('HardWareSN', false);
RegKeySettDw(HKEY_LOCAL_MACHINE,'HardWareSN',SerialNum);
reg.CloseKey;
reg.Free;
end;
end;
```

Багато робіт присвячено захисту даних, захисту програмного забезпечення та захисту інформацій в цілому [42, 43, 44, 45, 46, 47, 48, 49, 50], проте просто захист від несанкціонованого копіювання робити недоцільно. Бажано давати демонстрацію роботи програми. Наприклад, можна давати час та/або кількість відпрацювань, для чого можна використати реєстр.

```
function Beta:boolean;
uses Registry;
var regD,regK:TRegistry;
```

```
Rd:TDateTime;
Rk:DWORD;
begin
regD := TRegistry.Create;
regD.RootKey := HKEY_LOCAL_MACHINE;
if reg.OpenKey('HardWareData', false)
then
begin
Rd:=RegKeyGetDateTime(HKEY_LOCAL_MACHINE,'HardWareData');
regK:=TRegistry.Create;
regK.OpenKey('HardWareK', true);
Rk:= RegKeyGetDw(HKEY_LOCAL_MACHINE,'HardWareK');
if  Rd+30>Date and Rk>100
then
Result := false
else
begin
RegKeySettDw(HKEY_LOCAL_MACHINE,'HardWareK',Rk+1);
Result := true;
end;
end
else
begin
regD.OpenKey('HardWareData}',true)
RegKeySetDateTime(HKEY_LOCAL_MACHINE,'HardWareD',Date);
regK := TRegistry.Create;
regK.OpenKey('HardWareK}', true)
RegKeySettDw(HKEY_LOCAL_MACHINE,'HardWareK',1);
result := true;
end;
regD.CloseKey;
regD.Free;
regK.CloseKey;
regK.Free;
end;
```

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для розгортання системи потрібен сервер з описаними в попередньому розділі вимогами для фреймворку. Також потрібна база даних MySQL.

Тепер можемо перейти до встановлення проекту. Для початку скачаємо проект за допомогою команди git clone. Наступним кроком буде створення файлу з конфігураціями, у якому потрібно прописати доступи до бази. Після цього можемо встановити проект за допомогою Composer – composer install. Й встановити інші пакети, які потрібні для веб-сайту, – npm install. Далі запуск міграцій – php artisan migrate. Було створено Seeder для збереження користувача-адміністратора. Тому після запуску міграції треба виконати команду - php artisan db:seed.

І, у залежності від домену, можна заходить на веб-сайт, яким являється встановлена система. Заповнювати базу об’єктами, завантажувати файли до них.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для Web-додатка для збереження спогадів з застосуванням фреймворка Spring.

Метою розробки є дослідження та програмна реалізація Web-додатка для збереження спогадів з застосуванням фреймворка Spring.

Об'єктом дослідження є правильні практики створення веб-систем на базі Spring Framework та існуючі методи взаємодій з ними.

Предметом дослідження є Spring Framework як засіб створення веб-систем.

Методи дослідження базуються на методах тестування та навантаження, дослідження безпеки, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

- досліджено основні методи та можливості використання Spring Framework, як потужного інструменту для створення гнучких і безпечних додатків у різних галузях, включаючи розподілені системи та високонавантажені корпоративні рішення;

- із використанням JPA та Spring Data JPA реалізовано рівень даних додатку;

- створена взаємодія з веб системою по HTTP протоколу через розробку контролерів Spring MVC;

- розроблена система безпеки додатку із використанням Spring Security;

- створено клієнт для відображення даних від веб системи на базі платформи Android.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та програмної реалізації Web-додатка для збереження спогадів з використанням фреймворка Spring можуть бути цікавими для різних груп людей і організацій, включаючи(рис. 7.1).

Розробники, які працюють з Java і хочуть дізнатися про можливості фреймворка Spring для створення веб-додатків.
Особи, які навчаються розробці програмного забезпечення та зацікавлені в практичних проектах.
Фахівці, які вивчають механізми збереження пам'яті, можуть використовувати результати для аналізу впливу технологій на спогади та їхню збереженість.
Вивчення того, як технології впливають на соціальну взаємодію та спогади в різних культурах.
Підприємці, які планують запускати сервіси для збереження спогадів або цифрових альбомів.
Організації, які надають послуги збереження пам'яті, можуть зацікавитися новими технологіями.
Люди, які хочуть зберігати спогади у цифровому форматі (фото, відео, тексти).
Сім'ї, які прагнуть зберігати спогади про важливі події (дитинство, святкування тощо).
Університети та дослідницькі центри, які можуть використовувати результати для подальшого вивчення технологій збереження спогадів.
Для дослідження ефекту технологій на пам'ять і спогади це буде цікаво інститутам психології.
Компанії, які пропонують хостинг для веб-додатків, можуть зацікавитися реалізацією подібних рішень для своїх клієнтів.
Компанії, які виготовляють пристрої для зберігання даних, можуть використовувати результати для поліпшення своїх продуктів.
Організації, які займаються збереженням культурних спадщин та спогадів, можуть використовувати такі рішення для вдосконалення своїх процесів.

Рисунок 7.1 – Цільова аудиторія

Результати дослідження та реалізації Web-додатка для збереження

спогадів на основі фреймворка Spring мають широкий спектр потенційних користувачів і зацікавлених сторін, від розробників до звичайних користувачів, науковців та підприємців. Це дозволяє створювати різноманітні можливості для співпраці та впровадження нових технологій у цій галузі.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Оцінка привабливості проекту програмної реалізації Web-додатка для збереження спогадів з використанням фреймворка Spring може бути проведена за допомогою експертних оцінок. Цей процес передбачає залучення фахівців з різних областей, які оцінюють проект за визначеними критеріями.

Обираємо критерії, які будуть використані для оцінки привабливості проекту (рис. 7.2).

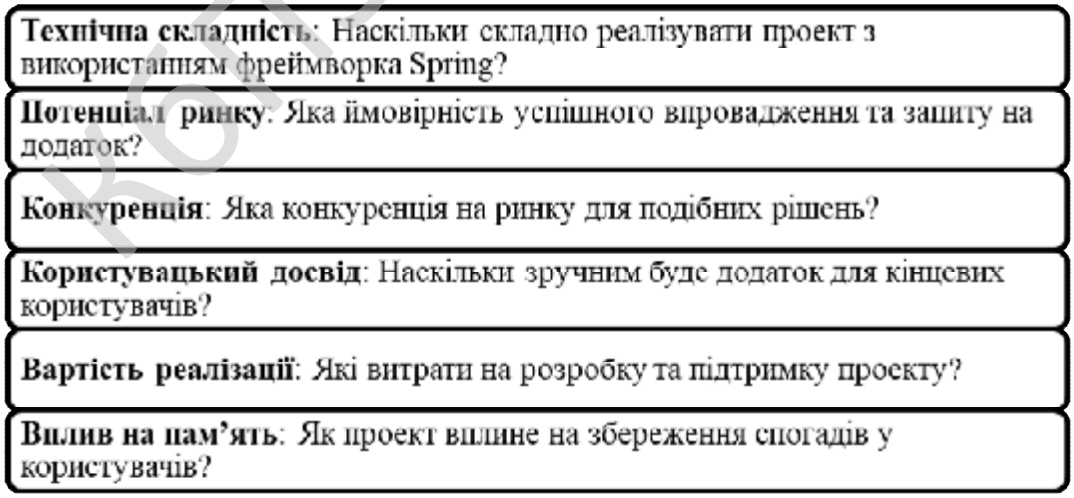

Рисунок 7.2 – Критерії експертної оцінки

Формуємо групу експертів у відповідних галузях: розробники програмного забезпечення (для оцінки технічної складності), маркетологи (для оцінки потенціалу ринку та конкуренції), дизайнери (для оцінки

користувацького досвіду), фінансисти (для оцінки вартості реалізації).

Використовуємо анкету для збору оцінок за обраними критеріями. Експерти можуть оцінювати кожен критерій за шкалою, наприклад, від 1 до 5 (де 1 – дуже погано, 5 – дуже добре). Після збору оцінок, підраховуємо середні значення для кожного критерію. Це дозволить отримати загальну картину привабливості проекту.

Формулюємо загальну оцінку проекту, враховуючи вагу кожного критерію. Дані зводимо в таблицю 7.1.

Таблиця 7.1 – Зведені результати експертних оцінок

Критерій	**Оцінка (1-5)**	**Вага (%)**	**Вагова оцінка**
Технічна складність	4	20	0.8
Потенціал ринку	5	30	1.5
Конкуренція	3	15	0.45
Користувацький досвід	4	20	0.8
Вартість реалізації	3	10	0.3
Вплив на пам’ять	4	5	0.2
Загальна оцінка		100	4.0

Загальна оцінка проекту становить 4.0 з 5, що вказує на високу привабливість проекту для реалізації. Таким чином, експертні оцінки можуть слугувати надійним інструментом для визначення потенціалу проекту та прийняття рішень щодо його подальшої реалізації.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості програмної реалізації Web-додатка для збереження спогадів з використанням фреймворка Spring можна використовувати кілька методів. Вибір методу оцінки вартості залежить від ресурсів, термінів та наявності даних. Якщо є доступ до історичних даних аналогічних проектів, метод аналогій може бути корисним. Для детальної та обґрунтованої оцінки вартості, метод кошторису або метод функціональних точок є більш підходящими. Якщо є прагнення врахувати ризики, варто використовувати метод трьох оцінок. Залежно від специфіки проекту, комбінація кількох методів може бути найбільш ефективною для отримання точнішої оцінки вартості. Наприклад, можна використовувати метод функціональних точок для первинної оцінки, а потім уточнити результати за допомогою методу експертних оцінок або кошторису.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Впровадження Web-додатка для збереження спогадів з використанням фреймворка Spring може принести економічну ефективність в різних аспектах (рисунок 7.3).

Таким чином, впровадження Web-додатка для збереження спогадів з використанням фреймворка Spring може привести до значної економічної ефективності, що проявляється в збільшенні доходів, зниженні витрат і поліпшенні користувацького досвіду.

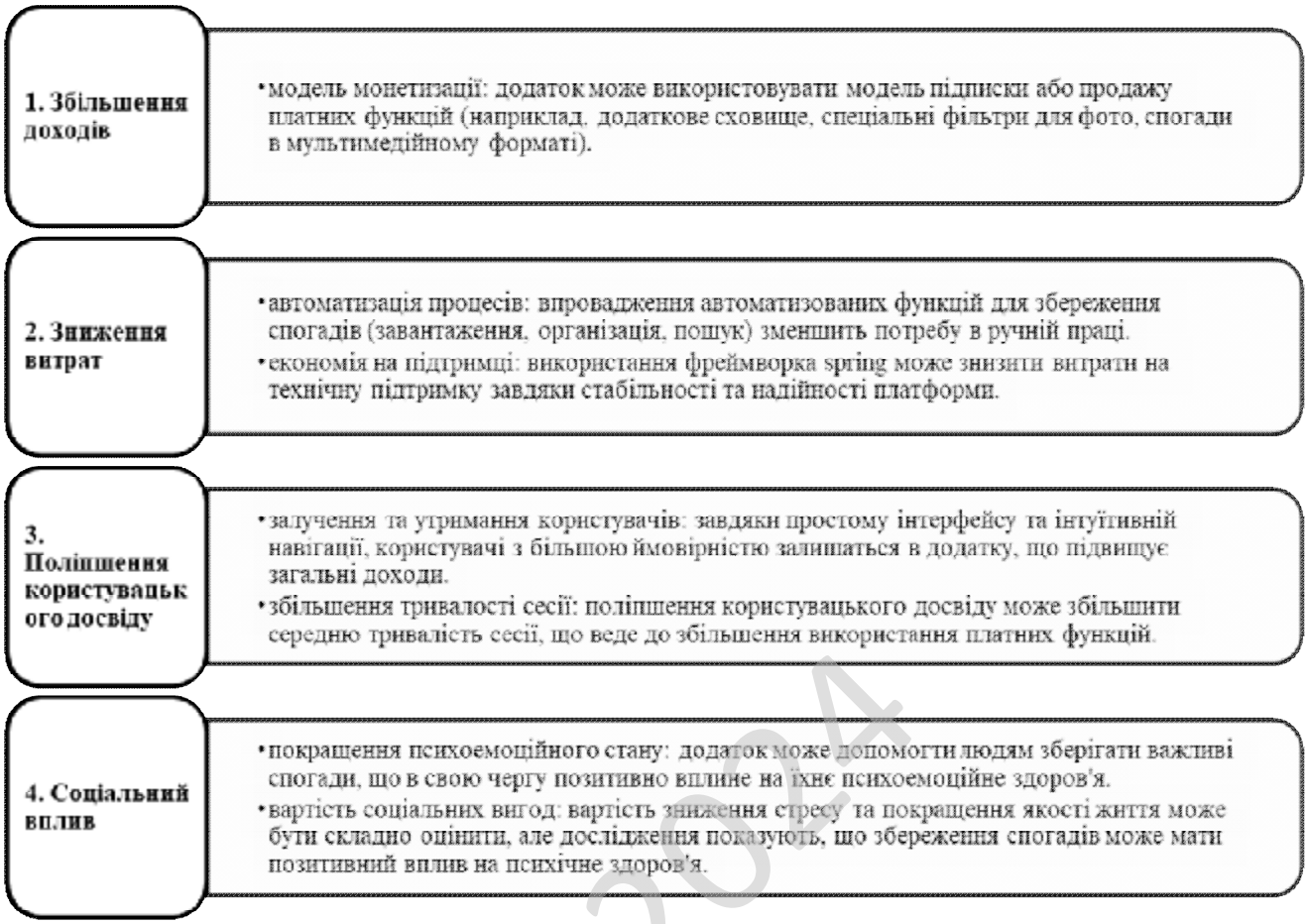

Рисунок 7.3 – Ключові аспекти ефективності впровадження проєкту для клієнта

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Алгоритм просування проєкту програмної реалізації Web-додатка для збереження спогадів з використанням фреймворка Spring. Цей алгоритм охоплює основні етапи, які допоможуть привернути увагу до вашого продукту і залучити користувачів (рисунок 7.4).

Цей алгоритм просування проєкту охоплює всі основні етапи, від дослідження ринку до постійного оновлення продукту. Дотримуючись цього плану, ви зможете успішно просувати ваш Web-додаток для збереження спогадів і залучати нових користувачів.

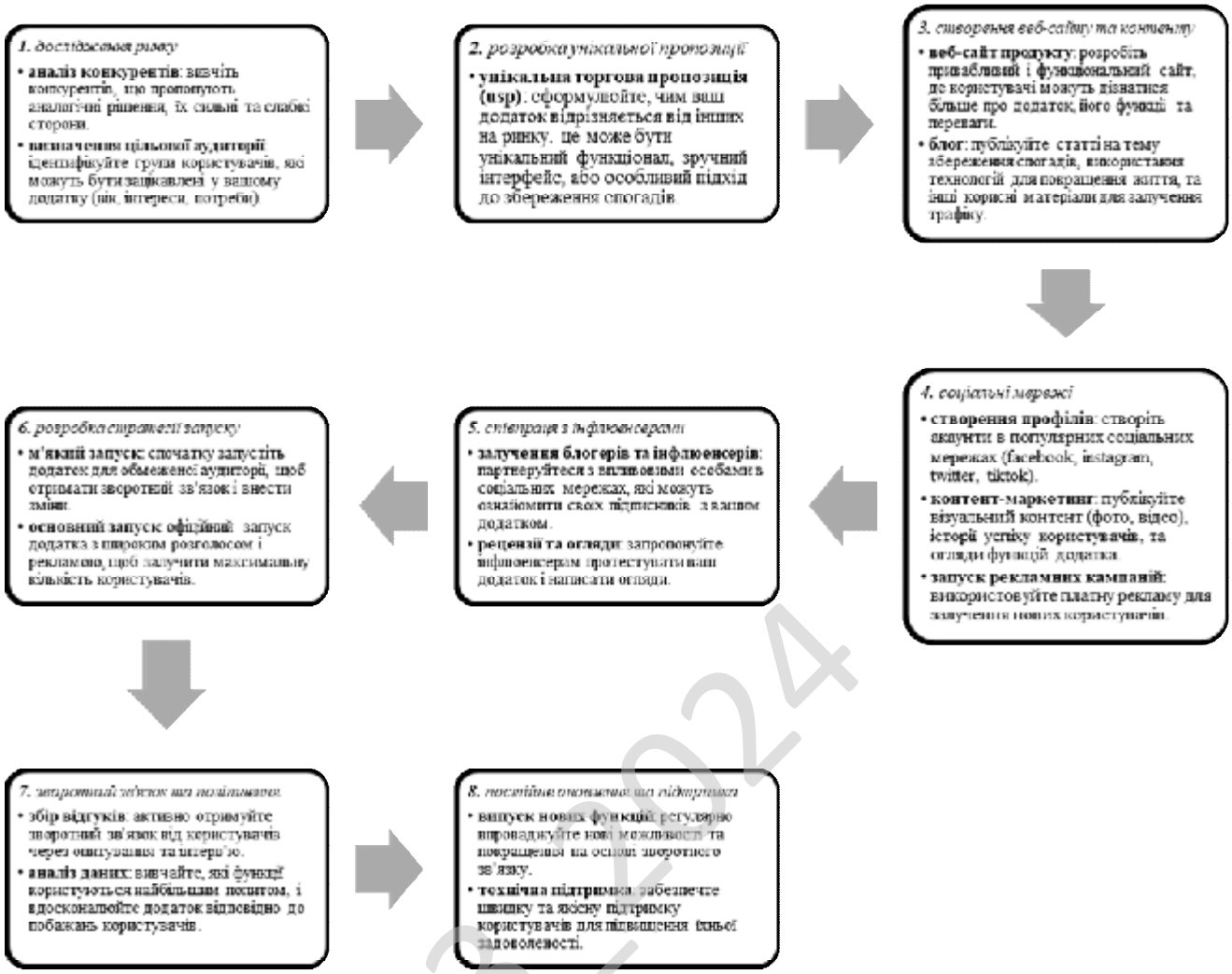

Рисунок 7.4 – Алгоритм просування проєкту

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту та шляхів реалізації проєкту програмної реалізації Web-додатка для збереження спогадів із застосуванням фреймворка Spring можна застосувати перераховані на рисунку 7.5 стратегії.

Оптимізація каналів збуту та шляхів реалізації вашого Web-додатка для збереження спогадів потребує комплексного підходу. Використання мультіканальної стратегії, активне залучення партнерів, аналіз зворотного зв'язку, а також використання даних для покращення продукту дозволять вам максимально ефективно просувати ваш проект.

Рисунок 7.5 – Шляхи оптимізації каналів збуту

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключові фактори успіху проєкту програмної реалізації Web-додатка для збереження спогадів із застосуванням фреймворка Spring можуть включати:

- якість продукту;
- задоволення потреб користувачів;
- маркетинг і просування;
- технічна реалізація;
- підтримка та обслуговування;
- фінансова стабільність;
- аналіз та адаптація.

Варто зосередитись більш детально на кожному ключовому факторі успішності реалізації проєкту.

Питання якості продукту має під собою розуміння, що додаток повинен бути стабільним і безпечним для користувачів, адже він зберігає особисті спогади та дані; швидкість завантаження та обробки даних повинні відповідати сучасним стандартам, щоб користувачі не відчували затримок.

В рамках задоволення потреб користувачів має бути простий та

зрозумілий інтерфейс, що забезпечує зручний доступ до функцій додатка, обов'язковим є включення функцій, які відповідають вимогам цільової аудиторії, наприклад, можливість створення колекцій спогадів, інтеграція з соціальними мережами тощо.

Ефективна стратегія маркетингу передбачає правильний підбір каналів просування, які допоможуть залучити цільову аудиторію, а також активна присутність у соціальних мережах, залучення впливових осіб для популяризації додатка серед потенційних користувачів.

Використання сучасних технологій дозволить забезпечечити надійність, масштабованості та гнучкості додатка. Використання CI/CD практик для регулярного впровадження нових функцій та виправлення помилок.

Наявність служби підтримки, що швидко реагує на запити користувачів і допомагає вирішувати проблеми, а постійне вдосконалення продукту, впровадження нових функцій та виправлення помилок на основі зворотного зв'язку від користувачів.

Залучення доходів через підписки, рекламу чи продаж додаткових функцій, що допоможе забезпечити фінансову стабільність проєкту, має бути оптимізація витрат на розробку, маркетинг та підтримку, щоб максимізувати прибуток.

Регулярний моніторинг користувацької активності та вподобань для коригування стратегії розвитку продукту та швидка адаптація до змін на ринку та в технологіях дозволить залишатися конкурентоспроможним.

Ці фактори є критично важливими для успішної реалізації Web-додатка для збереження спогадів. Зосередившись на якості продукту, задоволенні потреб користувачів, ефективному маркетингу та постійній підтримці, ви зможете створити успішний і стійкий проєкт.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Аналізуючи умови працівників it-сфери, на перший погляд, може здатися, що працівники сфери інформаційних технологій не схильні до ризиків на виробництві, та якщо більш глибоко розглянути умови і специфіку праці фахівців сфері it-індустрії, можна виявити ряд факторів які будуть мати негативний сплив на стан охорони праці, та на самого it-фахівця зокрема. Сюди можна віднести як невідповідність освітлення, так і високий рівень шуму, що негативно позначатимуться як на емоційному так і на фізичному стані фахівця, призводитимуть до зниження ефективності праці та виробничих травм. Також, важливим моментом охорони праці it-фахівця є врахування його психологічних можливостей (швидкість реакції, особливості пам'яті та уваги, емоційний стан, тощо). Для того, щоб забезпечити ефективну роботу it-фахівця, потрібно враховувати та максимально компенсувати такі негативні фактои як: надмірне нервово-емоційне навантаження, довготривалі статичні перевантаження, обмежена рухова активність. Всі ці чинники призводить до різноманітних відхилень у стані здоров’я, зокрема до перевтоми, зниження фізичної та розумової працездатності, неврозів, захворювань серцево-судинної системи тощо. Метою даного розділу є огляд конкретних умов праці спеціаліста у сфері it-індустрії. Завданнями для даного розділу є: аналіз умов праці на робочому місці фахівця it-індустрії, розробка конкретних рекомендацій щодо покрашення умов праці фахівців it-індустрії, огляд пожежної безпеки на it-підприємстві та розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ – фахівці.

8.2. Аналіз умов праці на робочому місці ІТ-фахівця

На робочому місці ІТ-фахівця (або програміста) виникають небезпечні та шкідливі для безпечної життєдіяльності фактори:

– підвищений рівень шуму;

– несприятливі мікрокліматичні умови;

– недостатній рівень освітленості;

– шкідливі речовини;

– підвищений рівень електромагнітних випромінювань радіочастот;

– висока напруга електричної мережі;

– статична електрика та інші.

Робота програміста супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи. Таким чином, вивчення умов праці на робочому місці програміста є необхідною умовою запобігання негативних наслідків впливу небезпечних та шкідливих факторів. Робоче місце, добре пристосоване до трудової діяльності інженера, правильно і доцільно організоване, щодо простору, форми, розміру забезпечує йому зручне положення при роботі і високу продуктивність праці при найменшому фізичному і психічному напруженні.

Нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт. Для постійних робочих місць, якими є робочі місця ІТ-фахівців, встановлені оптимальні параметри мікроклімату, а за неможливості їх дотримання використовують допустимі параметри. Робота ІТ-фахівця за важкістю відноситься до Іа (роботи, що виконуються сидячи і не потребують фізичного напруження) та Іб (роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним напруженням) категорій. В таблиці 8.1. наведені оптимальні параметри мікроклімату в приміщеннях.

Таблиця 8.1 – Параметри мікроклімату для приміщень з ПК

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	22...24°С; 40... 60%; до 0,1 м/с
Теплий	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	23...25 °С 40...60% 0,1...0,2 м/с

Виміряні за допомогою приладів температура та вологість у приміщеннях праці ІТ-фахівців повинні відповідати зазначеним у таблиці для теплого періоду року. Слід зазначити, що для нормалізації параметрів мікроклімату слід використовувати у приміщеннях кондиціювання повітря, або забезпечити подачу свіжого повітря системами вентиляції. Норми подачі свіжого повітря наведені у таблиці 8.2.

Таблиця 8.2 – Норми подачі свіжого повітря в приміщення

Характеристика приміщення	Об'ємна витрата свіжого повітря, що подається в приміщення, $м^3$ на одну людину в годину
Об'єм до 20 $м^3$ на людину 20... 40 $м^3$ на людину Більше 40 $м^3$на людину	Не менше 30 Не менше 20 Може біти використана природна вентиляція

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення його привабливості, позитивно

впливає на продуктивність праці. Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою. У службових приміщеннях, у яких виконується одноманітна розумова робота, що вимагає значної нервової напруги і великого зосередження, забарвлення повинно бути спокійних тонів – малонасичені відтінки холодного зеленого або блакитного кольорів.

При розробці оптимальних умов праці програміста необхідно враховувати освітленість. Раціональне освітлення робочого місця є одним з найважливіших факторів, що впливають на ефективність трудової діяльності людини, що попереджають травматизм і професійні захворювання. Правильно організоване освітлення створює сприятливі умови праці, підвищує працездатність і продуктивність праці. Освітлення на робочому місці програміста повинно бути таким, щоб працівник міг без напруги зору виконувати свою роботу. Стомлюваність органів зору залежить від ряду причин: недостатність освітленості; надмірна освітленість; неправильний напрям світла. Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань. [2]

8.3 Пропозиції щодо підвищення працездатності ІТ – фахівців

Поява та впровадження нових інформаційно-комунікаційних технологій зумовлює необхідність подальшого вдосконалення охорони праці фахівців it-індустрії. Все це потребує розробки нових нормативно-правових актів з регламентації праці та відпочинку фахівців it-індустрії і

стандартів підприємств, центрів комп'ютерної техніки, центрів інформаційних технологій, сучасних комп'ютерних класів. Для підвищення розумової працездатності то зорової роботи повинна здійснюватися ергономічна оптимізація в рамках системи «оператор-термінал», яка сприятиме результативній фізичній та інтелектуальний працездатності і відновленню психосоматичного здоров'я фахівців it-індустрії. Всі наведені заходи щодо вдосконалення охорони праці фахівців it-індустрії повинні контролюватися службою охорони праці та комісією з охорони праці підприємства. Особливе значення у соціальному захисті цієї категорії працівників належить прийняття комплексного договору, який може забезпечити фахівців додатковими пільгами та компенсаціями.

Для більшого розуміння, пропозиції щодо підвищення працездатності it-фахівців, розіб'ємо на декілька категорій:

1 Середовище і розпорядок праці. Для мінімізації негативних ефектів, що пов'язані з перевтомленням it-фахівців, потрібно чітко прописати і реалізувати графік періодів праці-відпочинку, щоб фахівець міг можливість переключити увагу, дати можливість відпочити очам, мозку, елементарно, встати розім'яти ноги. Також потрібно зробити максимально комфортними умови мікроклімату у офісному приміщенні, де працюють it-фахівці. Мається на увазі встановлення і експлуатація, коли виникає необхідність, кондиціонерів, опалення, та системи вентиляції, задля попередження перегрівання, переохолодження it-фахівців, і подальшої неможливості ними виконувати свої функції. Також, за можливості, нами пропонується введення практики віддаленої праці it-фахівцями, якщо роботодавець не може забезпечити оптимальні і безпечні умови в офісному приміщенні, або якщо фахівця вони не влаштовують із певних причин.

2 Фізичні і психоемоційні чинники. Першим і найважливішим чинником, що впливає на працездатність it-фахівців є робоче місце, і саме тому, роботодавець має забезпечити максимальний його комфорт і безпеку. Гарантією цих факторів може слугувати сертифікація меблів, що

використовуються на підприємстві it-галузі. Тому нами пропонується закупівля тільки меблів , які пошли сертифікацію на відповідність. Під психоемоційними чинниками ми розуміємо гарне самопочуття фахівців, позитивний настій, гарний психологічний клімат у колективі, тощо. Задля того, щоб психоемоційні чинники мали максимально позитивний ефект, керівництву слід поводити заходи, які сприятимуть укріпленню і покращенню міжособистісних стосунків у колективі, таких як психологічні тренінги, тімбілдінг, спортивні змагання і естафети. Також, сюди можна віднести розробку і впровадження системи мотивації працівників, як фінансової, так моральної і адміністративної.

8.4 Розрахунок системи загального штучного освітлення виробничого приміщення де працюють ІТ-фахівці

Приміщення з ПК повинні мати природне і штучне освітлення, яке відповідало б вимогам ДБН В.2.5-28-2006 «Природне і штучне освітлення», ДСАНПІН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин». Приміщення для роботи із ПЕОМ повинні мати природне й штучне освітлення. Віконні прорізи повинні бути орієнтовані на північ або на північний схід, забезпечувати коефіцієнт природної освітленості (К.П.О.) не менш 1,5% і мати жалюзі або штори. Віконні прорізи повинні мати регульовані пристрої для відкривання, а також жалюзі, завіски, зовнішні козирки тощо. Приміщення із ПЕОМ повинні бути обладнані системою загального рівномірного освітлення. У виробничих і адміністративно-суспільних 130 приміщеннях, де переважно ведеться робота з документами, допускається комбінована система штучного освітлення. Штучне освітлення має здійснюватися системою загального рівномірного освітлення, яка включає суцільні або такі, що перериваються лінії світильників, розташованих збоку робочих місць (переважно ліворуч), паралельно лінії зору користувачів ПК. Світильники повинні мати

розсіювачі світла. У світильниках місцевого освітлення можна використовувати лампи накалювання. При розміщенні ПК по периметру приміщення лінії світильників штучного освітлення повинні розміщуватися локально над робочими місцями. Система освітлення робочого місця користувача ПК має відповідати наступним вимогам (рис. 8.1).

Рисунок 8.1 – Вимоги до системи освітлення робочого місця користувача ПК

Освітленість на робочому столі користувача в зоні розташування документів має бути в межах 300-500 лк. Якщо цей рівень освітленості неможливо забезпечити системою загального освітлення то допускається застосування світильників місцевого освітлення, але при цьому не повинно бути відблисків на поверхні екрану (яскравість відблисків не повинна перевищувати 40 кд/м^2) та перевищення його освітленості більше ніж 300 лк. Яскравість світильників загального освітлення, а також яскравість стелі

при застосуванні системи відбитого освітлення не повинна перевищувати 200 кд/м^2. Величина коефіцієнта пульсації освітленості не повинна перевищувати 5%. Що стосується розподілу яскравості в полі зору працюючих за дисплеями ПК, то відношення значень яскравості робочих поверхонь не повинно перевищувати 3:1

Проведемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 6 м, довжина – 7 м, висота – 2,9 м. У зазначеному приміщенні працює 7 людей.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою [1]:

$$F=ESKZ/n,$$

де:

F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

S – площа освітлюваного приміщення (у нашому випадку *S*=6 х 7 = 42 *м*^{2});

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z = 1,1*);

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці [8]; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін **($\rho_{стін.}$)** і стелі **($\rho_{стелі}$),** значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де:

S – площа приміщення, *S* = 42 *м²;*

h – розрахункова висота підвісу, *h* = 2,9 *м (співпадає з в*исотою стелі, т.я. лампи освітлення закріплюються на стелі)*;*

A – ширина приміщення, *A = 6 м;*

B – довжина приміщення, *B* = 7 *м.*

Підставимо всі значення у формулу та визначемо індекса приміщення:

$$i=1,4.$$

Знаючи індекс приміщення, за знаходимо n = 0,29 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом лампам) [*8].* Підставимо всі значення у формулу, визначемо світловий потік: *F=71689 Лм.*

Для розрахунку дудемо використовувати *світлодіодні стельові панелі Delux LED Panel 41 44Bm.*, світловий потік яких $F_л$ = *3600* Лм.

Число ламп визначається по формулі:

$$N=F/F_л$$

де:

F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначемо індекса приміщення: *N= 71689 / 3600=19,9 шт.*

Приймаємо необхідну кількість *світлодіодних світильників* 20 шт.

8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню

здоров’я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз умов праці на робочому місці ІТ-фахівця, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров’я програміста. Розроблено заходи з умов поліпшення охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для Web-додатка для збереження спогадів людини.

Було реалізовано систему, яка відповідає технічному завданню, яке було описане на початку роботи.

Була досягнута мета – розробка Web -додатка для збереження спогадів людини на базі Spring Framework, а також були виконані наступні завдання:

– розглянуто сучасний стан Spring Framework;

– спроектовано архітектуру веб системи;

– спроектовано структуру бази даних;

– із використанням JPA та Spring Data JPA реалізовано рівень даних додатку;

– була створена взаємодія з веб системою по HTTP протоколу через розробку контролерів Spring MVC;

– була розроблена система безпеки додатку із використанням Spring Security;

– було створено клієнт для відображення даних від веб системи на базі платформи Android.

Веб-додаток - це самостійна система, до якої можуть авторизуватися користувачі. В залежності від ролі користувача він має різні права в системі. База даних, використана у додатку, - MySQL. Створення таблиць було реалізовано за допомогою міграцій з фреймворку, а запити до неї через Eloquent ORM.

Для верстки було використано Bootstrap, безкоштовний набір інструментів із відкритим кодом, призначений для створення веб-сайтів та веб-додатків

Залучення доходів через підписки, рекламу чи продаж додаткових функцій має забезпечити фінансову стабільність проекту але має бути оптимізація витрат на розробку, маркетинг та підтримку, щоб максимізувати прибуток.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Our Customers [Електронний ресурс] / Pivotval Inc. – Режим доступу: http://pivotal.io/customers. – Загол. з екрану.
2. Rod J. Framework Reference Documentation [Електронний ресурс] / Johnson R., Hoeller J. , Donald K. – Режим доступу: http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/. – Загол. з екрану.
3. Інверсія управління [Електронний ресурс] / Wikipedia.com – Режим доступу: http://uk.wikipedia.org/wiki/Інверсія_управління . – Загол. з екрану.
4. Aspect Oriented Programming with Spring [Електронний ресурс] / Режим доступу: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html . – Загол. з екрану.
5. Web MVC framework [Електронний ресурс] / Режим доступу: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html . – Загол. з екрану.
6. Spring Security Reference Documentation [Електронний ресурс] / Режим доступу: http://docs.spring.io/spring-security/site/docs/3.0.x/reference/springsecurity.html . – Загол. з екрану.
7. Spring Data – One API To Rule Them All? [Електронний ресурс] / Режим доступу: http://www.infoq.com/articles/spring-data-intro . – Загол. з екрану.
8. Spring Data Jpa Reference Documentation [Електронний ресурс] / Режим доступу: http://docs.spring.io/spring-data/jpa/docs/1.8.0.RELEASE/reference/html/ . – Загол. з екрану.
9. Крейг Воллс. Spring in Action, 4 видання / Крейг В. - Manning, 2014. – 624 с.
10. Wheeler W. Spring in Practice / W. Wheeler, J. White – Manning, 2013. – 560с.

11. Lui H. H. Spring 4 for Developing Enterprise Applications: An End-to-End Approach / H. H. Lui – CreateSpace Independent Publishing Platform – 432c.

12. Amuthan G. Spring MVC Beginner's Guide / G. Amuthan – PACKT – 304 c.

13. Winch R. Spring Security 3.1 / R. Winch, P. Mularien – PACKT – 456 c.

14. Kainulainen P. Spring Data / P. Kainulainen – PACKT – 160 c.

15. How FriendFeed uses MySQL to store schema-less data [Електронний ресурс] / Режим доступу: http://backchannel.org/blog/friendfeed-schemaless-mysql. – Загол. з екрану.

16. Maven. Часть 2 – Dependency [Електронний ресурс] / Режим доступу: http://devcolibri.com/769. – Загол. з екрану.

17. Spring Data JPA. Работа с БД. Часть 1 [Електронний ресурс] / Режим доступу: http://devcolibri.com/3966. – Загол. з екрану.

18. Spring Data JPA. Пишем DAO и Services. Часть 2 [Електронний ресурс] / Режим доступу: http://devcolibri.com/4149. – Загол. з екрану.

19. Package Index Reference Android API LEVEL 9 [Електронний ресурс] / Режим доступу: http://developer.android.com/reference/packages.html . – Загол. з екрану.

20. Up and running with material design [Електронний ресурс] / Режим доступу: http://developer.android.com/design/index.html . – Загол. з екрану.

21. User Interface [Електронний ресурс] / Режим доступу: https://developer.android.com/guide/topics/ui/index.html. – Загол. з екрану.

22. Передача даниxx з одного Activity на інше Activity [Електронний ресурс] / Режим доступу: http://devcolibri.com/2132 . – Загол. з екрану.

23. Захист програм від злому [Електронний ресурс] - http://easy-code.com.ua/2011/04/zaxist-program-vid-zlomu/

24. Офіційна документація Spring: [Електронний ресурс] / Режим доступу: https://docs.spring.io/spring-framework/docs/current/reference/html/

25. Документація з Spring Security: [Електронний ресурс] / Режим доступу: https://docs.spring.io/spring-security/reference/

26. Офіційна документація від Oracle для Java SE: [Електронний ресурс] / Режим доступу: https://docs.oracle.com/en/java/javase/

27. Серія уроків та статей про Spring Security: [Електронний ресурс] / Режим доступу: https://www.baeldung.com/spring-security

28. Підручники по розробці Android-додатків: [Електронний ресурс] / Режим доступу: https://www.vogella.com/tutorials/android.html

29. Спільнота, де можна знайти відповіді на питання, пов'язані з Java: [Електронний ресурс] / Режим доступу: https://stackoverflow.com/questions/tagged/java

30. Репозиторій Spring на GitHub: [Електронний ресурс] / Режим доступу: https://github.com/spring-projects/spring-framework

31. Офіційна документація для розробників Android: [Електронний ресурс] / Режим доступу: https://developer.android.com/guide

32. Офіційна документація для розробників Android: [Електронний ресурс] / Режим доступу: https://developer.android.com/guide

33. Приклад налаштування аутентифікації за допомогою JDBC в Spring Security: [Електронний ресурс] / Режим доступу: https://mkyong.com/spring-boot/spring-security-jdbc-authentication-example/

34. Керівництво з найкращих практик у Spring Security: [Електронний ресурс] / Режим доступу: https://dzone.com/articles/spring-security-best-practices

35. Alexander Osterwalder, Yves Pigneur – Business Model Generation. Wiley, 2010. – 288 стор.

36. Eric Ries – The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business, 2011. – 336 стор.

37. Steve Blank, Bob Dorf – The Startup Owner's Manual: The Step-by-Step Guide for Building a Great Company. K&S Ranch, 2012. – 608 стор.

38. Philip Kotler – Marketing Management (15th Edition). Pearson, 2015. – 832 стор.

39. Kenneth C. Laudon, Jane P. Laudon – Management Information Systems: Managing the Digital Firm (16th Edition). Pearson, 2020. – 688 стор.

40. Олексій Васильєв - Програмування мовою Java. Навчальна книга – Богдан. 2019. – 696 стор.

41. Кеті Сьєрра, Берт Бейтс - Інтерактивний посібник з основ Java з практичними завданнями. Фабула. 2021. – 720 стор.

42. Герберт Шилдт - Java. Бібліотека професіонала. Видавництво Старого Лева. 2020. -1136 стор.

43. Жидецький В.Ц. Основи охорони праці: підруч. 3-є вид., перероб і доп. Львів : УАД, 2006. 336 стор.

44. Босов Є.П., Жесан Р.В., Каліч В.М., Голик О.П., Зубенко В.О. Охорона праці при проектуванні систем автоматизації виробництва : навч. посіб. 2-е вид., перероб. і доп. Кропивницький : ЦНТУ, 2022. – 208 стор..

45. Жидецький В.Ц., Джигирей В.С., Сторожук В.М., Туряб Л.В., Лико Х.І. Практикум з охорони праці. Львів : Афіша, 2000. – 352 стор.

46. Конституція України. [Електронний ресурс] / Режим доступу: https://zakon.rada.gov.ua/laws/main/254%D0%BA/96-%D0%B2%D1%80.

47. Про охорону праці : Закон України. [Електронний ресурс] / Режим доступу: https://zakon.rada.gov.ua/laws/main/2694-12#Text.

48. Основи законодавства України про охорону здоров'я : Закон України. [Електронний ресурс] / Режим доступу: https://zakon.rada.gov.ua/laws/show/2801-12#Text.

49. Про систему громадського здоров'я : Закон України. [Електронний ресурс] / Режим доступу: https://zakon.rada.gov.ua/laws/show/2573-20#n840

50. Кодекс цивільного захисту України. [Електронний ресурс] / Режим доступу: https://zakon.rada.gov.ua/laws/main/5403-17#Text

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи Web-додатка для збереження спогадів з застосуванням фреймворка Spring.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 19-13 від 07.08.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи Web-додатка для збереження спогадів з застосуванням фреймворка Spring.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка апаратної та програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи Web-додатка для збереження спогадів з застосуванням фреймворка Spring;

– цілісність даних у процесі роботи, обробки та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

XML, Java.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2024 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий аналіз умов праці на робочому місці ІТ-фахівця.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 86 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 02.12.2024 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист __.__.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник бакалаврської дипломної роботи

_____ Кислун О.А.

Дослідження та програмна реалізація Web-додатка для збереження спогадів з застосуванням фреймворка Spring

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 20

Літера: РП

Кропивницький – 2024 року

Створення сутностей із використанням стандарту JPA

```
@Entity
@Table(name = "friend_relations")
public class FriendRelation {

    @Id
    @GeneratedValue
    @Column(name = "id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user1_id")
    private User user1;

    @ManyToOne
    @JoinColumn(name = "user2_id")
    private User user2;

    @ManyToOne
    @JoinColumn(name = "status_id")
    private StatusRelation status;

    @Temporal(value = TemporalType.TIMESTAMP)
    @Column(name = "date_action")
    private Date dateAction;

    public Date getDateAction() {
        return dateAction;
    }
    public void setDateAction(Date dateAction) {
        this.dateAction = dateAction;
    }
    public StatusRelation getStatus() {
        return status;
    }
    public void setStatus(StatusRelation status) {
        this.status = status;
    }
    public User getUser2() {
        return user2;
    }
    public void setUser2(User user2) {
        this.user2 = user2;
    }
    public User getUser1() {
        return user1;
    }
    public void setUser1(User user1) {
        this.user1 = user1;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
}
```

Реалізація репозиторія до даних таблиці *FRIEND_RELATIONS* із бази даних

```
@Service
public class FriendRelationsService
extends DataService<FriendRelation> {

    @Autowired
    private FriendRelationsRepository friendRelationsRepository;

    @Override
    public void persist(FriendRelation entity) {
        friendRelationsRepository.save(entity);
    }
    @Override
    public void update(FriendRelation entity) {
        friendRelationsRepository.save(entity);
    }
    @Override
    public FriendRelation findById(Long id) {
        return friendRelationsRepository.findOne(id);
    }
    @Override
    public void delete(Long id) {
        friendRelationsRepository.delete(id);
    }
    @Override
    public List<FriendRelation> findAll() {
        return Util.makeCollection(
            friendRelationsRepository.findAll());
    }

    @Override
    public void deleteAll() {
        friendRelationsRepository.deleteAll();
    }
    @Override
    protected AbstractDao<FriendRelation> getDao() {
        return null;
    }
}
```

Розробка контролерів для роботи через HTTP протокол.

```
@Controller
@RequestMapping("/memory")
public class MemoriesController extends AbstractController {

    @Autowired
    private MemoriesService memoriesService;

    @ResponseBody
    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public Memory getMemoryById(@PathVariable("id") Long id,
                                HttpServletResponse response)
                throws ServletException, IOException {
        Memory memory = memoriesService.findById(id);
        if (memory == null) {
            response.sendRedirect("/memory/error/"
                    + "?errorText=" + "Memory with id:" + id +
                        " doesn't exists.");
            return null;
        }
        return memory;
    }

    @ResponseBody
    @RequestMapping(value = "/user", method = RequestMethod.GET)
    public List<MemoryVO> getMemoriesByUser(
            @RequestParam(value = "id",required = false) Long id,
                  @RequestParam(value = "login",required = false)
                        String login,
            HttpServletResponse response)
                  throws ServletException, IOException {
        List<MemoryVO> list;
        if (id != null)
            list = memoriesService.findVOByUserId(id);
        else
            list = memoriesService.findVOByUserLogin(login);
        if (list.isEmpty()) {
            response.sendRedirect("/memory/error/"
                    + "?errorText=" +
                        "Memories not found, id: " + id);
            return null;
        }
        return list;
    }
}
```

Використання класу MemoryVO для повернення результатів

```
public class MemoryVO {

    private Long id;
    private String text;
    private Long moodId;
    private String moodDescription;
    private List<String> picturesUrls;

    public MemoryVO(Long id, String text,
                    Long moodId, String moodDescr,
                    List<String> urls) {
        this.id = id;
        this.text = text;
        this.moodId = moodId;
        this.moodDescription = moodDescr;
        this.picturesUrls = urls;
    }

    public Long getId() {
        return id;
    }

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

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public Long getMoodId() {
        return moodId;
    }

    public void setMoodId(Long moodId) {
        this.moodId = moodId;
    }

    public String getMoodDescription() {
        return moodDescription;
    }

    public void setMoodDescription(String moodDescription) {
        this.moodDescription = moodDescription;
    }

    public List<String> getPictureUrls() {
        return picturesUrls;
    }

    public void setPictureUrls(List<String> pictureUrls) {
        this.picturesUrls = pictureUrls;
    }

    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        builder.append("id: " + id + "\n");
        builder.append("text: " + text + "\n");
        builder.append("moodId: " + moodId + "\n");
        builder.append("moodDescr: " + moodDescription + "\n");
```

```
        builder.append("picturesUrls: ");
        for (String str : picturesUrls)
            builder.append(str);
        return  builder.toString();
    }
}
```

Інтегрування Spring Security у додаток

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    private DataSource dataSource;

    private final String userQuery =
            "SELECT login, password, TRUE "
                  + "FROM users "
                  + "WHERE login = ?";

    private final String authoritiesQuery =
            "select login, role_of_user " +
                  " from users where login = ?;";


    @Autowired
    public void configureGlobal(
      AuthenticationManagerBuilder auth)
            throws Exception {
        auth
                .jdbcAuthentication()
                .dataSource(dataSource)
                .usersByUsernameQuery(userQuery)
                .authoritiesByUsernameQuery(authoritiesQuery);
    }

    protected void configure(HttpSecurity http)
            throws Exception {
        http
                    .csrf().disable()
                    .authorizeRequests()
                    .antMatchers("/css/**",
                        "/fonts/**",
                       "/image/**",
                       "/js/**")
                                .permitAll()
                    .anyRequest().authenticated()
                .and()
                    .formLogin()
                    .loginProcessingUrl("/login").permitAll()
                    .usernameParameter("login")
                    .passwordParameter("password")
                          .successHandler(
      new MemoriesRestAuthenticationSuccessHandler())
                    .failureHandler(
new SimpleUrlAuthenticationFailureHandler())
                .and()
                    .logout()
                    .logoutUrl("/logout");
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/memory_row_selector"
    android:paddingRight="8dp"
    android:paddingLeft="8dp"
    android:paddingTop="3dp"
    android:paddingBottom="2dp">

    <ImageView
        android:id="@+id/userPhoto"
        android:layout_width="@dimen/userPhoto"
        android:layout_height="@dimen/userPhoto"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:paddingRight="@dimen/userPhotoPaddingRight"/>

    <TextView
        android:id="@+id/userName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/userPhoto"
        android:layout_marginTop="@dimen/userNameMarginTopBottom"
        android:layout_marginBottom=
            "@dimen/userNameMarginTopBottom"
        android:layout_toRightOf="@id/userPhoto"
        android:textSize="@dimen/userName"
        android:textColor="@android:color/black"/>

    <Button
        android:id="@+id/memoryMoodBtn"
        android:layout_width="@dimen/memoryMoodBtnWidth"
        android:layout_height="@dimen/memoryMoodBtnHeight"
        android:layout_below="@+id/userName"
        android:layout_toRightOf="@+id/userPhoto"/>

    <TextView
        android:id="@+id/memoryMoodDescr"
        android:layout_width="wrap_content"
        android:layout_height="@dimen/memoryMoodDescrHeight"
        android:layout_below="@+id/userName"
        android:layout_marginTop=
            "@dimen/memoryMoodDescrMarginTop"
        android:layout_toRightOf="@id/memoryMoodBtn"
        android:layout_marginLeft=
            "@dimen/memoryMoodDescrMarginLeft"
        android:gravity="center"
        android:textSize="@dimen/memoryMoodDescrTextSize"
        android:textColor="#212121"/>

    <Button
        android:id="@+id/memoryOptions"
        android:layout_width="@dimen/memoryOptionsBtnWidth"
        android:layout_height="@dimen/memoryOptionsBtnHeight"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:layout_marginTop=
            "@dimen/memoryOptionsBtnMarginTop"
        android:background=
            "@drawable/ic_menu_moreoverflow_normal_holo_light"/>

    <TextView
        android:id="@+id/memoryText"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/userPhoto"
        android:textSize="@dimen/memoryTextSize"
        android:textColor="@android:color/black"
        android:paddingLeft="@dimen/memoryTextPaddingRightLeft"
        android:paddingRight="@dimen/memoryTextPaddingRightLeft"
        android:layout_marginTop="@dimen/memoryTextMarginTop"/>

    <ImageView
        android:id="@+id/memoryPicture"
        android:layout_width="fill_parent"
        android:layout_height="@dimen/memoryPic"
        android:layout_below="@+id/memoryText"
        android:paddingTop="@dimen/memoryPicPaddingTop"/>

</RelativeLayout>
```

Код для відправки запиту та отримання результату від серверу

```
public static <S> S createService(Class<S> serviceClass,
      Context context) {
        OkHttpClient okHttpClient = new OkHttpClient();
        okHttpClient.interceptors().add(
            new ReceivedCookieInterceptor(context));

        RestAdapter.Builder builder = new RestAdapter.Builder()
                .setEndpoint(BASE_URL)
                .setClient(new OkClient(okHttpClient));

        RestAdapter adapter = builder.build();

        return adapter.create(serviceClass);
    }

public class ReceivedCookieInterceptor
                  implements Interceptor {

    private final Context context;

    public ReceivedCookieInterceptor(Context context) {
        this.context = context;
    }

    @Override
    public Response intercept(Chain chain) throws IOException {
        Response originalResponse =
                  chain.proceed(chain.request());

        if (!originalResponse.headers("Set-Cookie").isEmpty()) {
            SecurePreferences securePreferences =
                    new SecurePreferences();

            String[] cookies;
            String[] cookieParts;

            for (String header :
                 originalResponse.headers("Set-Cookie")) {
                cookies = header.split("; ");
                for (String cookie : cookies) {
                    cookieParts = cookie.split("=");
                    if (cookieParts[0].equals("JSESSIONID")) {
                        securePreferences.put("JSESSIONID",
                              cookieParts[1]);
                        break;
                    }
                }
            }
        }
        return originalResponse;
    }

}
```

Аутентифікація в системі через кукі

```
public class AddCookieInterceptor
            implements Interceptor {

    private final Context context;

    public AddCookieInterceptor(Context context) {
        this.context = context;
    }

    @Override
    public Response intercept(Chain chain) throws IOException {
        Request.Builder builder = chain.request().newBuilder();
        SecurePreferences securePreferences =
                  new SecurePreferences();
        String cookie = "JSESSIONID=" +
                  securePreferences.getString("JSESSIONID");
        builder.addHeader("Cookie", cookie);
        return chain.proceed(builder.build());
    }
}

public class LoginActivity extends Activity {

    private UserInfoStorage userInfoStorage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        userInfoStorage = new UserInfoStorage(this);
        if (userInfoStorage.hasUserInfo())
            authorize();

        setContentView(R.layout.activity_login);

        Button submit = (Button) findViewById(R.id.submit);

        submit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String login =
                          ((EditText)
            findViewById(R.id.login)).getText().toString();
                    String password =
                          ((EditText)
            findViewById(R.id.password)).getText().toString();
                    if (login.isEmpty() || password.isEmpty()) {
                        showAlertDialog("Hey",
                            Please fill fields login and password!");
                        return;
                    }
                    authorize(login, password);
            }
        });
    }

    private void authorize() {
        User user = userInfoStorage.loadUserInfo();
        new LoginTask(user.getLogin(), user.getPassword(),
      LoginActivity.this).execute();
    }

    private void authorize(String login, String password) {
        new LoginTask(login, password,
                  LoginActivity.this).execute();
```

```
    }
}

public class LoginTask
            extends AbstractAsyncTask<Void,Void,Void> {

    private final static String tag = "MEMORIES_LOGIN";

    private final String username;
    private final String password;

    public LoginTask(String username, String password,
                     Context context) {
        super(context);
        HandlerThread uiThread = new HandlerThread("UIHandler");
        uiThread.start();
        this.username = username;
        this.password = password;
    }

    @Override
    protected Void doInBackground(Void... params) {

        LoginService loginService =
                ServiceGenerator.createService(
                        LoginService.class, getContext());
        loginService.login(username, password,
                 new LoginRetrofitCallback());

        return null;
    }

    public class LoginRetrofitCallback
            implements retrofit.Callback<Response> {

        @Override public void success(Response response,
                      Response response2) {
            UpdateUserInfoTask task =
                 new UpdateUserInfoTask(getContext());
            try {
                task.execute(username).get();
            } catch (Exception e) {
                Log.d(tag, e.toString());
            }
            Intent intent = new Intent(getContext(),
                 MemoriesListActivity.class);
            getContext().startActivity(intent);
        }

        @Override public void failure(RetrofitError error) {
            Log.d(tag, "Fail: " + error.toString());
            Toast.makeText(getContext(), error.toString(),
                 Toast.LENGTH_LONG).show();
        }
    }
}
```

Активність списку спогадів

```
public class MemoriesListAdapter extends BaseAdapter {

    private static final String log = "MEMORY_LIST_LOAD";

    private Context context;
    private LayoutInflater inflater;
    private List<Memory> memoryList = new ArrayList<>();
    private Picasso picasso;
    private User user;

    public MemoriesListAdapter(Context context,
                 Picasso picasso) {
        this.context = context;
        this.inflater = (LayoutInflater)
                context.getSystemService(
                        Context.LAYOUT_INFLATER_SERVICE);
        this.picasso = picasso;
        this.user = new UserInfoStorage(context).loadUserInfo();
        this.memoryList = new ArrayList<>();
        loadMemories();
    }

    private void loadMemories() {
        LoadMemoriesService service =
                ServiceGenerator.createLoadDataService(
                        LoadMemoriesService.class, context);
        service.loadMemoriesByUser(user.getLogin(),
                        new LoadMemoriesCallback(memoryList));
    }

    @Override
    public int getCount() {
        return memoryList.size();
    }

    @Override
    public Object getItem(int position) {
        return memoryList.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView,
                        ViewGroup parent) {
        if (convertView == null)
            convertView = inflater.inflate(
                  R.layout.memory_item, null);

        ImageView userPhoto =
                (ImageView)
             convertView.findViewById(R.id.userPhoto);
        TextView userName =
                (TextView)
            convertView.findViewById(R.id.userName);
        Button moodBtn =
                (Button)
            convertView.findViewById(R.id.memoryMoodBtn);
        TextView moodDescr =
                (TextView)
            convertView.findViewById(R.id.memoryMoodDescr);
        TextView memoryText =
                (TextView)
            convertView.findViewById(R.id.memoryText);
```

```
            ImageView picture =
                    (ImageView)
                convertView.findViewById(R.id.memoryPicture);

            // Get memory
            Memory m = memoryList.get(position);
            // set userPhoto
            picasso.load(user.getPhotoUrl())
                    .resize(200, 200)
                    .transform(new CropCircleTransformation())
                    .into(userPhoto);

            userName.setText(user.getUserName());
            moodBtn.setBackgroundResource(
                DrawableGetter.getMoodDrawableById(m.getMoodId()));
            moodDescr.setText(m.getMoodDescription());

            memoryText.setText(m.getText());

            picasso.load(m.getPicturesUrls().length > 0 ?
                            m.getPicturesUrls()[0] : null)
                    .resize(400, 400)
                    .placeholder(DrawableGetter.
                                        getPicturesPlaceholder())
                    .into(picture);

            return convertView;
        }

        public User getUser() {
            return user;
        }

        private class LoadMemoriesCallback
                implements Callback<Memory[]> {

            private List<Memory> memoriesList;

            public LoadMemoriesCallback(List<Memory> list) {
                this.memoriesList = list;
            }

            @Override public void success(Memory[] memories,
                    Response response) {
                Log.d(MemoriesListAdapter.log,
                        "Success load memories");
                Collections.addAll(memoriesList, memories);
                MemoriesListAdapter.this.notifyDataSetChanged();
            }

            @Override public void failure(RetrofitError error) {
                Log.d(MemoriesListAdapter.log, error.toString());
                Intent intent =
                      new Intent(context, LoginActivity.class);
                context.startActivity(intent);
            }
        }
    }
```

Реалізація HTTP клієнту

```
public class ReceivedCookieInterceptor
                implements Interceptor {

    private final Context context;

    public ReceivedCookieInterceptor(Context context) {
        this.context = context;
    }

    @Override
    public Response intercept(Chain chain) throws IOException {
        Response originalResponse =
                chain.proceed(chain.request());

        if (!originalResponse.headers("Set-Cookie").isEmpty()) {
            SecurePreferences securePreferences =
                    new SecurePreferences();

            String[] cookies;
            String[] cookieParts;

            for (String header :
                  originalResponse.headers("Set-Cookie")) {
                cookies = header.split("; ");
                for (String cookie : cookies) {
                    cookieParts = cookie.split("=");
                    if (cookieParts[0].equals("JSESSIONID")) {
                        securePreferences.put("JSESSIONID",
                             cookieParts[1]);
                        break;
                    }
                }
            }
        }
        return originalResponse;
    }

}
public class AddCookieInterceptor
          implements Interceptor {

    private final Context context;

    public AddCookieInterceptor(Context context) {
        this.context = context;
    }

    @Override
    public Response intercept(Chain chain) throws IOException {
        Request.Builder builder = chain.request().newBuilder();
        SecurePreferences securePreferences =
                new SecurePreferences();
        String cookie = "JSESSIONID=" +
                securePreferences.getString("JSESSIONID");
        builder.addHeader("Cookie", cookie);
        return chain.proceed(builder.build());
    }
}
```

Активність для аутентифікації користувача у веб системі

```
public class LoginActivity extends Activity {

    private UserInfoStorage userInfoStorage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        userInfoStorage = new UserInfoStorage(this);
        if (userInfoStorage.hasUserInfo())
            authorize();

        setContentView(R.layout.activity_login);

        Button submit = (Button) findViewById(R.id.submit);

        submit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String login =
                        ((EditText)
            findViewById(R.id.login)).getText().toString();
                String password =
                        ((EditText)
            findViewById(R.id.password)).getText().toString();
                if (login.isEmpty() || password.isEmpty()) {
                    showAlertDialog("Hey",
                        Please fill fields login and password!");
                    return;
                }
                authorize(login, password);
            }
        });
    }

    private void authorize() {
        User user = userInfoStorage.loadUserInfo();
        new LoginTask(user.getLogin(), user.getPassword(),
      LoginActivity.this).execute();
    }

    private void authorize(String login, String password) {
        new LoginTask(login, password,
                  LoginActivity.this).execute();
    }
}


public class LoginTask
            extends AbstractAsyncTask<Void,Void,Void> {

    private final static String tag = "MEMORIES_LOGIN";

    private final String username;
    private final String password;

    public LoginTask(String username, String password,
                     Context context) {
        super(context);
        HandlerThread uiThread = new HandlerThread("UIHandler");
        uiThread.start();
        this.username = username;
        this.password = password;
    }

    @Override
    protected Void doInBackground(Void... params) {
```

```
        LoginService loginService =
                ServiceGenerator.createService(
                        LoginService.class, getContext());
        loginService.login(username, password,
                  new LoginRetrofitCallback());

        return null;
    }

    public class LoginRetrofitCallback
            implements retrofit.Callback<Response> {

        @Override public void success(Response response,
                        Response response2) {
            UpdateUserInfoTask task =
                  new UpdateUserInfoTask(getContext());
            try {
                task.execute(username).get();
            } catch (Exception e) {
                Log.d(tag, e.toString());
            }
            Intent intent = new Intent(getContext(),
                  MemoriesListActivity.class);
            getContext().startActivity(intent);
        }

        @Override public void failure(RetrofitError error) {
            Log.d(tag, "Fail: " + error.toString());
            Toast.makeText(getContext(), error.toString(),
                  Toast.LENGTH_LONG).show();
        }
    }
}
```

Адаптер списку спогадів

```
public class MemoriesListAdapter extends BaseAdapter {

    private static final String log = "MEMORY_LIST_LOAD";

    private Context context;
    private LayoutInflater inflater;
    private List<Memory> memoryList = new ArrayList<>();
    private Picasso picasso;
    private User user;

    public MemoriesListAdapter(Context context,
                 Picasso picasso) {
        this.context = context;
        this.inflater = (LayoutInflater)
                context.getSystemService(
                        Context.LAYOUT_INFLATER_SERVICE);
        this.picasso = picasso;
        this.user = new UserInfoStorage(context).loadUserInfo();
        this.memoryList = new ArrayList<>();
        loadMemories();
    }

    private void loadMemories() {
        LoadMemoriesService service =
                ServiceGenerator.createLoadDataService(
                        LoadMemoriesService.class, context);
        service.loadMemoriesByUser(user.getLogin(),
                        new LoadMemoriesCallback(memoryList));
    }

    @Override
    public int getCount() {
        return memoryList.size();
    }

    @Override
    public Object getItem(int position) {
        return memoryList.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView,
                        ViewGroup parent) {
        if (convertView == null)
            convertView = inflater.inflate(
                  R.layout.memory_item, null);

        ImageView userPhoto =
                (ImageView)
             convertView.findViewById(R.id.userPhoto);
        TextView userName =
                (TextView)
            convertView.findViewById(R.id.userName);
        Button moodBtn =
                (Button)
            convertView.findViewById(R.id.memoryMoodBtn);
        TextView moodDescr =
                (TextView)
            convertView.findViewById(R.id.memoryMoodDescr);
        TextView memoryText =
                (TextView)
            convertView.findViewById(R.id.memoryText);
```

```
            ImageView picture =
                    (ImageView)
                convertView.findViewById(R.id.memoryPicture);

            // Get memory
            Memory m = memoryList.get(position);
            // set userPhoto
            picasso.load(user.getPhotoUrl())
                    .resize(200, 200)
                    .transform(new CropCircleTransformation())
                    .into(userPhoto);

            userName.setText(user.getUserName());
            moodBtn.setBackgroundResource(
                DrawableGetter.getMoodDrawableById(m.getMoodId()));
            moodDescr.setText(m.getMoodDescription());

            memoryText.setText(m.getText());

            picasso.load(m.getPicturesUrls().length > 0 ?
                            m.getPicturesUrls()[0] : null)
                    .resize(400, 400)
                    .placeholder(DrawableGetter.
                                        getPicturesPlaceholder())
                    .into(picture);

            return convertView;
        }

        public User getUser() {
            return user;
        }

        private class LoadMemoriesCallback
                implements Callback<Memory[]> {

            private List<Memory> memoriesList;

            public LoadMemoriesCallback(List<Memory> list) {
                this.memoriesList = list;
            }

            @Override public void success(Memory[] memories,
                    Response response) {
                Log.d(MemoriesListAdapter.log,
                        "Success load memories");
                Collections.addAll(memoriesList, memories);
                MemoriesListAdapter.this.notifyDataSetChanged();
            }

            @Override public void failure(RetrofitError error) {
                Log.d(MemoriesListAdapter.log, error.toString());
                Intent intent =
                      new Intent(context, LoginActivity.class);
                context.startActivity(intent);
            }
        }
    }
```

Закінчення реалізації рівня клієнт

```
public class MemoriesListActivity extends ActionBarActivity {

    private ListView listView;
    private MemoriesListAdapter adapter;
    private Picasso picasso;
    private ListView navigList;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_memories_list);

        this.picasso = new Picasso.Builder(this)
                .memoryCache(new LruCache(this))
                .build();

        listView = (ListView) findViewById(R.id.listMemories);
        adapter = new MemoriesListAdapter(this,picasso);
        listView.setAdapter(adapter);
        setSideMenu();
    }

    private void setSideMenu() {
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        loadProfileSideMenu();
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setHomeButtonEnabled(true);
    }

    private void loadProfileSideMenu() {
        ImageView userPhotoSM = (ImageView)
            findViewById(R.id.userPhotoSideMenu);
        TextView userNameSM = (TextView)
            findViewById(R.id.userNameSideMenu);

        picasso.load(adapter.getUser().getPhotoUrl())
                .resize(100,100)
                .transform(new CropCircleTransformation())
                .into(userPhotoSM);
        userNameSM.setText(adapter.getUser().getUserName());
    }
}
```