Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

РОЗРОБКА WEB ЗАСТОСУВАННЯ ДЛЯ ПЛАНУВАЛЬНИКА ЗАДАЧ ТА МОНІТОРИНГА САЙТІВ

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник курсової роботи

Кандидат фізико-математичних наук,

Сініцина Р. Б.

_____

(підпис)

“____” _____ 2021 р.

Виконала студентка ІПЗ-3

Гінкул А. О.

“____” _____ 2021 р.

Київ-2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

_____

(прізвище та ініціали)

_____

(підпис)

“____” _____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Гінкул Анні Олександрівні

факультету інформатики 3 р.н. бакалаврської програми

ТЕМА: Розробка Web застосування для планувальника задач та моніторинга сайтів

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Розділ 1. Аналіз предметної області

Розділ 2. Теоретичні відомості

Розділ 3. Опис розробки програмного продукту

Висновки

Список використаної літератури

Дата видачі “____” _____ 2020 р. Керівник _____*(підпис)*

Завдання отримала _____*(підпис)*

Тема: Розробка Web застосування для планувальника задач та моніторинга сайтів

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання завдання на курсову роботу	Жовтень 2020	
2.	Пошук літератури за темою	Листопад 2020	
3.	Ознайомлення зі знайденою літературою	Грудень 2020	
4.	Проектування та розробка бази даних	Січень 2021	
5.	Розробка структури програми та її архітектури	Лютий 2021	
6.	Реалізація клієнтської частини застосунку	Березень 2021	
7.	Написання теоретичної частини курсової роботи	Квітень 2021	
8.	Написання практичної частини курсової роботи	Травень 2021	
9.	Створення презентації	Травень 2021	
10.	Захист роботи	Травень 2021	

Студенка Гінкул А. О.

Керівник Сініцина Р. Б.

“___” _____

Зміст

Використані скорочення

БД – база даних

ER-модель – Entity-Relationship model

JPA – Java Persistence API

ACID – Atomicity (Атомарність), Consistency (Узгодженість), Isolation (Ізольованість), Durability (Довговічність). Гарантує надійність роботи транзакцій в БД

POJO – Plain Old Data Object

ВСТУП

У сучасному світі нестримно зростає об’єм задач та цілей, котрі ставить перед собою людина. Вони можуть бути абсолютно різними, від доручень по роботі до заданого домашнього завдання у школі, від розвитку власного бізнесу до звичайного списку покупок. У будь-якому з випадків постає проблема – як запам’ятати всю цю інформацію? Правильна відповідь полягає у тому, що її варто не запам’ятовувати, а записувати, і найбільш зручний для цього варіант – це веб-застосунок, який у будь-який момент часу буде доступним користувачу для запису своїх планів.

Метою даної роботи є створення веб-застосунку, який надавав би користувачам можливість створювати свої власні проекти, задачі всередині них, позначати етапи виконання завдань та зберігати корисні посилання на інші сайти, які, на погляд користувача, здаються йому корисними при роботі над проектом, або які він хоче регулярно моніторити.

Об’єктом дослідження у роботі є створення веб-додатку для керування задачами у своїх проектах.

Предметом дослідження є інші існуючі веб-застосунки, що також дозволяють планувати виконання завдань.

Розроблений застосунок написаний на мові програмування Java з використанням фреймворку Spring (серверна частина) та Javascript (на стороні клієнта). Було досліджено та використано шаблонізатор для html-сторінок thymeleaf.

Робота складається зі вступу, трьох розділів, висновків та списку використаної літератури.

Перший розділ аналізує актуальність даної задачі, аналоги застосунку та на основі цього ставить завдання для кінцевого продукту.

У другому розділі розповідається про досліджені та застосовані технології для розробки веб-додатків, розкриваються можливості фреймворку Spring при побудові клієнт-серверних застосунків.

Третій розділ описує етапи розробки додатку, його складові частини. Зокрема приділено увагу проектування бази даних та архітектурі застосунку. Наведений приклад остаточної роботи веб-застосунку.

РОЗДІЛ 1: АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Обґрунтування актуальності задачі

Кожна людина хоча б раз за все своє життя стикалася з великими проектами або ж просто громіздкими за своїм об’ємом завданнями, виконання яких потребує багато часу та зусиль. В такі моменти неможливо та навіть просто шкідливо тримати усе це в голові, ризикуючи забути про щось важливе та потім постраждати через це. На сьогодні мало хто обмежується лише одним проектом: людина має по кілька проектів на роботі, власні проекти, що стосуються її особистих інтересів. Через це з кожним роком кількість засобів, що допомагають розв’язувати поставлені перед собою задачі стрімко збільшується. Це можуть бути як паперові планери, так і сучасні веб-застосування, які мають багато переваг перед вже застарівшим варіантом, такі як доступ до задач з будь-якого девайсу та зручний різноманітний функціонал.

Веб-версії планерів зокрема дозволяють:

– Створювати нескінченно велику кількість задач, оскільки не обмежуються обсягом паперової сторінки.
– Редагування задачі, змінення її назви чи кінцевого терміну здачі без затрати на це багатьох зусиль.
– Мати одночасний доступ до однієї і тієї ж задачі кільком співробітникам.

У сучасному світі такі системи значно покращують життя людей, а також підвищують продуктивність та зручність роботи над існуючими проектами та задачами.

1.2 Огляд існуючих аналогів веб-застосунку

Через актуальність даної задачі існує багато аналогів веб-додатків для планування та менеджменту задач і проектів. Одними з найвідоміших сайтів є

Todoist, Trello та Notion, а також веб-розширення для браузеру Session Buddy, яке фокусується не на створенні та постановці задач, а на фіксації обраних ресурсів у вигляді веб-посилань на сторінки для подальшого їх використання у проектах чи дослідженнях. Зробимо аналіз перерахованих вище застосунків, щоб чітко з'ясувати, які є переваги у даних сайтів, які недоліки та що все ж таки змушує користувача користуватися ними та залишатися з цими компаніями.

1.2.1 Todoist

Todoist – дуже проста для використання та ознайомлення з нею програма. Дозволяє створювати декілька «проектів» всередині, щоб відокремити їх, або щоб відокремити різні за категоріями задачі. Всередині проекту знаходиться звичайний список, з можливістю встановлювати декілька пріоритетів для вхідних задач, їх кінцеву дату виконання, тощо. Не останню роль у сервісі грає можливість відкриття спільного доступу для проектів – це дозволяє користуватися даним сайтом як засобом для менеджменту складних проектів у великих корпораціях, так і в якості простого списку покупок, для якого відкритий доступ для усієї родини. Серед недоліків найбільш критичним є відсутність функціоналу позначення, що виконання певної задачі було почате, що робить незручним написання до списку справ більш часозатратних задач, які неможливо виконати одразу.

1.2.2 Trello

Trello – більш професійно орієнтований застосунок. На відміну від попереднього прикладу, він дозволяє користувачам розбивати проекти на списки, в яких можна окремо розміщувати вхідні завдання, ті, що знаходяться в процесі виконання та вже виконані.

1.2.3 Notion

Notion – єдиний з наведених ресурсів, який можна використовувати для створення як нотаток, так і списків справ. Він дозволяє форматувати написаний текст у багатьох форматах, робити вставку зображень та таблиць, закріплювати

за певним шматочком тексту посилання на будь-яку сторінку. Notion підходить навіть для написання конспектів. Проте саме те, що є його сильною стороною, одночасно робить цей сайт і незручним. Кількість функціоналу настільки велика за обсягом, що значний відсоток людей просто боїться починати їм користуватися, оскільки розуміють, що для організації їм знадобиться витратити як мінімум день свого часу та ще один, що навчитися користуватися потрібними саме їм можливостями сайту. Це насправді потрібно далеко не всім, оскільки на сьогодні конкуренція серед продуктів будь-якого типу дуже велика і набагато простіше обрати відразу той сайт, що задовольняє потреби людини і не вимагає від неї створення чогось, для чого вже існує окрема програма.

1.2.4 Session Buddy

Session Buddy – зовсім не підходить для планування задач, проте його функціонал не можна замінити переважною більшістю сучасних програм-планувальників. Він дозволяє зберігати посилання на обрані ресурси для подальшого зручного їх відкриття. Дуже корисно при роботі над проектом, для якого необхідно зберегти посилання на навчальні курси, цікаві статті, або будь що інше та легко продовжити роботу, навіть якщо ви кілька тижнів не поверталися до конкретного проекту. Даний варіант є майже ідеальним, якщо використовувати його з будь-яким іншим зручним саме для вас менеджером завдань, проте недолік такого варіанту є очевидним – зручніше користуватися лише одним ресурсом, який покривав би усі потреби при виконанні конкретного проекту, а не тримати у браузері кілька вкладок лише тому, що додатку,який поєднував би їх в собі, просто не існує.

Підсумовуючи усе вище сказане, отримуємо результат: на даний момент часу не існує поширених планувальників задач, що покривали б як можливість керування проектами та задачами всередині них, так і додавання посилань, що можуть знадобитися в майбутньому. Єдиний з сайтів, що підходить під даний опис – Notion – є сильно ускладненим та перевантаженим функціоналом. До того ж, він більше є сильно покращеною версією онлайн текстового редактору,

аніж засобом для постановки задач та досягнення цілей. Саме тому було вирішено створили власну версію планувальника, яка в повній мірі відповідала би наведеним вище вимогам (створення задач і зберігання ресурсів одночасно).

1.3 Постановка завдання

Метою даної роботи є реалізація веб-застосунку для створення та керування задачами, який відповідав би наступним вимогам:

1. Зручний та інтуїтивно зрозумілий інтерфейс користувача для забезпечення низького порогу входження для нових клієнтів та їх утримання.
2. Наявність реєстрації на сайті для збереження інформації, що була додана користувачем.
3. Можливість створювати декілька проектів, в кожному з яких розподіляти задачі по трьом спискам – вхідні, в процесі виконання та вже завершені.
4. Створення, редагування та видалення нових задач.
5. Встановлення для кожної задачі пріоритету (від одного то трьох), зазначення кінцевого терміну її виконання (за бажанням), після якого вказана дата буде підсвічуватись червоним для привернення уваги до таких задач.
6. Можливість додавати, редагувати та видаляти до кожного з проектів окремо посилання на інші сторінки, для подальшої зручної операції з ресурсами.
7. Наявність спільного доступу користувачів до проектів для керування груповими завданнями.
8. Приємна, не перевантажена зайвими деталями, кольорова гама, яка дозволить фокусуватись на головному – вмісті сайтів.
9. Адаптивний інтерфейс.
10. Коректна взаємодія з сервісною частиною застосунку.

Вимогами до серверної частини є:

1. Правильні проектування та взаємодія з БД.
2. Коректна взаємодія з клієнтською частиною застосунку.
3. Безпечність застосунку, перевірка вхідних даних у запитах, обов’язкове шифрування паролів для безпечної авторизації та автентифікації.

РОЗДІЛ 2: ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 JavaScript

JavaScript – об’єктно орієнтована мова програмування. В той час, як HTML визначає зміст сторінки, а CSS – її представлення, JavaScript несе відповідальність за її поведінку. [1]

JavaScript не створює додаткове навантаження на сервер, працюючи на стороні клієнта, а також підтримується багатьма браузерами, що впливає на поширення цієї мови програмування. [2]

Однією з найбільших переваг JavaScript є можливість динамічно змінювати вміст сторінки, при цьому не оновлюючи її після надсилання до серверу ajax-запитів та отримання запитуваної інформації. Це пришвидшує роботу програми, дозволяє зменшити трафік і кількість звернень до сервера та бази даних для отримання тієї частини даних, якою користувач вже володіє. Також цей метод дозволяє створювати односторінкові веб-застосунки, які просто оновлюють свій зміст під час свої роботи. [3]

2.2 Фреймворк Spring

Spring – це вільно доступний фреймворк, створений Родом Джонсоном. Він має дуже багато можливостей, проте в його основу закладене одне – спростити розробку додатків мовою Java. [4]

Як ми бачимо на рисунку 2.2.1, фреймворк Spring складається з модулів, які можна віднести до шести різних категорій та підключати їх до застосунку за потреби.

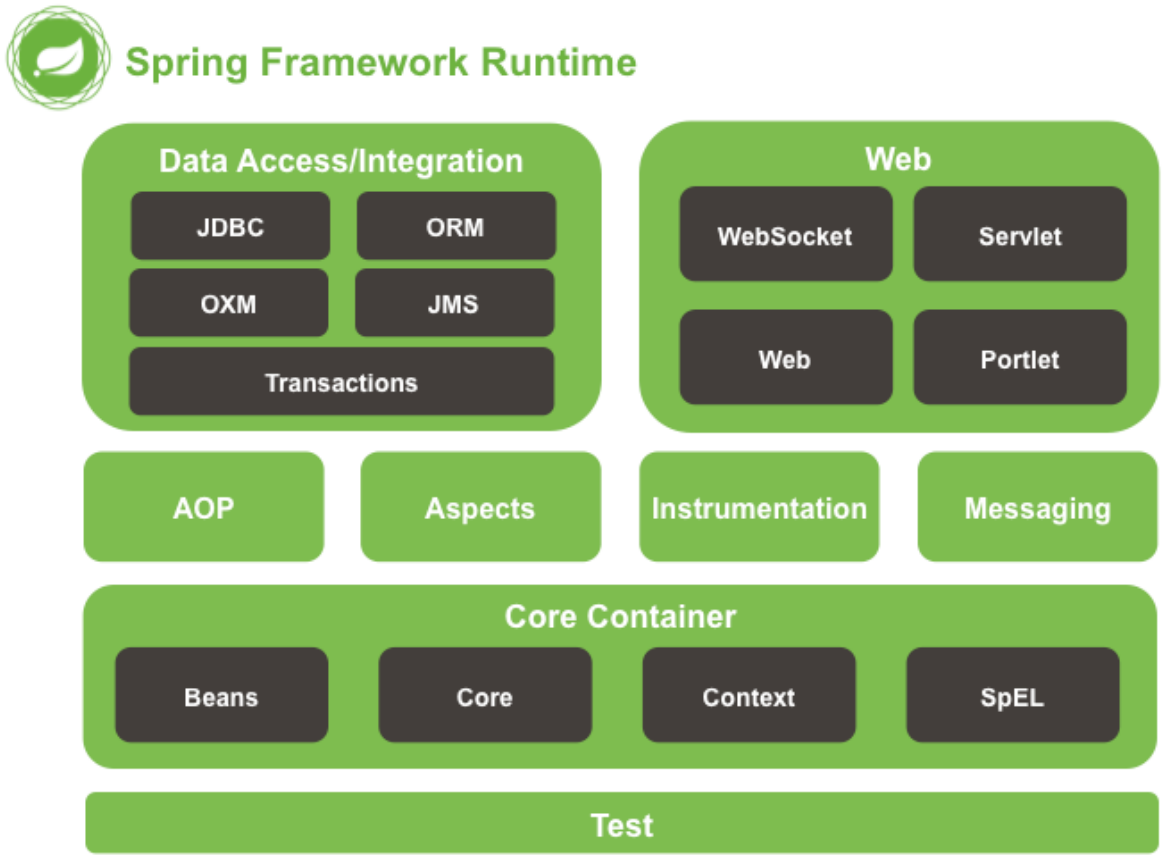

Рисунок 2.2.1 – модулі Spring [5]

2.2.1 Компоненти Spring

Spring Security

Spring Security – це основна технологія при реалізації безпеки у застосунках Spring. Він дозволяє гнучко реалізувати аутентифікацію та авторизацію, заборонити користувачу доступ до сторінок, що мають бути недоступними для нього. [6]

Методи, що дозволяють дізнатися інформацію про «роль» поточного користувача також відносяться до Spring Security.

Spring Boot

Мета Spring Boot – спростити процес створення коду, зменшити його кількість та перекласти велику кількість налаштувань у проекті на себе. Всі

його можливості розділені на пакети, що також називають стартерами. [7] Для того, щоб користуватися потрібними для роботи пакетами, необхідно додати їх до проекту в якості залежностей до файлу «application.properties». Spring Boot автоматично налаштує додаток.

Spring Data

Частина фреймворку Spring, що несе відповідальність за взаємодію з сутностями бази даних, їх отримання та взаємодію з репозиторіями.

2.2.2 Основні поняття Spring

Основою роботи фреймворку Spring є контекст ApplicationContext, він відповідає за створення та видалення бінів на основі конфігурації.

Клас, який є конфігурацією, анотується @Configuration. При створенні контексту, буде створено об’єкти, що знаходяться всередині класів-конфігурацій.

Методи всередині цього класу, що мають анотацію @Bean, будуть викликані після конфігурації, а біни (які по суті є просто об’єктами), які вони повернуть, буде додано до конфігурації. Пізніше ми отримаємо можливість отримувати необхідні нам біни, звертаючись до контексту. Класи об’єктів, які повертають методи з анотацією @Bean мають бути проанотовані @Component, така анотація може бути додана для будь-якого класу.

2.2.3 Реалізація та робота з базами даних в Spring

Для зберігання таблиці бази даних в програмі перш за все необхідно створити клас, який відповідав би за об’єкти конкретної реляційної моделі. Такі класи мають анотацію @Entity, кожен об’єкт якої вказує на кортеж у базі даних. Також такі класи обов’язково повинні мати анотацію @Table з атрибутом name, який вказує назву відповідної таблиці в базі даних.

Змінні таких класів, що відповідають атрибутам у базі даних, анотуються @Column, також з атрибутом name, що показує назву атрибуту в реляційній моделі. Первинні ключі потребують анотації @Id та @GeneratedValue, якщо подалі планується автоматично надавати об’єктам даної сутності ід, з атрибутом strategy. Він описує, як саме Spring буде генерувати первинні ключі для нових об’єктів класу. Стратегія GenerationType.IDENTITY відповідає звичайному AUTO_INCREMENT в MySQL.

Інколи необхідно назначити в якості первинних ключів одразу декілька атрибутів реляції. Spring не дозволяє додавати анотацію @Id одразу до кількох змінних класу, натомість він надає нам анотацію @EmbeddedId, яка дозволяє представити одразу кілька змінних в одному полі за допомогою додаткового класу. Змінні, які є первинними ключами, стають змінними класу, що наслідується від Serializable. Надалі з ним можна працювати, як зі звичайним об’єктом.

Також у Spring передбачена робота з валідацією даних. Існують окремі анотації для позначення мінімального та максимального допустимого значення числа, довжини стрічки, валідація з використанням регулярних виразів та інші.

Перевагою Spring є можливість автоматично отримувати з бази даних об’єкти сутностей, на які посилаються реляції за допомогою зовнішнього ключа. Такий підхід дозволяє економити час та ресурси за рахунок меншої кількості звернень до бази даних.

Якщо клас «А» має в якості зовнішнього ключа у реляційні моделі сутність, що описана класом «Б», до об’єкту класу «А» існує можливість додати поле типу «Б» з анотацією @JoinColumn і атрибутом name. Також вказується тип даного відношення, за допомогою анотацій @ManyToOne, @ManyToMany, @OneToMany чи @OneToOne, які означають кратність зв’язку. Такі об’єкти можуть одразу отримуватись з бази даних, або ж бути проініціалізованими лише за потреби, коли ми явно до них звернемось.

Наступний крок – створення класу, який наслідувався б від JpaRepository. JpaRepository – це інтерфейс фреймворку Spring Data, що надає набір стандартних методів JPA для роботи з базою даних. [8] При наслідуванні необхідно обов'язково вказати клас, для роботи з об'єктами якого ми створили даний репозиторій, а також клас типу його первинного ключа, використовуючи узагальнення (Generics). JpaRepository містить в собі вже написані найбільш необхідні методи для роботи з базою даних, якщо надати їм правильну назву, то Spring Data зможе перетворити їх у коректні запити, такі як отримання всіх об'єктів, пошук їх за первинним ключем (повертає об'єкт класу Optional) або за будь-яким іншим полем (повертає список) та багато інших. Також даний клас дозволяє створювати власні запити до бази даних, використовуючи анотацію @Query з атрибутом value. Стрічка всередині не містить чистого коду SQL-запиту, його реалізацію згенерує Spring Data. Нижче наведено приклад параметризованого запиту:

```
@Query(value="SELECT a " +
        "FROM MyObject a " +
        "WHERE a.id  = :id")
   List< MyObject > get MyObjectById(@Param("id") Integer id);
```

Також замість «:id» можна вказувати порядковий номер необхідного параметру в методі в форматі «?*[номер]*», де замість значення в квадратних дужках вказати число, починаючи нумерацію з одиниці (квадратні дужки залишати не потрібно).

Деякі SQL-запити змінюють дані, а не просто отримують їх, передаючи далі. Методи, що реалізовують подібні запити потребують додаткової анотації @Modifying.

Якщо необхідно, щоб база даних виконувала зміни лише в тому випадку, коли кожна операція в запиті виконалась успішно, тобто, щоб він виконувався в

транзакції та відповідав концепції ACID, до методу додається анотація-маркер @Transactional.

Для використання написаних методів репозиторію створеним проектом, гарним тоном вважається створення сервісів, оскільки отримання інформації з бази даних та передача її на користувацький інтерфейс напряму не прийнято. [8]

Для кожного з репозиторіїв варто створити свій власний сервер, в якому викликати необхідні методи для роботи з базою даних, а також реалізовувати іншу бізнес-логіку. Також клас-сервер анотується @Server. Методи цього класу можна викликати у контролерах для взаємодії між клієнтом та сервером.

2.2.4 Робота з запитами на сервер

Обробка запитів на сервер відбувається у класах-контролерах. Щоб показати Spring, які запити будуть оброблятися в проекті, над методами, які за це відповідають, додається анотація @RequestMapping, де вказується шлях і тип запиту. GET-методи, які мають перевести користувача на іншу сторінку застосунку, у відповідь повертають стрічку з назвою html файлу тієї сторінки, яку необхідно відкрити.

Для отримання параметрів метод має приймати змінну, анотовану @RequestParam з іменем необхідного параметру. Для отримання значення параметру у тому випадку, коли він передається в якості частини шляху, використовується @PathVariable, приклад коду з застосуванням якої наведено нижче:

```
@RequestMapping(“/my-site/{id}”, method = RequestMethod.GET)
Public String pathVariableExample(@PathVariable(“id”) int id, Model model){
    model.addAtribute(“idParameter”,id);
    return “html-page-name”;
```

}

Модель у Spring здатна зберігати всередині себе POJO-класи та пізніше звертатись до них під час рендерингу сторінки.

Spring Boot підтримує REST-підхід та передачу та отримання даних у форматі JSON.

Representational State Transfer (REST) – це набір правил, визначаючих стиль архітектури системи. [9]

Усього таких обов’язкових обмежень шість. Серед них:

1. Застосунок має відповідати моделі кліент-сервер.
2. Сервер не зберігає жодної інформації про клієнта, тобто є Stateless.
3. Хешування відповідей клієнта.
4. Можливість використання посередників між клієнтом та сервером, яке є невидимим для клієнта, тобто, він не може точно сказати, чи спілкується з сервером напряму.

Одною зі складових REST є методи запитів (GET, POST, PUT, DELETE). Для позначення використаного методу у Spring необхідно в анотації @RequestMapping окрім шляху запиту також зазначити «method = RequestMethod.GET», як це було показано вище, де «GET» можна замінити на будь-який інший метод.

Для позначення того, що методи контролера повертають дані у JSON форматі, до кожного з них треба додати анотацію @ResponseBody.

Дані, надіслані в тілі запиту, можна передавати в метод, створивши для об’єкту, який передається, окремий клас та, передаючи його в якості параметру методу, додати перед ним анотацію @RequestBody.

Для зручності та зменшення кількості коду всі анотації @ResponseBody можна опускати, якщо замінити @Controller перед класом на @RestController.

Дана анотація позначає саме те, що методи цього класу повертають JSON та що він є контролером.

2.3 Шаблонізатор Thymeleaf

Thymeleaf – шаблонізатор для web-середовища, що використовує в якості шаблона зокрема HTML5. Для використання шаблонізатора на html-сторінці необхідно підключити до неї простір імен thymeleaf, додавши до сторінки код:

<html xmlns:th="http://www.thymeleaf.org"> [10]

Використовуючи цей шаблонізатор, можна підставляти в елементи html-сторінки необхідні, невідомі наперед, значення, після обробки сторінки за допомогою даного шаблонізатора текст сторінки буде відображено без помилок та користувач зможе побачити сторінку такою, якою вона задумувалася. Для цього необхідно до елемента, який потрібно змінити, додати атрибут th:text="…", де на місті трьох крапок написати необхідний текст.

Встановивши на рівні контролера змінну контексту з ім’ям "variable", можна отримати до неї доступ зі сторінки таким чином: th:text="${variable}".

Thymeleaf підтримує цикли, що дозволяє легко відмальовувати на сторінці декілька елементів з масиву значень (наприклад, товари у магазині). За допомогою атрибута th:each можна створити локальну змінну всередині цього фрагмента і оперувати з нею далі так, як зі звичайною змінною контексту.

th:each="variable : ${variables}".

Атрибут th:if та протилежний йому th:unless дозволяють відображати (або навпаки, ні) необхідні елементи на html сторінці, таким чином, маючи можливість кардинально змінити її зміст.

Ще один потужний інструмент – створення фрагментів, які можуть бути додані до основної сторінки неодноразово, такі як заголовок та нижній колонтитул веб-сайтів, що зменшує кількість коду та робить розробку

інтерфейсу легшою, якщо пізніше знадобиться змінити у використаному в кількох місцях фрагменті код. [11]

2.4 Використані бібліотеки

В даному веб-застосунку було використано такі бібліотеки, як Bootstrap та jQuery, про які буде надано основну інформацію в наступних двох підрозділах.

2.4.1 Bootstrap

Bootstrap – це фреймворк для розробки користувацького інтерфейсу, який дозволяє пришвидшити його розробку, скориставшись вже існуючими компонентами для верстки html-сторінок.

2.4.2 jQuery

JQuery – це бібліотека, яка працює саме у поєднанні з JavaScript та покращує взаємодію між ним та html. За допомогою jQuery можна легко отримувати доступ до необхідних елементів на сторінці, додавати до них класи чи стилі, надсилати на сервер ajax-запити для спілкування з ним та багато іншого.

РОЗДІЛ 3: ОПИС РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

3.1 Реалізація бази даних

Для реалізації правильної бази даних необхідно виконати два кроки – розробити ER-модель та спроектувати на її основі реляційну модель бази даних, що і буде розглядатися в наступних двох розділах.

3.1.1 ER-модель

Розробка будь-якої бази даних починається з ER-моделі, що описувала б сутності та їх взаємодію між собою.

В даній роботі наявні п’ять сутностей:

1. Користувач – яка зберігає в собі всі його дані, логін та паролі (у зашифрованому вигляді).
2. Проект – сутність, яка відповідає за проекти, що створюють користувачі, та зберігає назву проекту.
3. Список – зберігає назви списків, в яких знаходяться завдання всередині проектів. Для всіх проектів по системі є однаковими та не передбачають постійних оновлень всередині застосунку. Потрібні, щоб мати доступ до статичної інформації про їх назви.
4. Завдання – відповідає за завдання, які користувачі додають до проектів. Серед їх атрибутів наявні ім’я, термін здачі завдання, його пріоритет (число від одного до трьох) та позиція. Позиція (ніяк не пов’язана з пріоритетом) показує те, на якому місці у списку, до якого належить дане завдання, воно знаходиться. Це передбачає для користувача можливість сортувати завдання в будь-якому порядку, незалежно від їх пріоритетів. Це можна використовувати, щоб розташувати нові вхідні завдання вище старих, або спланувати таким чином, в якому порядку користувач збирається починати виконувати

ці завдання, оскільки на це не завжди впливають пріоритети чи дедлайни, а, наприклад, складність завдань.

5. Ресурс – зберігає посилання на будь-який онлайн ресурс та ім’я цього ресурсу.

Наведена на рисунку 3.1.1.1 схема ER-моделі ілюструє роботу цих сутностей:

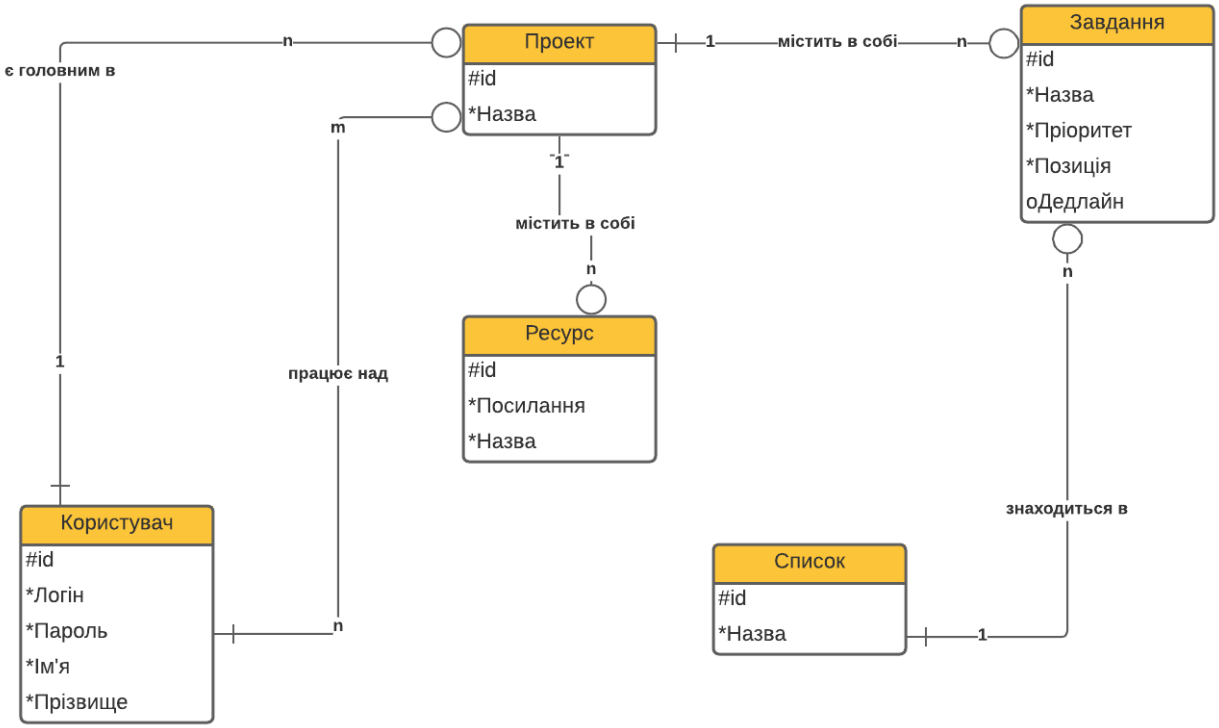

Рисунок 3.1.1.1 – ER-модель БД

Взаємодія між сутностями «Користувач» і «Проект» - «працює над» - показує, які користувачі мають доступ до яких проектів в цілому, оскільки застосунок передбачає роботу над проектами в групах, даний зв’язок має арність н до м.

Зв’язок «є головним в» відповідає за зберігання інформації про те, який користувач створив проект, а не просто має до нього доступ. Під час подальшої розробки цієї роботи завжди залишиться можливість надати користувачу, який створив проект, більше прав та можливостей при роботі над проектом чи

завданнями, або не видавати якісь з вже існуючих прав іншим користувачам, що не були причасні до створення проекту.

3.1.2 Реляційна модель

Другий етап розробки бази даних – проектування реляційної моделі даних, яка має відповідати побудованій попередньо ER-моделі. В даному випадку матимемо шість реляцій (на одну більше, аніж було сутностей, оскільки вона необхідна для зберігання зв'язку типу «багато до багатьох» між користувачами та проектами).

У наведених нижче таблицях показані всі реляції, їх ключі, типи а також стратегії «ON UPDATE» і «ON DELETE».

Реляційна модель user – Таблиця 3.1.2.1					
Ключ	Ім'я атрибуту	Тип атрибуту	NULL/NOT NULL	FK: ON DELETE	FK: ON UPDATE
PK	id	int	NOT NULL		
	login	varchar(30)	NOT NULL		
	password	varchar(80)	NOT NULL		
	name	varchar(20)	NOT NULL		
	surname	varchar(20)	NOT NULL		

Логін є унікальним значенням, проте його було вирішено не робити первинним ключем, оскільки б це збільшило об'єм інформації через велику кількість зовнішніх ключів. Також, це трохи заповільнило б отримання інформації про користувачів, що працюють над конкретним проектом.

Реляційна модель project – Таблиця 3.1.2.2					
Ключ	Ім'я атрибуту	Тип атрибуту	NULL/NOT NULL	FK: ON DELETE	FK: ON UPDATE
PK	id	int	NOT NULL		
FK	user_id	int	NOT NULL	CASCADE	CASCADE
	name	varchar(30)	NOT NULL		

Пояснення: при видаленні користувача видаляються всі його проекти, навіть групові (які створив власне він, не всі), оскільки вважається, що він був головним у цьому проекті.

Реляційна модель user_to_project – Таблиця 3.1.2.3					
Ключ	Ім’я атрибуту	Тип атрибуту	NULL/NOT NULL	FK: ON DELETE	FK: ON UPDATE
FK	user_id	int	NOT NULL	CASCADE	CASCADE
FK	project_id	int	NOT NULL	CASCADE	CASCADE

Пояснення: при видаленні проекту користувач більше не має до нього доступу, при видаленні користувача він більше не знаходиться в проекті.

Реляційна модель list – Таблиця 3.1.2.4					
Ключ	Ім’я атрибуту	Тип атрибуту	NULL/NOT NULL	FK: ON DELETE	FK: ON UPDATE
PK	id	int	NOT NULL		
	name	varchar(30)	NOT NULL		

Реляційна модель resource – Таблиця 3.1.2.5					
Ключ	Ім’я атрибуту	Тип атрибуту	NULL/NOT NULL	FK: ON DELETE	FK: ON UPDATE
PK	id	int	NOT NULL		
FK	project_id	int	NOT NULL	CASCADE	CASCADE
	name	varchar(30)	NOT NULL		
	url	varchar(250)	NOT NULL		

Пояснення: ресурси стають непотрібними, якщо видалили проект, в якому вони знаходились.

Реляційна модель task – Таблиця 3.1.2.6					
Ключ	Ім’я атрибуту	Тип атрибуту	NULL/NOT NULL	FK: ON DELETE	FK: ON UPDATE
PK	id	int	NOT NULL		

Продовження таблиці 3.1.2.6					
FK	project_id	int	NOT NULL	CASCADE	CASCADE
FK	list_id	int	NOT NULL	RESTRICT	CASCADE
	name	varchar(30)	NOT NULL		
	priority	int	NOT NULL		
	position	int	NOT NULL		
	deadline	datetime	NULL		

Пояснення: завдання також стають непотрібними, якщо видалили проект, в якому вони знаходились.

Не можна видаляти списки, в яких є якісь завдання, оскільки списки у даному застосунку є однаковими для всіх користувачів та не передбачають регулярних змін чи видалення. Вони зберігають статичну інформацію про імена списків у системі.

Пріоритет задачі має значення від одного до трьох.

Позиціє має бути більше або рівною нулеві, а також не має повторюватись в межах одного списку на одному проекті.

Обмежень щодо дати немає, веб-планувальник передбачає додавання завдань, термін виконання яких вже минув.

3.2 Опис серверної частини застосунку

Даний застосунок використовує трьохрівневу архітектуру, що складається з рівня представлення, прикладного рівня та рівня даних. У такої архітектури багато переваг, оскільки вона легко маштабується та, за потреби, дозволяє легко замінити одні компоненти на інші. Схематично її можна зобразити таким чином – Рисунок 3.2.1.

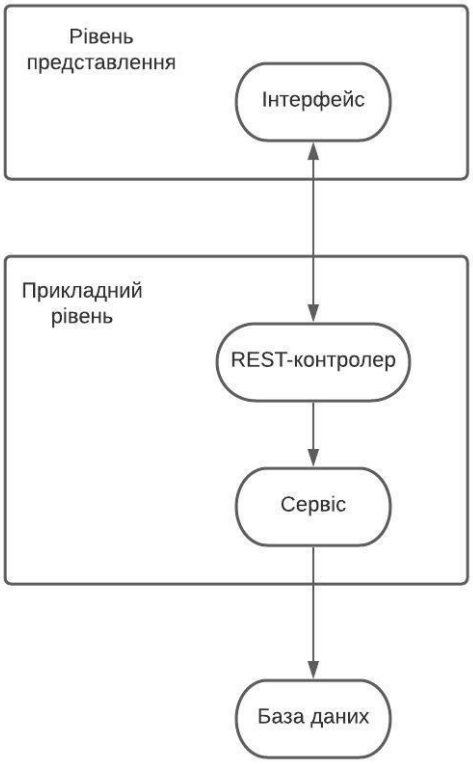

Рисунок 3.2.1 – схема архітектури застосунку

Серед запитів, які доступні клієнту для спілкування з сервером, є такі:

– POST-запити (приймають від клієнта JSON з полями класу реалізації вказаної в назві запиту реляції):
  - add_project
  - add_resourse
  - invite_user – на відміну від інших, приймає id користувача та проекту, створюючи між ними зв’язок за допомогою таблиці user_to_project
  - signup – створює нового користувача
  - add_task
– PUT-запити (також приймають JSON):
  - edit_resourse
  - edit_user
  - edit_task

    - update_tasks_order – змінює порядок задач у списках. Всередині JSON необхідно вказати наступні дані: id проекту, завдання, початкового списку, кінцевого списку (може співпадати з початковим), початкову позицію та кінцеву позицію завдання. Про нього буде більш детально розказано в наступному підрозділі.
- DELETE-запити (приймає від клієнта id об’єкту):
    - delete_project
    - delete_resourse
    - delete_task

В усіх записах, де це є потрібним, наявна перевірка, що авторизований користувач (якщо такий є) має доступ до вказаного проекту або його завдання чи ресурсу. Інакше жодної інформації в базі даних не буде додано/змінено/видалено, а користувачу буде повернене повідомлення про помилку доступу.

Для перевірки правильності роботи цих запитів, було використано додаток Postman – він дозволяє створювати та відсилати запити на сервер, що дуже полегшує тестування, якщо необхідно перевірити роботу, наприклад, POST-запитів, при яких клієнт має відправити на серверну частину застосунку JSON.

3.3 Реалізація збереження зміни позиції задачі в базі даних

Даний SQL-запит дуже рідко вимагає своєї реалізації в програмах через вузьку специфіку та тому, на відміну від додавання, редагування чи видалення, автоматична генерація запиту для нього фреймворком Spring не передбачена.

Дану функцію було розділено на два різних запити – перший при зміні позиції в своєму списку і другий при зміні і списку, і позиції.

Якщо ми змінюємо розташування задачі в одному списку, перетягуючи її з позиції з індексом n на позицію m (на даному етапі припустимо, що n>m), то змінення порядку задач у списках буде виглядати так, як показано на рисунку 3.3.1 (змінюються задачі з позиціями лише від n до m).

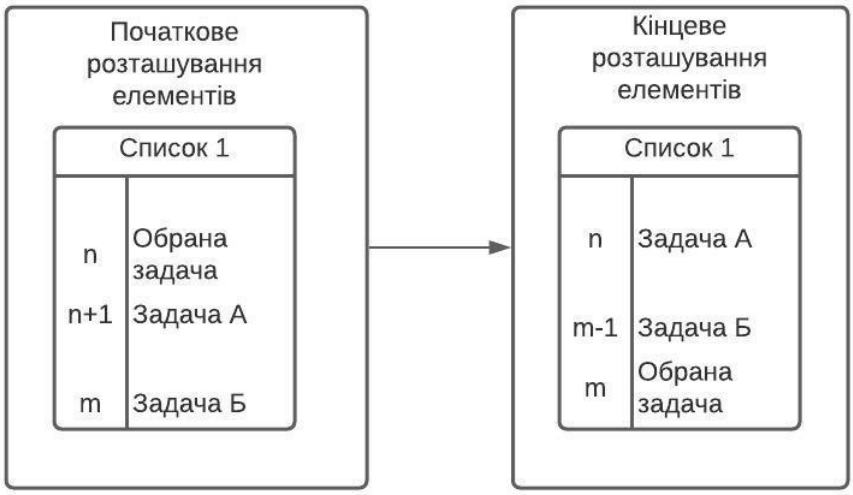

Рисунок 3.3.1 – схема зміни порядку задач в одному списку

Якщо m>n, схематичне зображення задач у списках виглядатиме схожим чином, необхідно лише поміняти місцями список у початковому розташуванні елементів зі списком у кінцевому розташуванні елементів.

Реалізація даного методу у Spring наведена на рисунку 3.3.2.

```
@Modifying
@Transactional
@Query(value = "UPDATE Task a "+
        "SET    a.position = CASE WHEN (a.position = ?2) THEN "+
        "?3 "+
        "WHEN (?3 < ?2) AND (a.position < ?2) THEN "+
        "(a.position + 1) "+
        " WHEN (?3 > ?2) AND (a.position > ?2) THEN "+
        "(a.position - 1) "+
        "ELSE "+
        "a.position "+
        "END "+
        "WHERE a.project = ?1 AND a.list = ?4")
void updateTasksOrder(Integer project_id, Integer start_pos, Integer finish_pos, Integer start_list);
```

Рисунок 3.3.2 – реалізація зміни порядку задач в одному списку.

Другий випадок – зміна списку. При цьому немає значення, яка з позицій (початкова чи кінцева) більша іншої, перетворення порядку задач відбудеться єдиним чином. В обох списках змінюються позиції задач, чиї номери більші або рівні за n (у першому списку) чи m (у другому). Схему зміни позицій наведено на рисунку 3.3.3.

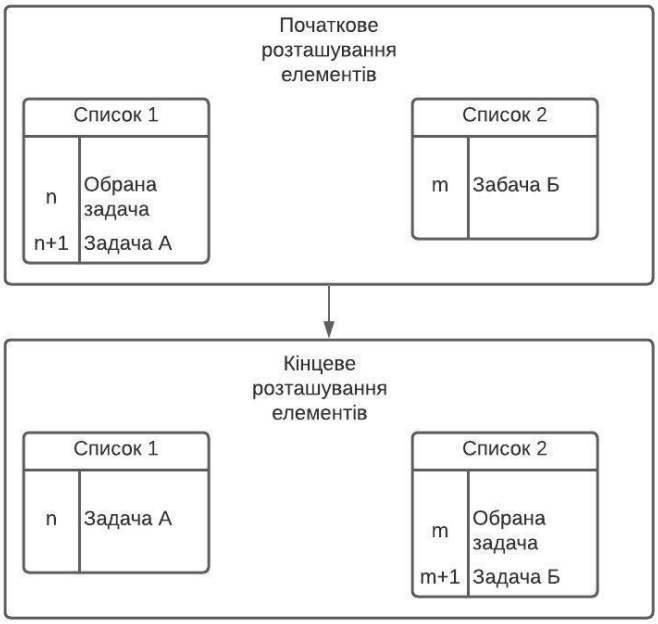

Рисунок 3.3.3 – схема зміни порядку задач при зміні списку

Реалізація даного методу у Spring наведена на рисунку 3.3.4.

```
@Modifying
@Transactional
@Query(value = "UPDATE Task a "+
        "SET    a.position = CASE WHEN (a.list = ?4 AND a.position = ?2 AND a.id=?6) THEN "+
        "?3 "+
        "WHEN (a.position > ?2) AND (a.list = ?4) THEN "+
        "(a.position - 1) "+
        "WHEN (a.position >= ?3) AND (a.list = ?5) THEN "+
        "(a.position + 1) "+
        "ELSE "+
        "a.position "+
        "END ,"+
        " a.list = CASE WHEN (a.id=?6) THEN " +
        "?5 " +
        "ELSE "+
        "a.list "+
        "END "+
        "WHERE a.project = ?1 AND (a.list = ?4 OR a.list = ?5)")
void updateTasksOrder(Integer project_id, Integer start_pos, Integer finish_pos,
                      Integer start_list, Integer finish_list, Integer task_id);
```

Рисунок 3.3.4 – реалізація зміни порядку задач при зміні списку

3.4 Опис клієнтської частини застосунку

Для реалізації клієнтської частини застосунку були використані, описані в другому розділі, шаблонізатор thymeleaf та мова програмування JavaScript. Клієнтська частина знаходиться в теці «resources.templates». Для кожної з основних сторінок, а саме – головна сторінка, сторінка реєстрації та входу на сайт, сторінка проекту – існує окремий файл з розширенням html, в той час як весь основний функціонал розміщено на сторінці з проектом. Додавання, редагування і видалення ресурсів та завдань виконується за допомогою ajax-запитів до серверу, дозволяючи динамічно змінити відображання інформації на сторінці.

Деякі частини контенту сторінок знаходяться в теці «fragments», в якій зберігаються всі фрагменти, присутні в даному застосунку. Серед них зокрема знаходяться фрагменти, що необхідні бути присутніми на будь-якій сторінці застосунку, тобто, заголовок сторінки та бокова панель меню для керування панелями (вона ж, завдяки thymeleaf, не відображається зовсім, якщо користувач не зайшов на сайт під своїм логіном. Таким чином він не бачить елементів сайту, які йому є непотрібними до того моменту, поки він не виконає операцію логіну), або ті, що відповідають за конкретні компоненти сторінок, які відображаються thymeleaf в циклі під час обробки сторінки – задачі та ресурси.

3.5 Тестування роботи програми

Після запуску програми сайт стає доступним за посиланням «http://localhost:8080/».

Головна сторінка сайту – єдина, крім форми входу, що доступна незареєстрованим користувачам – містить на собі опис сайту та коротку рекламну інформацію про даний застосунок для планування задач. Вигляд сторінки наведено на рисунку 3.5.1.

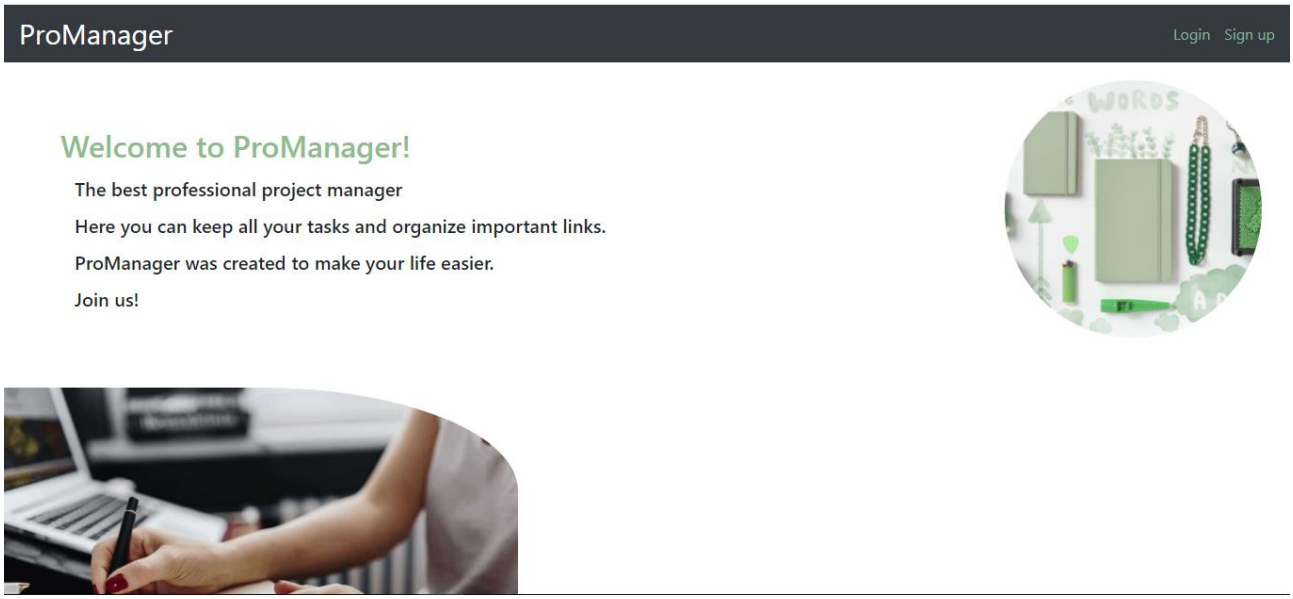

Рисунок 3.5.1 – вигляд головної сторінки веб-застосунку

На заголовку сторінки відображається назва застосунку та посилання на сторінку логіну та реєстрації (після того, як користувач зайде на сайт, дані посилання заміняться на посилання для виходу з аккаунту користувача, а зліва – кнопка додаткового меню). Див. рисунок 3.5.2.

Рисунок 3.5.2 – заголовок сторінки після авторизації користувача

Скориставшись сторінкою входу на сайт, користувачу стає доступним один з головних елементів його керування – бокова панель з переліком усіх проектів (рисунок 3.5.3), або написом, що такі відсутні (рисунок 3.5.4). За потреби всередині панелі також з’являється смуга прокрутки.

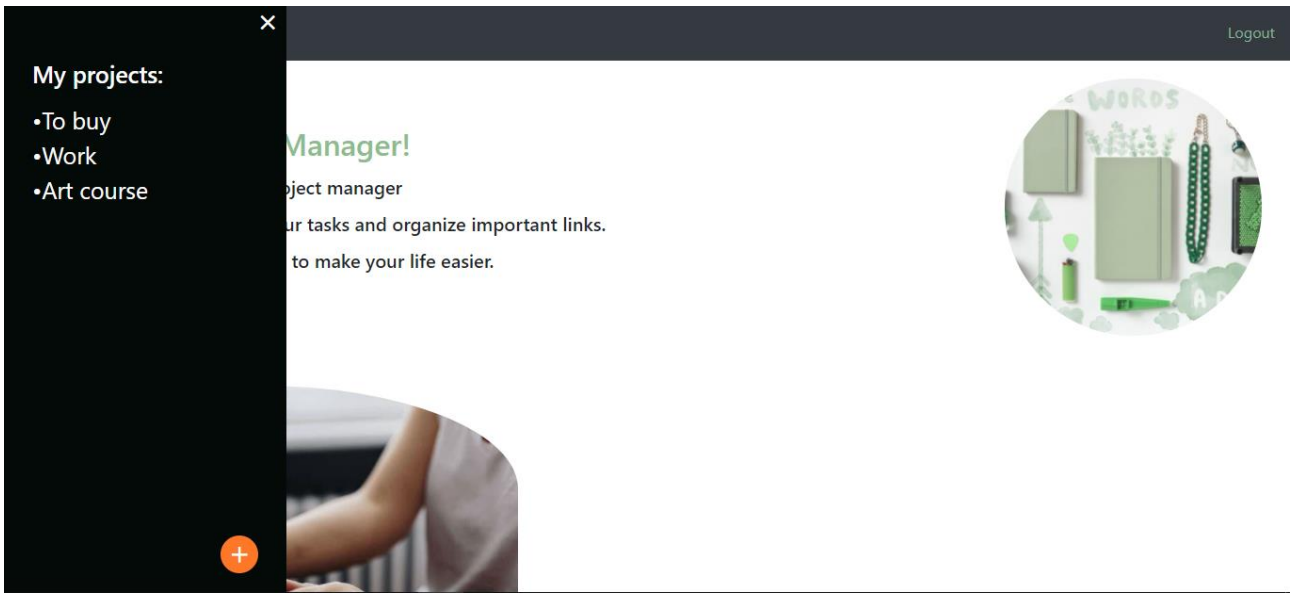

Рисунок 3.5.3 – бокове меню зі списком проектів

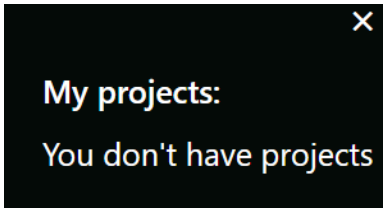

Рисунок 3.5.4 – бокове меню при відсутніх проектах

В правому нижньому кутку знаходиться контрастна до всіх інших елементів на сторінках кнопка для створення нових проектів, яка одразу привертає до себе увагу. Після натискання на неї відкривається модальне вікно, в якому користувач має ввести назву для проекту (наявна перевірка коректності введеної ним інформації). Вікно проілюстроване на рисунку 3.5.5.

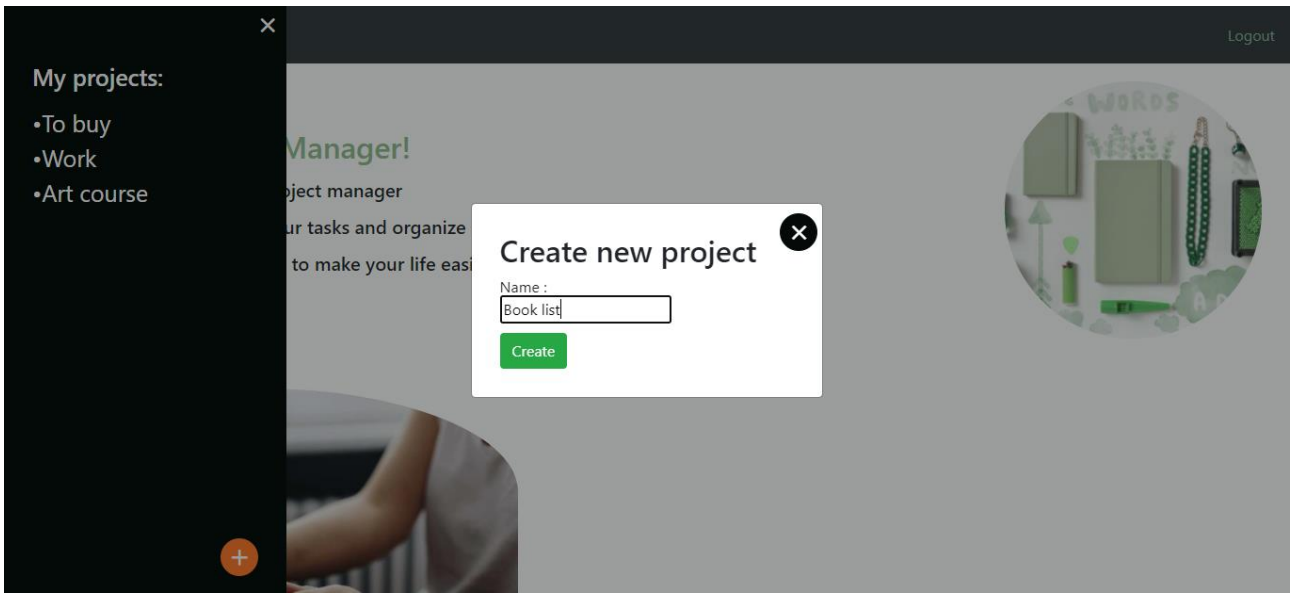

Рисунок 3.5.5 – створення нового проекту

На головній сторінці проекту зверху можемо побачити назву проекту та три кнопки праворуч. Можна видалити проект, якщо ствердно відповісти на запитання у спливаючому модальному вікні. При наведенні миші на них, кнопки додатково підсвічуються заданим для них кольором, як показано на рисунку 3.5.6.

Рисунок 3.5.6 – частина сторінки з проектом

Інші дві кнопки відповідають за керування груповим проектом. Будь-який проект можна зробити груповим, додавши до нього користувача за логіном, якщо такий існує в системі – інакше користувачу виведеться на екран повідомлення про некоректну введену інформацію (див. рисунок 3.5.7).

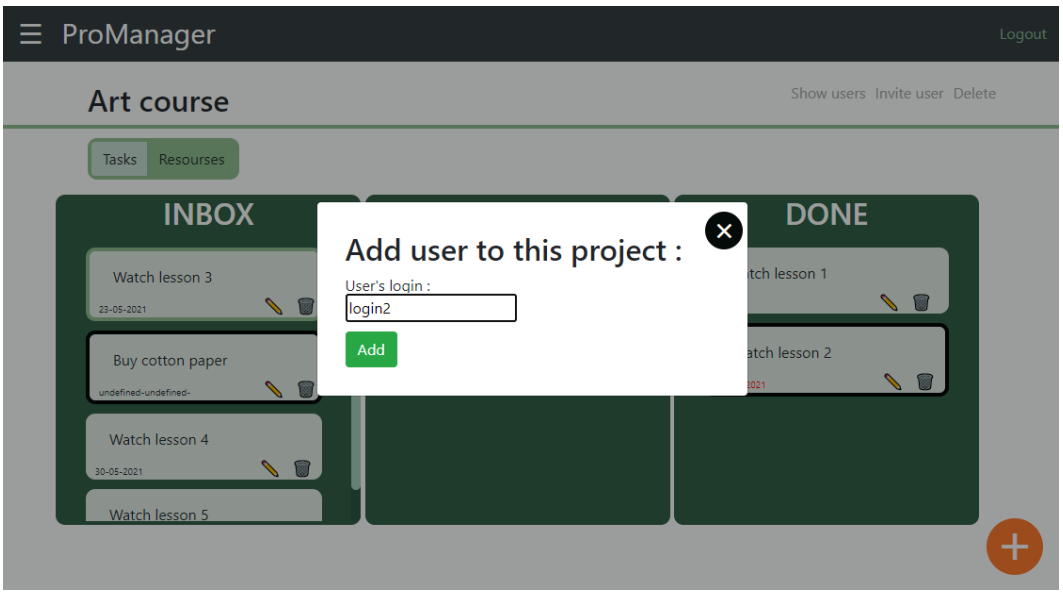

Рисунок 3.5.7 – запрошення користувача за логіном до свого проекту

За потреби можна переглянути, яким користувачам відкритий повний доступ до даного проекту.

На основній частині сторінки в першу чергу відображаються три списки з завданнями. Ті з них, що мають більший пріоритет, виділені товстішою рамкою. Повний вигляд сторінки показано на рисунку 3.5.8.

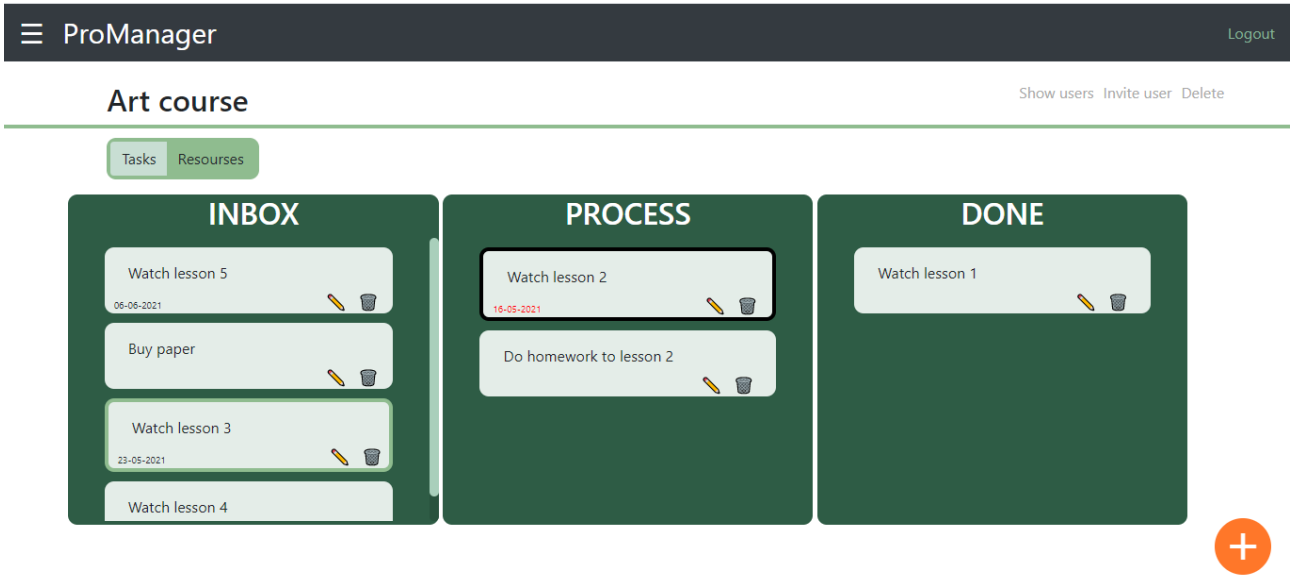

Рисунок 3.5.8 – сторінка з завданнями проекту

Як і на боковій панелі, справа знизу наявна така сама за виглядом кнопка додавання задачі, яка відкриває користувачу доступ до вікна створення задач (див. рисунок 3.5.9).

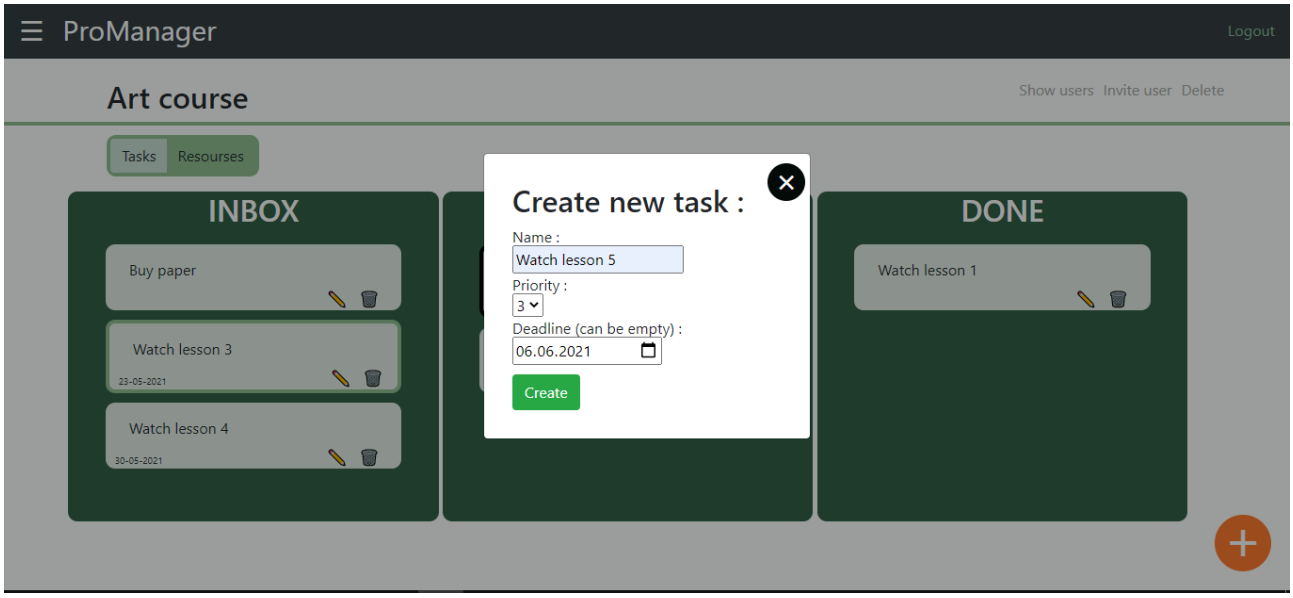

Рисунок 3.5.9 – створення нової задачі

Маленькі кнопки редагування та видалення біля кожної задачі відкривають відповідні до свого функціоналу вікон, що показані на рисунках 3.5.10 та 3.5.11, в яких користувач може або підтвердити те, що він планує зробити, або просто закрити вікно.

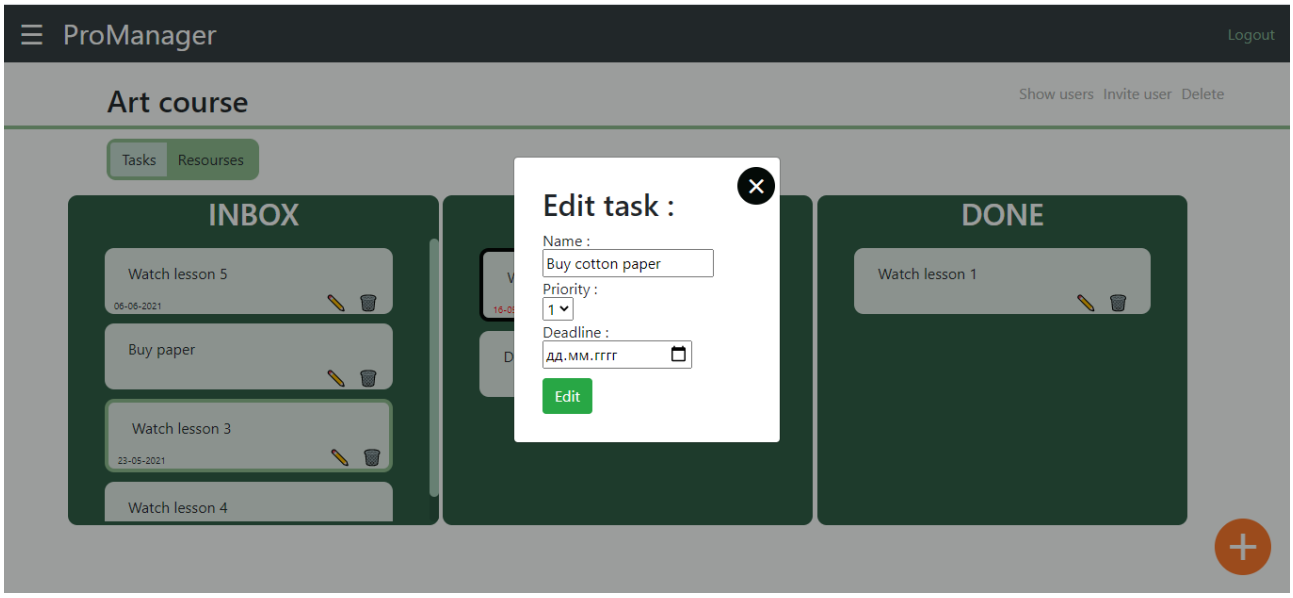

Рисунок 3.5.10 – редагування існуючої задачі

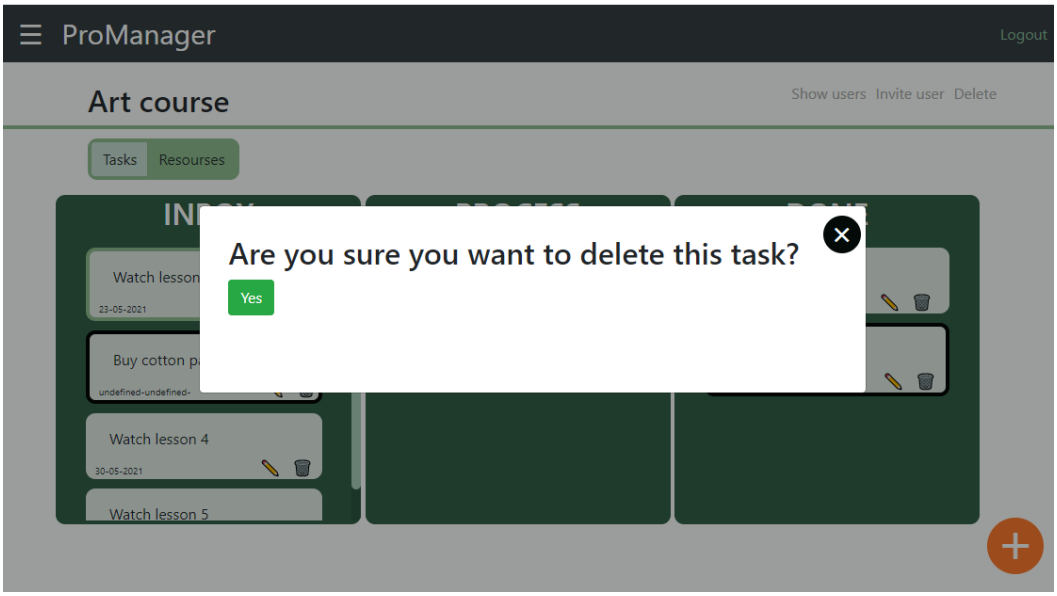

Рисунок 3.5.11 – підтвердження видалення задачі

Інтуїтивно зрозуміле затискання кнопки миші на задачі та перетягування її в інший список, або в інше місце в поточному списку (англійською – drag and drop). Користувач має можливість одразу побачити, в яке місце в списку попаде задача, якщо він її відпустить. Рисунку 3.5.12 та 3.5.13 ілюструють виконання даної функції.

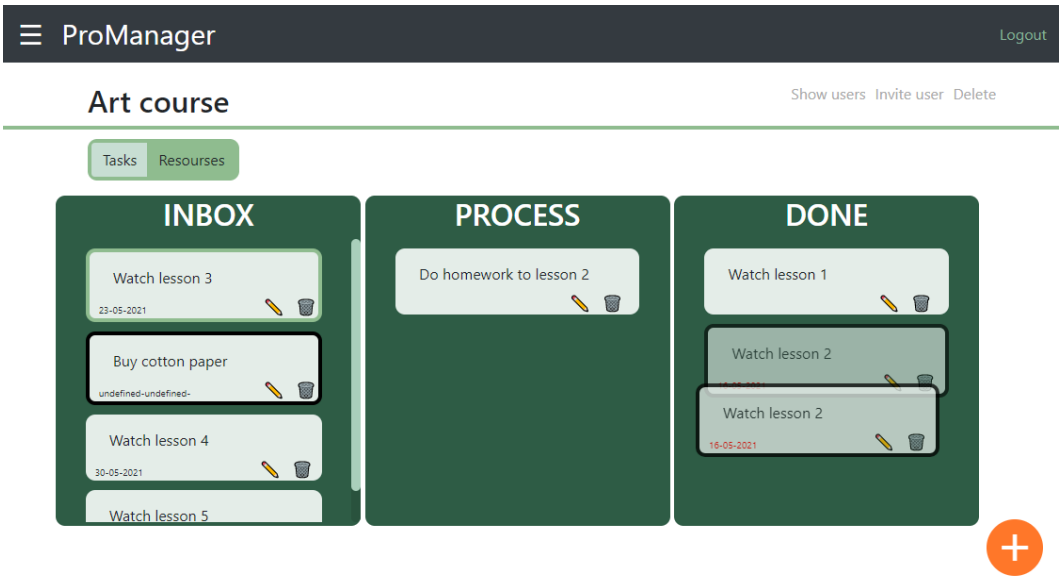

Рисунок 3.5.12 – перетягування задачі зі списку «Process» до «Done»

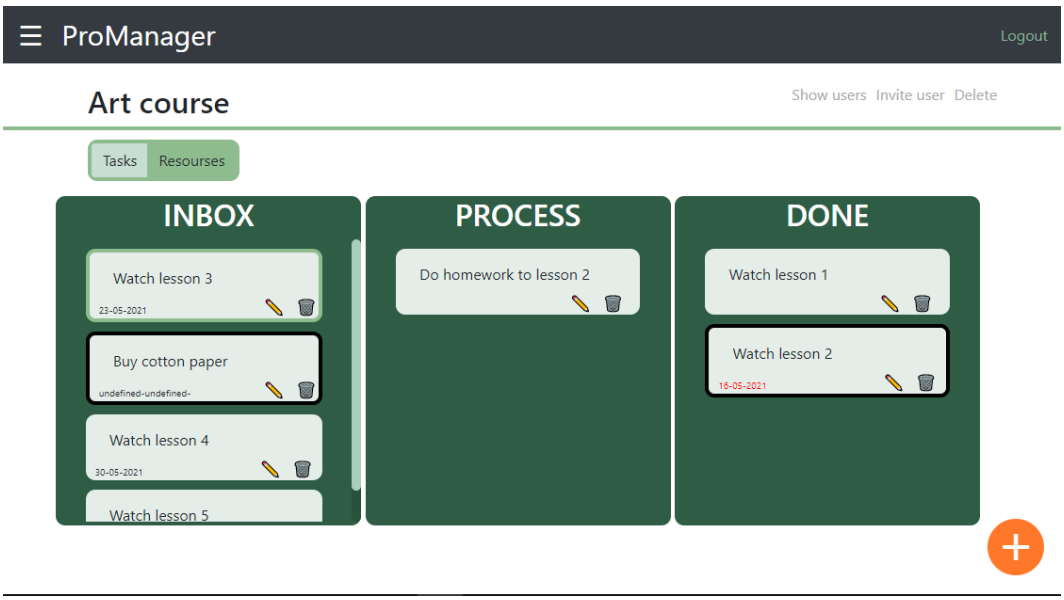

Рисунок 3.5.12 – результат перетягування задачі

Щоб побачити ресурси, які збережені в цьому проекту (рисунок 3.5.13), необхідно обрати вкладку «Resources» на панелі, що знаходиться над завданнями. Якщо таких нема, з’явиться відповідний напис, аби в користувача не було сумнівів, чи сторінка повністю завантажилась (див. рисунок 3.5.14).

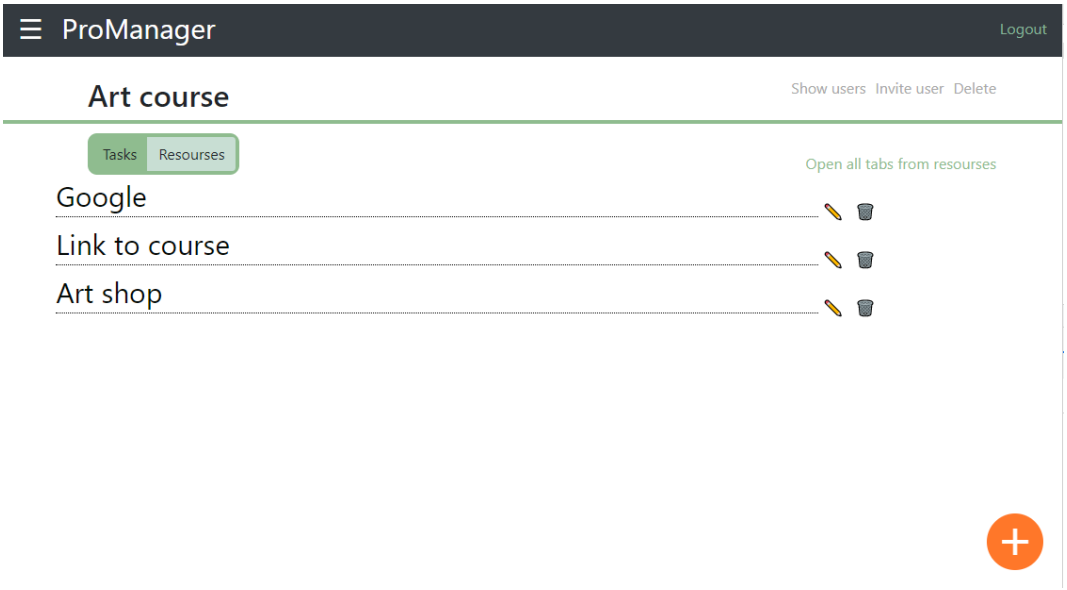

Рисунок 3.5.13 – сторінка ресурсів проекту

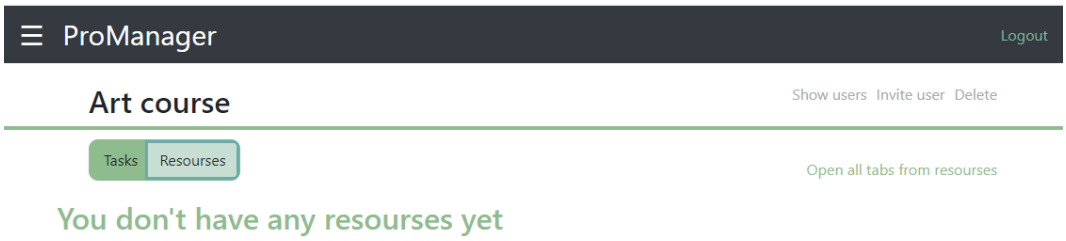

Рисунок 3.5.14 – сторінка ресурсів проекту при їх відсутності

Тримаючи курсор миші над назвою ресурсу, стає видимим вікно з підказкою – на той випадок, коли користувач не пам’ятає, на який сайт він додав посилання за відповідним ім’ям, або якщо імена ресурсів однакові (див. рисунок 3.5.15). Натиснувши на назву ресурсу, сторінка, на яку він посилається, відкриється в новому вікні.

Рисунок 3.5.15 – видиме посилання на ресурс

Зверху справа наявне додаткове посилання, що відкриє одразу всі ресурси (рисунок 3.5.16). Це дуже зручно, коли їх збережено небагато, та вони всі є потрібними для початку роботи.

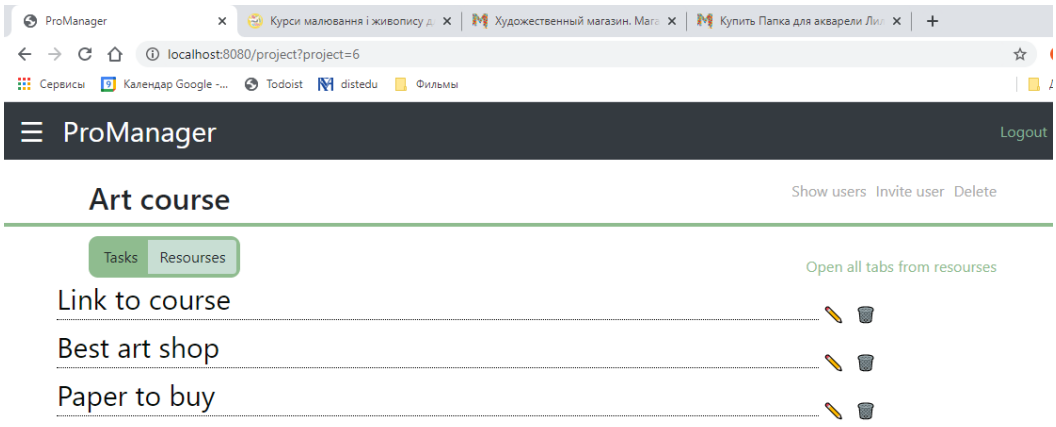

Рисунок 3.5.16 – відкриті ресурси у нових вкладках

Може трапитись, що дане посилання працюватиме некоректно. Це трапляється через налаштування браузерів, оскільки багато хто за замовченням

вмикає блокування спливаючих вікон на сайтах та переадресації. В такому випадку треба перейти на відповідну для браузеру користувача сторінку (у випадку з Chrome – «chrome://settings/content/popups») та додати «http://localhost:8080/» до списку сайтів, яким переадресація доступна (див. рисунок 3.5.17).

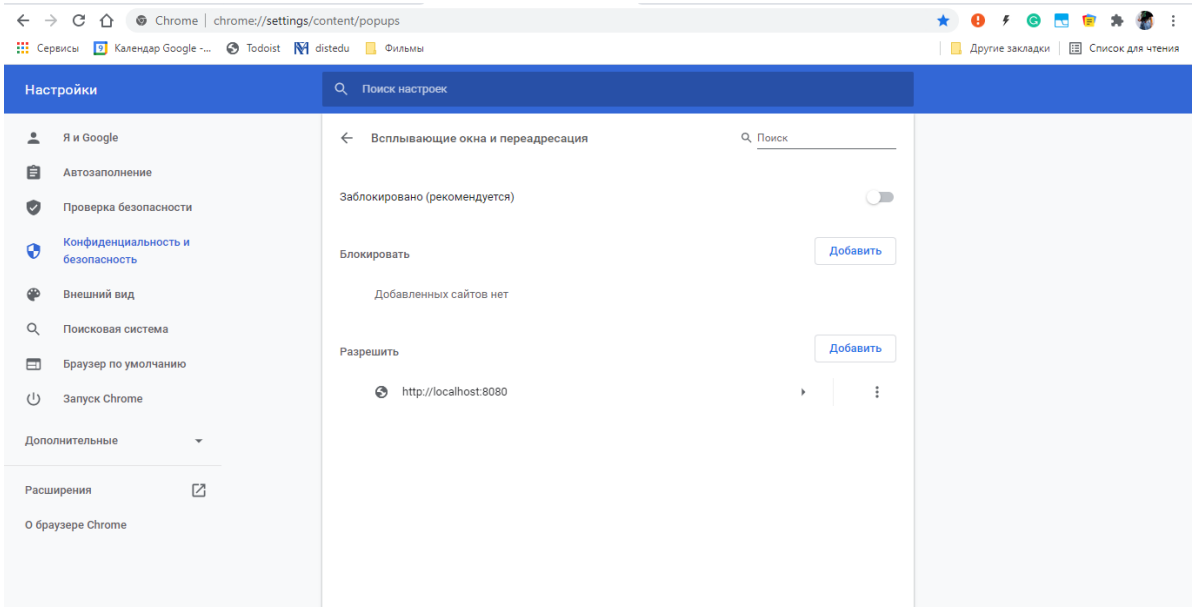

Рисунок 3.5.17 – необхідні налаштування браузеру на прикладі Chrome

Додавання, редагування та видалення ресурсів виглядає так само, як в задачах, для підтримання веб-застосунком цілісності з точки зору його інтерфейсу.

При спробі користувача перейти на недоступну для нього сторінку (наприклад, сторінку проекту, до якого він не має доступу) за посиланням, на екрані з’явиться напис про заборону доступу та користувач не отримує жодних даних, які не передбачувалося (див. рисунок 3.5.18).

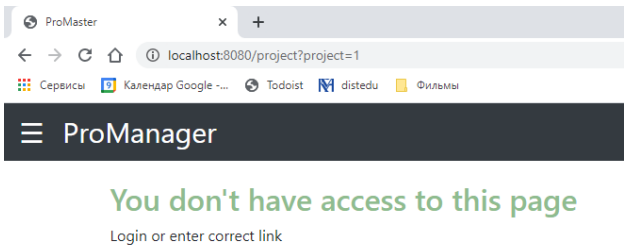

Рисунок 3.5.18 – помилка заборони доступу

ВИСНОВКИ

Під час дослідження було проведено аналіз технологій, які використовуються при написанні веб-застосунків, зокрема, досліджено фреймворк Spring, що значно спрощує написання великих веб-застосунків, автоматично реалізуючи взаємодію між деякими з класів проекту. Особливої уваги заслуговує робота Spring з базами даних, яка також дозволяє розробнику автоматично реалізувати багато з запитів до неї чи зв’язок між різними сутностями в ній. Досліджено шаблонізатор Thymeleaf для кращої реалізації відображення елементів на сторінці, використовуючи фрагменти та інші його можливості.

У ході написання курсової роботи було розроблено веб-додаток для планування задач, що повністю відповідав би поставленому завданню та міг задовільнити вимоги потенційного користувача.

При подальшій роботі над даним веб-застосунком можна додати функціонал, який зробить досвід користувачів даної програми при роботі з нею ще кращим. Серед них:

1. Надання користувачу, що створив проект, додаткових прав на нього, таких як закриття доступу до проекту людям, що за якихось причин більше не працюють над ним (наприклад після звільнення співробітника).
2. Заборона видалення проекту для тих, хто його не створював, оскільки наразі всі мають рівні права на проект.
3. Україномовна локалізація веб-застосунку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Дэвид Флэнаган. JavaScript. Подробное руководство, 5-е издание. Пер. с англ. – СПб: Символ-Плюс, 2008. 256 с.
2. JavaScript Versions [Електронний ресурс]: стан на 17 травня 2021. Режим доступу: URL: https://www.w3schools.com/js/js_versions.asp
3. Дэвид Флэнаган. JavaScript. Подробное руководство, 5-е издание. Пер. с англ. – СПб: Символ-Плюс, 2008. 513 с.
4. Уоллс Крейг. Spring в действии. – М.: ДМК Пресс, 2013. 35 с.
5. Ілюстрація. [Електронний ресурс] : стан на 17 травня 2021. Режим доступу: URL: https://docs.spring.io/spring-framework/docs/5.0.0.RC2/spring-framework-reference/images/spring-overview.png
6. Laurentiu Spilca. Spring Security in action. Manning Publications, 2020. 3 p.
7. Введение в Spring Boot: создание простого REST API на Java [Електронний ресурс] : стан на 17 травня 2021. Режим доступу: URL: https://habr.com/ru/post/435144/
8. Spring Data JPA. Пишем DAO и Services. Часть 2 [Електронний ресурс] : стан на 17 травня 2021. Режим доступу: URL: https://devcolibri.com/spring-data-jpa-%D0%BF%D0%B8%D1%88%D0%B5%D0%BC-dao-%D0%B8-services-%D1%87%D0%B0%D1%81%D1%82%D1%8C-2/
9. Web-сервисы Java. – Спб.: БХВ-Петербург, 2012. 295 с.
10. Движок шаблонов Thymeleaf [Електронний ресурс] : стан на 17 травня 2021. Режим доступу: URL: https://alexkosarev.name/2017/08/08/thymeleaf-template-engine/
11. Tutorial: Using Thymeleaf [Електронний ресурс] : стан на 17 травня 2021. Режим доступу: URL: https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#introducing-thymeleaf