МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ “ОДЕСЬКА ПОЛІТЕХНІКА”

Інститут комп'ютерних систем

Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни:

«Об'єктно-орієнтоване програмування»

на тему:

«Створення веб-додатку для планування задач»

Розробив:

Бойчук О.В.

Група:

АІ-205

Керівник:

Годовіченко М.А.

2022 рік

ЗМІСТ

ВСТУП

Коли час твій робочий інструмент, то їм необхідно володіти повною мірою. Щодня утримувати великий потік інформації в голові неможливо. І в таких випадках допомагають програми-планери.

При слові «планування» завжди спрацьовує стандартна асоціація — товстий щоденник у шкіряній палітурці, де виведено акуратний список щоденних завдань. Однак у рамках сучасних реалій цей спосіб втратив свою актуальність, поступаючись місцем різним сервісам та додаткам, якими можна легко користуватися як з десктопу, так і через мобільний додаток.

Так як проблема планування завжди актуальна, доцільним є створення веб-додатку, що задовольняє потреби користувача у плануванні задач.

Метою роботи є розробка планувальника задач у вигляді веб-додатку.

Для досягнення мети слід:

– проаналізувати існуючі додатки схожого типу;

– обрати технології для проектування та реалізації веб-додатку для планування задач;

– визначити функціональні вимоги та здійснити проектування інформаційної системи;

– реалізувати спроектовану систему;

– провести тестування розробленої інформаційної системи та переконатися у тому, що розроблена система задовольняє вимогам, що були визначені на етапі проектування.

I. АНАЛІЗ ЗАДАЧІ СТВОРЕННЯ ВЕБ-ДОДАТКУ ДЛЯ ПЛАНУВАННЯ ЗАДАЧ

1.1. Постановка задачі

Створити веб-додаток для планування задач, виконавши наступні дії:

- Створення сутностей системи;
- Авторизація користувачів системи;
- Аутентифікація користувачів;
- Розробка функціоналу сайту.

1.2. Огляд аналогів

Основними аналогами даної роботи можна назвати такі додатки:

- Any.do;
- Google Tasks;
- Todoist;
- Microsoft To-Do.

1.2.1. Any.do

Any.do — відмінний додаток із простим та зрозумілим інтерфейсом, який забезпечує швидке та просте управління завданнями та навіть інтегрується з додатком iOS Reminders. Таким чином, ви можете повідомити Siri нагадування, і воно з'явиться у Any.do. Однак синхронізація працює тільки в одному напрямку: видалення завдань з Any.do не призведе до його видалення з програми нагадування для iPhone, але якщо ви використовуєте Any.do як основну програму, це не буде проблемою. Є також додаткові зручні функції, такі як автоматичне сортування списків продуктів та планування дня, щоб допомогти розставити пріоритети для завдань.

Синхронізація між ПК, планшетом та телефоном – ще один приємний додаток до підтримки iOS та Android. За перехід на преміум-версію потрібна щомісячна плата в розмірі 3,50 $ для Android і 5 $ для iOS, але вона розширює можливості програми завдяки необмеженим завданням, кольоровим міткам і прапорцям, нагадуванням про місцезнаходження і 100 ГБ для зберігання файлів.

Переваги:

- Крос-платформність;
- Синхронізація з голосовими помічниками Siri, Alexa.

Недоліки:

- Працює не завжди коректно - іноді завдання можуть просто "злітати".

1.2.2. Google Tasks

Google Tasks такий самий простий, як список справ на папері. Це приголомшливо мінімалістичний і добре розроблений додаток, який робить саме те, що має, і не більше. Ви можете створювати завдання, складати їх опис, а потім додавати підзавдання. Вони з'являються в маркованому списку, і можна помітити кожне підзавдання завершеним, коли прийде час. Кожне завдання знаходиться під списком, і кількість списків, які можна створити, не обмежена.

Ви можете мати список покупок, список справ та багато іншого. В обмін на простоту Google Tasks втрачаємо деякі глибші теги та організаційні функції, які ви можете знайти в інших програмах. Завдання Google доступні на iOS та Android. Якщо ви використовуєте Gmail в Інтернеті, ви можете побачити огляд своїх завдань у правому краю інтерфейсу, поруч із програмами Календар та Google Keep.

Переваги:

- Безкоштовність;
- Простий інтерфейс.

Недоліки:

- Обмежені можливості деталізації та планування.

1.2.3. Todoist

Якщо вам потрібна спеціальна програма зі списком справ, то до Todoist варто придивитися. Це одна з найбільших програм з величезною кількістю користувачів та перевіреною роками ефективністю. Ви можете зареєструватися за допомогою свого профілю Facebook або облікового запису Google, і почати роботу так само просто, як ввести своє перше завдання і натиснути «Надіслати».

Можна встановити крайній термін виконання, а також призначити пріоритет задачі або покласти її в групу схожих завдань. Виконання включає установку галочки поруч із завданням, і є певне задоволення від цієї дії і його анімації. Налаштування нагадувань про завдання, додаткові активні проекти, коментарі до завдань та автоматичне резервне копіювання — це додаткові функції, а передплата на Todoist Premium приноситиме вам близько 29$ на рік. Тим не менш, якщо ви випробували його і вважаєте, що воно того варте, то 29 $ - це розумно для річної ціни.

Переваги:

- Популярність;
- Велика кількість налаштувань;
- Взаємодія з десятками сервісів, таких як: Gmail, Dropbox, Amazon Alexa;
- Групове використання для виконання спільних проектів.

Недоліки:

- Багато функцій заблоковано у безкоштовній версії.

1.2.4. Microsoft To-Do

Організаційні інструменти рідко існують у «вакуумі». Якщо ви підключені до екосистеми Microsoft за допомогою електронної пошти Outlook і роботи в Office, вам може бути цікаво дізнатися, що Microsoft має власну програму для ведення справ.

Microsoft To-Do, створена командою Wunderlist після того, як Microsoft придбала цю програму в 2015 році. Продукт від «дрібном'яких» напрочуд схожий на Wunderlist — і це непогано; постановка нових завдань проста, і вона пропонує майже те саме з погляду інструментів та функцій. Microsoft To-Do відрізняється тим, що робить акцент на My Day, а саме на ідеї, що ви починаєте щодня з чистого аркуша, а на початку кожного дня приділяєте час тому, щоб записати, чого ви хочете досягти цього дня. Це філософія життя без суєти, і вона спрямована на те, щоб користувачі зосередилися на тому, що відбувається тут і зараз. Це підходить не для всіх, і якщо ви любите планувати заздалегідь, Microsoft To-Do це теж дозволяє. У нього навіть вбудований інтелектуальний інструмент підказок, який запропонує вам завдання на основі вашого попереднього запису.

Здається, у майбутньому Microsoft To-Do синхронізуватиметься з Wunderlist, і ви зможете імпортувати завдання Wunderlist, якщо ви переходите з цієї програми. У майбутньому планується інтеграція з іншими службами Microsoft, що також дуже зручно.

Переваги:

- Простота використання;
- Синхронізація із сервісами Microsoft.

Недоліки:

- Поки що не весь функціонал доступний.[1]

1.3. Вибір технологій

Виходячи зі сформованих базових вимог до функціоналу додатка, можна провести вибір стека технологій, які дозволять реалізувати інформаційну систему планувальника задач.

Одним з можливих архітектурних шаблонів є клієнт-серверна система, яку можна представити триланковою архітектурою (рис. 1.1).

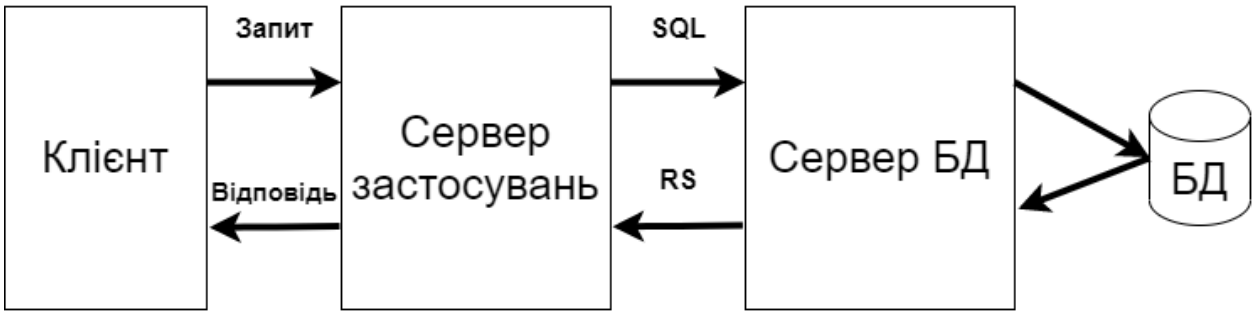


Рисунок 1.1 – Логічна схема класичної триланкової архітектури

Компоненти триланкової архітектури:

– клієнт – цей компонент відповідає за подання даних кінцевому користувачеві;

– виділений сервер додатків – тут міститься бізнес-логіка програми;

– сервер БД – надає запитувані дані.

Архітектура "клієнт-сервер" визначає загальні принципи організації взаємодії, де є сервери (вузли-постачальники деяких специфічних функцій і сервісів) і клієнти, (споживачі цих сервісів).

Між клієнтами і серверами повинні бути встановлені правила взаємодії, які називаються протоколом взаємодії або протоколом обміну. Кожна частина взаємодіє один з одним, обмінюючись повідомленнями в заздалегідь узгодженому форматі.

В результаті аналізу можливих підходів до реалізації триланкової клієнт-серверної архітектури було вирішено використовувати монолітну архітектуру.

Як клієнт в даній інформаційній системі було обрано html-документи для взаємодії із серверною частиною.

У якості фреймворку для реалізації серверної частини був обраний фреймворк Java Spring [2]. Вибір був обумовлений тим, що, також як і для розробки Android-клієнта, в даному фреймворку використовується мова Java, що спрощує і прискорює процес розробки і внесення змін в інформаційну систему. Також фактором є попереднє знайомство з даними фреймворком в процесі вивчення дисципліни «Об'єктно-орієнтоване програмування».

Java Spring – вільно-розповсюджуваний легкий фреймворк, покликаний спростити розробку корпоративних і веб-додатків (можна використовувати і для будь-яких інших типів додатків) на мові Java (є альтернативною стеку JavaEE).

Дамо коротку характеристику деяким модулям Spring:

– Spring Core - ядро платформи, надає базові засоби для створення додатків - управління компонентами (бінамі, beans), впровадження залежностей, MVC фреймворк, транзакції, базовий доступ до БД. В основному це низькорівневі компоненти і абстракції. По суті, неявно використовується всіма іншими компонентами;

– Spring MVC - забезпечує архітектуру паттерна Model-View-Controller за допомогою слабо пов'язаних готових компонентів для розробки веб-додатків;

– Spring Data - забезпечує доступ до даних: реляційні і нереляційні БД, KV сховища тощо;

– Spring Security - авторизація та аутентифікація, доступ до даних, методи OAuth, LDAP, і різні провайдери.

Проект Spring Boot – рішення, яке дозволяє вам легко створювати повноцінні програми Spring, про які можна сказати «просто запусти».

Spring Boot дозволяє швидко створити і настроїти (тобто налаштувати залежності між компонентами) додаток, упакувати його в виконуваний самодостатній артефакт [3]. Це та сполучна ланка, яка об'єднує разом набір компонентів в готовий додаток. Особливості Spring Boot:

– створення повноцінних Spring-додатків;

– вбудований сервлет-контейнер (Tomcat або Jetty);

– використовується принцип «convention over configuration». Для більшості конфігурацій не потрібно нічого налаштовувати.

1.4. Висновки

В даному розділі була проаналізована предметна область планувальника задач. Було виявлено, що дана галузь буде стабільно затребуваною.

Був проведений аналіз популярних додатків у сфері планування задач. У результаті аналізу переваг та недоліків цих додатків було виявлено, що більшість з них здатні задовольнити потреби користувача, але частина функцій доступна за плату.

Також був проведений аналіз та були обрані технології для реалізації розроблюваної інформаційної системи. Була обрана класична триланкова клієнт-серверна архітектура. У якості клієнтської частини був обраний веб-додаток. У якості серверної частини був обраний фреймворк Java Spring, що дозволяє, завдяки технології Spring Boot, швидко розробити серверну частину та забезпечити функціональні потреби інформаційної системи.

II. ПРОЕКТУВАННЯ СИСТЕМИ

2.1. Мета та задачі інформаційної системи

Розроблювана інформаційна система являє собою веб-додаток, який призначений для офісних працівників, домашньої роботи.

Мета інформаційної системи: додаток дозволяє спланувати виконання будь-яких задач, планів.

Цільова аудиторія даного продукту – людина майже будь-якого соціального статусу, що потребує структуризації та планування своїх намірів.

Головними функціями програми є:

- Створення, видалення, редактування задач;
- Перегляд задач за умовами;
- Аутентифікація та авторизація користувача у системі;
- Видалення користувача з системи.

2.2. Функціональні вимоги до додатку

До інформаційної системи пред'являються наступні функціональні вимоги.

FR1 Реєстрація користувача.

FR1.1 Незареєстрований користувач натискає кнопку «Реєстрація», після чого вводить бажаний логін і пароль (не менше 8 символів), після чого натискає кнопку «Зареєструватися»;

FR1.2 У разі успішної реєстрації користувач потрапляє на головну сторінку додатка;

FR1.3 У разі невдачі реєстрації (такий логін вже є в системі), додаток просить користувача змінити логін, так як такий логін вже є в системі. Поля логіна і пароля в це випадку очищаються;

FR1.4 Користувач може натиснути кнопку «Показати пароль», щоб побачити введений пароль.

FR2 Авторизація користувача.

FR2.1 Незареєстрований користувач вводить логін і пароль в форму введення і натискає кнопку «Увійти»;

FR2.2 Якщо користувач ввів логін і \ або пароль некоректно, система видає помилку і просить ввести логін і пароль повторно. Поля введення логіна і пароля очищуються;

FR2.3 У разі вдалої авторизації, користувач потрапляє на головну сторінку додатка;

FR2.4 Користувач також може вийти зі свого акаунту, натиснувши на відповідну кнопку.

F3 Створення, перегляд, видалення та редактування задач

FR3.1 Користувач може створити задачу увівши у поле назву задачі.

FR3.2 Користувач може помітити задачу як виконану.

FR3.3 Користувач може помітити задачу як важливу.

FR3.4 Користувач може видалити задачу.

FR3.5 Користувач може додати дату до задачі, також можливо видалити додану дату.

FR3.6 Користувач може додати категорію або декілька категорій до задачі, також можливо видалити додані категорії.

FR3.7 Користувач може змінити назву задачі.

F4 Перегляд створених задач.

FR4.1 Натиснувши на відповідну вкладку на боковій панелі, користувач може отримати лише виконані задачі.

FR4.2 Натиснувши на відповідну вкладку на боковій панелі, користувач може отримати лише важливі задачі.

FR4.3 Натиснувши на відповідну вкладку на боковій панелі, користувач може отримати лише задачі, що потрібно виконати, тобто невиконані задачі.

FR4.4 Натиснувши на відповідну вкладку на боковій панелі, користувач може отримати лише задачі, що мають дату.

FR4.5 Натиснувши на відповідну вкладку на боковій панелі, користувач може отримати лише задачі, що не мають дати.

FR5 Сортування задач.

FR5.1 Користувач може відсортувати задачі за часом створення, алфавітом, важливістю або датою.

FR6 Перегляд та редагування профілю користувача.

FR6.1 Користувач може переглянути свій профіль, в якому міститься інформація про нікнейм.

FR6.2 Користувач може, переглянувши свій профіль, видалити його.

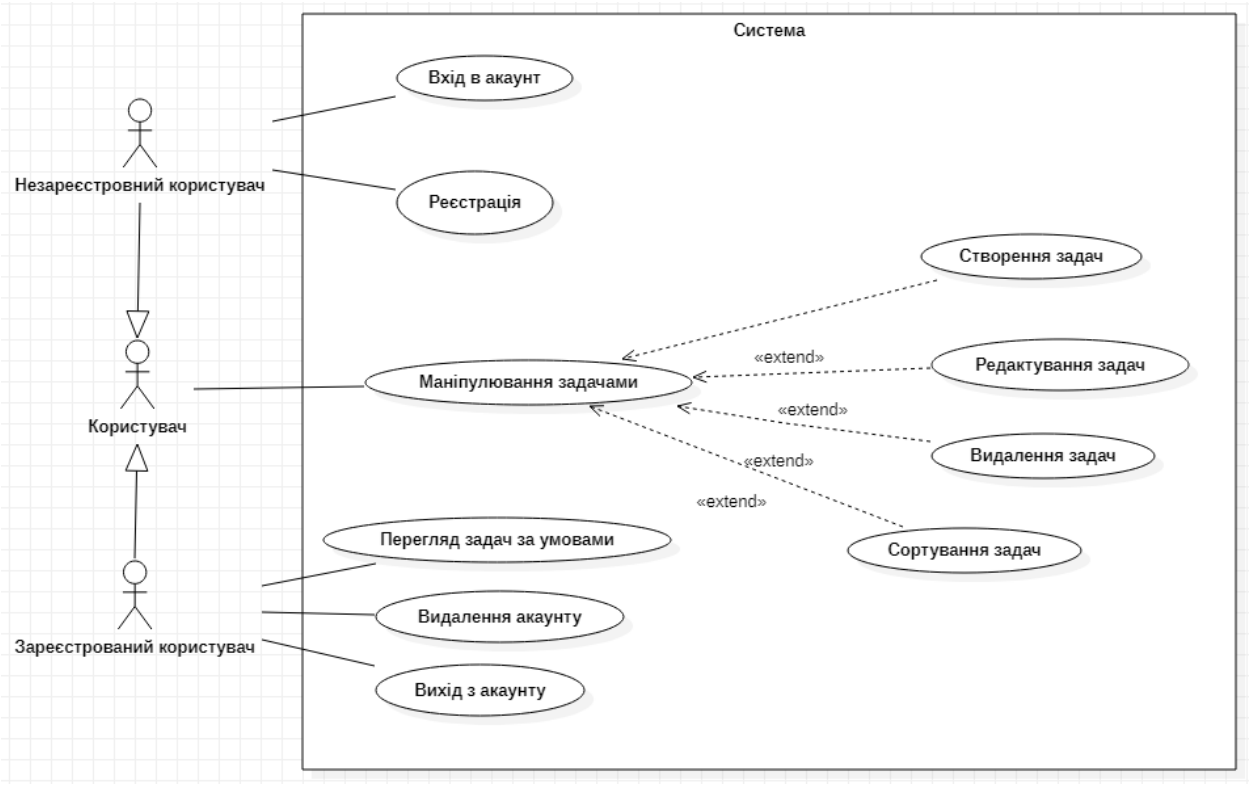
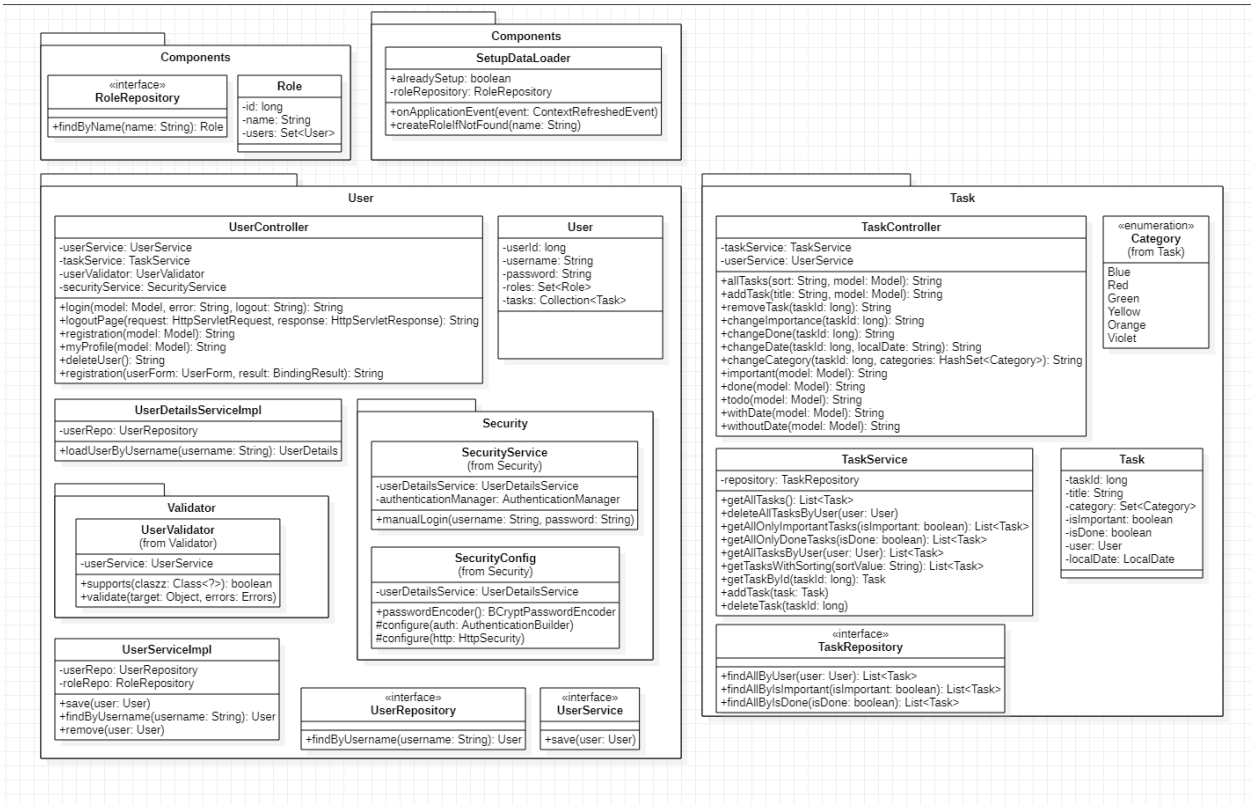


Рис. №2.1. Діаграма основних сценаріїв

2.3. Проектування структури додатку

На рисунках №2.1 та №2.2 показані концептуальні діаграми класів, на яких вказані основні класи додатку, які важливі для розуміння структури додатка в цілому, також в діаграмі зазначені основні поля й операції в класах.



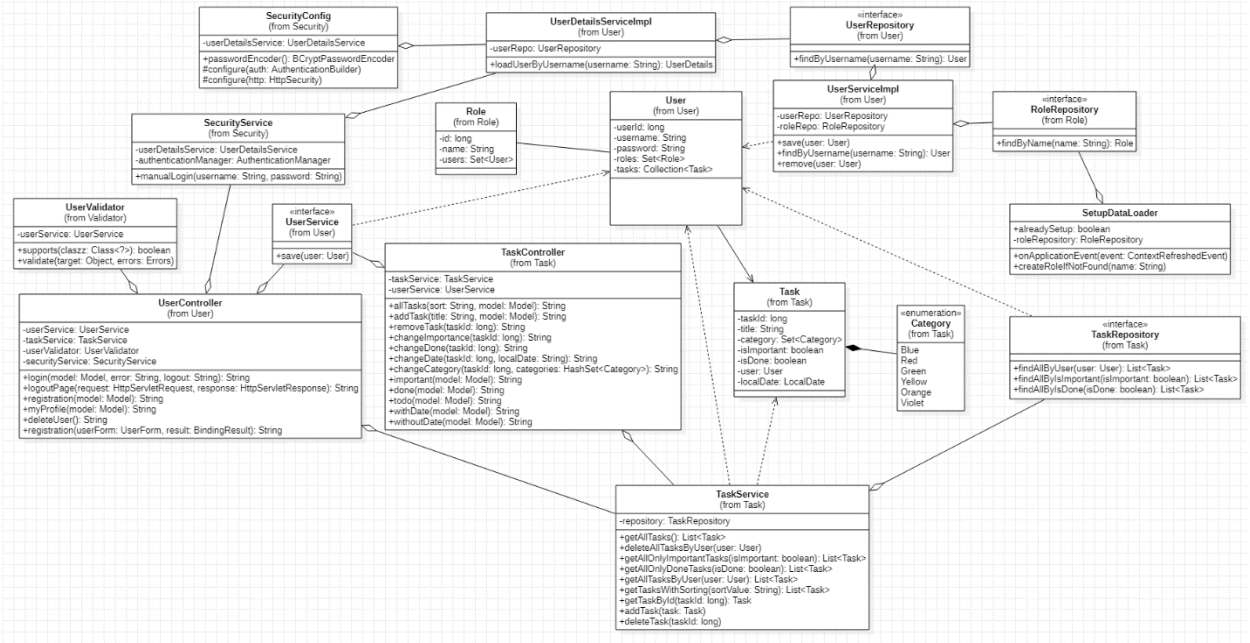


Рис. №2.2-2.3. Концептуальні діаграми класів додатка

2.4. Проектування сценаріїв

Для розуміння поведінки системи в динаміці необхідно змоделювати процеси, які відбуваються в системі в процесі реалізації сценаріїв роботи програми.

Розглянемо сценарій створення задачі. В рамках даного сценарію виконуються наступна послідовність дій:

– Користувач вводить заголовок задачі;

– У методі контроллера створюється нова задача з дефолними параметрами та даним заголовком;

– За допомогою екземпляру (об’єкту) сервісного класу задача створюється у БД, пройшовши шлях репозиторію, що інтерпритує запит об’єкту у БД.

На рисунку №2.4 представлена діаграма послідовності, яка дозволить більш детально описати взаємодію між об'єктами інформаційної системи для сценарію «Виведення задач за умовою».

На рисунку №2.5 представлена діаграма послідовності, яка дозволить більш детально описати взаємодію між об'єктами інформаційної системи для сценарію «Створення задачі».

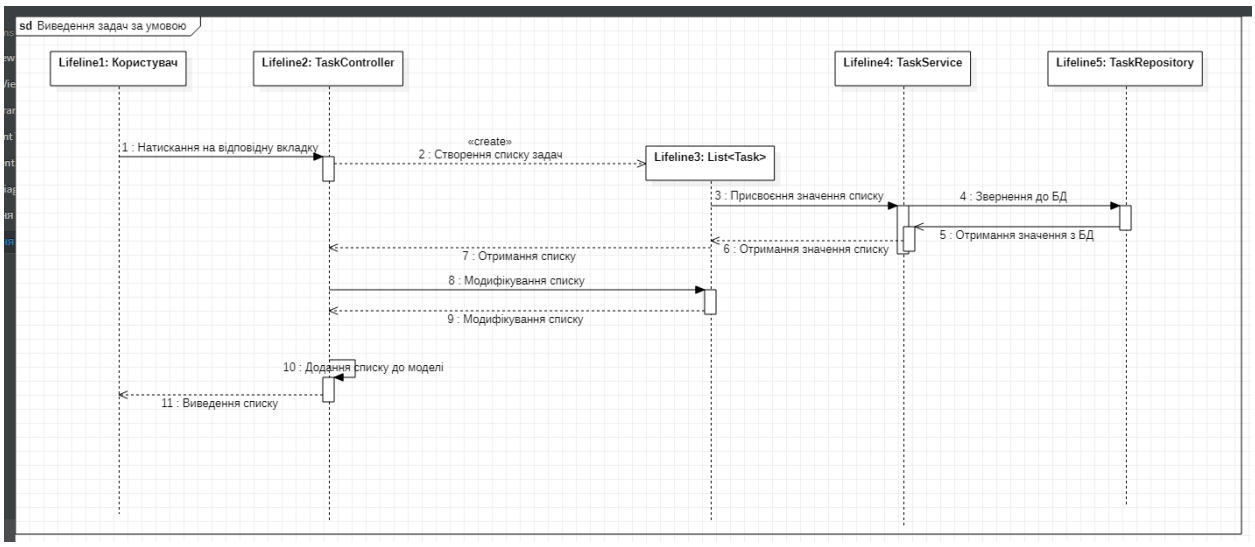


Рисунок №2.4 – Діаграма послідовностей сценарію «Перегляд задач за умовою»

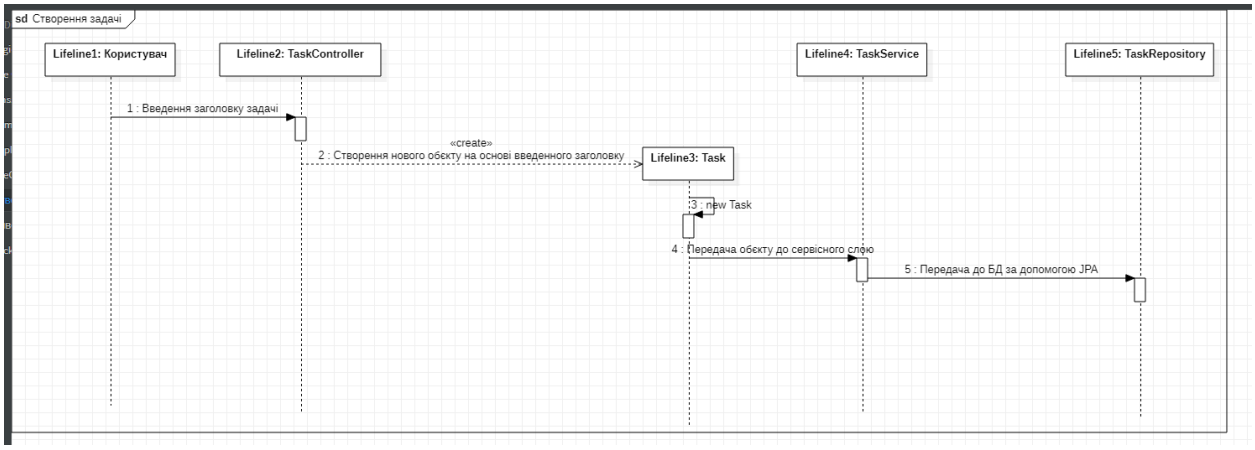


Рисунок №2.5 – Діаграма послідовностей сценарію «Створення задачі»

III. РЕАЛІЗАЦІЯ ДОДАТКУ ДЛЯ ПЛАНУВАННЯ ЗАДАЧ

3.1. Структура програмного проекту

Формування структури проекту інформаційної системи вироблено згідно з проектом. Додаток складається з серверної та клієнтської частин.

Клієнтська частина представлена у вигляді html файлів з вбудованими стилями, графічними зображеннями й скриптами.

Серверна частина інформаційної системи являє собою серверний додаток, яке базується на фреймворку Spring. Проект був виконаний в середовищі розробки IntelliJ IDEA Community Edition. Структура проекту зображена на рисунку 3.1.

Розглянемо основні класи й інтерфейси проекту і дамо їм коротку характеристику:

Task – клас, потрібний для створення, редактування задач.

User – клас, потрібний для створення, редактування даних про користувачів. Цей клас є важливим для аутентифікації користувачів.

Role – клас, потрібний для створення, редактування ролей. Ролі потрібні для присвоєння їх юзерам задля авторизації. Було прийнято рішення не створювати окремий клас для привілегій, так як усі користувачі мають право на читання, запис, видалення данних.

TaskController, UserController – класи для маппінгу та маніпулювання вищеназваними класами.

TaskRepository, UserRepository, RoleRepository – інтерфейси, що наслідують інший інтерфейс – JpaRepository, що дозволяє переводити запити з мови Java на мову SQL. Ці інтерфейси можуть мати свої методи, що не належать до JpaRepository.

UserDetailsServiceImpl – клас для «ручної» авторизації, тобто для задання своїх правил авторизації.

UserServiceImp – клас, задачою якого є збереження, видалення, пошук за нікнеймом користувачів за допомогою репозиторію.

UserService – інтерфейс, який наслідує клас UserServiceImpl.

SecurityConfig – клас, що задає конфігурацію роботи SpringSecurity, тобто налаштування безпеки сайту для правильної роботи логіки доступу до веб-компонентів.

SecurityService – клас для створення аутинтифікаційного токену, що є частиною логіки SpringSecurity.

UserValidator – клас, що перевіряє деякі поля на задоволення деяких умов.

SetupDataLoader – клас, що створений для автоматичного створення деяких данних під час запуску додатку.

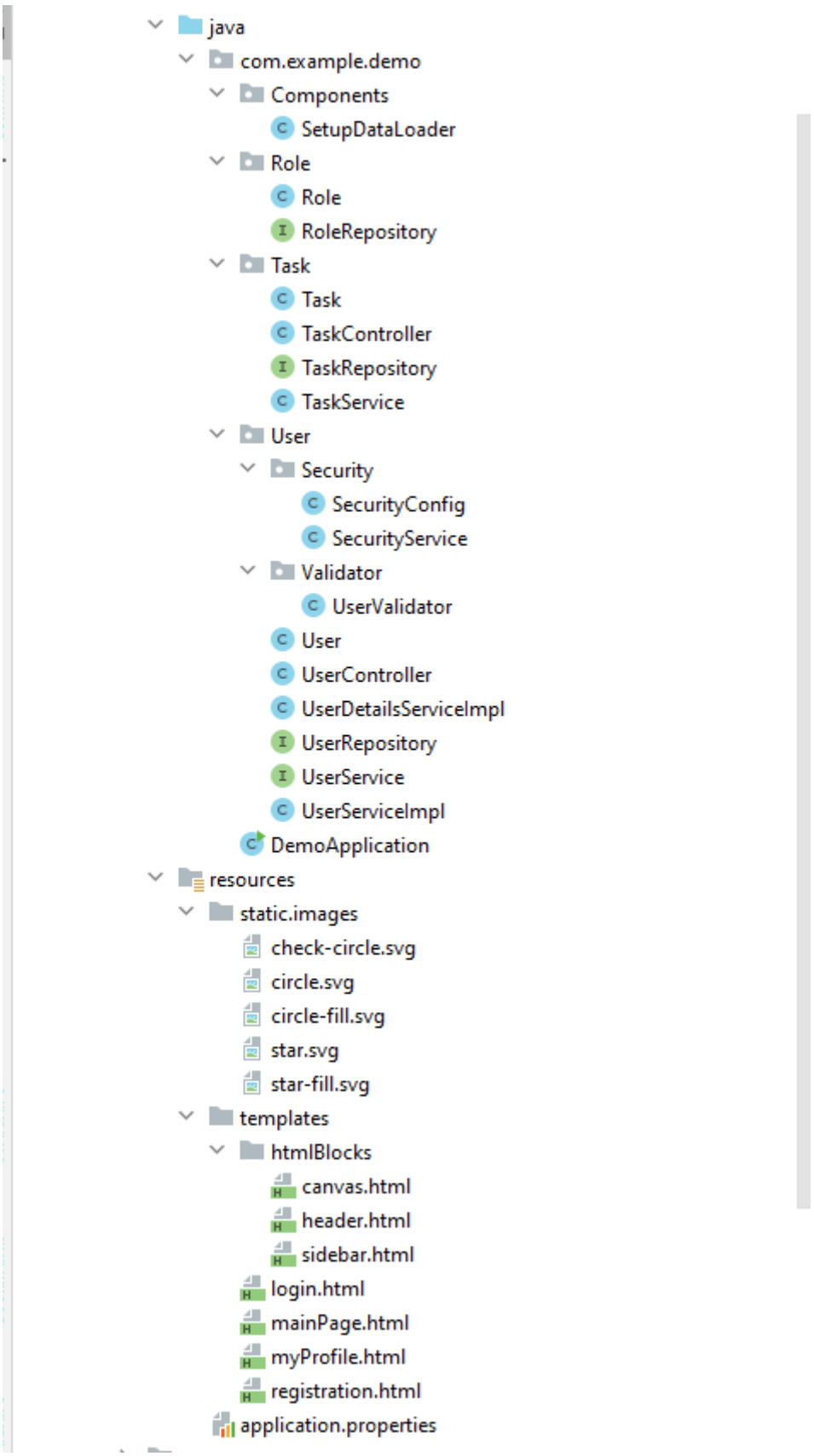


Рисунок 3.1 – Структура додатку

3.2. Реалізація основних алгоритмів

Алгоритм виведення усіх задач користувача, використовуючи сортування:

```
@RequestMapping(value = {"/","/tasks", "/home"}, method = RequestMethod.GET)
public String allTasks(@RequestParam(defaultValue = "taskId") String sort,
Model model){

    List<Task> list = taskService.getTasksWithSorting(sort);

    if(userService.findByUsername(getUsernameUsingSecurityContext())!=null) {

list.retainAll(taskService.getAllTasksByUser(userService.findByUsername(getUs
ernameUsingSecurityContext())));
    } else
    {
        list.removeIf(user -> user.getUser()!=null);
    }

    model.addAttribute("categories", new HashSet<Task.Category>());
    model.addAttribute("title", "My tasks");
    model.addAttribute("all_tasks", list);
    model.addAttribute("username", getUsernameUsingSecurityContext());

    return "mainPage";
}
```

Створюється список, що отримує усі задачі відсортовані за значенням «sort», що приймається з select у html-документі. Виконується перевірка на існування користувача за імям що повертає аутентифікатор:

- Якщо такий користувач існує, то за допомогою методу «retainAll» список зберігає лише обєкти, що передані у цей метод. У цей метод передаються усі задачі, що належать користувачу «username»;
- Якщо такого користувача не існує, то зі списку з усіма задачами видаляються усі задачі, що мають відношення до будь-якого користувача.

Отриманий список передається до моделі як атрибут із назвою «all_tasks», що використовується для виведення даних.

Алгоритм створення (додання до БД) задачі:

```
@PostMapping("/tasks/add")
public String addTask(@RequestParam String title,Model model)
{
```

```
    if(title!=null)
    {
        Task task = new Task(title, false,
false,userService.findByUsername(getUsernameUsingSecurityContext()));

        taskService.addTask(task);
        model.addAttribute("added_task", task);
    } else
    {
        throw new NullPointerException();
    }

    return "redirect:/tasks";
}
```

Значення «title» вводиться користувачем у поле input у html-документі.

– Якщо «title» не дорівнює null, то створюється обєкт класу Task, що у поле Task.title приймає значення «title», у поле Task.user приймає обєкт User, юзернейм якого збігається з юзернеймом аутентифікованого зараз юзера, а інші поля або отримують дефолтні значення, що встановленні програмістом (false, false), або отримують null значення.
  Далі ця задача (Task) додається до БД за допомогою сервісу, що викликає метод репозиторію.
  Далі задача додається до моделі для використання у html-документі або іншому контролері;
– Якщо «title» дорівнює нулю, то викидається NullPoinerException.

Перевірку на userService.findByUsername(getUsernameUsingSecurityContext())!=null робити не потрібно, тому що якщо метод повертає null то цей null записується у Task.user, що означає що задача не має свого користувача, тобто це задача гостя веб-додатку.

Алгоритм видалення задачі:

```
@RequestMapping(value="/tasks/remove/{id}", method = RequestMethod.POST)
public String removeTask(@PathVariable(value="id") long taskId)
{
    if (taskId>0) {
```

```
        taskService.deleteTask(taskId);
    } else
    {
        throw new IllegalArgumentException("Error! This task has wrong id
value");
    }
    return "redirect:/tasks";
}
```

«taskId» передається у метод для того щоб знати яку саме задачу слід видалити.

- Якщо «taskId»>0, то задача видаляється з БД за допомогою сервісу, що викликає у своєму методі (deleteTask) відповідний метод репозиторію для видалення задачі;
- Якщо «taskId»<=0, то викидається виняток IllegalArgumentException.

Редактування вмісту задачі:

```
@PostMapping(value = "/tasks/changeTitle/{id}")
public String changeTitle(@PathVariable(value="id") long
taskId,@RequestParam("titleChange") String title, Model model)
{

    if(title!=null && taskId>0 && taskService.getTaskById(taskId)!=null)
    {
        Task task = taskService.getTaskById(taskId);
        task.setTitle(title);
        taskService.addTask(task);
    } else
    {
        throw new IllegalArgumentException("Error! This task has wrong id
value or title value is null or task with this id doesnt exist");
    }

    return "redirect:/tasks";
}


@PostMapping(value = "/tasks/changeImportance/{id}")
public String changeImportance(@PathVariable(value="id") long taskId)
{
    if (taskId>0 && taskService.getTaskById(taskId)!=null) {
        Task task = taskService.getTaskById(taskId);
        task.setImportant(!taskService.getTaskById(taskId).isImportant());
        taskService.addTask(task);
    } else
    {
        throw new IllegalArgumentException("Error! This task has wrong id
value or such a task does not exist");
    }
```

```
    return "redirect:/tasks";
}
```

«taskId» передається у метод для того щоб знати яку саме задачу слід змінити. Також у метод може передаватись значення, на яке потрібно змінити поле обєкту задачі.

Перевіряється значення змінних, що отримав метод, для корректної роботи програми. Також перевіряється чи існує задача з даним айді («taskId»). Якщо значення задовольняють умовам, то створюється задача, що посилається на задачу з даним айді («taskId»). Після цього поле задачі змінюється на протилежне (changeImportance) або на передане у методі (changeTitle) й ця задача «додається» до БД (задача з тим ж айді змінює свої значення у БД).

3.3. Розгортання додатку

Для розгортання додатку було обранно хмарне середовище Heroku. Деплоювання додатку здійснювалось за допомогою GitHub репозиторію та аддону Heroku Postgres.

Посилання на додаток розміщенний у GitHub репозиторію boychuk-ol/kursach.

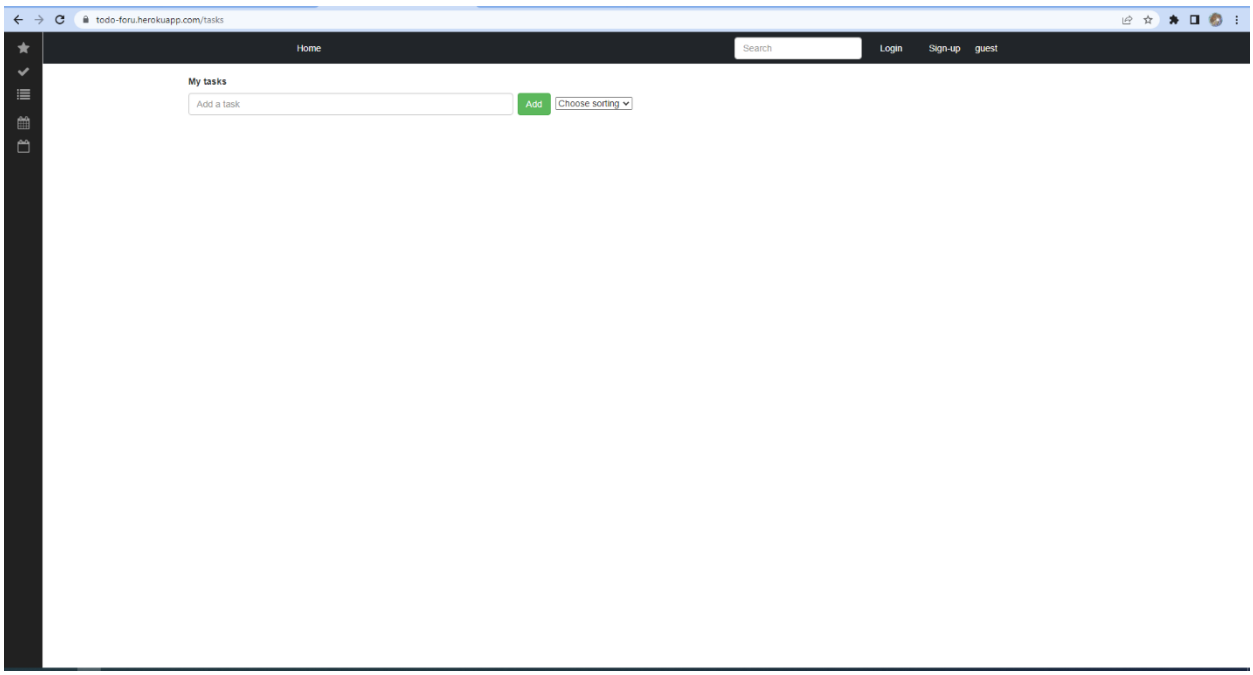


Рис. №3.2. Розгортання додатку за допомогою хмарного середовища Heroku

3.4. Документування інформаційної системи

У якості документації до розробленої інформаційної системи було створено керівництво користувача, яке дозволить ознайомитися з основним функціоналом системи. Воно забезпечене зображеннями з пояснювальним текстом на цих зображеннях.

Розроблена мінімалістична документація приведена в Додатку Б даної пояснювальної записки.

ВИСНОВКИ

У даній курсовій роботі була розроблена інформаційна система для планування задач.

Таким чином, мета, поставлена перед даною роботою досягнута в повному обсязі.

Для досягнення мети були вирішені наступні задачі. У першому розділі булу проаналізована предметна область планувальника задач. Був проведений аналіз схожих популярних веб-додатків. На підставі аналізу предметної області та аналізу додатків-аналогів були сформовані вимоги до основних функцій розроблюваної інформаційної системи. Також був проведений аналіз та були обрані технології для реалізації розроблюваної інформаційної системи.

У іншому розділі було здійснено проектування інформаційної системи. Були виділені мета, завдання системи, цільова аудиторія. Були виділені групи користувачів, для яких була сформована діаграма прецедентів та вимоги до системи. Були спроектовані макети інтерфейсу, було розроблено діаграму логічного представлення системи, а також діаграму розгортання. Були сформовані діаграми шарів системи.

У третьому розділі була реалізована спроектована інформаційна система. Були наведені структури проектів, а також розроблені концептуальні діаграми класів клієнтської і серверної частин. Були наведені описи ключових класів додатку. Була розроблена інструкція користувача.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Планировщики дел: 10 лучших программ» - Режим доступу: https://amssoft.ru/amsblog/planirovshchik-zadach.php

2. Spring Home [Електронний ресурс] – Режим доступу: https://spring.io/

3. Spring Boot [Електронний ресурс] – Режим доступу: https://spring.io/projects/spring-boot

ДОДАТОК А ПРОГРАМНИЙ КОД

```
package com.example.demo.Components;

import com.example.demo.Role.Role;
import com.example.demo.Role.RoleRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.stereotype.Component;
import javax.transaction.Transactional;


//Завантаження у бд данних при запуску серверу
@Component
public class SetupDataLoader implements ApplicationListener<ContextRefreshedEvent> {

    boolean alreadySetup = false;

    @Autowired
    private RoleRepository roleRepository;

    @Override
    @Transactional
    public void onApplicationEvent(ContextRefreshedEvent event) {

        if (alreadySetup)
            return;
        createRoleIfNotFound("user");
        createRoleIfNotFound("admin");
    }

    @Transactional
    Role createRoleIfNotFound(String name) {

        Role role = roleRepository.findByName(name);
        if (role == null) {
            role = new Role(name);
            roleRepository.save(role);
        }
        return role;
    }

    public RoleRepository getRoleRepository() {
        return roleRepository;
    }

    public void setRoleRepository(RoleRepository roleRepository) {
        this.roleRepository = roleRepository;
    }

}
```

```
package com.example.demo.Role;
import com.example.demo.User.User;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.sun.istack.NotNull;
import org.springframework.beans.factory.annotation.Autowired;

import javax.persistence.*;
import java.util.Set;


// Клас для надання ролі юзеру
@Entity
@Table(name="role_s")
public class Role {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private long id;

    @NotNull
    private String name;
```

```
    @Autowired
    public Role(String name) {
        this.name = name;
    }

    @Autowired
    public Role() {}

    public long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public Set<User> getUsers() {
        return users;
    }

    public void setUsers(Set<User> users) {
        this.users = users;
    }

    // Юзер може мати декілька ролей. Роль може належати декілька юзерам
    @ManyToMany(mappedBy = "roles")
    @JsonIgnore
    private Set<User> users;

}
```

```
package com.example.demo.Role;

import org.springframework.data.jpa.repository.JpaRepository;

public interface RoleRepository extends JpaRepository<Role, Long> {
    Role findByName(String name);
}
```

```
package com.example.demo.Task;

import com.example.demo.User.User;
import com.fasterxml.jackson.annotation.JsonIgnore;
import org.springframework.beans.factory.annotation.Autowired;
import javax.persistence.*;
import java.time.LocalDate;
import java.util.*;

// Клас для збереження задач
@Entity
@Table(name = "tasks")
public class Task {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private long taskId;

    private String title;

    // Завдання може мати деякі категорії, що відрізняються за кольором
    @ElementCollection
    private Set<Category> category;

    public enum Category {
        Blue, Red, Green, Yellow, Orange, Violet;
    }

    private boolean isImportant,isDone;
```

```
    private LocalDate localDate;

    // Юзер може мати багато завдань. Кожне завдання має відношення лише до одного юзера
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="user_id")
    @JsonIgnore
    private User user;

    @Autowired
    public Task(String title, boolean isImportant, boolean isDone) {
        this.title = title;
        this.category = new HashSet<>();
        this.isImportant = isImportant;
        this.isDone = isDone;
    }

    @Autowired
    public Task(String title, boolean isImportant, boolean isDone, User user) {
        this.title = title;
        this.category = new HashSet<>();
        this.user = user;
        this.isImportant = isImportant;
        this.isDone = isDone;
    }

    @Autowired
    public Task() {}

    public long getTaskId() {
        return taskId;
    }

    public void setTaskId(long taskId) {
        this.taskId = taskId;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public Set<Category> getCategory() {return category;}

    public void setCategory(Set<Category> category) {this.category = category;}

    public boolean isImportant() {
        return isImportant;
    }

    public void setImportant(boolean important) {
        this.isImportant = important;
    }

    public boolean isDone() {
        return isDone;
    }

    public void setDone(boolean done) {
        this.isDone = done;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public LocalDate getLocalDate() {return localDate;}

    public void setLocalDate(LocalDate localDate) {
        this.localDate = localDate;
    }
}
```

```java
package com.example.demo.Task;

import com.example.demo.User.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import java.time.LocalDate;
import java.util.*;


@Controller
public class TaskController {

    private final TaskService taskService;
    private final UserService userService;

    @Autowired
    public TaskController(TaskService taskService, UserService userService) {
        this.taskService = taskService;
        this.userService = userService;
    }

    public String getUsernameUsingSecurityContext()
    {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

        return authentication.getName();
    }

    // Головна сторінка сайту. Приймаємо параметр sort з документу для можливого сортування
завдань
    @RequestMapping(value = {"/","/tasks", "/home"}, method = RequestMethod.GET)
    public String allTasks(@RequestParam(defaultValue = "taskId") String sort, Model model){

        List<Task> list = taskService.getTasksWithSorting(sort);

        if(userService.findByUsername(getUsernameUsingSecurityContext())!=null) {

list.retainAll(taskService.getAllTasksByUser(userService.findByUsername(getUsernameUsingSecurityC
ontext())));
        } else
        {
            list.removeIf(user -> user.getUser()!=null);
        }

        model.addAttribute("categories", new HashSet<Task.Category>());
        model.addAttribute("title", "My tasks");
        model.addAttribute("all_tasks", list);
        model.addAttribute("username", getUsernameUsingSecurityContext());

        return "mainPage";
    }

    // Додання(створення) нового завдання. Приймаємо параметр title з документу(input) для
сетування поля Task.title
    @PostMapping("/tasks/add")
    public String addTask(@RequestParam String title,Model model)
    {

        if(title!=null)
        {
            Task task = new Task(title, false,
false,userService.findByUsername(getUsernameUsingSecurityContext()));

            taskService.addTask(task);
            model.addAttribute("added task", task);
        } else
        {
            throw new NullPointerException();
        }

        return "redirect:/tasks";
    }

    // Видалення завдання
```

```
    @RequestMapping(value="/tasks/remove/{id}", method = RequestMethod.POST)
    public String removeTask(@PathVariable(value="id") long taskId)
    {
        if (taskId>0) {
            taskService.deleteTask(taskId);
        } else
        {
            throw new IllegalArgumentException("Error! This task has wrong id value");
        }
        return "redirect:/tasks";
    }

    // Зміна заголовку завдання. Приймається параметр titleChange з документу(input) для зміни
    @PostMapping(value = "/tasks/changeTitle/{id}")
    public String changeTitle(@PathVariable(value="id") long taskId,@RequestParam("titleChange")
String title, Model model)
    {

        if(title!=null && taskId>0 && taskService.getTaskById(taskId)!=null)
        {
            Task task = taskService.getTaskById(taskId);
            task.setTitle(title);
            taskService.addTask(task);
        } else
        {
            throw new IllegalArgumentException("Error! This task has wrong id value or title
value is null or task with this id doesnt exist");
        }

        return "redirect:/tasks";
    }

    // Зміна важливості завдання. Для зміни використовується чекбокс, що перемикає значення
    @PostMapping(value = "/tasks/changeImportance/{id}")
    public String changeImportance(@PathVariable(value="id") long taskId)
    {
        if (taskId>0 && taskService.getTaskById(taskId)!=null) {
            Task task = taskService.getTaskById(taskId);
            task.setImportant(!taskService.getTaskById(taskId).isImportant());
            taskService.addTask(task);
        } else
        {
            throw new IllegalArgumentException("Error! This task has wrong id value or such a
task does not exist");
        }

        return "redirect:/tasks";
    }

    // Зміна статусу виконання завдання. Для зміни використовується чекбокс, що перемикає
значення
    @PostMapping(value = "/tasks/changeDone/{id}")
    public String changeDone(@PathVariable(value="id") long taskId)
    {
        if (taskId>0 && taskService.getTaskById(taskId)!=null) {
            Task task = taskService.getTaskById(taskId);
            task.setDone(!taskService.getTaskById(taskId).isDone());
            taskService.addTask(task);
        } else
        {
            throw new IllegalArgumentException("Error! This task has wrong id value or such a
task does not exist");
        }

        return "redirect:/tasks";
    }

    // Зміна дати завдання. Для зміни використовується параметр localDate з документу(input)
    @PostMapping(value = "/tasks/changeDate/{id}")
    public String changeDate(@PathVariable(value="id") long taskId, @RequestParam String
localDate)
    {

        if (taskService.getTaskById(taskId)!=null && taskId>0) {

            Task task = taskService.getTaskById(taskId);
            if (localDate != "" && localDate != null) {
                task.setLocalDate(LocalDate.parse(localDate));
            } else {
                task.setLocalDate(null);
```

```
            }
            taskService.addTask(task);
        } else
        {
            throw new IllegalArgumentException("Error! This task has wrong id value or such a
task does not exist");
        }

        return "redirect:/tasks";
    }

    // Додання/видалення категорій, що асоціюються із завданням. Для зміни використовується
параметр categories, що було передано у документ як значення modelAttribute у вигляді об'єкту
HashSet<Category>
    @PostMapping(value = "/tasks/changeCategory/{id}")
    public String changeCategory(@PathVariable(value="id") long taskId, @RequestParam(name =
"categories",required = false) HashSet<Task.Category> categories)
    {

        if (taskService.getTaskById(taskId)!=null && taskId>0) {
            Task task = taskService.getTaskById(taskId);
            task.setCategory(categories);
            taskService.addTask(task);
        } else
        {
            throw new IllegalArgumentException("Error! This task has wrong id value or such a
task does not exist");
        }

        return "redirect:/tasks";
    }


    // Отримання лише всіх важливих завдань
    @GetMapping(value = "/important")
    public String important(Model model)
    {

        List<Task> list = taskService.getAllOnlyImportantTasks(true);

        if (userService.findByUsername(getUsernameUsingSecurityContext())!=null) {

list.retainAll(taskService.getAllTasksByUser(userService.findByUsername(getUsernameUsingSecurityC
ontext())));
        } else
        {
            list.removeIf(user -> user.getUser()!=null);
        }

        model.addAttribute("title", "Important");
        model.addAttribute("all tasks", list);
        model.addAttribute("username", getUsernameUsingSecurityContext());

        return "mainPage";
    }

    // Отримання лише всіх виконаних завдань
    @GetMapping(value = "/done")
    public String done(Model model)
    {

        List<Task> list = taskService.getAllOnlyDoneTasks(true);

        if (userService.findByUsername(getUsernameUsingSecurityContext())!=null) {

list.retainAll(taskService.getAllTasksByUser(userService.findByUsername(getUsernameUsingSecurityC
ontext())));
        } else
        {
            list.removeIf(user -> user.getUser()!=null);
        }

        model.addAttribute("title", "Done");
        model.addAttribute("all_tasks", list);
        model.addAttribute("username", getUsernameUsingSecurityContext());

        return "mainPage";
    }

    // Отримання лише всіх невиконаних завдань
```

```
    @GetMapping(value = "/todo")
    public String todo(Model model)
    {

        List<Task> list = taskService.getAllOnlyDoneTasks(false);

        if (userService.findByUsername(getUsernameUsingSecurityContext())!=null) {

list.retainAll(taskService.getAllTasksByUser(userService.findByUsername(getUsernameUsingSecurityC
ontext())));
        } else
        {
            list.removeIf(user -> user.getUser()!=null);
        }

        model.addAttribute("title", "To do");
        model.addAttribute("all tasks", list);
        model.addAttribute("username", getUsernameUsingSecurityContext());

        return "mainPage";
    }

    // Отримання лише всіх завдань з датою
    @GetMapping(value = "/withDate")
    public String withDate(Model model)
    {

        List<Task> list = taskService.getAllTasks();
        list.removeIf(Task -> Task.getLocalDate()==null);

        if (userService.findByUsername(getUsernameUsingSecurityContext())!=null) {

list.retainAll(taskService.getAllTasksByUser(userService.findByUsername(getUsernameUsingSecurityC
ontext())));
        } else
        {
            list.removeIf(user -> user.getUser()!=null);
        }

        model.addAttribute("title", "With date");
        model.addAttribute("all tasks", list);
        model.addAttribute("username", getUsernameUsingSecurityContext());

        return "mainPage";
    }

    // Отримання лише всіх завдань без дати
    @GetMapping(value = "/withoutDate")
    public String withoutDate(Model model)
    {

        List<Task> list = taskService.getAllTasks();
        list.removeIf(Task -> Task.getLocalDate()!=null);

        if (userService.findByUsername(getUsernameUsingSecurityContext())!=null) {

list.retainAll(taskService.getAllTasksByUser(userService.findByUsername(getUsernameUsingSecurityC
ontext())));
        } else
        {
            list.removeIf(user -> user.getUser()!=null);
        }

        model.addAttribute("title", "Without date");
        model.addAttribute("all tasks", list);
        model.addAttribute("username", getUsernameUsingSecurityContext());

        return "mainPage";
    }
}
```

```
package com.example.demo.Task;

import com.example.demo.User.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface TaskRepository extends JpaRepository<Task, Long> {
```

```
    List<Task> findAllByUser(User user);
    List<Task> findAllByIsImportant(boolean isImportant);
    List<Task> findAllByIsDone(boolean isDone);
}
```

```
package com.example.demo.Task;

import com.example.demo.User.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class TaskService {

    private TaskRepository repository;

    @Autowired
    public TaskService(TaskRepository repository)
    {
        this.repository = repository;
    }

    public List<Task> getAllTasks()
    {
        return repository.findAll();
    }

    public void deleteAllTasksByUser(User user)
{repository.deleteAll(this.getAllTasksByUser(user));}

    public List<Task> getAllOnlyImportantTasks(boolean isImportant){return
repository.findAllByIsImportant(isImportant);}

    public List<Task> getAllOnlyDoneTasks(boolean isDone){return
repository.findAllByIsDone(isDone);}

    public List<Task> getAllTasksByUser(User user) {return repository.findAllByUser(user);}

    public List<Task> getTasksWithSorting(String sortValue){return
repository.findAll(Sort.by(Sort.Direction.ASC,sortValue));}

    public Task getTaskById(long taskId){return repository.getOne(taskId);}

    public void addTask(Task task)
    {
        repository.saveAndFlush(task);
    }

    public void deleteTask(long taskId){
        repository.deleteById(taskId);
    }

}
```

```
package com.example.demo.User.Security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuild
er;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

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

```
    private UserDetailsService userDetailsService;

    @Autowired
    public void setUserDetailsService(UserDetailsService userDetailsService) {
        this.userDetailsService = userDetailsService;
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(12);
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.authorizeRequests()
                .antMatchers("/tasks", "/", "/home", "/registration").permitAll()
                .antMatchers("/done", "/important", "/withDate", "/withoutDate",
"/todo").authenticated()
                .and()
                .formLogin()
                .loginPage("/login").permitAll()
                .usernameParameter("username")
                .passwordParameter("password")
                .defaultSuccessUrl("/tasks",true)
                .failureUrl("/login?error")
                .and()
                .logout().permitAll()
                .logoutUrl("/login?logout")
                .logoutSuccessUrl("/tasks")
                .invalidateHttpSession(true)
                .deleteCookies("JSESSIONID");
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception
    {
        return super.authenticationManagerBean();
    }
}



package com.example.demo.User.Security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.stereotype.Service;

@Service
public class SecurityService {

    private final UserDetailsService userDetailsService;
    private final AuthenticationManager authenticationManager;

    @Autowired
    public SecurityService(UserDetailsService userDetailsService, AuthenticationManager
authenticationManager) {
        this.userDetailsService = userDetailsService;
        this.authenticationManager = authenticationManager;
    }

    public void manualLogin(String username, String password)
    {

        UserDetails userDetails = userDetailsService.loadUserByUsername(username);

        UsernamePasswordAuthenticationToken token = new
UsernamePasswordAuthenticationToken(userDetails,password,userDetails.getAuthorities());
```

```
        authenticationManager.authenticate(token);

        if(token.isAuthenticated())
        {
            SecurityContextHolder.getContext().setAuthentication(token);
        }

    }
}
```

```
package com.example.demo.User.Validator;

import com.example.demo.User.User;
import com.example.demo.User.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

@Service
public class UserValidator implements Validator {

    private final UserService userService;

    @Autowired
    public UserValidator(UserService userService) {
        this.userService = userService;
    }

    @Override
    public boolean supports(Class<?> clazz) {
        return User.class.equals(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        User user = (User) target;

        // Поле username не должно быть пустым
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "username", "", "This field is
required.");
        // Поле username должно быть длиной от 8 до 32 символов
        if (user.getUsername().length() < 8 || user.getUsername().length() > 32) {
            errors.rejectValue("username", "", "Username must be between 8 and 32 characters");
        }
        // Поле username должно быть уникальным в системе И нельзя создать пользователя с
юзернеймом 'anonymousUser'
        if (userService.findByUsername(user.getUsername()) != null ||
user.getUsername().equals("anonymousUser")) {
            errors.rejectValue("username", "", "Username is already exists.");
        }

        // Поле password не должно быть пустым
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password", "", "This field is
required.");
        // Поле password должно быть длиной от 8 до 32 символов
        if (user.getPassword().length() < 8 || user.getPassword().length() > 32) {
            errors.rejectValue("password", "", "password must be between 8 and 32 characters");
        }
        // Поле password должно совпадать с полем confirmPassword
        if (!user.getPasswordConfirm().equals(user.getPassword())) {
            errors.rejectValue("password", "", "Passwords don't match!");
        }
    }
}
```

```
package com.example.demo.User;

import com.example.demo.Role.Role;
import com.example.demo.Task.Task;
import javax.persistence.*;
import java.util.Collection;
import java.util.Set;

// Клас юзерів(користувачів)
```

```java
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "user_id", nullable = false)
    private long userId;

    private String username,password;

    @Transient
    private String passwordConfirm;

    // Юзер може мати декілька ролей. Роль може належати декілька юзерам
    @ManyToMany
    @JoinTable(
            name = "user role",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role id")
    )
    private Set<Role> roles;

    // Юзер може мати безліч завдань. Будь-яке з завдань належить лише одному юзеру
    @OneToMany(mappedBy = "user",fetch = FetchType.EAGER)
    private Collection<Task> tasks;

    public String getPasswordConfirm() {
        return passwordConfirm;
    }

    public void setPasswordConfirm(String passwordConfirm) {
        this.passwordConfirm = passwordConfirm;
    }

    public long getUserId() {
        return userId;
    }

    public void setUserId(long userId) {
        this.userId = userId;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {this.username = username;}

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }
}
```

```java
package com.example.demo.User;

import com.example.demo.Task.TaskService;
import com.example.demo.User.Security.SecurityService;
import com.example.demo.User.Validator.UserValidator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
```

```java
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


@Controller
public class UserController {

    private final UserService userService;
    private final TaskService taskService;
    private final UserValidator userValidator;
    private final SecurityService securityService;

    @Autowired
    public UserController(UserService userService, TaskService taskService, UserValidator
userValidator, SecurityService securityService) {
        this.userService = userService;
        this.taskService = taskService;
        this.userValidator = userValidator;
        this.securityService = securityService;
    }

    public String getUsernameUsingSecurityContext()
    {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        return authentication.getName();
    }

    @GetMapping("/login")
    public String login(Model model, String error, String logout) {

        if(error != null)
        {
            model.addAttribute("error","Username or password is incorrect");
        }
        if (logout !=null)
        {
            model.addAttribute("message", "Logged out successfully");
        }

        return "login";
    }

    @RequestMapping(value="/logout", method = RequestMethod.GET)
    public String logoutPage (HttpServletRequest request, HttpServletResponse response) {

        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        if (auth != null){
            new SecurityContextLogoutHandler().logout(request, response, auth);
        }
        return "redirect:/login?logout";
    }

    // Передача браузеру страницы с формой
    @RequestMapping(value = {"/registration"}, method = RequestMethod.GET)
    public String registration(Model model) {
        model.addAttribute("userForm", new User());
        return "registration";
    }

    // Обработка данных формы
    @RequestMapping(value = "/registration", method = RequestMethod.POST)
    public String registration(@ModelAttribute("userForm") User userForm, BindingResult result) {

        // Валидация с помощью валидатора
        userValidator.validate(userForm, result);

        // Если есть ошибки - показ формы с сообщениями об ошибках
        if (result.hasErrors()) {
            return "registration";
        }

        // Сохранение пользователя в базе
        userService.save(userForm);

        securityService.manualLogin(userForm.getUsername(), userForm.getPasswordConfirm());

        // Перенаправление на страницу
        return "redirect:/tasks";
```

```
    }

    @GetMapping("/deleteUser")
    public String deleteUser() {

        if (userService.findByUsername(getUsernameUsingSecurityContext())!=null) {

taskService.deleteAllTasksByUser(userService.findByUsername(getUsernameUsingSecurityContext()));
            userService.remove(userService.findByUsername(getUsernameUsingSecurityContext()));
        }
        else throw new NullPointerException();

        return "redirect:/logout";
    }

    @RequestMapping(value = {"/myprofile"}, method = RequestMethod.GET)
    public String myProfile(Model model) {
        model.addAttribute("userName",getUsernameUsingSecurityContext());
        return "myProfile";
    }

}

package com.example.demo.User;

import com.example.demo.Role.Role;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.HashSet;
import java.util.Set;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    private UserRepository userRepo;

    @Autowired
    public UserDetailsServiceImpl(UserRepository userRepo) {
        this.userRepo = userRepo;
    }

    @Override
    @Transactional(readOnly = true)
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepo.findByUsername(username);

        Set<GrantedAuthority> grantedAuthorities = new HashSet<>();

        if(user!=null) {
            for (Role role : user.getRoles()) {
                grantedAuthorities.add(new SimpleGrantedAuthority(role.getName()));
            }
        }

        return new
org.springframework.security.core.userdetails.User(Objects.requireNonNull(user).getUsername(),
user.getPassword(), grantedAuthorities);

    }
}
package com.example.demo.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername (String username);
}

package com.example.demo.User;

public interface UserService {
    void save(User user);
    User findByUsername(String username);
```

```
    void remove(User user);
}
```

```
package com.example.demo.User;

import com.example.demo.Role.Role;
import com.example.demo.Role.RoleRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.HashSet;
import java.util.Set;

@Service
public class UserServiceImpl implements UserService {

    private final UserRepository userRepo;
    private final RoleRepository roleRepo;
    private final BCryptPasswordEncoder bCryptPasswordEncoder;


    @Autowired
    public UserServiceImpl(UserRepository userRepo, RoleRepository roleRepo,
BCryptPasswordEncoder bCryptPasswordEncoder) {
        this.userRepo = userRepo;
        this.roleRepo = roleRepo;
        this.bCryptPasswordEncoder = bCryptPasswordEncoder;
    }

    @Override
    public void save(User user) {
        user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        Set<Role> roles = new HashSet<>();
        roles.add(roleRepo.findByName("user"));
        user.setRoles(roles);
        userRepo.save(user);
    }

    @Override
    public User findByUsername(String username) {
        return userRepo.findByUsername(username);
    }

    @Override
    public void remove(User user) { userRepo.delete(user);}
}
```

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

   public static void main(String[] args) {

      SpringApplication.run(DemoApplication.class, args);

   }
}
```

ДОДАТОК Б ІНСТРУКЦІЯ КОРИСТУВАЧА

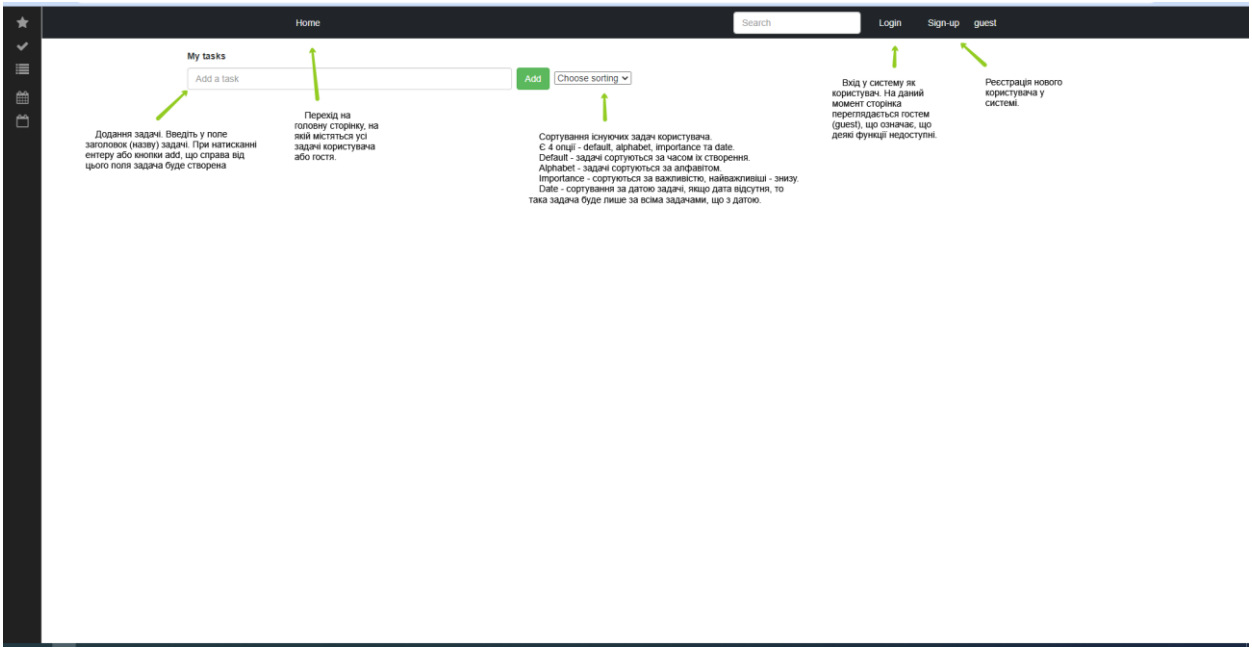


Рис. №4.1. Інструкція для гостя.

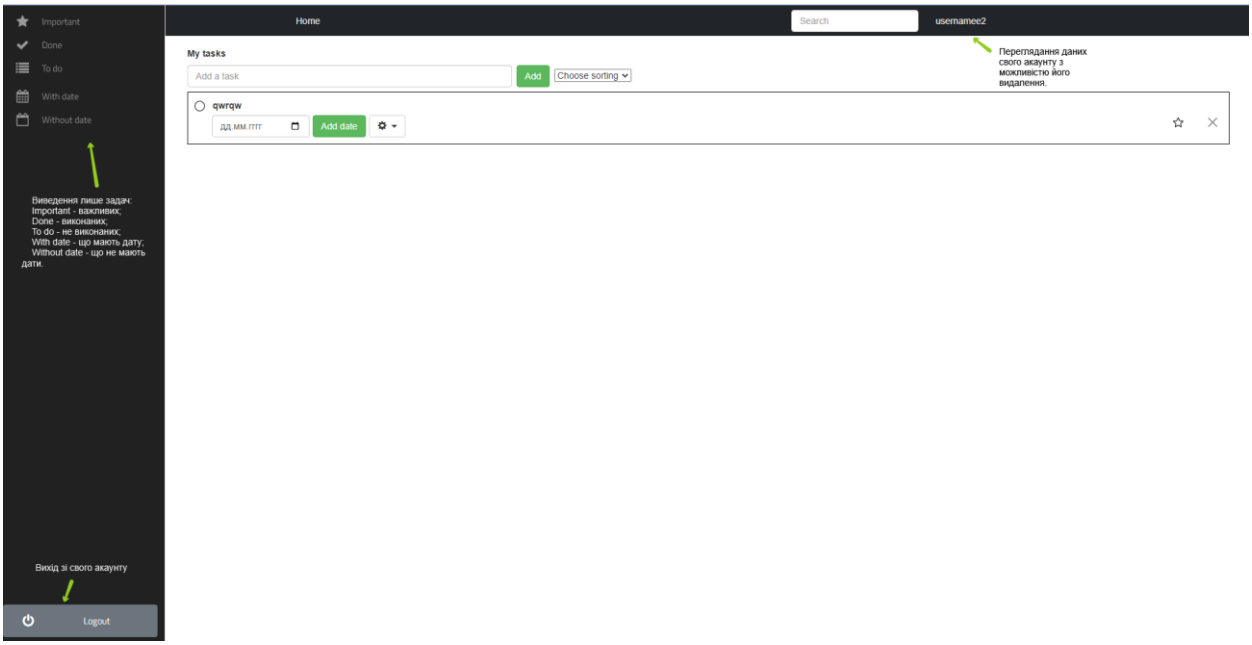


Рис. №4.2. Інструкція для користувача.

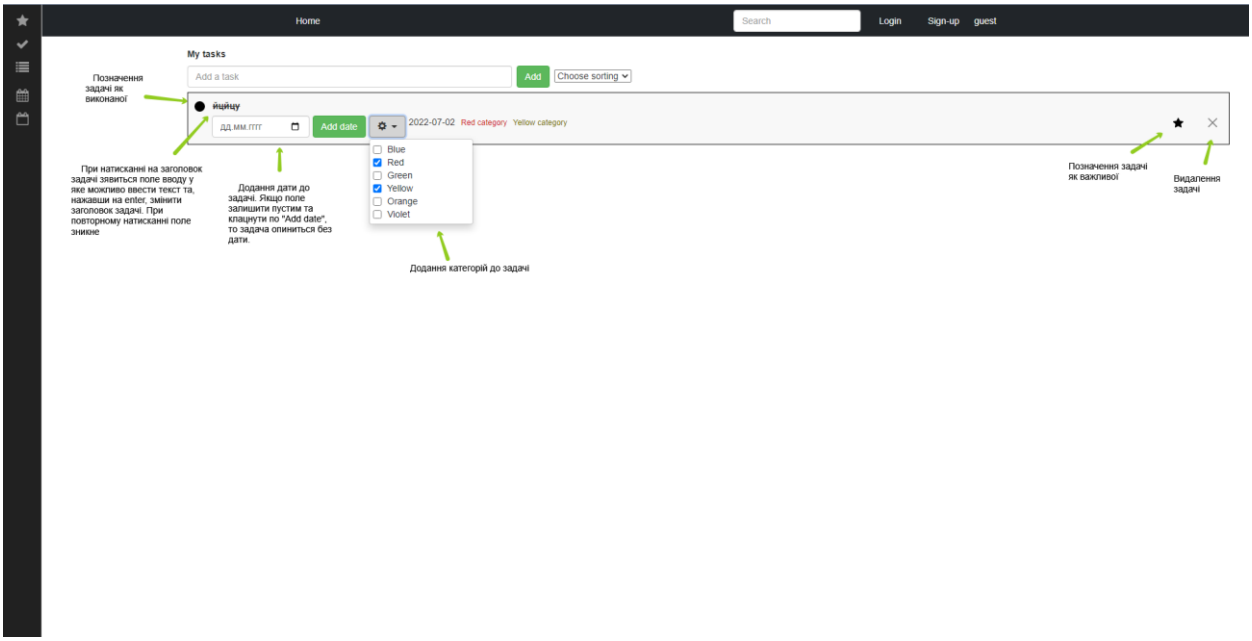


Рис. №4.3. Інструкція роботи із задачами.