Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система планування персональних завдань та навичок
(тема)

Виконав:
студент 4 курсу, групи ПЗПІп-22-2

Кривохатський Р.О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. Вечур О.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ доц. Дудар З.В.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кривохатському Роману Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система планування персональних завдань та навичок

Затверджена наказом по університету від 17.06.2024р. № 588 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 21.07.2024

3. Вихідні дані до роботи Розробити програмний додаток з можливістю створювати завдання та навички для планування

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.05.2024	*виконано*
2	Створення специфікації ПЗ	10.06. 2024	*виконано*
3	Проектування ПЗ	12.06. 2024	*виконано*
4	Розробка ПЗ	28.06. 2024	*виконано*
5	Тестування ПЗ	30.06. 2024	*виконано*
6	Оформлення пояснювальної записки	15.07. 2024	*виконано*
7	Підготовка презентації та доповіді	17.07. 2024	*виконано*
8	Попередній захист	21.07. 2024	*виконано*
9	Нормоконтроль, рецензування	21.07. 2024	*виконано*
10	Здача роботи у електронний архів	19.07. 2024	*виконано*
11	Допуск до захисту у зав. кафедри	21.07. 2024	*виконано*

Дата видачі завдання 17 червня 2024р.

Студент (ка)_____ Кривохатський Р.О.
(підпис)

Керівник роботи _____ доц. Вечур О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 59 стор., 14 рис., 2 табл., 6 джерел.

ПЛАНУВАННЯ, ЗАВДАННЯ, НАВИЧКИ, JAVA.

Об’єкт розробки – програмна система планування персональних завдань та навичок.

Мета розробки – створення програмної системи планування персональних завдань та навичок.

Метод рішення – середовище розробки Intelij Idea, мови програмування Java.

Система дозволяє користувачам ефективно організовувати свої завдання, відстежувати прогрес у розвитку навичок та оптимізувати особисту продуктивність за допомогою інтуїтивно зрозумілого інтерфейсу та потужних функцій планування.

Explanatory note to the bachelor's thesis, 59 pages, 14 figures, 2 tables, 6 sources.

PLANNING, TASKS, SKILLS, JAVA.

The object of development is a software system for planning personal tasks and skills.

The purpose of the development is to create a software system for planning personal tasks and skills.

The solution method is the Intelij Idea development environment, the Java programming language.

The object of development is a software system for planning personal tasks and skills. The goal of the development is to create a software system for planning personal tasks and skills. The solution method involves using the IntelliJ IDEA development environment and the Java programming language. The system allows users to effectively organize their tasks, track progress in skill development, and optimize personal productivity through an intuitive interface and powerful planning features.

Я, Кривохатський Роман Олексійович, студент гр. ПЗПІп-22-2 здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система планування персональних завдань та навичок», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 7
1 Аналіз предметної галузі 9
1.1 Аналіз предметної галузі 9
1.2 Виявлення та вирішення проблем 11
1.3 Постановка задачі 15
2 Формування вимог до програмної системи 19
3 Архітектура та проєктування програмного забезпечення 23
3.1 UML проєктування ПЗ 23
3.2 Проєктування архітектури ПЗ 28
3.3 Проєктування структури зберігання даних 31
3.4 Рішення дизайну для програмного продукту 36
4 Опис прийнятих програмних рішень 38
5 Тестування програмного забезпечення 49
6 Впровадження програмного забезпечення 52
Висновки 53
Перелік джерел посилання 55
Додаток А 56
Додаток Б 57

ВСТУП

У сучасному динамічному світі більшість ділових людей мають величезну кількість задач та цілей, які потрібно ефективно планувати та виконувати. Для досягнення успіху необхідно постійно розвивати свої навички, що є непростим завданням. Щоб полегшити цей процес та підвищити продуктивність, потрібна багатофункціональна програма, яка дозволить користувачам вести облік своїх задач та навичок, планувати, фільтрувати, пов'язувати задачі з конкретними навичками, розставляти пріоритети та категоризувати їх.

Також, важливим є отримання статистичної інформації у вигляді звітів, наявність дошок або таблиць у додатку, де користувач зможе концентруватися на пріоритетних і важливих задачах, не забуваючи при цьому про рутинні завдання. Ефективне управління часом та підвищення продуктивності є актуальними проблемами, які потребують вирішення.

З цією метою розробляється програмна система, що дозволяє планувати повсякденні справи, підвищуючи таким чином продуктивність та забезпечуючи ефективне управління часом. Під час розробки використовується реляційна база даних та архітектура клієнт-сервер. Для реалізації доступу даних для веб-клієнта застосовується REST підхід, а доступ до даних СУБД виконується за допомогою JPA API та імплементації ORM Hibernate. Для розробки фронтенду використовується архітектура SPA (Single Page Application), реалізована за допомоги Angular Framework, TypeScript та BootStrap.

Отже, актуальність роботи полягає у створенні зручного інструменту для ефективного планування задач, управління навичками та підвищення продуктивності, що є вкрай важливим для ділових людей у сучасних умовах. Метою роботи є розробка багатофункціональної програмної системи для планування повсякденних справ, обліку задач та навичок, отримання статистичної інформації. Завданнями роботи є:

– Розробити систему для зручного планування задач та обліку навичок.

– Забезпечити можливість фільтрації, категоризації та встановлення пріоритетів задач.

– Реалізувати функціонал для отримання статистичної інформації у вигляді звітів та наочних дашбордів.

– Використати сучасні технології та підходи розробки, такі як REST, JPA, Hibernate, Angular Framework.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Ефективне планування завдань є невід'ємною складовою успішного виконання роботи та досягнення поставлених цілей. На сьогоднішній день існує широкий спектр інструментів та підходів до планування завдань, проте кожен з них має свої переваги та недоліки. Розглянемо детальніше деякі з найпоширеніших інструментів та методів планування завдань.

Можна виділити наступні інструменти планування завдань:

- списки завдань: Один з найпростіших та найпоширеніших інструментів для планування завдань. Списки завдань дозволяють зберігати та організовувати завдання у вигляді впорядкованого переліку, що полегшує їх відстеження та виконання;
- календарі: Календарі є корисним інструментом для планування завдань на певну дату або час. Вони дозволяють візуалізувати розклад та розподіляти завдання відповідно до часових рамок;
- органайзери: Органайзери поєднують функції списків завдань та календарів, забезпечуючи зручний спосіб зберігання, організації та нагадування про завдання;
- програмне забезпечення для управління проектами: Ці інструменти призначені для планування, відстеження та керування проектами, що включають множину взаємопов'язаних завдань, ресурсів та графіків.

Можна виділити наступні методи планування завдань:

- метод Помодоро: Цей метод передбачає розбиття робочого часу на короткі інтервали (зазвичай 25 хвилин), які чергуються з короткими

перервами. Такий підхід допомагає зосередитися та підвищити продуктивність;

- метод GTD (Getting Things Done): Метод GTD базується на принципі "розвантаження" розуму від завдань шляхом їх впорядкування та зосередження на найбільш важливих справах;
- метод Ivy Lee: Згідно з цим методом, наприкінці дня складається список з шести найважливіших завдань на наступний день, які виконуються у порядку пріоритетності;
- метод "Їж ту жабу": Цей метод передбачає виконання найскладнішого або найбільш неприємного завдання першим, що дозволяє уникнути прокрастинації та звільнити решту дня для інших справ.

Ведення особистого планування завдань є критично важливим для підтримання високої продуктивності та досягнення поставлених цілей. Сучасні ділові люди стикаються з величезною кількістю завдань та цілей, що вимагає постійного розвитку та вдосконалення навичок. Однак, ефективне управління задачами та навичками є непростим завданням, яке потребує використання відповідних інструментів та методів.

На сьогоднішній день існує ряд програмних рішень для планування завдань, проте більшість з них або є надто спрощеними, обмежуючись функціоналом списків завдань, або ж призначені виключно для управління проектами на рівні організацій. Прості додатки зі списками завдань мають обмежені можливості та не враховують потребу користувачів у контролі та плануванні власних навичок.

Таким чином, актуальною є розробка багатофункціонального додатку, орієнтованого на персональні потреби користувача в плануванні завдань, відстеженні цілей та розвитку навичок. Такий додаток повинен надавати можливості для ведення обліку завдань та навичок, їх планування, фільтрації,

встановлення зв'язків між завданнями та конкретними навичками, розстановки пріоритетів та категоризації.

1.2 Виявлення та вирішення проблем

В контексті програмних додатків для планування завдань виявлення та вирішення проблем набуває особливої специфіки. Основною проблемою багатьох існуючих рішень є їхня обмеженість функціоналом простих списків завдань, що не відповідає комплексним потребам сучасних користувачів. Такі додатки часто не враховують необхідність інтеграції з іншими аспектами життя та роботи користувача, що призводить до фрагментації інформації та зниження ефективності планування.

Інша поширена проблема полягає у складності інтерфейсу та надмірній функціональності деяких програмних рішень. Користувачі можуть відчувати перевантаження від великої кількості опцій та налаштувань, що paradoxically знижує продуктивність замість її підвищення. Це особливо актуально для корпоративних систем управління проектами, які часто виявляються занадто громіздкими для індивідуального використання.

Недостатня гнучкість у категоризації та пріоритизації завдань є ще однією проблемою багатьох додатків. Користувачі потребують можливості легко адаптувати систему категорій та пріоритетів під свої унікальні потреби, що не завжди передбачено в існуючих рішеннях. Це може призводити до неефективного розподілу часу та ресурсів, коли важливі завдання губляться серед менш пріоритетних.

Синхронізація даних між різними пристроями та платформами часто виявляється проблематичною. Незважаючи на поширеність хмарних технологій, багато додатків все ще стикаються з проблемами у забезпеченні безперебійного

доступу до актуальної інформації з різних пристроїв, що може призводити до втрати даних або роботи з неактуальною інформацією.

Інтеграція з іншими інструментами продуктивності, такими як календарі, електронна пошта або месенджери, часто реалізована недостатньо ефективно. Це змушує користувачів постійно перемикатися між різними додатками, що знижує загальну продуктивність та створює ризик пропустити важливу інформацію.

Багато додатків для планування завдань не надають достатньо розвинутих інструментів для аналітики та звітності. Користувачі потребують можливості аналізувати свою продуктивність, виявляти паттерни та тренди у виконанні завдань, але часто ці функції або відсутні, або реалізовані поверхнево.

Проблема надмірної складності у налаштуванні та початковому використанні додатків може відштовхувати потенційних користувачів. Багато людей відмовляються від використання корисних інструментів через високий поріг входу та необхідність витрачати значний час на освоєння всіх функцій.

Недостатня підтримка колаборативної роботи є суттєвим недоліком багатьох персональних додатків для планування завдань. В сучасному світі, де навіть індивідуальні проекти часто вимагають взаємодії з іншими, відсутність функцій спільного доступу та редагування завдань може значно обмежувати корисність додатку.

Проблема перевантаження користувацького інтерфейсу сповіщеннями та нагадуваннями також заслуговує на увагу. Надмірна кількість push-повідомлень може призводити до ігнорування важливої інформації або відволікання від поточних завдань, що знижує загальну ефективність роботи.

Недостатня адаптивність додатків до індивідуального стилю роботи користувача є ще одним важливим аспектом. Багато рішень пропонують фіксований набір методологій планування, не враховуючи, що різні люди можуть мати різні підходи до організації свого часу та завдань.

Для вирішення цих проблем розробники програмних додатків для планування завдань можуть застосовувати різноманітні підходи. Одним з ключових напрямків є розробка більш інтуїтивних та адаптивних інтерфейсів, які дозволяють користувачам легко налаштовувати додаток під свої потреби без необхідності витрачати багато часу на вивчення всіх функцій.

Впровадження систем машинного навчання та штучного інтелекту може значно покращити функціональність додатків. Ці технології можуть допомогти в автоматичній категоризації завдань, пропонуванні оптимальних часових слотів для виконання певних типів завдань, а також в аналізі продуктивності користувача та наданні персоналізованих рекомендацій щодо покращення ефективності роботи.

Покращення інтеграції з іншими популярними інструментами та платформами є критично важливим для створення єдиного інформаційного простору користувача. Це може включати глибшу інтеграцію з календарями, поштовими клієнтами, хмарними сховищами документів та інструментами для командної роботи.

Розробка більш гнучких систем категоризації та пріоритизації завдань дозволить користувачам створювати власні ієрархії та системи міток, що відповідають їхнім унікальним потребам. Це може включати можливість створення користувацьких полів для завдань, налаштування власних фільтрів та представлень даних.

Впровадження більш розвинутих інструментів для аналітики та візуалізації даних допоможе користувачам краще розуміти свої паттерни продуктивності та виявляти області для покращення. Це може включати інтерактивні графіки, теплові карти активності, аналіз трендів виконання завдань тощо.

Покращення механізмів синхронізації та забезпечення безперебійного доступу до даних з різних пристроїв є критично важливим для сучасних мобільних користувачів. Це вимагає не лише технічних рішень для ефективної

синхронізації, але й розробки інтуїтивних інтерфейсів, які забезпечують однаковий досвід користування на різних платформах.

Розробка більш ефективних систем сповіщень та нагадувань, які враховують контекст та пріоритетність завдань, допоможе уникнути інформаційного перевантаження користувачів. Це може включати використання алгоритмів, які аналізують патерни поведінки користувача та оптимізують частоту та зміст сповіщень.

Впровадження функцій для колаборативної роботи навіть у персональних додатках для планування завдань дозволить користувачам легко ділитися окремими завданнями або проектами з колегами чи партнерами без необхідності переходу на окрему платформу для управління проектами.

Розробка модульної архітектури додатків, яка дозволяє користувачам вибирати та активувати лише ті функції, які їм необхідні, може допомогти уникнути проблеми надмірної складності інтерфейсу. Це також дозволить поступово розширювати функціональність додатку в міру зростання потреб користувача.

Впровадження елементів гейміфікації та систем винагород може підвищити мотивацію користувачів до регулярного використання додатку та виконання запланованих завдань. Це може включати системи досягнень, візуалізацію прогресу, челленджі та інші елементи, які роблять процес планування та виконання завдань більш захоплюючим.

Забезпечення розширеної підтримки користувачів через вбудовані системи допомоги, навчальні матеріали та можливості персоналізованої підтримки може значно знизити поріг входу для нових користувачів та підвищити загальну задоволеність від використання додатку.

Розробка функцій для інтеграції з системами управління розвитком навичок та цілями може допомогти користувачам більш ефективно пов'язувати свої щоденні завдання з довгостроковими цілями та планами особистого

розвитку. Це може включати можливість відстеження прогресу у розвитку конкретних навичок через виконання пов'язаних завдань.

Впровадження систем автоматичного планування та оптимізації розкладу на основі введених користувачем параметрів та пріоритетів може значно полегшити процес розподілу завдань та ресурсів. Такі системи могли б враховувати енергетичні цикли користувача, доступність ресурсів, дедлайни та інші фактори для створення оптимального графіку роботи.

Розробка функцій для інтеграції з системами тайм-трекінгу дозволить користувачам більш точно оцінювати час, необхідний для виконання різних типів завдань, та відповідно покращувати свої навички планування. Це також може допомогти у виявленні часових витрат та оптимізації робочих процесів. індивідуальному стилю роботи та потребам.

1.3 Постановка задачі

Постановка задачі на проектування даного програмного забезпечення (ПЗ) "Task Planner" передбачає створення багатофункціонального веб-додатку для ефективного планування та управління завданнями, орієнтованого на персональні потреби користувачів та розвиток їхніх навичок.

Основна мета розробки – створити інтуїтивно зрозумілий та гнучкий інструмент, який дозволить користувачам ефективно організовувати свої завдання, відстежувати прогрес у їх виконанні, а також планувати та розвивати свої навички. Система повинна забезпечувати можливість роботи з множиною користувачів, кожен з яких матиме власний набір завдань та навичок.

Ключові функціональні вимоги до системи включають:

– управління користувачами, тобто можливість додавання нових користувачів з введенням основної інформації (ім'я, прізвище, вік, email,

телефон), система повинна підтримувати перегляд списку користувачів та можливість їх видалення;

- управління завданнями як функціонал для створення, редагування та видалення завдань, кожне завдання повинно мати такі атрибути як назва, опис, статус, пріоритет, дедлайн та прогрес виконання, а користувачі повинні мати можливість призначати завдання собі або іншим користувачам системи;
- управління навичками, таке як можливість додавання та категоризації навичок, де кожна навичка повинна мати назву, опис та категорію, а система повинна дозволяти користувачам зв'язувати завдання з конкретними навичками для відстеження прогресу в їх розвитку;
- фільтрація та пошук як реалізація функціоналу для пошуку завдань за ключовими словами, а також фільтрації за статусом та пріоритетом;
- звітність та аналітика, тобто генерація загальної статистики по кількості користувачів та завдань, індивідуальної статистики по кожному користувачу (кількість завдань та навичок), а також відображення топ-навичок та завдань з найближчими дедлайнами;
- експорт даних, а саме можливість вивантаження даних про завдання у форматах CSV та PDF;
- сповіщення як функціонал для надсилання сповіщень про завдання з наближенням дедлайнів;
- інтерфейс користувача як от розробка інтуїтивно зрозумілого та responsive веб-інтерфейсу, який забезпечує зручну навігацію та швидкий доступ до всіх функцій системи.

Нефункціональні вимоги до системи включають:

- продуктивність (система повинна забезпечувати швидку роботу навіть при великій кількості користувачів та завдань);

- масштабованість (архітектура системи повинна дозволяти легко розширювати функціонал та збільшувати навантаження);
- безпека (реалізація механізмів аутентифікації та авторизації для захисту персональних даних користувачів);
- надійність (система повинна стабільно працювати та мати механізми резервного копіювання даних);
- зручність використання (нтерфейс повинен бути інтуїтивно зрозумілим для користувачів різного рівня технічної підготовки);
- кросплатформенність(еб-додаток повинен коректно працювати в різних браузерах та на різних пристроях (десктоп, планшет, смартфон)).

Технічні вимоги до реалізації:

- використання сучасних веб-технологій для розробки frontend частини (наприклад, React, Angular або Vue.js);
- розробка backend частини з використанням надійних та масштабованих технологій (наприклад, Node.js з Express, Python з Django або Java з Spring).
- використання реляційної бази даних для зберігання даних (наприклад, PostgreSQL або MySQL);
- реалізація API для забезпечення взаємодії між frontend та backend частинами;
- використання системи контролю версій (наприклад, Git) для управління кодовою базою проекту;
- реалізація автоматизованого тестування для забезпечення якості кодової бази.

Процес розробки повинен включати наступні етапи:

- детальний аналіз вимог та створення технічного завдання.
- проектування архітектури системи та дизайну інтерфейсу.
- розробка прототипу основних функцій системи.

- ітеративна розробка з регулярним тестуванням та отриманням зворотного зв'язку від потенційних користувачів.
- тестування системи, включаючи функціональне, інтеграційне та навантажувальне тестування.
- розгортання системи на виробничому середовищі.
- підготовка документації для користувачів та адміністраторів системи.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Метою даної роботи є розробка багатофункціональної програмної системи для планування повсякденних завдань, обліку та розвитку навичок, а також отримання статистичної інформації про виконані завдання та прогрес у розвитку навичок. Система повинна вирішити низку важливих проблем, пов'язаних з ефективним плануванням завдань та управлінням навичками.

Основними вимогами до розроблюваної системи є:

а) інтеграція завдань та навичок:

1) можливість створювати та вести облік завдань;

2) можливість створювати та вести облік навичок користувача;

3) можливість встановлювати зв'язки між завданнями та відповідними навичками, необхідними для їх виконання;

4) відстеження прогресу у розвитку навичок через виконання пов'язаних завдань;

б) гнучкі можливості фільтрації та категоризації:

1) реалізація потужного функціоналу фільтрації завдань за різними критеріями, такими як пріоритет, термін виконання, пов'язані навички, категорії тощо;

2) можливість категоризації завдань та навичок за різними ознаками для зручної організації та управління;

в) Зручні візуальні представлення:

1) інтуїтивний та зрозумілий інтерфейс для відображення завдань, навичок та їх взаємозв'язків;

2) використання дошок, діаграм, таблиць та інших візуальних елементів для наочного представлення інформації;

3) можливість концентрації уваги на пріоритетних та важливих завданнях;

г) звітність та аналітика:

1) генерація різноманітних звітів про виконані завдання, прогрес у розвитку навичок, статистичні дані тощо;

2) надання аналітичної інформації для прийняття обґрунтованих рішень щодо планування завдань та розвитку навичок;

д) крос-платформність:

1) розробка веб-версії системи для доступу з різних пристроїв та операційних систем.

Для реалізації поставлених завдань необхідно використати сучасні технології та підходи до розробки програмного забезпечення, такі як:

- клієнт-серверна архітектура для забезпечення масштабованості та можливості роботи з різних пристроїв;
- REST API для реалізації доступу до даних для веб-клієнта.
- реляційна база даних для зберігання даних про завдання, навички та їх зв'язки;
- фреймворк Angular та TypeScript для розробки клієнтської частини з використанням підходу SPA (Single Page Application) ;
- фреймворк Spring Boot та мова Java для розробки серверної частини та забезпечення доступу до даних за допомогою JPA API та ORM Hibernate.

Розроблювана система повинна забезпечити ділових людей потужним інструментом для ефективного планування повсякденних завдань, обліку та розвитку навичок, а також надавати статистичну інформацію та аналітику для прийняття обґрунтованих рішень. Система має бути зручною у використанні, гнучкою та масштабованою, щоб задовольнити потреби різних категорій користувачів.

Функціональні вимоги до ПЗ зведемо в таблицю 2.1.

Таблиця 2.1 – Функціональні вимоги до ПЗ

Функціональна вимога	**Опис**
Управління завданнями	- Створення, редагування та видалення завдань. - Встановлення пріоритетів, термінів виконання та опису для завдань.
Управління навичками	- Створення, редагування та видалення навичок. - Можливість пов'язувати завдання з необхідними навичками.
Фільтрація та категоризація	- Фільтрація завдань за пріоритетом, терміном виконання, пов'язаними навичками та категоріями.- Категоризація завдань та навичок для зручної організації.
Візуалізація	- Відображення завдань у вигляді дошок або списків. - Візуальне представлення зв'язків між завданнями та навичками.
Звітність	- Генерація звітів про виконані завдання за певний період. - Відстеження прогресу розвитку навичок на основі виконаних завдань.
Нагадування	- Можливість встановлювати нагадування про терміни виконання завдань.
Синхронізація	- Синхронізація даних про завдання та навички між різними пристроями.
Персоналізація	- Налаштування інтерфейсу та параметрів відображення завдань та навичок.

Дана система для планування завдань та розвитку навичок повинна забезпечити користувачам зручний та ефективний інструмент для організації повсякденних справ, встановлення цілей та відстеження прогресу у їх виконанні.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML-проектування ПЗ

Окрім збереження та редагування інформації про завдання, планувальник повинен мати можливість створювати звіти для аналізу продуктивності користувача. Також важливо мати можливість надсилати повідомлення користувачам про нагадування про невиконане завдання або про закриття завдання. Враховуючи все вище сказане, можна розробити use-case діаграму, яка буде відображати всі сценарії роботи програми, що зображено на рисунку 3.1.

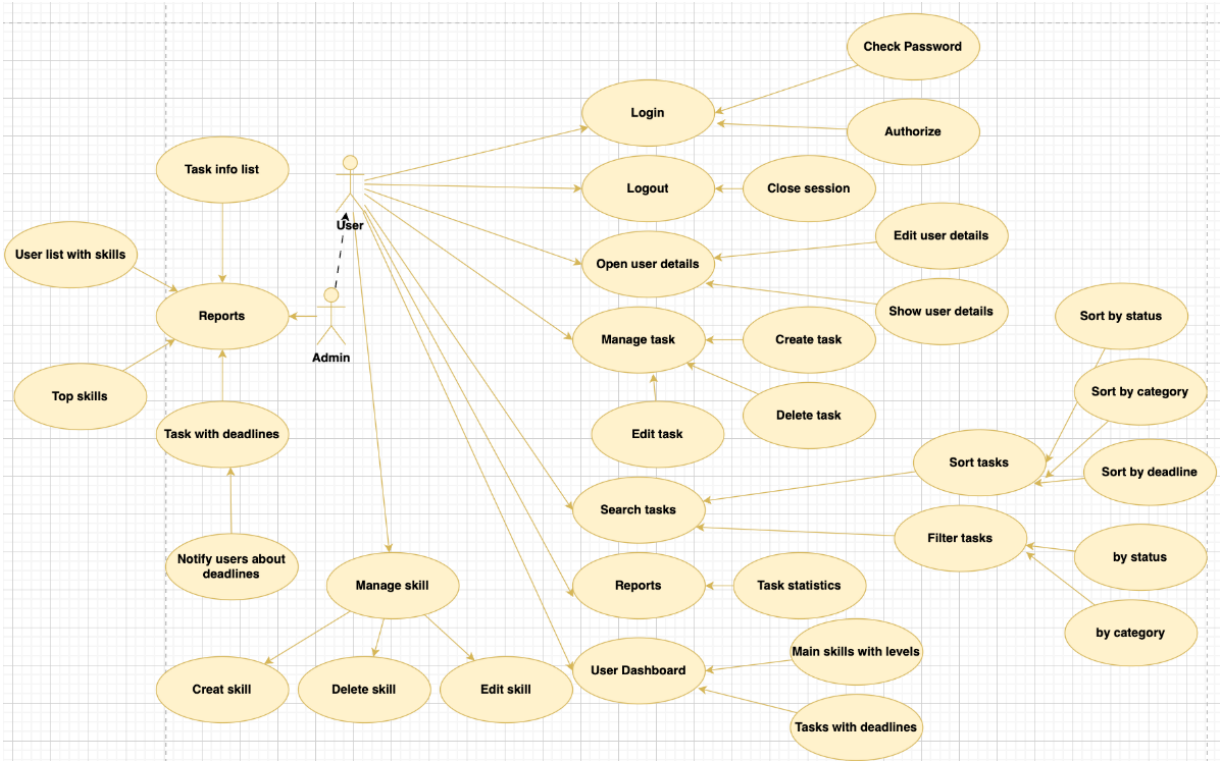


Рисунок 3.1 – Діаграма варіантів використання застосунку

Опис концептів програмної області на рівні основних класів та зв’язків між ними наведені у вигляді діаграми класів, що представлено на рисунку 3.2.

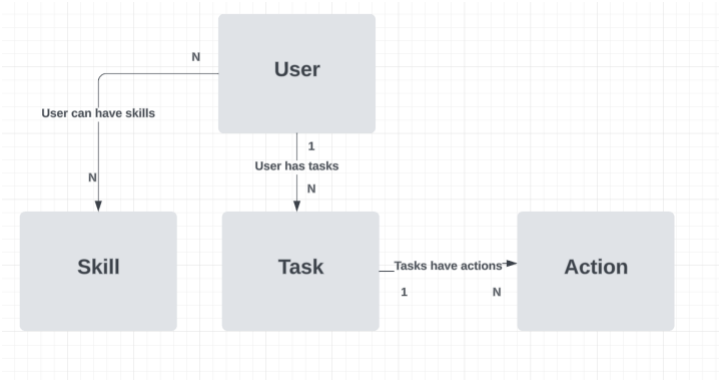


Рисунок 3.2 – Діаграма класів

Здійснимо проектування архітектури системи додатку. Діаграми активності (Activity Diagrams) – це один із видів діаграм, що використовуються в уніфікованій мові моделювання (UML) для візуального подання потоків керування в системі. Вони відображають послідовність дій або діяльностей, що відбуваються у процесі виконання певної операції або реалізації деякої функціональності [1].

Діаграми активності дозволяють змоделювати різні сценарії виконання процесів, включаючи паралельні потоки, альтернативні шляхи, точки прийняття рішень та розгалуження. Вони містять графічні елементи, такі як стани дій, переходи, розгалуження, злиття, вузли рішень, вузли потоків об'єктів тощо.

Основні елементи діаграм активності:

- початкова точка (Start Node) – вказує на початок потоку керування;
- дія (Action) – представляє певну задачу або крок, що виконується;
- контрольна точка (Control Flow) – визначає послідовність виконання дій;
- розгалуження (Decision Node) – точка, в якій потік керування розділяється на альтернативні шляхи залежно від умови;
- з'єднання (Merge Node) – точка, в якій альтернативні потоки керування об'єднуються;

– вузол потоку об'єктів (Object Flow Node) – представляє потік даних між діями;
– кінцева точка (End Node) – вказує на завершення потоку керування.

Діаграми активності використовуються для моделювання бізнес-процесів, потоків керування в програмному забезпеченні, процесів обробки даних, алгоритмів тощо. Вони допомагають візуалізувати та аналізувати логіку виконання операцій, виявляти можливі проблеми або вузькі місця, а також полегшують комунікацію між різними зацікавленими сторонами проекту [2].

Діаграми активності є значно кращим і прогресивнішим підходом до представлення алгоритмів та процесів у порівнянні зі звичайними текстовими або блок-схемами. Їхня перевага полягає у використанні стандартизованої нотації UML, яка забезпечує наочну візуалізацію потоку керування, послідовності дій, паралельних потоків, точок прийняття рішень, альтернативних шляхів та розгалужень. Це робить діаграми активності надзвичайно зручними для розуміння складних систем з одночасним виконанням декількох задач та логікою прийняття рішень. Діаграми активності дозволяють відображати не лише потік керування, а й потоки об'єктів або даних між діями, що допомагає краще зрозуміти взаємодію між різними компонентами системи та обмін даними. Нотація UML є розширюваною, що забезпечує гнучкість і можливість адаптації діаграм активності до специфічних потреб проекту. Існує безліч інструментів моделювання, які підтримують створення та редагування діаграм активності UML, полегшуючи співпрацю та обмін моделями. Це робить процес розробки та документування більш ефективним порівняно з використанням звичайних текстових або блок-схем алгоритмів.

На даній діаграмі активності зображено процес виконання завдань додатку планування завдань. Діаграма складається з двох основних частин: процес авторизації користувача та процес виконання завдань.

Процес авторизації користувача:

- спочатку користувач відкриває веб-сайт, що ініціює крок "Open site";
- після цього користувачу відображається сторінка входу ("Login Page"), на якій він вводить свої логін та пароль ("Enter login and password");
- введені дані надсилаються на сервер ("Server") для перевірки;
- на сервері відбувається крок "check password", де перевіряється коректність введеного паролю;
- якщо пароль невірний, користувачу відображаються відповідні повідомлення про помилки ("Errors");
- Якщо пароль вірний, користувач переходить на головну сторінку додатку ("Home page").

Процес виконання завдань:

- користувач ініціює виконання завдань ("run cron job");
- після цього відбувається старт процесу виконання завдань ("Job started");
- далі йде запит до бази даних для отримання списку завдань ("query DB for tasks");
- відбувається перевірка завдань у базі даних ("Check tasks in DB");
- потім відбувається крок "check result", де перевіряється, чи були знайдені відповідні завдання;
- якщо завдання були знайдені, генерується повідомлення ("Generate message") та надсилається електронний лист з звітом ("Email with report was sent");
- якщо завдань не знайдено, процес переходить до кроку "Complete job", що завершує виконання поточного процесу.

Таким чином, дана діаграма активності демонструє два основні процеси: авторизацію користувача та виконання завдань у додатку планування завдань. Процес авторизації включає перевірку паролю та надання доступу до головної

сторінки додатку, а процес виконання завдань включає запити до бази даних, генерацію звітів та надсилання електронних листів (див. рис. 3.3).

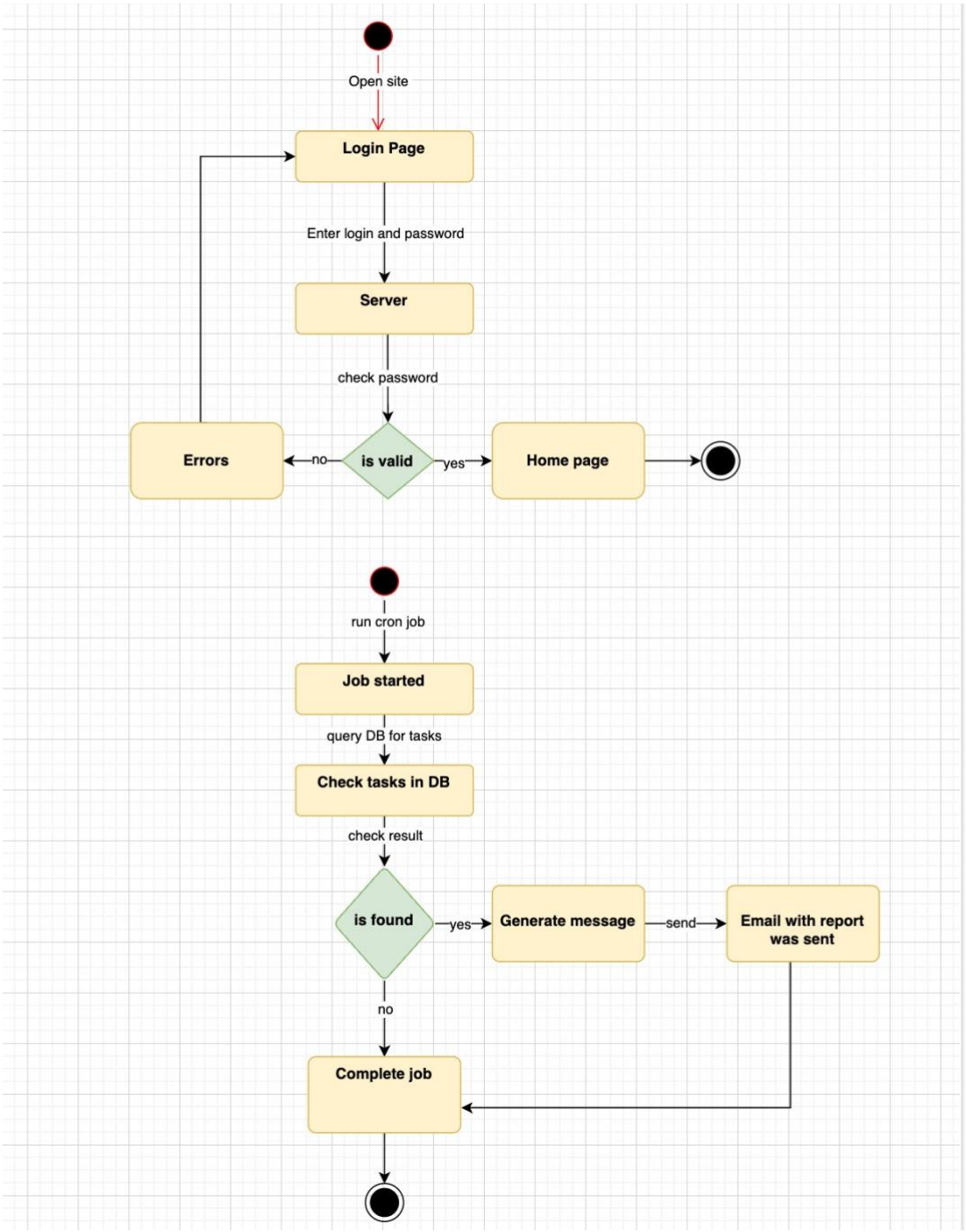


Рисунок 3.3 – Діаграма активності системи

3.2 Проектування архітектури ПЗ

Проектування архітектури програмного забезпечення для додатку Task Planner передбачає створення модульної структури, яка забезпечить гнучкість, масштабованість та легкість в обслуговуванні. Основними модулями ПЗ будуть:

Модуль управління користувачами відповідатиме за реєстрацію, аутентифікацію та авторизацію користувачів. Він забезпечуватиме функціонал створення нових облікових записів, редагування профілів користувачів, а також керування правами доступу. Цей модуль взаємодіятиме з модулем безпеки для забезпечення захисту персональних даних та з модулем зберігання даних для збереження інформації про користувачів.

Модуль управління завданнями є ключовим компонентом системи, що відповідає за створення, редагування, видалення та відстеження завдань. Він забезпечуватиме функціонал для встановлення пріоритетів, дедлайнів, статусів завдань та їх призначення користувачам. Цей модуль тісно взаємодіятиме з модулем управління користувачами для забезпечення персоналізованого доступу до завдань, а також з модулем аналітики для збору даних про виконання завдань.

Модуль управління навичками забезпечуватиме функціонал для створення, категоризації та відстеження розвитку навичок користувачів. Він дозволить зв'язувати завдання з конкретними навичками, що дасть можливість аналізувати прогрес у їх розвитку. Цей модуль взаємодіятиме з модулем управління завданнями для встановлення зв'язків між завданнями та навичками, а також з модулем аналітики для генерації звітів про розвиток навичок.

Модуль аналітики та звітності відповідатиме за збір, обробку та візуалізацію даних про виконання завдань, розвиток навичок та загальну активність користувачів. Він забезпечуватиме генерацію різноманітних звітів, графіків та дашбордів. Цей модуль взаємодіятиме з усіма іншими модулями системи для збору необхідних даних та надання аналітичної інформації.

Модуль сповіщень забезпечуватиме функціонал для створення та надсилання нагадувань про дедлайни завдань, зміни в статусах завдань та інші важливі події. Він взаємодіятиме з модулем управління завданнями для отримання інформації про терміни виконання та з модулем управління користувачами для визначення адресатів сповіщень.

Модуль пошуку та фільтрації надаватиме функціонал для швидкого пошуку завдань, навичок та користувачів за різними критеріями. Він забезпечуватиме ефективну фільтрацію даних для зручного відображення інформації. Цей модуль взаємодіятиме з модулями управління завданнями, навичками та користувачами для отримання необхідних даних.

Модуль експорту даних відповідатиме за генерацію файлів у форматах CSV та PDF, що міститимуть інформацію про завдання, навички та статистику. Він взаємодіятиме з модулем аналітики та звітності для отримання необхідних даних та їх форматування.

Модуль інтерфейсу користувача забезпечуватиме презентаційний рівень системи, надаючи зручний та інтуїтивно зрозумілий веб-інтерфейс для взаємодії з усіма функціями додатку. Цей модуль взаємодіятиме з усіма іншими модулями через API для отримання та відображення даних.

Модуль API забезпечуватиме комунікацію між frontend та backend частинами системи, надаючи стандартизований інтерфейс для обміну даними. Він взаємодіятиме з усіма іншими модулями системи, забезпечуючи передачу запитів та відповідей.

Модуль безпеки відповідатиме за захист даних користувачів, шифрування чутливої інформації, управління сесіями та забезпечення безпечного доступу до системи. Цей модуль взаємодіятиме з модулем управління користувачами для аутентифікації та авторизації, а також з усіма іншими модулями для забезпечення безпеки операцій.

Модуль зберігання даних забезпечуватиме взаємодію з базою даних, виконуючи операції читання, запису, оновлення та видалення даних. Він взаємодіятиме з усіма іншими модулями системи, забезпечуючи персистентність даних та їх ефективне зберігання.

Взаємодія між модулями відбуватиметься через чітко визначені інтерфейси, що забезпечить низьку зв'язаність компонентів системи. Модуль API виступатиме як центральний компонент, через який проходитимуть усі запити від клієнтської частини до серверної. Модуль безпеки забезпечуватиме контроль доступу до всіх операцій, перевіряючи права користувачів перед виконанням будь-яких дій.

Модуль управління користувачами тісно взаємодіятиме з модулем управління завданнями та навичками, забезпечуючи персоналізований доступ до функціоналу системи. Модуль аналітики та звітності отримуватиме дані від усіх інших модулів для генерації комплексних звітів та аналітичних даних.

Модуль сповіщень взаємодіятиме з модулем управління завданнями для відстеження дедлайнів та змін у статусах завдань, а також з модулем управління користувачами для визначення отримувачів сповіщень. Модуль пошуку та фільтрації забезпечуватиме швидкий доступ до даних через взаємодію з модулями управління завданнями, навичками та користувачами.

Загальна концепція ІС – це веб застосунок, в якому є клієнтська та серверні частини. Користувачі повинні мати десктопні чи мобільні веб браузери, за допомогою яких користувач має доступ до інтерфейсу. Загальну архітектуру системи можна побачити на контекстній діаграмі (див. рис. 3.4).

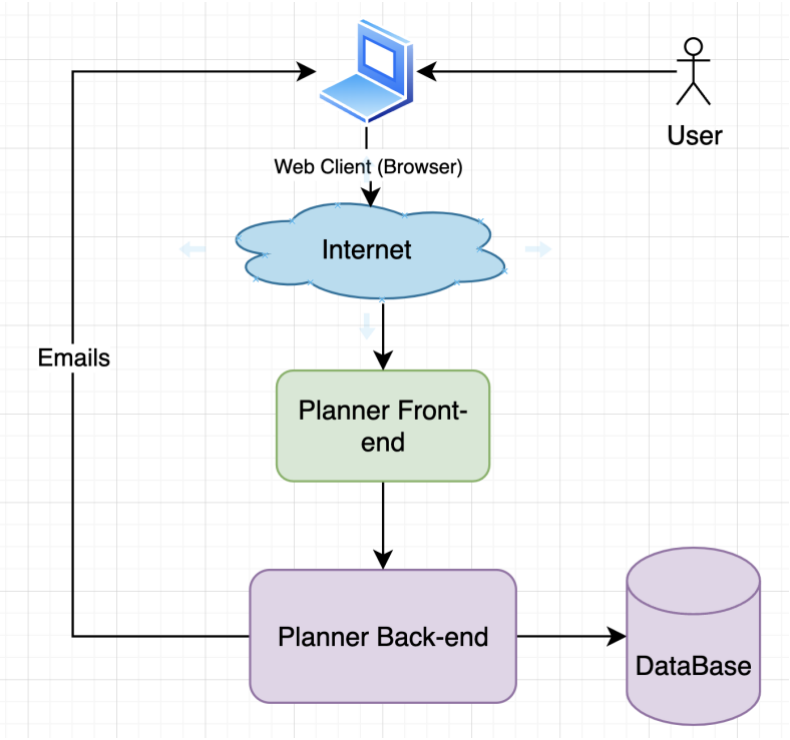


Рисунок 3.4 – Контекстна діаграма

ІС планування завдань повинна мати зручний та зрозумілий інтерфейс, який буде однаково працювати як в браузері, так і в мобільному девайсі. Основні операції повинні виконуватися швидко, а також зручно відображати актуально інформацію для юзера. Також система повинна надавати корисну інформацію та мати систему нотифікацій юзерів. Дані повинні зберігатися надійно і безпечно. Крім того, потрібна автентифікація та авторизація користувачів.

3.3 Проєктування структури зберігання даних

Науково-технічний прогрес нашого часу забезпечує можливість створення різноманітних програмних продуктів, зокрема планувальника завдань. Для ефективної розробки такого продукту необхідно мати чіткий концептуальний план, що включає такі моделі як завдання, навички, користувачі, категорії та дії. Основними вимогами є зберігання, редагування, оновлення, фільтрування та

сортування інформації. Також важливо мати можливість створювати звіти та надсилати повідомлення.

Для ефективної роботи планувальника завдань важливо мати можливість фільтрувати та сортувати інформацію за різними параметрами, такими як дата, статус, пріоритет та інші. Також необхідно мати можливість редагувати та оновлювати інформацію про завдання, навички, користувачів та категорії.

Для роботи з базою даних планувальника завдань використовуються операції CRUD (створення, читання, оновлення та видалення). Це дозволяє зберігати, зчитувати, оновлювати та видаляти інформацію з бази даних.

Концептуальна модель планувальника завдань має включати такі моделі, як завдання, навички, користувачі, категорії та дії. Модель завдання повинна містити інформацію про назву завдання, опис, дату початку та закінчення, статус та пріоритет. Модель навичок повинна містити інформацію про назву навички та опис. Модель користувача повинна містити інформацію про ім’я, прізвище, електронну пошту та пароль. Модель категорії повинна містити інформацію про назву категорії та опис. Модель дій повинна містити інформацію про назву дії та опис.

Була розроблена ER-діаграма за нотацією Баркера (див. рис. 3.5).

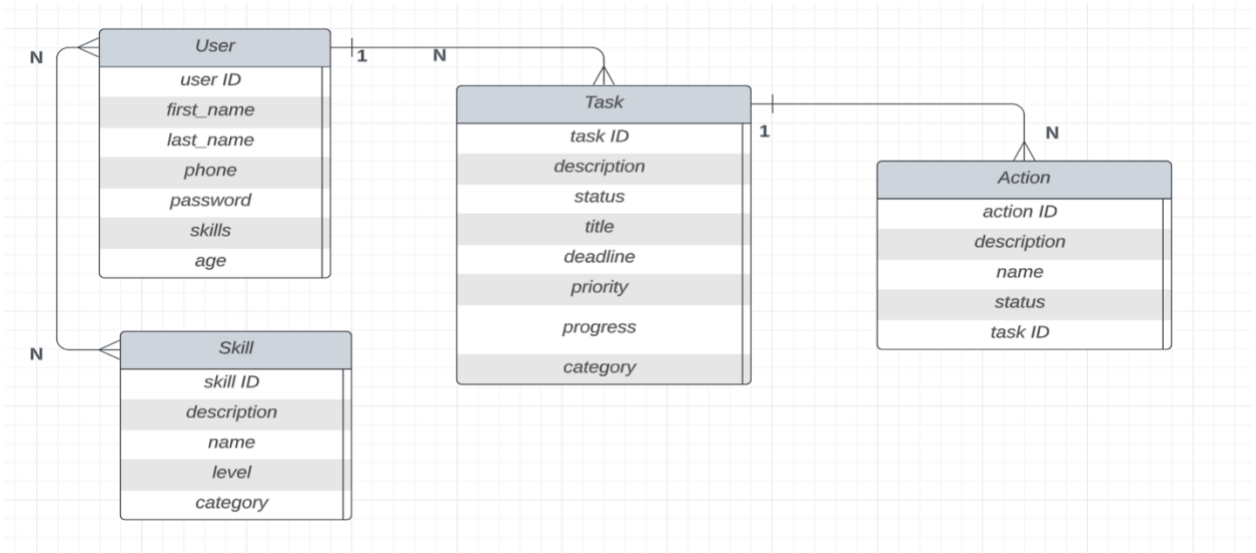


Рисунок 3.5 – ER-діаграма

Реляційна база даних – це база даних, яка використовується для зберігання та організації доступу до взаємопов’язаних елементів інформації. Реляційні бази даних ґрунтуються на реляційній моделі – інтуїтивно зрозумілому, наочному поданні інформації у вигляді таблиць. Кожен рядок у таблиці такої бази даних являє собою запис унікальним ідентифікатором – ключем [3]. Стовпці таблиць мають атрибути даних, що дає змогу встановлювати зв’язки між елементами даних. Саме цьому, реляційна модель бази даних є найбільш ефективною для вирішення задачі даного курсового проекту (див. рис. 3.6).

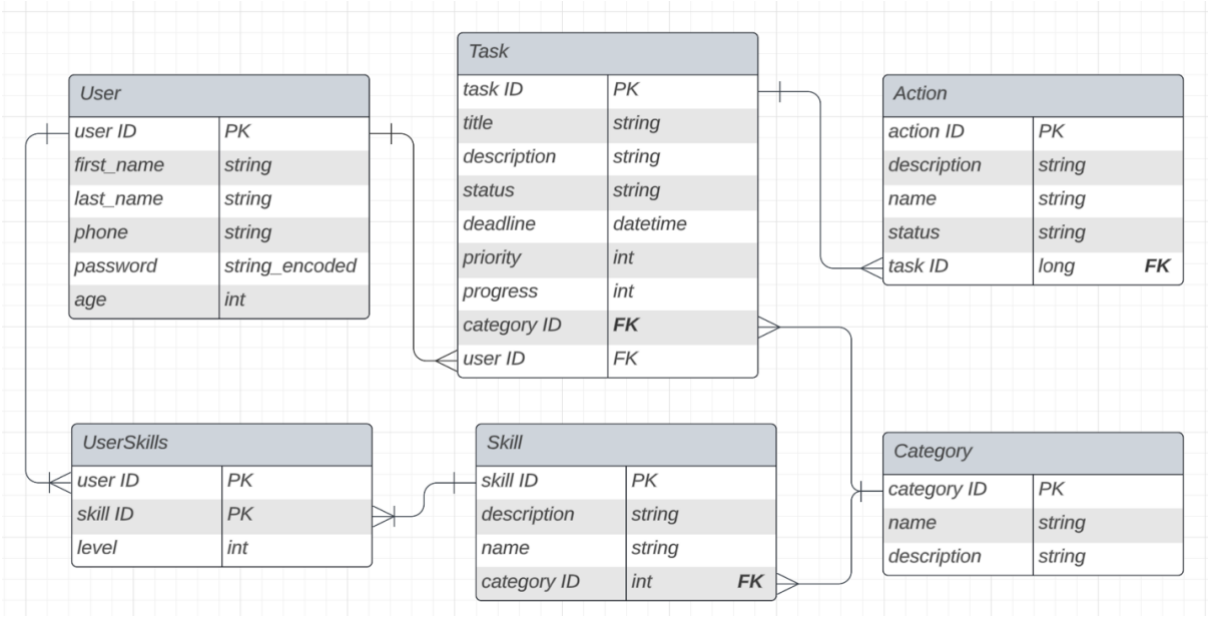


Рисунок 3.6 – Схема реляційної бази даних

Нормальна форма – це вимога, що висувається до структури таблиць в теорії реляційних баз даних для усунення з бази зайвих функціональних залежностей між атрибутами. У нашому проекті при нормалізації виконується приведення таблиць до третьої нормальної форми. Відношення знаходиться у третій нормальній формі, коли воно знаходиться у першій та другій нормальній формі, тобто всі атрибути атомарні, кожний не ключовий атрибут повністю функціонально залежить від первинного ключа відношення та між неключовими атрибутами немає транзитивних залежностей [4].

В усіх відношеннях всі атрибути є атомарними, тому вони знаходяться в першій нормальній формі. При зведенні до другої нормальної форми було доведено, що всі відношення не містять неповних функціональних залежностей, тобто в складі потенційного ключа відсутня менша підмножина атрибутів від якої можна також вивести дану функціональну залежність.

При зведенні до третьої нормальної форми була проведена перевірка на відсутність транзитивних функціональних залежностей.

Отже, отримана схема БД знаходиться у третій нормальній формі. Опишемо основні класи сутностей БД:

Класи сутностей (Entity classes):

а) user:

1) атрибути: userId (PK), firstName, lastName, phone, password, email, age;
2) відносини: один-до-багатьох з Task і UserSkills.

б) task:

1) атрибути: taskId (PK), title, description, status, deadline, priority, progress, categoryId (FK), userId (FK);
2) відносини: багато-до-одного з User і Category, один-до-багатьох з Action.

в) skill:

1) атрибути: skillId (PK), name, description, categoryId (FK);
2) відносини: багато-до-одного з Category, один-до-багатьох з UserSkills.

г) category:

1) атрибути: categoryId (PK), name, description;
2) відносини: один-до-багатьох з Skill і Task.

д) userSkills:

1) атрибути: userId (PK/FK), skillId (PK/FK), level;
2) відносини: багато-до-одного з User і Skill.

е) action:

1) атрибути: actionId (PK), description, name, status, taskId (FK);
2) відносини: багато-до-одного з Task.

Обґрунтування: ці класи відображають основні сутності системи, визначені в схемі бази даних. Вони забезпечують структуру для зберігання та управління даними про користувачів, завдання, навички, категорії та дії.

Класи репозиторіїв:

- UserRepository;
- TaskRepository;
- SkillRepository;
- CategoryRepository;
- UserSkillsRepository.

Обґрунтування: ці інтерфейси розширюють JpaRepository для кожної сутності, забезпечуючи стандартні операції CRUD та можливість створення користувацьких запитів. Вони є частиною шару доступу до даних у архітектурі Spring.

Головний клас застосунку PlannerApplication:

- анотації: @SpringBootApplication, @ComponentScan, @EnableWebSecurity, @EnableJpaRepositories, @EntityScan, @EnableScheduling;
- метод: main().

Обґрунтування: цей клас є точкою входу в додаток Spring Boot. Він налаштовує середовище виконання, сканує компоненти, активує безпеку та JPA репозиторії.

Взаємозв'язки між класами:

- User -> Task: композиція (один-до-багатьох);
- User -> UserSkills: композиція (один-до-багатьох);
- Skill -> UserSkills: композиція (один-до-багатьох);

- Category -> Skill: композиція (один-до-багатьох);
- Category -> Task: асоціація (один-до-багатьох);
- Task -> Action: композиція (один-до-багатьох).

Обґрунтування: ці зв'язки відображають логічні відносини між сутностями. Наприклад, користувач може мати багато завдань, а завдання належить одному користувачеві.

Додаткові елементи:

- анотації JPA (@Entity, @Id, @GeneratedValue, @ManyToOne, @OneToMany) для класів сутностей;
- методи доступу (геттери і сеттери) для всіх атрибутів;
- конструктори для створення об'єктів.

Обґрунтування: ці елементи необхідні для правильної роботи JPA і забезпечення інкапсуляції даних.

3.4 Рішення дизайну для програмного продукту

Розробляючи дизайн для додатку Task Planner, важливо створити інтуїтивно зрозумілий та функціональний інтерфейс. Почнемо з головного вікна, де розмістимо панель навігації у верхній частині екрану. Ця панель матиме кнопки для переходу до основних розділів: "Головна", "Список користувачів", "Додати користувача", "Завдання", "Додати завдання" та "Додати навичку". Кнопки навігації будуть виконані у яскравому кольорі, щоб привертати увагу та полегшувати навігацію.

Для кольорової схеми пропоную використовувати спокійні, професійні тони. Основний фон буде білим для забезпечення чистоти та легкості сприйняття інформації. Акцентний колір, наприклад, бірюзовий, застосуємо для кнопок та

інших інтерактивних елементів. Це створить приємний контраст і допоможе користувачам швидко ідентифікувати активні елементи інтерфейсу.

У вікні додавання завдань розмістимо форму з полями для введення назви завдання, опису, вибору користувача, статусу, пріоритету та дедлайну. Важливо зробити ці поля достатньо великими для зручного введення інформації. Додамо випадаючі списки для вибору користувача та статусу, а також зручний календар для вибору дати дедлайну.

Для вікна зі списком завдань створимо таблицю з можливістю сортування та фільтрації. Над таблицею розмістимо поле пошуку та кнопки для експорту даних у різні формати. Кожен рядок таблиці буде містити основну інформацію про завдання з можливістю швидкого редагування статусу та пріоритету.

У розділі статистики важливо візуалізувати дані за допомогою графіків та діаграм. Це дозволить користувачам швидко оцінювати прогрес та продуктивність. Розмістимо тут загальну статистику, топ навичок та список найближчих дедлайнів.

Для забезпечення зручності використання на різних пристроях, застосуємо адаптивний дизайн. Елементи інтерфейсу будуть автоматично підлаштовуватися під розмір екрану, зберігаючи функціональність та естетичний вигляд.

Шрифт обереться читабельний, без зарубок, щоб забезпечити легкість сприйняття інформації на різних пристроях. Розмір шрифту буде достатньо великим для комфортного читання, але не настільки, щоб займати занадто багато місця на екрані.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

Основна частина коду додатку демонструє архітектурний підхід до побудови серверної частини веб-додатку для планування персональних завдань та навичок, використовуючи принципи REST та фреймворк Spring. Архітектура додатку базується на концепції розділення відповідальності, де контролери відповідають за обробку HTTP-запитів, а репозиторії забезпечують взаємодію з базою даних.

Центральним елементом архітектури є REST-контролери, зокрема CategoryController та SkillsController, які обробляють HTTP-запити, пов'язані з категоріями та навичками відповідно. Ці контролери реалізують основні операції CRUD (Create, Read, Update, Delete) для відповідних сутностей, забезпечуючи RESTful API для клієнтської частини додатку [5].

Система використовує об'єктно-реляційне відображення (ORM) через Spring Data JPA, що дозволяє абстрагувати роботу з базою даних та оперувати об'єктами Java замість прямих SQL-запитів. Це видно з використання репозиторіїв, таких як CategoryRepository, SkillRepository, UserRepository та UserSkillsRepository, які розширюють базові інтерфейси Spring Data [6].

Архітектура додатку передбачає чітке розмежування між шаром контролерів (web-пакет) та шаром доступу до даних (persistance-пакет). Така структура сприяє модульності та полегшує подальше розширення функціоналу системи.

Важливим аспектом архітектури є використання принципу впровадження залежностей (Dependency Injection), що реалізується через конструктори контролерів. Це забезпечує слабку зв'язність між компонентами системи та підвищує її тестованість.

Система оперує кількома ключовими сутностями: Category, Skill, User та UserSkills. Ці сутності формують основу доменної моделі додатку, відображаючи взаємозв'язки між різними аспектами системи планування завдань та навичок.

Обробка помилок реалізована через користувацькі виключення, такі як SkillNotFoundException, що дозволяє гнучко керувати нестандартними ситуаціями та повертати відповідні HTTP-статуси (див.рис. 4.1).

```
@RestController
public class SkillsController {
    private final SkillRepository repository;
    private final UserSkillsRepository userSkillsRepository;
    private final UserRepository userRepository;

    SkillsController(SkillRepository repository,
UserRepository userRepository, UserSkillsRepository
userSkillsRepository) {
        this.userRepository = userRepository;
        this.repository = repository;
        this.userSkillsRepository = userSkillsRepository;
    }

    @GetMapping("/Skills")
    Iterable<Skill> all() {
        return repository.findAll();
    }

    @PostMapping("/Skills")
    Skill newSkill(@RequestBody Skill newSkill) {
        return repository.save(newSkill);
    }

    @GetMapping("/Skills/{id}")
    Skill one(@PathVariable Long id) throws
SkillNotFoundException {

        return repository.findById(id)
                .orElseThrow(() -> new
SkillNotFoundException(id));
    }

    @GetMapping("/Skills/user/{userId}")
    Iterable<Skill> oneByUser(@PathVariable Long userId){
        Optional<User> user = userRepository.findById(userId);

        Set<UserSkills> byUser =
userSkillsRepository.findByUser(user.get());

        List<Skill> skills = byUser.stream().map(userSkills ->
userSkills.getSkill()).collect(Collectors.toList());

        return skills;

    }
```

Рисунок 4.1 – Фрагмент основного коду додатку (Скілс Контролер)

Представлений нижче фрагмент коду є реалізацією контролера завдань (TaskController) в рамках програмної системи планування персональних завдань та навичок. Цей контролер відіграє ключову роль у обробці HTTP-запитів, пов'язаних з управлінням завданнями.

Контролер використовує анотацію @RestController, що вказує на його призначення для обробки REST-запитів. Архітектурно, він слідує принципам REST, надаючи endpoint'и для виконання CRUD-операцій над завданнями, а також для різноманітних статистичних та пошукових запитів.

TaskController взаємодіє з кількома репозиторіями (TaskRepository, UserRepository, CategoryRepository, SearchTaskRepository) для доступу до даних, що зберігаються в базі даних. Це демонструє використання шаблону Repository для абстрагування логіки доступу до даних.

Контролер надає широкий спектр функціональності. Він дозволяє отримувати всі завдання, створювати нові, оновлювати та видаляти існуючі. Крім того, він надає можливості для сортування завдань, фільтрації за різними критеріями, пошуку за ключовими словами, та отримання різноманітної статистики.

Особливу увагу приділено генерації звітів. Контролер містить методи для створення CSV та PDF звітів, які можуть бути завантажені користувачем. Це реалізовано за допомогою класів PdfGenerator та стандартних засобів Java для роботи з файлами.

Важливим аспектом функціональності є інтеграція з email-сервісом. Контролер містить метод для відправки статистики по email, що реалізовано через TaskEmailSender.

Обробка помилок реалізована через користувацьке виключення TaskNotFoundException, що дозволяє елегантно обробляти ситуації, коли запитуване завдання не знайдено.

Контролер також демонструє використання параметризованих запитів, наприклад, для отримання завдань конкретного користувача або для фільтрації завдань за різними критеріями (див.рис. 4.2).

```
  @GetMapping("/Tasks/user/{userId}")
  Iterable<Task> oneByUser(@PathVariable Long userId)  {
      Optional<User> user = userRepository.findById(userId);
      return repository.findByUser(user.get());
  }

  @GetMapping("/Tasks/stat/{userId}")
  Iterable<UserTaskData> userTasks(@PathVariable Long
erId) {
      Optional<User> user = userRepository.findById(userId);
      return repository.tasksByUser(user.get());
  }

  @GetMapping("/Tasks/stat")
  Iterable<UserTaskData> tasksByUsers() {
      return repository.tasksByUser();
  }

  @GetMapping("/Tasks/stat/deadline")
  Iterable<Task> tasksByDeadline()  {
      return repository.tasksByDeadline(new Date());
  }

  @GetMapping("/Tasks/stat/week")
  List<WeekTaskStat> tasksByWeek() {
      return taskEmailSender.getWeekTaskStats();
  }


  @GetMapping("/Tasks/stat/top")
  Iterable<SkillsData> topTasks()  {
      Set<SkillsData> skillsData = repository.topTasks();
      List<SkillsData> result = new ArrayList<>();
      Iterator<SkillsData> iterator = skillsData.iterator();
      for (int i=0;i<3; i++) {
          if (iterator.hasNext()) {
              result.add(iterator.next());
          }
      }
      return result;
  }
```

Рисунок 4.2 – Фрагмент коду контролеру завдань (Таск контролер)

Наступна частина коду реалізує UsersController, який є ключовим компонентом в системі управління користувачами в рамках програмної системи планування персональних завдань та навичок. Цей контролер використовує анотацію @RestController, що вказує на його призначення для обробки REST-запитів, пов'язаних з користувачами.

Контролер забезпечує повний набір CRUD (Create, Read, Update, Delete) операцій для сутності User. Він взаємодіє з UserRepository для доступу до даних користувачів, SkillRepository для роботи з навичками, та використовує EntityManager для низькорівневих операцій з базою даних, демонструючи багаторівневий підхід до роботи з даними.

Важливою особливістю контролера є використання анотації @CrossOrigin, яка дозволяє запити з будь-якого джерела, що є критичним для взаємодії з фронтенд-частиною додатку, особливо якщо вона розгорнута на іншому домені.

UsersController надає ендпоінт для отримання списку всіх користувачів з можливістю сортування за вказаним полем та напрямком. Це реалізовано за допомогою методу all(), який використовує Spring Data JPA для динамічного створення запиту сортування.

Контролер також надає можливість отримати кількість користувачів у системі через метод count(), що демонструє використання агрегаційних функцій на рівні репозиторію.

Створення нового користувача реалізовано через метод newUser(), який приймає об'єкт User у тілі запиту та зберігає його в базі даних. Отримання інформації про конкретного користувача за його ідентифікатором реалізовано в методі one(), який також демонструє обробку виняткових ситуацій, викидаючи UserNotFoundException, якщо користувача не знайдено.

Оновлення даних користувача реалізовано в методі replaceUser(), який демонструє гнучкий підхід до оновлення: якщо користувач з вказаним

ідентифікатором існує, його дані оновлюються, якщо ні – створюється новий користувач з вказаним ідентифікатором.

Видалення користувача реалізовано через метод deleteUser(), який просто видаляє користувача за вказаним ідентифікатором (див.рис. 4.3).

```
@CrossOrigin(origins = "*", allowedHeaders = "*")
@RestController
public class UsersController {

    private final UserRepository repository;

    private final SkillRepository skillRepository;
    private final EntityManager entityManager;
    public UsersController(UserRepository repository,
SkillRepository skillRepository, EntityManager entityManager)
{
        this.repository = repository;
        this.skillRepository = skillRepository;
        this.entityManager = entityManager;
    }

    @CrossOrigin(origins = "*", allowedHeaders = "*")
    @GetMapping("/users/{sortField}/{direction}")
    Iterable<User> all(@PathVariable String sortField,
@PathVariable String direction) {
        return
repository.findAll(Sort.by(Sort.Direction.fromOptionalString(d
irection).get(), sortField));
    }

    @GetMapping("/users/count")
    Integer count() {
        return repository.users();
    }

    @PostMapping("/users")
    User newUser(@RequestBody User newUser) {
        return repository.save(newUser);
    }

    @GetMapping("/users/{id}")
    User one(@PathVariable Long id) throws
UserNotFoundException {

        return repository.findById(id)
                .orElseThrow(() -> new
UserNotFoundException(id));
    }
```

Рисунок 4.3 – Реалізація Юзер контролеру

Одна з основних форм розробленого додатку містить поля для введення назви завдання, опису, вибору користувача, статусу, пріоритету та встановлення дедлайну. Це дозволяє користувачам створювати нові завдання з детальною інформацією (див.рис. 4.4).

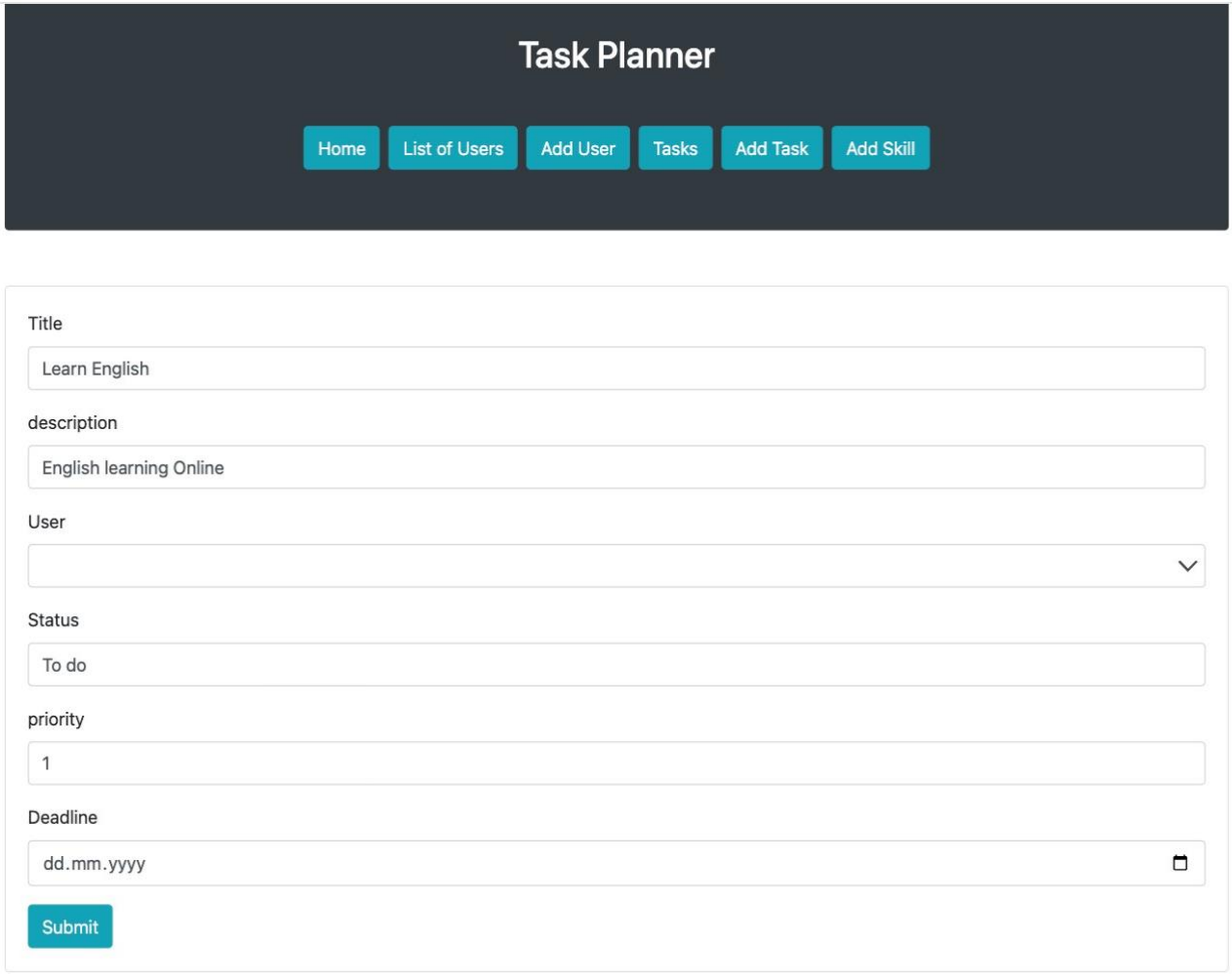


Рисунок 4.4 – Форма додавання нового завдання

Наступне вікно додатку відображає форму додавання нового користувача та таблицю з переліком навичок. Форма містить поля для введення особистої інформації користувача, такої як ім'я, прізвище, вік, телефон та email. Нижче розташована таблиця навичок з можливістю додавання нових (див.рис. 4.5).

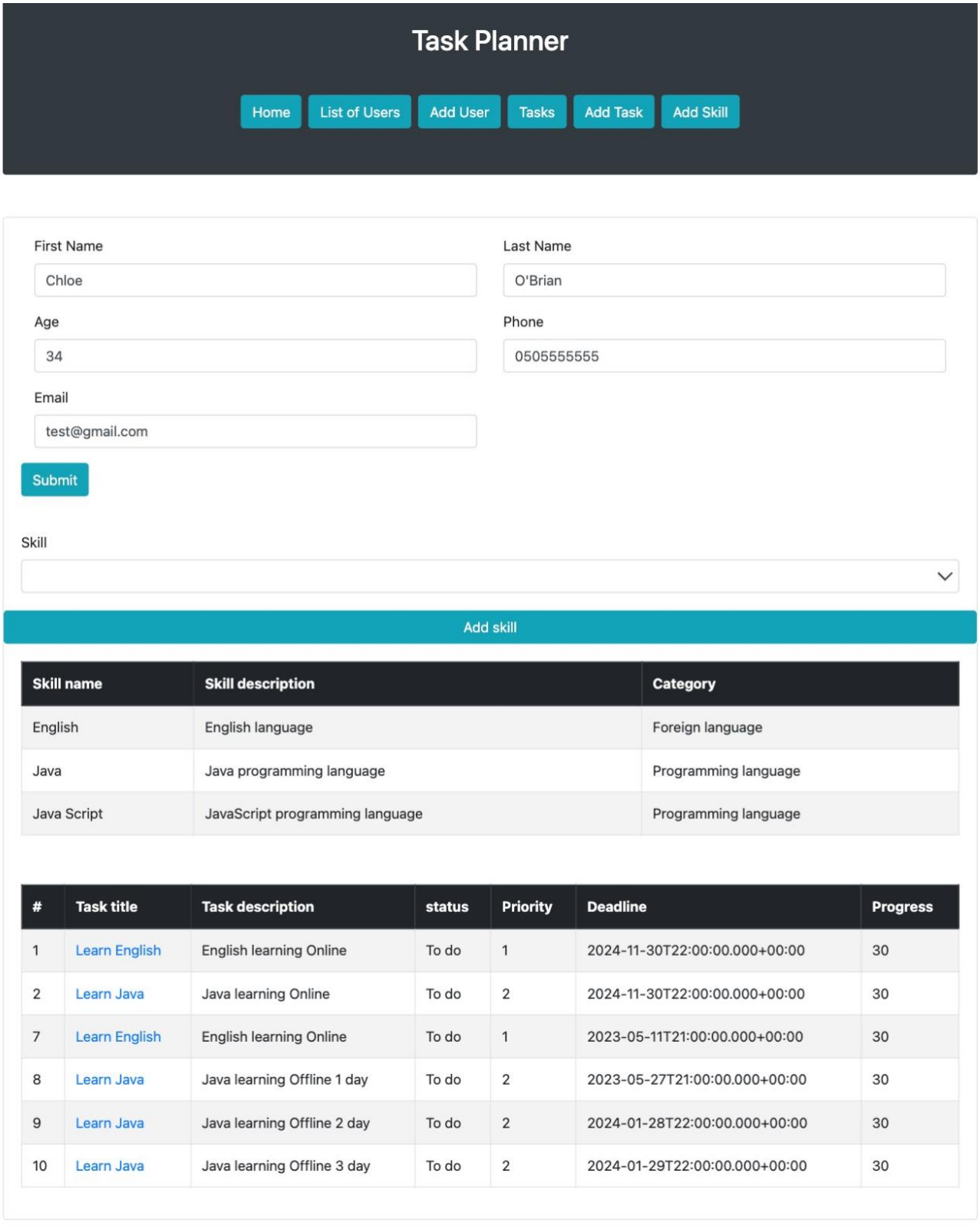
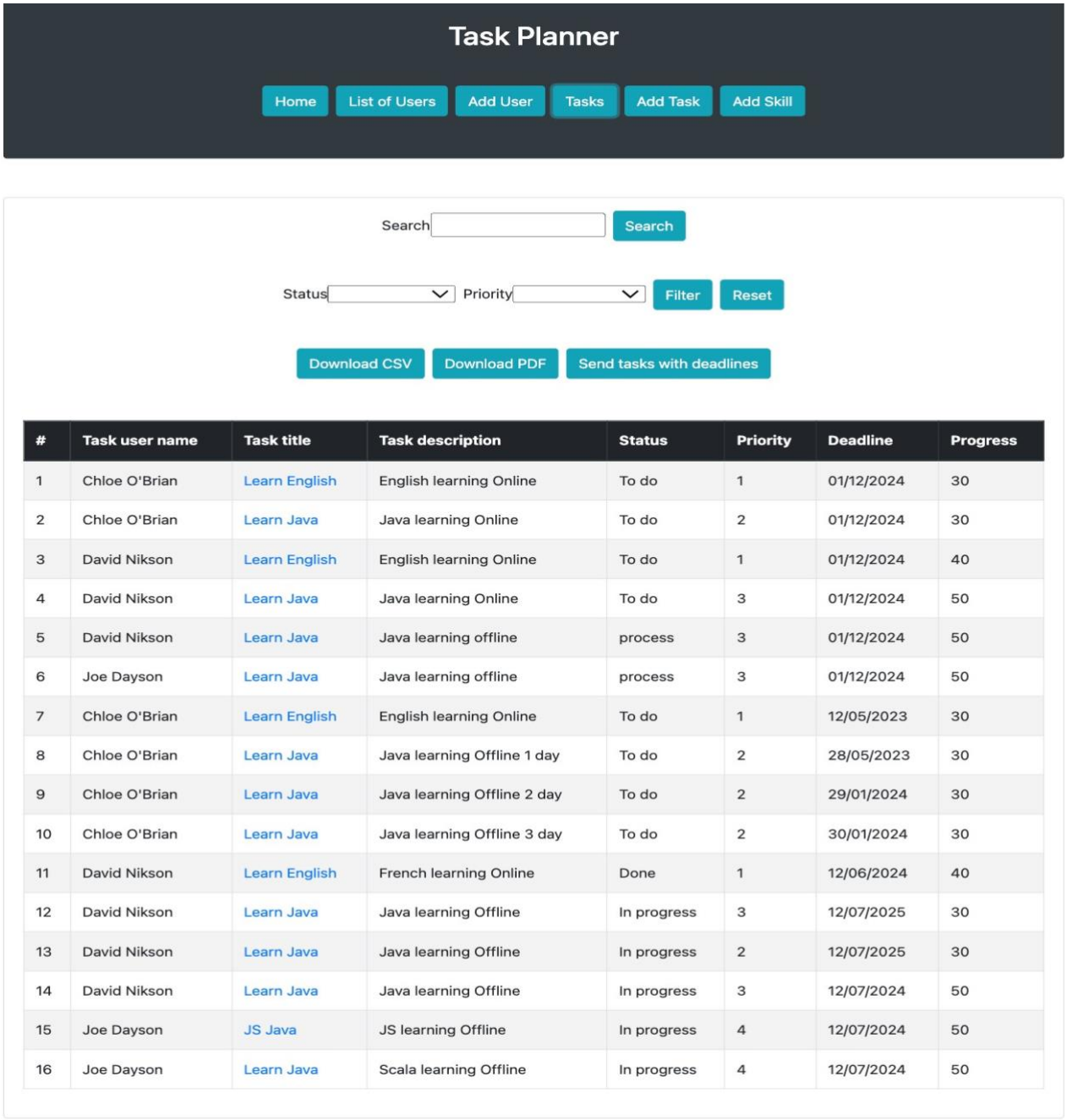


Рисунок 4.5 – Форма додавання нового користувача

Важливим є вікно, що показує основний інтерфейс управління завданнями. Він включає функції пошуку, фільтрації за статусом та пріоритетом, а також можливості експорту даних у CSV та PDF формати. Таблиця завдань містить детальну інформацію про кожне завдання, включаючи ім'я користувача, назву, опис, статус, пріоритет, дедлайн та прогрес виконання (див.рис. 4.6).



#	Task user name	Task title	Task description	Status	Priority	Deadline	Progress
1	Chloe O'Brian	Learn English	English learning Online	To do	1	01/12/2024	30
2	Chloe O'Brian	Learn Java	Java learning Online	To do	2	01/12/2024	30
3	David Nikson	Learn English	English learning Online	To do	1	01/12/2024	40
4	David Nikson	Learn Java	Java learning Online	To do	3	01/12/2024	50
5	David Nikson	Learn Java	Java learning offline	process	3	01/12/2024	50
6	Joe Dayson	Learn Java	Java learning offline	process	3	01/12/2024	50
7	Chloe O'Brian	Learn English	English learning Online	To do	1	12/05/2023	30
8	Chloe O'Brian	Learn Java	Java learning Offline 1 day	To do	2	28/05/2023	30
9	Chloe O'Brian	Learn Java	Java learning Offline 2 day	To do	2	29/01/2024	30
10	Chloe O'Brian	Learn Java	Java learning Offline 3 day	To do	2	30/01/2024	30
11	David Nikson	Learn English	French learning Online	Done	1	12/06/2024	40
12	David Nikson	Learn Java	Java learning Offline	In progress	3	12/07/2025	30
13	David Nikson	Learn Java	Java learning Offline	In progress	2	12/07/2025	30
14	David Nikson	Learn Java	Java learning Offline	In progress	3	12/07/2024	50
15	Joe Dayson	JS Java	JS learning Offline	In progress	4	12/07/2024	50
16	Joe Dayson	Learn Java	Scala learning Offline	In progress	4	12/07/2024	50

Рисунок 4.6 – Інтерфейс управління завданнями

Наступний модуль демонструє загальну статистику системи. Він містить інформацію про загальну кількість користувачів та завдань, статистику завдань по користувачах, топ-3 найпопулярніших навички та перелік завдань з найближчими дедлайнами (див.рис. 4.7).

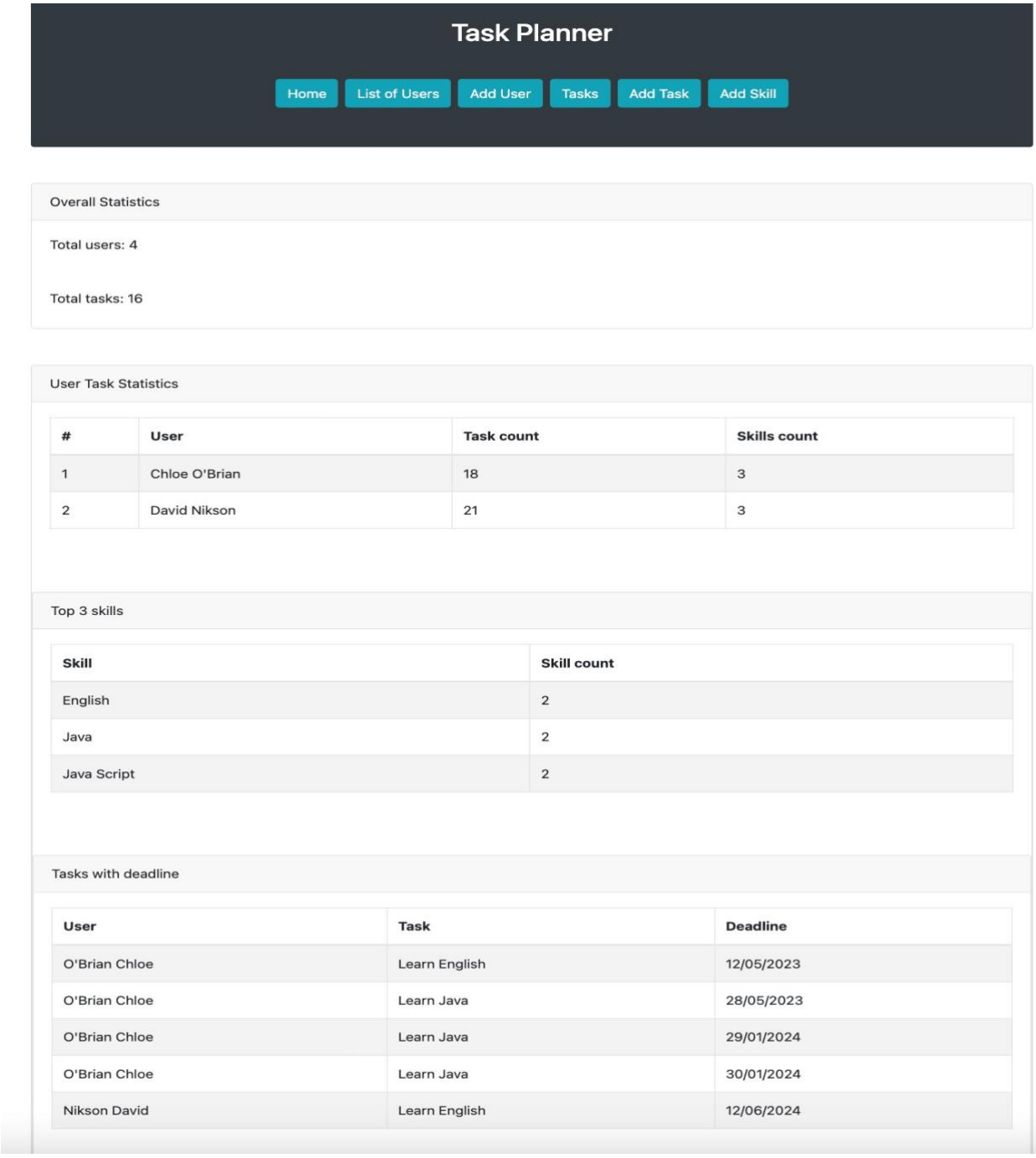


Рисунок 4.7 – Статистика навичок

Наступний скріншот відображає список користувачів системи у вигляді таблиці з їх основною інформацією та можливістю видалення кожного користувача (див.рис. 4.8).

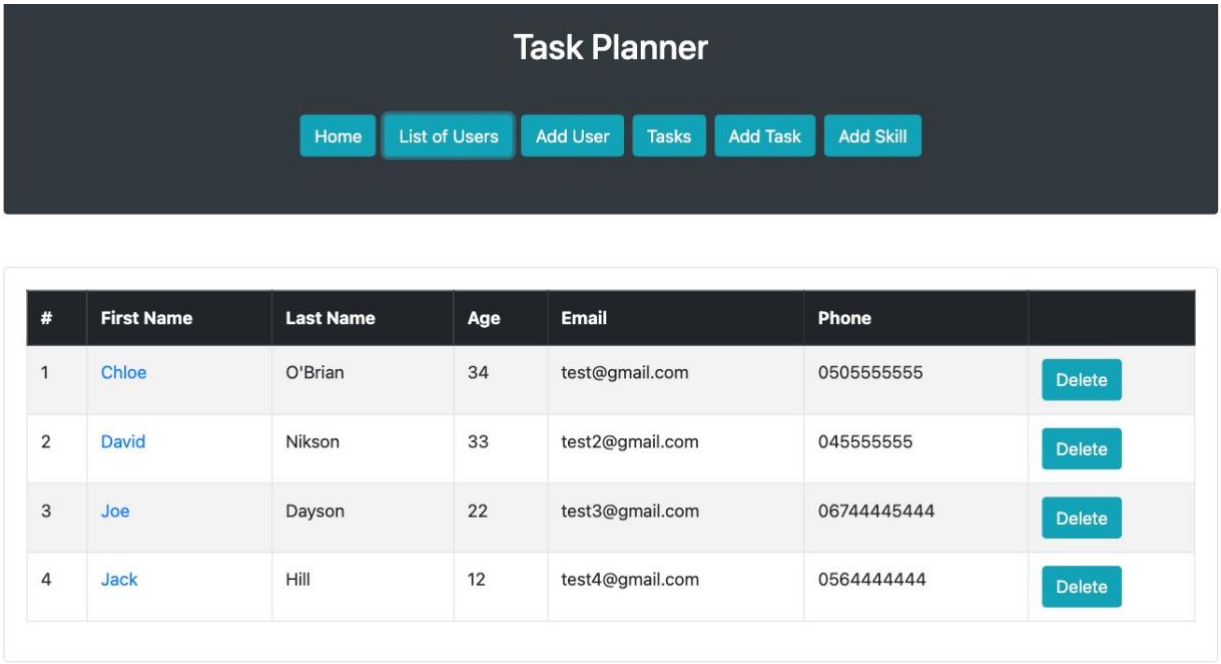


Рисунок 4.8 – Список користувачів системи

Додаток для планування завдань "Task Planner" демонструє кілька рівнів обробки виключень та валідації вводу, що спрямовані на покращення користувацького досвіду та забезпечення цілісності даних. На рівні інтерфейсу користувача додаток реалізує перевірку обов'язкових полів. Коли користувач залишає важливе поле порожнім, наприклад, заголовок завдання чи опис, система відображає повідомлення про помилку червоним кольором під відповідним полем. Ці повідомлення чітко вказують, яке саме поле потребує заповнення, наприклад "Заголовок обов'язковий" або "Опис обов'язковий". Окрім текстових повідомлень, додаток візуально підкреслює проблемні поля, підсвічуючи їх світло-червоним фоном. Це допомагає користувачам швидко ідентифікувати, які саме дані вони пропустили.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В ході тестування розробленого додатку було проведено наступні види тестування.

Перевірка функціональності: Перевірити коректність роботи всіх пунктів меню (Головна, Список користувачів, Додати користувача, Завдання, Додати завдання, Додати навичку). Протестувати додавання нових користувачів з валідними та невалідними даними. Спробувати додати користувачів, що дублюються. Додати завдання та призначити їх користувачам. Перевірити додавання навичок та їх зв'язок з користувачами. Перевірити функції фільтрації та сортування завдань. Протестувати функціональність пошуку. Підтвердити правильність завантаження CSV та PDF файлів. Перевірити функцію сповіщення електронною поштою про завдання з крайніми термінами.

Валідація даних: Протестувати поля введення з валідними та невалідними типами даних (наприклад, текст у числових полях). Спробувати відправити форми з відсутніми обов'язковими полями. Перевірити поля дат з минулими, поточними та майбутніми датами. Перевірити коректну обробку спеціальних символів у текстових полях.

Інтерфейс користувача: Перевірити адаптивність на різних розмірах екранів. Перевірити правильне вирівнювання та стилізацію всіх елементів інтерфейсу. Протестувати навігацію за допомогою клавіатури. Забезпечити відображення належних повідомлень про помилки. Перевірити наявність повідомлень про підтвердження після успішних дій.

Продуктивність: Протестувати додаток з великою кількістю користувачів та завдань. Виміряти час завантаження сторінок. Перевірити поведінку системи при одночасному доступі багатьох користувачів.

Безпека: Спробувати несанкціонований доступ до захищених сторінок. Перевірити поля введення на вразливості до SQL-ін'єкцій. Перевірити належне шифрування конфіденційних даних, таких як паролі [7].

Сумісність: Протестувати на різних браузерах (Chrome, Firefox, Safari, Edge). Перевірити функціональність на різних операційних системах.

Інтеграція: Перевірити синхронізацію даних між різними модулями (користувачі, завдання, навички). Перевірити точність розрахованої статистики.

Зручність використання: Оцінити легкість додавання та керування завданнями. Перевірити інтуїтивність макету інтерфейсу та навігації. Оцінити чіткість повідомлень про помилки та інструкцій.

Основні тест-кейси зведемо в таблицю 5.1.

Таблиця 5.1 – Основні тест-кейси при тестуванні розробленого ПЗ

D тест-кейсу	**Опис**	**Кроки тестування**	**Очікуваний результат**	**Фактичний результат**	**Статус**
ДК001	Додати користувача з валідними даними	1. Натиснути "Додати користувача" 2. Заповнити всі поля валідними даними 3. Натиснути "Підтвердити"	Користувача успішно додано	Успішно	Проведено
ДК002	Додати користувача з відсутнім	1. Натиснути "Додати користувача"	Відображено повідомл	Успішно	Проведено

	обов'язковим полем	2. Залишити поле "Ім'я" порожнім 3. Заповнити інші поля 4. Натиснути "Підтвердити"	ення про помилку		
ДК003	Додати користувача з невалідним форматом email	1. Натиснути "Додати користувача" 2. Ввести невалідний email (напр. "test@test") 3. Заповнити інші поля 4. Натиснути "Підтвердити"	Відображено повідомлення про невалідний email	Успішно	Протестовано
ДК004	Додати користувача з існуючим email	1. Натиснути "Додати користувача" 2. Ввести email існуючого користувача 3. Заповнити інші поля 4. Натиснути "Підтвердити"	Відображено повідомлення про дублювання email	Успішно	Протестовано

Таким чином, було повністю підтверджено коректність роботи розробленого додатку.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розроблене програмне забезпечення "Планувальник завдань" може бути впроваджено в різноманітних організаціях та компаніях для ефективного управління завданнями та проєктами. Дане ПЗ особливо корисне для команд, які працюють над різними проєктами одночасно.

У корпоративному середовищі це ПЗ може використовуватися менеджерами проєктів для розподілу завдань між членами команди, відстеження прогресу та встановлення пріоритетів. HR-відділи можуть застосовувати його для планування навчання співробітників та відстеження розвитку їхніх навичок.

В освітніх закладах викладачі можуть використовувати цей інструмент для призначення завдань студентам та моніторингу їхнього прогресу. Студенти, в свою чергу, можуть застосовувати його для організації свого навчального процесу та управління часом.

Фрілансери та індивідуальні підприємці можуть використовувати це ПЗ для планування своїх проєктів, встановлення дедлайнів та відстеження прогресу роботи з різними клієнтами.

Некомерційні організації можуть застосовувати цей інструмент для координації волонтерської діяльності, розподілу завдань між волонтерами та відстеження виконання різних ініціатив.

Загалом, це ПЗ може бути корисним для будь-якої організації чи особи, яка потребує ефективного інструменту для управління завданнями, проєктами та розвитком навичок.

ВИСНОВКИ

У сучасному динамічному світі більшість ділових людей стикаються з великою кількістю задач та цілей, які потребують ефективного планування та виконання. Досягнення успіху вимагає постійного розвитку навичок, що є складним завданням. Для полегшення цього процесу та підвищення продуктивності необхідна багатофункціональна програма, яка дозволить користувачам вести облік своїх задач та навичок, планувати, фільтрувати, пов'язувати задачі з конкретними навичками, розставляти пріоритети та категоризувати їх.

Важливим аспектом є можливість отримання статистичної інформації у вигляді звітів, а також наявність в додатку дошок або таблиць, де користувач може зосереджуватися на пріоритетних і важливих задачах, не забуваючи про рутинні завдання. Ефективне управління часом та підвищення продуктивності залишаються актуальними проблемами, що потребують вирішення. З цією метою розробляється програмна система, яка дозволяє планувати повсякденні справи, підвищуючи продуктивність та забезпечуючи ефективне управління часом.

Під час розробки використовується реляційна база даних та архітектура клієнт-сервер. Для реалізації доступу до даних для веб-клієнта застосовується REST підхід, а доступ до даних СУБД виконується за допомогою JPA API та імплементації ORM Hibernate. Для розробки фронтенду використовується архітектура SPA (Single Page Application), реалізована за допомогою Angular Framework, TypeScript та BootStrap.

Актуальність роботи полягає у створенні зручного інструменту для ефективного планування задач, управління навичками та підвищення продуктивності, що є вкрай важливим для ділових людей у сучасних умовах. Метою роботи є розробка багатофункціональної програмної системи для планування повсякденних справ, обліку задач та навичок, отримання

статистичної інформації. Завданнями роботи є розробка системи для зручного планування задач та обліку навичок, забезпечення можливості фільтрації, категоризації та встановлення пріоритетів задач, реалізація функціоналу для отримання статистичної інформації у вигляді звітів та наочних дашбордів, використання сучасних технологій та підходів розробки, таких як REST, JPA, Hibernate, Angular Framework.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Koc H., Erdogan A., Barjakly Y., Peker S. UML Diagrams in Software Engineering Research: A Systematic Literature Review // Proceedings. 2021. № 74. С. 13-23. DOI: 10.3390/proceedings2021074013.

2. Bhavik B., Muskaan N. An Overview of Structural UML Diagrams // International Research Journal of Engineering and Technology. 2021. № 8. С. 1577-1583.

3. Suster I., Ranisavljevic T. Optimization of MySQL database // Management and New Technologies. 2023. № 11. С. 141-151.

4. Методичні вказівки до практичних занять з дисципліни «Бази даних » для студентів усіх форм навчання спеціальності 6.050103 – «Програмна інженерія» / Упоряд.: О.О.Мазурова, М.С.Широкопетлєва, Ю.Ю.Черепанова. Харків: ХНУРЕ, 2022. 44 с.

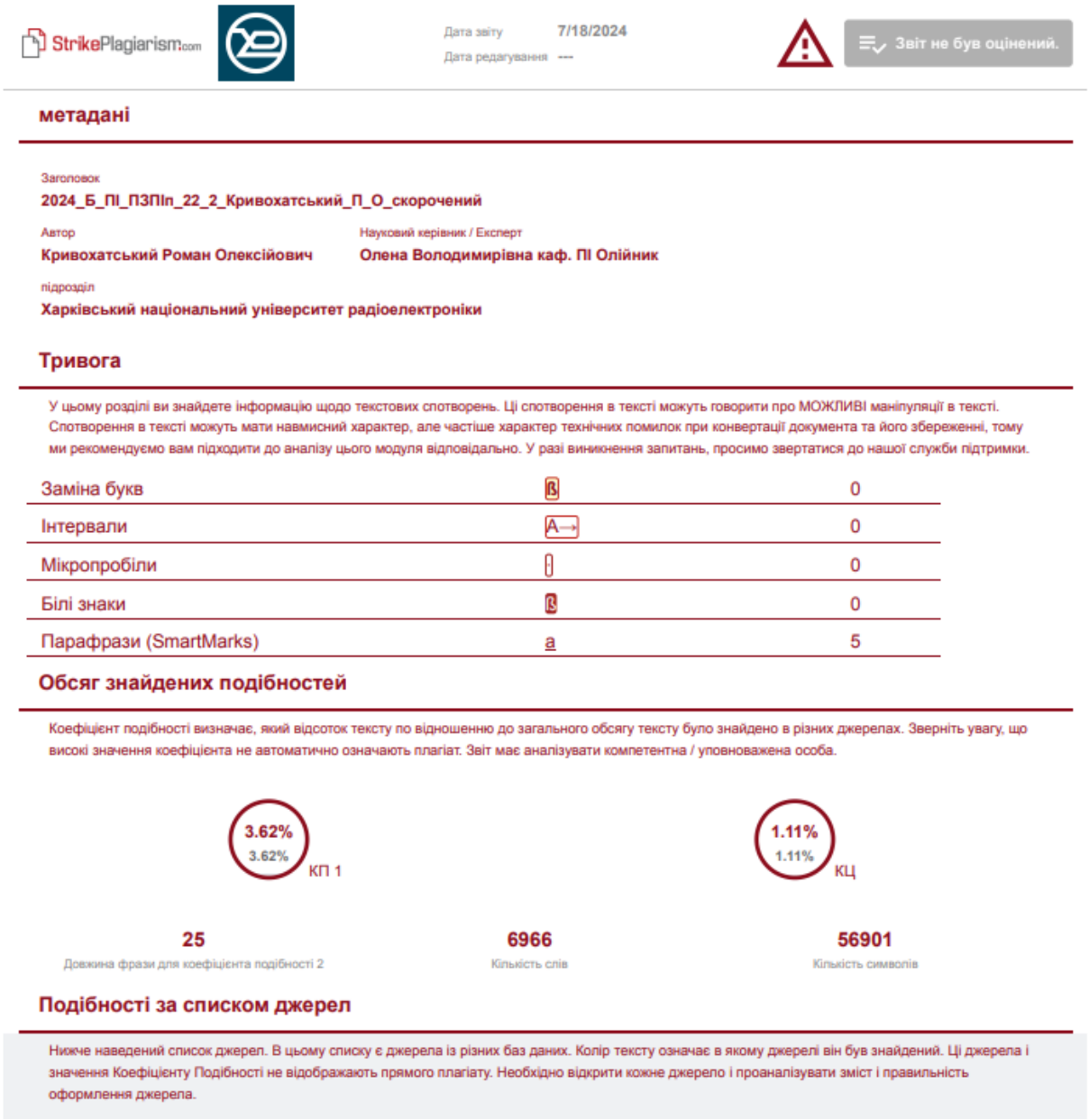
5. Leonard Richardson, Mike Amundsen, Sam Ruby. RESTful Web APIs: Services for a Changing World, 2013. С.34-37.

6. Spring Framework // URL: https://spring.io/ (дата звернення: 17.07.2024).

7. Gayathri Mohan. Full Stack Testing: A Practical Guide for Delivering High Quality Software 2022. С.120-140.

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 7/18/2024
Дата редагування ---

Звіт не був оцінений.

метадані

Заголовок
2024_Б_ПІ_ПЗПІп_22_2_Кривохатський_П_О_скорочений

Автор
Кривохатський Роман Олексійович

Науковий керівник / Експерт
Олена Володимирівна каф. ПІ Олійник

підрозділ
Харківський національний університет радіоелектроніки

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	0
Інтервали	A→	0
Мікропробіли	¶	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	5

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

3.62% 3.62% КП 1

1.11% 1.11% КЦ

25
Довжина фрази для коефіцієнта подібності 2

6966
Кількість слів

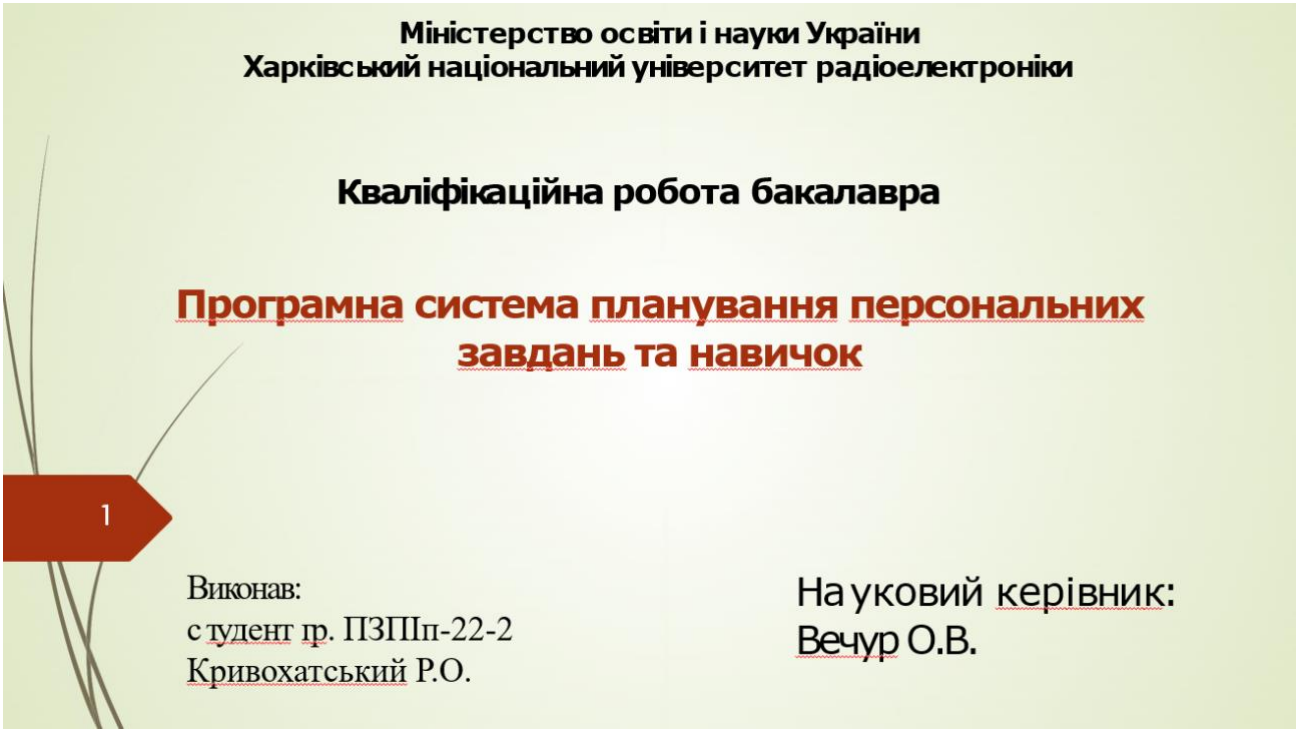
56901
Кількість символів

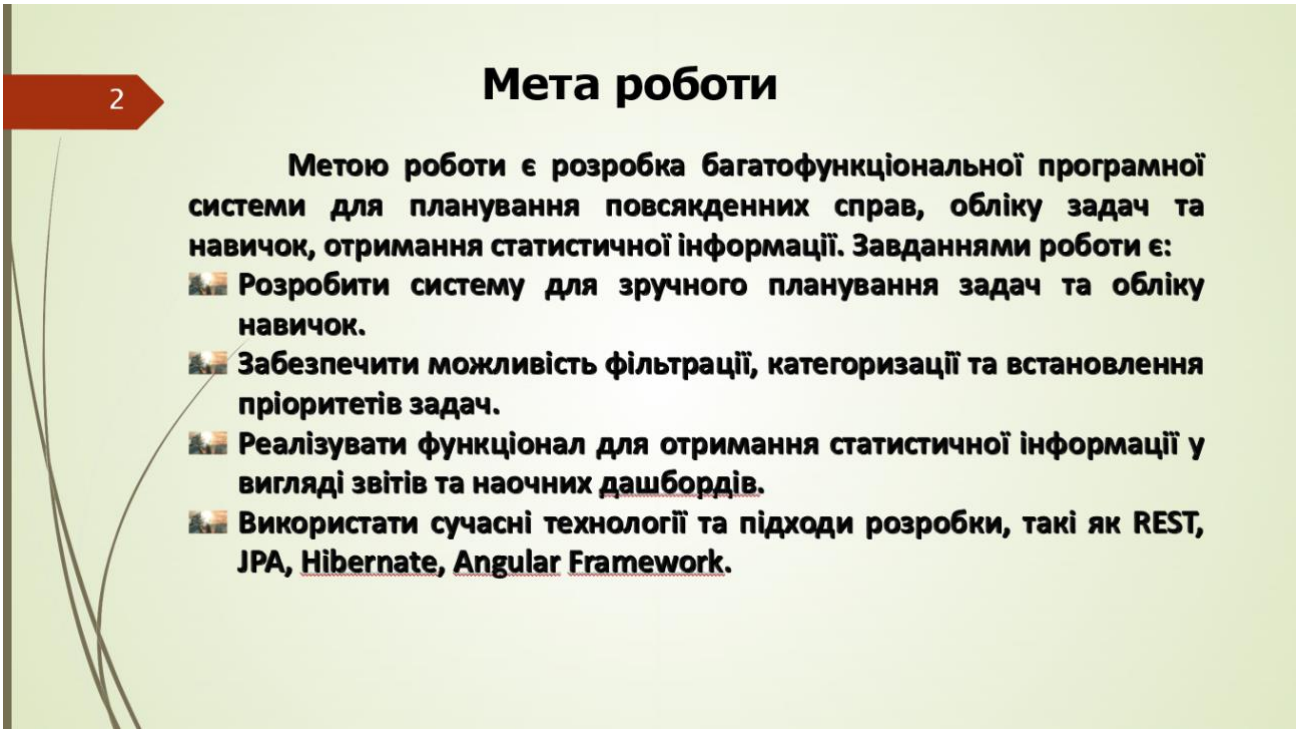
Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

ДОДАТОК Б

Слайди презентації



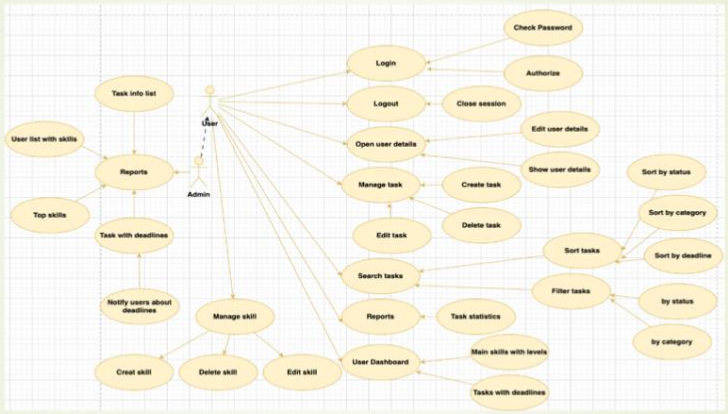


Список функціональних вимог до ПЗ

Функціональна вимога	Опис
Управління завданнями	- Створення, редагування та видалення завдань. - Встановлення пріоритетів, термінів виконання та опису для завдань.
Управління навичками	- Створення, редагування та видалення навичок. - Можливість пов'язувати завдання з необхідними навичками.
Фільтрація та категоризація	- Фільтрація завдань за пріоритетом, терміном виконання, пов'язаними навичками та категоріями.- Категоризація завдань та навичок для зручної організації.
Візуалізація	- Відображення завдань у вигляді дошок або списків. - Візуальне представлення зв'язків між завданнями та навичками.
Звітність	- Генерація звітів про виконані завдання за певний період. - Відстеження прогресу розвитку навичок на основі виконаних завдань.
Нагадування	- Можливість встановлювати нагадування про терміни виконання завдань.
Синхронізація	- Синхронізація даних про завдання та навички між різними пристроями.
Персоналізація	- Налаштування інтерфейсу та параметрів відображення завдань та навичок.

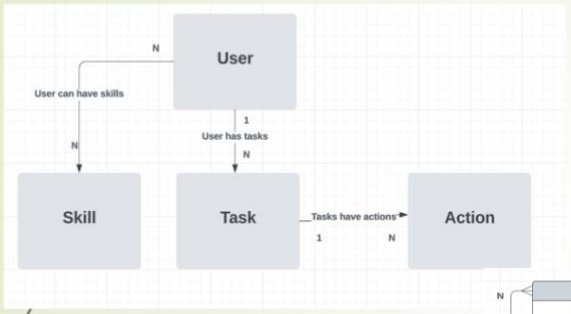
5

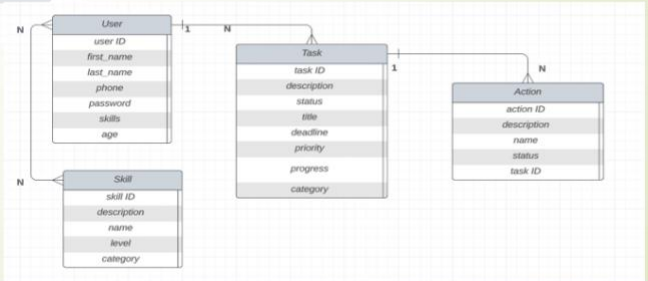
Діаграма прецедентів



6

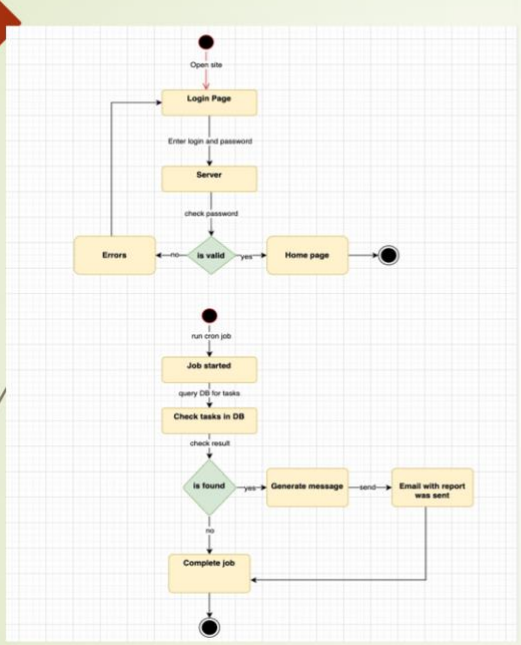
Діаграма класів

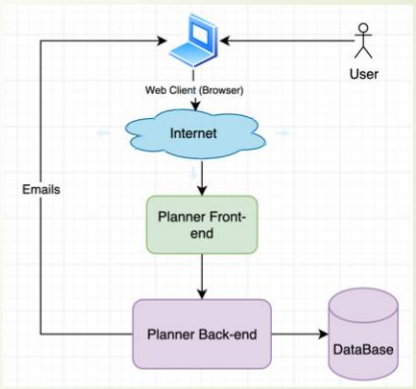




Діаграма активності та контекстна діаграма

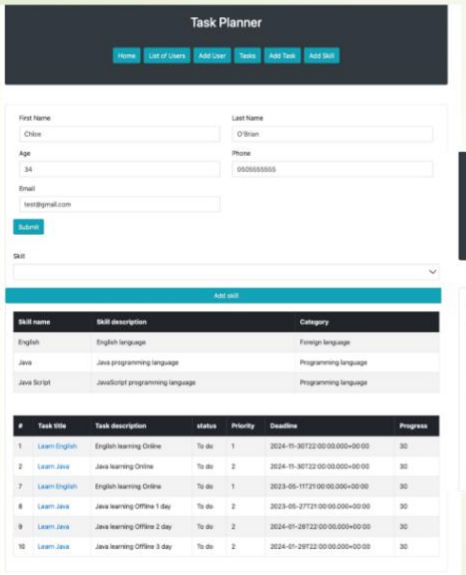
7

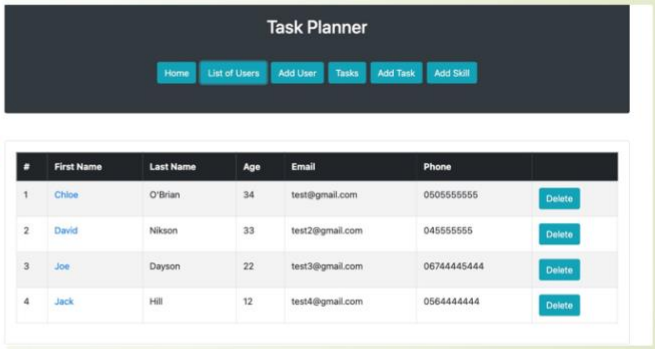




Інтерфейс користувача

8





Декотрі базові тест кейси

D тест-кейсу	Опис	Кроки тестування	Очікуваний результат	Фактичний результат	Статус
ДК001	Додати користувача з валідними даними	1. Натиснути "Додати користувача" 2. Заповнити всі поля валідними даними 3. Натиснути "Підтвердити"	Користувача успішно додано	Успішно	Проведено
ДК002	Додати користувача з відсутнім обов'язковим полем	1. Натиснути "Додати користувача" 2. Залишити поле "Ім'я" порожнім 3. Заповнити інші поля 4. Натиснути "Підтвердити"	Відображено повідомлення про помилку	Успішно	Проведено
ДК003	Додати користувача з невалідним форматом email	1. Натиснути "Додати користувача" 2. Ввести невалідний email (напр. "test@test") 3. Заповнити інші поля 4. Натиснути "Підтвердити"	Відображено повідомлення про невалідний email	Успішно	Проведено
ДК004	Додати користувача з існуючим email	1. Натиснути "Додати користувача" 2. Ввести email існуючого користувача 3. Заповнити інші поля	Відображено повідомлення про дублювання email	Успішно	Проведено

Висновки

10

Важливим аспектом є можливість отримання статистичної інформації у вигляді звітів, а також наявність в додатку дошок або таблиць, де користувач може зосереджуватися на пріоритетних і важливих задачах, не забуваючи про рутинні завдання. Ефективне управління часом та підвищення продуктивності залишаються актуальними проблемами, що потребують вирішення. З цією метою розробляється програмна система, яка дозволяє планувати повсякденні справи, підвищуючи продуктивність та забезпечуючи ефективне управління часом.

Під час розробки використовується реляційна база даних та архітектура клієнт-сервер. Для реалізації доступу до даних для веб-клієнта застосовується REST підхід, а доступ до даних СУБД виконується за допомогою JPA API та імплементації ORM Hibernate. Для розробки фронтенду використовується архітектура SPA (Single Page Application), реалізована за допомогою Angular Framework, TypeScript та BootStrap.