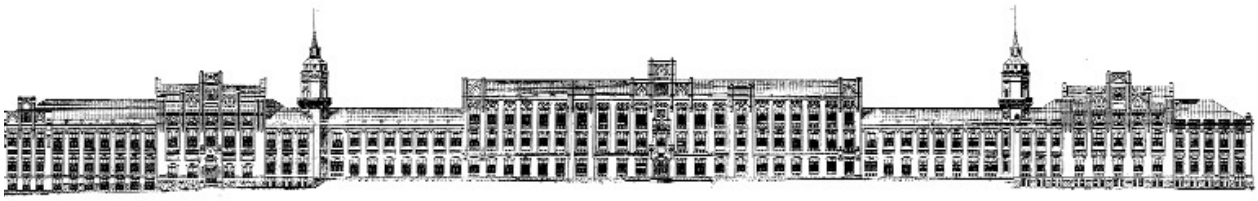


Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Звіт

з дисципліни «Технології розроблення програмного забезпечення»

Варіант: 23

Виконала
студентка групи ІА-23:
Павленко Анастасія

Перевірив:
Мягкий Михайло
Юрійович

Київ 2024

Лабораторна робота №2

Діаграма варіантів використання. Сценарії варіантів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи.

Варіант 23: Project Management software (proxy, chain of responsibility, abstract factory, bridge, flyweight, client-server)

Програмне забезпечення для управління проектами повинно мати наступні функції: супровід завдань/вимог/проектів, списків команд, поточних завдань, планування за методологіями agile/kanban/rup (включаючи дошку завдань, ітерації тощо), мати можливість прикріплювати вкладені файли до завдань та посилатися на конкретні версії програми, зберігати виконувані файли для кожної версії.

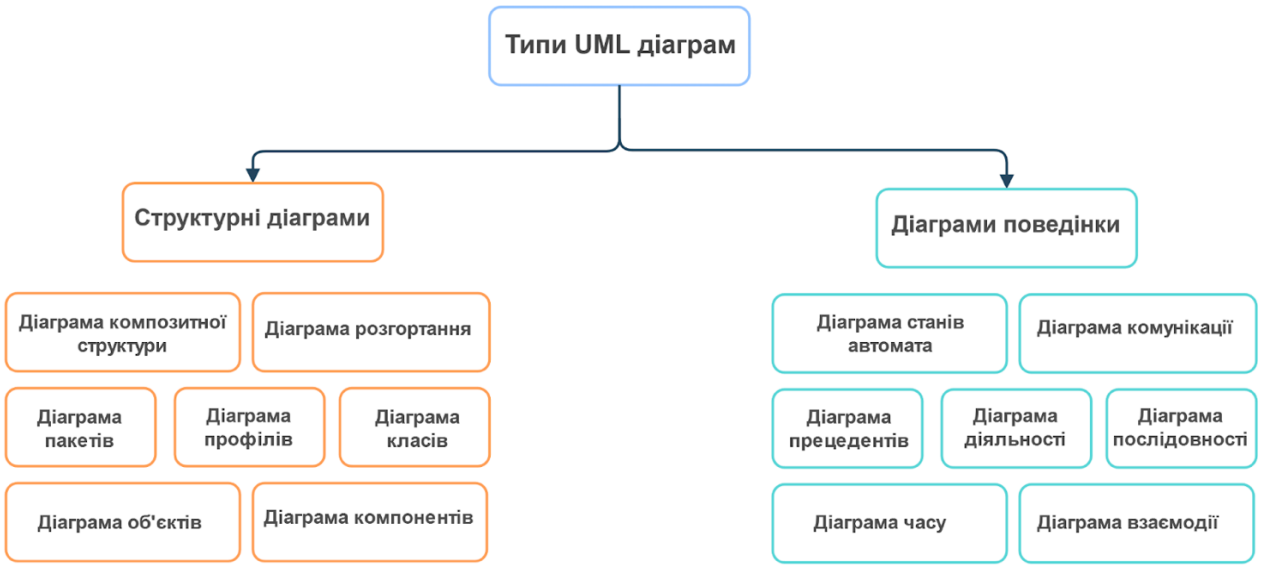
Завдання.

1. Ознайомитися з короткими теоретичними відомостями
2. Проаналізуйте тему та намалюйте схему прецеденту, що відповідає обраній темі лабораторної.
3. Намалюйте діаграму класів для реалізованої частини системи.
4. Виберіть 3 прецеденти і напишіть на їх основі прецеденти.
5. Розробити основні класи і структуру системи баз даних.
6. Класи даних повинні реалізувати шаблон Репозиторію для взаємодії з базою даних.
7. Підготувати звіт про хід виконання лабораторних робіт. Звіт, що подається повинен містити: діаграму прецедентів, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних.

Короткі теоретичні відомості

UML (Unified Modeling Language) — уніфікована мова моделювання, що використовується розробниками для візуалізації роботи систем.

UML-діаграми поділяються на 14 типів:



Діаграма варіантів використання дозволяє уявити типи ролей та їх взаємодію із системою, проте не показує порядок виконання кроків. Зображує функціональні вимоги з точки зору користувача.

Діаграми варіантів використання складаються з 4 об’єктів:

Актор. Поняття когось, або чогось, що взаємодіє з системою, але не належить до неї. Усіх акторів умовно можна розділити на первинних (ті, що ініціюють взаємодію з системою) та вторинних (ті що реагують).

Випадок використання (прецедент). Прецеденти визначають очікувану поведінку та відповідають на питання, що робить система. Представляють набір можливих функцій, дій або завдань.

Система. Те, що моделюється.

Зв’язки (відношення). Усі актори мають бути пов’язані з прецедентом. Проте не усі прецеденти повинні бути пов’язані з акторами. Частіше всього для зображення зв’язку використовується суцільна лінія.

У діаграмі варіантів використання існує 4 типи зв’язків:

1. **Асоціація (Association).** Звичайний зв’язок актора та прецеденту. Незалежно від типу зв’язку, будь-який актор повинен бути пов’язаний принаймні з одним варіантом використання.
2. **Розширення (Extend).** Показує додаткову функціональність або можливий не обов’язковий варіант поведінки системи.
3. **Включення (Include).** Показує, що поведінка одного прецеденту включається як складовий компонент у послідовність поведінки іншого прецеденту. Ілюструє, що саме використовує базовий варіант для виконання операції.
4. **Генералізація (Generalization).** Генералізація це батьківсько-дочірні відношення.

Діаграма класів – це візуальна нотація, яка використовується для побудови та візуалізації об'єктно-орієнтованих систем. Діаграма класів в уніфікованій мові моделювання — це статична структурна діаграма, що демонструє властивості системи, класи, операції та зв’язки між об’єктами для опису структури системи.

На діаграмі класи представлені у вигляді прямокутників, які містять три відділення:

- Верхня частина містить **назву класу**.
- Середнє відділення містить **атрибути класу**.
- Нижній відсік містить **операції, які може виконувати клас**.

Відношення на рівні екземплярів

Залежність (англ. dependency) — це тип асоціації, де існує семантичний зв'язок між залежним і незалежним елементами моделі. Вона існує між двома елементами, якщо зміни у визначенні одного елемента можуть спричинити зміни в іншому.

Асоціація (англ. association) показує, що об'єкти однієї сутності пов'язані з об'єктами іншої сутності.

Агрегація (англ. aggregation)— проста асоціація між двома класами, де один з класів має вищий ранг і складається з декількох менших за рангом.
Композиція (англ. composition) — більш строгий варіант агрегації.

Відношення на рівні класів

Узагальнення/Наслідування – один з двох споріднених класів вважається спеціалізованою формою іншого, а суперклас вважається узагальненням підкласу.
Реалізація/Впровадження – зв'язок між двома елементами моделі, в якому один елемент моделі реалізує поведінку, яку визначає інший елемент моделі.

Концептуальна модель системи — це абстрактна модель, яка представляє основні сутності та їх взаємозв’язки у системі. Вона не деталізує конкретні технічні аспекти або реалізацію, а фокусується на тому, що система повинна робити і з чим вона взаємодіє. Основними компонентами концептуальної моделі є:

- *Сутності* (Entities) — ключові об’єкти, з якими система працює.
- *Атрибути* (Attributes) — властивості сутностей.
- *Зв'язки* (Relationships) — взаємодії між сутностями.

Нормальні форми баз даних — це правила або критерії для структурування таблиць баз даних з метою зменшення дублювання даних і забезпечення узгодженості. Основні нормальні форми:

- **Перша нормальна форма**: всі поля містять лише атомарні значення.
- **Друга нормальна форма**: таблиця повинна бути в 1NF і не мати часткових залежностей.
- **Третя нормальна форма**: таблиця повинна бути в 2NF і не мати транзитивних залежностей.

Хід виконання лабораторної роботи

1. Аналіз теми обраного проєкту.

Розглянемо вимоги до системи:

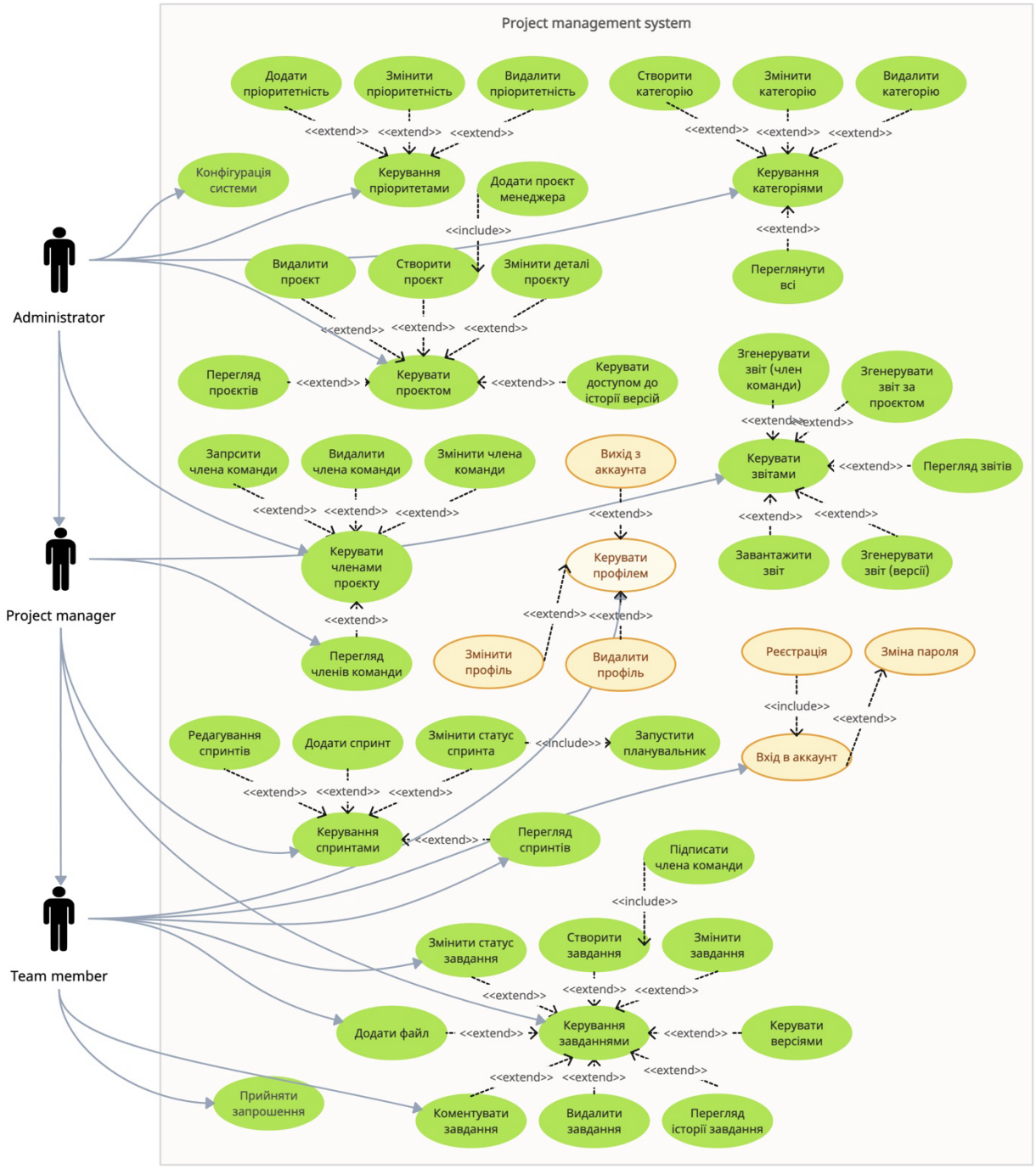
1. Управління завданнями, вимогами та проєктами – користувачі мають мати змогу створювати, редагувати та видаляти завдання та проєкти, керувати ними за допомогою дошок завдань (agile, kanban).
2. Управління командами – має бути можливість призначати завдання командним учасникам, створювати команди та редагувати їх.
3. Планування за допомогою Agile, Kanban та Scrum – система повинна включати ітерації, можливість перегляду завдань у вигляді дошки.
4. Прикріплення файлів – до завдань потрібно мати можливість додавати файли та відстежувати версії проєктів. Збереження виконуваних файлів для кожної версії програми.

Опис діаграми прецедентів:

На наданій діаграмі прецедентів зображені три основні ролі в системі:

1. **Проєктний менеджер (Project Manager)**:
   - Може керувати спринтами: додавати, змінювати статус і модифікувати.
   - Має можливість призначати завдання членам команди.
   - Керує завданнями: створює, редагує, видаляє та оновлює статус завдань.
   - Керує своїм профілем: редагує або видаляє профіль.
   - Відповідає за звіти про проєкт та учасників команди, завантажує звіти.
   - Управляє командою: додає нових учасників та видаляє їх з команди.
2. **Член команди (Team Member)**:
   - Може переглядати свої проєкти, створювати коментарі до завдань, прикріплювати файли до завдань.
   - Бачить історію завдань, управляє своїм профілем.
3. **Адміністратор (Administrator)**:
   - Керує пріоритетами завдань, створює категорії для проєктів, редагує та видаляє їх.

- Може створювати нові проєкти, редагувати деталі існуючих проєктів і видаляти їх.
- Відповідає за додавання та призначення проєктних менеджерів.
- Може управляти власним профілем, керувати користувачами та налаштуваннями системи.



Більш чітке зображення можна переглянути в папці лабораторної роботи №2 на гітхаб.

Розберемо декілька прецедентів більш детально:

Назва прецеденту: Створити проект

Актор: Адміністратор

Мета: Створити проєкт у застосунку

Передумови: Проєкт не створений у застосунку

Результат: Проєкт створений у застосунку

Основний сценарій:

1. Адміністратор натискає на елемент меню.
2. Адміністратор обирає опцію «Створити новий проєкт».
3. Адміністратор вводить назву проєкту.
4. Адміністратор вводить опис проєкту.
5. Адміністратор обирає дату початку роботи над проєктом.
6. Адміністратор обирає менеджера проєкту.
7. Адміністратор підтверджує введені дані за допомогою кнопки «Додати проєкт».
8. Лист підтвердження приходить на пошту обраного менеджера проєкту про створення нового проєкту.

Можливі помилки та винятки:

1. **Порожня назва проєкту:**
   - **Умова:** Адміністратор не ввів назву проєкту.
   - **Помилка:** «Назва проєкту не може бути порожньою».
   - **Дії:** Система просить ввести назву проєкту.
2. **Порожній опис проєкту:**
   - **Умова:** Адміністратор не ввів опис проєкту.
   - **Помилка:** «Опис проєкту обов'язковий для заповнення».
   - **Дії:** Система вимагає заповнити поле з описом.
3. **Некоректна дата початку:**

- **Умова:** Введена дата початку роботи над проєктом є в минулому або не відповідає формату.
- **Помилка:** «Дата початку не може бути в минулому або має бути у форматі дд/мм/рррр».
- **Дії:** Система просить ввести коректну дату.

4. **Назва проєкту вже існує:**
   - **Умова:** Назва, введена адміністратором, вже використовується для іншого проєкту.
   - **Помилка:** «Проєкт із такою назвою вже існує. Введіть іншу назву».
   - **Дії:** Система просить змінити назву проєкту.

Назва прецеденту: Створити завдання

Актор: Проєкт менеджер

Мета: Створити нове завдання

Передумови: Завдання не існує в проєкті

Результат: Список завдань проєкту оновлений, нове завдання додано до проєкту

Основний сценарій:

1. Проєкт менеджер натискає на елемент меню завдань.
2. Проєкт менеджер обирає опцію «Створити нове завдання».
3. Проєкт менеджер обирає категорію завдання.
4. Проєкт менеджер обирає пріоритетність завдання.
5. Проєкт менеджер вводить опис нового завдання.
6. Проєкт менеджер призначає завдання певному члену команди, який працює над проєктом.
7. Опціонально проєкт менеджер може додати завдання до спринта.
8. Проєкт менеджер підтверджує створення нового завдання кнопкою «Додати нове завдання».

Можливі помилки та винятки:

1. **Порожній опис завдання:**
   - **Умова:** Опис завдання не введений.
   - **Помилка:** «Опис завдання не може бути порожнім».
   - **Дії:** Система просить ввести опис завдання.
2. **Завдання не може бути додано до спринта:**
   - **Умова:** Спринт, до якого менеджер намагається додати завдання, закритий або не існує.
   - **Помилка:** «Неможливо додати завдання до обраного спринта».
   - **Дії:** Система пропонує вибрати інший спринт або створити новий.
3. **Перевищення ліміту завдань у спринті:**
   - **Умова:** Менеджер намагається додати завдання до спринта, який вже має максимальну кількість завдань.
   - **Помилка:** «Спринт вже містить максимальну кількість завдань».
   - **Дії:** Система пропонує створити новий спринт або зменшити кількість завдань у поточному.
4. **Конфлікт з іншими завданнями:**
   - **Умова:** Завдання з таким самим описом або деталями вже існує в проєкті.
   - **Помилка:** «Завдання з подібними характеристиками вже існує в проєкті».
   - **Дії:** Система пропонує переглянути список завдань або змінити опис завдання.

Назва прецеденту: Додати спринт

Актор: Проєкт менеджер

Мета: Створити новий спринт

Передумови: Спринт не був створений

Результат: Новий спринт створений у проєкті

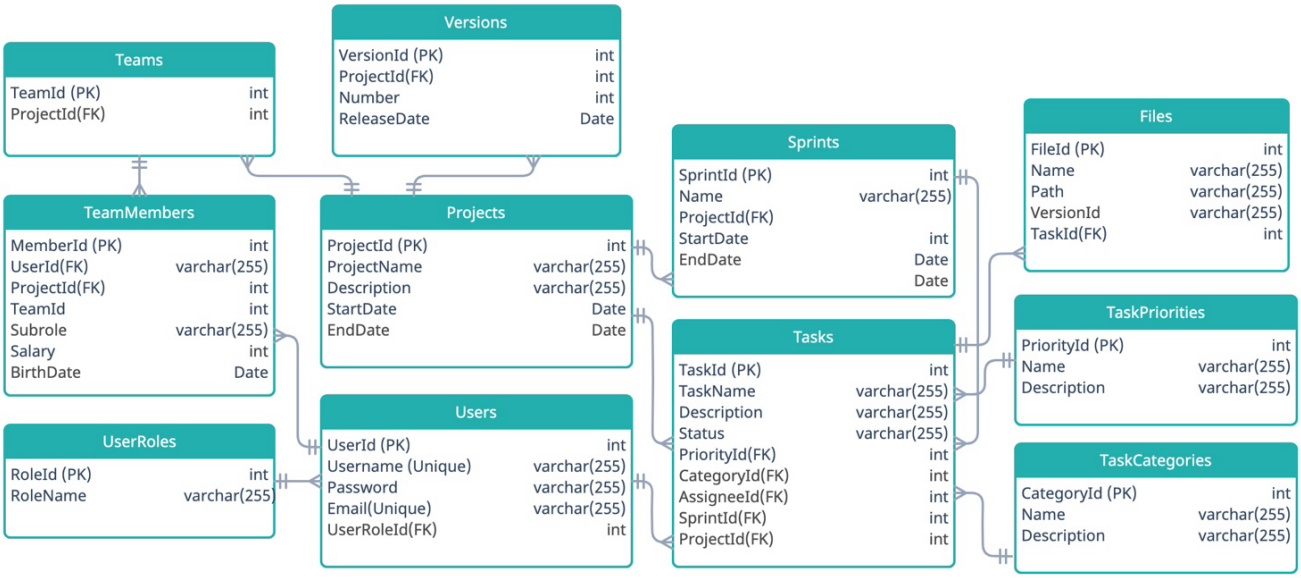
Основний сценарій:

1. Проєкт менеджер натискає на елемент меню проєкту.
2. Проєкт менеджер обирає опцію «Створити новий спринт».
3. Проєкт менеджер обирає тип спринта.
4. Проєкт менеджер вводить назву нового спринта.
5. Проєкт менеджер обирає початкову та кінцеву дату нового спринта (проєкт не може мати декілька активних спринтів одночасно).
6. Проєкт менеджер може обрати список завдань для нового спринта.

Можливі помилки та винятки:

1. **Некоректні дати спринта:**
   - **Умова:** Початкова дата пізніше кінцевої або дати не відповідають формату.
   - **Помилка:** «Початкова дата не може бути пізніше кінцевої, або формат дат некоректний».
   - **Дії:** Система просить ввести коректні дати.
2. **Активний спринт вже існує:**
   - **Умова:** У проєкті вже є активний спринт.
   - **Помилка:** «У проєкті не може бути кілька активних спринтів одночасно».
   - **Дії:** Система пропонує завершити або закрити поточний спринт перед створенням нового.
3. **Не обраний тип спринта:**
   - **Умова:** Тип спринта не обрано.
   - **Помилка:** «Тип спринта обов'язковий для вибору».
   - **Дії:** Система просить обрати тип спринта (наприклад, планування, виконання, завершення).

Розробимо структуру бази даних:



Більш чітке зображення можна переглянути в папці лабораторної роботи №2 на гітхаб.

Таблиці бази даних

Users (Користувачі)

- user_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор користувача)
- username — VARCHAR(255), UNIQUE (унікальний логін)
- password — VARCHAR(255)
- email — VARCHAR(255), UNIQUE (унікальний email)
- role_id — INT, FK (посилання на User_Roles.role_id) (роль користувача)
- created_at — TIMESTAMP, DEFAULT CURRENT_TIMESTAMP

User_Roles (Ролі користувачів)

- role_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор ролі)
- role_name — VARCHAR(100), UNIQUE (назва ролі, наприклад: адміністратор, менеджер, учасник)

Teams (Команди)

- team_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор команди)

- project_id — INT, FK (посилання на Projects.project_id) (ідентифікатор проекту)
- team_name — VARCHAR(255) (назва команди)
- **Зв'язки**:
    - project_id → Projects.project_id (один до багатьох)

Team_Members (Члени команди)

- team_member_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор члена команди)
- user_id — INT, FK (посилання на Users.user_id) (ідентифікатор користувача)
- team_id — INT, FK (посилання на Teams.team_id) (ідентифікатор команди)
- sub_role — VARCHAR(100) (конкретизована роль: дизайнер, девелопер тощо)
- salary — DECIMAL(10, 2) (зарплата учасника)
- additional_data — TEXT (окремі дані про учасника)
- **Зв'язки**:
    - user_id → Users.user_id (багато до одного)
    - team_id → Teams.team_id (багато до одного)

Projects (Проекти)

- project_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор проекту)
- project_name — VARCHAR(255) (назва проекту)
- description — TEXT (опис проекту)
- start_date — DATE
- end_date — DATE

Task_Priorities (Пріоритети проектів)

- priority_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор пріоритету)

- priority_name — VARCHAR(100), UNIQUE (назва пріоритету, наприклад: низький, середній, високий)

Task_Categories (Категорії проектів)

- category_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор категорії)
- category_name — VARCHAR(100), UNIQUE (назва категорії, наприклад: розробка, тестування)

Tasks (Завдання)

- task_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор завдання)
- project_id — INT, FK (посилання на Projects.project_id) (ідентифікатор проекту)
- task_name — VARCHAR(255) (назва завдання)
- description — TEXT (опис завдання)
- priority_id — VARCHAR(50) (пріоритет завдання)
- category_id — VARCHAR(100) (категорія завдання)
- status — VARCHAR(50) (статус: to-do, in-progress, done)
- assignee_id — INT, FK (посилання на Users.user_id) (ідентифікатор користувача, відповідального за завдання)
- sprint_id — INT, FK (посилання на Sprints.sprint_id) (ідентифікатор спринту)
- **Зв'язки**:
  - project_id → Projects.project_id (багато до одного)
  - assignee_id → Users.user_id (багато до одного)
  - sprint_id → Sprints.sprint_id (багато до одного)
  - priority_id → Task_Priorities.priority_id (багато до одного)
  - category_id → Task_Categories.category_id (багато до одного)

Sprints (Спринти)

- sprint_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор спринту)
- project_id — INT, FK (посилання на Projects.project_id) (ідентифікатор проекту)
- sprint_name — VARCHAR(255) (назва спринту)
- start_date — DATE
- end_date — DATE
- **Зв'язки**:
  - project_id → Projects.project_id (багато до одного)

Files (Файли)

- file_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор файлу)
- task_id — INT, FK (посилання на Tasks.task_id) (ідентифікатор завдання)
- file_name — VARCHAR(255) (назва файлу)
- file_path — VARCHAR(255) (шлях до файлу)
- version_id — INT, FK (посилання на Versions.version_id) (ідентифікатор версії, до якої належить файл)
- **Зв'язки**:
  - task_id → Tasks.task_id (багато до одного)
  - version_id → Versions.version_id (багато до одного)

Versions (Версії програм)

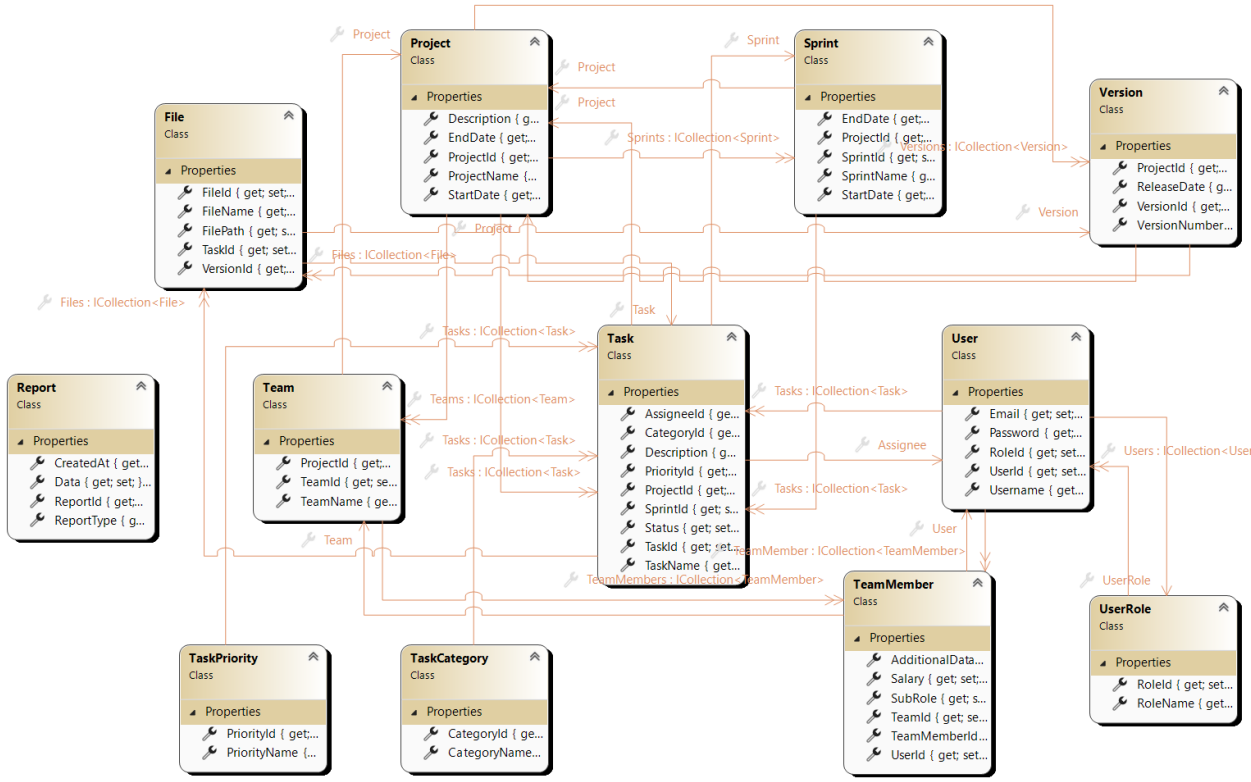
- version_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор версії)
- project_id — INT, FK (посилання на Projects.project_id) (ідентифікатор проекту)
- version_number — VARCHAR(50) (номер версії)
- release_date — DATE
- **Зв'язки**:

- project_id → Projects.project_id (багато до одного)

Reports (Звіти)

- report_id — INT, PK, AUTO_INCREMENT (унікальний ідентифікатор звіту)
- report_type — VARCHAR(50) (тип звіту: проект, завдання, користувач)
- created_at — TIMESTAMP, DEFAULT CURRENT_TIMESTAMP
- data — JSON (дані звіту у вигляді JSON)
- **Зв'язки**:
  - Можливі посилання на різні таблиці в залежності від типу звіту (проект, завдання, користувач).

Діаграма класів:



Більш чітке зображення можна переглянути в папці лабораторної роботи №2 на гітхаб.

Реалізуємо шаблон Репозиторію для взаємодії з базою даних:

На наведеному зображенні створені **репозиторії** для кожного класу даних у системі управління проєктами, які реалізують шаблон Репозиторію. Кожен репозиторій містить базові методи для CRUD-операцій: **додавання (Add)**, **отримання за ідентифікатором (GetById)**, **отримання всіх записів (GetAll)**, **оновлення (Update)** та **видалення (Delete)**. Ось детальний опис кожного елемента:

1. **UserRepository**
   - Реалізує інтерфейс IUserRepository для роботи з користувачами.
   - Методи:
     - Add: Додає нового користувача.

- GetById: Повертає користувача за унікальним ідентифікатором.
- GetAll: Повертає всіх користувачів.
- Update: Оновлює дані користувача.
- Delete: Видаляє користувача за ідентифікатором.

2. **UserRoleRepository**
   - Реалізує IUserRoleRepository для роботи з ролями користувачів.
   - Методи аналогічні, але для ролей користувачів: додавання, отримання за ідентифікатором, отримання всіх ролей, оновлення та видалення.
3. **TeamRepository**
   - Реалізує ITeamRepository для керування командами.
   - Операції для додавання нових команд, отримання команд за ідентифікатором, отримання всіх команд, оновлення та видалення команд.
4. **TeamMemberRepository**
   - Реалізує ITeamMemberRepository для роботи з членами команд.
   - Включає CRUD-операції для додавання, отримання, оновлення та видалення членів команд.
5. **ProjectRepository**
   - Реалізує IProjectRepository для керування проєктами.
   - Включає методи для роботи з проєктами: додавання, отримання за ідентифікатором, оновлення та видалення.
6. **ProjectPriorityRepository**
   - Реалізує IProjectPriorityRepository для роботи з пріоритетами проєктів.
   - Включає CRUD-методи для пріоритетів проєктів.
7. **TaskRepository**
   - Реалізує ITaskRepository для керування завданнями.
   - Включає базові методи для додавання, отримання завдань, їх оновлення та видалення.

8. **SprintRepository**
   - Реалізує ISprintRepository для роботи зі спринтами (етапами розробки).
   - Включає CRUD-операції для керування спринтами.
9. **FileRepository**
   - Реалізує IFileRepository для роботи з файлами.
   - Включає базові операції для додавання, отримання та видалення файлів у системі.
10. **VersionRepository**
   - Реалізує IVersionRepository для роботи з версіями продукту або компонентів.
   - Включає CRUD-операції для версій.
11. **ReportRepository**
   - Реалізує IReportRepository для роботи зі звітами.
   - Включає операції для додавання, отримання, оновлення та видалення звітів.

Суть реалізації:

Інтерфейси репозиторіїв декларують необхідні операції для відповідних класів.

Класи репозиторіїв реалізують ці інтерфейси, надаючи логіку для доступу до даних через базові CRUD-операції.

Кожен репозиторій відповідає за окремий аспект роботи з даними (користувачі, команди, проєкти тощо), що робить систему гнучкою та легкою для масштабування. Таким чином, шаблон Репозиторію дозволяє ізолювати доступ до даних і використовувати його у сервісах або бізнес-логіці.

Код до створеного проєкту можна переглянути на github.

Висновок: У ході виконання лабораторної роботи було реалізовано декілька важливих етапів для створення системи управління проєктами:

Було проаналізовано тему проєкту, розглянуто вимоги до функціональності системи, визначено прецеденти використання розроблюваної системи, акторів (користувачів продукту) та побудовано діаграму використання на основі цього аналізу. Було детально розглянуто деякі прецеденти та описано сценарій їх використання разом з можливими помилками та винятками в процеси виконання.

Були створені класи-моделі, які відображають основні сутності системи: користувачі, ролі, команди, проєкти, завдання, спринти, файли та звіти. Ці моделі відображають структуру даних, що зберігаються в базі, а також їхні атрибути та зв'язки між сутностями, наприклад, користувачі та ролі, проєкти та команди.

На основі розроблених моделей було створено структуру бази даних. Було визначено основні таблиці для збереження інформації про користувачів, проєкти, завдання та інші сутності. Також були налаштовані зв'язки між таблицями (наприклад, «один до багатьох» між користувачем і роллю або між проєктом і завданням), що забезпечує логічну цілісність і ефективне збереження даних.

Для взаємодії з базою даних було створено шаблони репозиторіїв. Ці репозиторії виконують CRUD-операції (створення, читання, оновлення, видалення), що дозволяє маніпулювати даними в базі. Кожен репозиторій реалізовує інтерфейс для конкретної сутності, що забезпечує чітку та гнучку структуру коду.

Загалом, лабораторна робота допомогла закріпити практичні навички в проєктуванні баз даних, створенні моделей та використанні шаблону Репозиторію для ефективної взаємодії з даними. Така система є легкою для розширення та підтримки, що є важливим аспектом при розробці масштабованих інформаційних систем.