

Харківський національний університет радіоелектроніки
(повне найменування вищого навчального закладу)

Кафедра штучного інтелекту
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА З ДИСЦИПЛІНИ Програмування під .NET Core

Використання технології ASP.NET Core MVC для розробки сайту
«Менеджер завдань»
(тема проекту)

Студента(ки) 4 курсу групи ІТШІ-20-2
напряму підготовки КН

Науменко А.С.
(прізвище та ініціали)

Керівник ст. викл. Бібічков І.Є.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____

Кількість балів: _____ Оцінка ECTS _____

Члени комісії

_____	<u>Філатов В.О.</u>
(підпис)	(прізвище та ініціали)
_____	<u>Губін В.О.</u>
(підпис)	(прізвище та ініціали)
_____	<u>Бібічков І.Є.</u>
(підпис)	(прізвище та ініціали)

м. Харків - 2024 рік

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ
(повне найменування вищого навчального закладу)

Факультет ком'ютерних наук
Кафедра штучного інтелекту
Освітньо-кваліфікаційний рівень бакалавр

ЗАТВЕРДЖУЮ
Завідувач кафедри ШІ

(В.О. Філатов)

“ ” _____ 2024 року

З А В Д А Н Н Я
НА КУРСОВУ РОБОТУ СТУДЕНТУ

Науменку Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Використання технології ASP.NET Core MVC для розробки сайту «Менеджер завдань»

керівник роботи ст. викл. Бібічков І.Є.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 27.03.2023

3. Вихідні дані до роботи Специфікація технології ASP.NET Інформаційні джерела з обраної предметної галузі, специфікація бази даних, технічне завдання з вимогами до роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналіз предметної області та постановка задачі, проектування і реалізація бази даних, програмна реалізація проєкту

5. Перелік графічного матеріалу (рисунки, екранні форми) Знімки екрану з зображенням сайту, знімки екрану з кодом

6. Дата видачі завдання 25.02.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/П	Назва етапів проекту	Строк виконання етапів проекту	Примітка
1	Узгодження і затвердження теми	25.02 – 26.02	виконано
2	Аналіз предметної області та постановка задачі	26.02 – 28.02	виконано
3	Проектування і реалізація бази даних	28.02 – 01.03	виконано
4	Програмна реалізація проекту	01.03 – 09.03	виконано
5	Оформлення пояснювальної записки	09.03 – 10.03	виконано
6	Захист роботи	11.03 – 27.03	

Студент


(підпис)

Науменко А.С.

(прізвище та ініціали)

Керівник роботи

(підпис)

Бібічков І.Є.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 37 с., 16 рис., 1 додаток, 10 джерел.

БАЗА ДАНИХ, САЙТ, ПРОГРАМНИЙ ЗАСТОСУНОК, ФІЛЬТРАЦІЯ ASP.NET Core, C#, CONTROLLER, MANAGEMENT, MVC, MySQL, TASK, VIEW

Темою роботи є використання технології ASP.NET Core MVC для розробки сайту «Менеджер завдань» з підключенням до бази даних MySQL.

Метою роботи є розробити веб-сайт, який дозволить користувачам ефективно керувати завданнями, співробітниками та проектами, використовуючи технологію ASP.NET Core MVC та базу даних MySQL.

Для виконання цієї роботи було використано середовище програмування JetBrains Rider, яке надає розширені можливості для розробки веб-додатків на платформі ASP.NET Core MVC. Для зберігання даних було обрано базу даних MySQL, яка забезпечує надійність та швидкодію роботи з великим обсягом інформації.

Основні властивості і можливості розробленої програми: Створений веб-сайт «Менеджер завдань» надає користувачам наступні можливості:

Управління задачами, співробітниками та проектами: користувачі можуть додавати, переглядати, редагувати та видаляти усі сутності.

Додатково, сайт надає можливість фільтрації задач за окремими співробітниками або проектами, що дозволяє швидко знаходити необхідну інформацію. Використання технології ASP.NET Core MVC дозволило забезпечити швидкий та зручний інтерфейс для користувачів, а підключення до бази даних MySQL забезпечило надійність та ефективність зберігання та обробки даних.

ЗМІСТ

ВСТУП	6
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ	7
1.1 Аналіз предметної області.....	7
1.2 Постановка задачі.....	8
2 ПРОЕКТУВАННЯ СИСТЕМИ.....	10
2.1 Проектування моделей	10
2.2 Модель Employee	10
2.3 Модель Project	12
2.4 Модель Issue	13
2.6 Проектування контролерів	14
2.7 Проектування бази даних	19
3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ	22
3.1 Опис використаних стандартних бібліотек	22
3.2 Опис представлень та візуального інтерфейсу	23
ВИСНОВКИ.....	28
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	29
ДОДАТОК А КОД	30

ВСТУП

В сучасному інформаційному суспільстві ефективне управління завданнями, співробітниками та проектами стає ключовим фактором успіху для багатьох організацій. З урахуванням швидкого темпу розвитку технологій і зростання конкуренції на ринку, потреба в надійному і зручному інструменті для організації робочих процесів стає все більш актуальною. У цьому контексті використання технології ASP.NET Core MVC для розробки веб-сайту «Менеджер завдань» виявляється важливим і доцільним.

ASP.NET Core MVC - це потужний інструмент для розробки веб-додатків, який надає розширені можливості управління різноманітними аспектами веб-сайту. Його використання дозволяє створювати швидкі, масштабовані та безпечні додатки, які забезпечують зручний інтерфейс користувача і ефективне взаємодію з базою даних. Завдяки вбудованим засобам маршрутизації, контролерам та моделям, ASP.NET Core MVC спрощує процес розробки і підтримки веб-додатків, забезпечуючи високу продуктивність та зручність у використанні.

Враховуючи зазначені переваги та потреби сучасного бізнесу в ефективному управлінні ресурсами, використання технології ASP.NET Core MVC для створення веб-сайту «Менеджер завдань» стає відповідною відповіддю на поставлені завдання. Цей підхід дозволить забезпечити швидке розгортання додатку, його стабільну роботу та зручний інтерфейс користувача, що в свою чергу сприятиме підвищенню ефективності роботи організації та досягненню поставлених цілей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Предметна область "Менеджер завдань" охоплює широкий спектр діяльності, пов'язаний з організацією та контролем за завданнями, співробітниками та проектами в рамках певної організації чи команди. Управління завданнями є ключовою складовою ефективного ведення бізнесу, оскільки воно дозволяє раціоналізувати процеси виконання роботи, розподілити відповідальності та забезпечити вчасне виконання завдань. Використання технологій ASP.NET Core MVC та MySQL для реалізації цієї предметної області надає значні переваги у плані ефективності, надійності та швидкодії.

MySQL є потужною та надійною системою управління базами даних, яка забезпечує збереження та обробку великого обсягу інформації. Її використання для зберігання даних про завдання, співробітників та проекти забезпечує консистентність та доступність даних у будь-який момент часу. MySQL дозволяє ефективно працювати з реляційними даними, що відображають взаємозв'язки між різними сутностями в предметній області, такими як завдання, співробітники та проекти.

ASP.NET Core MVC, у свою чергу, надає потужні інструменти для розробки веб-додатків, які забезпечують зручний та ефективний інтерфейс користувача. Модель-вид-контролер (MVC) підхід дозволяє розділити логіку додатку на компоненти, що полегшує його розробку та підтримку. ASP.NET Core MVC забезпечує зручну маршрутизацію, обробку запитів та генерацію веб-сторінок, що дозволяє розробникам швидко створювати функціональні та естетично збалансовані веб-додатки.

Використання технології ASP.NET Core MVC для розробки веб-сайту "Менеджер завдань" спільно з базою даних MySQL дозволяє створити потужний та надійний інструмент для управління робочими процесами в

організації. Цей підхід забезпечує високу продуктивність, зручний інтерфейс користувача та надійність збереження даних, що є ключовими факторами успіху в сучасному бізнес-середовищі.

Задача (Task): Основна одиниця роботи, яка визначається заголовком, описом, статусом виконання, датою створення та призначеним виконавцем. Задачі можуть бути призначені для конкретних проектів або співробітників.

Співробітник (Employee): Представляє працівника організації, що бере участь у виконанні різних завдань. Інформація про співробітника може включати його/її ім'я, посаду та контактні дані.

Проект (Project): Група завдань, які мають спільну мету або призначення. Кожен проект має назву, опис, дату початку та закінчення, а також список учасників.

Процеси, в яких беруть участь ці сутності, включають створення, редагування та видалення завдань, які можуть бути призначені конкретним співробітникам та проектам, а також організацію співробітників у рамках проектів, визначення їх ролей та відповідальності. І також включають планування та відстеження прогресу проектів шляхом призначення завдань та моніторингу їх виконання. На сайті також доступна фільтрація та пошук завдань за певними критеріями, такими як виконавець чи проект.

Для спрощення програми можна припустити, що кожен співробітник може мати лише одну активну роль у проекті, а завдання можуть бути призначені лише одному виконавцю. Також можна прийняти припущення про те, що в один момент часу завдання може бути призначене лише для одного проекту.

1.2 Постановка задачі

У даній роботі метою є розробка веб-додатку "Менеджер завдань" з використанням технології ASP.NET Core MVC та підключенням до бази

даних MySQL. Додаток має забезпечувати зручне управління завданнями, співробітниками та проектами в рамках організації.

Основними функціональними вимогами до програми є:

- реалізація класів моделі для сутностей "Задача" (Task), "Співробітник" (Employee) та "Проект" (Project) з урахуванням їхніх характеристик та взаємозв'язків між ними;
- створення контролерів та методів дії для реалізації основної функціональності програми, зокрема додавання, редагування, видалення та перегляд даних про завдання, співробітників та проекти;
- реалізація необхідних представлень для відображення інформації про завдання, співробітників та проекти в зручному та зрозумілому форматі;
- використання майстер-сторінок, секцій, часткових представлень, стандартних та оригінальних хелперів для забезпечення зручного та ефективного інтерфейсу користувача;
- організація взаємодії із сесіями для забезпечення функціональності аналогічної авторизації та кошика, відповідно до вимог завдання;
- заборона використання стандартних елементів програм, таких як авторизація та кошик, та їхнє заміщення згідно з вказаними вимогами;
- програма повинна забезпечувати можливість фільтрації та пошуку завдань за окремими співробітниками або проектами, що дозволяє користувачам швидко знаходити необхідну інформацію.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Проектування моделей

У процесі розробки веб-додатку "Менеджер завдань" було вирішено почати з проектування моделей, оскільки вони є основою програми і визначають структуру даних, що використовується в додатку. При розробці моделей було враховано необхідність відображення сутностей "Задача", "Співробітник" та "Проект", які є основними об'єктами у предметній області.

Згідно з традиційною структурою проектів ASP.NET Core MVC, класи моделей було розміщено всередині папки Models (рис. 2.1).

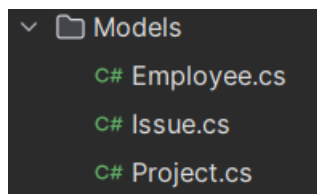


Рисунок 2.1 – Ієрархія моделей

2.2 Модель Employee

Сутність Employee є важливою складовою системи "Менеджер завдань". Ця сутність представляє інформацію про співробітників організації, які беруть участь у виконанні різних завдань та проектів (рис 2.2).

```

public class Employee
{
    [Key]
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
}

```

Рисунок 2.2 – Клас Employee

Для коректного опису цієї сутності моделі було додано декілька властивостей. Id - унікальний ідентифікатор співробітника, представлений у форматі GUID. Це поле є первинним ключем в базі даних і дозволяє однозначно ідентифікувати кожного співробітника.

Name - властивість, яка містить ім'я співробітника. Це поле дозволяє зберігати та відображати інформацію про ім'я співробітника у системі.

Email - властивість, яка містить електронну пошту співробітника. Використання цього поля дозволяє забезпечити можливість зв'язку зі співробітником через електронну пошту.

Phone - властивість, яка містить номер телефону співробітника. Це поле дозволяє зберігати контактну інформацію та забезпечити можливість зв'язку зі співробітником за допомогою телефону.

Клас Employee має модифікатор доступу public, що означає, що він доступний для використання у всій програмі. Використання атрибута [Key] перед полем Id вказує, що це поле є первинним ключем таблиці в базі даних. Кожен інший атрибут { get; set; } вказує на те, що для цього поля доступні методи читання та запису.

2.3 Модель Project

Клас Project визначає сутність проекту у системі "Менеджер завдань". Вона отримала необхідні властивості (рис. 2.3).

```
public class Country : BaseModel
{
    4 usages
    public string Name { get; set; }
    1 usage
    public long Population { get; set; }

    1 usage
    public List<Footballer> Footballers { get; set; }
}
```

Рисунок 2.3 – Модель Project

Id - унікальний ідентифікатор проекту. Це поле використовується як первинний ключ в базі даних для ідентифікації кожного окремого проекту. Ідентифікатор представлений у форматі GUID, що забезпечує унікальність та невідтворюваність значень.

Name - рядок, що містить назву проекту. Це поле дозволяє ідентифікувати проект та відображати його назву в інтерфейсі користувача.

Client - рядок, що містить ім'я клієнта або організації, яка замовила проект. Це поле дозволяє вказати, для кого конкретно виконується проект, що може бути корисною інформацією для команди проекту.

Budget - дійсне число, яке представляє бюджет проекту. Це поле визначає фінансові обмеження або ресурси, які доступні для виконання проекту. Інформація про бюджет може бути використана для керування витратами та ресурсами в процесі виконання проекту.

Клас Project також має атрибут [Key], що вказує, що поле Id є первинним ключем таблиці в базі даних. Усі поля класу мають модифікатор

доступу public, що означає їхню доступність для зовнішнього використання та модифікації. Це дозволяє зручно працювати з об'єктами проекту у різних частинах програми.

2.4 Модель Issue

Клас Issue представляє сутність "Задача" у системі "Менеджер завдань" (рис. 2.4).

Id: Унікальний ідентифікатор задачі. Це поле використовується як первинний ключ в базі даних для ідентифікації кожної окремої задачі. Ідентифікатор представлений у форматі GUID, що забезпечує унікальність та невідтворюваність значень.

Summary - рядок, що містить короткий опис задачі. Це поле дозволяє швидко ідентифікувати суть задачі без необхідності детального аналізу.

Description - рядок, що містить докладний опис задачі. Це поле призначене для зберігання детальної інформації про задачу, що допомагає зрозуміти її суть та контекст.

Priority - рядок, що визначає пріоритет задачі. Це поле дозволяє встановити важливість та терміновість виконання задачі, наприклад, "Високий", "Середній", "Низький" тощо.

EmployeeId - унікальний ідентифікатор співробітника, який відповідає за виконання задачі. Це поле є зовнішнім ключем і посилається на Id співробітника.

Employee - зв'язок з об'єктом співробітника, який відповідає за виконання задачі. Це поле представляє собою зв'язок багато до одного (багато задач можуть бути пов'язані з одним співробітником).

```

public class Issue
{
    [Key]
    public Guid Id { get; set; }

    public string Summary { get; set; }

    public string Description { get; set; }

    public string Priority { get; set; }

    public Guid? EmployeeId { get; set; }
    [JsonIgnore]
    public Employee Employee { get; set; }

    public Guid? ProjectId { get; set; }
    [JsonIgnore]
    public Project Project { get; set; }

    public DateTime Created { get; set; }
}

```

Рисунок 2.4 – Модель Issue

ProjectId - унікальний ідентифікатор проекту, до якого відноситься задача. Це поле є зовнішнім ключем і посилається на Id проекту.

Project - зв'язок з об'єктом проекту, до якого відноситься задача. Це поле представляє собою зв'язок багато до одного (багато задач можуть бути пов'язані з одним проектом).

Created - дата та час створення задачі. Це поле визначає час створення запису про задачу у системі.

2.6 Проектування контролерів

Контролери в ASP.NET Core MVC відповідають за обробку вхідних HTTP-запитів від користувачів та відправку відповідей. Вони є важливою складовою архітектури веб-додатків, оскільки визначають логіку взаємодії

між клієнтом та сервером. У вашому проєкті було створено 4 контролери: TransferController, FootballerController, CountryController та ClubController. Кожен з цих контролерів відповідає за обробку запитів, пов'язаних з певною моделлю даних, яка зберігається в базі даних та керується через інтерфейс користувача. Це дозволяє організувати логічну структуру вашого додатку та розділити функціональність на окремі частини для кращого управління та розвитку програми.

Контролери HomeController (рис. 2.5), IssueController та EmployeeController відповідають за усі CRUD операції з відповідними моделями за допомогою представлень, які вони надають. Представлені відповідають назві методів.

```
public class HomeController : Controller
{
    private readonly IRepository _issueRepository;
    private readonly IRepository _projectRepository;
    private readonly IRepository _employeeRepository;

    public HomeController(IRepository issueRepository,
        IRepository projectRepository, IRepository employeeRepository)

    [6 usages]
    public IActionResult Index(Guid? id, string filterType){...}

    [1 usage]
    public IActionResult DeleteIssue(Guid id){...}

    [1 usage]
    public IActionResult EditIssue(Guid id){...}

    public IActionResult EditedIssue(Issue issue){...}

    [1 usage]
    public IActionResult AddIssue(){...}

    public IActionResult AddedIssue(Issue issue){...}
}
```

Рисунок 2.5 – Інтерфейс класу HomeController

Для роботи класу HomeController були описані необхідні методи. Контролер HomeController у програмі відповідає за обробку запитів, пов'язаних із головною сторінкою та управлінням задачами.

Index(Guid? id, string filterType) - цей метод відображає головну сторінку додатка. Він приймає параметри id і filterType, щоб фільтрувати список задач за проектом або співробітником, якщо такий фільтр встановлено. Потім він відображає список задач за відповідним фільтром.

DeleteIssue(Guid id) видаляє задачу з вказаним ідентифікатором id. Після видалення він перенаправляє користувача на головну сторінку додатка.

EditIssue(Guid id) відображає форму для редагування задачі з вказаним ідентифікатором id. Він також передає в представлення списки проектів та співробітників для використання у випадючих списках.

EditedIssue(Issue issue) приймає відредаговану задачу і оновлює її в репозиторії. Після цього він перенаправляє користувача на головну сторінку додатка.

AddIssue() відображає форму для додавання нової задачі. Він також передає в представлення списки проектів та співробітників для використання у випадючих списках.

AddedIssue(Issue issue) приймає нову задачу, додає її до репозиторію з задачами і перенаправляє користувача на головну сторінку додатка.

Контролер EmployeeController (рис. 2.6) відповідає за обробку запитів, пов'язаних із управлінням співробітниками в системі.


```

public class EmployeeController : Controller
{
    private readonly EmployeeRepository _employeeRepository;

    public EmployeeController(EmployeeRepository employeeRepository)

    [4 usages]
    public IActionResult Index(){...}

    [1 usage]
    public IActionResult DeleteEmployee(Guid id){...}

    [1 usage]
    public IActionResult EditEmployee(Guid id){...}

    public IActionResult EditedEmployee(Employee employee){...}

    [1 usage]
    public IActionResult AddEmployee(){...}

    public IActionResult AddedEmployee(Employee employee){...}
}

```

Рисунок 2.6 – Інтерфейс класу EmployeeController

Index() – метод, що відображає головну сторінку управління співробітниками, де користувач може переглянути список усіх співробітників.

DeleteEmployee(Guid id) видаляє співробітника з вказаним ідентифікатором id. Після видалення він перенаправляє користувача на головну сторінку управління співробітниками.

EditEmployee(Guid id) відображає форму для редагування інформації про співробітника з вказаним ідентифікатором id.

EditedEmployee(Employee employee) приймає відредаговані дані про співробітника і оновлює їх у репозиторії. Після цього він перенаправляє користувача на головну сторінку управління співробітниками.

AddEmployee() відображає форму для додавання нового співробітника.

AddedEmployee(Employee employee) приймає дані про нового співробітника, додає їх у репозиторій та перенаправляє користувача на головну сторінку управління співробітниками.

Контролер ProjectController (рис. 2.7) відповідає за обробку запитів, пов'язаних із управлінням проектами в системі.

```
public class EmployeeController : Controller
{
    private readonly EmployeeRepository _employeeRepository;

    public EmployeeController(EmployeeRepository employeeRepository)

    [4 usages]
    public IActionResult Index() {...}

    [1 usage]
    public IActionResult DeleteEmployee(Guid id) {...}

    [1 usage]
    public IActionResult EditEmployee(Guid id) {...}

    public IActionResult EditedEmployee(Employee employee) {...}

    [1 usage]
    public IActionResult AddEmployee() {...}

    public IActionResult AddedEmployee(Employee employee) {...}
}
```

Рисунок 2.7 – Інтерфейс класу ProjectController

Index() - відображає головну сторінку управління проектами, де користувач може переглянути список усіх проектів.

DeleteProject(Guid id) - видаляє проект з вказаним ідентифікатором id та перенаправляє користувача на головну сторінку управління проектами.

EditProject(Guid id) - відображає форму для редагування інформації про проект з вказаним ідентифікатором id.

EditedProject(Project project) - приймає відредаговані дані про проект і оновлює їх у репозиторії, після чого перенаправляє користувача на головну сторінку управління проектами.

AddProject() - відображає форму для додавання нового проекту.

AddedProject(Project project) - приймає дані про новий проект, додає їх у репозиторій та перенаправляє користувача на головну сторінку управління проектами.

Ці методи дозволяють користувачам взаємодіяти з інформацією про проекти у додатку, включаючи перегляд, редагування, видалення та додавання нових записів. Крім того, вони використовують репозиторій ProjectRepository для доступу до даних та виконання необхідних операцій.

2.7 Проектування бази даних

Вибір бази даних MySQL для цього проекту може бути пояснений кількома факторами. По-перше, MySQL є відкритою та безкоштовною реляційною базою даних, що робить її привабливим варіантом для проектів з обмеженим бюджетом або для стартапів. По-друге, MySQL є однією з найпопулярніших баз даних у світі, що означає наявність великої кількості матеріалів, документації та спільноти розробників для отримання допомоги та підтримки.

Крім того, MySQL добре відома своєю високою продуктивністю та здатністю до масштабування, що важливо для проектів з великою кількістю операцій з базою даних та потенційно великою кількістю користувачів. Нарешті, MySQL має добре підтримуваний драйвер EF Core для спільної роботи з ASP.NET Core, що робить його привабливим вибором для проектів, побудованих на платформі ASP.NET Core. Таким чином, вибір MySQL для цього проекту може бути обумовлений комбінацією його відкритості, популярності, продуктивності та сумісності зі стеком технологій, які використовуються у проекті.

База даних для системи управління завданнями (task_management) містить три таблиці: employee, project та task.

Таблиця employee містить поля Id (унікальний ідентифікатор співробітника), Name (ім'я співробітника), Email (електронна адреса співробітника) та Phone (номер телефону співробітника), з примарним ключем на поле Id.

Таблиця project містить поля Id (унікальний ідентифікатор проекту), Name (назва проекту), Client (клієнт проекту) та Budget (бюджет проекту), з примарним ключем на поле Id.

Таблиця task містить поля Id (унікальний ідентифікатор завдання), Summary (короткий опис завдання), Description (детальний опис завдання), Priority (пріоритет завдання), EmployeeId (зовнішній ключ, посилається на Id співробітника в таблиці employee), ProjectId (зовнішній ключ, посилається на Id проекту в таблиці project), Created (дата створення завдання), з примарним ключем на поле Id. Зовнішні ключі EmployeeId та ProjectId в таблиці task забезпечують зв'язок між завданнями, співробітниками та проектами.

Така структура бази даних дозволяє зберігати і керувати інформацією про співробітників, проекти та завдання в системі управління завданнями.

Для взаємодії з базою даних через код було створено TaskDbContext (рис. 2.8), що є спадкомцем DbContext з фреймворку Entity Framework.

```
public class TaskDbContext : DbContext
{
    [4 usages]
    public DbSet<Employee> Employees { get; set; }
    [4 usages]
    public DbSet<Project> Projects { get; set; }
    [4 usages]
    public DbSet<Issue> Issues { get; set; }

    public TaskDbContext(DbContextOptions options) : base(options) { ... }

    protected override void OnModelCreating(ModelBuilder modelBuilder) {
    }
}
```

Рисунок 2.8 – Інтерфейс класу TaskDbContext

Клас `TaskDbContext` визначає контекст бази даних для зберігання даних про співробітників, проекти та завдання. Ось опис його основних властивостей та методів.

`Employees` - Властивість типу `DbSet<Employee>`, яка представляє набір співробітників у базі даних.

`Projects` - Властивість типу `DbSet<Project>`, яка представляє набір проектів у базі даних.

`Issues` - Властивість типу `DbSet<Issue>`, яка представляє набір завдань у базі даних.

`TaskDbContext(DbContextOptions options)` - Конструктор, який приймає параметр `options` типу `DbContextOptions` і передає його базовому класу `DbContext`.

`OnModelCreating(ModelBuilder modelBuilder)` - Метод, в якому визначається конфігурація моделей для Code First підходу. У цьому методі виконуються налаштування для відображення моделей на таблиці бази даних, а також встановлюються відносини між сутностями. Наприклад, встановлюються зовнішні ключі, а також правила видалення та оновлення даних в зв'язаних таблицях.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Опис використаних стандартних бібліотек

У роботі була використана певна кількість стандартних бібліотек. Бібліотека `Pomelo.EntityFrameworkCore.MySql` є реалізацією пакета `Entity Framework Core` для взаємодії з базами даних `MySQL` в середовищі розробки на платформі `.NET`. `Entity Framework Core (EF Core)` надає зручний і об'єктно-орієнтований спосіб взаємодії з базами даних, а `Pomelo.EntityFrameworkCore.MySql` розширює ці можливості, дозволяючи розробникам працювати з `MySQL` за допомогою `EF Core`.

Ця бібліотека підтримує стандартні функції `Entity Framework Core`, забезпечуючи високу продуктивність та оптимізацію для роботи з `MySQL`. Вона інтегрується з проектами `.NET` безпосередньо і пропонує ряд переваг, включаючи підтримку останніх версій `MySQL`, зручний механізм міграцій для зміни схеми бази даних, а також активну спільноту розробників, що підтримує і розвиває цей інструмент.

Загалом, `Pomelo.EntityFrameworkCore.MySql` дозволяє зручно та ефективно працювати з базами даних `MySQL` в рамках проектів `.NET`, сприяючи швидкому розгортанню та розвитку додатків з використанням цих технологій.

3.2 Опис представлень та візуального інтерфейсу

Найголовнішим представленням проекту є представлення під назвою `_Layout.cshtml`. У ньому описана основна розмітка сторінок. Тобто та, яка буде присутня на кожній сторінці. Це для треба для того, щоб, наприклад, відображати заголовок (рис. 3.1) і не переписувати його в кожному новому представленні.

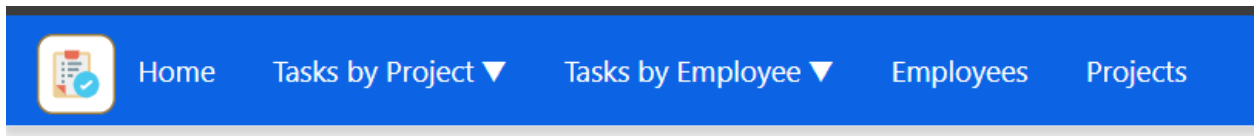


Рисунок 3.1 – Заголовок на сайті

Варту звернути увагу, що на заголовку присутні фільтрації задач по робітникам та проектам (рис. 3.2). Це реалізовано за допомогою таких елементів фреймворку, як компоненти.

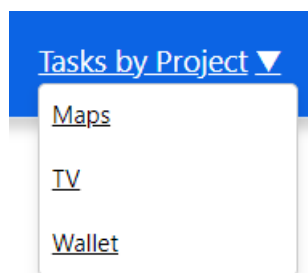


Рисунок 3.1 – Фільтр по проектам

Представлення `Index.cshtml` контролеру `HomeController` було створено для того, щоб відображати усіх наявні task та їхні основні характеристики або є таски за параметрами фільтрів. Фактично воно відображає головну сторінку сайту (рис. 3.3). Для цього є кнопки на плитках з задачами та кнопка зліва від таблиці для додавання нових задач.

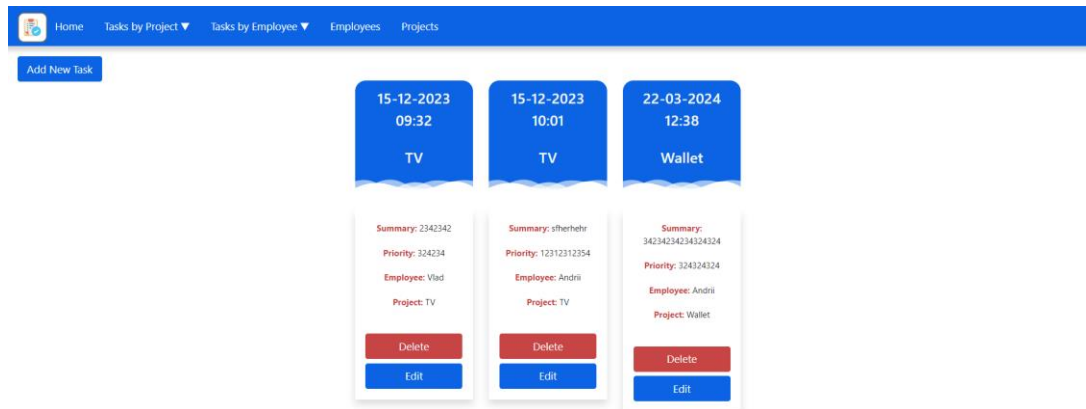
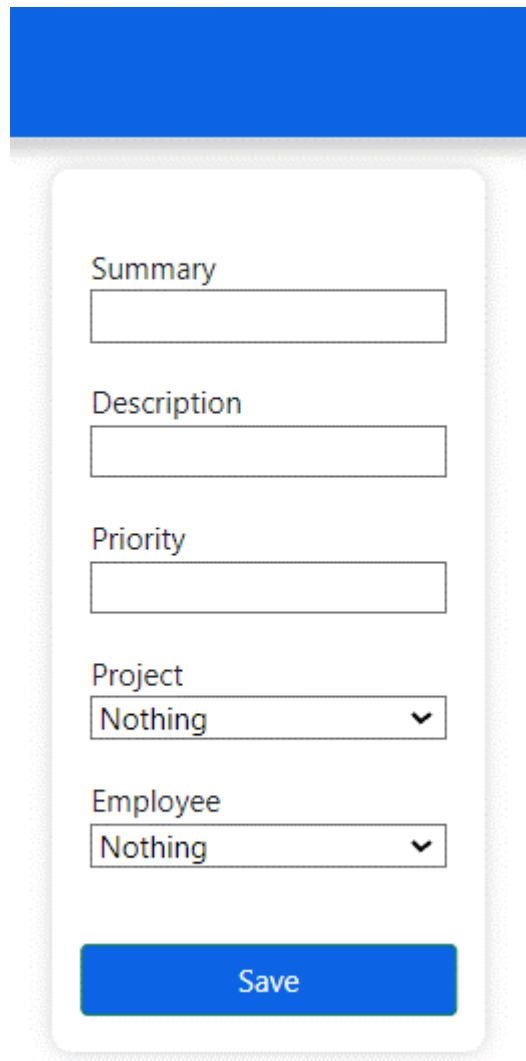


Рисунок 3.3 – Головна сторінка

Для редагування та додання задач існує представлення `EditIssue.cshtml`. Воно представляє собою набір усіх полів та лейблів, а також кнопку збереження (рис. 3.4).

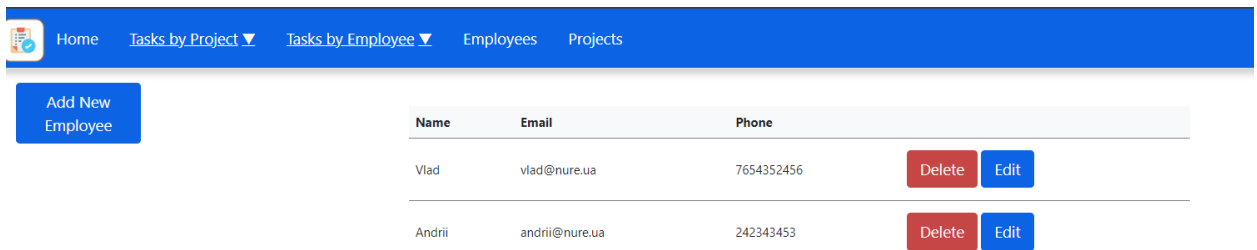


The image shows a mobile application interface for editing a task. It features a blue header bar at the top. Below it is a white rounded rectangle containing the form. The form has five input fields: 'Summary', 'Description', 'Priority', 'Project', and 'Employee'. The 'Project' and 'Employee' fields are dropdown menus currently showing 'Nothing'. At the bottom of the form is a blue 'Save' button.

Summary
Description
Priority
Project Nothing
Employee Nothing
Save

Рисунок 3.4 – Вікно редагування задачі

Наступним представлення є `Index.cshtml` контролеру `EmployeeController`. Він потрібний для відображення усіх робітників та для операцій з ними (рис. 3.5).




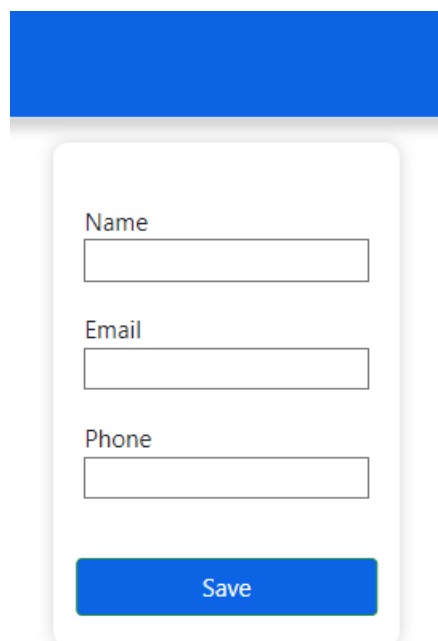
 Home Tasks by Project Tasks by Employee Employees Projects				
Add New Employee				
Name	Email	Phone		
Vlad	vlad@nure.ua	7654352456	Delete	Edit
Andrii	andrii@nure.ua	242343453	Delete	Edit

Рисунок 3.5 – Таблиця робітників

Звісно є у робітника є своя форма для створення та редагування, за яку відповідає EditEmployee.cshtml (рис. 3.6)



Name

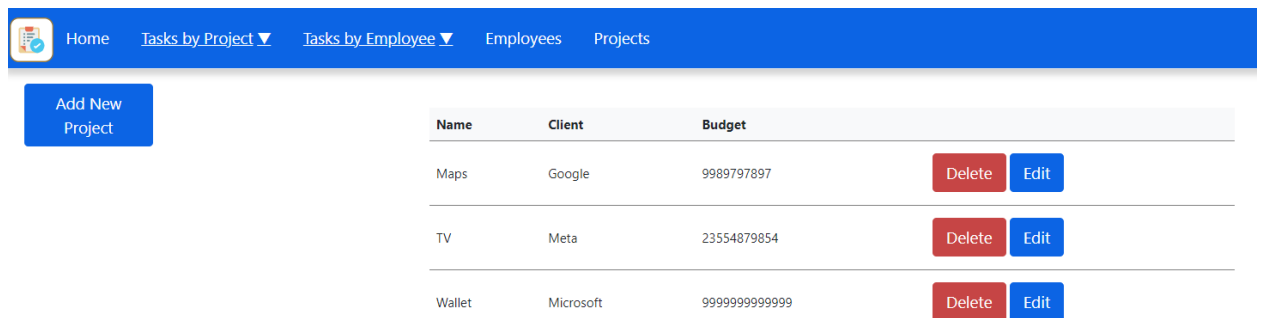
Email

Phone

Save

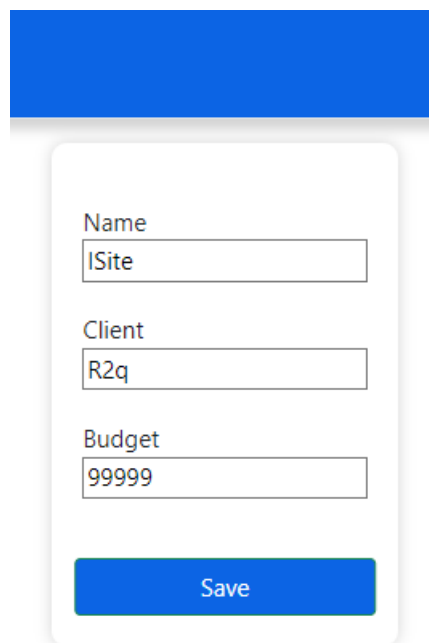
Рисунок 3.6 – Форма додавання робітника

Далі йде Index.cshtml (рис. 3.7) та EditProject.cshtml (рис. 3.8) для редагування проектів, аналогічно з робітниками.



Name	Client	Budget		
Maps	Google	9989797897	Delete	Edit
TV	Meta	23554879854	Delete	Edit
Wallet	Microsoft	999999999999	Delete	Edit

Рисунок 3.7 – Таблиця проектів



Name
ISite

Client
R2q

Budget
99999

Save

Рисунок 3.8 – Редагування проекту

ВИСНОВКИ

У ході курсової роботи "Використання технології ASP.NET Core MVC для розробки сайту «Менеджер завдань»" мною було реалізовано веб-додаток, який дозволяє ефективно керувати завданнями, співробітниками та проектами в організації. За допомогою технології ASP.NET Core MVC та бази даних MySQL було створено функціональний і зручний у використанні інструмент, що забезпечує швидке розгортання та надійну роботу з даними.

У рамках розробленого додатку реалізовано можливість пошуку завдань за фільтрами, такими як співробітник або проект, що дозволяє знайти необхідну інформацію швидко та ефективно. Також реалізовано функціонал редагування усіх сутностей: задач, співробітників та проектів, що дає можливість змінювати та оновлювати дані відповідно до потреб користувача.

Отримані результати свідчать про успішне виконання поставлених завдань та досягнення мети розробленого веб-додатку. Використання технології ASP.NET Core MVC та бази даних MySQL дозволило створити потужний та функціональний інструмент для управління завданнями та проектами в організації.

Проте, є можливість подальшого удосконалення роботи. Наприклад, можна розширити функціонал додатку, додавши можливість призначення та відстеження пріоритету завдань, а також реалізувати механізм сповіщень для користувачів про нагальні або важливі завдання. Також можна покращити інтерфейс користувача, зробивши його ще більш зручним та інтуїтивно зрозумілим для користувачів.

Загалом, розроблений веб-додаток відповідає вимогам та надає значну користь для організації, проте його можна подальшу вдосконалити для досягнення ще більшої ефективності та зручності у використанні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ЗВІТИ У СФЕРІ НАУКИ І ТЕХНІКИ Структура та правила оформлювання. [На заміну ДСТУ 3008-95; чинний від 2017-07-01]. Вид. офіц. Київ, 2016. 31 с. (Інформація та документація) (дата звернення: 09.03.2024).
2. Web Database Access Technology Based on ASP.NET // Springer. URL: https://link.springer.com/chapter/10.1007/978-3-642-25989-0_43 (дата звернення: 09.03.2024).
3. ASP.NET AND ITS FRAMEWORK // IJIRT. URL: <https://ijirt.org/Article?manuscript=142726> (дата звернення: 05.03.2024).
4. Beginner to Entity Framework // Stack Overflow. URL: <https://stackoverflow.com/questions/42023096/beginner-to-entity-framework> (дата звернення: 05.03.2024).
5. Entity Framework // Wikipedia. URL: https://en.wikipedia.org/wiki/Entity_Framework (дата звернення: 05.03.2024).
6. .NET // Microsoft. URL: <https://dotnet.microsoft.com/en-us/download/dotnet> (дата звернення: 01.03.2024).
7. Design Patterns Explained – Dependency Injection with Code Examples // Stackify. URL: <https://stackify.com/dependency-injection/> (дата звернення: 01.03.2024).
8. Документація ASP.NET Core // Microsoft. URL: <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0> (дата звернення: 02.03.2024).
9. ASP and ASP.NET Tutorials // W3 Schools. URL: <https://github.com/POPWorldMedia/POPForums> (дата звернення: 02.03.2024).
10. Why is RemoveRange not clearing DbSet? // Stack Overflow. URL: <https://www.w3schools.com/asp/default.ASP> (дата звернення: 02.03.2024).

ДОДАТОК А КОД

```

using Microsoft.AspNetCore.Mvc;
using TaskManagement.File;
using TaskManagement.Models;

namespace TaskManagement.Controllers;

public class EmployeeController : Controller
{
    private readonly EmployeeRepository _employeeRepository;

    public EmployeeController(EmployeeRepository employeeRepository)
    {
        _employeeRepository = employeeRepository;
    }

    public IActionResult Index()
    {
        return View(_employeeRepository.Employees);
    }

    public IActionResult DeleteEmployee(Guid id)
    {
        _employeeRepository.RemoveEmployee(id);
        return RedirectToAction("Index");
    }

    public IActionResult EditEmployee(Guid id)
    {
        var employee = _employeeRepository.GetEmployee(id);
        return View(employee);
    }

    public IActionResult EditedEmployee(Employee employee)
    {
        _employeeRepository.UpdateEmployee(employee);
        return RedirectToAction("Index");
    }

    public IActionResult AddEmployee()
    {
        return View("EditEmployee", new Employee());
    }

    public IActionResult AddedEmployee(Employee employee)
    {
        employee.Id = Guid.NewGuid();
        _employeeRepository.AddEmployee(employee);
        return RedirectToAction("Index");
    }
}

using Microsoft.AspNetCore.Mvc;
using TaskManagement.File;
using TaskManagement.Models;

namespace TaskManagement.Controllers;

public class HomeController : Controller
{
    private readonly IssueRepository _issueRepository;
    private readonly ProjectRepository _projectRepository;
    private readonly EmployeeRepository _employeeRepository;

    public HomeController(IssueRepository issueRepository,
        ProjectRepository projectRepository, EmployeeRepository employeeRepository)

```

```

{
    _issueRepository = issueRepository;
    _projectRepository = projectRepository;
    _employeeRepository = employeeRepository;
}

public IActionResult Index(Guid? id, string filterType)
{
    var issues = _issueRepository.Issues;

    if (filterType == "Project")
    {
        issues = issues.Where(issue => issue.ProjectId == id).ToList();
    }

    if (filterType == "Employee")
    {
        issues = issues.Where(issue => issue.EmployeeId == id).ToList();
    }

    return View(issues);
}

public IActionResult DeleteIssue(Guid id)
{
    _issueRepository.RemoveIssue(id);
    return RedirectToAction("Index");
}

public IActionResult EditIssue(Guid id)
{
    ViewBag.Projects = _projectRepository.GetProjectsListItems();
    ViewBag.Employees = _employeeRepository.GetEmployeesListItems();
    var issue = _issueRepository.GetIssue(id);
    return View(issue);
}

public IActionResult EditedIssue(Issue issue)
{
    _issueRepository.UpdateIssue(issue);
    return RedirectToAction("Index");
}

public IActionResult AddIssue()
{
    ViewBag.Projects = _projectRepository.GetProjectsListItems();
    ViewBag.Employees = _employeeRepository.GetEmployeesListItems();
    return View("EditIssue", new Issue());
}

public IActionResult AddedIssue(Issue issue)
{
    issue.Id = Guid.NewGuid();
    issue.Created = DateTime.Now;
    _issueRepository.AddIssue(issue);
    return RedirectToAction("Index");
}
}

using Microsoft.AspNetCore.Mvc;
using TaskManagement.File;
using TaskManagement.Models;

namespace TaskManagement.Controllers;

public class ProjectController : Controller
{
    private readonly ProjectRepository _projectRepository;

```

```

public ProjectController(ProjectRepository projectRepository)
{
    _projectRepository = projectRepository;
}

public IActionResult Index()
{
    return View(_projectRepository.Projects);
}

public IActionResult DeleteProject(Guid id)
{
    _projectRepository.RemoveProject(id);
    return RedirectToAction("Index");
}

public IActionResult EditProject(Guid id)
{
    var project = _projectRepository.GetProject(id);
    return View(project);
}

public IActionResult EditedProject(Project project)
{
    _projectRepository.UpdateProject(project);
    return RedirectToAction("Index");
}

public IActionResult AddProject()
{
    return View("EditProject", new Project());
}

public IActionResult AddedProject(Project project)
{
    project.Id = Guid.NewGuid();
    _projectRepository.AddProject(project);
    return RedirectToAction("Index");
}
}

using Microsoft.EntityFrameworkCore;
using TaskManagement.Models;

namespace TaskManagement.Db;

public class TaskDbContext : DbContext
{
    public DbSet<Employee> Employees { get; set; }
    public DbSet<Project> Projects { get; set; }
    public DbSet<Issue> Issues { get; set; }

    public TaskDbContext(DbContextOptions options) : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<Issue>(entry =>
        {
            entry.ToTable("task");

            entry.HasOne(t => t.Employee)
                .WithMany()
                .HasForeignKey(t => t.EmployeeId)
                .OnDelete(DeleteBehavior.SetNull);
        });
    }
}

```



```

        entry.HasOne(t => t.Project)
            .WithMany()
            .HasForeignKey(t => t.ProjectId)
            .OnDelete(DeleteBehavior.SetNull);
    });

    modelBuilder.Entity<Project>(entry => { entry.ToTable("project"); });
    modelBuilder.Entity<Employee>(entry => { entry.ToTable("employee"); });
}

}

using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using TaskManagement.Db;
using TaskManagement.Models;

namespace TaskManagement.File;

public class EmployeeRepository
{
    private readonly TaskDbContext _dbContext;

    public List<Issue> Issues => _dbContext.Issues
        .Include(i => i.Employee)
        .Include(i => i.Project)
        .OrderBy(i => i.Created)
        .ToList();

    public List<Employee> Employees => _dbContext.Employees.ToList();
    public List<Project> Projects => _dbContext.Projects.ToList();

    public EmployeeRepository(TaskDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public List<SelectListItem> GetEmployeesListItems()
    {
        List<SelectListItem> selectListItems = new List<SelectListItem>();
        foreach (var employee in Employees)
        {
            selectListItems.Add(new (employee.Name, employee.Id.ToString()));
        }
        return selectListItems;
    }

    public void RemoveEmployee(Guid id)
    {
        var employee = _dbContext.Employees.Find(id);
        _dbContext.Remove(employee);
        _dbContext.SaveChanges();
    }

    public void AddEmployee(Employee employee)
    {
        _dbContext.Add(employee);
        _dbContext.SaveChanges();
    }

    public void UpdateEmployee(Employee employee)
    {
        _dbContext.Update(employee);
        _dbContext.SaveChanges();
    }

    public Employee GetEmployee(Guid id)
    {
        return Employees.FirstOrDefault(x => x.Id == id);
    }
}

```

```

    }
}

using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Newtonsoft.Json;
using TaskManagement.Db;
using TaskManagement.Models;

namespace TaskManagement.File;

public class IssueRepository
{
    private readonly TaskDbContext _dbContext;

    public List<Issue> Issues => _dbContext.Issues
        .Include(i => i.Employee)
        .Include(i => i.Project)
        .OrderBy(i => i.Created)
        .ToList();
    public List<Employee> Employees => _dbContext.Employees.ToList();
    public List<Project> Projects => _dbContext.Projects.ToList();

    public IssueRepository(TaskDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public List<SelectListItem> GetIssuesListItems()
    {
        List<SelectListItem> selectListItems = new List<SelectListItem>();
        foreach (var issue in Issues)
        {
            selectListItems.Add(new (issue.Summary, issue.Id.ToString()));
        }
        return selectListItems;
    }

    public void RemoveIssue(Guid id)
    {
        var issue = _dbContext.Issues.Find(id);
        _dbContext.Remove(issue);
        _dbContext.SaveChanges();
    }

    public void AddIssue(Issue issue)
    {
        _dbContext.Add(issue);
        _dbContext.SaveChanges();
    }

    public void UpdateIssue(Issue issue)
    {
        _dbContext.Update(issue);
        _dbContext.SaveChanges();
    }

    public Issue GetIssue(Guid id)
    {
        return Issues.FirstOrDefault(x => x.Id == id);
    }
}

using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using TaskManagement.Db;
using TaskManagement.Models;

```

```

namespace TaskManagement.File;

public class ProjectRepository
{
    private readonly TaskDbContext _dbContext;

    public List<Issue> Issues => _dbContext.Issues
        .Include(i => i.Project)
        .Include(i => i.Project)
        .OrderBy(i => i.Created)
        .ToList();
    public List<Project> Projects => _dbContext.Projects.ToList();
    public List<Employee> Employees => _dbContext.Employees.ToList();

    public ProjectRepository(TaskDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public List<SelectListItem> GetProjectsListItems()
    {
        List<SelectListItem> selectListItems = new List<SelectListItem>();
        foreach (var project in Projects)
        {
            selectListItems.Add(new (project.Name, project.Id.ToString()));
        }
        return selectListItems;
    }

    public void RemoveProject(Guid id)
    {
        var project = _dbContext.Projects.Find(id);
        _dbContext.Remove(project);
        _dbContext.SaveChanges();
    }

    public void AddProject(Project project)
    {
        _dbContext.Add(project);
        _dbContext.SaveChanges();
    }

    public void UpdateProject(Project project)
    {
        _dbContext.Update(project);
        _dbContext.SaveChanges();
    }

    public Project GetProject(Guid id)
    {
        return Projects.FirstOrDefault(x => x.Id == id);
    }
}

using System.ComponentModel.DataAnnotations;

namespace TaskManagement.Models;

public class Employee
{
    [Key]
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
}

```

```

using System.ComponentModel.DataAnnotations;
using System.Text.Json.Serialization;

namespace TaskManagement.Models;

public class Issue
{
    [Key]
    public Guid Id { get; set; }
    public string Summary { get; set; }
    public string Description { get; set; }
    public string Priority { get; set; }

    public Guid? EmployeeId { get; set; }
    [JsonIgnore]
    public Employee Employee { get; set; }

    public Guid? ProjectId { get; set; }
    [JsonIgnore]
    public Project Project { get; set; }
    public DateTime Created { get; set; }
}

using System.ComponentModel.DataAnnotations;

namespace TaskManagement.Models;

public class Project
{
    [Key]
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Client { get; set; }
    public double Budget { get; set; }
}

using System.Linq.Expressions;
using Microsoft.AspNetCore.Html;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace TaskManagement.MyHelpers;

public static class MyHelpers
{
    public static IHtmlContent MyTextBox<TModel, TValue>(this IHtmlHelper<TModel>
htmlHelper,
        Expression<Func<TModel, TValue>> expression, string labelText)
    {
        var label = $"<div><label>{ labelText }</label></div><div>";
        var input = htmlHelper.TextBoxFor(expression);

        var tbContainer = new TagBuilder("div");
        tbContainer.MergeAttribute("style", "margin-top: 20px");
        tbContainer.InnerHtml.AppendHtml(label);
        tbContainer.InnerHtml.AppendHtml(input);
        tbContainer.InnerHtml.AppendHtml("</div>");
        return tbContainer;
    }

    public static IHtmlContent MyTableLi(this IHtmlHelper htmlHelper, string text,
object value)
    {
        var textTag = $"<b>{ text }</b>";
        var valueTag = value?.ToString();

        var liContainer = new TagBuilder("li");

```

```

        liContainer.InnerHtml.AppendHtml(textTag);
        liContainer.InnerHtml.AppendHtml(valueTag ?? "");
        return liContainer;
    }

    public static IHtmlContent MyTableH3(this IHtmlHelper htmlHelper, object value)
    {
        var valueTag = value?.ToString();

        var liContainer = new TagBuilder("h3");
        liContainer.AddCssClass("text-light");
        liContainer.InnerHtml.AppendHtml(valueTag ?? "");
        return liContainer;
    }
}

using Microsoft.AspNetCore.Mvc;
using TaskManagement.File;

namespace TaskManagement.Components
{
    public class TasksByEmployee : ViewComponent
    {
        private readonly EmployeeRepository _employeeRepository;
        public TasksByEmployee(EmployeeRepository employeeRepository)
        {
            _employeeRepository = employeeRepository;
        }

        public IViewComponentResult Invoke()
        {
            var projects = _employeeRepository.Employees.OrderBy(p => p.Name);
            return View(projects);
        }
    }
}

using Microsoft.AspNetCore.Mvc;
using TaskManagement.File;

namespace TaskManagement.Components
{
    public class TasksByProject : ViewComponent
    {
        private readonly ProjectRepository _projectRepository;
        public TasksByProject(ProjectRepository projectRepository)
        {
            _projectRepository = projectRepository;
        }

        public IViewComponentResult Invoke()
        {
            var projects = _projectRepository.Projects.OrderBy(p => p.Name);
            return View(projects);
        }
    }
}

```