Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Інформатики та інтелектуальної власності"

**Звіт**

**Лабораторна робота 6**

з дисципліни "Проєктування серверних застосунків"

Виконав: студент групи КН-922В

Шершньов М.О.

Перевірив:

Івашко А.В.

Харків 2025

Зміст

[Завдання 3](#_Toc194430112)

[Класи weaktypization, stricttypization, dataadding, datadeleting, specificdeleting, dataediting, mdfdatabase, databaseinitializing, subjectadding 5](#_Toc194430113)

[Клас WeakTypization 5](#_Toc194430114)

[Клас StrictTypization 6](#_Toc194430115)

[Клас DataAdding 8](#_Toc194430116)

[Клас DataDeleting 10](#_Toc194430117)

[Клас SpecificDeleting 11](#_Toc194430118)

[Клас DataEditing 13](#_Toc194430119)

[Клас MDFDatabase 15](#_Toc194430120)

[Клас DatabaseInitializing 17](#_Toc194430121)

[Клас SubjectAdding 19](#_Toc194430122)

[Висновок 22](#_Toc194430123)

**Використання моделей (model) в MVC WEB-застосунку**

**Мета роботи:** придбати практику у створенні та використанні моделей даних під час обробки WEB-форм та застосування різних засобів обробки БД у WEB-застосунках

Завдання

1. Створити застосунок, який за допомогою слабо типізованої форми (кожне поле окрема змінна в контролері), що відповідає моделі читав би дані з відповідних полів форми та демонстрував би їх на тому ж подані нижче.

Мій варіант:

Изображение выглядит как текст, Шрифт, снимок экрана, число

Контент, сгенерированный ИИ, может содержать ошибки.

1. Виконати попереднє завдання, але замість слабо типізованої форми використати суворо типізовану форму, поля якої відповідають властивостям об'єкту моделі в контролері застосунку.
2. Модифікувати попереднє завдання додавши до форми кнопку додання даних до таблиці відповідної MySQL БД з подальшою демонстрацією вмісту цієї таблиці під формою подання.
3. Модифікувати попереднє завдання додавши до форми кнопку видалення усіх записів таблиці відповідної MySQL БД з подальшою демон-страцією порожньої таблиці під формою подання.
4. \*Модифікувати попереднє завдання додавши до форми засоби видалення бажаного запису з таблиці відповідної MySQL БД з подальшою демонстрацією вмісту таблиці без видаленого запису під формою по-дання.
5. \*Модифікувати попереднє завдання додавши до форми засоби редагування бажаного запису з таблиці відповідної MySQL БД в окремому поданні з подальшою демонстрацією вмісту таблиці без видаленого запису під формою попереднього подання.
6. За допомогою Entity Framework (EF) створити застосунок обробки (додання, редагування, видалення) Microsoft SQL БД, яка зберігається в файлі \*.mdf.
7. \*Модифікувати минуле завдання зробивши початкове заповнення таблиці БД трьома записами з таблиці власного варіанту.
8. Модифікувати минуле завдання додавши до структури таблиці БД будь яке поле за змістом предметної сфери. Перевірити працездатність застосунку

Класи для роботи:

1. «Слабка типізація» - **WeakTypization**
2. «Сувора типізація» – **StrictTypization**
3. «Додавання даних» – **DataAdding**
4. «Видалення даних» – **DataDeleting**
5. «Окреме видалення» – **SpecificDeleting**
6. «Редагування даних» – **DataEditing**
7. «База даних MDF» – **MDFDatabase**
8. «Початкове заповнення» – **DatabaseInitializing**
9. «Додавання предметної сфери» – **SubjectAdding**

Класи weaktypization, stricttypization, dataadding, datadeleting, specificdeleting, dataediting, mdfdatabase, databaseinitializing, subjectadding

Клас WeakTypization

Клас **WeakTypization** працює зі слабко типізованими формами для додавання сільськогосподарських підприємств.

Основні функції класу:

1. **Зберігає список підприємств** у статичному List<dynamic>, дозволяючи зберігати різні типи даних без перевірки на етапі компіляції.
2. **Приймає дані від користувача** через окремі змінні (string name, string ownershipType, int landArea, int employeesCount), а не через строго типізовану модель.
3. **Створює об'єкт dynamic** на основі отриманих значень і додає його до списку підприємств.
4. **Повертає оновлений список** на сторінку Index, де всі додані підприємства відображаються у вигляді таблиці.

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc;

using Task\_1.Models;

namespace Task\_1.Controllers

{

public class HomeController : Controller

{

private static List<AgriculturalEnterprise> \_enterprises = new();

[HttpGet]

public IActionResult Index()

{

return View(\_enterprises);

}

[HttpPost]

public IActionResult Index(string name, string ownershipType, int landArea, int employeesCount)

{

var enterprise = new AgriculturalEnterprise

{

Name = name,

OwnershipType = ownershipType,

LandArea = landArea,

EmployeesCount = employeesCount

};

\_enterprises.Add(enterprise);

return View(\_enterprises);

}

}

}

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунки 6.1-6.2 – Результат роботи класу WeakTypization

Клас StrictTypization

Клас **StrictTypization** працює із суворо типізованою формою для додавання сільськогосподарських підприємств.

Основні функції класу:

1. **Використовує модель EnterpriseFormViewModel**, яка містить список підприємств та об'єкт нового підприємства.
2. **Передає модель у представлення Index**, забезпечуючи строгий контроль над переданими даними.
3. **Перевіряє коректність введених даних (ModelState.IsValid)**, щоб уникнути додавання пустих або некоректних записів.
4. **Додає лише валідні підприємства** до списку \_enterprises та повертає оновлену модель у представлення.

using Microsoft.AspNetCore.Mvc;

using Task\_2.Models;

namespace Task\_2.Controllers

{

public class HomeController : Controller

{

private static List<AgriculturalEnterprise> \_enterprises = new();

[HttpGet]

public IActionResult Index()

{

return View(new EnterpriseFormViewModel { Enterprises = \_enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult Index(EnterpriseFormViewModel model)

{

if (ModelState.IsValid && !string.IsNullOrWhiteSpace(model.NewEnterprise.Name))

{

\_enterprises.Add(model.NewEnterprise);

}

return View(new EnterpriseFormViewModel { Enterprises = \_enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

}

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунки 6.3-6.4 – Результат роботи класу StrictTypization

Клас DataAdding

Клас **DataAdding** керує додаванням підприємств до бази даних за допомогою Entity Framework.

Основні функції класу:

1. **Отримує список підприємств із бази даних (ApplicationDbContext)** та передає їх у представлення.
2. **Додає нове підприємство (NewEnterprise)** до бази даних після перевірки валідності моделі (ModelState.IsValid).
3. **Використовує \_context.SaveChanges()**, щоб зберегти зміни у базі даних.
4. **Оновлює список підприємств після додавання нового запису** та передає його у представлення.

using Microsoft.AspNetCore.Mvc;

using System.Linq;

using Task\_3.Models;

namespace Task\_3.Controllers

{

public class HomeController : Controller

{

private readonly ApplicationDbContext \_context;

public HomeController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult Index(EnterpriseFormViewModel model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Add(model.NewEnterprise);

\_context.SaveChanges();

}

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

}

}

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, линия, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунки 6.5-6.6 – Результат роботи класу DataAdding

Клас DataDeleting

Клас **DataDeletingController** керує додаванням і видаленням підприємств у базі даних за допомогою Entity Framework.

Основні функції класу:

1. **Отримує список підприємств із бази (ApplicationDbContext)** та передає його у представлення.
2. **Додає нове підприємство (NewEnterprise)** до бази даних після перевірки валідності моделі (ModelState.IsValid).
3. **Видаляє всі записи з таблиці підприємств** за допомогою RemoveRange().
4. **Зберігає зміни у базі (\_context.SaveChanges())** та оновлює список записів.
5. **Перенаправляє користувача на оновлену сторінку (RedirectToAction("Index"))** після видалення всіх підприємств.

using Microsoft.AspNetCore.Mvc;

using Task\_4.Models;

namespace Task\_4.Controllers

{

public class HomeController : Controller

{

private readonly ApplicationDbContext \_context;

public HomeController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult Index(EnterpriseFormViewModel model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Add(model.NewEnterprise);

\_context.SaveChanges();

}

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult DeleteAll()

{

\_context.AgriculturalEnterprises.RemoveRange(\_context.AgriculturalEnterprises);

\_context.SaveChanges();

return RedirectToAction("Index");

}

}

}

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.7 – Результат роботи класу DataDeleting

Клас SpecificDeleting

Клас **SpecificDeleting** керує додаванням, переглядом і видаленням підприємств у базі даних за допомогою Entity Framework.

Основні функції класу:

1. **Отримує список підприємств із бази** та передає їх у представлення.
2. **Додає нове підприємство** після перевірки валідності моделі.
3. **Видаляє всі підприємства** за допомогою RemoveRange().
4. **Видаляє конкретне підприємство за id**, якщо запис існує в базі (Find(id)).
5. **Зберігає зміни в базі (\_context.SaveChanges())** та оновлює список підприємств.
6. **Перенаправляє користувача на оновлену сторінку** після видалення запису.

using Microsoft.AspNetCore.Mvc;

using Task\_5.Models;

namespace Task\_5.Controllers

{

public class HomeController : Controller

{

private readonly ApplicationDbContext \_context;

public HomeController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult Index(EnterpriseFormViewModel model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Add(model.NewEnterprise);

\_context.SaveChanges();

}

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult DeleteAll()

{

\_context.AgriculturalEnterprises.RemoveRange(\_context.AgriculturalEnterprises);

\_context.SaveChanges();

return RedirectToAction("Index");

}

[HttpPost]

public IActionResult Delete(int id)

{

var enterprise = \_context.AgriculturalEnterprises.Find(id);

if (enterprise != null)

{

\_context.AgriculturalEnterprises.Remove(enterprise);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

}

}

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.8 – Результат роботи класу SpecificDeleting

Клас DataEditing

Клас **DataEditing** керує додаванням, переглядом, редагуванням і видаленням підприємств у базі даних за допомогою Entity Framework.

Основні функції класу:

1. **Отримує список підприємств із бази** та передає їх у представлення.
2. **Додає нове підприємство**, якщо модель валідна.
3. **Видаляє всі підприємства** (RemoveRange()).
4. **Видаляє конкретне підприємство за id**, якщо запис існує (Find(id)).
5. **Отримує підприємство для редагування** (Edit(int id)) і повертає представлення.
6. **Оновлює дані підприємства** (Update(model)) після перевірки валідності.
7. **Зберігає зміни в базі (\_context.SaveChanges())** та оновлює сторінку.

using Microsoft.AspNetCore.Mvc;

using Task\_6.Models;

namespace Task\_6.Controllers

{

public class HomeController : Controller

{

private readonly ApplicationDbContext \_context;

public HomeController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult Index(EnterpriseFormViewModel model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Add(model.NewEnterprise);

\_context.SaveChanges();

}

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult DeleteAll()

{

\_context.AgriculturalEnterprises.RemoveRange(\_context.AgriculturalEnterprises);

\_context.SaveChanges();

return RedirectToAction("Index");

}

[HttpPost]

public IActionResult Delete(int id)

{

var enterprise = \_context.AgriculturalEnterprises.Find(id);

if (enterprise != null)

{

\_context.AgriculturalEnterprises.Remove(enterprise);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

[HttpGet]

public IActionResult Edit(int id)

{

var enterprise = \_context.AgriculturalEnterprises.Find(id);

if (enterprise == null)

{

return NotFound();

}

return View(enterprise);

}

[HttpPost]

public IActionResult Edit(AgriculturalEnterprise model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Update(model);

\_context.SaveChanges();

return RedirectToAction("Index");

}

return View(model);

}

}

}

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.9 – Результат роботи класу DataEditing

Клас MDFDatabase

Клас **MDFDatabase** керує взаємодією з базою даних (MDF-файлом) для збереження, оновлення та видалення записів про підприємства.

Основні функції класу:

1. **Отримує всі підприємства з бази даних** та передає у представлення (Index()).
2. **Додає нове підприємство**, якщо модель валідна (Add(model.NewEnterprise)).
3. **Видаляє всі підприємства** (RemoveRange()), очищуючи базу.
4. **Видаляє конкретне підприємство за id**, якщо запис існує.
5. **Отримує підприємство для редагування** та повертає його у форму (Edit(int id)).
6. **Оновлює існуюче підприємство** після перевірки валідності (Update(model)).
7. **Використовує Entity Framework для взаємодії з MDF-файлом** через \_context.SaveChanges().

using Microsoft.AspNetCore.Mvc;

using Task\_7.Models;

namespace Task\_7.Controllers

{

public class HomeController : Controller

{

private readonly ApplicationDbContext \_context;

public HomeController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult Index(EnterpriseFormViewModel model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Add(model.NewEnterprise);

\_context.SaveChanges();

}

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult DeleteAll()

{

\_context.AgriculturalEnterprises.RemoveRange(\_context.AgriculturalEnterprises);

\_context.SaveChanges();

return RedirectToAction("Index");

}

[HttpPost]

public IActionResult Delete(int id)

{

var enterprise = \_context.AgriculturalEnterprises.Find(id);

if (enterprise != null)

{

\_context.AgriculturalEnterprises.Remove(enterprise);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

[HttpGet]

public IActionResult Edit(int id)

{

var enterprise = \_context.AgriculturalEnterprises.Find(id);

if (enterprise == null)

{

return NotFound();

}

return View(enterprise);

}

[HttpPost]

public IActionResult Edit(AgriculturalEnterprise model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Update(model);

\_context.SaveChanges();

return RedirectToAction("Index");

}

return View(model);

}

}

}

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, программное обеспечение, Значок на компьютере, Операционная система

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунки 6.10-6.11 – Результат роботи класу MDFDatabase

Клас DatabaseInitializing

**Клас DatabaseInitializing** керує ініціалізацією та налаштуванням бази даних, забезпечуючи, щоб необхідні таблиці та початкові дані були створені при першому запуску програми.

Основні функції класу:

* Перевіряє, чи існує база даних, та створює її, якщо вона відсутня.
* Заповнює базу даних початковими даними, такими як зразки підприємств, для демонстраційних цілей.
* Використовує механізм міграцій Entity Framework для автоматизації оновлення бази даних під час змін у моделях.
* Забезпечує цілісність бази даних, за потреби оновлюючи або видаляючи неналежні дані.
* Інтегрується у застосунок під час старту, налаштовуючи базу даних для подальшого використання.

using Microsoft.AspNetCore.Mvc;

using Task\_8.Models;

namespace Task\_8.Controllers

{

public class HomeController : Controller

{

private readonly ApplicationDbContext \_context;

public HomeController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult Index(EnterpriseFormViewModel model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Add(model.NewEnterprise);

\_context.SaveChanges();

}

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult DeleteAll()

{

\_context.AgriculturalEnterprises.RemoveRange(\_context.AgriculturalEnterprises);

\_context.SaveChanges();

return RedirectToAction("Index");

}

[HttpPost]

public IActionResult Delete(int id)

{

var enterprise = \_context.AgriculturalEnterprises.Find(id);

if (enterprise != null)

{

\_context.AgriculturalEnterprises.Remove(enterprise);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

[HttpGet]

public IActionResult Edit(int id)

{

var enterprise = \_context.AgriculturalEnterprises.Find(id);

if (enterprise == null)

{

return NotFound();

}

return View(enterprise);

}

[HttpPost]

public IActionResult Edit(AgriculturalEnterprise model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Update(model);

\_context.SaveChanges();

return RedirectToAction("Index");

}

return View(model);

}

}

}

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.12 – Результат роботи класу DatabaseInitializing

Клас SubjectAdding

**Клас SubjectAdding** відповідає за додавання нових предметів або тем до системи, забезпечуючи валідацію введених даних і збереження інформації в базі даних.

Основні функції класу:

* Перевіряє, чи модель для нового предмета є валідною перед її збереженням у базі даних.
* Додає новий предмет до контексту бази даних, використовуючи методи Entity Framework для взаємодії з базою даних.
* Забезпечує можливість надання повідомлень про помилки, якщо вхідні дані не відповідають вимогам валідації.
* Повертатиме користувача на сторінку введення з уже введеними даними, щоб виправити помилки, якщо такі є.
* Після успішного додавання предмета перенаправляє на сторінку зі списком всіх предметів, щоб користувач міг побачити результат своїх дій.

using Microsoft.AspNetCore.Mvc;

using Task\_9.Models;

namespace Task\_9.Controllers

{

public class HomeController : Controller

{

private readonly ApplicationDbContext \_context;

public HomeController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult Index(EnterpriseFormViewModel model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Add(model.NewEnterprise);

\_context.SaveChanges();

}

var enterprises = \_context.AgriculturalEnterprises.ToList();

return View(new EnterpriseFormViewModel { Enterprises = enterprises, NewEnterprise = new AgriculturalEnterprise() });

}

[HttpPost]

public IActionResult DeleteAll()

{

\_context.AgriculturalEnterprises.RemoveRange(\_context.AgriculturalEnterprises);

\_context.SaveChanges();

return RedirectToAction("Index");

}

[HttpPost]

public IActionResult Delete(int id)

{

var enterprise = \_context.AgriculturalEnterprises.Find(id);

if (enterprise != null)

{

\_context.AgriculturalEnterprises.Remove(enterprise);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

[HttpGet]

public IActionResult Edit(int id)

{

var enterprise = \_context.AgriculturalEnterprises.Find(id);

if (enterprise == null)

{

return NotFound();

}

return View(enterprise);

}

[HttpPost]

public IActionResult Edit(AgriculturalEnterprise model)

{

if (ModelState.IsValid)

{

\_context.AgriculturalEnterprises.Update(model);

\_context.SaveChanges();

return RedirectToAction("Index");

}

return View(model);

}

}

}

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, число, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунки 6.13-6.14 – Результат роботи класу SubjectAdding

Висновок

У процесі виконання лабораторної роботи з використанням моделей у MVC WEB-застосунку я ознайомився з основними принципами роботи з даними в контексті архітектурного патерну Model-View-Controller. Було реалізовано ряд завдань, що підкреслюють важливість моделей у організації структури даних, їх обробці та взаємодії з базою даних.

Основні компоненти, які я реалізував, включають:

* **WeakTypization** — клас, що демонструє використання слабо типізованих форм для збору даних від користувача. Це дозволило показати, як обробку даних можна виконувати без строгого контролю типів, що полегшує гнучкість на етапі введення інформації.
* **StrictTypization** — клас, який ілюструє переваги суворо типізованих форм, де кожне поле відповідає властивостям моделі. Це підвищує рівень валідації та безпеки даних.
* **DataAdding** і **DataDeleting** — класи, які реалізують функціональність додавання нових записів до бази даних та видалення всіх записів, показуючи, як через Entity Framework можна ефективно управляти базою даних MySQL.
* **SpecificDeleting** — клас, що реалізує механізм видалення конкретного запису за вказаним ідентифікатором, що підкреслює важливість роботи з унікальними ідентифікаторами.
* **DataEditing** — клас, що дозволяє редагувати існуючі записи, демонструючи можливість взаємодії з даними на всіх етапах їх життєвого циклу.
* **MDFDatabase** та **DatabaseInitializing** — класи для роботи зі структурою бази даних і початковим заповненням таблиць, показуючи, як важливо правильно організувати бази даних при розробці застосунків.
* **SubjectAdding** — клас, який реалізує функціональність додавання нових полів до бази даних відповідно до потреб предметної області, що ілюструє можливості гнучкої адаптації моделей до змін.

Ця лабораторна робота допомогла мені глибше зрозуміти використання моделей у MVC-програмах, навчитися ефективно працювати із даними, а також забезпечити побудову структурованих і зрозумілих веб-додатків. Отримані знання та навички стануть основою для подальшої розробки складних веб-застосунків, що потребують надійної й ефективної взаємодії з базами даних і обробки інформації.