ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА КІБЕРНЕТИКИ КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Л.Л. Омельчук
А.О. Свистунов

ІНСТРУМЕНТАЛЬНІ СЕРЕДОВИЩА ТА ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

Лабораторний практикум

Навчальний посібник

Київ 2023

УДК 519.85 (075.8)

ББК 32.973.2-018я73

О57

Рецензенти:

доктор фіз.-мат. наук, проф. *С.С. Шкільняк,*

кандидат фіз.-мат. наук, асист. *О.В.Шишацька*

Рекомендовано до друку вченою радою факультету комп’ютерних наук та кібернетики (протокол № 9 від 21 березня 2023 року)

Омельчук Л.Л.

О57 Інструментальні середовища та технології програмування. Лабораторний практикум: навчальний посібник / Л.Л. Омельчук, А.О. Свистунов – Київ: 2023. – 160 с.

УДК 519.85 (075.8)

ББК 32.973.2-018я73

О57

Викладено матеріали до лабораторного практикуму з обов’язкової навчальної дисципліни «Інструментальні середовища та технології програмування». В посібнику наведено умови та інструкції до виконання лабораторних проєктів, порядок розрахунку семестрової оцінки, а також приклади контрольних робіт.

Для студентів другого курсу факультету комп’ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, які навчаються за освітньо-професійною програмою «Інформатика» спеціальності 122 «Комп’ютерні науки».

ЗМІСТ

ВСТУП

Навчальна дисципліна “Інструментальні середовища та технології програмування” є складовою освітньо-професійної програми підготовки фахівців за першим (*бакалаврським*) рівнем вищої освіти *галузі знань* 12 „Інформаційні технології” зі *спеціальності 122* „Комп’ютерні науки”, *освітньо-професійної програми* –„Інформатика”.

Дана дисципліна є обов’язковою навчальною дисципліною за ***програмою “Інформатика”***.
Викладається у **4** семестрі **2** курсу в **обсязі – 150 год.**

(5 кредитів ECTS) зокрема: *лекції – 34 год., лабораторні – 38 год., консультації – 2 год., самостійна робота – 76 год.* У курсі передбачено **2** *частини* та 2 *контрольні роботи*. Завершується дисципліна – **іспитом.**

Мета дисципліни – засвоєння знань з інструментальних середовищ та технологій програмування. Оволодіння базовими навичками проектування програмних систем, набуття навичок використання інструментальних середовищ програмування, та використання технологій роботи з даними та технологій створення вебдодатків.

Попередні вимоги до опанування або вибору навчальної дисципліни:

1. Знати: основні поняття об’єктно-орієнтованого програмування, основні етапи життєвого циклу ПС, шаблони, антишаблони та принципи об’єктно-орієнтованого проєктування програмного забезпечення.

2. Вміти: застосовувати на практиці інструментальні програмні засоби проєктування та розробки програмного забезпечення.

3. Володіти елементарними навичками: програмування мовою C#.

Завдання (навчальні цілі):

- набуття знань, умінь та навичок (компетенцій) на рівні новітніх досягнень у програмуванні, відповідно до освітньої кваліфікації „Бакалавр з комп’ютерних наук”. Зокрема, розвивати:
- здатність застосовувати знання у практичних ситуаціях;
- здатність оцінювати та забезпечувати якість виконуваних робіт;
- здатність проектувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об’єктно-орієнтованого, функціонального, логічного, з відповідними

моделями, методами й алгоритмами обчислень, структурами даних і механізмами управління;

- здатність реалізувати багаторівневу обчислювальну модель на основі архітектури клієнт-сервер, включаючи бази даних, знань і сховища даних, , виконувати розподілену обробку великих наборів даних на кластерах стандартних серверів для забезпечення обчислювальних потреб користувачів, у тому числі на хмарних сервісах;
- здатність застосовувати методології, технології та інструментальні засоби для управління процесами життєвого циклу інформаційних і програмних систем, продуктів і сервісів інформаційних технологій відповідно до вимог замовника.

Місце дисципліни. Обов'язкова навчальна дисципліна „Інструментальні середовища та технології програмування" є складовою циклу професійної підготовки за освітньо-професійною програмою першого (бакалаврського) рівня вищої освіти "Інформатика", що реалізується факультетом комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка за спеціальністю 122 "Комп'ютерні науки".

Зв'язок з іншими дисциплінами. Для допуску до дисципліни „Інструментальні середовища та технології програмування" освітньо-професійної програми „Інформатика" студент повинен опанувати компетентності та результати навчання, які надає дисципліна „Об'єктно-орієнтоване програмування" освітньо-професійної програми „Інформатика". Дисципліна „Інструментальні середовища та технології програмування" пов'язана з дисципліною „Бази даних та інформаційні системи". Дисципліна „Інструментальні середовища та технології програмування" є базовою для засвоєння дисципліни „Системного програмування".

Результати навчання за дисципліною:

Код	Результат навчання	Форми (та/або методи і технології) викладання і навчання	Методи оцінювання та пороговий критерій оцінювання (за необхідності)	Відсоток у підсумковій оцінці з дисципліни
РН1.1	*Знати основи реляційних баз даних та мови запитів SQL, технології розробки інформаційних програмних систем*	*Лекція*	*Контрольна робота, іспит*	*20%*
РН1.2	*Знати принципи роботи технологій доступу до даних*	*Лекція*		
РН1.3	*Знати основи HTML, CSS, JavaScript*	*Лекція*	*Контрольна робота, іспит*	*20%*
РН1.4	*Знати базові елементи програмної інженерії*	*Лекція*		
РН1.5	*Знати принципи роботи технологій створення веб-застосунків на прикладі ASP.Net Core*	*Лекція*		
РН2.1	*Вміти працювати з технологією ADO.Net Core на автономному рівні*	*лабораторна робота, самостійна робота*	*Захист лабораторного проєкту, іспит*	*25%*
РН2.2	*Вміти працювати з технологією Entity Framework Core*	*лабораторна робота, самостійна робота*	*Захист лабораторного проєкту, іспит*	
РН2.3	*Вміти працювати з технологією ASP.Net Core*	*лабораторна робота, самостійна робота*	*Захист лабораторного проєкту, іспит*	*25%*
РН4.1	*Здатність до пошуку, оброблення та аналізу інформації з різних джерел*	*лабораторна робота, самостійна робота*	*Захист звіту*	*10%*

Співвідношення результатів навчання дисципліни із програмними результатами навчання

Результати навчання дисципліни **Програмні результати навчання**	**РН 1.1**	**РН 1.2**	**РН 1.3**	**РН 1.4**	**РН 1.5**	**РН 2.1**	**РН 2.2**	**РН 2.3**	**РН 4.1**
(з опису освітньої програми)									
ПРН9. Розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв'язання задач в галузі комп'ютерних наук.	+	+			+	+		+	
ПРН11. Володіти навичками управління життєвим циклом програмного забезпечення, продуктів і сервісів інформаційних технологій відповідно до вимог і обмежень замовника, вміти розробляти проектну документацію (техніко-економічне обґрунтування, технічне завдання, бізнес-план, угоду, договір, контракт).		+	+	+			+		+
ПРН15. Застосовувати знання методології та CASE-засобів проектування складних систем, методів структурного аналізу систем, об'єктно-орієнтованої методології проектування при розробці і дослідженні функціональних моделей організаційно-економічних і виробничо-технічних систем.	+	+	+	+		+	+	+	

СХЕМА ФОРМУВАННЯ ОЦІНКИ

На першому лабораторному занятті з „Інструментальні середовища та технології програмування” кожен студент визначається з темою лабораторного проєкту.

Перелік завдань для лабораторних занять не є вичерпним та визначається викладачем індивідуально.

Вибір студента фіксується на першому занятті.

Окрім усього передбачається, що до кінця семестру кожен студент:

- виконає та захистить два лабораторних проєкти;
- напише звіт до першого лабораторного проєкту;
- успішно напише дві контрольні роботи.

В кінці семестру передбачено іспит за всіма темами семестру.

Семестрове оцінювання:

1. Контрольна робота (тест): РН 1.1., РН 1.2 — 10 балів/6 балів.

2. Контрольна робота (тест): РН1.3, РН 1.4., РН 1.5 - 10 балів/6 балів.

3. Лабораторна робота 1 (етапи 1.0-1.7): РН 2.1, РН2.3 – 20 балів/12 балів.

4. Лабораторна робота 2 (етапи 2.0-2.4): РН 2.2 – 10 балів/6 балів.

5. Звіт (етап 3.0): РН4.1 – 10 балів/6 балів.

- підсумкове оцінювання (у формі іспиту)*:*

- максимальна кількість балів які можуть бути отримані студентом: 40 балів;

- результати навчання які будуть оцінюватись: РН1.1, РН1.2, РН1.3, РН1.4, РН1.5, РН2.1, РН2.2, РН2.3;

- форма проведення і види завдань: письмова робота.

За бажанням студента та опанування ним дисципліни „Об’єктно-орієнтоване програмування” з оцінкою не нижче 75 балів лабораторна робота 1 та звіт можуть бути замінені участю у командних проєктах в межах міжлисциплінарного проєкту зі студентами ОПП „Інформатика”.

Терміни проведення форм оцінювання:

1. Контрольна робота (тест): до 7 тижня семестру.

2. Контрольна робота (тест): до 15 тижня семестру.

3. Лабораторний проєкт 1: до 13 тижня семестру.
Етап 1.0: до 2 тижня семестру;
Етап 1.1: до 3 тижня семестру;
Етап 1.2: до 6 тижня семестру;
Етап 1.3: до 6 тижня семестру;
Етап 1.4: до 7 тижня семестру;
Етап 1.5: до 8 тижня семестру;
Етап 1.6: до 10 тижня семестру;
Етап 1.7: до 13 тижня семестру;
4. Лабораторний проєкт 2: до 10 тижня семестру.
Етап 2.0: до 13 тижня семестру;
Етап 2.1: до 15 тижня семестру;
Етап 2.2: до 16 тижня семестру;
Етап 2.3: до 18 тижня семестру;
Етап 2.4: до 18 тижня семестру;
5. Звіт у форматі курсової роботи (етап 3.0): до 15 тижня семестру.

Студент має право на одне перескладання кожної контрольної роботи із можливістю отримання максимально 8 балів за кожну. Термін перескладання визначається викладачем.

У разі неякісного виконання лабораторної роботи, викладач має право не зарахувати лабораторну роботу, або знизити за неї бали.

Студент має право здавати лабораторні роботи після закінчення визначеного для них терміну, але з втратою одного балу за кожен тиждень, який пройшов з моменту закінчення терміну її здачі.

Усі види робіт приймаються ВИКЛЮЧНО під час лабораторних занять.

Шкала відповідності оцінок

Відмінно / Excellent	90-100
Добре / Good	75-89
Задовільно / Satisfactory	60-74
Незадовільно / Fail	0-59

УМОВИ ЛАБОРАТОРНИХ ПРОЄКТІВ

Наведені нижче умови завдань не є вичерпними чи абсолютно точними. Автору розробки надається можливість розширювати поставлені завдання та виходити за їх рамки.

Загальна постановка задачі

Метою виконання лабораторних робіт є вивчення методів конструювання інформаційних систем. Результатом виконання лабораторних робіт 1 та 2 є інформаційна система з візуальним інтерейсом та прикладним програмним мережевим інтерфейсом (REST API), що містить необхідні декларативні засоби для контейнеризації та публікації у хмарних середовищах оркестрації, відкрита до розширення і придатна до горизонтального масштабування.

До лабораторної роботи необхідно оформити звіт. Звіт до лабораторної роботи повинен відповідати усім вимогам, які висуваються до курсових робіт. Звіт має включати постановку задачі, обґрунтування обраних методів розв'язання та інструкцію користувача. Під час тестування провадиться введення нових даних у таблиці бази даних, виконання запитів та ін.

Усі проєкти студенти повинні розміщувати в розподіленій системі контролю версій (наприклад, GitHub [13]). Напередодні здачі лабораторної роботи необхідно надіслати посилання на публічний репозиторій викладачу лабораторного практикуму.

Загальна умова ***може бути змінена (після попереднього узгодження з викладачем), але зі збереженням мінімальної складності моделі даних*** (кількості таблиць).

Вимоги до вибору варіанту (доменної області)

Модель даних (кількість таблиць) має складатися мінімум з 5 сутностей, з яких мінімум 2 мастер таблиці (корені агрегатів, *aggregate root*), кількість властивостей (атрибутів таблиць) не регламентується, але має бути мінімально достатньої для моделювання сутності в рамках доменної області. У рамках агрегатів мають бути використані допоміжні таблиці (класифікатори, довідники, словники та ін.). Об’єкти предметної області (як мінімум корені агрегатів) мають мати властивості, що відображають дату створення та останньої зміни об’єкта. Мінімум одна з сутностей має мати історію змін (окрему сутність з зображенням суті зміни та дати зміни). Інформаційна система має орієнтуватися на потреби кінцевого користувача (а не програміста).

Вимоги до інтерфейсу користувача

Кількість форм у візуальній версії застосунку та методів API мусить бути достатньою для повної реалізації бізнес процесу. В рамках візуального інтерфейсу користувача повинна бути подана коротка інформація про систему та її використання. Діалог україномовний. До одного з інтерфейсів включити можливість пошуку інформації за ключовими словами, атрибутами, тощо з метою динамічної генерації запитів.

Загальні вимоги до всіх лабораторних робіт

1. На основі сутностей предметної області створити таблиці бази даних. Рекомендована СУБД: Microsoft SQL Server.
2. Застосунок повинен підтримувати роботу з кирилицею (бути багатомовним) в тому числі при збереженні інформації в БД.
3. Код повинен бути документований.
4. Застосунок повинен коректно реагувати на помилки та виключення.
5. Принаймні в одному з застосунків (лабораторних робіт) повинна бути реалізована система Авторизації та Аутентифікації.

Вимоги до оформлення

1. Оформлення коду повинно відповідати C# Coding Conventions [14] і бути консистентним у всьому проєкті.
2. Система має бути наповнена коректними даними, мінімально необхідними для демонстрації проєкту (не використовувати як дані: aaa, bbb, ccc).
3. Проєкт повинен бути збережений у системі контролю версій (Git, Perforce) і завантажений у публічний репозиторій (GitHub, GitLab, Bitbucket, Azure DevOps). Основною метою даної умови є надати можливість викладачу переглядати та завантажувати файли вихідного коду з метою перевірки, а також ідентифікація автора.

ПРИКЛАДИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

Приклади індивідуальних завдань (номер індивідуального завдання відповідає номеру в списку групи). Ці завдання після узгодження з викладачем можуть бути змінені (БАЖАНО ОБРАТИ СВОЮ ТЕМАТИКУ):

1. "Наукові роботи кафедри"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення) (департамент, відділення), кафедра, лабораторія, посада (із зазначенням початку та кінця перебування на посаді), назва виконуваної наукової чи дослідної роботи, замовник, його адреса, підпорядкування, галузь та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу робіт, виконуваних кафедрою (окремо динаміка змін).

2. "Кафедральна бібліотека"

Таблиці містять таку інформацію: П.І.Б. автора, назва, анотація, кваліфікаційні ознаки видання, для читачів - П.І.Б., факультет, кафедра, посада та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу складу читачів бібліотеки та її фондів (окремо динаміка змін).

3. "Гуртожиток"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра , курс, дані про місце проживання (із датами)та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу розселення в гуртожитку (окремо динаміка змін).

4. "Електронний архів факультету"

Таблиці містять таку інформацію: П.І.Б. автора, назва матеріалу, факультет (департамент, відділення), кафедра, вид матеріалу, обсяг, дата створення та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок та пошуку інформації (окремо динаміка змін).

5. "Розклад"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра викладача, дані про аудиторії, навчальний план та склад студентів.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок та пошуку інформації (окремо динаміка змін).

6. "Успішність студентів"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, дані про навчальні дисципліни та успішність студентів та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок та пошуку інформації (окремо динаміка змін).

7."Інформаційні ресурси кафедри"

Таблиці БД містять таку інформацію: Назва ресурсу (ресурс - будь-яка інформація навчального та наукового спрямування), анотація, вид, автор (П.І.Б., факультет (департамент, відділення), кафедра….), умови використання, адреса та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок

8. "Розклад роботи дисплейних класів факультету та АРМ кафедр"

Таблиці БД містять таку інформацію: Інформація про класи та окремі робочі місця, інформація про користувачів, дані про розклад планових занять та викладачів, час вільного доступу

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок

9. "Програмне забезпечення на мережі факультету"

Таблиці БД містять таку інформацію: Назва програмного засобу, анотація, вид, версія, автор (….), умови використання, де записано дистрибутив

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок

10. "Інформаційні сервіси мережі факультету"

Таблиці БД містять таку інформацію: Назва сервісу, анотація, вид, версія, автор (….), умови та правила використання, інформація при реєстрації користувачів

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок

11. "Мережна газета факультету"

Таблиці БД містять таку інформацію: Назва статті чи графічного матеріалу, анотація, автор (….), де записано файли, інформація про читачів та їх відгуки

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок

12. "Індивідуальна робота кафедри зі студентами"

Таблиці БД містять таку інформацію: Інформація про студентів та викладачів (…), теми та графік роботи, виконання завдань, допоміжні інформаційні матеріали до тем, запитання-відповіді

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок

13. " Спорт на факультеті"

Таблиці БД містять таку інформацію: Інформація про студентів та викладачів (….), що займаються у спортивних секціях, графік роботи секцій, план проведення змагань

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему одержання довідок

14. "Кадри науковців (Аналіз штатного розпису)"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення) (департамент, відділення), кафедра, лабораторія, посада (із зазначенням початку та кінця перебування на посаді) та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу кадрів, що працюють у підрозділах.

15. "Кадри (Освіта)"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, освіта (базова, додаткова, аспірантура, докторантура), навчальний заклад, де здобувалась освіта з датами, та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу кадрів за освітою (окремо динаміка зміни).

16. "Кадри науковців (Звання)"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра , науковий ступінь, вчене звання (із датами).

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу кадрів за званням, ступенем (окремо динаміка зміни).

17. "Кадри науковців (Пенсія)"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, дата народження, стать та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу кадрів пенсійного та передпенсійного віку з урахуванням статі, вченого звання, наукового ступеня.

18. "Кадри науковців (Публікації)"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, дані про публікації.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу виданих книг, наукових статей та ін. Окремо подати динаміку змін. Публікацій з урахуванням: видавництва, видання, назви журналу (збірника), обсягу в стор. або др. арк., дати.

19. "Кадри науковців (Зарплата)"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, посада, посадовий оклад, час перебування на посаді.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу планового фонду зарплати (окремо динаміка зміни).

20. "Конференції"

Таблиці містять інформацію про наукові конференції: назва, терміни проведення, тематика (для якого підрозділу цікаво), організатор, місце проведення, термін подачі рукописів, вартість участі та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу участі в конференціях: що планується на протязі місяця (кварталу, року, де брали участь....). Окремо подати динаміку змін.

21. "Аспірантура"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, фах, форма навчання, початок навчання, графік іспитів та заліків, форми та терміни звітності на кафедрі та ін. Використати допоміжні таблиці (класифікатори, довідники, словники та ін.)

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу виконання плану аспірантом, даних про кількість, фах аспірантів та ін..... Окремо подати динаміку змін.

22. "Студентські гуртки та семінарами"

Таблиці містять таку інформацію: Назва гуртка/ семінару, факультет (департамент, відділення), кафедра, день та час проведення засідань, староста, орієнтація (на який курс та яку тематику), П.І.Б. керівника.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу діяльності гуртків/ семінарів (окремо динаміка зміни кількості учасників, кількості гуртків при кафедрі).

23. "Кадри науковців"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, посада, посадовий оклад. прийняття та звільнення, переміщення з посади на посаду, зміни посадового окладу та ін.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу поточного стану кадрів (окремо динаміка зміни загальної кількості та за окремими показниками).

24. "Облік випускників"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, спеціальність, час вступу та закінчення, переміщення з посади на посаду та з одного місця роботи на інше.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу складу випускників кафедри (підрозділу) за місцем роботи, посадою, оплатою праці та ін. Окремо подати динаміку зміни кількісних показників та аналіз за окремими показниками.

25. "Міжнародні контакти"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, спеціальність, часові рамки виду співробітництва (перебування у науковому відрядженні, читання лекцій, передача наукової інформації за кордон, одержання інформації із-за кордону, проведення семінару.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу ефективності міжнародних контактів кафедри (підрозділу) за місцем та видом роботи, кількістю та якісним складом наукових груп,

навчальних семінарів та ін. Окремо подати динаміку зміни кількісних показників та аналіз за окремими показниками.

26. "День факультету"

Таблиці містять таку інформацію: назва заходу на день факультету, П.І.Б., факультет (департамент, відділення), кафедра, посада відповідального за проведення, виконавців окремих доручень щодо підготовки та проведення дня факультету, терміни підготовки, час, місце проведення, неформальна характеристика та ін. на розсуд виконавця Лабораторної роботи.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу поточного стану кадрів (окремо динаміка зміни загальної кількості та за окремими показниками).

27. "Наукове товариство студентів та аспірантів"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, спеціальність, час вступу до товариства його членів, план проведення заходів (що, де, коли, неформальна характеристика) на інше.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу складу товариства факультету, кафедри (підрозділу) за окремими показниками та ін. Окремо подати динаміку зміни кількісних показників та аналіз за окремими показниками.

28. "Студентський парламент"

Таблиці містять таку інформацію: П.І.Б., факультет (департамент, відділення), кафедра, спеціальність, часові рамки виду заходу, що проводить студентський парламент.

Розробити інформаційну систему, яка б включала підсистему введення нової інформації та підсистему аналізу ефективності роботи студентського парламенту факультету, кафедри (підрозділу) Окремо подати динаміку зміни кількісних показників та аналіз за окремими показниками.

ВИМОГИ ДО ЗВІТУ

Звіт до лабораторної роботи повинен відповідати усім вимогам, які висуваються до курсових робіт.

Звіт має включати постановку задачі, обґрунтування обраних методів розв'язання та інструкцію користувача. Структура звіту має відображати плани (в тому числі потижневі) та наміри автора розробки у першій частині звіту та отримані результати – у другій його частині.

Текст звіту може бути поданий викладачу практикуму у письмовій формі чи як електронний документ за дозволом викладача. Розвинена допомога не заміняє звіт. Датою прийняття лабораторної роботи лектором є дата, що буде зафіксована скринькою електронної пошти лектора за електронною адресою, що повідомляється на лекції, або системою електронного навчання. Формат імені архіву, що додаються до електронного листа moroz_25_1.zip (rar) означає, що автором є студент Мороз із групи К25 і в ньому всі матеріали для перевірки лабораторної роботи № 1.

Перевірка виконаних робіт здійснюється за розкладом занять викладача. Разом з проєктом має бути набір тестових файлів та інших матеріалів, що свідчать про правильність створених програм. Автор проєкту повинен щотижня інформувати викладача про виконані етапи робіт. Звіт є обов'язковою складовою частиною роботи. Для одержання найвищої оцінки обов'язковим є використання C# (чи іншого середовища розробки), ООП та виконання індивідуального завдання, що узгоджується студентом та викладачем (наприклад, написати функцію для спеціалізованого редагування HTML-сторінки й інше).

Зразок структури звіту

Вступ (в якому зазначити **актуальність теми** (висвітити актуальність розробки проєкту Вашої тематики), **методи дослідження**, **цілі і завдання** (не забути крім прикладних цілей вагомих для обраної предметної області зазначити й професійні цілі «закріпити знання, ознайомитися, поглибити знання з технологій НАВЕСТИ ПЕРЕЛІК ТЕХНОЛОГІЙ»)).

Кількість розділів визначається відповідно до поставлених задач. Не допускається вміст розділу менше 3 сторінок. За необхідності доцільно об’єднувати декілька розділів в один.

Розділ 1. Огляд наявних на ринку систем (короткий огляд конкуруючих систем з їх перевагами та недоліками. Висновки, що запозичили, в чому переваги саме Вашого проєкту).

Розділ 2. Огляд використаних технологій (бажано крім огляду зазначити чому обрали саме обрали ці технології).

Розділ 3. Призначення і цілі створення системи.

3.1. Призначення системи

Призначенням системи «НАЗВА ВАШОЇ ПРОГРАМИ» є автоматизація процесу НАПИСАТИ ПРОЦЕСИ ОБРАНОЇ ПРЕДМЕТНОЇ ОБЛАСТІ, що дає ПЕРЕРАХУВАТИ КОНКРЕТНІ ПЕРЕВАГИ ВИКОРИСТАННЯ ВАШОЇ ПРОГРАМИ.

Лабораторна робота передбачає:

- аналіз методів, методик і моделей, що застосовуються для розв'язання задач ПЕРЕРАХУВАТИ ЗАДАЧІ, ЩО ПІДЛЯГАЮТЬ АВТОМАТИЗАЦІЇ.
- аналіз наявних програмних засобів в галузі ЗАЗНАЧИТИ КОНКРЕТНУ ГАЛУЗЬ.
- проєктування та програмну реалізацію системи «НАЗВА ВАШОЇ ПРОГРАМИ».
- апробацію «НАЗВА ВАШОЇ ПРОГРАМИ».

3.2. Цілі створення системи(бажано з Use Case діаграмою)

«НАЗВА ВАШОЇ ПРОГРАМИ» створюється з метою:

- забезпечення збору, обробки та аналізу інформації про НАВЕСТИ КОНКРЕТНИЙ ПЕРЕЛІК;
- можливість проаналізувати НАВЕСТИ КОНКРЕТНИЙ ПЕРЕЛІК;

Також система призначена для НАВЕСТИ КОНКРЕТНИЙ ПЕРЕЛІК.

4. Вимоги до системи

4.1. Вимоги до системи в цілому

4.1.1. Вимоги до структури та функціонування системи

Система «НАЗВА ВАШОЇ ПРОГРАМИ» повинна …

В Системі передбачається виділити наступні функціональні підсистеми:

- **адміністративна**, призначена …
- : **підсистема користувача** …

4.2. Вимоги до функцій, які виконуються системою

4.2.1. Підсистема адміністратора.

Таблиця 1 – Перелік функцій, задач що підлягають автоматизації

Функція	Задача
Керувати роботою користувачів системи	Редагування та видалення інформації про користувачів системи
	Формування та візуалізація звітів про користувачів системи
Робота з …	

4.2.2. Підсистема користувача

Таблиця 2 – Перелік функцій, задач що підлягають автоматизації

Функція	**Задача**
Реєстрація	Введення інформації про себе
	Оновлення реєстрації
	Видалення інформації про себе
Робота з …	

4.2. Технічні вимоги

Браузери: Сайт повинен коректно відображатися в сучасних інтернет-браузерах: останні версії Google Chrome, Mozilla Firefox, Miscrosoft Edge, тощо. Підтримка застарілих версій – за бажанням.

На довільні некоректні дії користувача, пов'язані з введенням невірних даних, не заповненням обов'язкових полів введення в формах та інші, які можуть бути оброблені системою, генеруються відповідні повідомлення про помилки українською мовою, в межах загального дизайну сайта.

Операційна система: ЗАЗНАЧИТИ КОНКРЕТНІ НАЗВИ.

Джерелом даних для Системи повинна бути інформаційна система (СУБД ЗАЗНАЧИТИ НАЗВУ).

4.3. Логічна структура бази даних

Зазначити яка СУБД використовується

Рисунок НОМЕР РИСУНКУ - Діаграма бази даних «НАЗВА БАЗИ ДАНИХ»

Наводите перелік таблиць (повний, чи розбитий за логічними блоками):

Таблиця – ... Опис…

Номер	**Таблиця**	**Опис**
1	aspnet_Applications	Таблиця для збереження інформації про вебзастосунки, які використовують цю БД
2	aspnet_Paths	...
3	...	...

4.4. Опис таблиць

За кожною таблицею навести опис даних:

Таблиця –.... Опис даних

Таблиця НАЗВА Атрибут	**Тип**	**Опис**
ID	integer	Ідентифікатор
...	...	...

4.5. Реалізація системи

Опис загальної структури, можливі невеликі принципові фрагменти коду.

5. Інструкція користувача

(з ілюстраціями)

Висновки

Згадати всі перераховані у вступі та розділі 3 задачі та оцінити рівень їх виконання.

Розробив систему….

Вивчив, дослідив, поглибив знання з…..

ПЕРЕДМОВА ДО ЛАБОРАТОРНОГО ПРАКТИКУМУ

Організація процесу виконання лабораторного практикуму

Лабораторний практикум містить два лабораторних проєкти (роботи), у яких студентам запропоновано створити два вебзастосунки з використанням різних підходів реалізації інтерфейсу – графічний вебінтерфейс та прикладний програмний інтерфейс. Лабораторні проєкти розбиті на етапи, що відтворюють ітеративний процес розробки програмного забезпечення. Порядок виконання етапів (окрім проєктування) довільний, проте, рекомендовано притримуватися наведеної послідовності.

Архітектура системи

Результатом виконання лабораторних проєктів1 та 2 є інформаційна система, що складається з бази даних та двох вебзастосунків, що можуть бути розгорнуті в рамках кластеру (або інфраструктури іншої топології). Лабораторний проєкт №1 присвячений створенню вебсайту, що надає графічний інтерфейс доступу до системи користувачам. Лабораторний проєкт№2 присвячена створенню мережевого прикладного програмного інтерфейсу (WebAPI) для інших застосунків та контейнеризації всього рішення.

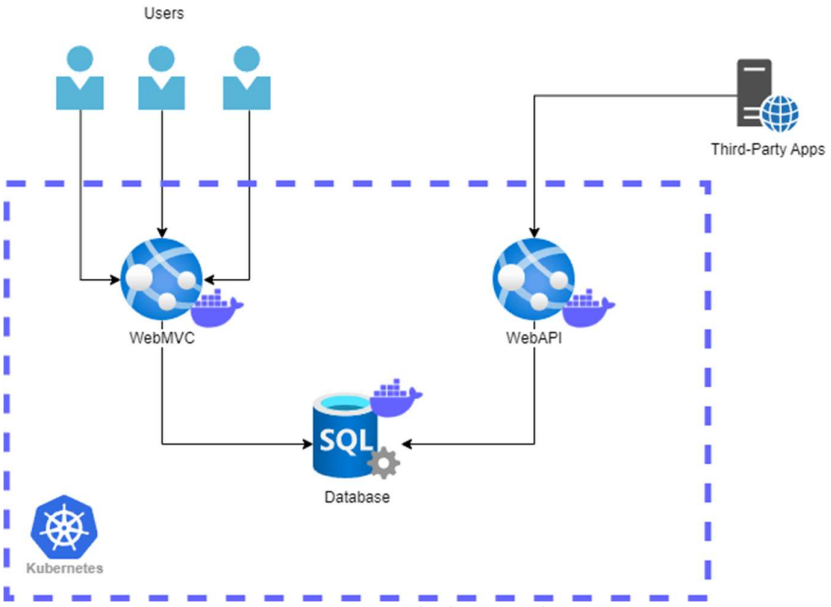


Рисунок 1 – Схема розгортання інформаційної системи

Вибір технології

В рамках лекційної частини курсу для створення інформаційної системи використовуються мова програмування C#, засоби платформи .NET, а саме фреймворки ASP.NET та Entity Framework, СУБД Microsoft SQL Server, система контролю версій Git, середовище для контейнеризації Docker (всі останніх LTS версій). За попереднім узгодженням з викладачем, студенти мають право обрати інші схожі технології.

Вибір предметної області

Будь-який програмний застосунок має полегшувати життєдіяльність користувачів. Основою будь-якого програмного продукту є *бізнес-процес*, або спрощено, послідовність дій для досягнення певної мети. В якості прикладу, візьмемо систему продажу квитків. Які бізнес-процеси закладені у системі продажу квитків? Очевидно, що можна виділити, як мінімум два:

- надати користувачеві можливість обрати кінотеатр, сеанс, місце, виконати оплату та отримати квиток;
- надати адміністратору кінотеатру можливість додавати, оновлювати та видаляти кінотеатри, сеанси, місця, та перевіряти валідність квитка.

Дана вправа здається простою, проте дозволяє визначити:

- основні сутності (майбутні таблиці) в рамках предметної області (користувач, квиток, сеанс, місце, кінотеатр);
- основних акторів системи (звичайний користувач та адміністратор);
- основні прецеденти для акторів системи (перегляд доступних сеансів, придбання квитка, здійснення оплати, тощо).

Вдало обрана тема та якісно проведений етап проєктування інформаційної системи дозволяє сфокусуватися на реалізації застосунку, уникаючи необхідності повертатися до проєктування і суттєво змінювати код проєкту.

Структура проєкту для лабораторних робіт

Перед початком виконання проєкту, рекомендовано встановити останні версії необхідного програмного забезпечення, а саме, інструментального середовища, засобів розробки обраної мови програмування та системи контролю версій.

Рекомендовано одразу створити теку для вашого проєкту в рамках лабораторного практику та ініціалізувати репозиторій у системі контролю версій (для Git – команда git init). Далі створіть базові файли та папки:

Таблиця 1.0 – Опис рекомендованих файлів папки проєкту

Назва	Тип	Опис
src	папка	Файли вихідного коду програми
img	папка	Файли зображень для документації (UML-діаграм)
.gitignore	файл	(лише для Git) Файл для приховування файлів, які будуть проігноровані у системі контролю версій
README.md	файл	Файл для документації у форматі markdown

Розгортання середовища для розробки програмного забезпечення з використанням Microsoft SQL Server

Розгортання СУБД

Спосіб 1. Розгортання локального інстансу SQL Server

Цей спосіб підходить для операційних систем Windows та дистрибутивів Linux (наприклад, Ubuntu). Інстанс SQL Server встановлюється в якості служби.

Інструкція під Windows: https://www.microsoft.com/en-us/sql-server/developer-get-started/csharp/win

Інструкція під Ubuntu/Linux: https://www.microsoft.com/en-us/sql-server/developer-get started/csharp/ubuntu

Спосіб 2. Використання SQL Server Express LocalDB з Visual Studio

Підходить тільки для Windows. ***Найпростіший спосіб!***

Крок 1. Відкрийте Visual Studio Installer (встановлюється разом з Visual Studio).

Крок 2. Оберіть вашу версію Visual Studio та оберіть пункт Modify.

Крок 3. У вкладці Individual components знайдіть SQL Server Express LocalDB. Оберіть та натисніть «Install».

Спосіб 3. Використання Docker-образу

Підходить абсолютно для всіх операційних систем. Потребує встановлення Docker, докладна інструкція як встановити Docker для Windows, Linux та MacOS: https://docs.docker.com/desktop/.

Після встановлення знайдіть образ SQL Server'у на Docker Hub https://hub.docker.com/_/microsoft mssql-server. Запустіть за прикладом.

Інструкція розгортання під MacOS: https://www.microsoft.com/en-us/sql-server/developer-get-started/csharp/macos (підійде і для Linux/Windows).

Увага! Не забувайте вказувати томи (volumes) для вашого контейнера, якщо хочете зберегти дані після перезавантаження/видалення контейнера.

Розгортання інтегрованого середовища для керування СУБД

Спосіб 1: Server Explorer у Visual Studio

Підходить для Windows, встановлюється за замовчуванням. https://learn.microsoft.com/en us/visualstudio/data-tools/add-new-connections?view=vs-2022

Спосіб 2: SQL Server Management Studio

Підходить для Windows, завантажити можна з офіційного сайту https://aka.ms/ssmsfullsetup.

Спосіб 3: Azure Data Studio

Для всіх платформ: https://learn.microsoft.com/en-us/sql/azure-data-studio/download-azure-data studio?view=sql-server-ver16&tabs=redhat-install%2Credhat-uninstall

ЛАБОРАТОРНИЙ ПРОЄКТ № 1
РОЗРОБКА ВЕБДОДАТКУ

Набір технологій (за замовчуванням):

- *Entity Framework Core (EF Core) Code First – ORM Framework*
- *ASP.NET Core MVC 6.x*

За попередньою домовленістю з викладачем (не пізніше 2-го тижня від початку семестру) набір технологій може бути змінений.

Наприклад:

для Java (Hibernate - ORM Java Framework, Spring MVC).

В таблиці наведено етапи, їх задачі та фінальний термін здачі конкретного етапу.

Таблиця 1.1 – Етапи лабораторного проєкту 1

Номер етапу	Фінальний термін здачі етапу (13 тиждень семестру)	Задача етапу та матеріали до його виконання	Форма здачі етапу	Максимальна кількість балів за етап
1.0	2-й тиждень від початку семестру	Початковим етапом до виконання лабораторних робіт є формулювання персонального завдання, розробка та затвердження викладачем діаграми прецедентів (Use-case діаграма) UML, яка демонструє особливості обраної предметної області, діаграма класів та проєкт WebMVC.Domain з класами-сутностями.	Розмістити діаграми (фото, *.png, *.jpg чи у іншому форматі) та вихідний код у системі контролю версій та надіслати посилання викладачу за 24 години до співбесіди за цим етапом. Перший етап ОБОВ'ЯЗКОВО повинен бути узгоджений з викладачем!	**2**

1.1	4-й тиждень від початку семестру	Створити клас DbContext у проєкті WebMVC.Infrastructure. Задати схему використовуючи Fluent API. Підключити базу даних у проєкті WebMVC та створити першу міграцію.	Розмістити оновлений вихідний код та PrntScr діаграми БД код у системі контролю версій та надіслати посилання на пошту викладачу принаймні за 24 години до співбесіди за цим етапом на лабораторному занятті.	**3**
1.2	6-й тиждень від початку семестру	Налаштування, створення контролелів та представлень (по одному контролеру мастер таблиці на етап)	Посилання на GitHub принаймні за 24 години до співбесіди за цим етапом на лабораторному занятті. При прийнятті враховується виконання етапів, їх якість, терміни, складність обраної предметної області. Особиста співбесіда є ОБОВ'ЯЗКОВОЮ! Без співбесіди бали не нараховуються.	**3**
1.3				**3**
1.4	7-й тиждень від початку семестру	Зміна майстер-сторінки та стилізація		**1**
1.5	8-й тиждень від початку семестру	Відображення даних на діаграмі	Без цих етапів лабораторна робота може бути зарахована максимум на 10 балів (без нарахування балів	**2**
1.6	10-й тиждень від	Робота з файлами (формування та збереження звітів)		**3** **+ 1**

	початку семестру	У форматі MS Excel (експорт та імпорт файлів) **За самостійну реалізацію роботи зі звітами у форматі .docx можливе отримання 1 додаткового балу.**	за ці етапи 2+2+3). Можна змінювати порядок здачі етапів 1.5-1.7. Ви також маєте право не усі з цих етапів. Ви не можете отримати додаткові бали за етап без здачі самого етапу.	**(додатковий бал)**
1.7	13-й тиждень від початку семестру	Робота з автентифікацією, авторизацією та ролями. **Можливість отримати до 2 додаткові бали** за реалізацію системи керування користувачами, зміни та валідації пароля, керування ролями, підтвердження по Email (https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio).	Без етапів 1.5-1.7 Ви маєте право оформити та здати звіт до лабораторної роботи. **Форма здачі:** Посилання на GitHub принаймні за 24 години до співбесіди за цим етапом на лабораторному занятті. **Крім того:** для здачі етапу 1.5 викладачу надсилаються зображення діаграм (PrntScr); для здачі етапу 1.6 викладачу надсилаються згенеровані та проаналізовані файли (у випадку якщо ці файли мають однаковий формат, то можна один файл).	**3** **+ 2 (додаткових бали)**

			При прийнятті враховується виконання етапів, їх якість, терміни, складність обраної предметної області. Особиста співбесіда є ОБОВ'ЯЗКОВОЮ! Без співбесіди бали не нараховуються.	

КРОКИ ВИКОНАННЯ ЕТАПУ 1.0

Створіть папку для вашого проєкту. Відкрийте командний рядок та перейдіть у цю папку використовуючи команду `cd <шлях-до-файлу>`:

```
C:\Users\anton>cd C:\Projects\istplabdemo
```

Створіть папки `src, img` та `docs` у папці вашого проєкту:

```
C:\Projects\istplabdemo>mkdir src img docs
```

Створіть файли `.gitignore` та `README.md` у папці вашого проєкту та:

```
C:\Projects\istplabdemo>echo >> .gitignore

C:\Projects\istplabdemo>echo >> README.md
```

Створіть репозиторій у папці:

```
C:\Projects\istplabdemo>git init
Initialized empty Git repository in C:/Projects/istplabdemo/.git/
```

Заповніть файл `.gitignore` відповідно до шаблону для вашої мови, наприклад, https://github.com/github/gitignore/blob/main/VisualStudio.gitignore.

Додайте зміни у репозиторій і створіть перший коміт:

```
C:\Projects\istplabdemo>git add .
C:\Projects\istplabdemo>git commit -m "Initial commit"
[master (root-commit) ee0f555] Initial commit
 2 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
```

Тепер, відкрийте Visual Studio та створіть нове порожнє рішення:

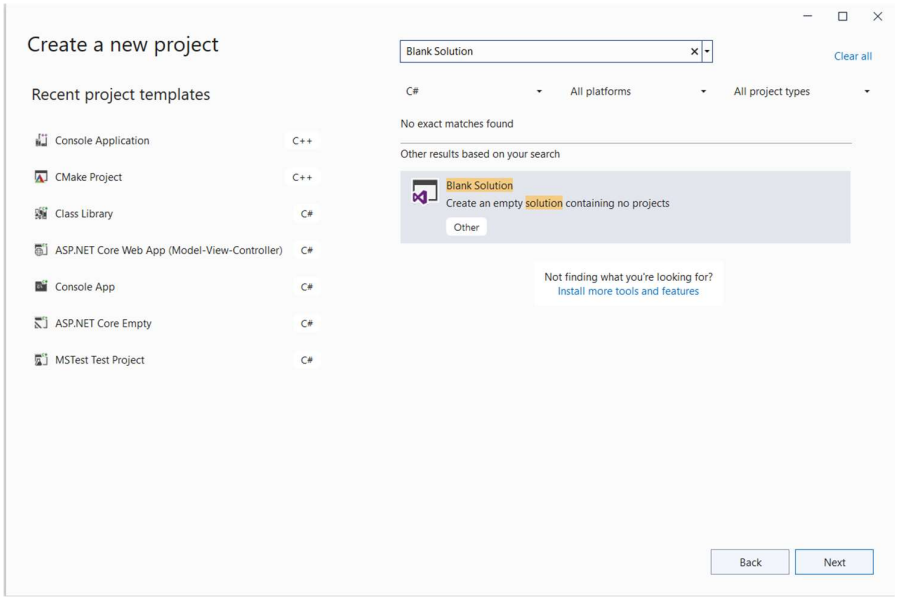


Рисунок 1.0.1. – Створення порожнього рішення

Оберіть шлях до папки `src` та створіть рішення. Створіть нову папку рішення використовуючи Solution Explorer. Назвіть папку MVC.

Папка рішення існує тільки всередині sln файлу рішення і не відображає реальний стан файлової системи. Проте, це зручний механізм для групування проєктів для відображення у Solution Explorer.

Після створення папки рішення, створіть проєкт `CinemaMVC.Domain` (замініть на назву вашого проєкту) типу Class Library. Саме тут знаходитимуться класи сутностей вашої предметної області. Створіть папку Entities та створіть абстрактний клас `Entity`:

```
namespace CinemaMVC.Domain.Entities;

public abstract class Entity
{
    public int Id { get; set; }
}
```

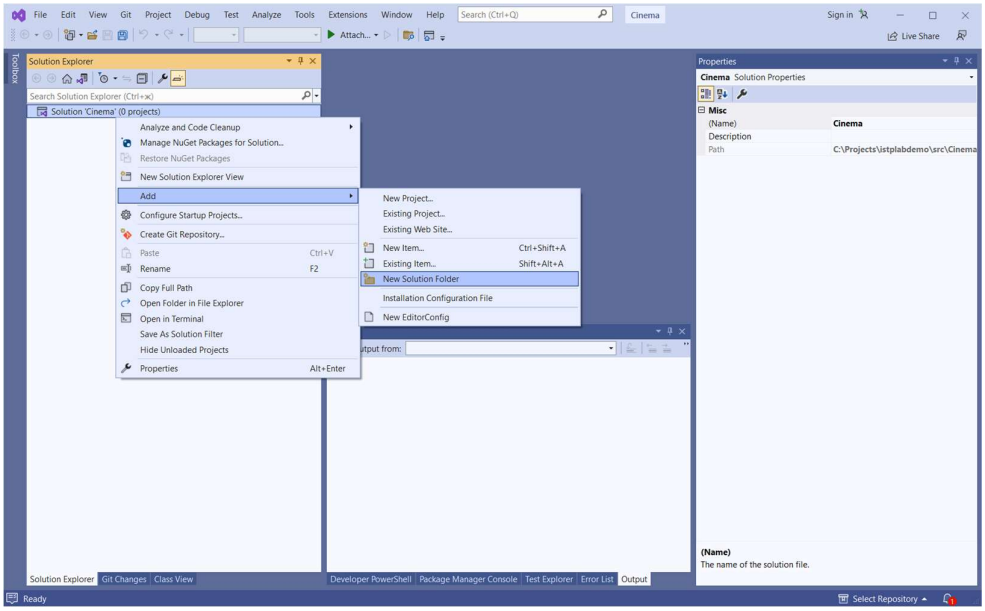


Рисунок 1.0.2 – Створення папки рішення

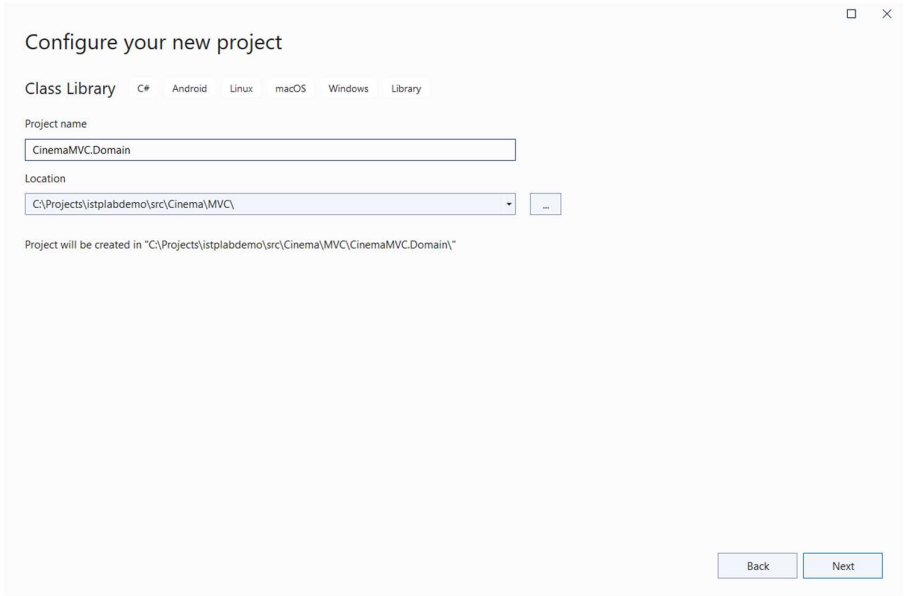


Рисунок 1.0.3 – Створення бібліотеки класів

Створіть порожній інтерфейс IAggregateRoot для позначення кореней агрегатів:

```
namespace CinemaMVC.Domain.Entities;

public interface IAggregateRoot
{
}
```

Далі створіть класи вашої предметної області. Кожна сутність має наслідувати абстрактний клас `Entity`. Майстер-стуності мають успадковуватися від `AggregateRoot`.

Підхід до реалізації класів може не відповідати всім принципам Domain Driven Design (DDD), проте, рекомендовано звернутися за натхненням до https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/net-core-microservice-domain-model.

У файлі `README.md` вкажіть власну уточнену постановку задачі лабораторної роботи: (вписати назву, додайте посилання на UML- діаграми у папці img). Приклади оформлення етапу взяті з робіт студентів попередніх років.

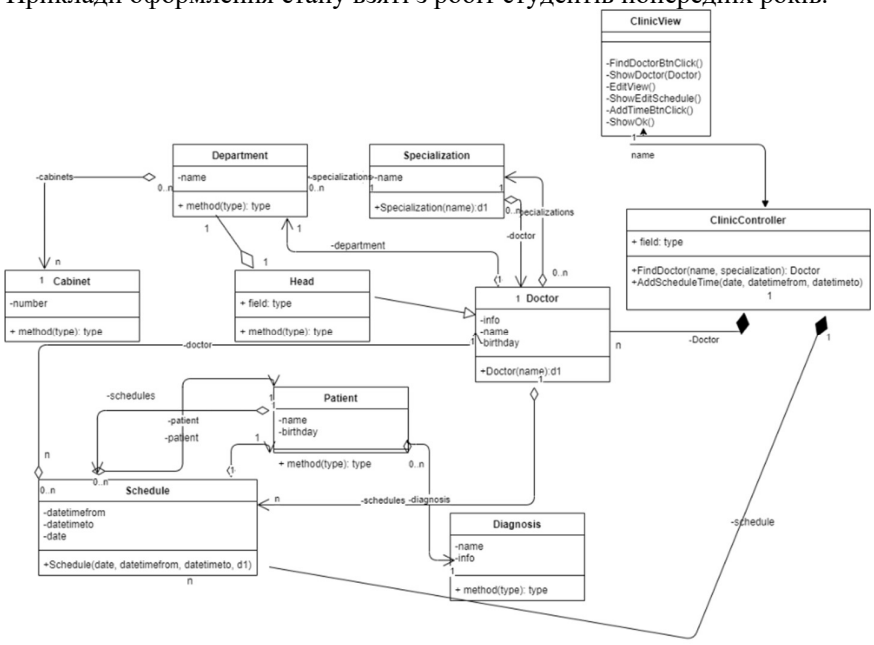


Рисунок 1.0.4 - Діаграма бази даних «онлайн поліклініка»

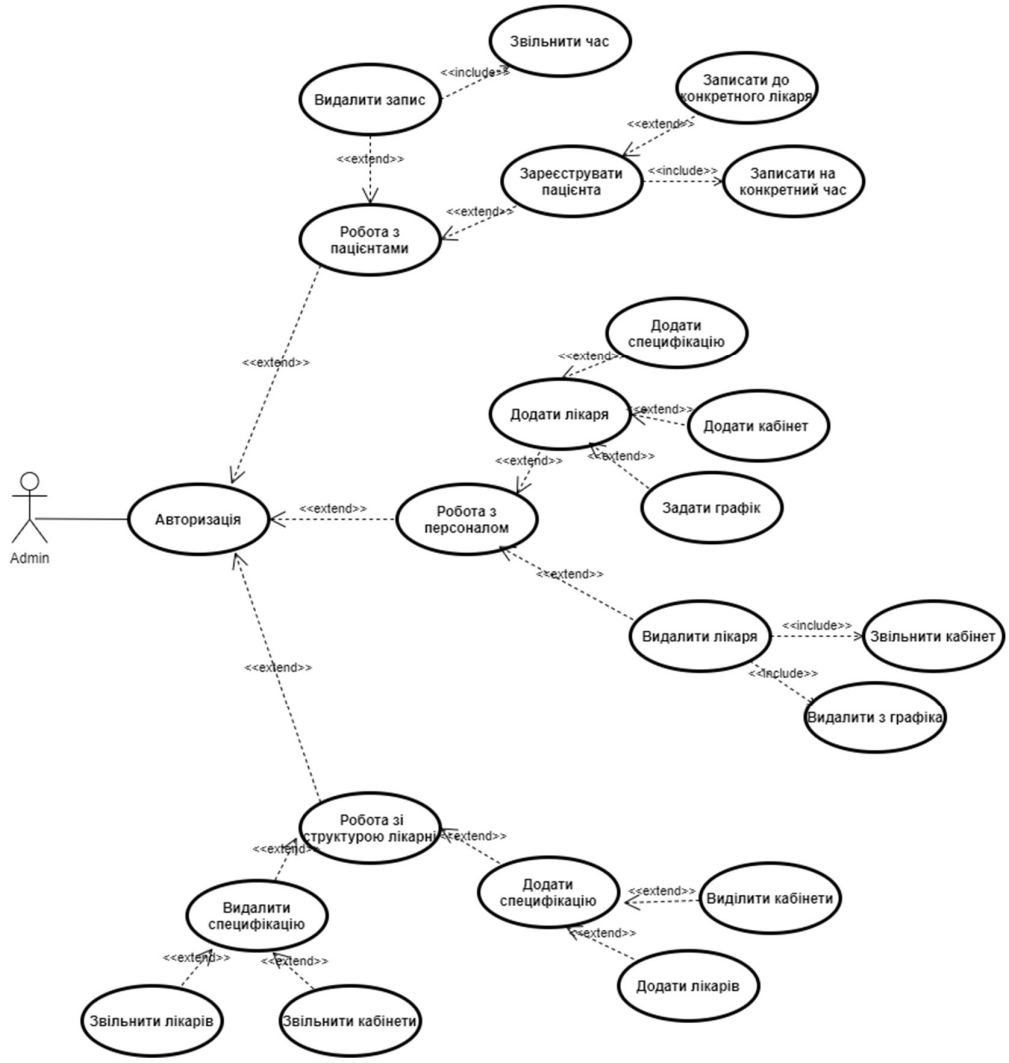


Рисунок 1.0.5 - UML- діаграма «Онлайн поліклініка»

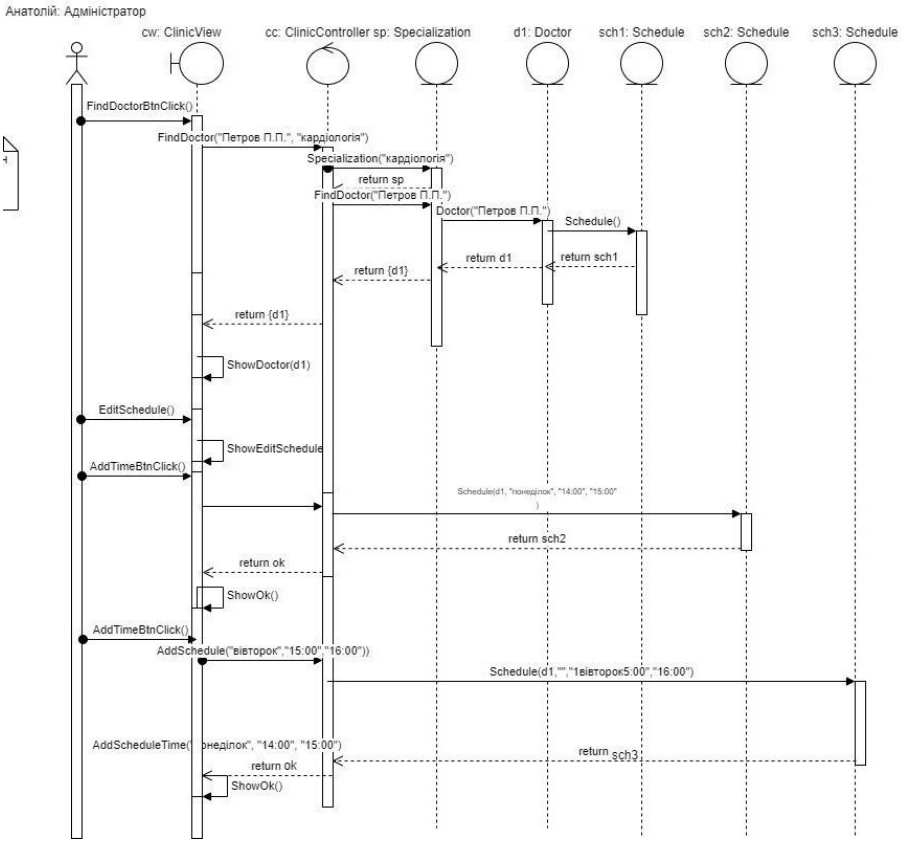


Рисунок 1.0.6 - UML- діаграма «Онлайн поліклініка»

Після створення необхідних класів та діаграм, створіть новий коміт у системі контролю версій на опублікуйте у ваш репозиторій у GitHub (чи у іншій системі) використовуючи команду `git push`. Ознайомлення з публікацією на GitHub рекомендовано почати з https://docs.github.com/en/get-started/quickstart/hello-world.

КРОКИ ВИКОНАННЯ ЕТАПУ 1.1

Створіть проєкт `CinemaMVC.Infrastracture` (замініть на назву вашого проєкту) типу Class Library. Саме тут знаходитимуться класи, що відповідають за шар інфраструктури вашого проєкту – контексти баз даних, реалізації репозиторіїв, тощо.

Відкрийте вікно пакетного менеджера NuGet для новоствореного проєкту, знайдіть та завантажте пакет `Microsoft.EntityFrameworkCore` (рис. 1.1.1 – 1.1.2). Дуже важливо обрати версію сумісну з версією платформи .NET вашого проєкту. Якщо версія вашого проєкту `.net6.0` (можна переглянути у `.csproj`-файлі), обирайте версію 6.X.X. Також додайте `Microsoft.EntityFrameworkCore.XXX` сумісну з типом вашого сховища, наприклад, для Microsoft SQL Server необхідно обрати пакет `Microsoft.EntityFrameworkCore.SqlServer` для отримання специфічних методів Fluent API під конкретне сховище даних.

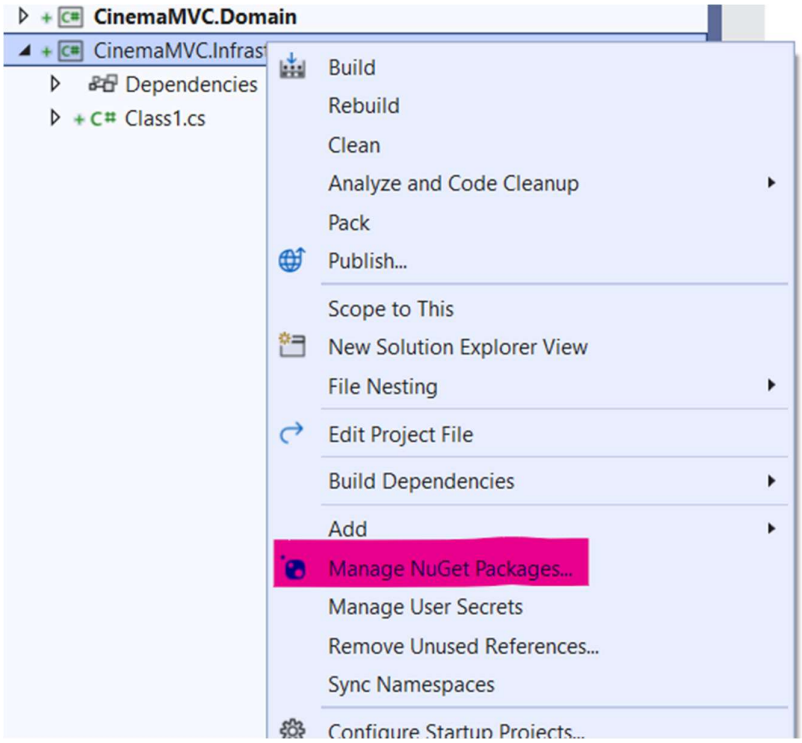


Рисунок 1.1.1 – Відкриття вікна пакетного менеджеру NuGet для проєкту

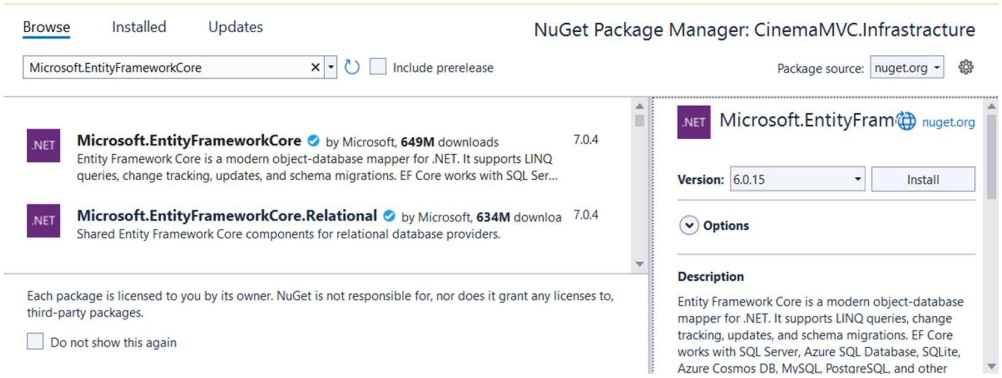


Рисунок 1.1.2 – Пошук та вибір версії пакету

Створіть клас контексту, що наслідує клас `DbContext`:

```
namespace CinemaMVC.Infrastructure;

public class CinemaContext : DbContext
{
    public CinemaContext(DbContextOptions<CinemaContext>
options)
        : base(options)
    {

    }

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        modelBuilder.ApplyConfiguration(new
MovieEntityTypeConfiguration());
        modelBuilder.ApplyConfiguration(new
ActorEntityTypeConfiguration());
        modelBuilder.ApplyConfiguration(new
DirectorEntityTypeConfiguration());
        modelBuilder.ApplyConfiguration(new
ArtistEntityTypeConfiguration());
    }
}
```

У лістингу вище пропонується використовувати підхід з винесенням конфігурацій сутностей у класи конфігурацій (`IEntityTypeConfiguration<>`).

Такий підхід дозволяє уникнути нагромадженню класу контексту. Альтернативою є вписувати весь код конфігурації моделі даних у методі `OnModelCreating`.

В якості прикладу використовується наступна ієрархія класів (рис. 1.1.3).

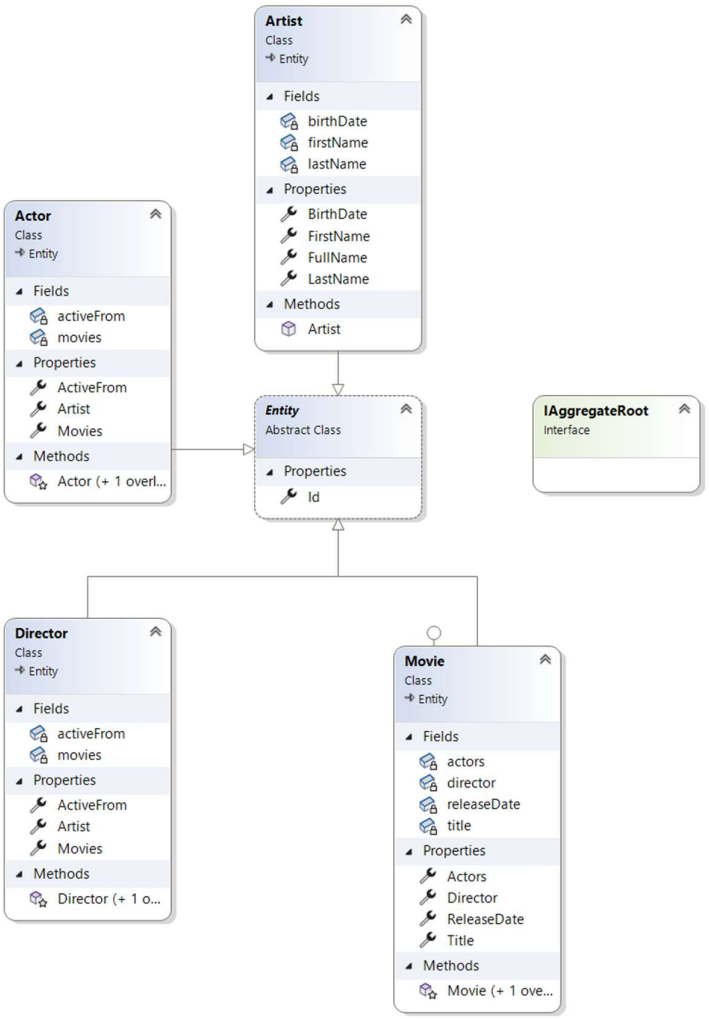


Рисунок 1.1.3 – Діаграма класів проєкту `CinemaMVC.Domain`

Для означення правил генерації моделі даних у сховищі рекомендовано використати підхід Fluent API.

```
namespace CinemaMVC.Infrastracture.EntityConfigurations;

internal class ActorEntityTypeConfiguration
    : IEntityTypeConfiguration<Actor>
{
    public void Configure(EntityTypeBuilder<Actor> builder)
    {
        builder.HasKey(actor => actor.Id);

        builder.HasOne(actor => actor.Artist);

        builder.Property(actor => actor.ActiveFrom)
            .HasField("activeFrom")
            .IsRequired(true);

        builder.Metadata
            .FindNavigation(nameof(Actor.Movies))?
            .SetPropertyAccessMode(PropertyAccessMode.Field);
    }
}
```

Для створення міграції, засобам командного рядка EntityFramework потрібно збудувати класи сутності, клас контексту та виконати порівняння існуючих міграцій з певним еталоном – базою даних, що відповідає снепшоту, що зберігається разом з міграціями. Для цього, необхідно створити точку входу для скриптів EF для встановлення з'єднання з вашим цільовим сховищем (а точніше екземпляром/копією). В якості такої точки входу може виступати ваш проєкт ASP.NET, або спеціальний клас що реалізує інтерфейс `IDesignTimeDbContextFactory`. У даному прикладі розглянемо перший варіант.

Створіть проєкт `CinemaMVC.WebMVC` (замініть на назву вашого проєкту) типу ASP.NET Core Web Application (Model-View-Controller).

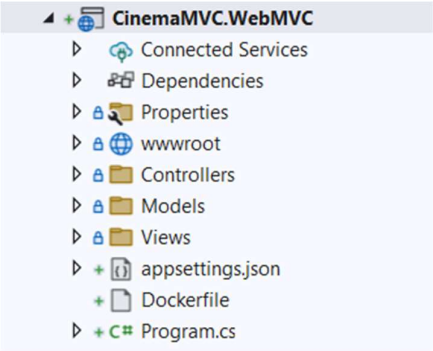


Рисунок 1.1.4 – Структура створеного проєкту типу ASP.NET Core Web Application (Model-View-Controller)

Відкрийте файл `appsettings.Development.json`. Додайте рядок підключення до секції `ConnectionStrings`:

```
{
  "ConnectionStrings": {
    "CinemaContext": "Data
Source=(LocalDb)\\MSSQLLocalDB;Initial Catalog=CinemaDb;Integrated
Security=true"
  }
}
```

Зверніть увагу, формат рядка підключення може відрізнятися в залежності від типу SQL Server інстансу.

Додайте у файл Program.cs перед викликом `builder.Build()`:

```
builder.Services
    .AddDbContext<CinemaContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString(n
ameof(CinemaContext))
        , sqlOptions =>
sqlOptions.MigrationsAssembly(typeof(Program).GetTypeInfo().Asse
mbly.GetName().Name));
```

Таким чином, ви додаєте ваш контекст у IoC контейнер та задаєте налаштування для підключення до бази даних. Зверніть увагу, для не SQL Server сховищ даних, виклик буде іншим. Рекомендовано звернутися до документації.

Також у даному виклику задається збірка для міграцій, в даному випадку це `CinemaMVC.WebMVC`.

Після підключення контексту, додайте до вашої збірки `CinemaMVC.WebMVC` пакети `Microsoft.EntityFrameworkCore.Tools` та `Microsoft.EntityFrameworkCore.Design`. Версія пакетів має сходитися з `Microsoft.EntityFrameworkCore` версією у проєкті `CinemaMVC.Infrastracture`.

Для створення міграції, перейдіть у консоль пакетного менеджера NuGet.

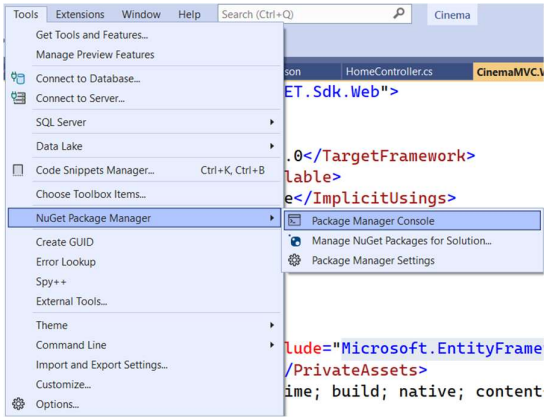


Рисунок 1.1.5 – Відкриття консолі пакетного менеджера NuGet

У консолі пакетного менеджера створіть нову міграцію використовуючи команду `Add-Migration`. В якості проєкту за замовчуванням виставте `CinemaMVC.WebMVC`.

```
Add-Migration Initial -Context CinemaMVC.Infrastructure.CinemaContext -OutputDir Infrastructure/Migrations
```

У даному прикладі, міграція створюється з ім’ям `Initial` у папку `Infrastructure/Migrations` на основі контексту (з обов’язковим вказанням простору імен) `CinemaMVC.Infrastructure.CinemaContext`.

Після успішного створення міграції, виконайте її над вашою базою даних використовуючи команду `Update-Database`.

Відкрийте SQL Server Management Studio або інше середовище для роботи з базами даних і перевірте згенеровану базу. Порівняйте діаграму класів з рисунків 1.1.3 та ER діаграму з 1.1.5.

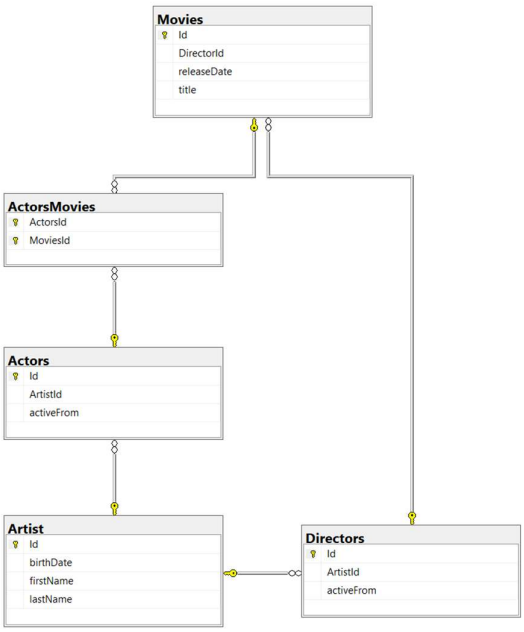


Рисунок 1.1.5 – ER-діаграма створеної бази даних

КРОКИ ВИКОНАННЯ ЕТАПІВ 1.2 та 1.3
НАВЕДЕНО ЛИШЕ ІЛЮСТРАТИВНІ ПРИКЛАДИ. ПРИ РЕАЛІЗАЦІЇ СВОГО ПРОЄКТУ ВИ ПОВИННІ РЕАЛІЗУВАТИ **ПОВНОФУНКЦІОНАЛЬНИЙ** ДОДАТОК!!!

Створення контролера для Entity Framework

Того, щоб виводити інформацію на вебсторінку, щоб користувачі бачили перелік категорій книжок і змогли б обрати одну з них потрібно створити метод (дію контролера) і представлення, яке відповідатиме за відображення потрібної інформації. Продемонструємо створення контролера для класу моделі Entity Framework.

Для цього перейдемо до папки Controllers, де поки що знаходиться єдиний контролер HomeController. На папці Controllers натиснемо праву кнопку миші та виберемо (Рис. 1.3.4 - 1.3.5).

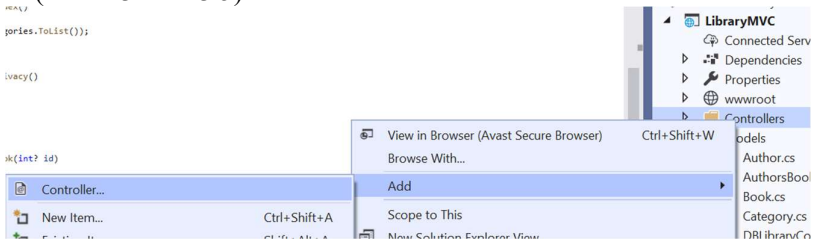


Рисунок 1.3.4 - Фрагмент вікна вибору контролеру

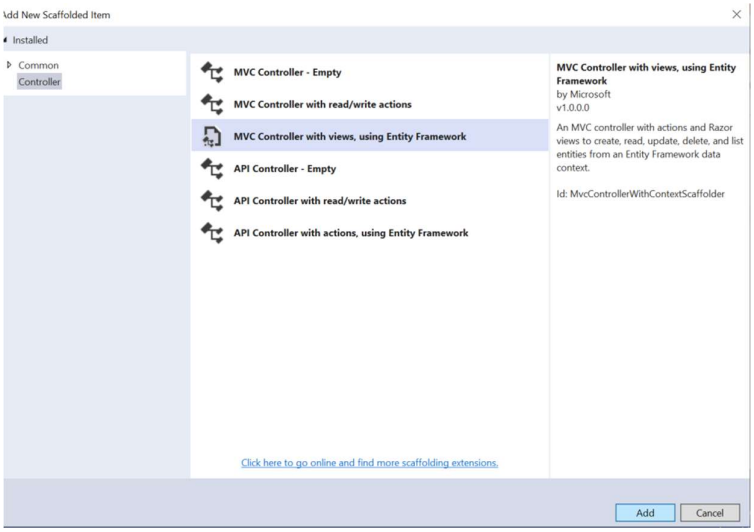


Рисунок 1.3.5 - Фрагмент вікна вибору контролеру

В налаштуваннях виберемо потрібний клас моделі та клас контексту даних. Поле з layout ПОКИ що (до наступного етапу) залишаємо порожнім (Рисунок 1.56).

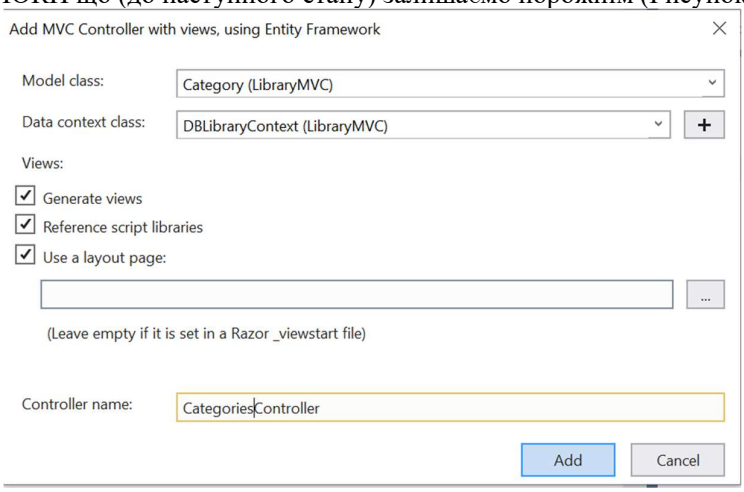


Рисунок 1.3.6 - Фрагмент вікна вибору класу моделі контролеру

Зверніть увагу, що щойно було згенеровано не лише файл контролера, а і низка файлів моделі. Перегляньте файли, що згенерувалися (поки що без внесення змін) (Рис. 1.3.7).

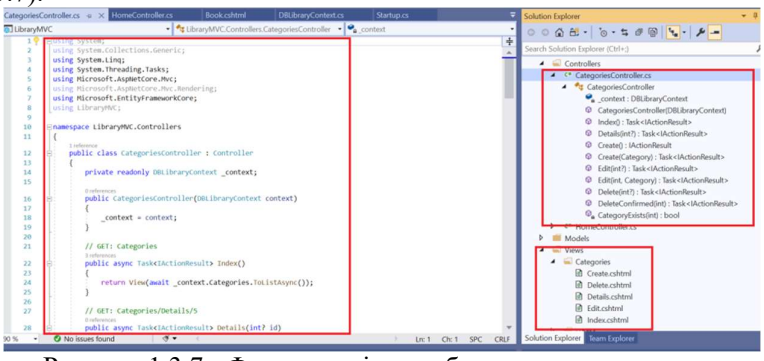


Рисунок 1.3.7 - Фрагмент вікна роботи з контролером

Проте, якщо зараз запустити додаток, то ці сторінки не будуть запущені за замовчуванням. Для того, щоб зробити саме сторінку Index контролера Categories такою, що запускається по замовчуванню внесемо зміни у файл Startup.cs (в корені проєкту) (Рис. 1.3.8).

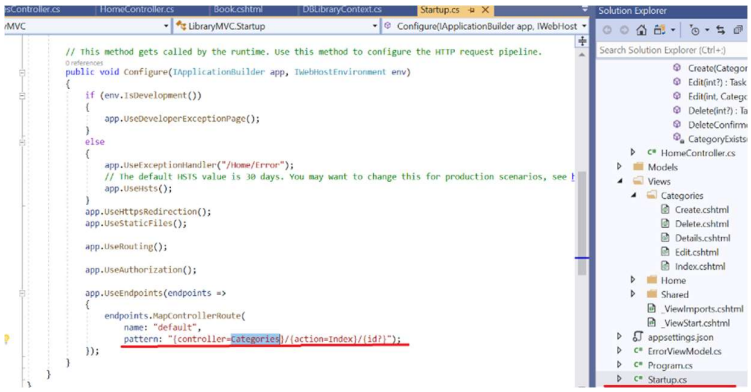


Рисунок 1.3.8 - Фрагмент вікна внесення зміни у файл Startup.cs

Запустимо проєкт. Неважко переконатися, що ми отримала достатньо багато реалізованого функціоналу (Рис. 1.3.9 – 1.3.11).

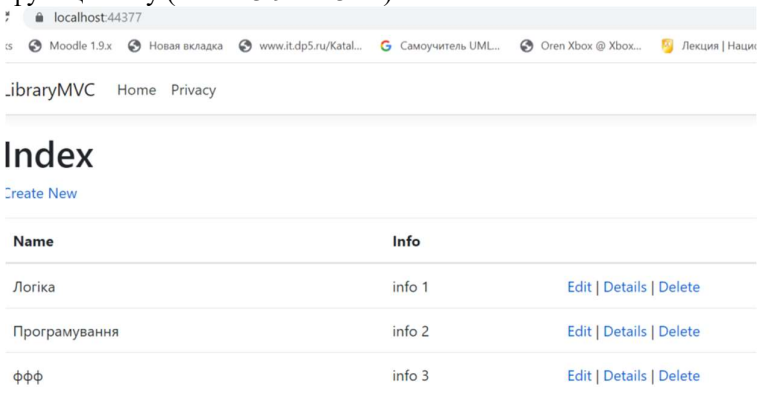


Рисунок 1.3.9 - Фрагмент вікна реалізації

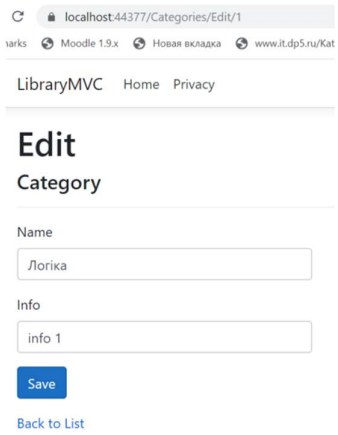


Рисунок 1.3.10 - Фрагмент вікна реалізації

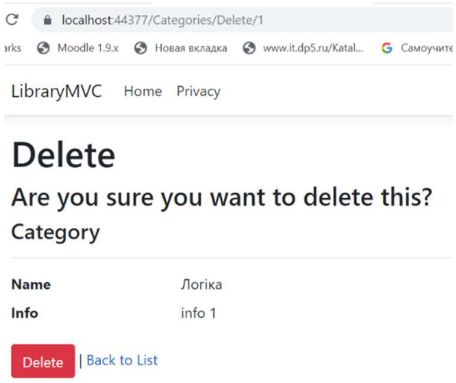


Рисунок 1.3.11 - Фрагмент вікна реалізації

Проте, не все нам подобається. Зверніть увагу на назви, які відображаються, вони нам зовсім не подобаються.

Переглянемо (поки не змінюємо) файл Index.cshtml (в папці Views/Categories) (Рис. 1.3.12).

Рисунок 1.3.12 - Фрагмент вікна файлу Index.cshtml

Як видно, ці назви в явному вигляді не виводяться. Проблема в моделі. Внесемо зміни (додамо атрибути) у файл з моделлю Category (Рис.1.3.13).

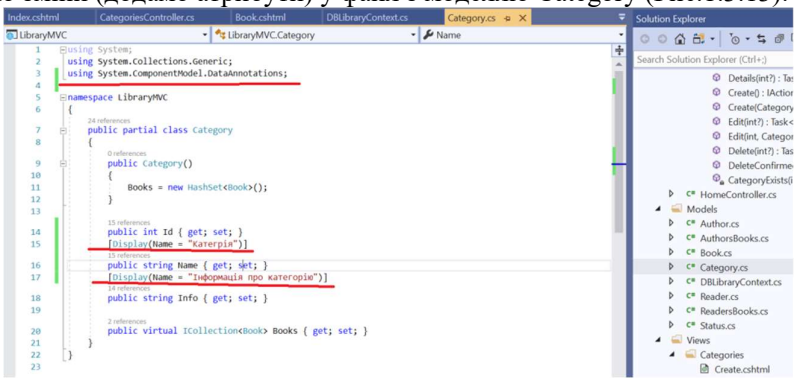


Рисунок 1.3.13 - Фрагмент вікна файлу з моделлю Category

Запускаємо (Рис. 1.3.14).

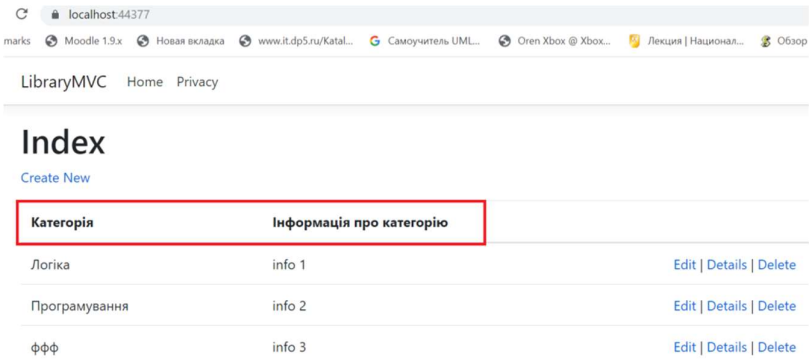


Рисунок 1.3.14 - Фрагмент робочого вікна програми

Але, ми і досі не задоволені. Зокрема, при створенні нової категорії поле з іменем можна залишити порожнім, але при збереженні в БД виникає помилка, адже там порожнє ім'я заборонене (Рис. 1.3.15).

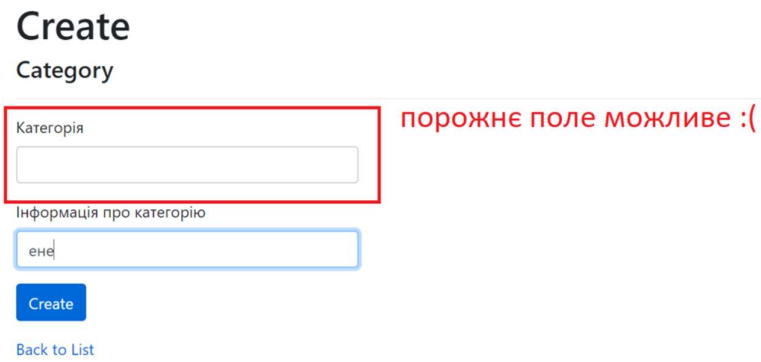


Рисунок 1.3.15 – Фрагмент вікна створення нової категорії

Для валідації (перевірки коректності) даних знову скористаємося додаванням атрибутів в модель (з більшою кількістю таких атрибутів ознайомтеся в лекції) (Рис. 1.3.16).

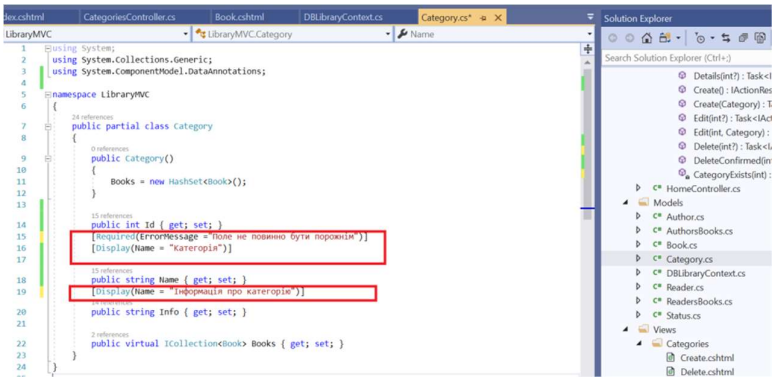


Рисунок 1.3.16 – Фрагмент коду

Запускаємо (Рис.1.3.17)

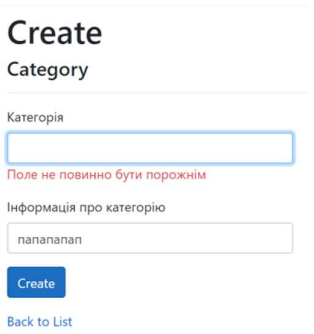


Рисунок 1.3.17 – Фрагмент створення категорії

Тепер, нам не подобається вікно з детальною інформацією про категорію книг (Details). На ньому, крім інформації про назву категорії бажано було б мати і перелік книг з цієї категорії з можливістю їх додавання перегляду та вилучення (Рис.1.3.18).

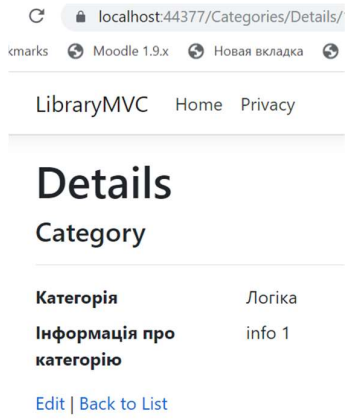


Рисунок 1.3.18 – Фрагмент вікна інформації про категорію

Створимо контролер для класу моделі Book (аналогічно до Categories) (Рис. 1.3.19 – 1.3.22).

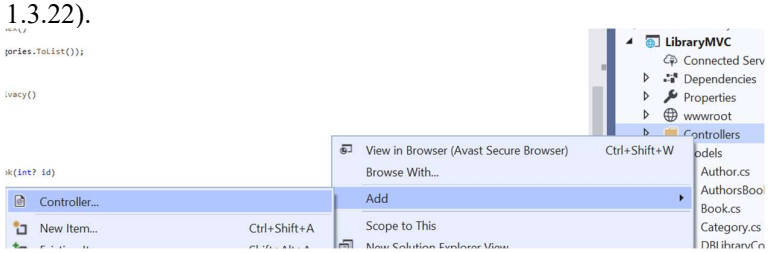


Рисунок 1.3.19 – Фрагмент вікна створення контролеру

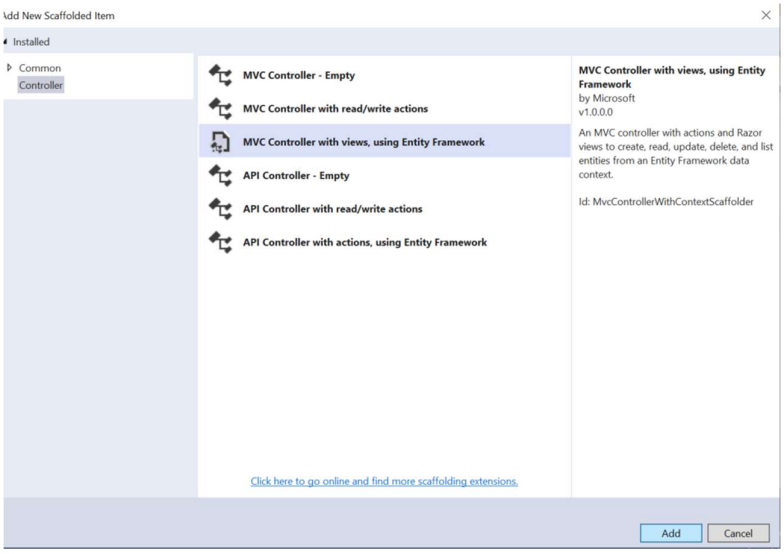


Рисунок 1.3.20 – Фрагмент вікна роботи з контролером

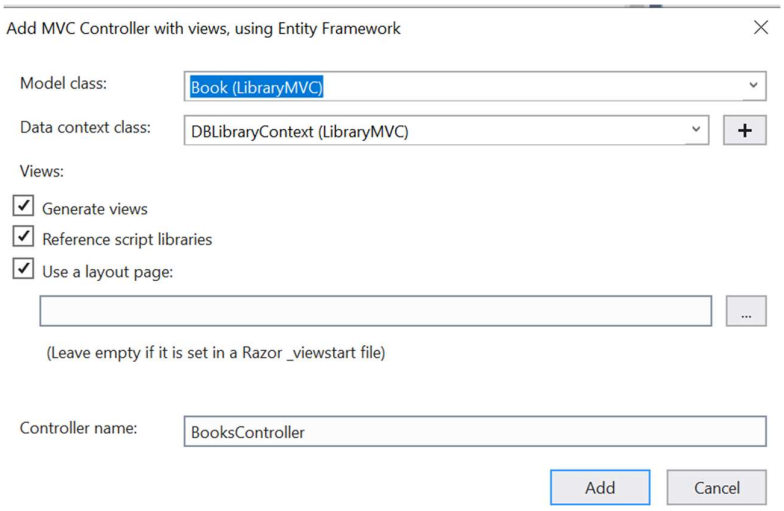


Рисунок 1.3.21 – Фрагмент вікна роботи з контролером

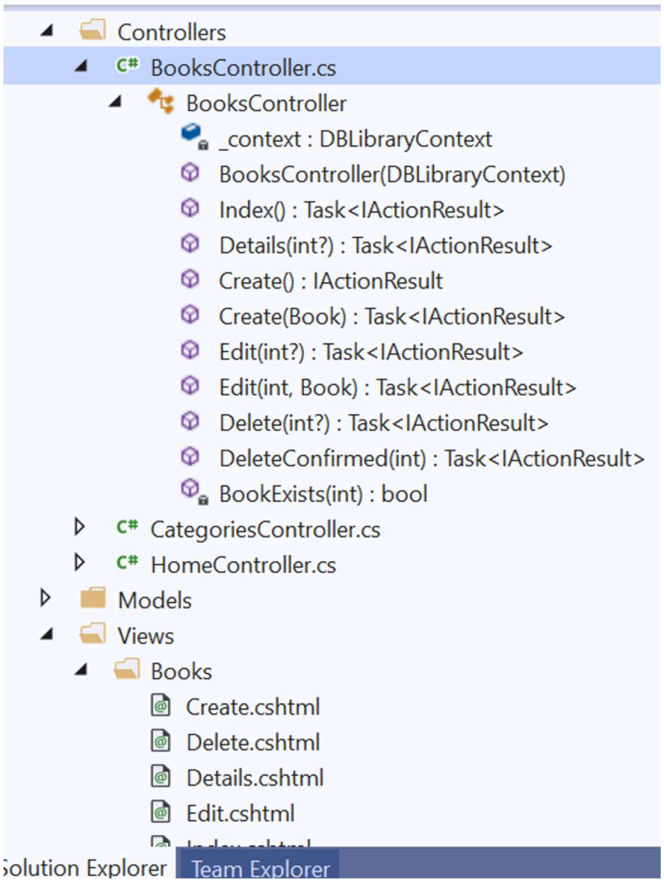


Рисунок 1.3.22 – Фрагмент вікна перегляду контролерів

Для того, щоб при натисканні на посилання «Details» з інформацією про потрібну категорію ми переходили на відображення книг за цією категорією внесемо зміни в контролер категорій (Рис.1.3.23).

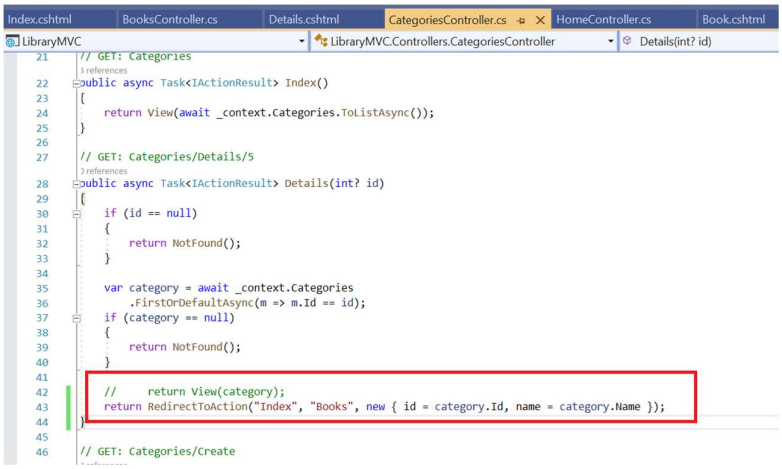


Рисунок 1.3.23 – Фрагмент коду

Тепер «навчимо» метод Index контролера «Books» розуміти інформацію про категорію і обробляти її. Для цього внесемо зміни у відповідний файл (Рис.1.3.24)

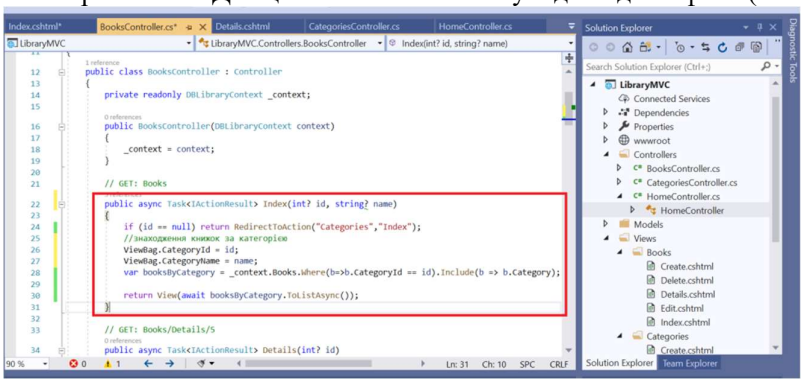


Рисунок 1.3.24 – Фрагмент коду

Для коректного відображення сторіки з книжками внесемо зміни і у файл відображення книжок “Index” (Рис. 1.3.25).

Рисунок 1.3.25 – Фрагмент коду з внесеними змінами

Можна запускати (Рис.1.3.26 – 1.3.27).

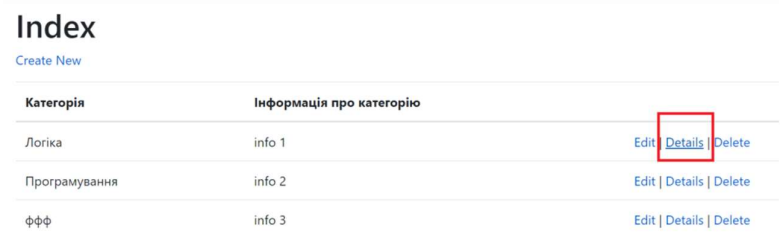


Рисунок 1.3.26 – Фрагмент вікна запуску програми

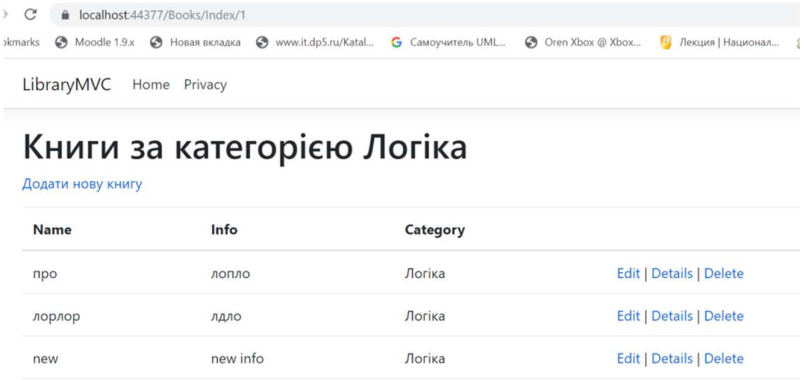


Рисунок 1.3.27 - Фрагмент вікна запуску програми

З правильними підписами та валідацією даних розберетеся самостійно.

Розберемося з додаванням нової книги в категорію, а саме опрацюємо посилання «Create», яке ми попередньо додали у представлення (рядок 10) (Рис.1.3.28).

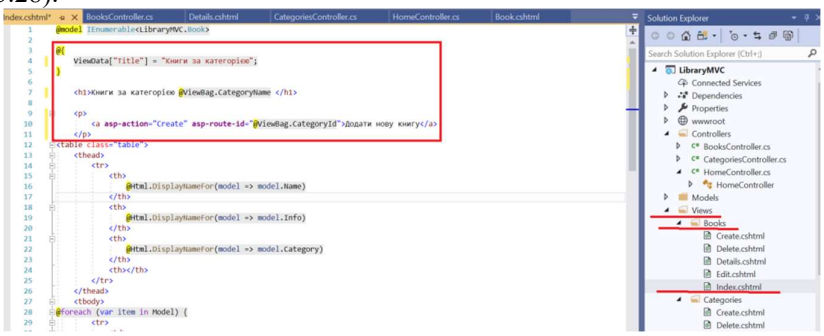


Рисунок 1.3.28 – Фрагмент коду опрацювання посилання «Create»

Для цього потрібно внести зміни у відповідний метод контролера Books та у відповідне представлення. Тепер цей метод повинен приймати у якості параметра ідентифікатор категорії (Рис. 1.3.29 – 1.3.31)

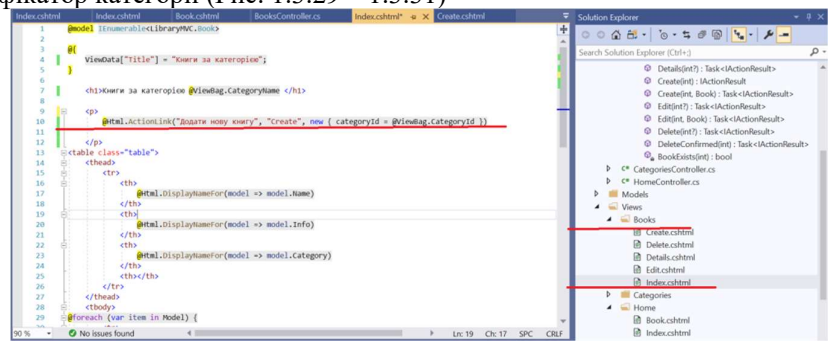


Рисунок 1.3.29 – Фрагмент вікна коду

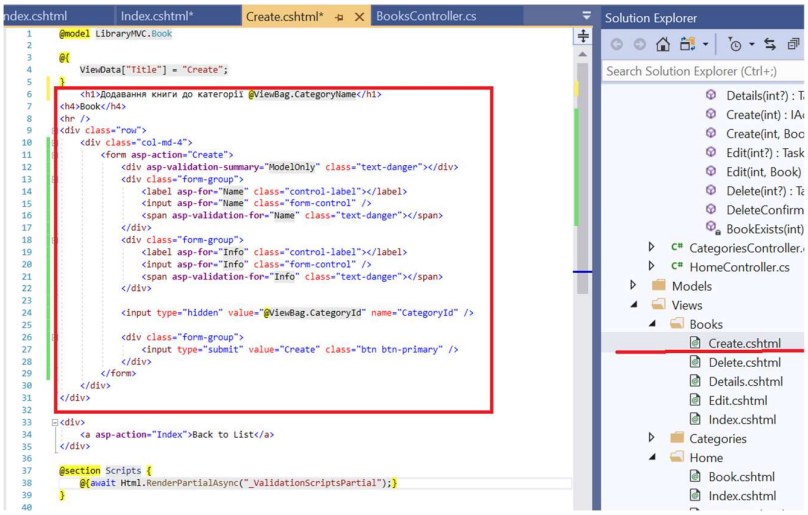


Рисунок 1.3.30 – Фрагмент вікна коду

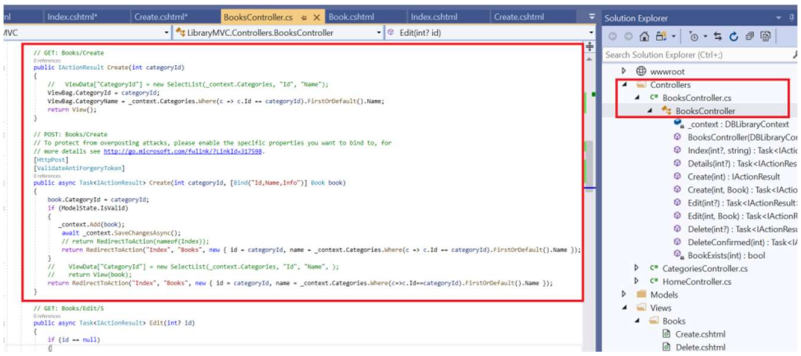


Рисунок 1.3.31 – Фрагмент вікна коду

Далі продовжуєте реалізацію.

Для отримання балів за етап – реалізуйте повнофункціональний додаток. Складність залежить він предметної області.

КРОКИ ВИКОНАННЯ ЕТАПУ 1.4

Отже, у нас є вже примітивний функціонал для виведення на вебсторінку категорій книг і додавання до них книг. Але розглянемо ще один аспект – створення однакового виду програми та дизайн.

У нас є декілька представлень, які фактично нагадують вебсторінки. І якщо ми захочемо застосувати до кожної вебсторінки будь-які стилі із зовнішнього файлу css, то нам доведеться в кожному представленні підключити цей файл стилів. Також якщо ми захочемо визначити загальне для всіх вебсторінок меню або якісь інші загальні елементи, наприклад, футер, то знову ж таки нам доведеться прописувати всі ці елементи в кожному представленні. Це не є оптимальною практикою, так як якщо нам треба внести зміни в такі загальні речі, то доведеться змінювати всі представлення, яких може бути в проєкті достатньо багато.

Набагато більш оптимальним способом є використання майстер-сторінок. За замовчуванням в проєкті ASP.NET MVC вже є майстер-сторінка, яка називається _Layout.cshtml і яка знаходиться в папці Views / Shared. Трохи змініть її під потреби свого проєкту (Рис. 1.4.1)

Рисунок 1.4.1 – Фрагмент вікна коду для використання майтер-сторінок

Майстер-сторінка являє собою звичайне представлення, яке включає в себе інші окремі представлення.

Спочатку підключаються стилі бібліотеки Bootstrap, яка за замовчуванням вже є в проєкті в папці wwwroot / lib / bootstrap / dist / css / (Рис. 1.4.2).

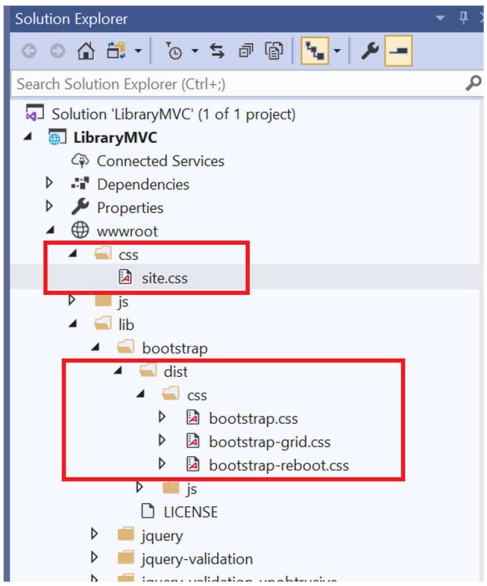


Рисунок 1.4.2 – Фрагмент вікна зміни стилів

Після секції head на майстер-сторінці йде створення меню. В якості одного єдиного пункту меню вказується посилання на головну сторінку Home. Для створення меню тут застосовуються стандартні класи Bootstrap.

Далі в основній частині йде виклик методу **RenderBody() – за допомогою цього методу в це місце буде підставлятися розмітка вже конкретних представлень**.

Зауваження: ***згадайте, як при створенні нових представлень при vb ми уникали відповіді на питання про Layout*** – тому, використовується майстер-сторінка по замовчуванню**.**

Це прописано на рисунку 1.4.3.

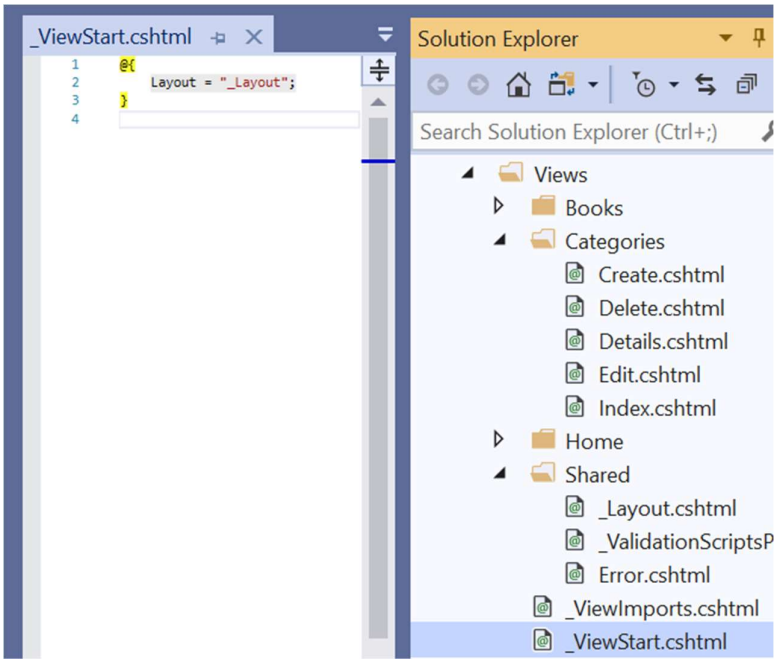


Рисунок 1.4.3 – Фрагмент вікна коду

Запустіть: зміни в _Layout.cshtml вплинули на УСІ сторінки (якщо на них явно не вказана інша майстер-сторінка).

Але, тема по замовчуванню зовсім нам не подобається. Для її зміни, або «руками» змінюйте CSS, або пошукаємо більш цікаву безкоштовну тему.

Для зміни теми bootstrap можна здійснити наступні дії:

Перейдіть на сайт, зазначений на рисунку 1.4.4 та оберіть тему, яка Вам сподобається:

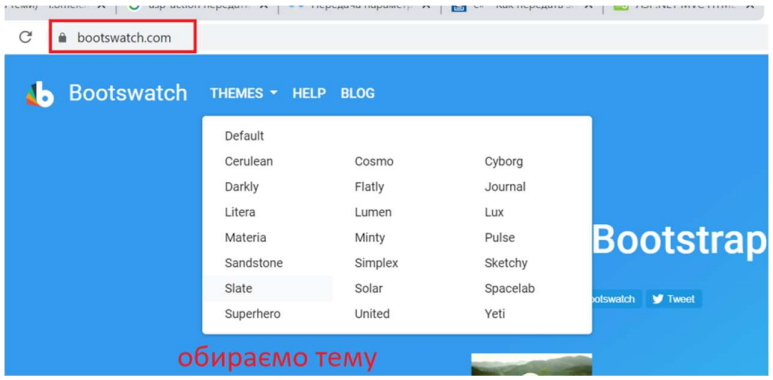


Рисунок 1.4.4 – Фрагмент вікна сайту

Завантажте потрібні файли (рис.1.4.5 – 1.4.6).

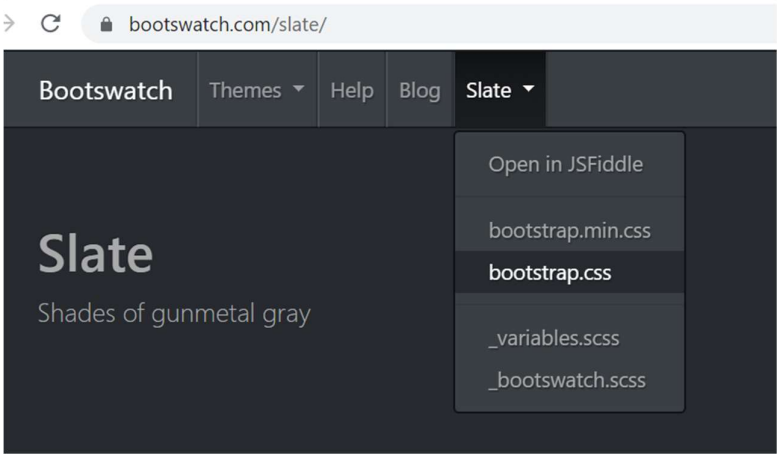


Рисунок 1.4.5 – Фрагмент вікна сайту

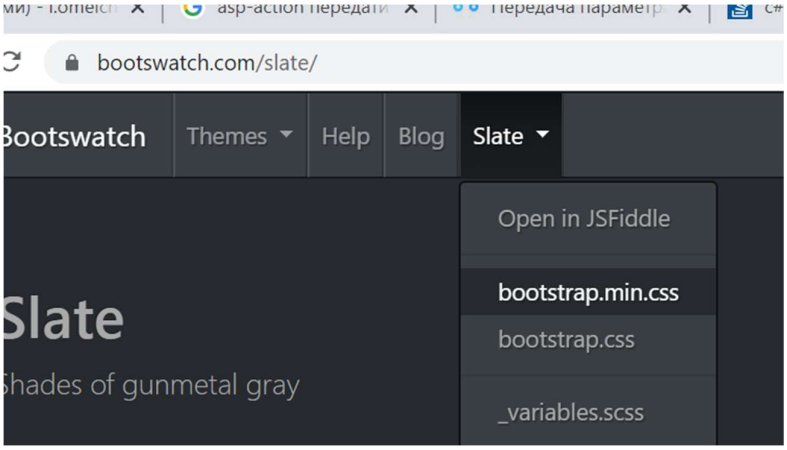


Рисунок 1.4.6 – Фрагмент вікна сайту

Для зручності перейменуйте їх (Рис. 1.4.7).

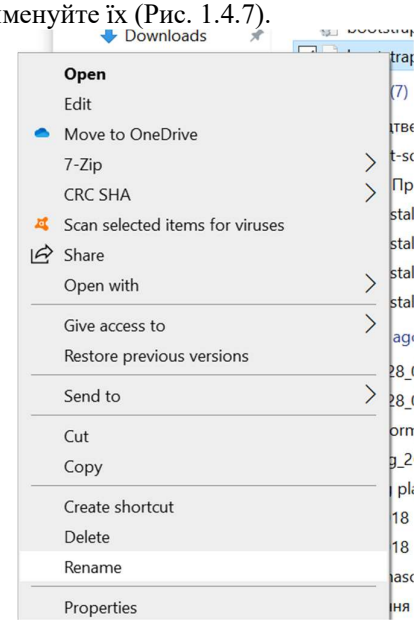


Рисунок 1.4.7 – Фрагмент вікна контекстного меню

Наприклад на:
bootstrap_state
та bootstrap_state.min
Додайте до проєкту (Рис. 1.4.8 – 1.4.11).

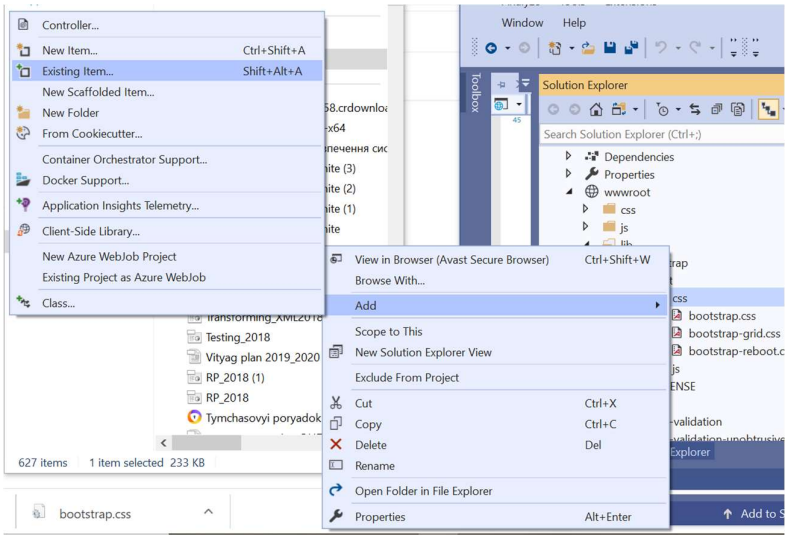


Рисунок 1.4.8 – Фрагмент вікна контекстного меню

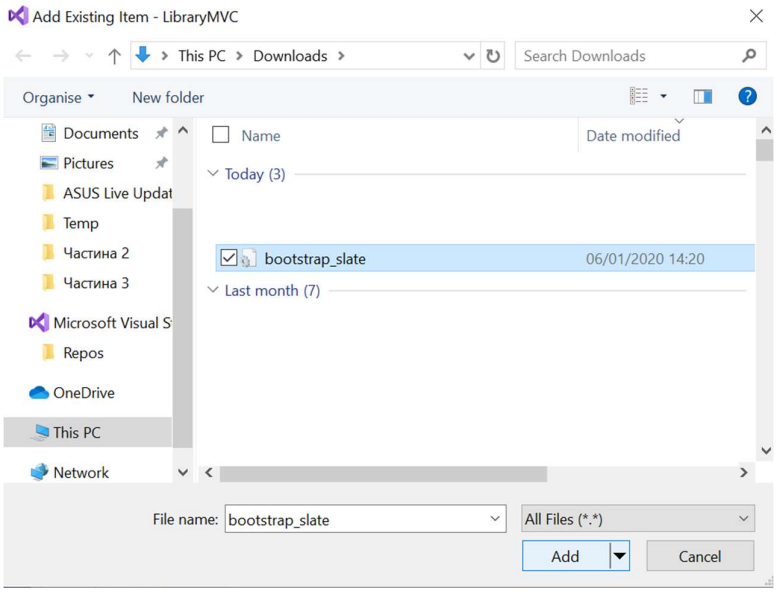


Рисунок 1.4.9 – Фрагмент вікна зміни імені

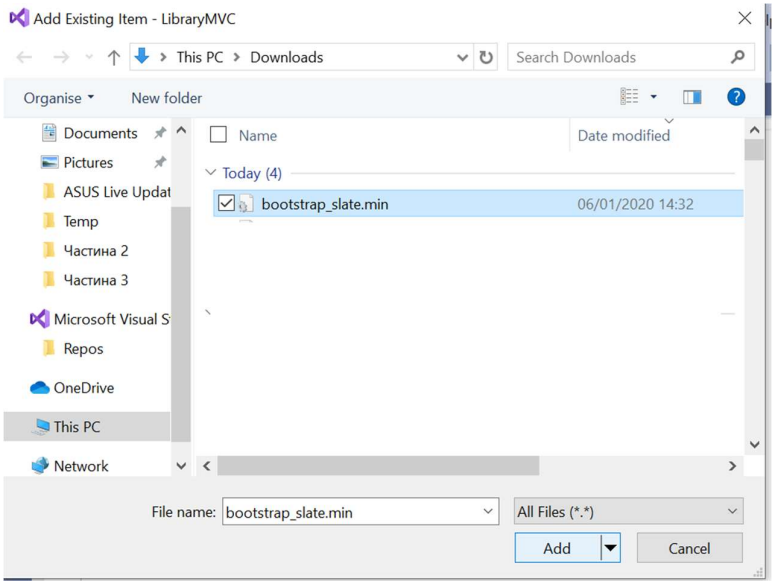


Рисунок 1.4.10 – Фрагмент вікна створення

Отримали:

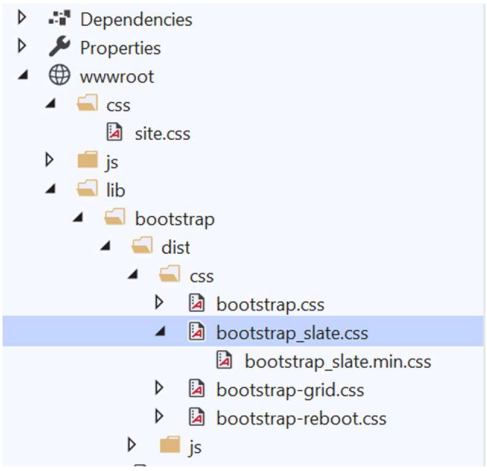


Рисунок 1.4.11 – Фрагмент вікна доступу до файлу

Змініть майстер-сторінку (посилання на новий файл) (Рис. 1.4.12).

Рисунок 1.4.12– Фрагмент вікна зміни коду

Запускайте (Рис. 1.4.13).

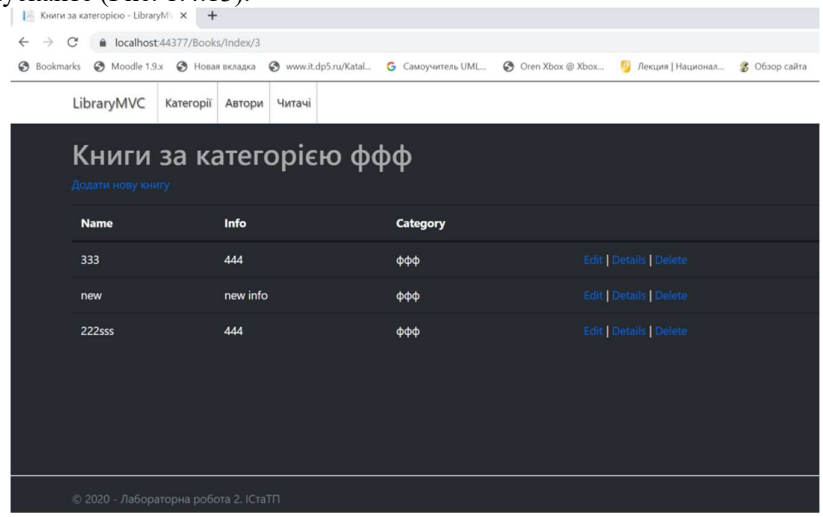


Рисунок 1.4.13 – Фрагмент вікна запуску програми

Якщо результат не сподобався, то змінюйте вручну, або шукайте нову тему.

Для отримання балів за етап – дизайн Вашого проєкту повинен сподобатися замовнику (у Вашому випадку – викладачу).

КРОКИ ВИКОНАННЯ ЕТАПУ 1.5

Одним з найбільш поширених графічних завдань є побудова діаграм. Наприклад, на сторінці, яка відповідає за відображення списку фільмів, необхідно додати діаграму, що показує кількість фільмів випущених у певний рік.

Існує безліч біблотек з готовими рішеннями для відображення діаграм на вебсторінці. В даному прикладі використано бібліотеку Google Charts (див. https://developers.google.com/chart).

При цьому, дані ми будемо передавати зі сторони клієнта. В папці з контролерами створимо новий контролер (але не MVC, а API контролер), який назвемо `ChartsController` (рис. 1.5.1 – 1.5.2).

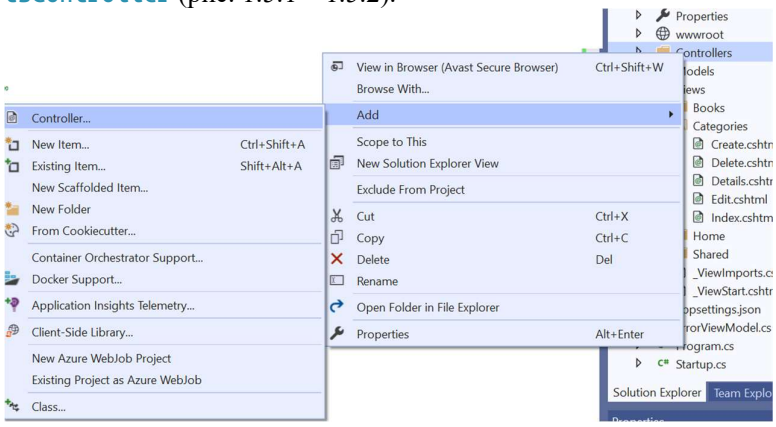


Рисунок 1.5.1 – Фрагмент вікна створення нового контролеру

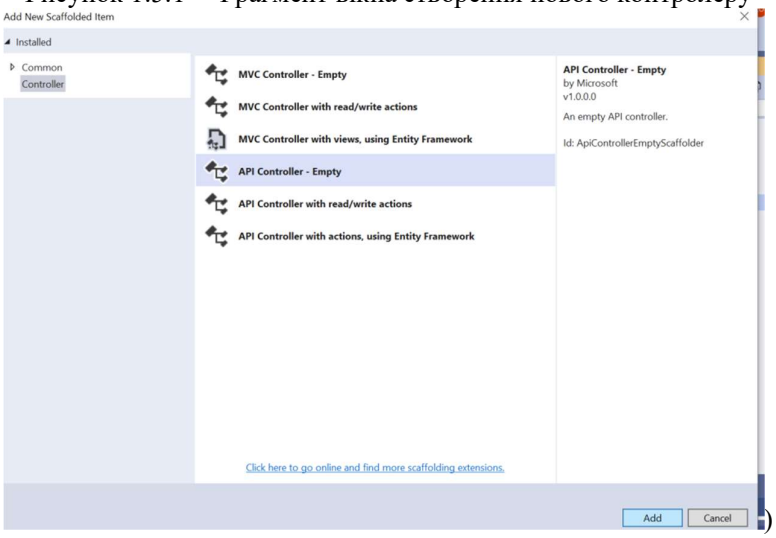


Рисунок 1.5.2 – Фрагмент вікна створення контролеру

Визначимо в ньому метод, який повертає в якості JsonResult дані з роками та кількістю фільмів за цей рік:

```
namespace CinemaMVC.WebMVC.Controllers;

[Route("api/[controller]")]
[ApiController]
public class ChartsController : ControllerBase
{
    private record CountByYearResponseItem(string Year, int
Count);

    private readonly CinemaContext cinemaContext;

    public ChartsController(CinemaContext cinemaContext)
    {
        this.cinemaContext = cinemaContext;
    }

    [HttpGet("countByYear")]
    public async Task<JsonResult>
GetCountByYearAsync(CancellationToken cancellationToken)
    {
        var responseItems = await cinemaContext
            .Movies
            .GroupBy(movie => movie.ReleaseDate.Year)
            .Select(group => new
CountByYearResponseItem(group.Key.ToString(), group.Count()))
            .ToListAsync(cancellationToken);

        return new JsonResult(responseItems);
    }
}
```

Щодо створення таких методів є декілька порад:

- створіть чітку модель даних (клас, який буде серіалізуватися у JSON), не повертайте анонімні об’єкти;
- іменуйте шлях запиту максимально зрозуміло;
- слідкуйте за формуванням вашого запиту з використанням LINQ та намагайтеся винести виконання операцій групування та проєкції на сторону СУБД.

Для того, щоб вивести графік на сторінку, у коді сторінки необхідно зробити запит на вивантаження даних з новоствореного методу та ініціалізувати графік

використовуючи засоби Google Charts. Додамо наступний код у файл Views/Movies/Index.cshtml:

```
<div class="row">
    <div class="col-3">
        <div id="countByYearChart"></div>
    </div>
</div>

@section Scripts
{
    <script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
    <script type="text/javascript">
        google.charts.load('current',
{'packages':['corechart']});
        google.charts.setOnLoadCallback(drawCharts);

        function drawCharts() {
            fetch('/api/charts/countByYear')
                .then(response => response.json())
                .then(data => {
                    const dataTable = new
google.visualization.DataTable();
                    dataTable.addColumn('string', 'Рік');
                    dataTable.addColumn('number', 'Кількість
фільмів');

                    data.forEach(item => {
                        dataTable.addRow([item.year,
item.count]);
                    });

                    const options = {
                        title: 'Стрічки за роками',
                        width: 600,
                        height: 400,
                        legend: { position: 'none' },
                    };

                    const chart = new
google.visualization.ColumnChart(document.getElementById('countB
yYearChart'));

                    chart.draw(dataTable, options);
                });
        }
    </script>
}
```

Після запуску вебзастосунку, на сторінці відобразиться графік (рис. 1.5.3).

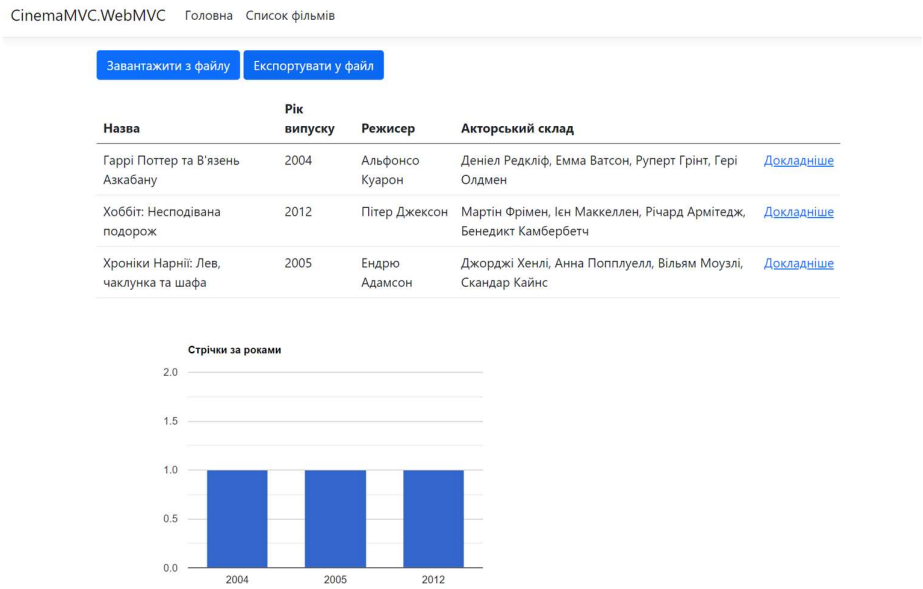


Рисунок 1.5.2 – Вигляд діаграми на сторінці

Для отримання балів за етап – створіть в своєму проєкті мінімум 2 діаграми з різними наборами даних (максимум 1 бал за діаграму, але не більше 2-х балів з урахуванням термінів).

КРОКИ ВИКОНАННЯ ЕТАПУ 1.6

Імпорт даних з Excel-файлів

Користувацький інтерфейс не завжди є найпростішим для заповнення великих баз даних. Зазвичай на етапі впровадження інформаційної системи необхідно внести у систему дані з попередніх засобів керування та зберігання даних. Такі модулі програм називають портами даних (англ. – «*Data Port*»). Оскільки наша система працює з табличними базами даниз (в залежності від обраної СУБД), природнім є додавання порту даних, що сприймає дані у найрозповсюдженішому форматі збереження таблиць – файлах Microsoft Excel. Справді, користувачі вашого програмного забезпечення, до впровадження вашого нового рішення, можуть зберігати дані у файлах Microsoft Excel (Google Sheets, або інші) або у системах, що теж мають порти даних для вивантаження у форматі Excel-файлів. Таким чином, впровадження вашої інформаційної системи буде більш природнім та швидким.

Відповідно для вашої інформаційної системи, можуть бути обрані різні формати вивантаження. Для початку створіть приклад Excel-файлу, який ви будете очікувати на вхід. Наприклад, для предметної області «Кінотеатр» файл буде зображати список фільмів, дату випуску, режисера та скорочений список акторів (рис. 1. 6. 1).

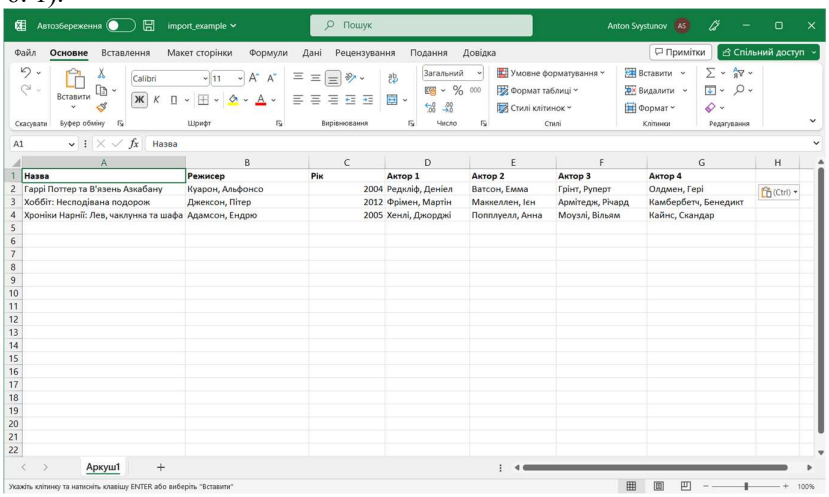


Рисунок 1.6.1 – Фрагмент вікна програми

Для створення та взаємодії з Excel файлами існують різні бібліотеки, оскільки формат Excel-файлів є відкритим. У даному прикладі розглядається бібліотека версії `ClosedXML` 0.100.3. Зверніть увагу, інтерфейси бібіотек змінюються, тому рекомендовано звернутися до докумнтації (вихідного коду) вашої поточної версії.

Для завантаження, відкрийте менеджер NuGet та додайте пакет `ClosedXML` в проєкт `CinemaMVC.WebMVC` (замініть на назву вашого проєкту) (рис. 1.6.2).

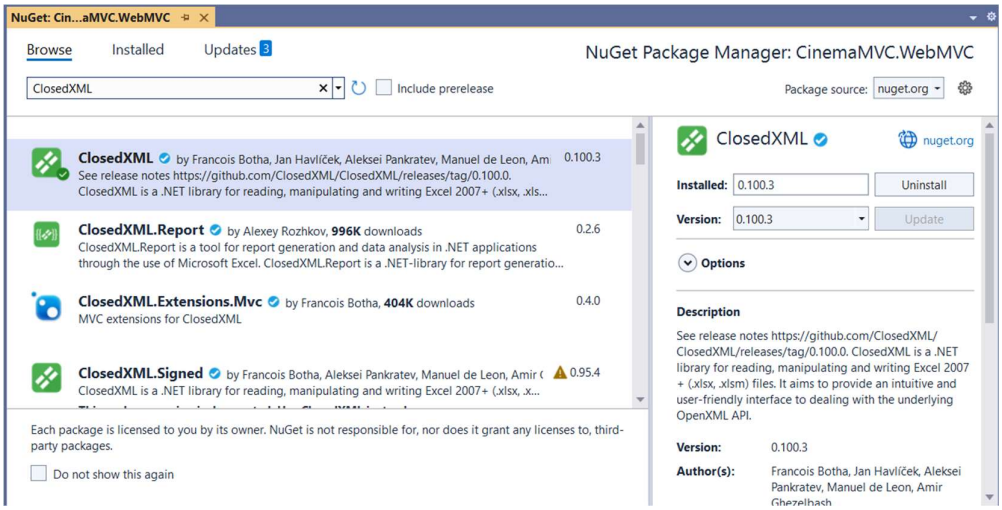


Рисунок 1.6.2 – Бібліотека ClosedXML у вікні пакетного менеджера NuGet

Почати необхідно з означення інтерфейсу для імпорту даних з Excel-файлу:

```
namespace CinemaMVC.WebMVC.Infrastructure.Services;

public interface IImportService<TEntity>
    where TEntity : Entity
{
    Task ImportFromStreamAsync(Stream stream, CancellationToken
cancellationToken);
}
```

Далі означимо інтерфейс фабрики, що буде повертати реалізацію цього інтерфейсу в залежності від типу вхідних даних:

```
using CinemaMVC.Domain.Entities;

namespace CinemaMVC.WebMVC.Infrastructure.Services;

public interface IDataPortServiceFactory<TEntity>
    where TEntity : Entity
{
    IImportService<TEntity> GetImportService(string
contentType);
```

```
    IExportService<TEntity> GetExportService(string
contentType);
}
```

Такий підхід дозволить в залежності від типу контенту на вхід віддати сервіс, що сприйматиме цей тип даних у потоці, що подаватиметься на вхід. Далі необхідно реалізувати сервіс для імпорту даних стрічок з Excel-файлу:

```
namespace CinemaMVC.WebMVC.Infrastructure.Services;

public class MovieImportService : IImportService<Movie>
{
    private readonly CinemaContext context;
      // реалізація AddMovieAsync та інших методів

    public MovieImportService(CinemaContext context)
    {
        this.context = context;
    }

    public async Task ImportFromStreamAsync(Stream stream,
CancellationToken cancellationToken)
    {
        if (!stream.CanRead)
        {
            throw new ArgumentException("Stream is not
readable", nameof(stream));
        }

        using var workBook = new XLWorkbook(stream);
        var worksheet = workBook.Worksheets.FirstOrDefault();
        if (worksheet is null)
        {
            return;
        }

        foreach (var rows in worksheet.RowsUsed().Skip(1)) //
пропустити перший рядок, бо це заголовок
        {
            await AddMovieAsync(rows, cancellationToken);
        }

        await context.SaveChangesAsync(cancellationToken);
    }
}
```

Спочатку додайте перевірку вхідних даних. У прикладі, перевіряється валідність потоку. Тут також рекомендовано означити клас винятку для вашої операції, щоб коректно видавати помилку при розборі даних у файлі. Рекомендовано не використовувати анти-паттерн «магічні числа та значення» і розділяти обробку відповідних сутностей у окремі методи. Розглянемо приклад реалізації методу `AddMovieAsync`:

```
private async Task AddMovieAsync(IXLRow row, CancellationToken
cancellationToken)
{
    var movieTitle = GetMovieTitle(row);

    var movie = await context.Movies.FirstOrDefaultAsync(movie
=> movie.Title == movieTitle, cancellationToken);

    if (movie is null)
    {
        movie = new Movie(movieTitle, GetReleaseDate(row));
        context.Add(movie);
    }

    if (movie.Director is null)
    {
        var director = await GetDirectorAsync(row,
cancellationToken);
        movie.SetDirector(director);
    }

    if (!movie.Actors.Any())
    {
        var actors = await GetActorsAsync(row,
cancellationToken);

        foreach (var actor in actors)
        {
            movie.AddActor(actor);
        }
    }
}
```

Зверніть увагу, що в даному методі не використовується жодних методів об'єкту `IXLRow` – всі специфічні маніпуляції з даними винесені в окремі методи, наприклад `GetMovieTitle` та `GetReleaseDate`:

```
private static string GetMovieTitle(IXLRow row)
{
    return row.Cell(1).GetValue<string>();
}
```

```
private static DateTime GetReleaseDate(IXLRow row)
{
    return DateTime.ParseExact(row.Cell(3).GetValue<string>(),
"yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None);
}
```

Обробка кожної сутності відбувається за наступною послідовністю дій:

- Отримати ідентифікуючі дані з Excel.
- Знайти сутність у базі даних за отриманими даними.
- Якщо сутність не знайдено – створити новий екземпляр і додати у відстежування контексту.

Окремі зв'язні сутності також мають свої методи для читання з Excel, наприклад,

```
private static (string firstName, string lastName)
GetNames(string fullName)
{
    var names = fullName.Split(", ").Reverse();

    return (names.FirstOrDefault(string.Empty),
names.LastOrDefault(string.Empty));
}

private async Task<Artist> GetArtistAsync(string fullName,
CancellationToken cancellationToken)
{
    var (firstName, lastName) = GetNames(fullName);

    var artist = await
context.Set<Artist>().FirstOrDefaultAsync(artist =>
artist.FirstName == firstName && artist.LastName == lastName,
cancellationToken);

    if (artist is null)
    {
        artist = new Artist(firstName, lastName,
DateTime.UtcNow);
        context.Add(artist);
    }

    return artist;
}

private async Task<Director> GetDirectorAsync(IXLRow row,
CancellationToken cancellationToken)
{
```

```
    var artist = await
GetArtistAsync(row.Cell(2).GetValue<string>(),
cancellationToken);

    Director? director = await
context.Directors.FirstOrDefaultAsync(director =>
director.Artist.Id == artist.Id, cancellationToken);
    if (director is null)
    {
        director = new Director(artist);
        context.Add(director);
        return director;
    }

    return director;
}

private async Task<IReadOnlyCollection<Actor>>
GetActorsAsync(IXLRow row, CancellationToken cancellationToken)
{
    const int actorsCount = 4;
    const int baseIndex = 4;

    var result = new List<Actor>();

    for (var cellIndex = baseIndex; cellIndex < actorsCount +
baseIndex; cellIndex++)
    {
        var fullName = row.Cell(cellIndex).GetValue<string>();

        if (string.IsNullOrEmpty(fullName))
        {
            break;
        }

        var artist = await GetArtistAsync(fullName,
cancellationToken);

        var actor = await
context.Actors.FirstOrDefaultAsync(actor => actor.Artist.Id ==
artist.Id, cancellationToken);

        if (actor is null)
        {
            actor = new Actor(artist, DateTime.UtcNow);
            context.Add(actor);
        }
```

```
            result.Add(actor);
        }

        return result;
    }
```

У прикладі відсутні валідації і додаткова логіка, які можуть бути додані у реалізації відповідного сервісу в рамках завдання лабораторного практикуму. Далі необхідно реалізувати інтерфейс `IDataPortServiceFactory` для класу `Movie` для подальшого використання у контролері (реалізація умовна):

```
namespace CinemaMVC.WebMVC.Infrastructure.Services;

public class MovieDataPortServiceFactory
    : IDataPortServiceFactory<Movie>
{
    private readonly CinemaContext cinemaContext;

    public MovieDataPortServiceFactory(CinemaContext
cinemaContext)
    {
        this.cinemaContext = cinemaContext;
    }

    public IExportService<Movie> GetExportService(string
contentType)
    {
        if (contentType is "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet")
        {
            return new MovieExportService(cinemaContext);
        }

        throw new NotImplementedException($"No export service
implemented for movies with content type {contentType}");
    }

    public IImportService<Movie> GetImportService(string
contentType)
    {
        if (contentType is "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet")
        {
            return new MovieImportService(cinemaContext);
        }

        throw new NotImplementedException($"No import service
implemented for movies with content type {contentType}");
```

```
    }
}
```

Імпорт зазвичай реалізується окремою сторінкою у вебзастосунку, тому необхідно додати кнопку на сторінку списку стрічок для переходу на новостворену сторінку імпорту у `Views/Movies/Index.cshtml`:

```
<a class="btn btn-primary mb-3" type="submit" asp-area="" asp-controller="Movies" asp-action="Import">Завантажити з файлу</a>
```

У файл `Views/Movies/Import.cshtml` необхідно додати форму для додавання файлу:

```
<div class="row">
    <div class="col-12">
        @using (Html.BeginForm("Import", "Movies", FormMethod.Post, new { enctype = "multipart/form-data" }))
        {
            <div class="mb-3">
                <label for="moviesFile" class="form-label">Оберіть файл для заватаження</label>
                <input type="file" name="moviesFile" id="moviesFile" />
            </div>
            <div>
                <input class="btn btn-primary mb-3" type="submit" value="Завантажити" />
            </div>
        }
    </div>
</div>
```

Додамо відповідний метод у клас контролер `MoviesController`. Додавання обробки вийнятків і відображення помилок на сторінці пропущені:

```
[HttpGet]
public IActionResult Import()
{
    return View();
}

[HttpPost]
public async Task<IActionResult> Import(IFormFile moviesFile, CancellationToken cancellationToken)
{
    var importService = movieDataPortServiceFactory.GetImportService(moviesFile.ContentType);
```

```
        using var stream = moviesFile.OpenReadStream();

        await importService.ImportFromStreamAsync(stream,
    cancellationToken);

        return RedirectToAction(nameof(Index));
    }
```

Зверніть увагу, назва аргументу `IFormFile` методу контролера має бути ідентичною з назвою файлу у HTML-формі.

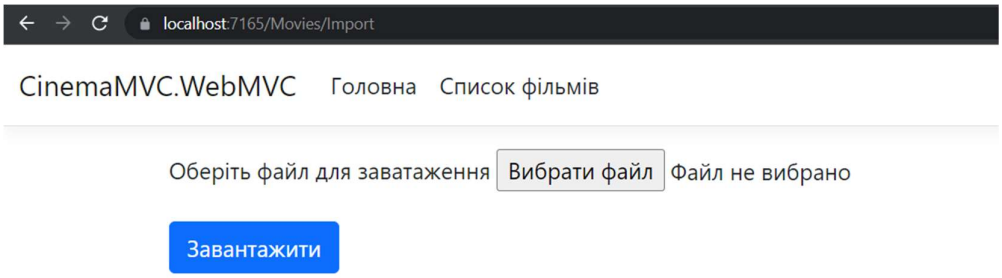


Рисунок 1.6.7 – Фрагмент вікна імпорту файлу

Експорт даних з Excel-файлів

З метою, вивантаження даних для подальшого використання у різного роду презентаціях, звітах, або додавання у інші інформаційні системи, необхідно забезпечити можливість експорту файлів у форматі Excel-файлу. У даному прикладі наведено вивантаження файлу подібної структури до імпорту, наведеному вище.

Для початку, необхідно означити інтерфейс для експорту даних у потік:

```
namespace CinemaMVC.WebMVC.Infrastructure.Services;

public interface IExportService<TEntity>
    where TEntity : Entity
{
    Task WriteToAsync(Stream stream, CancellationToken
cancellationToken);
}
```

У даному прикладі упускається можливість додавання фільтрації на дані, проте, це можна також реалізувати, розширивши список аргументів `WriteToAsync` колекцією `TEntity`. Реалізація цього інтерфейсу для цілової сутності будується за схожим принципом з `MovieImportService` (для кожного кроку – окреми метод, що приховує деталі реалізації запису у файл):

```
namespace CinemaMVC.WebMVC.Infrastructure.Services;

public class MovieExportService : IExportService<Movie>
{
    private const string RootWorksheetName = "Movies";

    private static readonly IReadOnlyList<string> HeaderNames =
new string[]
    {
        "Назва",
        "Режисер",
        "Рік",
        "Актор 1",
        "Актор 2",
        "Актор 3",
        "Актор 4",
    };

    private readonly CinemaContext context;

    private static string GetNameOrDefault(Artist? artist)
    {
        if (artist is null)
        {
            return string.Empty;
        }

        return $"{artist.LastName}, {artist.FirstName}";
    }

    private static void WriteHeader(IXLWorksheet worksheet)
    {
        for (int columnIndex = 0; columnIndex <
HeaderNames.Count; columnIndex++)
        {
            worksheet.Cell(1, columnIndex + 1).Value =
HeaderNames[columnIndex];
        }

        worksheet.Row(1).Style.Font.Bold = true;
    }
```

```
    private static void WriteMovie(IXLWorksheet worksheet, Movie
movie, int rowIndex)
    {
        var columnIndex = 1;

        worksheet.Cell(rowIndex, columnIndex++).Value =
movie.Title;
        worksheet.Cell(rowIndex, columnIndex++).Value =
GetNameOrDefault(movie.Director?.Artist);
        worksheet.Cell(rowIndex, columnIndex++).Value =
movie.ReleaseDate.Year;

        foreach (var actor in movie.Actors.Take(4))
        {
            worksheet.Cell(rowIndex, columnIndex++).Value =
GetNameOrDefault(actor.Artist);
        }
    }

    private static void WriteMovies(IXLWorksheet worksheet,
ICollection<Movie> movies)
    {
        WriteHeader(worksheet);

        int rowIndex = 2;
        foreach (var movie in movies)
        {
            WriteMovie(worksheet, movie, rowIndex);

            rowIndex += 1;
        }
    }

    public MovieExportService(CinemaContext context)
    {
        this.context = context;
    }

    public async Task WriteToAsync(Stream stream,
CancellationToken cancellationToken)
    {
        if (!stream.CanWrite)
        {
            throw new ArgumentException("Input stream is not
writable");
        }
```

```
            var movies = await context.Movies
                .Include(movie => movie.Actors)
                    .ThenInclude(actor => actor.Artist)
                .Include(movie => movie.Director)
                    .ThenInclude(director => director!.Artist)
                .ToListAsync(cancellationToken);

            using var workbook = new XLWorkbook();

            var worksheet =
        workbook.Worksheets.Add(RootWorksheetName);

            WriteMovies(worksheet, movies);

            workbook.SaveAs(stream);
        }
    }
```

Далі, необхідно додати кнопку для експорту даних кінострічок у файл `Views/Movies/Index.cshtml`:

```
    <a class="btn btn-primary mb-3" type="submit" asp-area="" asp-
controller="Movies" asp-action="Export">Експортувати у файл</a>
```

Після цього, додати відповідний метод у контролер `MoviesController`:

```
    [HttpGet]
    public async Task<IActionResult> Export([FromQuery] string
    contentType = "application/vnd.openxmlformats-
    officedocument.spreadsheetml.sheet", CancellationToken
    cancellationToken = default)
    {
        var exportService =
    movieDataPortServiceFactory.GetExportService(contentType);

        var memoryStream = new MemoryStream();

        await exportService.WriteToAsync(memoryStream,
    cancellationToken);

        await memoryStream.FlushAsync(cancellationToken);
        memoryStream.Position = 0;


        return new FileStreamResult(memoryStream, contentType)
        {
            FileDownloadName =
    $"movies_{DateTime.UtcNow.ToShortDateString()}.xlsx"
```

```
    };
}
```

Зверніть увагу, метод контролера є лише тимчасовим власником потоку на який посилається `memoryStream`, тому його необхідно залишити не знищеним після завершення методу для коректної обробки у `FileStreamResult`. Також необхідно звернути увагу на

```
await memoryStream.FlushAsync(cancellationToken);
memoryStream.Position = 0;
```

Цим забезпечується очищення буферу потоку і виставлення курсору на початок для подальшого читання у `FileStreamResult`.

КРОКИ ВИКОНАННЯ ЕТАПУ 1.7

Автентифікація та авторизація – це невід'ємні частини функціоналу інформаційної системи. Дуже рідко функціональність застосунку доступна анонімному користувачеві і всі користувачі мають однакові можливості у системі. Автентифікація дозволяє підтвердити, що система «знає» користувача, авторизація дозволяє підтвердити роль користувача у системі.

Існують різні підходи до реалізації автентифікації та авторизації. В залежності від складності та особливостей інформаційної системи, можуть бути використані різні послідовності процесу підтвердження особистості. Модулі автентифікації та авторизації можна імлементувати самостійно, або використати готові рішення, наприклад, для ASP.NET Core це буде «легковаговий» модуль Identity, або більш потужний IdentityServer. Також, середовища розробки містять готові шаблони для додавання автентифікації та авторизації, включно з необхідними елементами користувацького інтерфейсу. У даній демонстрації буде використано сімейство пакетів `Microsoft.AspNetCore.Identity`.

Додавання Identity в проєкт

ASP.NET Identity представляє вбудовану в ASP.NET систему автентифікації та авторизації. Дана система дозволяє створювати облікові записи користувачів системи, реалізує графічний інтерфейс входу в систему та підтвердження особистості, надає інтерфейс для керування обліковими записами, а також є інтегровною з широким спектром зовнішніх провайдерів, таких як Facebook, Google, Microsoft, Twitter та інших.

За принципом DDD є суперечність щодо використання користувачів у зв'язці з об'єктами доменної області. З одного боку, якщо користувач є сутністю вашої доменної області, вам необхідно буде передбачити окремий клас-посередник. Якщо ж користувачі будуть використовуватися незалежно від основної предметної області, то допустимо додавати класи користувачів та ролей лише у проєкті з вашим користувацьким інтерфейсом. У даній демонстрації, буде використано саме цей підхід.

Спершу, необхідно додати у ваш проєкт користувацького інтерфейсу пакет `Microsoft.AspNetCore.Identity.EntityFrameworkCore` останньої версії мажорної версії, що відповідає версії платформи (для проєкту `net6.0` це буде 6.x.x).

Створіть клас користувача, що наслідується від базового класу користувача `Microsoft.AspNetCore.Identity`. Таким чином, ви зможете додавати необхідні властивості користувача пізніше. Назвіть клас, наприклад `ApplicationUser`:

```
namespace CinemaMVC.WebMVC.Data.Identity;

public class ApplicationUser : IdentityUser
{
}
```

Далі необхідно створити клас контексту, що реалізує клас `IdentityDbContext<ApplicationUser>`.

```
namespace CinemaMVC.WebMVC.Data.Identity;

public class ApplicationIdentityContext :
IdentityDbContext<ApplicationUser>
{
    public
ApplicationIdentityContext(DbContextOptions<ApplicationIdentityC
ontext> options)
        : base(options)
    {
    }
}
```

Тут необхідно звернути увагу на те, де ви будете зберігати дані користувачів. Найпростіший варіант – створити окрему базу даних. Проте існує варіант виділити окрему схему у існуючій базі даних (міграції передбачають таку можливість). Для цього прикладу, буде використана нова база даних.

Додайте відповідний рядок підключення у файл конфігурації (як у кроці 1.1). Далі у класі `Program` підключіть новостворений контекст та додайте його як сховище даних для `Identity`:

```
builder.Services
    .AddDbContext<ApplicationIdentityContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString(n
ameof(ApplicationIdentityContext))
        , sqlOptions =>
sqlOptions.MigrationsAssembly(typeof(Program).GetTypeInfo().Asse
mbly.GetName().Name));

builder.Services.AddIdentity<ApplicationUser,
IdentityRole>(options => options.SignIn.RequireConfirmedAccount
= false)
    .AddEntityFrameworkStores<ApplicationIdentityContext>();
```

Створіть першу міграцію у NuGet консолі виконавши команду:

```
Add-Migration Identity_Initial -Context
CinemaMVC.WebMVC.Data.Identity.ApplicationIdentityContext -OutputDir
Data/Identity/Migrations
```

Після успішного створення міграції, виконайте її над вашою базою даних використовуючи команду :

`Update-Database -Context CinemaMVC.WebMVC.Data.Identity.ApplicationIdentityContext`.

Після виконання міграції, у вашій базі даних буде створено наступні таблиці (рис. 1.7.1).

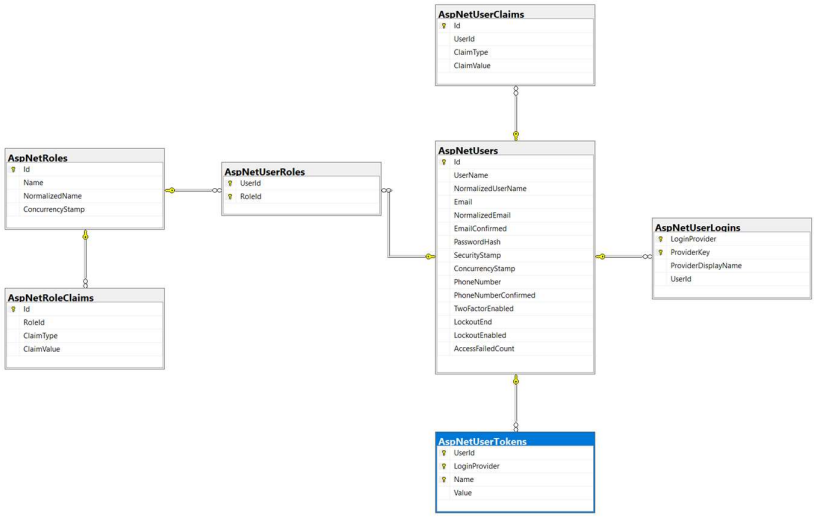


Рисунок 1.7.1 – Схема бази, створеної на основі `ApplicationIdentityContext`

База даних зараз не містить жодного користувача чи ролі, що ускладнить подальше тестування вашого рішення. Гарним підходом є створити клас, що буде додавати користувачів за замовчуванням при запуску вашого додатку у режимі розробки. Для цього додайте у файл конфігурації наступні секції:

```
"IdentityDefaults": {
    "SuperUser": {
      "Username": "admin@cinema",
      "Password": "P@ssw0rd!admin"
    },
    "DefaultUsers": [
      {
        "Username": "user1@cinema",
        "Password": "P@ssw0rd!user1"
      },
```

```
      {
        "Username": "user2@cinema",
        "Password": "P@ssw0rd!user2"
      }
    ]
  }
```

Увага! Намагайтеся не додавати жодних справжніх паролей у систему контролю версій, а також уникайте даного підходу (з файлом конфігурації) в умовах production-середовища.

Також необхідно не забувати про ролі. Оскільки назви ролей будуть використовуватися зокрема у коді вашого додатку, гарним тоном є створити клас з константами ролей, наприклад:

```
namespace CinemaMVC.WebMVC.Extensions;

public static class IdentityWebApplicationExtensions
{
    private record UserInfo(string Username, string Password)
    {
        public UserInfo()
            : this(string.Empty, string.Empty)
        {
        }
    }

    private static async Task
AddUserIfNotExistsAsync(UserManager<ApplicationUser> userManager
        , ILogger logger
        , string userName
        , string password
        , ICollection<string> roles)
    {
        var applicationUser = await
userManager.FindByEmailAsync(userName);

        if (applicationUser is null)
        {
            applicationUser = new ApplicationUser
            {
                UserName = userName,
                Email = userName
            };
            await userManager.CreateAsync(applicationUser,
password);
```

```
            logger.LogInformation("{username} user added",
userName);
        }
        else
        {
            logger.LogInformation("User {username} is already in
database", userName);
        }

        var existingRoles = await
userManager.GetRolesAsync(applicationUser);
        foreach (var role in roles.Where(role =>
!existingRoles.Contains(role)))
        {
            await userManager.AddToRoleAsync(applicationUser,
role);
            logger.LogInformation("{username} has {rolename}
assigned", userName, role);
        }
    }

    public static async Task InitializeRolesAsync(this
WebApplication app)
    {
        using var scope = app.Services.CreateScope();

        var serviceProvider = scope.ServiceProvider;

        var roleManager =
serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();

        foreach (var roleName in RoleNames.All)
        {
            if (!await roleManager.RoleExistsAsync(roleName))
            {
                var role = new IdentityRole { Name = roleName };
                await roleManager.CreateAsync(role);
            }
        }
    }

    public static async Task InitializeDefaultUsersAsync(this
WebApplication app
        , IConfiguration? superUserConfiguration
        , IConfiguration? defaultUsersConfiguration)
    {
        using var scope = app.Services.CreateScope();
```

```
        var serviceProvider = scope.ServiceProvider;

        var userManager =
serviceProvider.GetRequiredService<UserManager<ApplicationUser>>
();

        var superUserInfo =
superUserConfiguration.Get<UserInfo>();

        if (superUserInfo is not null)
        {
            await AddUserIfNotExistsAsync(userManager,
app.Logger, superUserInfo.Username, superUserInfo.Password,
RoleNames.All);
        }

        if (defaultUsersConfiguration is not null)
        {
            foreach (var defaultUserInfo in
defaultUsersConfiguration.Get<UserInfo[]>())
            {
                await AddUserIfNotExistsAsync(userManager,
app.Logger, defaultUserInfo.Username, defaultUserInfo.Password,
Array.Empty<string>());
            }
        }
    }
}
```

Додавання валідації залишається на самостійне опрацювання.
Далі необхідно викликати метод у класі `Program`:

```
if (app.Environment.IsDevelopment())
{
    await app.InitializeRolesAsync();
    await
app.InitializeDefaultUsersAsync(app.Configuration.GetSection("Id
entityDefaults:SuperUser"),
app.Configuration.GetSection("IdentityDefaults:DefaultUsers"));
}
```

Зверніть увагу, можливо є сенс винести ініціалізацію ролей за межі умови. Після запуску проєкту, користувачі і ролі будуть створені у базі даних (рис. 1.7.2).

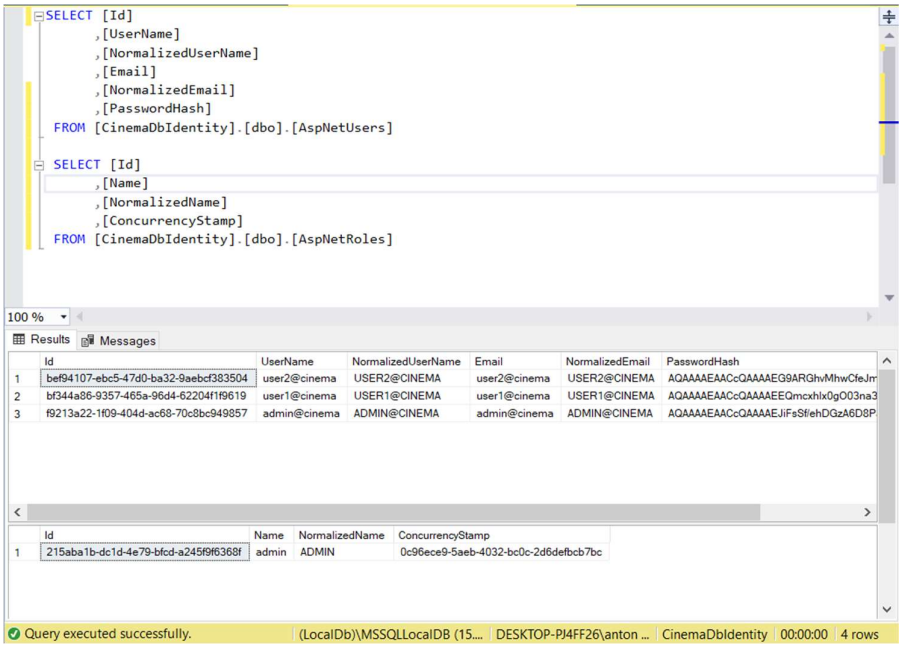


Рисунок 1.7.2 – Створені користувачі і ролі у базі даних

Розгортання користувацького інтерфейсу автентифікації та авторизації засобами середовища Visual Studio

Далі наведено приклад створення користувацького інтерфейсу з використанням засобів середовища Visual Studio. У діях над MVC проєктом необхідно обрати New Scaffolded Item, обрати створені класи користувача та контексту, та обрати сторінки, які мають бути створені в результаті операції (рис. 1.7.3 – 1.7.5).

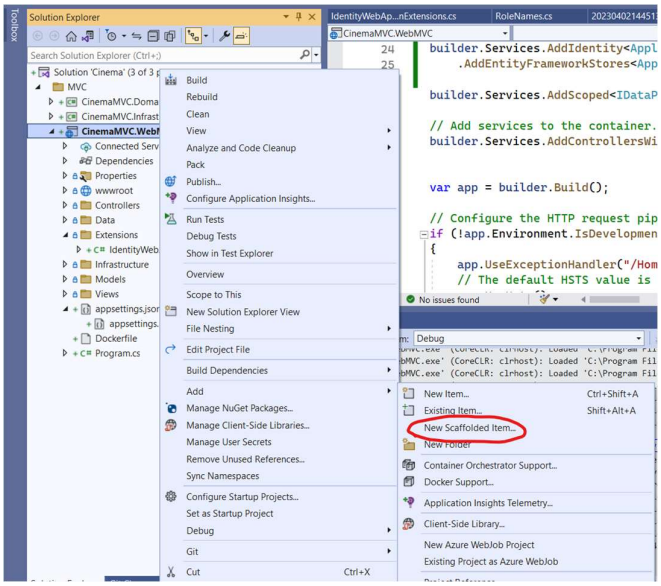


Рисунок 1.7.3 – Вибір функції розгортання шаблону

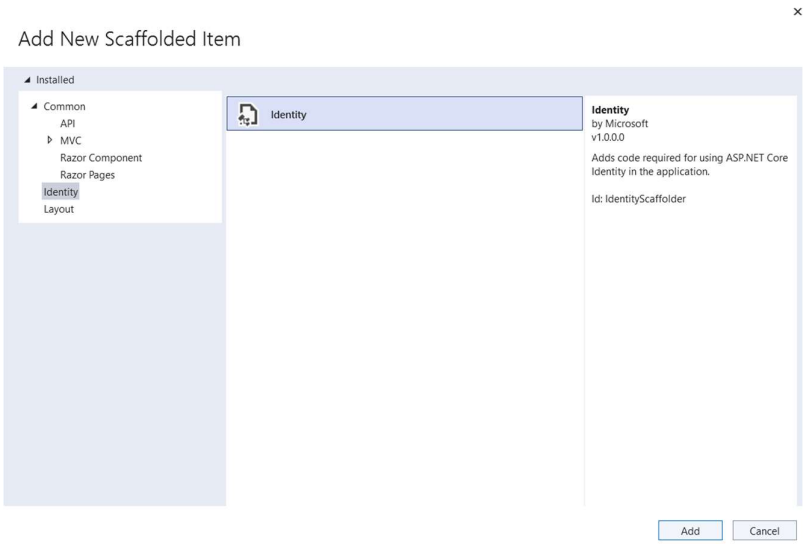


Рисунок 1.7.4 – Вибір шаблону Identity

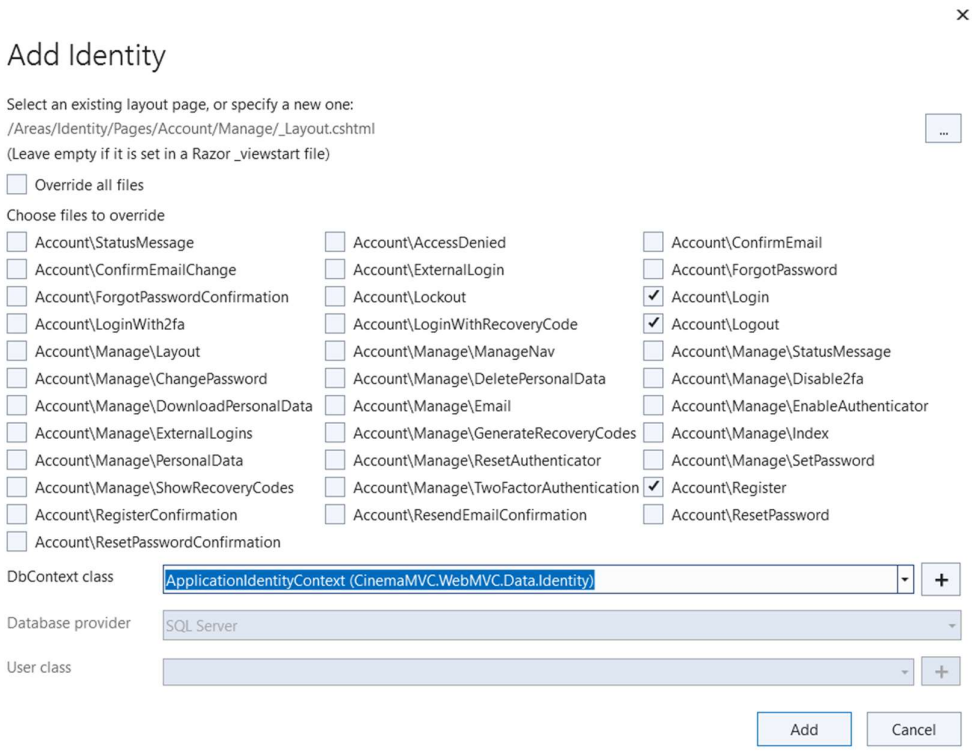


Рисунок 1.7.5 – Вибір необхідних сторінок і класу контекста

Після виконання розгортання інтерфейсу, будуть створені відповідні файли Razor-сторінок (рис. 1.7.6).

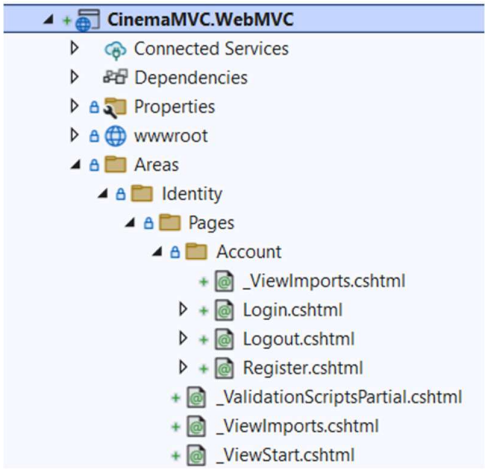


Рисунок 1.7.6 – Новостворені файли сторінок реєстрації та входу

Далі необхідно додати компонент логіну у шаблон сторінки вебзастосунку. Для цього необхідно у файл `/Views/Shared/_Layout.cshtml` додати у тег `div.navbar > div.container-fluid`:

```
<partial name="_LoginPartial" />
```

Після цього необхідно додати у клас `Program` сервіси для обробки RazorPages та додати авторизацію та автентифікацію у конвеєр обробки запитів:

```
// перед builder.Services.AddControllersWithViews();
builder.Services.AddRazorPages();
// після app.UseRouting() але до app.MapControllerRoute()
(порядок обов'язковий)
app.UseAuthentication();
app.UseAuthorization();
app.MapRazorPages();
```

Після запуску вебзастосунку, у заголовку сторінки зв’являться посилання на реєстрацію чи вхід у існуючий обліковий запис (рис. 1.7.7). Мову та зовнішній вигляд посилань можна змінити у файлі `/Views/Shared/_LoginPartial.cshtml`.

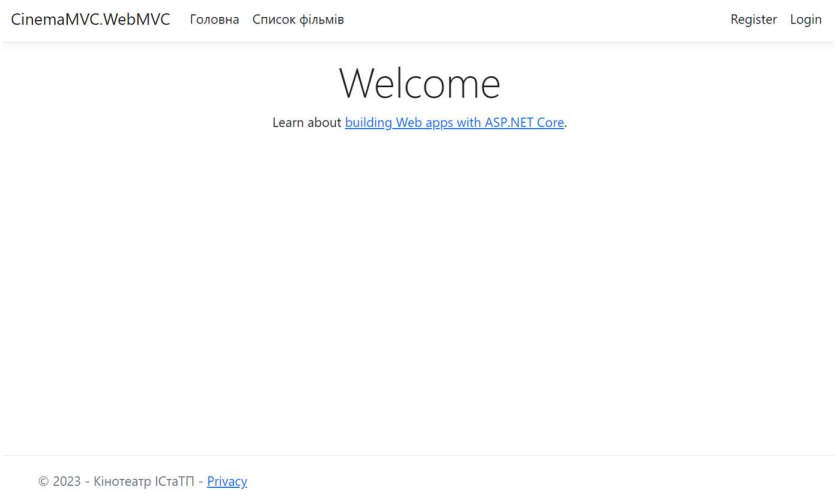


Рисунок 1.7.7 – Головна сторінка після додавання Identity

Зі сторінки логіну рекомендовано прибрати посилання на сторінки, які не були згенеровані (чи не потрібні в рамках вашої системи), а також змінити стилі та мову у відповідному файлі `.cshtml` (рис. 1.7.8).

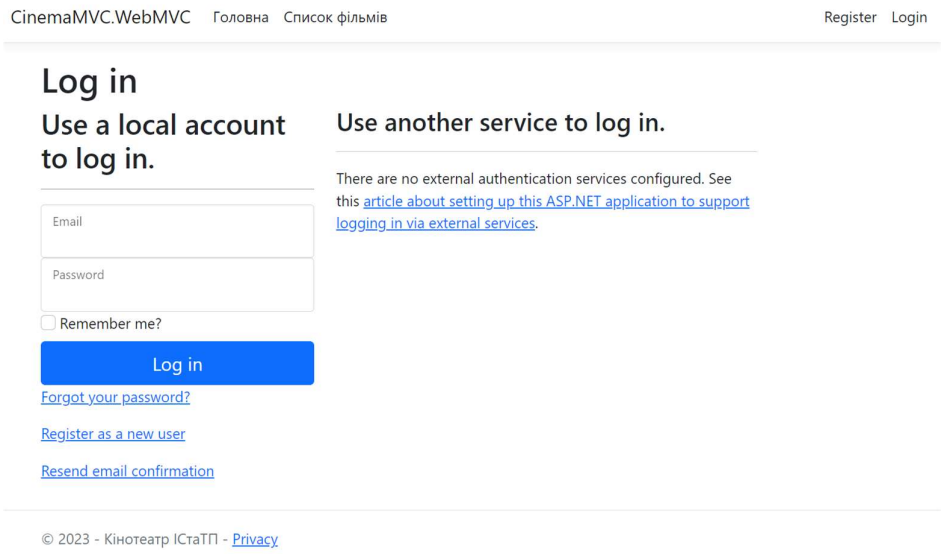


Рисунок 1.7.8 – Сторінка входу у обліковий запис

Після введення облікових даних з файлу конфігурації, користувачі будуть переадресовані на головну сторінку. Зверніть увагу, що в заголовку виводиться ім’я користувача (рис. 1.7.9).

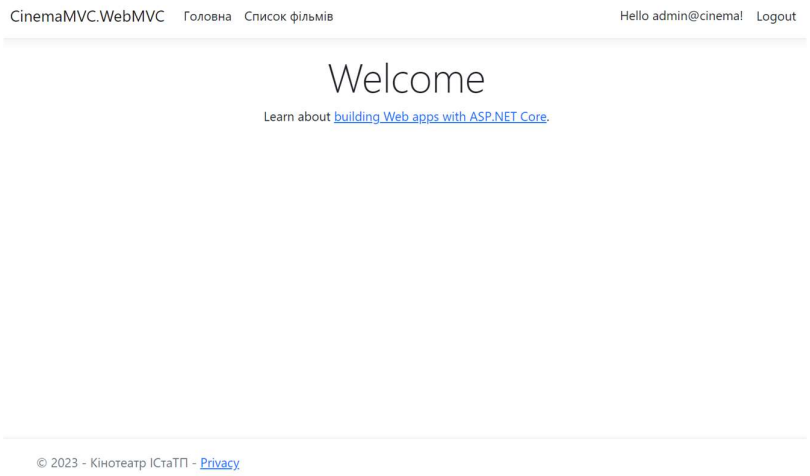


Рисунок 1.7.9 – Головна сторінка для авторизованого користувача

Приклад використання даних ролей у `.cshtml` файлах можна підглянути у `/Views/Shared/_LoginPartial.cshtml`:

```
@using Microsoft.AspNetCore.Identity
@using CinemaMVC.WebMVC.Data.Identity

@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager

<ul class="navbar-nav">
@if (SignInManager.IsSignedIn(User))
{
    <li class="nav-item">
        <a id="manage" class="nav-link text-dark" asp-
area="Identity" asp-page="/Account/Manage/Index"
title="Manage">Hello @UserManager.GetUserName(User)!</a>
    </li>
    <li class="nav-item">
        <form id="logoutForm" class="form-inline" asp-
area="Identity" asp-page="/Account/Logout" asp-route-
returnUrl="@Url.Action("Index", "Home", new { area = "" })">
            <button id="logout" type="submit" class="nav-link
btn btn-link text-dark">Logout</button>
        </form>
```

```
    </li>
}
else
{
    <li class="nav-item">
        <a class="nav-link text-dark" id="register" asp-
area="Identity" asp-page="/Account/Register">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" id="login" asp-
area="Identity" asp-page="/Account/Login">Login</a>
    </li>
}
</ul>
```

Зверніть увагу на анотацію `@inject` за допомогою неї можна додати сервіс у контекст рендерингу сторінки. Класи `UserManager` та `SignInManager` можна використовувати для отримання даних про користувача (посилання на об'єкт типу `ClaimsPrincipal` у змінній `User`).

Для прикладу використання авторизації, сховаємо можливість імпорту даних з файлу (див. етап 1.6) на рівні користувацького інтерфейсу та на рівні контролера.

Для приховування кнопки переходу на сторінку імпорту, необхідно додати Razor вираз-умову на сторінку `Views/Movies/Index.cshtml`:

```
@inject UserManager<ApplicationUser> UserManager
@inject SignInManager<ApplicationUser> SignInManager

@{
    var isAdmin = false;
    if (SignInManager.IsSignedIn(User))
    {
        var applicationUser = await
UserManager.GetUserAsync(User);

        if (applicationUser is not null)
        {
            isAdmin = await
UserManager.IsInRoleAsync(applicationUser, RoleNames.Admin);
        }
    }
}
@if (isAdmin)
{
```

```
    <a class="btn btn-primary mb-3" type="submit" asp-area=""
asp-controller="Movies" asp-action="Import">Завантажити з
файлу</a>
}
```

Для того, щоб не дозволяти користувачам без ролі адміністратора відкривати сторінку імпорту та завантажувати файл додайте атрибут `Authorize` зі списком ролей `Roles`, яким доступна ця операція до методів контролера `MoviesController`:

```
[HttpGet]
[Authorize(Roles = RoleNames.Admin)]
public IActionResult Import()
{
    return View();
}

[HttpPost]
[Authorize(Roles = RoleNames.Admin)]
public async Task<IActionResult> Import(IFormFile moviesFile,
CancellationToken cancellationToken)
{
    var importService =
movieDataPortServiceFactory.GetImportService(moviesFile.ContentT
ype);

    using var stream = moviesFile.OpenReadStream();

    await importService.ImportFromStreamAsync(stream,
cancellationToken);

    return RedirectToAction(nameof(Index));
}
```

Таким чином, рекомендовано обмежити доступ до операцій та видимість даних відповідно до розробленої Use-Case діаграми.

ЛАБОРАТОРНИЙ ПРОЄКТ № 2

Розробка вебдодатку

Набір технологій:

- *Entity Framework Core (EF Core) Database-First – ORM Framework*
- *Web API (ASP.NET Core 6.x)*
- *HTML, JavaScript*

За попередньою домовленістю з викладачем (не пізніше 14-го тижня від початку семестру) набір технологій може бути змінений.

В таблиці 2.1. наведено етапи, їх задачі та фінальний термін здачі конкретного етапу лабораторної роботи.

Таблиця 2.1 – Етапи лабораторної роботи 2

Номер етапу	Фінальний термін здачі етапу (18 тиждень семестру)	Задача етапу та матеріали до його виконання	Форма здачі етапу	Максимальна кількість балів за етап
2.0	Не пізніше 13-го тижня від початку семестру	Початковим етапом до виконання всіх лабораторних робіт є формулювання персонального завдання, розробка та затвердження викладачем діаграми прецедентів (Use-case діаграма) UML, яка демонструє особливості обраної предметної області.	Діаграми (фото, *.png, *.jpg чи у іншому форматі) надіслати на пошту викладачу принаймні за 24 години до співбесіди за цим етапом на лабораторному занятті. Перший етап має ОБОВ'ЯЗКОВО бути узгоджений з викладачем!!! З метою запобіганню використанню згенерованої в лабораторній роботі моделі, предметна область має відрізнятися від предметної області першої лабораторної роботи, або студент має обґрунтувати суттєві зміни в структурі моделі.	**2**
2.1	Не пізніше 15 тижня семестру	Створення додатку для Web API Створення моделі та класу контексту даних. (**ЛР_2_Етап_1_ІСтаТП.docx**).	Посилання на GitHub принаймні за 24 години до співбесіди за цим етапом на лабораторному занятті.	**3**

2.2	Не пізніше 16-го тижня від початку семестру	Реалізація контролерів та їх тестування (**ЛР_2_Етап_2_ІСтаТП.docx**).	Посилання на GitHub принаймні за 24 години до співбесіди за цим етапом на лабораторному занятті.	**2**
2.3	Не пізніше 18-го тижня від початку семестру	Виклик веб-API за допомогою JavaScript (**ЛР_2_Етап_3_ІСтаТП.docx**). **Можливість отримати 2 додаткові бали** за якісний Front end (за функціональністю та зовнішнім виглядом не має поступатися першій лабораторній роботі).	Посилання на GitHub принаймні за 24 години до співбесіди за цим етапом на лабораторному занятті. При прийнятті враховується виконання етапів, їх якість, терміни, складність обраної предметної області. Особиста співбесіда є ОБОВ'ЯЗКОВОЮ ! Без співбесіди бали не нараховуються. За бажанням, на додаткові бали Ви можете попрацювати самостійно і розробити повноцінний Front end. Див. наприклад (https://learn.microsoft.com/en-us/aspnet/core/client-side/spa/intro?view=aspnetcore-6.0)	**3** **+ 2 (додаткові бали)**

2.4	Не пізніше 18-го тижня від початку семестру	Контейнеризація рішення (на 1 додатковий бал). Створення Dockerfile для обох проєктів. Показати розгортання у оркестраторі (docker-compose, Kubernetes).	За бажанням можете самостійно опанувати цю тему.	**+ 1 (додатковий бал)**
2.5	Не пізніше 18-го тижня від початку семестру	Unit-тестування (на 1 додаткових бал) https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest	За бажанням можете самостійно опанувати цю тему.	**+ 1 (додатковий бал)**

КРОКИ ВИКОНАННЯ ЕТАПУ 2.1

Введення у Web API

Web API – це шаблон фреймворку ASP.NET Core, що дозволяє створювати мережеві застосунки з прикладним мережевим інтерфейсом. На відміну від шаблону MVC, в якому результатом виконання запиту була відмальована HTML-сторінка (здебільшого, бо також можливе повернення інших типів), шаблон Web API полегшує створення застосунків, з якими комунікує інше програмне забезпечення (інший сервер, браузер, мобільний застосунок, тощо).

В рамках даного лабораторного практикуму студентам запропоновано створити API для проєкту MVC з попереднього лабораторного практикуму, а також альтернативний клієнт для інформаційної системи, що буде використовувати створене Web API. Студент може обрати іншу тему, за умови збереження складності.

Найрозповсюдженішим архітектурним підходом до побудови Web API є REST (Representation State Transfer). REST-архітектура передбачає використання протоколу HTTP для доступу до ресурсів мережею. Ресурс описується за допомогою URI (з англ. – «Унікальний ідентифікатор ресурсу»), наприклад, https://cinema.app/api/tickets зображує унікальний ідентифікатор ресурсу квитків. URI можна розглядати як шлях у файловій системі вашої операційної системи. Для виконання операцій над ресурсами використовуються HTTP-запити на URI відповідного ресурсу з HTTP-дієсловом, рядком запиту та (опціонально) тілом запиту. Наприклад:

- GET-запит на https://cinema.app/api/tickets поверне всі квитки.
- GET-запит на https://cinema.app/api/tickets/1123 поверне один квиток з ідентифікатором 1123.
- POST-запит на https://cinema.app/api/tickets з даними для створення квитка у тілі запиту створить новий квиток
- DELETE-запит на https://cinema.app/api/tickets/1123 видалить квиток з ідентифікатором 1123.

Правильно спроєктований REST API дозволяє прозоро комунікувати з вебсервером використовуючи всі можливості протоколу HTTP. Таким чином до Web API можуть звертатися інші програми незалежно від технології чи мови програмування на якій написані агенти – це можуть бути інші вебзастосунки, мобільні або десктопні клієнти.

У сучасній веброзробці, підходи REST API на бекенді та Single Page Application на фронтенді (з використанням фреймворків Angular, React або Vue) є найрозповсюдженішим вибором для створення вебзастосунків.

Створення проєкту ASP.NET Core WebAPI

У даній демонстрації буде використано файл рішення, а також бази даних, створені в рамках лабораторного практикуму 1. При виборі іншої предметної

області, необхідно спроєктувати базу даних самостійно та створити відповідне Visual Studio рішення.

У проєкті `Cinema` створіть нову папку `API` проєкту і назвіть її API. Додайте проєкт шаблону ASP.NET Core Web API і назвіть його `CinemaWebAPI.WebAPI` (рис. 2.1.1 – 2.1.2).

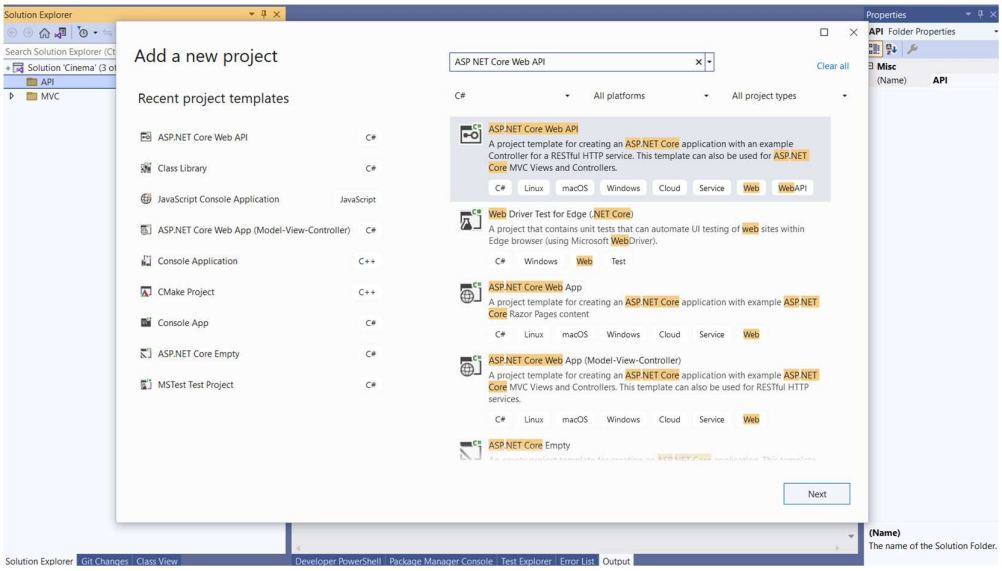


Рисунок 2.1.1 – Фрагмент вікна вибору шаблона

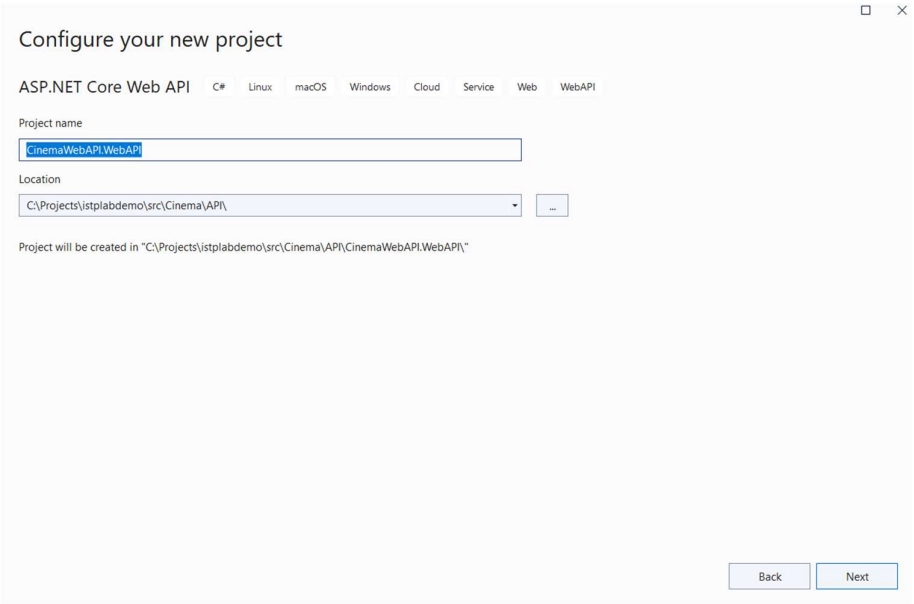


Рисунок 2.1.2 – Фрагмент вікна задання шляху до проєкту

Зверніть увагу, корінь простору імен був змінений для нового проєкту. За легендою лабораторного практикуму, Web API проєкт будується окремо від MVC, тому ці області рішення не мають мати жодних залежностей між собою.

Наступним етапом є створення класів моделей та контексту даних на основі бази даних (Database-First підхід EntityFrameworkCore). Проєкт Web API не буде власником бази даних, тому не буде створювати міграцій.

Для автоматичного створення необхідних класів, необхідно підключити пакети `Microsoft.EntityFrameworkCore.SqlServer` (або інший відповідно від СУБД, що використовується), `Microsoft.EntityFrameworkCore.Tools` та `Microsoft.EntityFrameworkCore.Design` у проєкт `CinemaWebAPI.WebAPI`. Важливо, версія має бути сумісною з версією мови .NET у проєкті `CinemaWebAPI.WebAPI`.

У консолі менеджера пакетів NuGet оберіть проєкт та введіть команду (замінивши рядок підключення та пакет СУБД, за потреби):

```
Scaffold-DbContext 'Data Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=CinemaDb;Integrated Security=true' Microsoft.EntityFrameworkCore.SqlServer -OutputDir Data
```

В результаті виконання даної команди, у папці `Data` будуть створені класи сутностей та контекст даних на основі наданої бази даних (рис. 2.1.3). Бажано

прибрати рядок підключення з методу `OnConfiguring` створеного контексту (прибрати метод взагалі).

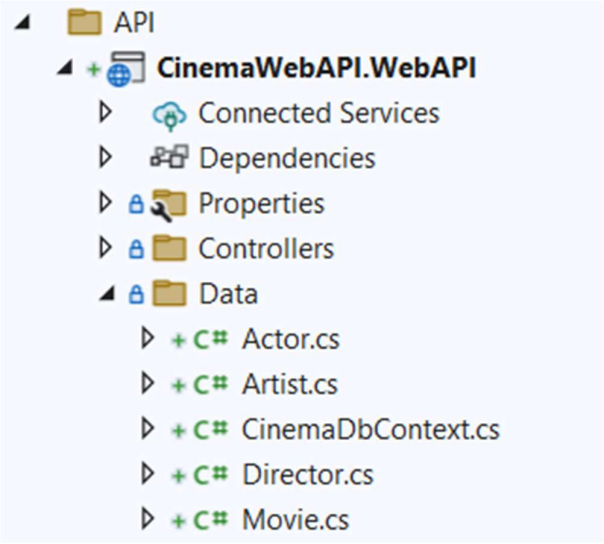


Рисунок 2.1.3 – Створені класи у папці `Data`

Далі необхідно додати рядок підключення у файл конфігурації та зареєструвати клас контексту у класі `Program`:

```
builder.Services
    .AddDbContext<CinemaDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString(n
ameof(CinemaDbContext))));
```

КРОКИ ВИКОНАННЯ ЕТАПУ 2.2

Створення контролерів

Контролери у ASP.NET Core WebAPI дуже схожі на контролери у ASP.NET Core MVC. Основною відмінністю контролерів у WebAPI є можливість винесення логіки підготовки відповіді за межі методів. Тобто, методи контролерів у WebAPI зазвичай (не обов'язково) повертають не JsonResult, а об'єкт певного класу, серіалізація якого у відповідь сервера відбувається за межами логіки класу контролера. Це дозволяє, наприклад, мати один і той же самий код контролерів для API, що може повертати JSON або XML документи в якості формату відповіді.

Створювати класи-контролери можна вручну, або скористатися шаблоном IDE. У Visual Studio, створіть новий контролер за шаблоном та оберіть клас моделі та контексту (рис. 2.2.1 – 2.2.3).

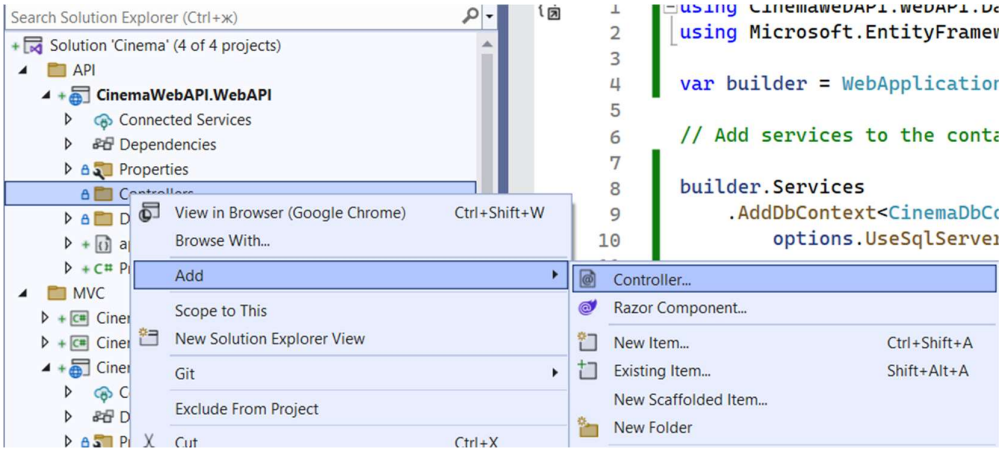


Рисунок 2.2.1 – Вибір варіанту створення контролера за шаблоном

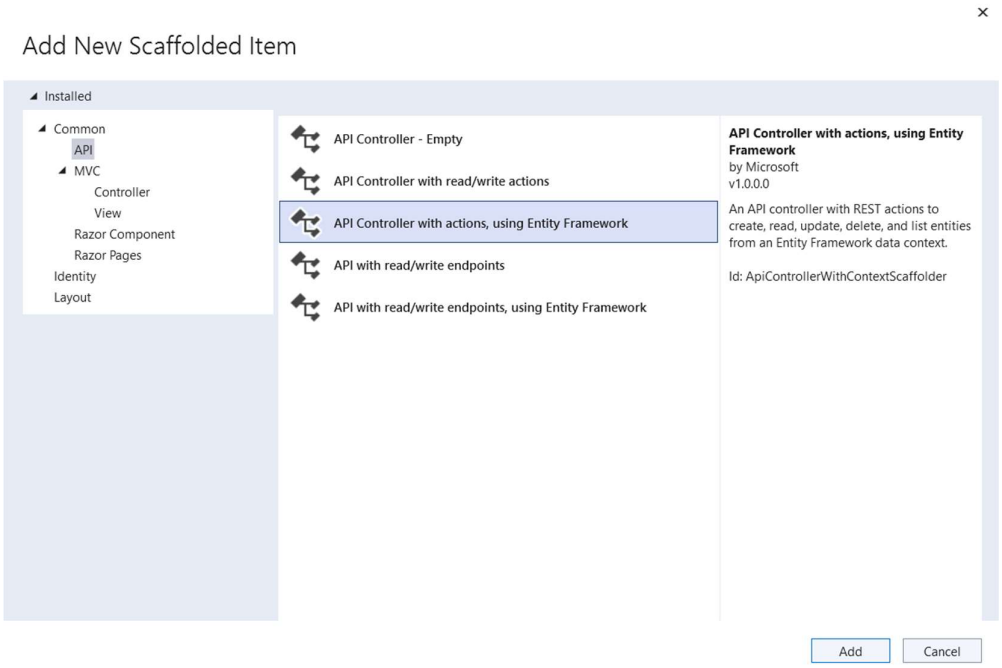


Рисунок 2.2.2 – Вибір шаблону створення контролеру з операціями читання та запису на основі контексту

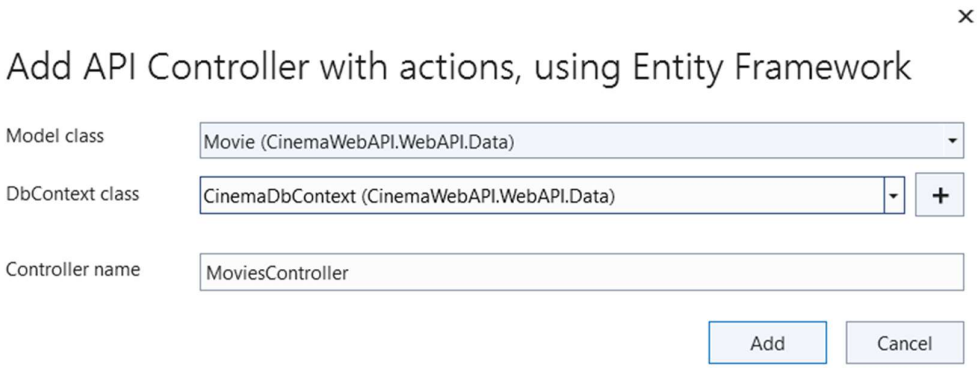


Рисунок 2.2.3 – Вибір класів моделі та контексту при створенні контролера

Розглянемо згенерований клас. До контролера застосовується атрибут `[ApiController]`, який дає зрозуміти фреймворку ASP.NET Core, що дані, що повертаються з методів контролеру мають бути оброблені перед відправкою. Також

до контролера застосовується атрибут `[Route("api/[controller]")]`, який вказує, що шлях до ресурсів контролера має починатися з `api/<назва-контролеру>` .

Конструктор контролера параметризується сервісами, що необхідні для його коректної роботи, в даному випадку, очікується клас.

Контролер API призначений для обробки HTTP-запитів. Методи контролера мають шлях та дієслово, що встановлюється атрибутами. Наприклад, метод позначений атрибутом `[HttpGet("{id}")]` оброблюватиме запити вигляду `api/<назва-контролеру>/<значення-id>`. В залежності від атрибутів, ядро фреймфорка підбирає метод, що підпадає під шаблон та дієслово, який задається атрибутами.

Аналогічно створюються контролери для усіх інших класів моделі (за необхідності методи можуть бути змінені).

Для тестування необхідно запустити проєкт. Якщо при створенні проєкту було додано OpenAPI, то при запуску проєкту буде згенерований OpenAPI документ та візуальний інтерфейс Swagger, що допоможуть тестувати API без зайвого програмного забезпечення. У генерації OpenAPI/Swagger означень є величезне значення – на основі їх легко автоматично згенерувати відповідні пакети/бібліотеки для взаємодії з API.

Після запуску WebAPI проєкту, автоматично відкриється сторінка SwaggerUI (рис. 2.2.4). Виконайте запит з використанням SwaggerUI, якщо ваш проєкт правильно сконфігуровано, то ви отримаєте відповідь у вигляді JSON-об’єкта (рис. 2.2.4).

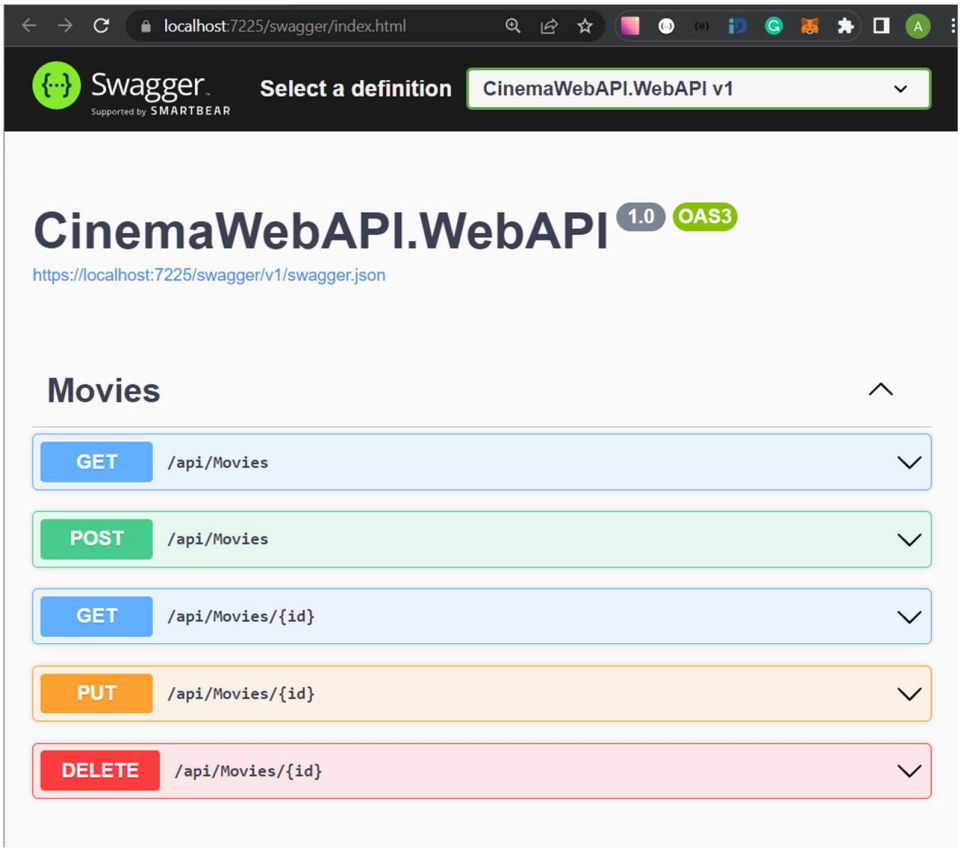


Рисунок 2.2.4 – Swagger UI на основі WebAPI

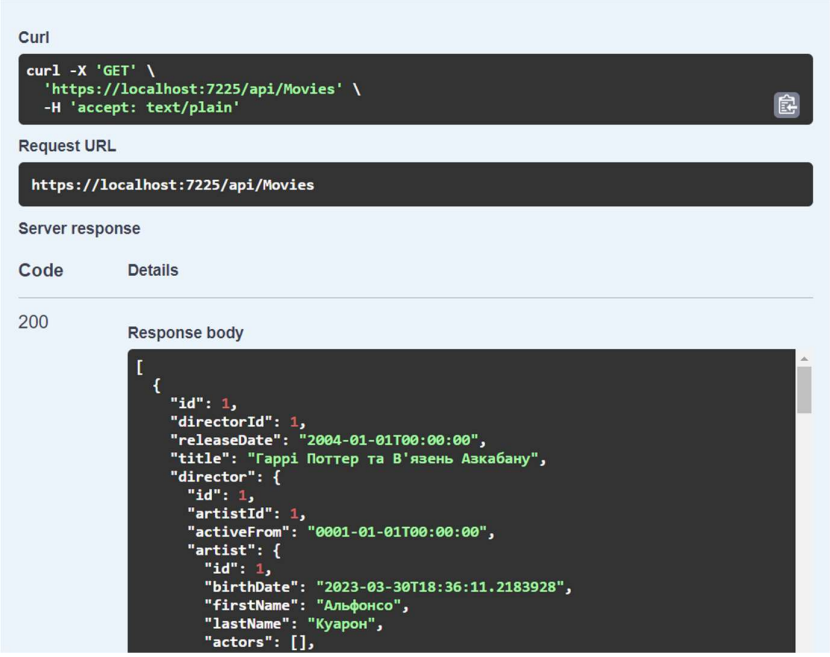


Рисунок 2.2.5 – Приклад виводу API у Swagger UI

Тестування контролерів

Для тестування контролера Web API можна застосовувати спеціальні інструменти, які встановлюються у вигляді окремих додатків, або у вигляді розширень для браузерів, наприклад, Fiddler або Postman.

Встановіть Postman (https://www.getpostman.com/downloads/) (Рис.2.2.10 – 2.2.11).

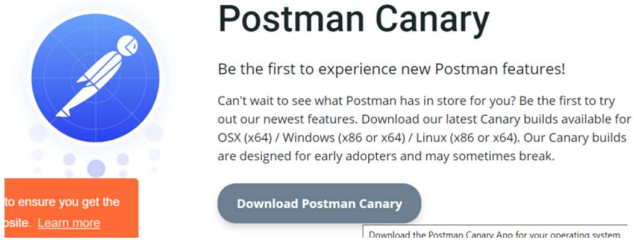


Рисунок 2.2.10 – Фрагмент вікна інсталяції програми

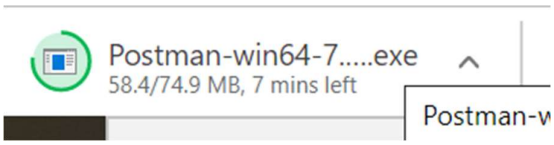


Рисунок 2.2.11 – Фрагмент вікна інсталяції програми

Перед тестуванням, запустіть вебзастосунок WebAPI. У Postman створіть новий запит (Рис. 2.2.12 – 2.2.15).

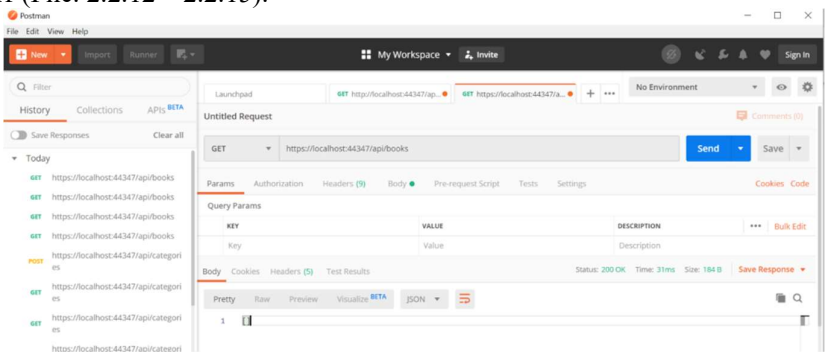


Рисунок 2.2.12 – Фрагмент вікна роботи з вебдодатком

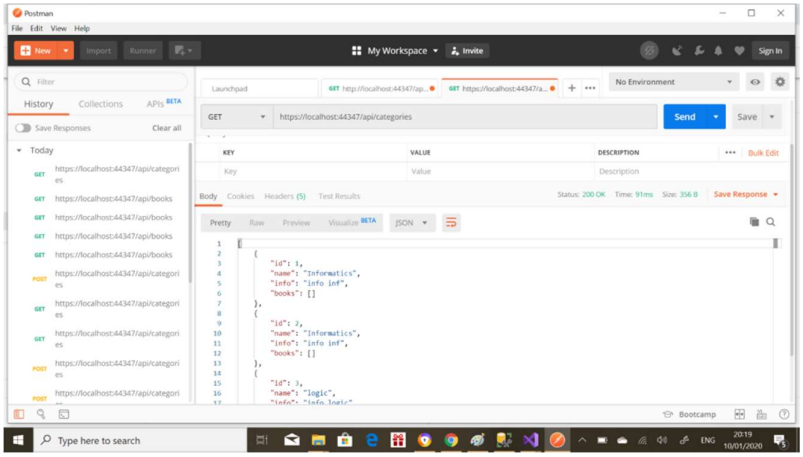


Рисунок 2.2.13 – Фрагмент вікна роботи з вебдодатком

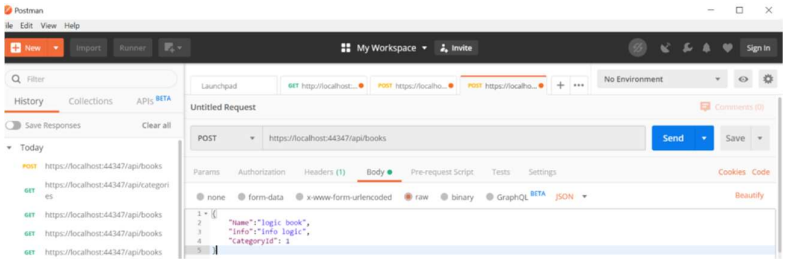


Рисунок 2.2.14 – Фрагмент вікна роботи з вебдодатком

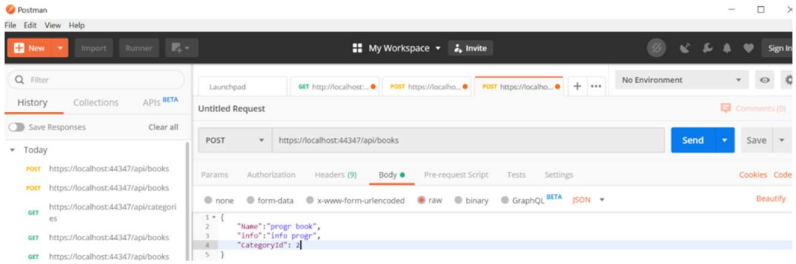


Рисунок 2.2.15 – Фрагмент вікна роботи з вебдодатком

Для здачі цього етапу підготуйте тестові приклади для усіх методів контролерів. Підказка, онову майбутньої Postman колекції можна згенерувати з використанням OpenAPI визначення.

КРОКИ ВИКОНАННЯ ЕТАПУ 2.3

Додамо HTML-сторінку, яка містить форми для створення і адміністрування категорій. Обробники подій приєднуються до елементів на сторінці. При використанні обробників подій створюються запити HTTP до методів дії веб-API. Функція Fetch API fetch ініціює кожен такий запит HTTP.

Функція fetch повертає об'єкт Promise, який містить відповідь HTTP, представлений у вигляді об'єкта Response. Поширеного підходу є отримання тексту відповіді JSON шляхом виклику функції json в об'єкті Response. JavaScript змінює сторінку, використовуючи відомості з відповіді API. Детальніша інформація про функцію fetch (https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API).

Налаштуйте в додатку обслуговування статичних файлів і включіть зіставлення файлів за замовчуванням. Вставте в клас `Prorgam` наступний код (рис. 2.3.1).

```
21  var app = builder.Build();
22
23  // Configure the HTTP request pipeline.
24  if (app.Environment.IsDevelopment())
25  {
26      app.UseSwagger();
27      app.UseSwaggerUI();
28  }
29
30  app.UseDefaultFiles();
31  app.UseStaticFiles();
32
33  app.UseHttpsRedirection();
34
35  app.UseAuthorization();
36
37  app.MapControllers();
38
39  app.Run();
40
```

Рисунок 2.3.1 – Фрагмент вікна роботи з класом `Prorgam`

Створіть папку `wwwroot` в кореневому каталозі проєкту.
Створіть папку `js` в папці `wwwroot` (рис. 2.3.2 – 2.3.3).

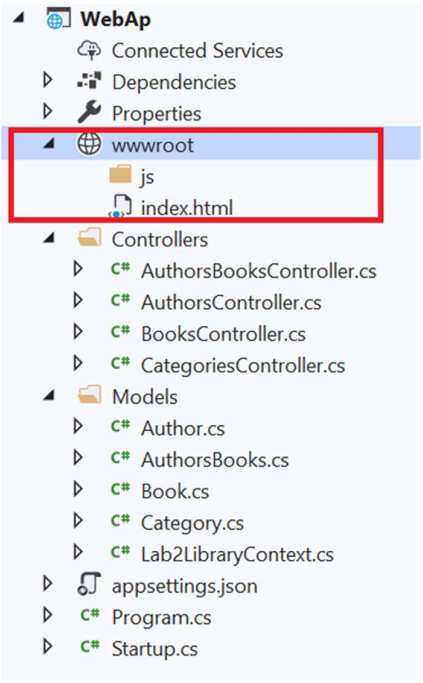


Рисунок 2.3.2 – Фрагмент вікна роботи з папкою **`wwwroot`**

Додайте HTML-файл **`index.html`** в папку **`wwwroot`**:

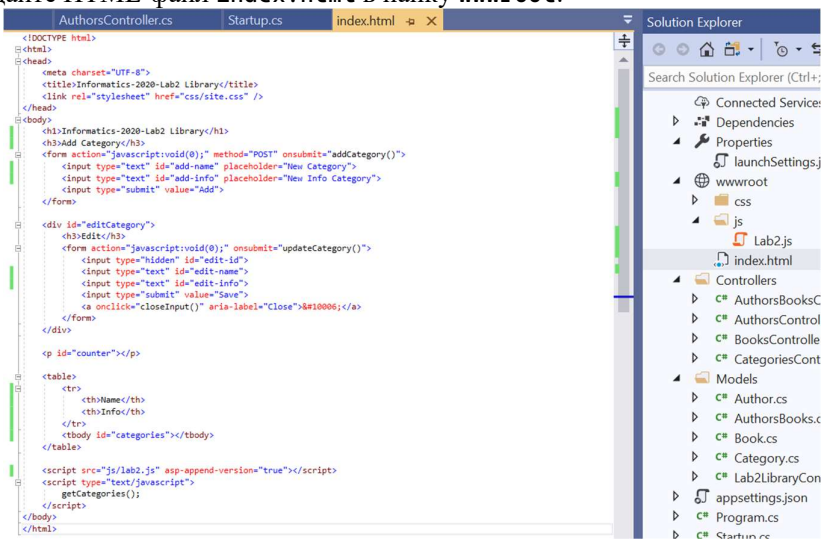


Рисунок 2.3.3 – Фрагмент вікна роботи з папкою **`wwwroot`**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Informatics-2020-Lab2 Library</title>
    <link rel="stylesheet" href="css/site.css" />
</head>
<body>
    <h1>Informatics-2020-Lab2 Library</h1>
    <h3>Add Category</h3>
    <form action="javascript:void(0);" method="POST"
onsubmit="addCategory()">
        <input type="text" id="add-name" placeholder="New
Category">
        <input type="text" id="add-info" placeholder="New Info
Category">
        <input type="submit" value="Add">
    </form>

    <div id="editCategory">
        <h3>Edit</h3>
        <form action="javascript:void(0);"
onsubmit="updateCategory()">
            <input type="hidden" id="edit-id">
            <input type="text" id="edit-name">
            <input type="text" id="edit-info">
            <input type="submit" value="Save">
            <a onclick="closeInput()" aria-
label="Close">&#10006;</a>
        </form>
    </div>
    <p id="counter"></p>
    <table>
        <tr>
            <th>Name</th>
            <th>Info</th>
        </tr>
        <tbody id="categories"></tbody>
    </table>

    <script src="js/lab2.js" asp-append-version="true"></script>
    <script type="text/javascript">
        getCategories();
    </script>
</body>
</html>
```

Додайте файл з іменем `lab2.js` в папку `wwwroot/js` (рис.2.3.4).

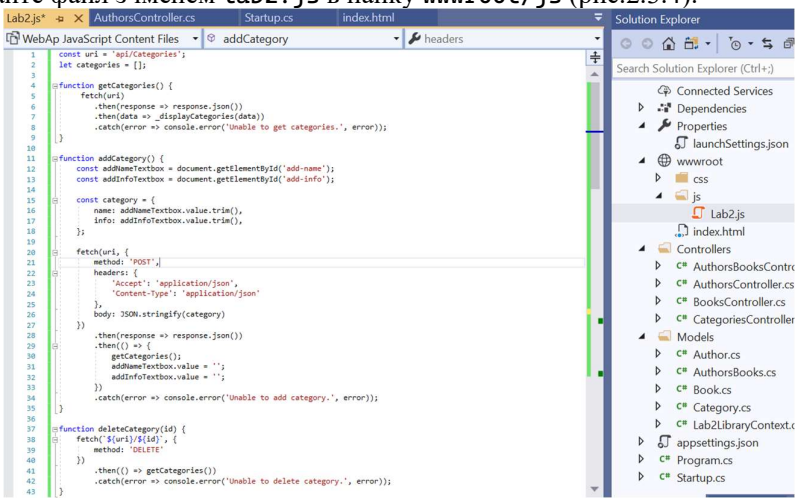


Рисунок 2.3.4 – Фрагмент вікна роботи з папкою `wwwroot/js`

```
const uri = 'api/Categories';
let categories = [];

function getCategories() {
    fetch(uri)
        .then(response => response.json())
        .then(data => _displayCategories(data))
        .catch(error => console.error('Unable to get
categories.', error));
}

function addCategory() {
    const addNameTextbox = document.getElementById('add-name');
    const addInfoTextbox = document.getElementById('add-info');

    const category = {
        name: addNameTextbox.value.trim(),
        info: addInfoTextbox.value.trim(),
    };

    fetch(uri, {
        method: 'POST',
        headers: {
            'Accept': 'application/json',
```

```
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(category)
    })
        .then(response => response.json())
        .then(() => {
            getCategories();
            addNameTextbox.value = '';
            addInfoTextbox.value = '';
        })
        .catch(error => console.error('Unable to add category.',
error));
}

function deleteCategory(id) {
    fetch(`${uri}/${id}`, {
        method: 'DELETE'
    })
        .then(() => getCategories())
        .catch(error => console.error('Unable to delete
category.', error));
}

function displayEditForm(id) {
    const category = categories.find(category => category.id ===
id);

    document.getElementById('edit-id').value = category.id;
    document.getElementById('edit-name').value = category.name;
    document.getElementById('edit-info').value = category.info;
    document.getElementById('editForm').style.display = 'block';
}

function updateCategory() {
    const categoryId = document.getElementById('edit-id').value;
    const category = {
        id: parseInt(categoryId, 10),
        name: document.getElementById('edit-name').value.trim(),
        info: document.getElementById('edit-info').value.trim()
    };

    fetch(`${uri}/${categoryId}`, {
        method: 'PUT',
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(category)
```

```
    })
        .then(() => getCategories())
        .catch(error => console.error('Unable to update
category.', error));

    closeInput();

    return false;
}

function closeInput() {
    document.getElementById('editForm').style.display = 'none';
}


function _displayCategories(data) {
    const tBody = document.getElementById('categories');
    tBody.innerHTML = '';


    const button = document.createElement('button');

    data.forEach(category => {
        let editButton = button.cloneNode(false);
        editButton.innerText = 'Edit';
        editButton.setAttribute('onclick',
`displayEditForm(${category.id})`);

        let deleteButton = button.cloneNode(false);
        deleteButton.innerText = 'Delete';
        deleteButton.setAttribute('onclick',
`deleteCategory(${category.id})`);

        let tr = tBody.insertRow();


        let td1 = tr.insertCell(0);
        let textNode = document.createTextNode(category.name);
        td1.appendChild(textNode);

        let td2 = tr.insertCell(1);
        let textNodeInfo =
document.createTextNode(category.info);
        td2.appendChild(textNodeInfo);

        let td3 = tr.insertCell(2);
        td3.appendChild(editButton);
```

```
            let td4 = tr.insertCell(3);
            td4.appendChild(deleteButton);
        });

        categories = data;
    }
```

За бажання можна додати в папку CSS файл `site.css` (проте, без цього етап може бути зараховано).

Після запуску, WebAPI віддасть файл вашої початкової сторінки (рис. 2.3.5).

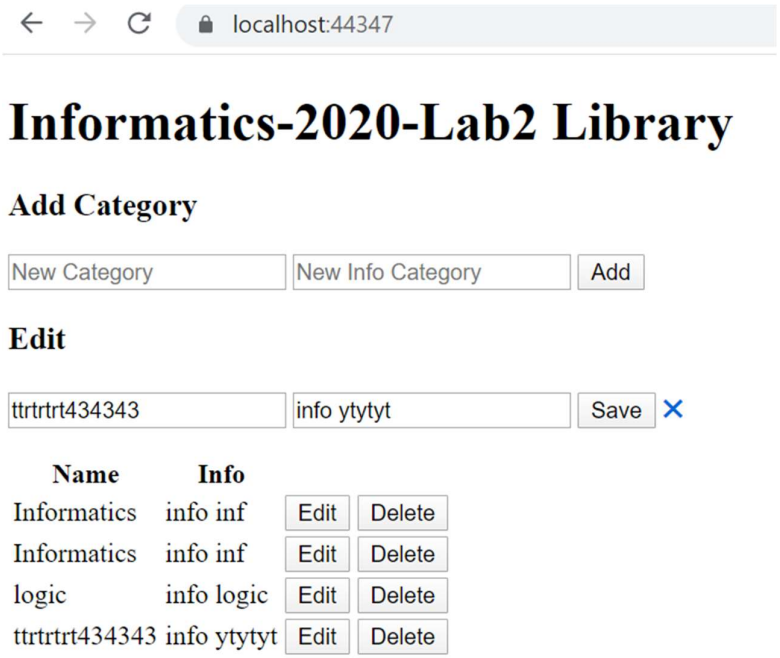


Рисунок 2.3.5 – Фрагмент вікна демонстрації етапу

Для здачі етапу достатньо створити HTML-сторінку, яка демонструє роботу одного контролера.

КРОКИ ВИКОНАННЯ ЕТАПУ 2.4

Поняття контейнеризації

Поняття контейнеризації тісно пов’язано з поняттям віртуалізації.

Віртуалізація – це технологія, що створює штучне середовище, що заміняє фізичне обладнання і дозволяє завантажувати образи операційних систем як окремі програми. В залежності від способу реалізації віртуалізація поділяється на емуляцію, повну віртуалізацію, пара-віртуалізацію та віртуалізацію рівня операційної системи.

Віртуальна машина – це копія обчислювальної машини разом з диском, операційною системою, пристроями вводу-виводу, тощо. Віртуальна машина запускається на базі певного моніторингового програмного забезпечення, що створює віртуальний контекст, повністю подібний до обладнання певного обчислювального пристрою. Образ віртуальної машини модифікований на одній фізичній машині може бути вивантажений у файл і переданий для розгортання на іншій фізичній машині. На одній фізичній машині може бути розгорнуто безліч віртуальних машин. Лімітом є лише обчислювана здатність фізичної машини. На рисунку зображено запуск двох віртуальних машин на одному фізичному комп’ютері.

Рисунок 2.4.1 – Архітектура технології віртуалізації

Контейнери – це засоби ізоляції (інкапсуляції) певного програмного забезпечення разом з його залежностями. Контейнери схожі на віртуальні машини тим, що також містять ізольований екземпляр певної операційної системи. Проте, контейнери використовують ресурси операційної системи фізичної машини на якій завантажуються. Це дозволяє більш ефективно використовувати ресурси, адже відсутній рівень емуляції обладнання. Контейнери дозволяють ізолювати інфраструктуру з мінімальною вартістю.

Docker – програмне забезпечення для створення та запуску контейнерів. Образ – збережений контейнер, модифікований на основі іншого образу. Базовим

образом є чиста версія операційної системи. Рівень – певне програмне забезпечення, наприклад, платформа .NET. Образи можуть бути завантажені з відкритого репозиторію образів.

Docker Compose – утиліта для створення і запуску застосунків, що складаються з декількох взаємопов'язаних контейнерів, так званий оркестратор. Наприклад, в одному контейнері може бути запущена СУБД, а у іншому додаток, що його використовує.

За допомогою Docker можна буде створити образ зі встановленим застосунком, а за допомогою Docker Compose створити конфігураційний файл з налаштуваннями необхідної архітектури.

Використання контейнерів дозволяє абстагувати розробку від деталей апаратного забезпечення та операційної системи на якій буде розгортатися застосунок. Постачальники хмарних послуг активно спонукають постачальників програмного забезпечення використовувати контейнери для розгортання систем у хмарних сервісах.

Встановлення Docker

Найпростіший спосіб встановити Docker на персональний комп'ютер – встановити Docker Desktop (https://docs.docker.com/desktop/). Це комплексний пакет різних утиліт для розробки контейнерів з використанням Docker: Docker Engine, Docker CLI client, Docker Compose, Docker Content Trust, Kubernetes та Credential Helper. На момент написання даного матеріалу Docker Desktop доступний на Windows, MacOS, та для декількох найпопулярніших дистрибутивів Linux (Ubuntu, Debian, Fedora). Для інших версій/дистрибутивів Linux також доступні всі вище згадані компоненти, але інсталювати їх треба буде по окремо.

Після встановлення, запустіть службу Docker Desktop та відкрийте вікно керування (рис 2.4.2).

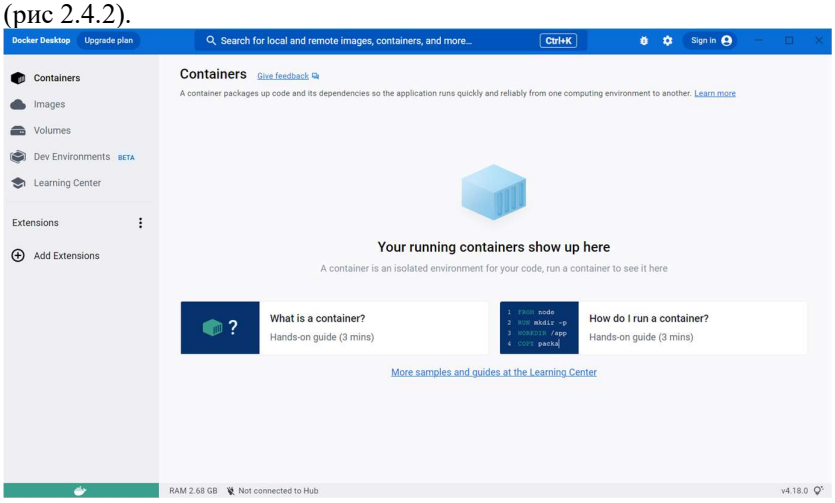


Рисунок 2.4.2 – Вікно керування Docker Desktop

Для перевірки успішності інсталяції Docker відкрийте командний рядок та введіть:

```
docker run -dp 80:80 docker/getting-started
```

Розглянемо виклик даної команди та її аргументи:

- `docker` – це CLI-застосунок для керування Docker-службою на вашій машині.
- Команда `run` – запускає контейнер у середовищі `docker` (див. https://docs.docker.com/engine/reference/run/)
- `d` та `p` – це параметри команди. `d` – означає detached mode, тобто контейнер буде запущений на фоні і керування терміналом повернеться до користувача. `p` – це відображення внутрішніх портів контейнера на реальні мережеві порти
- `docker/getting-started` – назва образу, що буде завантажений з локального кешу комп'ютера чи репозиторію (за замовчуванням, відкритий репозиторій Docker Hub).

Після виклику команди, відкрийте список контейнерів у інтерфейсі Docker Desktop (рис. 2.4.3).

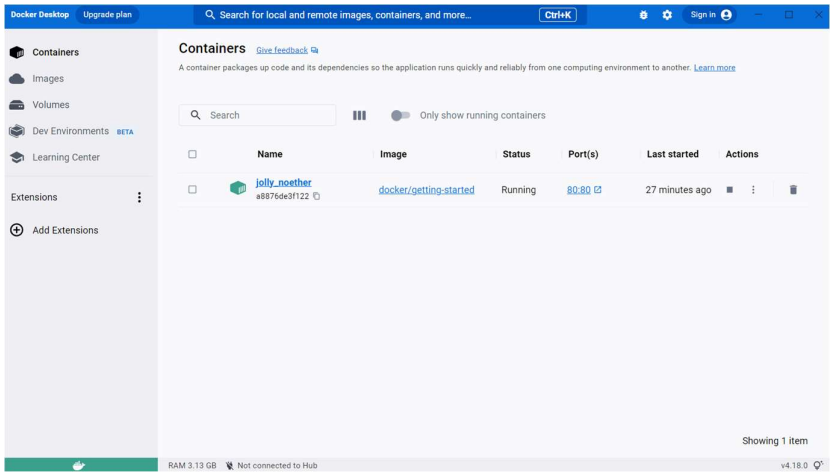


Рисунок 2.4.3 – Запущений контейнер у середовищі Docker Desktop

Контейнеризація додатку з використанням Visual Studio

У меню додавання елементу до проєкту ASP.NET Core, оберіть пункт «Підтримка Docker» (рис. 2.4.4).

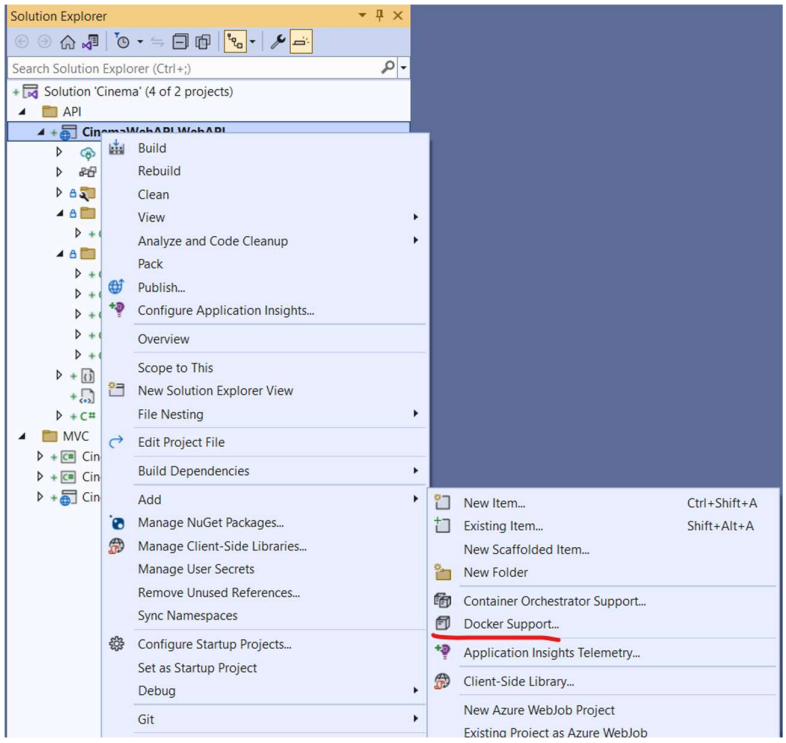


Рисунок 2.4.4 – Додавання підтримки Docker засобами Visual Studio

У папці проєкту буде створено файл `Dockerfile` наступного змісту:

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["API/CinemaWebAPI.WebAPI/CinemaWebAPI.WebAPI.csproj",
"API/CinemaWebAPI.WebAPI/"]
RUN dotnet restore
"API/CinemaWebAPI.WebAPI/CinemaWebAPI.WebAPI.csproj"
COPY . .
WORKDIR "/src/API/CinemaWebAPI.WebAPI"
RUN dotnet build "CinemaWebAPI.WebAPI.csproj" -c Release -o
/app/build

FROM build AS publish
```

```
RUN dotnet publish "CinemaWebAPI.WebAPI.csproj" -c Release -o
/app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "CinemaWebAPI.WebAPI.dll"]
```

Даний файл конфігурації містить інструкції зі збірки та запуску застосунку. Docker-образи будуються за принципом шарів. Нульовим шаром є операційна система. Якщо до шару застосовується якась операція зміни (команда), образ вважається новим самостійним образом. Таким чином, DockerFile складається інструкцій з означення образу:

- Означуємо базове зображення з рантаймом ASP.NET Core 6. Означуємо робочий каталог і відображаємо порти 80 та 433 (http та https відповідно)
- Означуємо образ для збірки проекту на основі .NET 6 SDK. Копіюємо файли проекту та встановлюємо залежності. Після цього виконується збірка проекту.
- Далі копіюємо файли до базового образу та запускаємо проект

Для запуску достатньо запустити debug-конфігурацію Docker у Visual Studio (рис. 2.4.5).

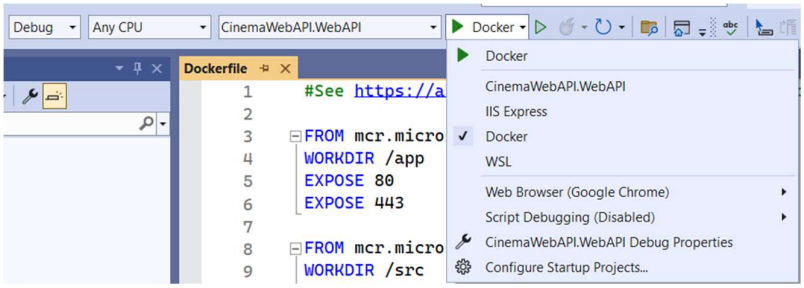


Рисунок 2.4.5 – Вибір конфігурації для відлагодження у Docker контейнері

Додавання підтримки середовища оркестрації Docker Compose

Середовища оркестрації контейнерів – це програмне забезпечення, що застосовується для керування та запуску множини контейнерів. Docker Compose вбудоване у стандартний пакет Docker Desktop середовище оркестрації. Дане середовище ідеально підходить для локального відлагодження систем, що складаються з більше ніж одного компонента.

Для додавання підтримки середовища оркестрації у Visual Studio необхідно створити проєкт docker-compose. Найпростіший спосіб створити проєкт даного типу – це обрати проєкт з `Dockerfile` та в меню додавання елементу проєкту обрати «Додати підтримку середовища оркестрації контейнерів» (рис. 2.4.6-2.4.7).

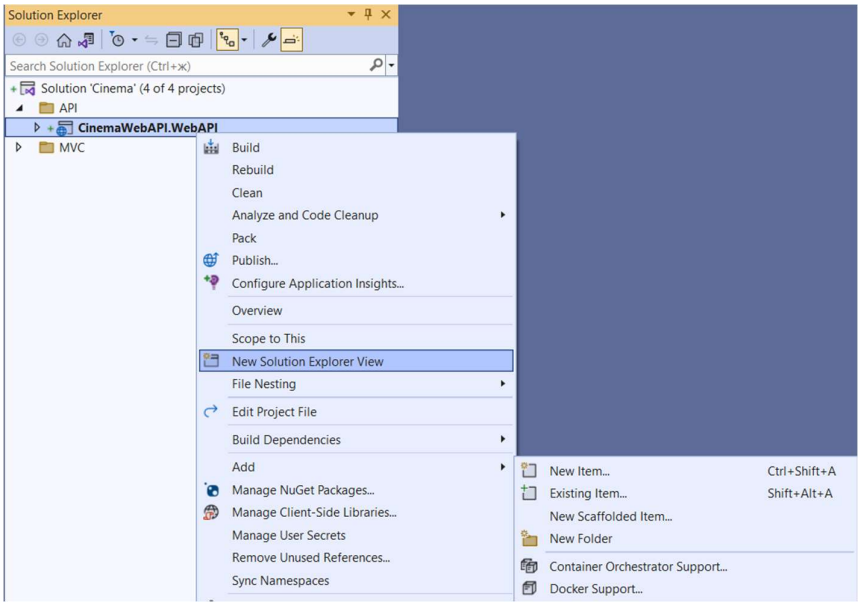


Рисунок 2.4.6 – Додавання підтримки середовища оркестрації контейнерів засобами Visual Studio

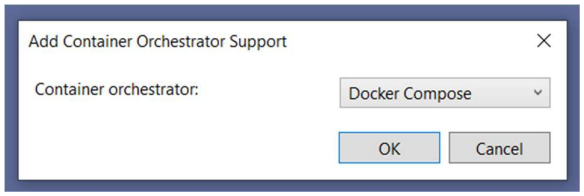


Рисунок 2.4.7 – Меню вибору середовища оркестрації

Після додавання підтримки середовища оркестрації контейнерів, буде створено `.dcproj` проєкт в якому вже доданий проєкт, на основі якого запускалася дія (рис. 2.4.8).

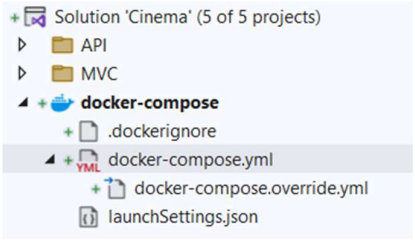


Рисунок 2.4.8 – Створений `.dcproj` проєкт

Разом з проєктом, створюються два файли `docker-compose.yml` та `docker-compose.override.yml`. Перший файл є основним, який читає Docker Compose, другий використовується для додаткової конфігурації (наприклад, окремий файл для кожного середовища). Додайте способом наведеним вище всі проєкти ASP.NET Core вашого рішення у файл `docker-compose.yml`.

Далі необхідно додати контейнер, що буде відповідати вашій базі даних. Для цього необхідно знайти відповідний образ у репозиторії Docker Hub та додати образ у `docker-compose.yml`. В результаті необхідно отримати `docker-compose.yml` наступного змісту:

```
version: '3.4'

services:
  cinemawebapi.webapi:
    image: ${DOCKER_REGISTRY-}cinemawebapiwebapi
    depends_on:
    - sqlserverService
    build:
      context: .
      dockerfile: API/CinemaWebAPI.WebAPI/Dockerfile

  cinemamvc.webmvc:
    image: ${DOCKER_REGISTRY-}cinemamvcwebmvc
    depends_on:
    - sqlserverService
    build:
      context: .
      dockerfile: MVC/CinemaMVC.WebMVC/Dockerfile

  sqlserverService:
    image: mcr.microsoft.com/mssql/server:latest
```

Для конфігурації з'єднання застосунків з доданим екземпляром SQL Server (або іншої бази даних) необхідно модифікувати файл `docker-compose.override.yml` наступним чином:

```
version: '3.4'

services:
  cinemawebapi.webapi:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
      -
ConnectionStrings__CinemaDbContext=Server=sqlserverService;Initi
al Catalog=cinemadb;User ID=sa;Password=BigPassw0rd1;
    ports:
```

```
      - "7661:80"
      - "7761:433"
    volumes:
      -
${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
  cinemamvc.webmvc:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
      -
ConnectionStrings__CinemaContext=Server=sqlserverService;Initial
Catalog=cinemadb;User ID=sa;Password=BigPassw0rd1;
      -
ConnectionStrings__ApplicationIdentityContext=Server=sqlserverSe
rvice;Initial Catalog=cinemadbidentity;User
ID=sa;Password=BigPassw0rd1;
    ports:
      - "7660:80"
      - "7760:433"
    volumes:
      -
${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
  sqlserverService:
    hostname: sqlserverService
    environment:
      ACCEPT_EULA: Y
      SA_PASSWORD: BigPassw0rd1
    volumes:
      - ./data/mssql:/var/opt/mssql3
    ports:
      - "5433:1433"
```

Зверніть увагу на наступні деталі:

- Рядки підключення до бази даних вказуються через змінні оточення (секція `environment`). Пароль має сходитися з наведеним у змінних контейнера, що відповідає базі даних (`sa` – адмін за замовчуванням у SQL Server).
- У даному прикладі публікуються лише HTTP порти. Для публікації HTTPS портів необхідно дати довіреність на SSL-сертифікати для ваших веб сервісів https://learn.microsoft.com/en-us/aspnet/core/security/docker-compose-https?view=aspnetcore-6.0. Для поточного варіанту розгортання необхідно відключити HTTPS переадресацію (прибрати у всіх файлах `Program.cs` виклик `app.UseHttpsRedirection()`)

- Для сховища необхідно означити том (секція `volumes`). Це необхідно для збереження даних СУБД у «реальну» файлову систему з метою не втратити дані після видалення контейнера чи віртуальної машини.

Для запуску з означення Docker Compose, необхідно обрати відповідну конфігурацію у Visual Studio та запустити рішення. В результаті, обидва додатки доступні за визначеними портами (рис. 2.4.9) та ділять базу даних. Також, з'являється можливість моніторити контейнери у інтерфейсі Docker Desktop (рис. 2.4.10).

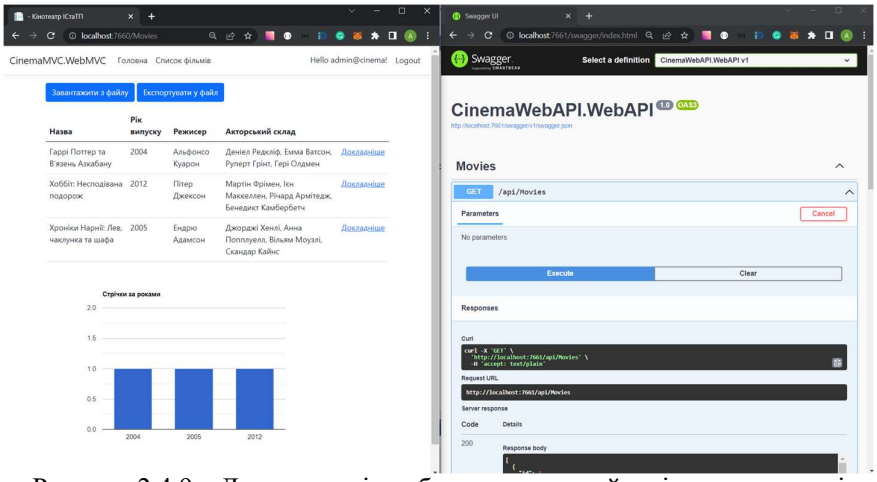


Рисунок 2.4.9 – Демонстрація роботи двох контейнерів у середовищі оркестрації

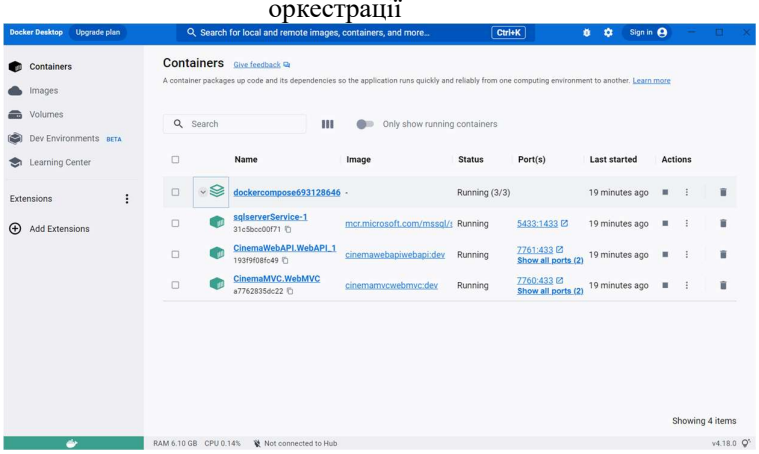


Рисунок 2.4.10 – Моніторинг контейнерів у інтефейсі Docker Desktop

ВИМОГИ І РЕКОМЕНДАЦІЇ З НАПИСАННЯ КОДУ

Написати програму – це більше, ніж правильно її оформити та змусити коректно виконуватись. Програми згодом неминуче модифікуються, повторно використовуються для побудови інших програм тощо. Тому велике значення має простота та зрозумілість їхньої внутрішньої структури. Інколи добре написану чужу програму набагато простіше зрозуміти, ніж власну, але написану погано. Культура написання коду сприяє зменшенню помилок у програмах і полегшує їхню модифікацію та повторне використання.

Під **стилем** програмування зазвичай розуміють набір прийомів і методів, що застосовуються з метою одержання правильних і ефективних програм, зручних для сприйняття, повторного застосування й модифікації.

ВИБІР ІМЕНІ

Імена мають проєкти, інтерфейси, класи, поля, тощо. Коли іменувань у програмі з десяток чи два, а над кодовою базою працює одна людина, то вибір не є дуже принциповим питанням. Однак у реальних проєктах, кількість різних проєктів, просторів імен, класів у коді системи сягає десятків, сотень, а то й тисяч, то, щоб код залишався зрозумілим для десятків чи сотень програмістів, які з ним працюють, при виборі імен необхідно дотримуватись визначених правил, загальних або у межах кола розробників програмного продукту.

Всі імена у кодовій базі мають явно вказувати на смисл понять, що ними подаються, тобто мають мати відповідну **мнемоніку**. Додатково, у коді можуть вказуватися метакоментарі до відповідних класів та методів.

```
/// <summary>
/// Encapsulates all HTTP-specific information about an
individual HTTP request.
/// </summary>
public abstract class HttpContext
{
    /// <summary>
    /// Gets the <see cref="HttpRequest"/> object for this
request.
    /// </summary>
    public abstract HttpRequest Request { get; }

    /// <summary>
    /// Gets the <see cref="HttpResponse"/> object for this
request.
```

```
        /// </summary>
        public abstract HttpResponse Response { get; }
    }
```

Для локальних змінних, допустимі короткі імена, проте лише у випадках, коли назва є загально прийнятою. Наприклад, для лічильників циклу зазвичай використовуються імена `i, j`, для рядків – `s, t`. Як правило, використовують англомовні назви змінних, назв класів, інтерфейсів, методів, властивостей, полів тощо, які відповідають їхній ролі в програмі.

Існує багато корпоративних домовленостей і традицій іменування, які фіксуються у спеціальних внутрішніх стандартах виробників. Наприклад, часто для методів використовуються імена, побудовані з дієслів та іменників, а для полів, класів, властивостей – іменники.

ФОРМАТУВАННЯ ТЕКСТУ

Оператори й вирази є основними конструкціями програм, тому їх слід оформлювати так, щоб їхній зміст був максимально зрозумілим. Найпростіший спосіб досягти цього – форматування коду за допомогою відступів, табуляцій, порожніх рядків. Розглянемо два тексти однієї й тієї самої програми. Очевидно, що другий текст більш читабельний саме завдяки використанню його якісного форматування.

Лістинг 1.

```
public class TestCollections
{
    public static void TestList()
    {
        var testList = new List<int> { 3, 4, 2, 1 };
        for (var i = testList.Count - 1; i >= 0; i--) { if
(testList[i] > 2) testList.RemoveAt(i); }
        testList.Sort(); foreach (int el in testList) {
Console.WriteLine(el); }
    }
}
```

Лістинг 2.

```
public class TestCollections
{
    public static void TestList()
    {
        var testList = new List<int> { 3, 4, 2, 1 };
        for (var i = testList.Count - 1; i >= 0; i--)
        {
            if (testList[i] > 2)
            {
                testList.RemoveAt(i);
            }
        }
        /*сортування в лексикографічному порядку*/
        testList.Sort();
        foreach (int item in testList)
        {
            Console.WriteLine(item);
        }
    }
}
```

Зрозумілість і стисліcть коду – не одне й те саме. Головним критерієм вибору синтаксису для коду має бути простота його сприйняття.

Відступи, табуляції, переходи на новий рядок допомагають краще структурувати код. Порожні рядки допомагаю розбивати код програми на логічні сегменти.

Декількома рядками можуть відділятися: секції у вихідному файлі, класи та інтерфейси.

Одним порожнім рядком відокремлюються один від одного: методи, локальні змінні від перших операторів, логічні секції всередині методу для більш зручного читання.

Один прогалик використовується в оголошенні методів після коми, але не перед дужками: `TestMethod (a, b, c);`

Приклад неправильного використання:

`TestMethod (a,b ,c); або TestMethod (a, b, c );`

Так само одиночний прогалик може бути використаний для виділення операторів: `a = b;` неправильне використання: `a=b;`

Також прогалики використовуються і при форматуванні циклів:

`for (int i = 0; i <10; ++i);` неправильне використання:

`for (int i=0; i<10; ++i),` або `for (int i = 0;i <10;+ + i).`

При оголошенні і ініціалізації змінних бажано використовувати «табличне» форматування:

```
string name = "Mr. Ed";
int myValue = 5;
Test aTest  = Test.TestYou;
```

Чи варто розташовувати відкриваючу фігурну дужку в тому самому рядку, що й if або while, чи в наступному? Важлива не стільки конкретика вибраного стилю, скільки логічність і послідовність його застосування.

ОСНОВНІ ДОМОВЛЕНОСТІ ТА РЕКОМЕНДАЦІЇ З НАПИСАННЯ КОДУ

Деякі домовленості та рекомендації з написання коду на C#.

- Не використовуйте публічних або захищених полів, замість цього використовуйте властивості.
- Використовуйте автоматичні властивості.
- Завжди вказуйте модифікатор доступу private, навіть якщо дозволено його опускати.
- Завжди ініціалізуйте змінні, навіть коли існує автоматична ініціалізація.
- Використовуйте порожній рядок між логічними секціями у вихідному файлі, класі, методі.
- Використовуйте проміжну змінну для передачі bool-значення результату функції в умовний вираз `if`.

```
bool isAvailable = Exists() && IsNotBooked() && !Expired();
if (isAvailable)
{
}
```

- Уникайте файлів з більш ніж 500 рядками коду.
- Уникайте методів з більш ніж 200 рядками коду.
- Уникайте методів з більш ніж 5 аргументами, використовуйте структури для передачі великого числа параметрів.
- Один рядок коду не повинна мати довжину більше 120 символів.
- Для коментарів у один рядок використовується «C++» подібний стиль: «//» Цей стиль найбільш зручний при документуванні параметрів. Переважно використовувати даний стиль замість `/* коментар */` там, де це можливо.

```
int i; // змінна для циклу.
```

- Для стандартних блокових коментарів використовуються наступні стилі:

```
/*
 * Line 1
 * Line 2
 * Line 3
 */
```

або такий стиль:

```
/* коментар */
```

- Загальноприйнятим стандартом в оголошеннях є один рядок на екземпляр. Завдяки цьому з'являється зручність читання і написання коментаря:

```
int level; // indentation level
int size; // size of table
```

Бажано не ставити оголошення в один рядок.

```
int a, b; // Неправильно!
```

- Розташовуйте відкриваючі та закриваючі фігурні дужки на новому рядку.
- Використовуйте фігурні дужки для виразів `if`, навіть коли у вираз входить лише один рядок коду.
- Не використовуйте пробіл замість символу табуляції.
- Намагайтеся застосовувати максимум інкапсуляції для примірників об'єктів які ви створюєте. Чи не ставте `public` там де це не потрібно. Використовуйте максимальний рівень захисту. Тобто намагайтеся відкривати доступ від мінімального (`private`) до максимального (`public`).
- Використовуйте властивості замість прямого відкриття `public`, для змінних класу.
- Не використовуйте так званих «магічних чисел». Замість цього оголошуйте константи і статичні змінні:

```
public class MyMath
{
    public const double PI = 3.14159;
}
```

СТИЛІ ВИКОРИСТАННЯ РЕГІСТРІВ

Нижче описані різні способи написання ідентифікаторів та рекомендації з їх застосування.

Стиль Pascal casing

Перша літера ідентифікатора і перша буква кожного наступного приєднаного слова є прописними. Стиль Pascal можна використовувати для ідентифікаторів, що складаються з трьох і більше букв.

Pascal casing рекомендується використовувати для опису:

• всіх визначень типів, у тому числі призначених для користувача класів, перерахувань, подій, делегатів і структур;

• значення перерахувань;

• `readonly` полів і констант;

• інтерфейсів;

• методів;

• просторів імен (namespace);

• властивостей;

• публічних полів;

Стиль Camel casing

Перша літера ідентифікатора рядкова, а перша літера кожного наступного приєднаного слова - прописана.

Стиль **Camel casing** рекомендується використовувати для опису імен:

• локальних змінних;

• аргументів методів;

• захищених (`protected`) полів.

Стиль Upper Case

Всі букви в назві є прописними. Цей стиль рекомендується використовувати лише при іменуванні «коротких» констант, наприклад `PI` або `E`. В інших випадках бажано використовувати Pascal Casing.

В таблиці представлено зведену таблицю використання регістрів, префіксів та суфіксів при наіменуванні ідентифікаторів в C#.

Таблиця. Зведена таблиця використання регістрів та префіксів

Ідентифі-катор	**Регістр**	**Приклад**
Клас / структура	Pascal Casing	`public class TestClass { }`
Інтерфейс	Pascal Casing Починається префіксом «I»	`public interface ITestClass { }`
Значення перелічного типу	Pascal Casing	`enum SampleEnum { FirstValue, SecondValue }`
Перелічний тип	Pascal Casing	
Делегат обробник події	Pascal Casing Закінчується суфіксом «EventHandler»	`public delegate void AnswerCreatedEventHandler(object sender, AnswerCreatedEventArgs e);`
Клас-нащадок від EventArgs	Pascal Casing Закінчується суфіксом «EventArgs»	`public class AnswerCreatedEventArgs : EventArgs { public int CreatedId; public int ParentId; public string CreatorName; }`
Класи виключень	Pascal Casing Закінчується суфіксом «Exception»	`public class SampleException : System.Exception { public SampleException() { } }`
Namespace, властивості, методи, public поля,	Pascal Casing	`namespace System.Security { ... }`

Protected/private поля, параметри	Camel Casing Починається з суфікса «_»	`private int _samplePrivateField;`

Користувацький атрибут	Pascal Casing Закінчується суфіксом «Attribute»	`[AttributeUsage(AttributeTargets.All, Inherited = false, AllowMultiple = true)]`
		`sealed class SampleAttribute : Attribute`
		`{`
		`  public SampleAttribute()`
		`  {`
		`  }`
		`}`

«Коротка» (1-3 літери) константа	Стиль Upper Case	`public const PI = 3.14159;`

ЗРАЗОК КОНТРОЛЬНОЇ РОБОТИ 1

"Інструментальні середовища та технології програмування". Контрольна робота № 1

"Інструментальні середовища та технології програмування". Контрольна робота № 1

* Indicates required question

1. Напишіть своє прізвище та ім'я: *

2. Виберіть свою групу: *

Mark only one oval.

- ◯ К-25
- ◯ К-26
- ◯ К-27
- ◯ К-28

3. Оберіть правильне визначення. Модель даних - це * 1 point

Mark only one oval.

- абстрактне, самодостатнє представлення об'єктів, операторів та інших елементів, які в сукупності складають абстрактну машину доступу до даних, з якими взаємодіє користувач.
- впорядкований набір логічно взаємопов'язаних даних, призначених для задоволення інформаційних потреб певної предметної області
- сукупність програмних засобів, що забезпечують керування створенням та використанням баз даних.
- це будь-які відомості про подію, процес чи об'єкт незалежно від форми їх подання
- це інформація, подана у формі, зручній для зберігання, передачі та обробки людиною чи технічними засобами

4. Оберіть правильне визначення. Транзакція - * 1 point

Mark only one oval.

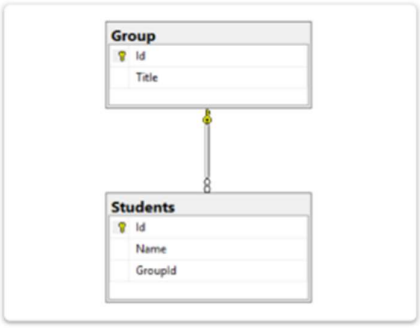
- це збережений запит, доступний як віртуальна динамічна таблиця, що складається з результатів запиту
- це декілька послідовних операторів SQL, які розглядаються як єдине ціле
- це збережена процедура, використання якої обумовлено настанням визначеної події у БД, як: додавання, вилучення чи зміна даних
- досліджувана частина (велика чи мала) реального світу
- інформація, подана у формі, зручній для зберігання, передачі та обробки людиною чи технічними засобами
- відображення інформації користувачу

5. Яка база даних буде сформована в результаті виконання наступного коду (EF Core, .NET 7)? * 3 points

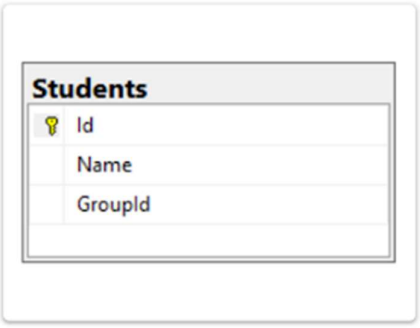
```
class StudentContext : DbContext
{
    1 reference
    public StudentContext()
    {
        Database.EnsureCreated();
    }
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer("Server=.\\SQLEXPRESS; Database=StudentTestDB; Trusted_Connection=True");
    }

    1 reference
    public DbSet<Student> GroupStudents { get; set; } = null!;
    0 references
    public DbSet<Group> Groups { get; set; } = null!;
}

4 references
public class Group
{
    0 references
    public int Id { get; set; }
    1 reference
    public string Title { get; set; } = null!;

    0 references
    public string Info { get; set; } = null!;
}

1 reference
public class Student
{
    0 references
    public int Id { get; set; }
    0 references
    public string Name { get; set; } = null!;

    [NotMapped]
    0 references
    public Group Group { get; set; } = null!;
}
```

p

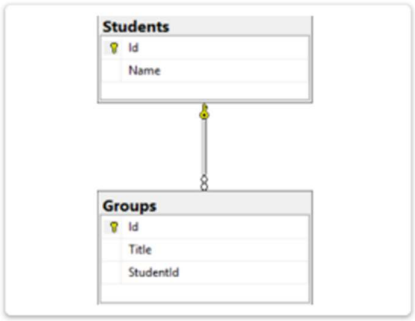
Mark only one oval.



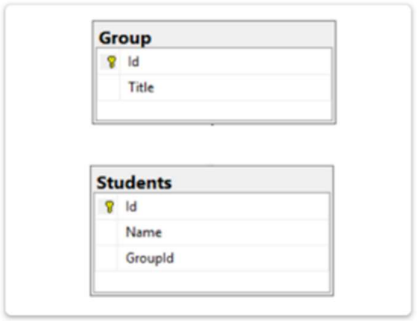
Варіант



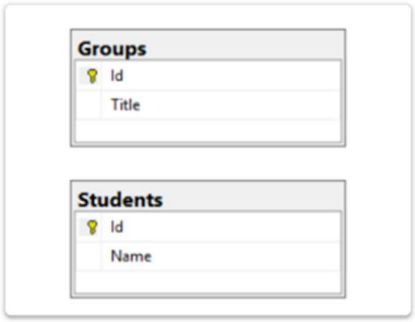
Варіант



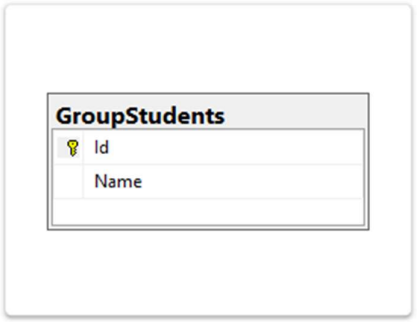
Варіант



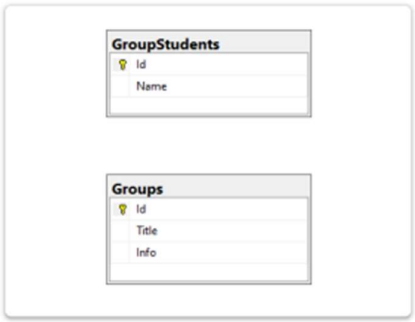
Варіант



Варіант



Варіант



Варіант

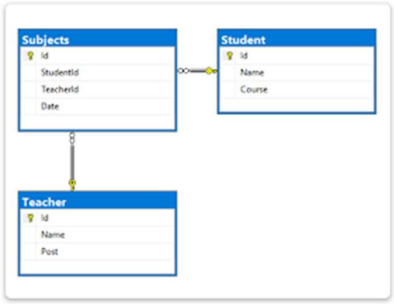
6. Яка база даних буде сформована в результаті виконання наступного коду (EF Core, .NET 7)? * 3 points

```
2 references
class Person
{
    0 references
    public int Id { get; set; }
    1 reference
    public string Name { get; set; }
}
4 references
class Student: Person
{
    0 references
    public int Course { get; set; } = 1;
}
[Table("Teacher")]
2 references
class Teacher:Person
{
    0 references
    public Post Post { get; set; }
}
[Table("Positions")]
1 reference
class Post
{
    0 references
    public int Id { get; set; }
    0 references
    public string PostName { get; set; }
}
1 reference
class Subject
{
    0 references
    public int Id { get; set; }
    0 references
    public Student Student { get; set; }
    0 references
    public Teacher Teacher { get; set; }
    0 references
    public DateTime Date { get; set; }
}
3 references
class StudentContext : DbContext
{
    1 reference
    public StudentContext()
    {
        Database.EnsureCreated();
    }
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer("Server=.\\SQLEXPRESS; Database=Test_DB_KR1; Trusted_Connection=True");
    }
    1 reference
    public DbSet<Student> Students { get; set; }
    0 references
    public DbSet<Teacher> Teachers { get; set; }
    0 references
    public DbSet<Subject> Subjects { get; set; }
}
```

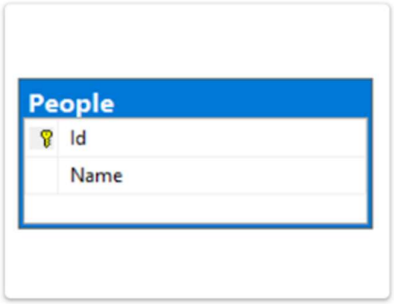
Mark only one oval.

○ База даних сформована не буде. Виникне помилка компіляції

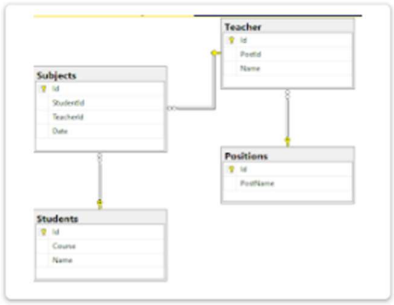
○ Варіант



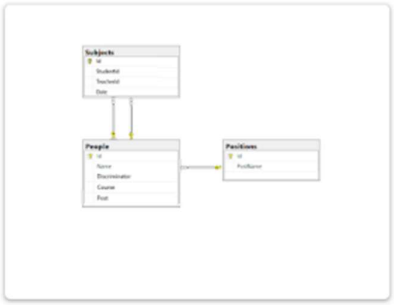
○ Варіант



○ Варіант



○ Варіант



○ Варіант

7. Який метод у класа DbTransaction "відкатує" транзакцію зі стану очікування? * 1 point

Mark only one oval.

- Rollback
- Transaction
- BeginTransaction
- Fixate
- Commit
- Refuse

8. У запиті LINQ, який починається з: from o in orderItems. Колекція OrderItems являє собою набір OrderItem з властивостями OrderID, ProductID, Quantity. Колекція ProductItems являє собою набір ProductItem з властивостями ProcuctID, UnitPrice, ProductName. Ви хочете знайти OrderID, ProcuctID, ProductName, Quantity та TotalPrice (UnitPrice * Quantity) та відсортувати результат за ProductName у спадаючому порядку. Яке ключове слово ви можете використовувати для сортування у запиті LINQ? * 1 point

Mark only one oval.

- ascending
- group
- join
- let
- descening
- by
- asc
- desc

9. Напишіть LINQ-запит для задачі: колекція OrderItems являє собою набір OrderItem з властивостями OrderID, ProductID, Quantity. Колекція ProductItems являє собою набір ProductItem з властивостями ProcuctID, UnitPrice, ProductName. Ви хочете знайти OrderID, ProcuctID, ProductName, Quantity та TotalPrice (UnitPrice * Quantity) та відсортувати результат за ProductName у спадному порядку. * 4 points

10. Програма підключається до сервера баз даних MS SQL. Програма використовує дані з декількох пов'язаних таблиць бази даних. Ви повинні переконатися, що програма може використовуватися, якщо зв'язок відключений або недоступний. Який тип об'єкта слід застосувати для збереження даних з таблиці? * 1 point

Mark only one oval.

- ◯ DataSet
- ◯ DataAdapter
- ◯ DataReader
- ◯ DataServices
- ◯ DBTransaction
- ◯ Command

11. Про який тип об'єкта йде мова: пересилає набори даних зі сховища даних до процесу, який їх викликає і назад. Містять підключення і набір з чотирьох внутрішніх об'єктів команд для вибірки, вставки, зміни та видалення інформації в сховищі даних? * 1 point

Mark only one oval.

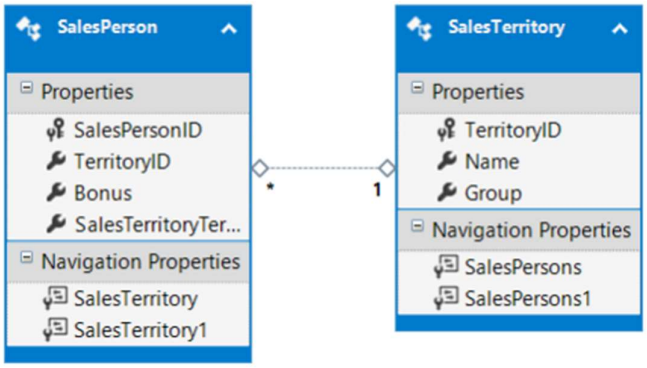
- Connection
- Command
- DataReader
- DataSet
- DataAdapter

12. Який метод виконання команди слід використовувати для виконання запитів (INSERT, UPDATE, DELETE чи виконання зберігаємих процедур, а також запитів мови маніпулювання даними та мови керування даними) інтерфейсу IDbCommand. При виклику метод повертає кількість записів, задіяних при виконанні запиту; * 1 point

Mark only one oval.

- ExecuteNonQuery()
- ExecuteReader()
- ExecuteScalar()
- Execute()
- ExecuteCommand()
- Read()

13. Програма має модель, наведену на рисунку. Вам потрібно розрахувати загальний бонус за продажі здійснені усіма особами в кожній території продажів. * 3 points



Mark only one oval.

```
model.SalesTerritory
  .GroupBy(territory => territory.SalesPersons)
  .SelectMany(group => group.Key)
  .Sum(person => person.Bonus);
```

Запит

```
model.SalesPersons
    .GroupBy(person => person.SalesTerritory)
    .SelectMany(group => group.Key.SalesPersons)
    .Sum(person => person.Bonus);
```

Запит:

```
from person in model.SalesPersons
  group person by person.SalesTerritory
  into territoryByPerson
  select new
    {
        SalesTerritory = territoryByPerson.Key,
        TotalBonus = territuryByPerson.Sum(person => person.Bonus)
    };
```

запит:

```
from person in model.SalesTerritories
    group person by person.SalesPerson
    into territoryByTerritories
    select new
    {
      SalesTerritory = territoryByTerritories.Key,
      TotalBonus = territoryByTerritories.Sum(person => person.Bonus)
      };
```

запит

14. Які з цих класів пересилають набори даних зі сховища даних до процесу, який їх викликає і назад. Містять підключення і набір з чотирьох внутрішніх об'єктів команд для вибірки, вставки, зміни та видалення інформації в сховищі даних. * 1 point

Check all that apply.

- [ ] System.Data.Odbc.OdbcConnection
- [ ] System.Data.Odbc.OdbcDataReader
- [ ] System.Data.SqlClient.SqlConnection
- [ ] System.Data.OleDb.OleDbConnection
- [ ] System.Data.SqlClient.SqlCommand
- [ ] System.Data.OleDb.OleDbCommand
- [ ] System.Data.Odbc.OdbcCommand
- [ ] System.Data.SqlClient.SqlDataAdapter
- [ ] System.Data.OleDb.OleDbDataAdapter
- [ ] System.Data.Odbc.OdbcDataAdapter
- [ ] System.Data.SqlClient.SqlDataReader
- [ ] System.Data.OleDb.OleDbDataReader
- [ ] Спеціальних класів не існує.

15. Про що йде мова: Пов'язані об'єкти завантажуються з бази даних, тільки тоді, коли відбувається звернення до асоційованої властивості сутнісного типу. * 1 point

Mark only one oval.

- ◯ «Ледаче» завантаження (LazyLoading)
- ◯ Явне завантаження
- ◯ Негайне завантаження
- ◯ Швидке завантаження
- ◯ Вірної відповіді немає
- ◯ Завантаження за зверненням

16. Що буде в результаті виконання наступного фрагмента коду? * 3 points

```
DataTable table = new DataTable("table");
DataColumn dcFirstName = new DataColumn(
"Column1", Type.GetType("System.String"));
table.Columns.Add(dcFirstName);
DataRow row;
row = table.NewRow();
Console.Write(row.RowState + "; ");
table.Rows.Add(row);
Console.Write(row.RowState + "; ");
table.AcceptChanges();
Console.Write(row.RowState + "; ");
row["Column1"] = "AAA";
Console.Write(row.RowState + "; ");
row.Delete();
table.AcceptChanges();
Console.Write(row.RowState + "; ");
```

Mark only one oval.

- ◯ Detached; Added; Unchanged; Modified; Deleted;
- ◯ Detached; Detached; Detached; Detached; Detached;
- ◯ Added; Added; Unchanged; Modified; Deleted;
- ◯ Detached; Added; Unchanged; Modified; Detached;
- ◯ Помилка компіляції
- ◯ Added; Added; Unchanged; Detached; Added;
- ◯ Unchanged; Modified; Deleted; Added; Detached;
- ◯ Detached; Added; Unchanged; Unchanged; Deleted;

17. Напишіть фрагмент коду. Опишіть клас контексту з іменем SolarSystemContext, що представляє базу даних SolarSystem. Описати властивості та метод OnModelCreating(), інші методи задавати не потрібно. Опишіть класи сутностей моделі, які представляють описані нижче таблиці бази даних SolarSystem: * 5 points

Опис таблиці Satellites:

Назва	Тип	Призначення	Обмеження
Id	INT	Id супутника, PK	
Name	NVARCHAR(50)	Назва супутника	NOT NULL, по замовчуванню «0»
Value	NVARCHAR(100)	Значення відповіді	NOT NULL, Обмеження на довжину від 1 до 100 символів.
Period	REAL	Період обертання навколо планети	Може бути NULL.
PlanetId	INT	Id планети, FK	

Опис таблиці Planets:

Назва	Тип	Призначення	Обмеження
Id	INT	Id планети, PK	
Name	NVARCHAR(50)	Назва планети	NOT NULL, Обмеження на довжину від 5 до 50 символів.
Period	REAL	Період обертання навколо Сонця	NOT NULL, Значення по замовчування «0».

ЗРАЗОК КОНТРОЛЬНОЇ РОБОТИ 2

"Інструментальні середовища та технології програмування". Контрольна робота № 2

"Інструментальні середовища та технології програмування". Контрольна робота № 2

* Indicates required question

Інформація про студента

1. Напишіть своє прізвище та ім'я: *

2. Виберіть свою групу: *

Mark only one oval.

- ◯ К-24
- ◯ К-25
- ◯ К-26
- ◯ К-27

3. При написанні цієї контрольної роботи зобов'язуюсь дотримуватися правил та принципів академічної доброчесності. *

Mark only one oval.

- ◯ Зобов'язуюсь

4. Який файл в ASP.NET Core є «точкою входу» для застосунку, реєструє набір проміжних сервісів (middleware) разом з застосунком? * 1 point

Mark only one oval.

- RouteConfig.cs
- BundleConfig.cs
- FilterConfig.cs
- Web.config
- Startup.cs
- Global.asax

5. Як буде відображено браузером наступний фрагмент представлення, яке використовує движок Razor? * 2 points

```
@{
    var a = 1;

    <h1>a = (a + 2) </h1>
}
```

Mark only one oval.

- a = 3
- a = (a + 2)
- a = (1 + 2)
- 3
- 1 = 3
- a = 1 + 2
- Виникне помилка
- a = @(a + 2)
- Other: _____

6. Оберіть правильне твердження. Контролери... * 1 point

Mark only one oval.

- ◯ служать для відображення інтерфейсу програми
- ◯ отримують вхідні дані від користувача, обробляють їх та відправляють результат обробки
- ◯ реалізують логіку даних програми
- ◯ обробляють певний запит, реалізують логіку даних програми та відображають результат за певним URL і повертають деякий результат обробки запиту

7. Що буде написано в повідомленні при виконанні наступного скрипта: * 1 point
<script type="text/javascript"> alert(true-false)</script>

Mark only one oval.

- ◯ 0
- ◯ 1 == true
- ◯ true
- ◯ false
- ◯ 1

8. Який атрибут призначений для повного приховування атрибуту (навіть в коді сторінки)? * 1 point

Mark only one oval.

- ◯ ScaffoldColumn
- ◯ HiddenInput
- ◯ Required
- ◯ DataType
- ◯ StringLength

9. Яке з перерахованих тверджень вірне? * 1 point

Mark only one oval.

- ◯ Контролер перенаправляє вхідний запит моделі.
- ◯ Контролер виконує вхідний запит.
- ◯ Контролер контролює дані.
- ◯ Контролер рендерить HTML для перегляду.

10. Яке з наведених відображень містить загальні частини інтерфейсу користувача? * 1 point

Mark only one oval.

- ◯ Partial view
- ◯ Html view
- ◯ Layout view
- ◯ Razor view
- ◯ Home view

11. Як з рядка браузера дістатися до Action - "BookDetails", контролера "BookController" з передачею змінної Id = 2? З урахуванням того що використовується стандартний роутинг ASP.NET MVC. * 2 points

Check all that apply.

- ☐ /BookController / BookDetails / 2
- ☐ /BookDetails / 2
- ☐ /BookController /BookDetails /? Id = 2
- ☐ / Book/BookDetails /? Id = 2
- ☐ / BookController / Book / 2
- ☐ /Book/BookDetails / 2

12. Є код, наведений нижче. Що буде виведено при звертанні до контролера HomeController Action - Index, з врахуванням того що в якості стандартного Layout встановлено _Layout.cshtml? * 2 points

HomeController:

```
public PartialViewResult Index()
{
    return PartialView();
}
```

_Layout.cshtml:

```
Контрольна робота
@RenderBody()
```

Index.cshtml:

```
№2, ІСтаТП
```

Mark only one oval.

- ◯ №2, ІСтаТП
- ◯ Помилка, через те, що не було задано Layout у View - Index.cshtml
- ◯ Контрольна робота №2, ІСтаТП
- ◯ Контрольна робота
- ◯ №2, ІСтаТП Контрольна робота
- ◯ Помилка через відсутність Action - "Index", що повертає тип ActionResult

13. Яку платформу підтримує ASP.NET Core? Оберіть найбільш повну відповідь. * 1 point

Mark only one oval.

- ◯ Windows
- ◯ Linux
- ◯ Mac
- ◯ Windows, Linux, Mac
- ◯ Windows, Linux
- ◯ Linux, Mac
- ◯ Windows, Mac

14. Ви створюєте ASP.NET MVC web-застосунок. В моделі є клас Book, код якого наведено на рис.1. Застосунок повинен надати можливість редагувати назву та інформацію про книгу з моделі (використовується строготипізоване представлення для класу Book). Під час перегляду джерела з веб-переглядача ви знайдете код, наведений на рис. 2. Який фрагмент синтаксису Razor було використано? * 3 points

Рисунок 1.

```
public class Book
{
    public Book()
    {
        AuthorBooks = new HashSet<AuthorBooks>();
    }

    public int Id { get; set; }

    [Required]
    [Display(Name = "Назва ")]
    public string BookTitle { get; set; }

    [Display(Name = "Інформація ")]
    public string BookInfo { get; set; }

    public Category Category { get; set; }

    public int CategoryId { get; set; }

    public ICollection<AuthorBooks> AuthorBooks { get; set; }
}
```

Рисунок 2.

```
Назва <input class="text-box single-line" id="BookTitle" name="BookTitle" type="text" value="Програмування">
<br>
Інформація <input class="text-box single-line" id="BookInfo" name="BookInfo" type="text" value="Інформація про Програмування">
<br>
```

Mark only one oval.

```
@Html.DisplayNameFor(model => model.BookTitle)
@Html.EditorFor(model => model.BookTitle)
<br>
@Html.DisplayNameFor(model => model.BookInfo)
@Html.EditorFor(model => model.BookInfo)
```

Варіант

```
@Html.DisplayNameFor(model => model.BookTitle)
@Html.DisplayFor(model => model.BookTitle)
<br>
@Html.DisplayNameFor(model => model.BookInfo)
@Html.DisplayFor(model => model.BookInfo)
```

Варіант

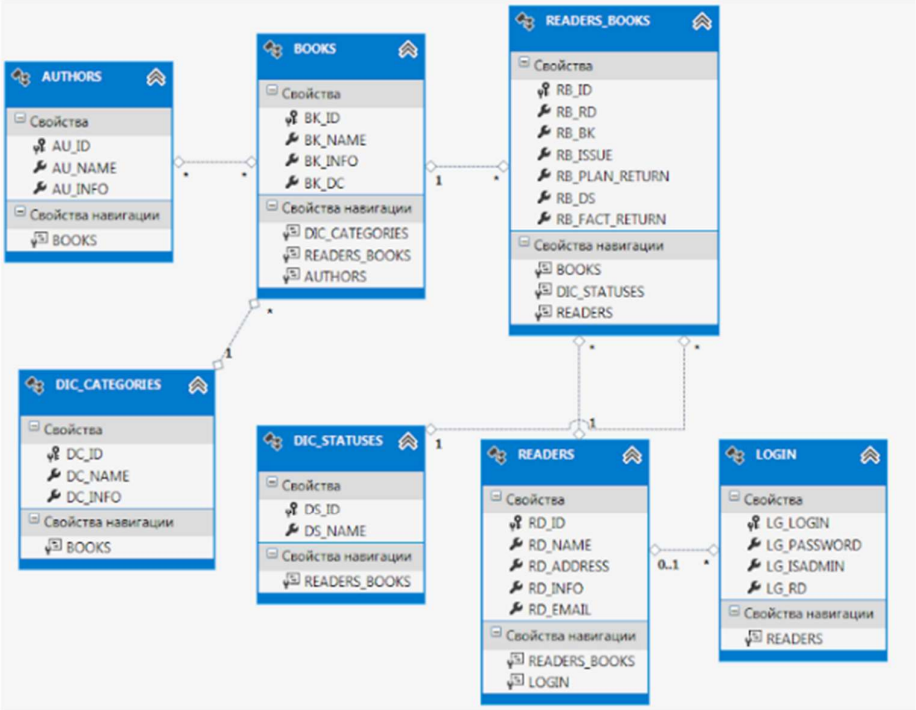
```
Назва <input id="BookTitle"
            name="BookTitle" type="text"
            value="Програмування">
<br>
Інформація <input id="BookInfo" name="BookInfo"
                type="text"
                value="Інформація про Програмування">
```

Варіант

```
Назва @Html.TextBox("BookTitle", Request["BookTitle"],
    new { @placeholder = "Програмування" })
<br>
Інформація @Html.TextBox("BookInfo", Request["BookInfo"],
    new { @placeholder = "Інформація про Програмування" })
```

Варіант

15. Написати метод API контролера для запиту Get, який за id категорії книги (DC_ID) знаходить список авторів (AUTHORS), у яких є книги з цією категорією. Контекст задано в класі контролера: LibraryContext _context;Передбачити можливість асинхронного виконання. Модель відповідає наступній схемі: * 3 points



ПЕРЕЛІК ВЖИВАНИХ СКОРОЧЕНЬ

API	Application Programming Interface
CLI	Command Line Interface – інтерфейс командного рядка
CSS	Cascading Style Sheets – каскадні таблиці стилів
DDD	Domain Driven Design – проєктування, зорієнтоване на домен
EF	Entity Framework
FCL	Framework Class Library – стандартна бібліотека класів платформи .NET Framework
HTML	HyperText Markup Language – мова розмітки гіпертекстових документів
HTTP	HyperText Transfer Protocol – протокол передачі гіпертексту
HTTPS	Hypertext Transfer Protocol Secure – захищений протокол передачі гіпертексту
IDE	Integrated development environment – інтегроване середовище розробки
IoC	Inversion of Control – інверсія управління
JSON	JavaScript Object Notation – нотація об'єктів JavaScript
LINQ	Language Integrated Query – мова інтегрованих запитів
LTS	Long Term Support – тип версії програмного забезпечення з подовженим терміном підтримки
MVC	Model-View-Controller – модель-представлення-контролер
ORM	Object-Relational Mapping – об'єктно-реляційне відображення
REST	Representational State Transfer – передача стану представлення
SQL	Structured Query Language – структурована мова запитів
UI	User Interface – інтерфейс користувача
UML	Unified Modeling Language – уніфікована мова моделювання
XML	Extensible Markup Language – розширювана мова розмітки
БД	база даних
ООП	об’єктно-орієнтоване програмування
ОПП	освітньо-професійна програма
П.І.Б.	прізвище, ім’я, по-батькові
ПС	програмна система
СУБД	Система управління базами даних

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides Released October 1994, Publisher(s): Addison-Wesley Professional – 417 p.
2. Зубенко В.В., Омельчук Л.Л.. Програмування : навчальний посібник (гриф МОН України) / - К. : ВПЦ "Київський університет", 2011. - 623 с.
3. Microsoft Docs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/.
4. Омельчук Л.Л. Об'єктно-орієнтоване програмування. Лабораторний практикум: навчальний посібник / Л.Л.Омельчук, А.С. Белова – Київ: електронна публікація на сайті факультету, 2022. - 273 с. – Режим доступу до ресурсу: http://csc.knu.ua/uk/filer/canonical/1663697725/2129/
5. Ерік Фрімен, Елізабет Робсон, Берт Бейтс, Кеті Сієрра Head First. Патерни проектування - Фабула, 2020 - 672с. ISBN 978-617-09-6159-4.
6. Роберт С. Мартін. Чиста архітектура. – Фабула, 2019 – 368 с. ISBN 978-617-09-5286-8.
7. Роберт С. Мартін. Чистий код. Створення, аналіз і рефакторинг. – Фабула, 2019. ISBN 978-617-09-5285-1.
8. C# docs - get started, tutorials, reference. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/
9. Unified Modeling Language User Guide, The (2 ed.). Addison-Wesley. 2005. p. 496. ISBN 0321267974. , See the sample content, look for history

Додаткова

10. Ставровский А.Б, Карнаух Т.О. Перші кроки програмування.-К.:Диалектика.-2005, с.389.
11. Ерік Фрімен, Елізабет Робсон, Берт Бейтс, Кеті Сієрра Head First. Патерни проектування - Фабула, 2020 - 672с. ISBN 978-617-09-6159-4.
12. Вирт Н. Систематическое программирование. Введение.- М.:Мир,1987.с.184.
13. Вирт Н. Алгоритмы и структуры данных. - М.:Мир,1989.с. 263.
14. Омельчук Л. Л. Методичні вказівки з підготовки та оформлення кваліфікаційних та курсових робіт для студентів факультету комп'ютерних наук та кібернетики [Електронний ресурс] / Л. Л. Омельчук, А. Б. Ставровський. – 2017. – Режим доступу до ресурсу: http://csc.knu.ua/media/filer_public/4f/74/4f7459c9-9e5a-4a77-b8f3-ef30a1f435d5/qualification_work.pdf.
15. Quickstart [Електронний ресурс] // GitHub – Режим доступу до ресурсу: https://docs.github.com/en/get-started/quickstart.
16. C# Coding Conventions (C# Programming Guide) [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions.

17. Introduction to Identity on ASP.NET Core [Електронний ресурс] // Microsoft Learn – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio.
18. Display live data on your site [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/chart.
19. Use the Angular project template with ASP.NET Core [Електронний ресурс] // Microsoft Learn – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/client-side/spa/angular?view=aspnetcore-6.0&tabs=visual-studio.
20. Angular & ASP.NET Core 3.0 - Deep Dive [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.infoq.com/articles/Angular-Core-3/.
21. Walkthrough: Create and run unit tests for managed code [Електронний ресурс] // Microsoft Learn – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code?view=vs-2022.
22. Download Postman for Windows [Електронний ресурс] – Режим доступу до ресурсу: https://www.getpostman.com/downloads/.

Навчальне видання

Омельчук Людмила Леонідівна
Свистунов Антон Олександрович

ІНСТРУМЕНТАЛЬНІ СЕРЕДОВИЩА ТА ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ. ЛАБОРАТОРНИЙ ПРАКТИКУМ

Навчальний посібник

Підписано до друку 21.03.2023 р.
Формат 64×90/16. Папір офс. Друк офс. Ум. друк. арк 21
Гарнітура Times New Roman
160