КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

ІНСТРУМЕНТАЛЬНІ СЕРЕДОВИЩА ТА ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ. ЛАБОРАТОРНИЙ ПРАКТИКУМ

Навчальний посібник

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 № 2 від 21 вересня 2020 року)

О57 Омельчук Л. Л. «Інструментальні середовища та технології програмування. Лабораторний практикум». Навчальний посібник / Л. Л. Омельчук, Н. Г. Русіна. – Одеса: Айс Принт, 2020. – 175 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 робіт, порядок розрахунку семестрової оцінки, а також фрагменти 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

Мета дисципліни – засвоєння знань з обов'язкового освітнього компонента «Інструментальні середовища та технології програмування» освітньо-професійної програми першого (бакалаврського) рівня вищої освіти «Інформатика» факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка. Оволодіння базовими навичками проєктування 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#.

Анотація навчальної дисципліни:

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 – **екзаменом в 4 семестрі.**

В результаті вивчення навчальної дисципліни студент повинен:

знати технології розробки інформаційних програмних систем, принципи роботи технологій доступу до даних на прикладі ADO.Net, основи реляційних баз даних та мови запитів SQL, основи HTML, CSS, JavaScript, базові елементи програмної інженерії, принципи роботи технологій створення вебзастосунків на прикладі ASP.Net Core.

вміти працювати з технологією ADO.Net на автономному рівні, працювати з технологією Entity Framework Core, працювати з технологією ASP.Net Core.

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».

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 проєктувати 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, сховища даних і бази знань, для забезпечення обчислювальних потреб багатьох користувачів, обробки транзакцій, 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;
- здатність використовувати інтелектуальні інформаційні технології;
- здатність реалізовувати фази та ітерації життєвого циклу створення програмних систем на основі моделей та методів розробки програмного забезпечення.

Місце дисципліни. Нормативна навчальна дисципліна «Інструментальні середовища та технології програмування» є складовою циклу професійної підготовки фахівців освітньо-кваліфікаційного рівня «бакалавр».

Зв'язок з іншими дисциплінами. Дисципліна «Інструментальні середовища та технології програмування» є базовою для засвоєння інших курсів з програмування, окремих розділів системного програмування та теорії програмування.

КОНТРОЛЬ ЗНАНЬ І РОЗПОДІЛ БАЛІВ, ЯКІ ОТРИМУЮТЬ СТУДЕНТИ

На першому лабораторному занятті з «Інструментальні середовища та технології програмування» кожен студент визначається з темою лабораторної роботи.

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 дві лабораторні роботи;
- напише звіт до однієї з лабораторних робіт;
- успішно напише дві контрольні роботи.

В кінці семестру передбачено іспит за всіма темами семестру.

Порядок розрахунку загальної оцінки. Лабораторні роботи (проєкти) – 30 = 20+10 балів. За невчасне чи неякісне виконання роботи знімаються штрафні бали. Семестрові контрольні роботи – 20 = 10+10 балів. Написання звіту – 10 балів (5 балів за якість, 5 балів за вчасність).

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

Терміни здачі лабораторних робіт першого семестру:

- 1 лабораторна робота – до 13 тижня семестру;
- 2 лабораторна робота – до 18 тижня семестру;

У разі неякісного виконання лабораторної роботи та звіту, 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 балів за кожен тиждень, який пройшов з моменту закінчення терміну її здачі.

Підсумкова екзаменаційна робота – 40 балів.

При цьому, кількість балів:

- **1-34** відповідає оцінці «незадовільно» з обов’язковим повторним вивченням дисципліни;
- **35-59** відповідає оцінці «незадовільно» з можливістю повторного складання;
- **60-64** відповідає оцінці «задовільно» («достатньо»);

- **65-74** відповідає оцінці «задовільно»;
- **75 - 84** відповідає оцінці «добре»;
- **85 - 89** відповідає оцінці «добре» («дуже добре»);
- **90 - 100** відповідає оцінці «відмінно».

Шкала відповідності

За 100 – бальною шкалою	За національною шкалою	
90 – 100	5	Відмінно
85 – 89	4	Добре
75 – 84		
65 – 74	3	Задовільно
60 – 64		
35 – 59	2	не задовільно
1 – 34		

УМОВИ ЛАБОРАТОРНИХ РОБІТ

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 простих інформаційних систем.

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

До всіх варіантів: Кількість таблиць >>5, їх обсяг не регламентується. Кількість форм та запитів мусить бути достатньою для повного аналізу інформації. Використати допоміжні таблиці (класифікатори, довідники, словники та ін.). Перелік полів таблиць БД не є вичерпним. Повинна бути подана коротка інформація про програму та надаватися допомога користувачу. Діалог україномовний. Передбачити таку структуру таблиць БД, щоб можна було виявляти динаміку змін окремих показників. До складу системи включити можливість пошуку інформації за ключовими словами та динамічної генерації запитів. 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. Інформаційна система має орієнтуватися на потреби кінцевого користувача (а не програміста).

До лабораторної роботи необхідно оформити звіт. Звіт до лабораторної роботи повинен відповідати усім вимогам, які висуваються до курсових робіт .

Усі проєкти студенти повинні розміщувати в розподіленій системі контролю версій GitHub [13, 14, 15]. Напередодні здачі лабораторної роботи необхідно надіслати посилання викладачу лабораторного практикуму.

Загальні вимоги до всіх лабораторних робіт

1. На основі сутностей предметної області створити таблиці бази даних. Рекомендована СКБД: Ms 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 [16].
2. Тестові дані в БД повинні бути коректними (не використовувати як дані: aaa, bbb, ccc) щоб не виникало проблем при презентації продукту.
3. Проєкт повинен бути розміщений в *GitHub*.

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.П., 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.П. автора, назва, анотація, кваліфікаційні ознаки видання, для читачів - П.І.П., 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.П., 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.П. 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.П., 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.П., 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.П., 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.П., 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.П., 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.П., 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.П., 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.П., 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.П., 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.П., 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.П. 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.П., 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.П., 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.П., 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.П., 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.П., 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.П., 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 інтернет-браузерах MS Internet Explorer 5.5 і вище, Mozilla 1.7 і вище, Opera 7.54 і вище.

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).

5. Опис організації інформаційної бази

5.1. 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	...	...

5.2. Опис таблиць

За кожною таблицею навести опис даних:

Таблиця …. Опис даних

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

6. Реалізація системи

Опис загальної структури, можливі невеликі принципові фрагменти коду.

7. 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…..

Лабораторна робота № 1
РОЗРОБКА ВЕБДОДАТКУ

Набір технологій:

- *Entity Framework Core (EF Core) Database First – ORM Framework*
- *ASP.NET Core 3.x MVC Bootstrap*

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 лабораторної роботи 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 та діаграми бази даних, на якій зображено усі таблиці з іменами їх стовпців, позначено ключові стовпці та усі зв'язки між таблицями.	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 ОБОВ'ЯЗКОВО повинен бути узгоджений з викладачем!	**2**
1.1	3-й тиждень від початку семестру	Створити базу даних в SQL Server Management Studio (SSMS), або в іншій, попередньо узгодженій з викладачем СКБД.	PrntScr діаграми БД на пошту викладачу принаймні за 24 години до співбесіди за цим	**3**

			етапом на лабораторному занятті.	
1.2	6-й тиждень від початку семестру	Створити новий проєкт та підключити існуючу БД (див. попередній етап). (**ЛР_1_Етап_2_ІСтаТП.docx**).	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.	**2**
1.3		Налаштування, створення контролелів та представлень (**ЛР_1_Етап_3_ІСтаТП.docx**).		**3**
1.4	7-й тиждень від початку семестру	Зміна майстер-сторінки та стилізація (**ЛР_1_Етап_4_ІСтаТП.docx**).		**2**
1.5	8-й тиждень від початку семестру	Відображення даних на діаграмі (**ЛР_1_Етап_5_ІСтаТП.docx**).	Без цих етапів лабораторна робота може бути зарахована максимум на 10 балів (без нарахування балів 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. Без етапів 1.5-1.7 Ви маєте право	**2**
1.6	10-й тиждень від початку семестру	Робота з файлами (формування та збереження звітів) У форматі MS Excel (експорт та імпорт файлів) (**ЛР_1_Етап_6_ІСтаТП.docx**). **За самостійну реалізацію роботи зі звітами у форматі .docx можливе отримання 1 додаткового балу.**		**3** **+ 1 (додатковий бал)**
1.7	13-й тиждень від	Робота з автентифікацією, авторизацією та ролями (**ЛР_1_Етап_7_ІСтаТП.docx**).		**3** **+ 2**

	початку семестру	**Можливість отримати до 2 додаткові бали** за реалізацію системи керування користувачами, зміни та валідації пароля, керування ролями, підтвердження по Email (https://metanit.com/sharp/aspnet5/16.7.php, https://metanit.com/sharp/aspnet5/16.8.php, https://metanit.com/sharp/aspnet5/16.9.php, https://metanit.com/sharp/aspnet5/16.10.php).	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). При прийнятті враховується виконання етапів, їх якість, терміни, складність обраної предметної області. Особиста співбесіда є ОБОВ'ЯЗКОВОЮ! Без співбесіди бали не нараховуються.	**(додаткових бали)**

ПРИКЛАДИ ВИКОНАННЯ ЕТАПУ 1.0
Завдання етапу 1.0 лабораторної роботи студента
освітньої програми «Інформатика»
2 курсу, бакалаврського рівня
групи К26 , Прізвище Ім'я
з дисципліни «Інструментальні середовища
та технології програмування»

Власна уточнена постановка умови лабораторної роботи: (вписати назву, додати UML- діаграму, та діаграму бази даних).

Приклади оформлення етапу взяті з робіт студентів попередніх років.

«Картини в музеях світу»

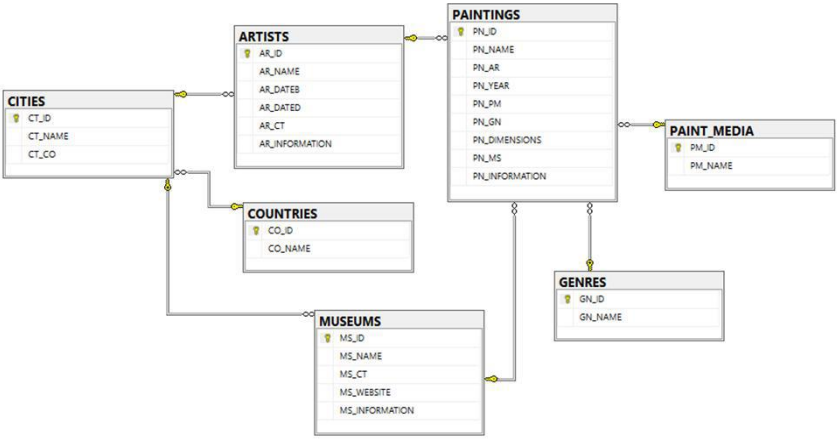


Рисунок 1.0.1 - Діаграма бази даних «Картини в музеях світу»

«Онлайн поліклініка»

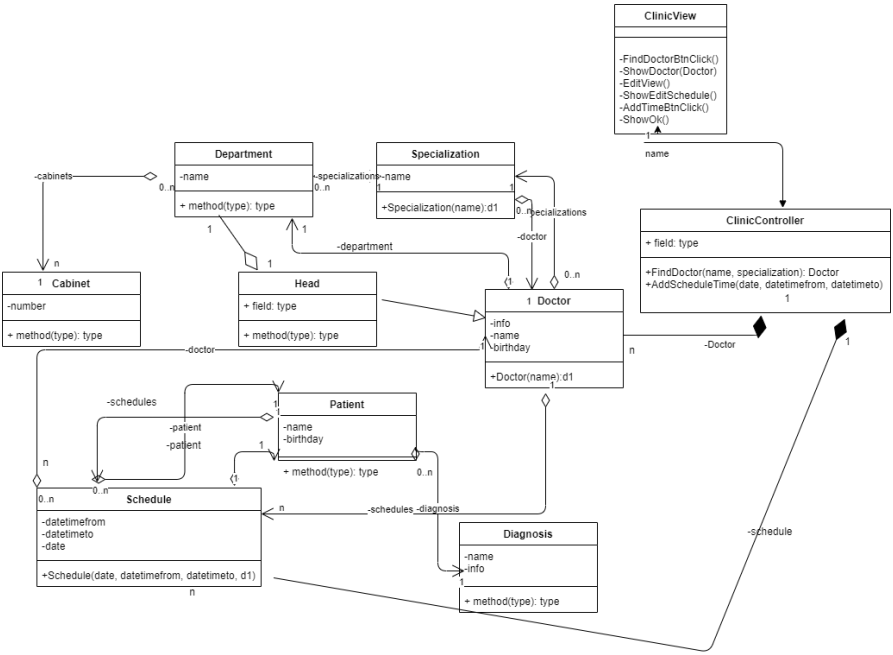


Рисунок 1.0.2 - Діаграма бази даних «онлайн поліклініка»

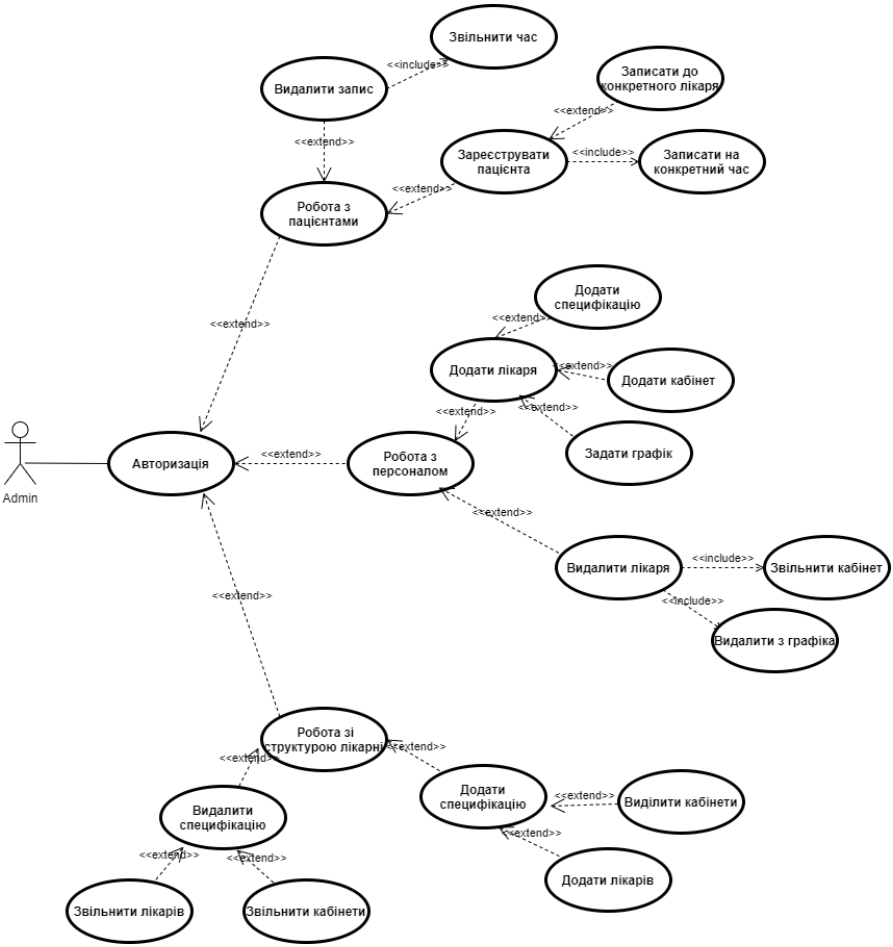


Рисунок 1.0.3 - UML- діаграма «Онлайн поліклініка»

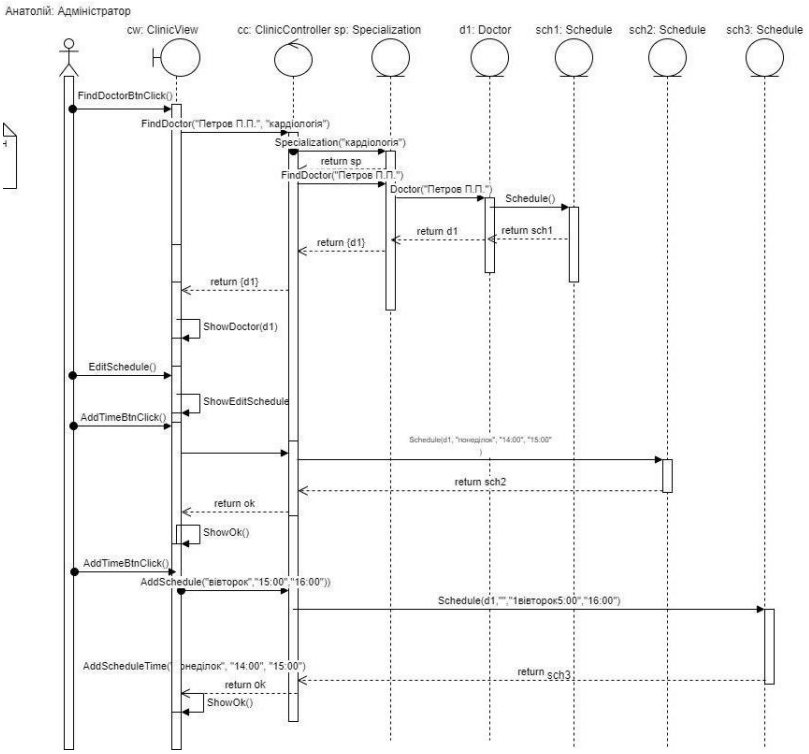


Рисунок 1.0.4 - UML- діаграма «Онлайн поліклініка»

КРОКИ ВИКОНАННЯ ЕТАПУ 1.1

Встановити Ms Visual Studio 2019 та Ms SQL Server Express 2017.

Створення нової бази даних за допомогою Microsoft SQL Server Management Studio (Рис. 1.1.1 - 1.1.2).

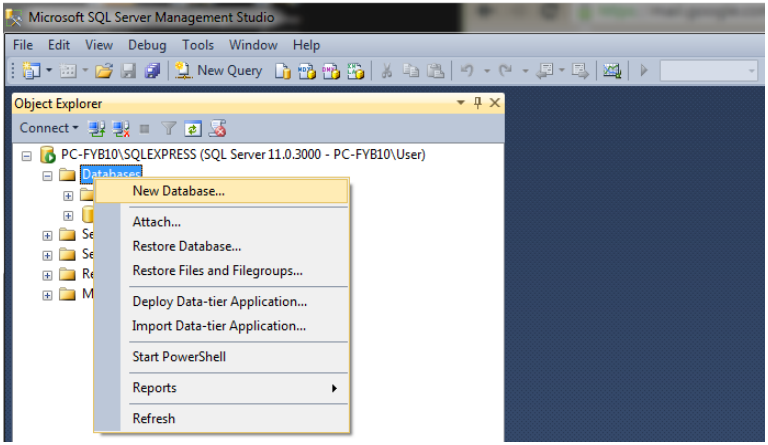


Рисунок 1.1.1 - Робоче вікно створення нової бази даних

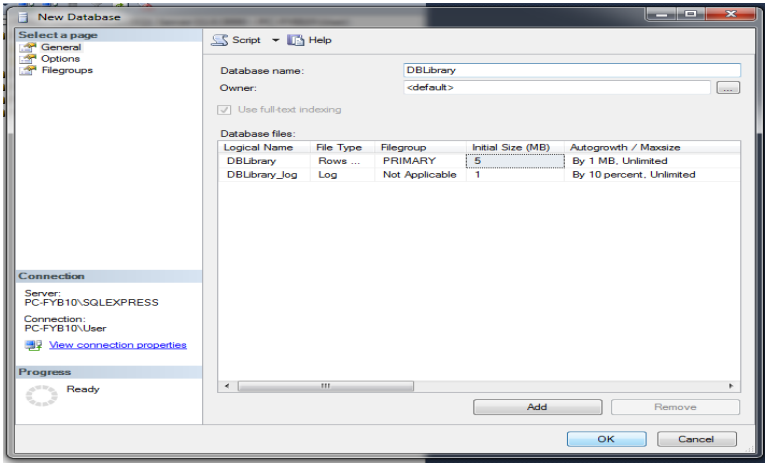


Рисунок 1.1.2 - Робоче вікно створення бази даних

Створення таблиць (Рис. 1.1.3 - 1.1.4).

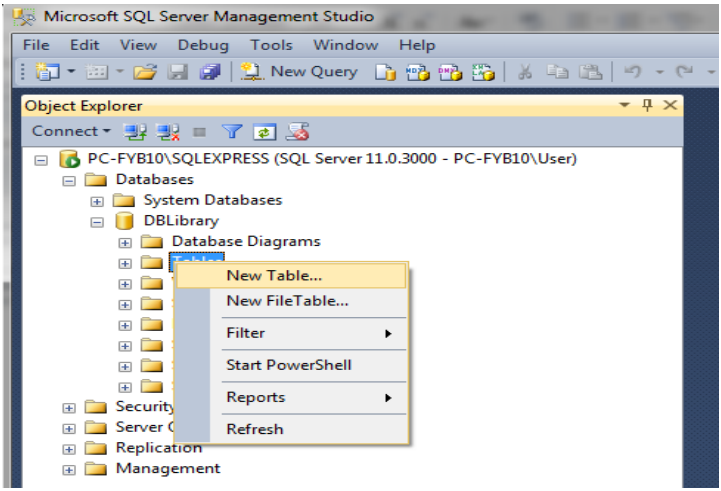


Рисунок 1.1.3 - Робоче вікно створення нової таблиці

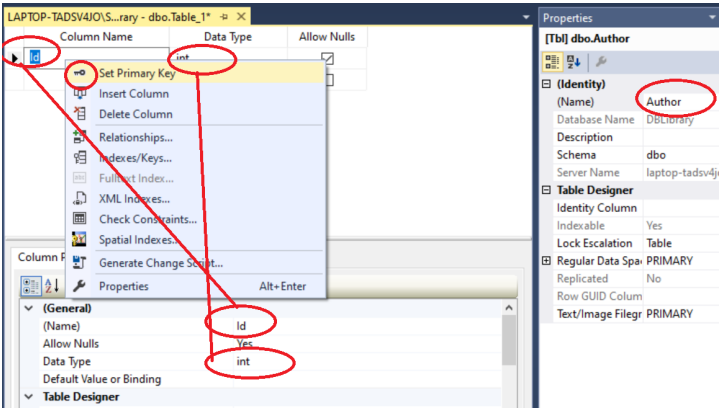


Рисунок 1.1.4 - Фрагмент вікна опису властивостей

Primary Key

Нижчепредставлені рисунки 1.1.5. - 1.1.12 – кроки роботи з Primary Key та їх властивостями..

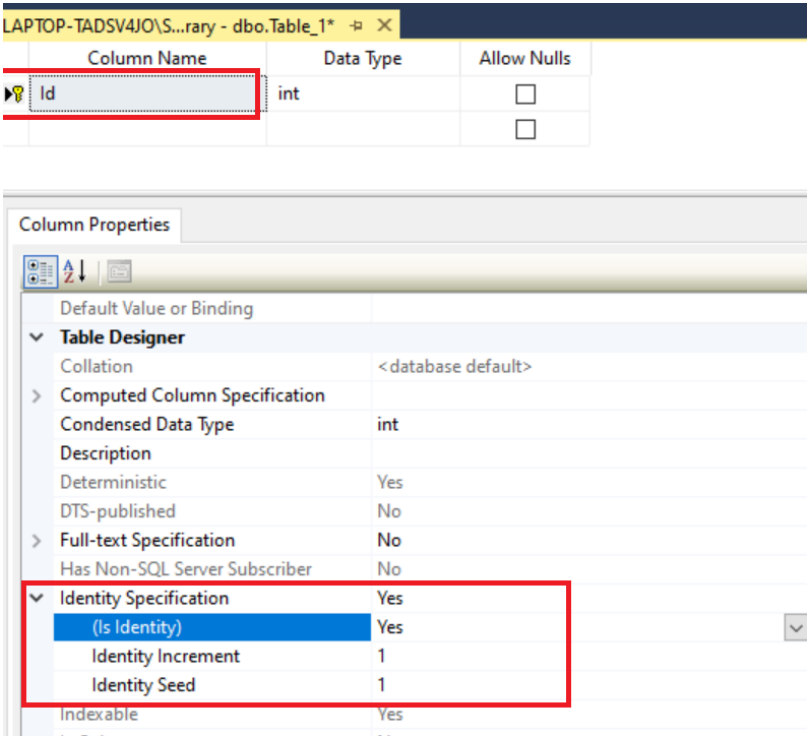


Рисунок 1.1.5 - Фрагмент вікна роботи з Primary Key

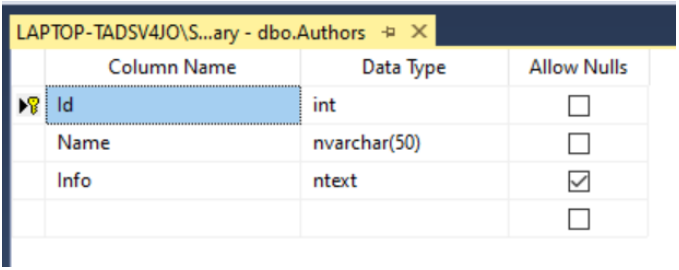


Рисунок 1.1.6 - Фрагмент вікна роботи з Primary Key

Далі створення відбувається аналогічно до попередніх кроків. В результаті повинні отримати наступні таблиці:

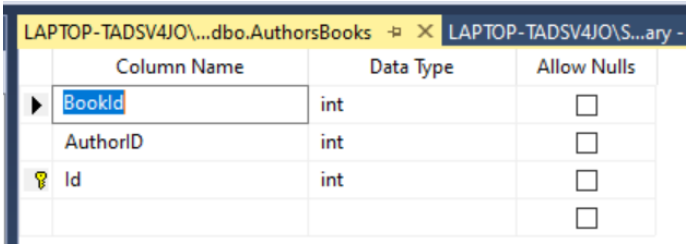


Рисунок 1.1.7 - Фрагмент вікна зміни назви

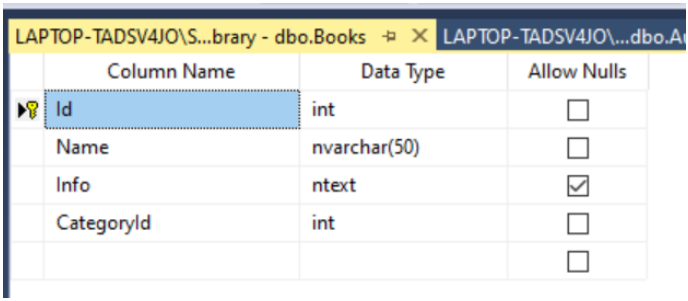


Рисунок 1.1.8 - Фрагмент вікна роботи з Primary Key

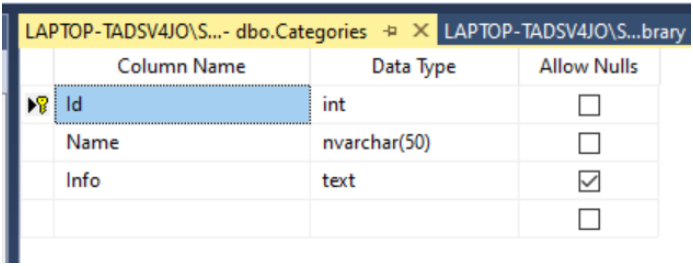


Рисунок 1.1.9 - Фрагмент вікна роботи з Primary Key

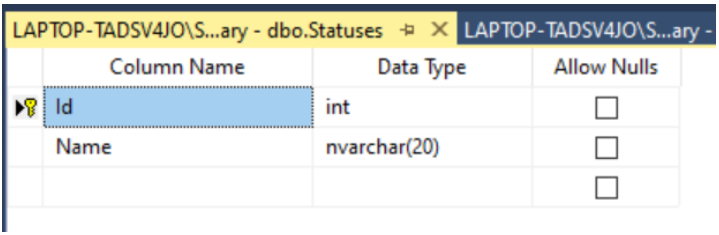


Рисунок 1.1.10 - Фрагмент вікна роботи з Primary Key

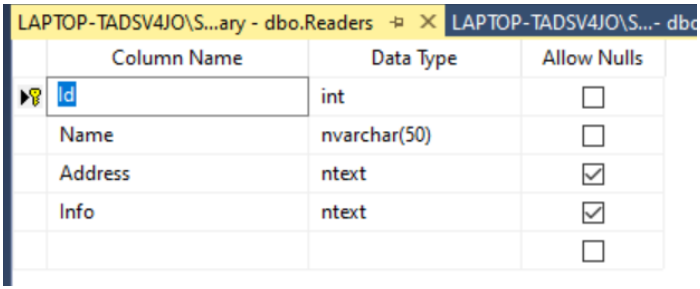


Рисунок 1.1.11 - Фрагмент вікна роботи з Primary Key

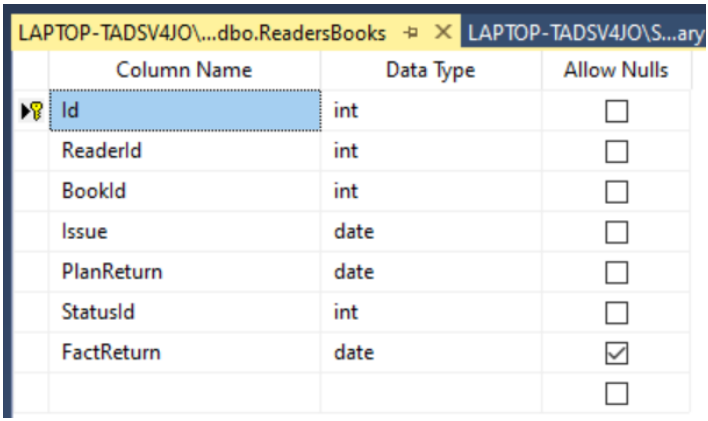


Рисунок 1.1.12 - Фрагмент вікна роботи з Primary Key

Реалізація зв’язків між таблицями представлена на рисунках 1.1.13 – 1.1.19.

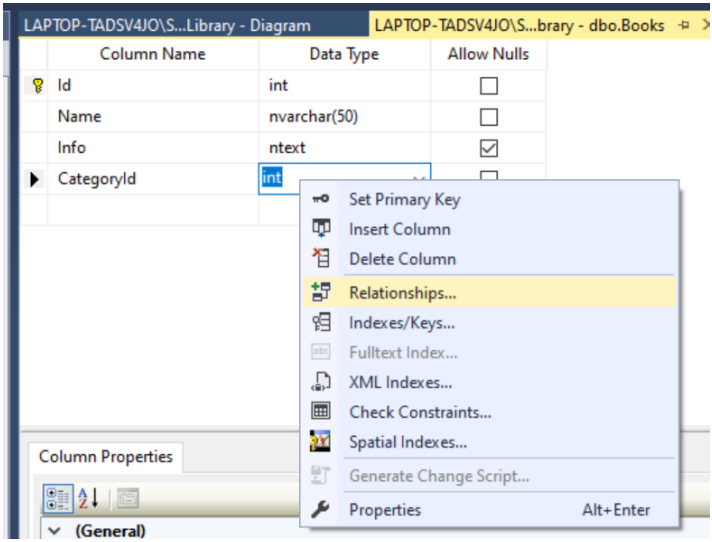


Рисунок 1.1.13 - Фрагмент вікна зв’язків

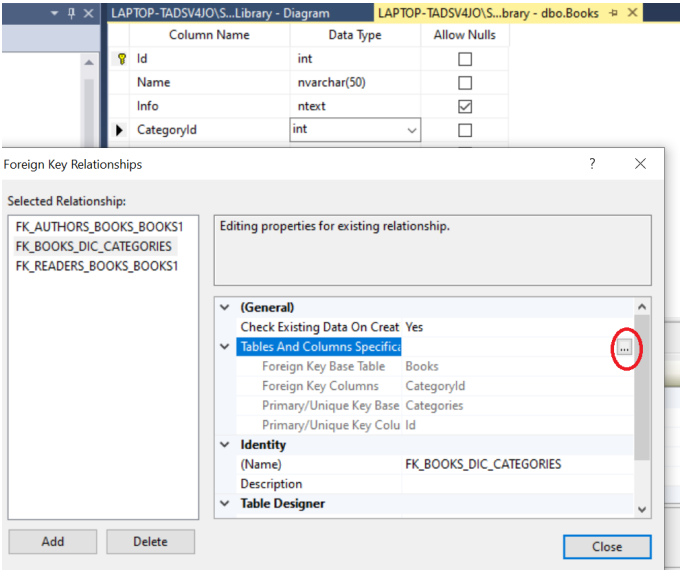


Рисунок 1.1.14 - Фрагмент вікна зв’язків між таблицями

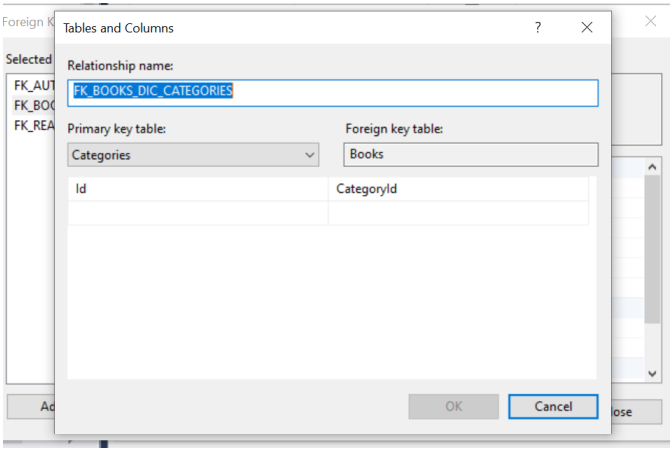


Рисунок 1.1.15 - Фрагмент вікна зміни назви зв’язків між таблицями

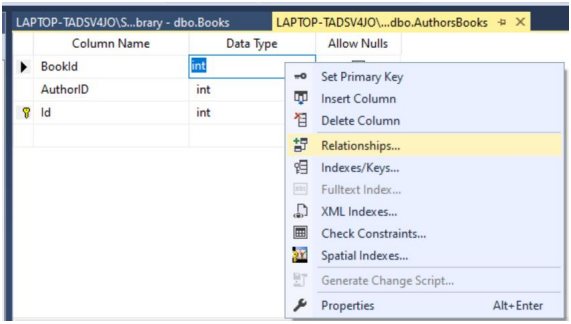


Рисунок 1.1.16 - Фрагмент вікна зв’язків

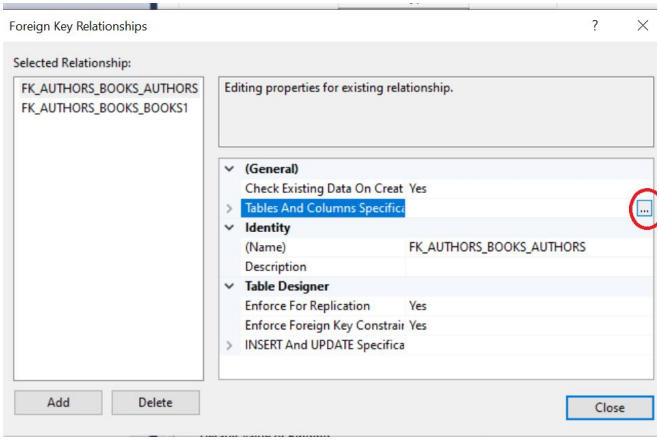


Рисунок 1.1.17 - Фрагмент вікна зв’язків

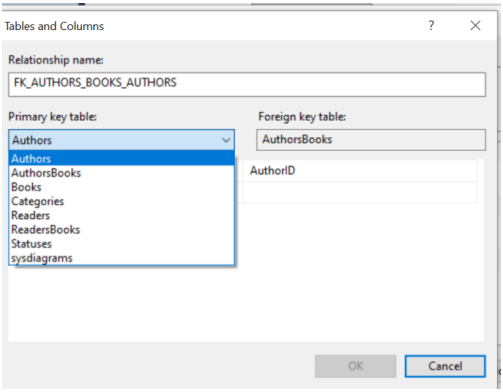


Рисунок 1.1.18 - Фрагмент вікна зв’язків

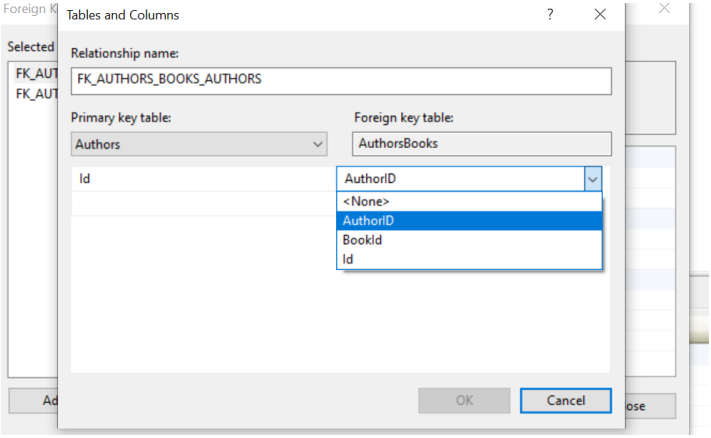


Рисунок 1.1.19 - Фрагмент вікна зв’язків

Далі аналогічно у відповідністю зі схемою бази даних (Рис. 1.1.20 – 1.1.22).

Діаграма бази даних

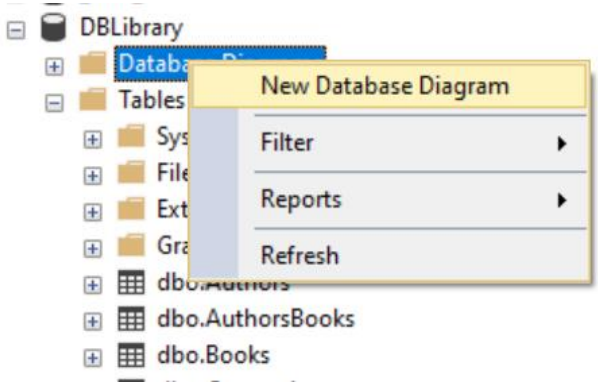


Рисунок 1.1.20 - Фрагмент вікна створення нової бази даних

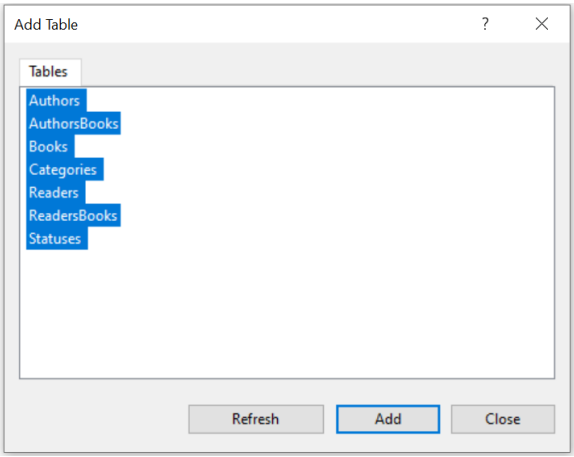


Рисунок 1.1.21 - Фрагмент вікна роботи з базої даних

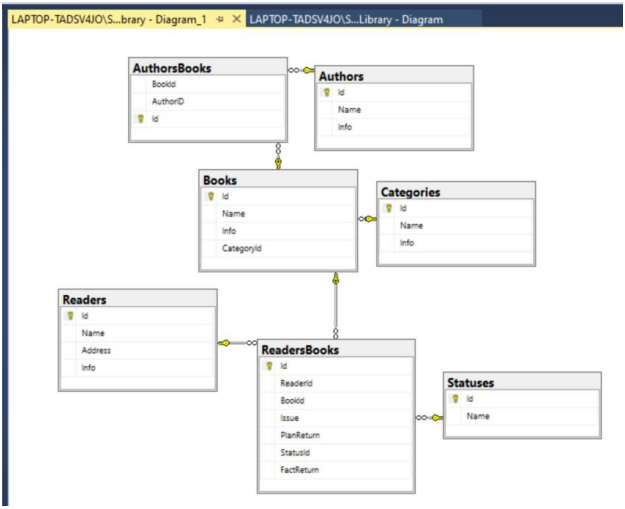


Рисунок 1.1.22- Фрагмент вікна діаграми бази даних

Редагування даних представлено на рисунках 1.1.23 – 1.1.24.

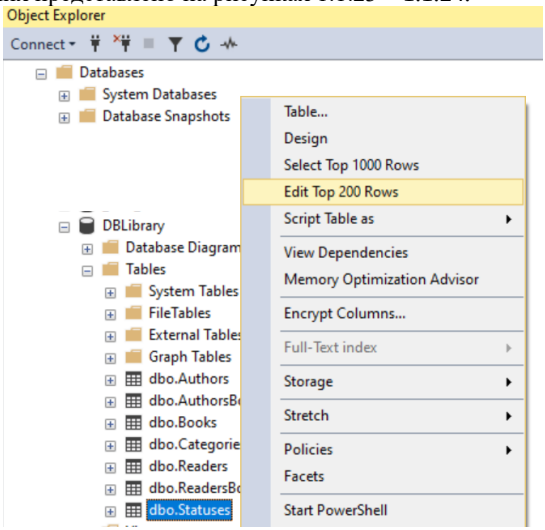


Рисунок 1.1.23 - Фрагмент вікна редагування бази даних

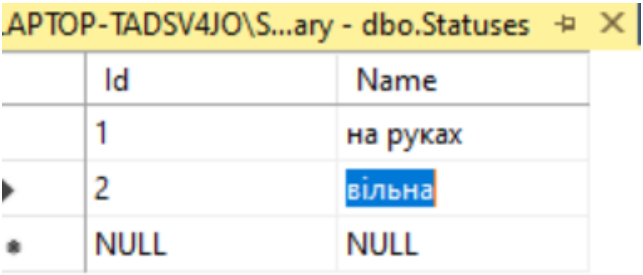


Рисунок 1.1.24 - Фрагмент вікна редагуваня бази даних

Періодично, якщо потрібно оновити виконуйте Refresh (рис.1.1.25).

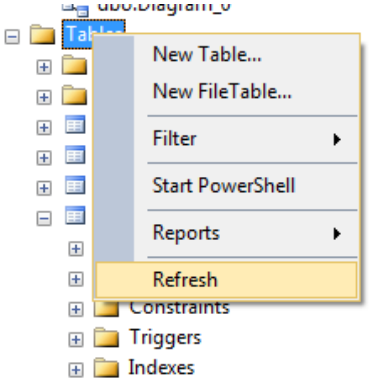


Рисунок 1.1.25 - Фрагмент контекстного меню з Refresh

Для зміни структури бази даних потрібно виконати наступне налаштування (Рис. 1.1.26).

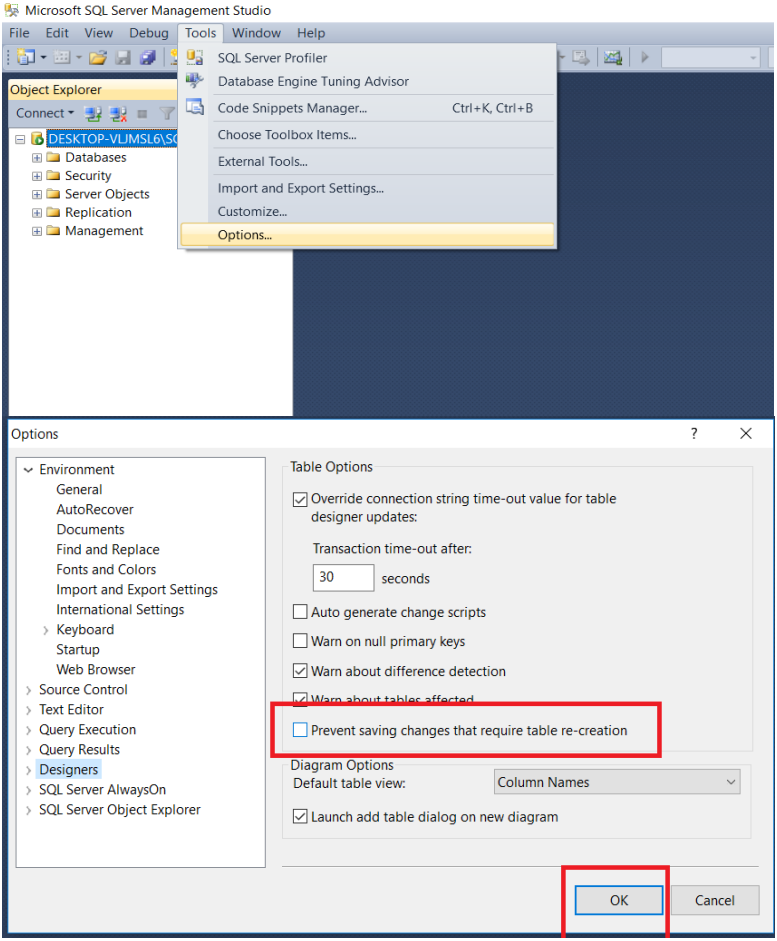


Рисунок 1.1.26 - Фрагмент вікна налаштування бази даних

КРОКИ ВИКОНАННЯ ЕТАПУ 1.2

Завантажити та встановити .NET Core 3.1. SDK (Рисунок 1.2.1-1.2.4).

https://dotnet.microsoft.com/download/dotnet-core/thank-you/sdk-3.1.100-windows-x64-installer

Рисунок 1.2.1 - Фрагмент вікна файлу інсталяції

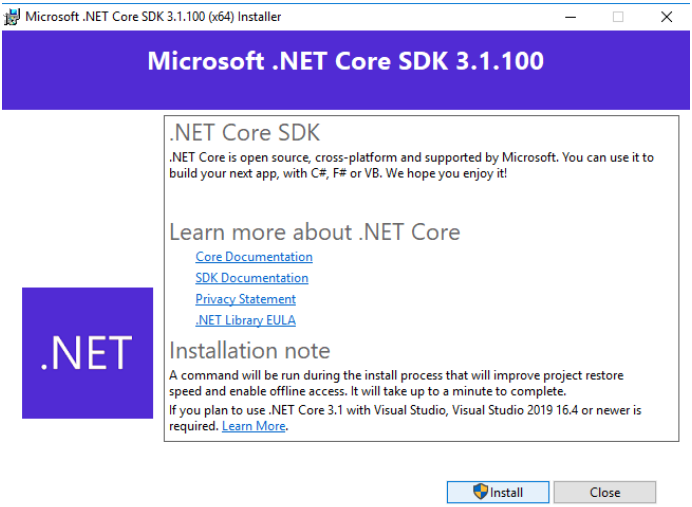


Рисунок 1.2.2 - Фрагмент вікна інсталяції програми

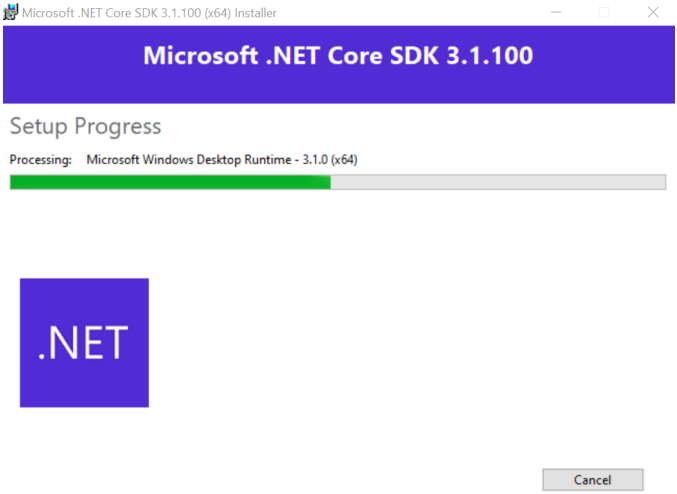


Рисунок 1.2.3 - Фрагмент вікна інсталяції програми

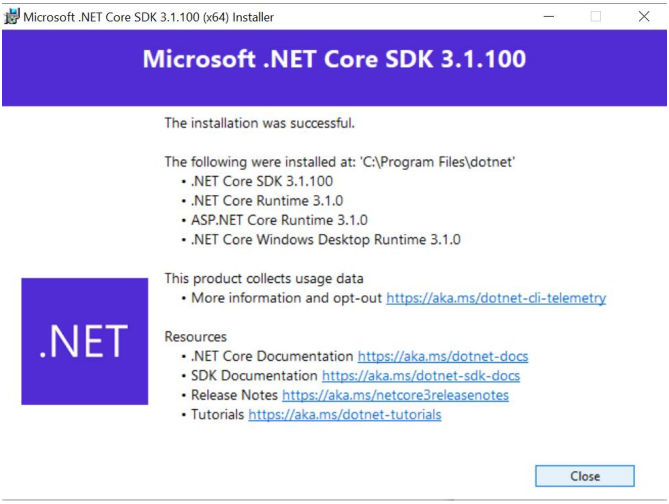


Рисунок 1.2.4 - Фрагмент вікна інсталяції програми

У випадку, якщо версія Visual Studio 19 <16.4, запустити Visual Studio Installer (Рис. 1.2.5 – 1.2.8).

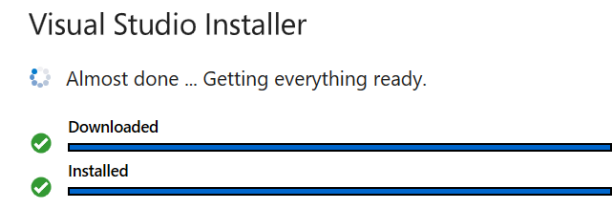


Рисунок 1.2.5 - Фрагмент вікна інсталяції програми

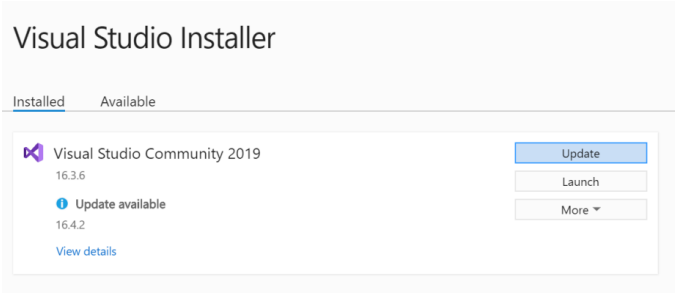


Рисунок 1.2.6 - Фрагмент вікна інсталяції програми

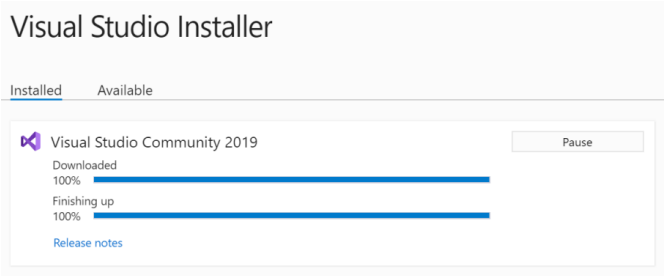


Рисунок 1.2.7 - Фрагмент вікна інсталяції програми

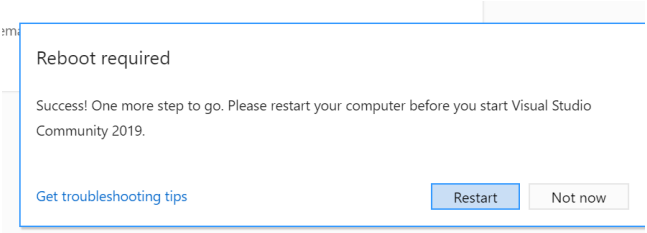


Рисунок 1.2.8 - Фрагмент вікна інсталяції програми

Перезавантажити комп'ютер.

Відкрити Visual Studio 2019 та створити новий проєкт (Рис. 1.2.9 – 1.2.12).

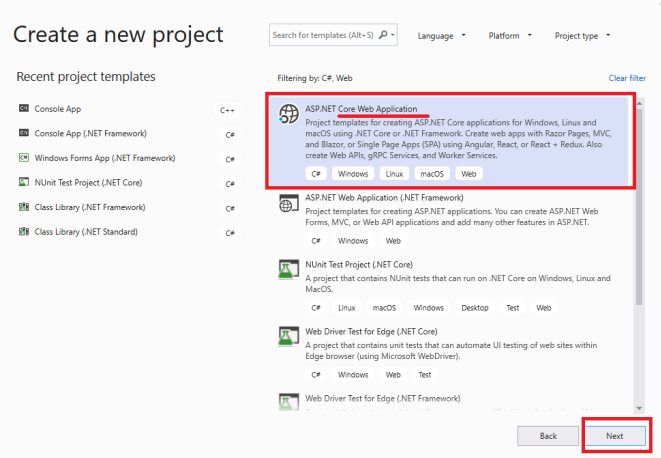


Рисунок 1.2.9 - Фрагмент вікна створення проєкту

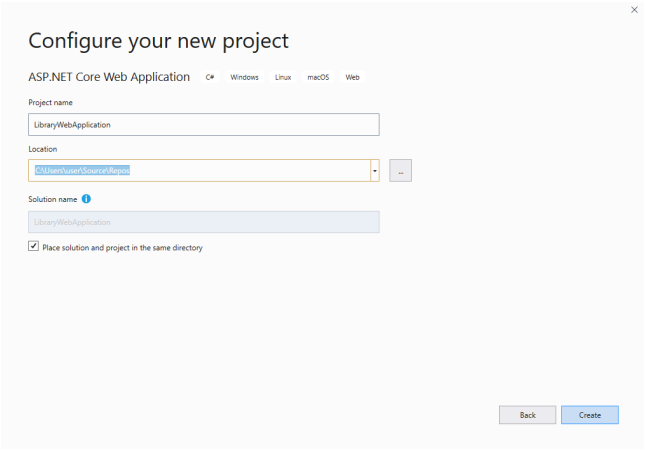


Рисунок 1.2.10 - Фрагмент вікна конфігурації проєкту

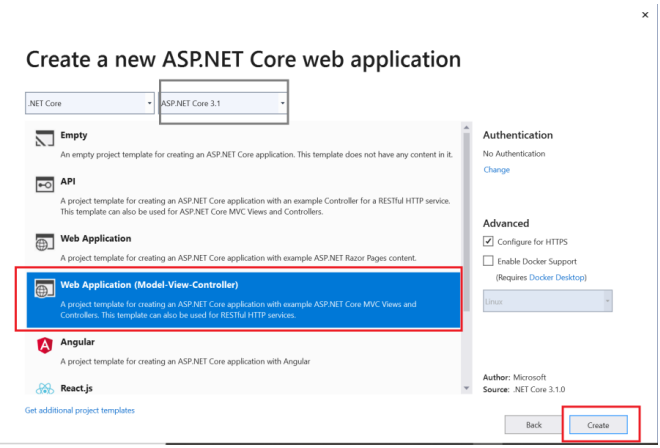


Рисунок 1.2.11 - Фрагмент вікна створення проєкту

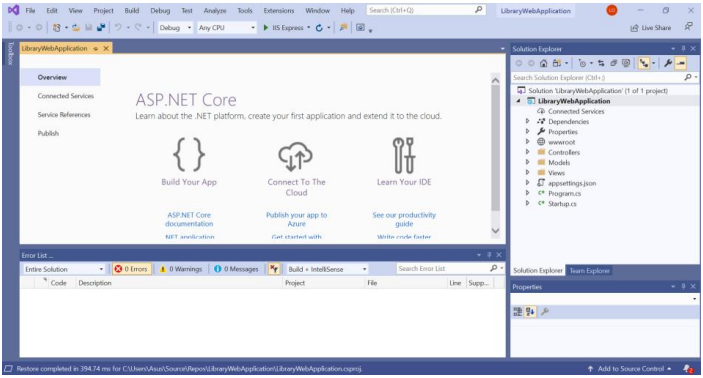


Рисунок 1.2.12 - Фрагмент вікна створення проєкту

Для роботи з існуючою БД MS SQL Server додаємо три пакети:

Microsoft.EntityFrameworkCore.SqlServer;
Microsoft.EntityFrameworkCore.Tools;
Microsoft.EntityFrameworkCore.SqlServer.Design.

Пакет "Microsoft.EntityFrameworkCore.SqlServer" надає функціональність Entity Framework для роботи з MS SQL Server, а пакети –

"Microsoft.EntityFrameworkCore.Tools" і "Microsoft.EntityFrameworkCore.SqlServer.Design" необхідні для створення класів за базою даних, тобто для reverse engineering.

Для додавання цих пакетів здійснимо наступні дії (Рис. 1.2.13 – 1.2.17).

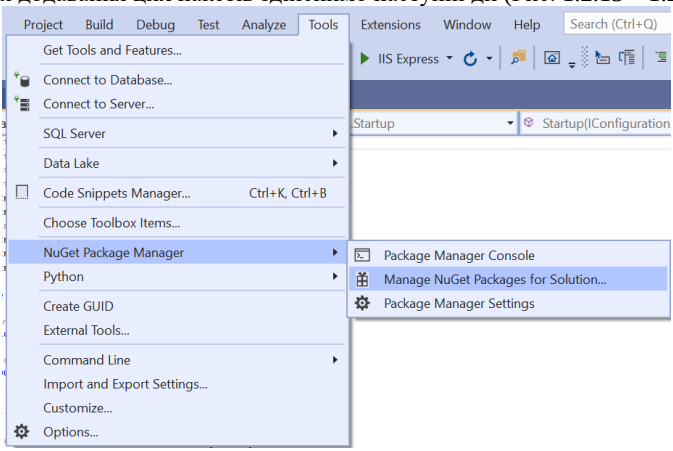


Рисунок 1.2.13 - Фрагмент вікна роботи з пакетом

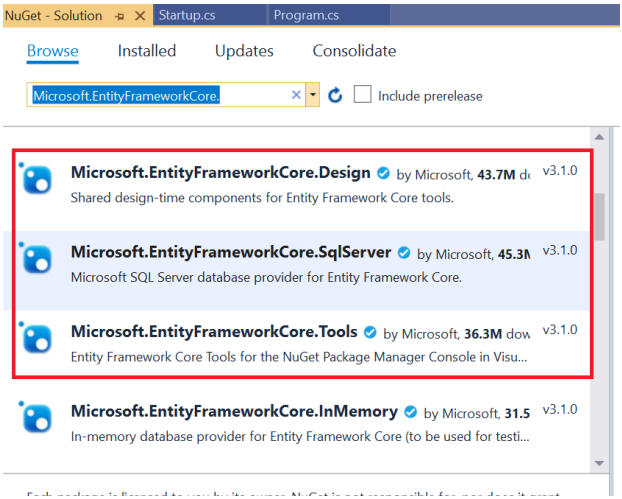


Рисунок 1.2.14 - Фрагмент вікна роботи з пакетом

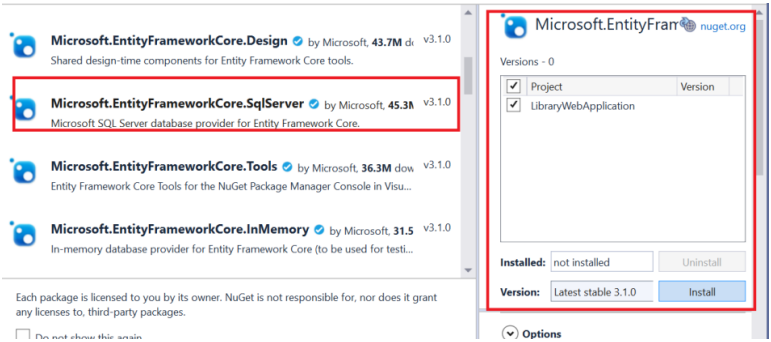


Рисунок 1.2.15 - Фрагмент вікна роботи з пакетом

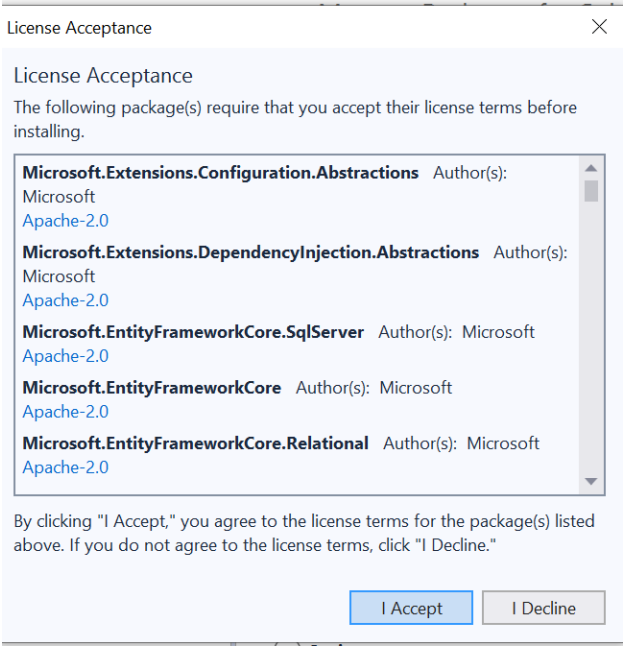


Рисунок 1.2.16 - Фрагмент вікна роботи з пакетом

Для двох інших пакетів здійснити аналогічні дії з їх встановлення.

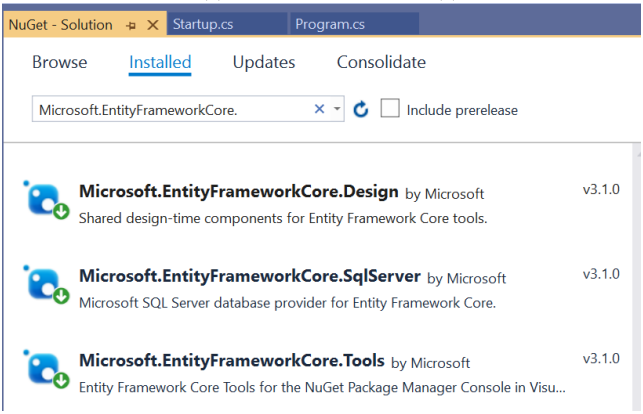


Рисунок 1.2.17 - Фрагмент вікна роботи з пакетом

Для підключення до бази даних, ***попередньо узгодженої з викладачем***, потрібно додати в проєкт класи моделей, що відповідають визначенням таблиць, і клас контекста даних, що відповідає БД.

Для цього в Entity Framework Core 3.1 є функція **Reverse Engineering**, яка дозволяє автоматично створити всі необхідні класи за базою даних.

З цією метою, перейдемо в Visual Studio до вікна Package Manager Console (Tools –> NuGet Package Manager –> Package Manager Console).

В Package Manager Console виконаємо наступну команду:

```
Scaffold-DbContext "Server=LAPTOP-TADSV4JO\SQLEXPRESS; Database=DBLibrary; Trusted_Connection=True; " Microsoft.EntityFrameworkCore.SqlServer
```

Тут в якості параметрів команді Scaffold-DbContext передається рядок підключення з зазначенням сервера і назвою бази даних.

Назву сервера можна дізнатися, наприклад в Ms SQL Server Management Studio (Рис. 1.2.18 – 1.2.21).

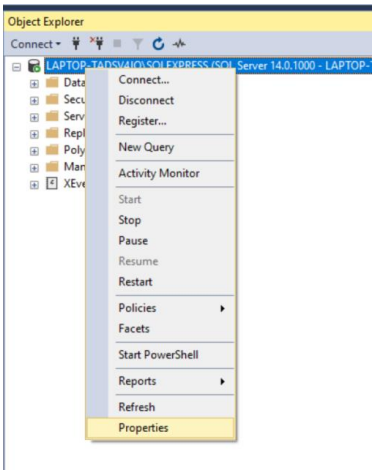


Рисунок 1.2.18 - Фрагмент вікна вибору властивостей

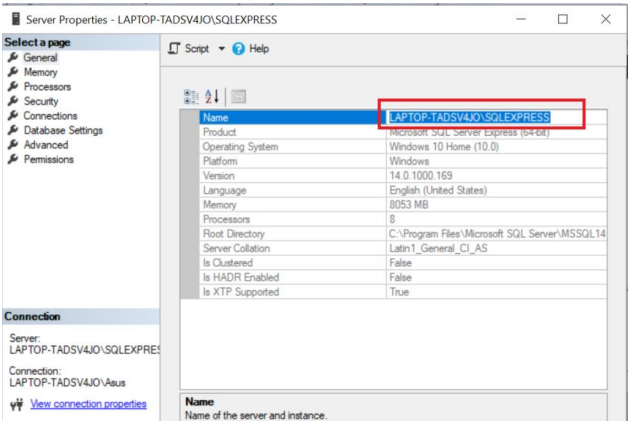


Рисунок 1.2.19 - Фрагмент вікна вибору властивостей

В Package Manager Console виконаємо наступну команду:

Scaffold-DbContext "Server=***LAPTOP-TADSV4JO\SQLEXPRESS***; Database=***DBLibrary***; Trusted_Connection=True; " Microsoft.EntityFrameworkCore.SqlServer

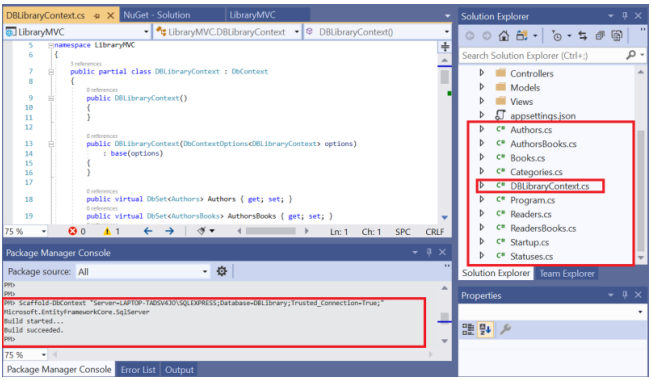


Рисунок 1.2.20 - Фрагмент вікна вибору властивостей

В результаті отримаємо класи моделей, що відповідають визначенням таблиць, і клас контекста даних, що відповідає БД (**детально дослідіть нові класи, проаналізуйте, як реалізовані зв'язки, як здійснено відповідність з таблицями та їх полями. АЛЕ НЕ ЗМІНЮЙТЕ!**). Для дотримання домовленостей, що прийняты в MVC перемістимо нові файли в папку Models.

При Reverse Engineering Entity Framework Core за замовчуванням не застосовує правила плюралізації (таблиці прийнято називати в множині, а класи в однині), тому створено класи Books, Authors, ... а не Book, Author. Якщо така назва не влаштовує, то ми можемо вручну змінити класи. При цьому відповідні зміни потрібно здійснити і в клас контексту даних

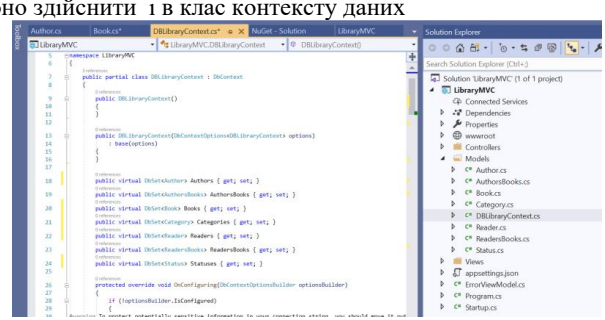


Рисунок 1.2.21 - Фрагмент вікна вибору властивостей

Тепер ми можемо працювати з базою даних. Етап завершено
Для отримання балів за етап – реалізувати для свого проєкту.

КРОКИ ВИКОНАННЯ ЕТАПУ 1.3

НАВЕДЕНО ЛИШЕ ІЛЮСТРАТИВНІ ПРИКЛАДИ. ПРИ РЕАЛІЗАЦІЇ СВОГО ПРОЄКТУ ВИ ПОВИННІ РЕАЛІЗУВАТИ **ПОВНОФУНКЦІОНАЛЬНИЙ** ДОДАТОК!!!

Рядок підключення

Для підключення до бази даних, нам треба задати параметри підключення. Для цього змінимо файл appsettings.json. За замовчуванням він містить тільки налаштування легування (Рис. 1.3.1- 1.3.2).

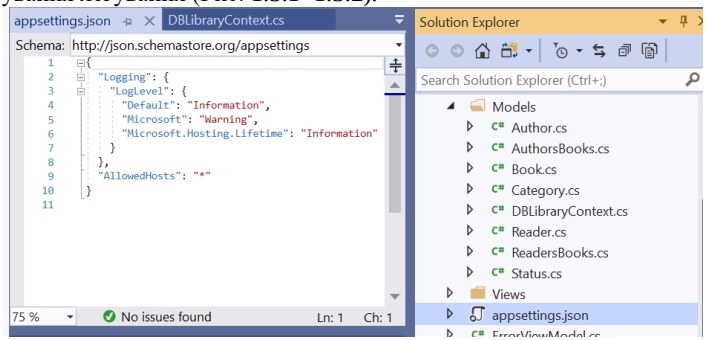


Рисунок 1.3.1 - Фрагмент вікна вибору параметрів

Внесемо зміни, додавши рядки підключення (***параметри рядка підключення див. в ЛР2_Еman_2_ІСmaТП.docx***):

"ConnectionStrings": {

"DefaultConnection": "Server=*LAPTOP-TADSV4JO\\SQLEXPRESS*; Database=*DBLibrary*; Trusted_Connection=True; MultipleActiveResultSets=true"

}

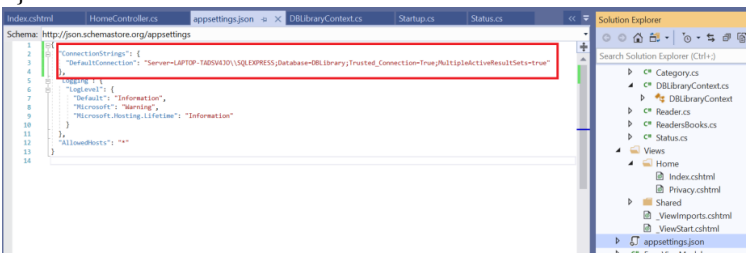


Рисунок 1.3.2 - Фрагмент вікна вибору параметрів

Зміни в файлі Startup.cs

У файлі Startup.cs потрібно змінити метод ConfigureServices() (Рис. 1.3.3).

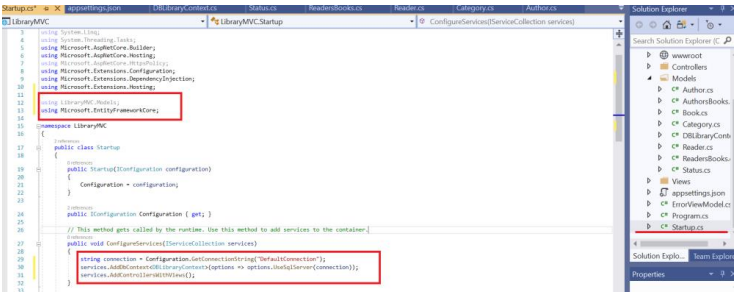


Рисунок 1.3.3 - Фрагмент вікна вибору методу

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).

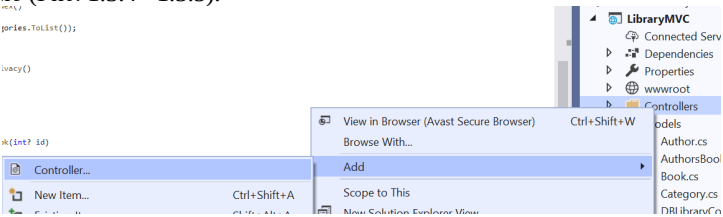


Рисунок 1.3.4 - Фрагмент вікна вибору контролеру

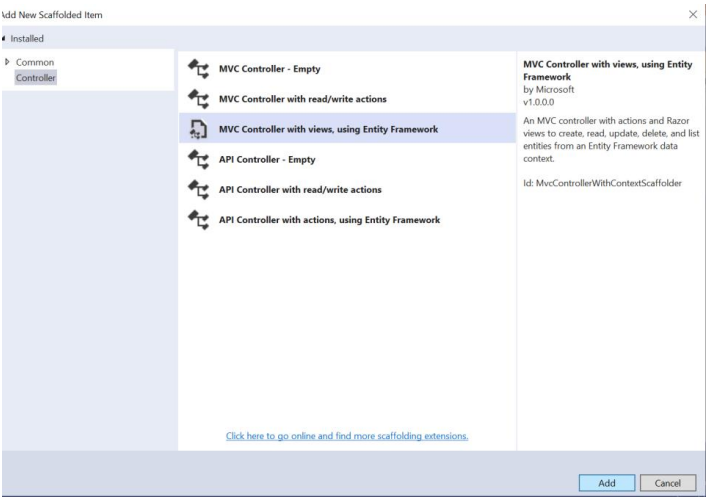


Рисунок 1.3.5 - Фрагмент вікна вибору контролеру

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

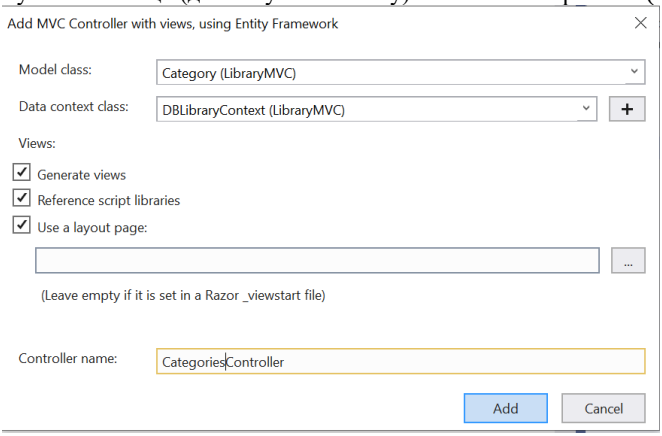


Рисунок 1.3.6 - Фрагмент вікна вибору класу моделі контролеру

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).

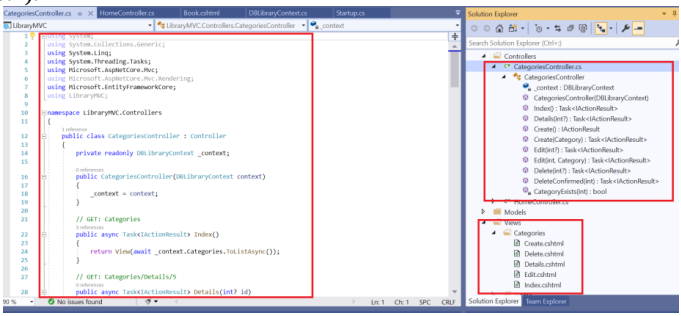


Рисунок 1.3.7 - Фрагмент вікна роботи з контролером

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).

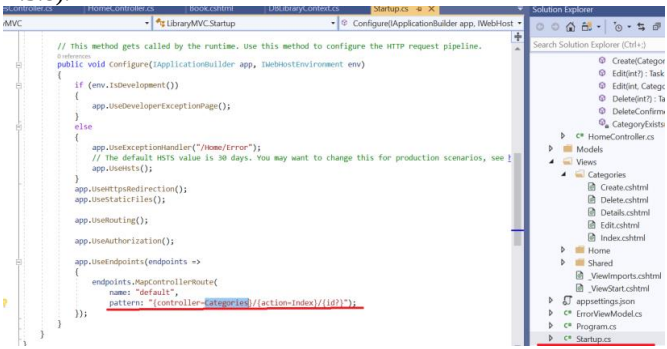


Рисунок 1.3.8 - Фрагмент вікна внесення зміни у файл Startup.cs

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

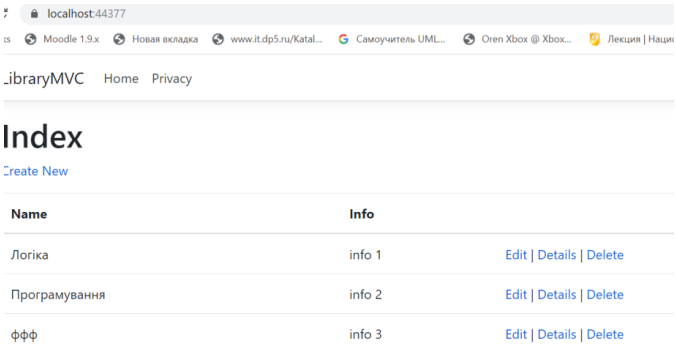


Рисунок 1.3.9 - Фрагмент вікна реалізації

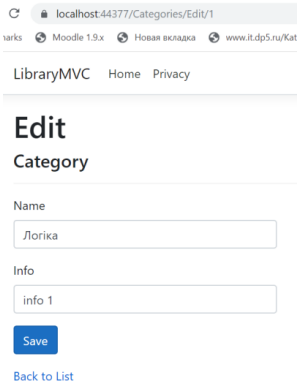


Рисунок 1.3.10 - Фрагмент вікна реалізації

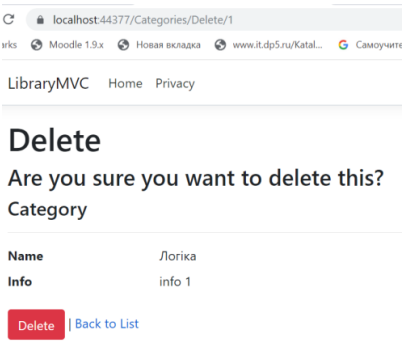


Рисунок 1.3.11 - Фрагмент вікна реалізації

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).

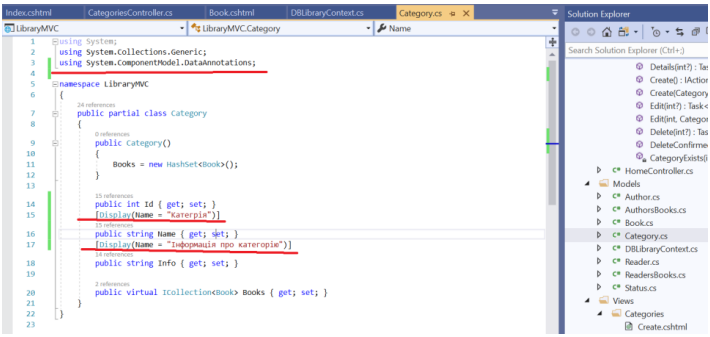


Рисунок 1.3.13 - Фрагмент вікна файлу з моделлю Category

Запускаємо (Рис. 1.3.14).

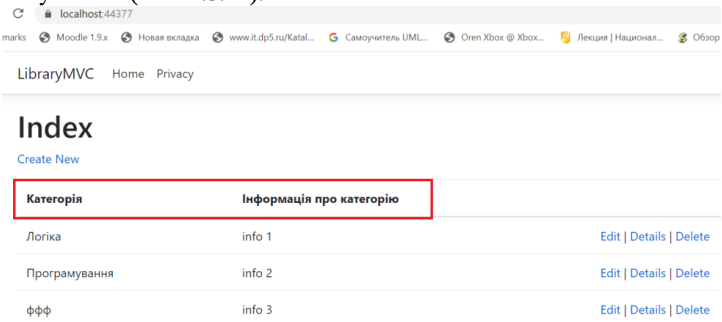


Рисунок 1.3.14 - Фрагмент робочого вікна програми

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).

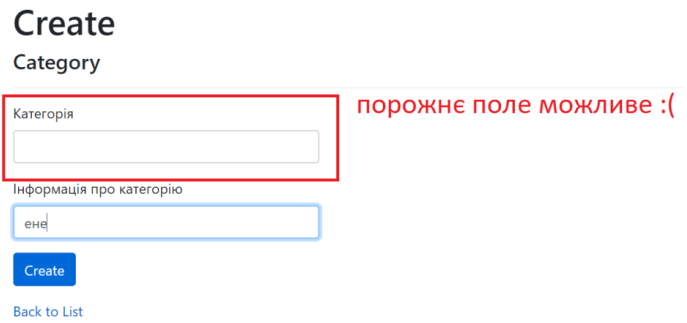


Рисунок 1.3.15 – Фрагмент вікна створення нової категорії

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

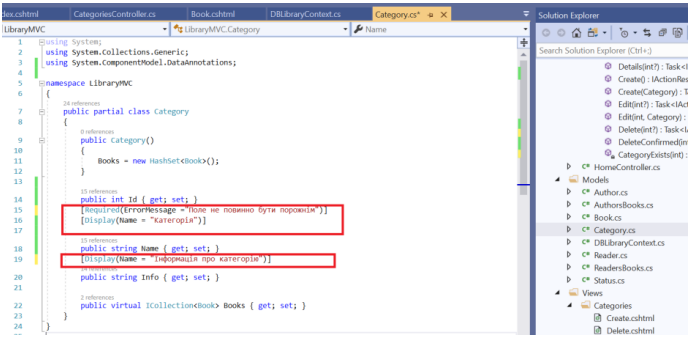


Рисунок 1.3.16 – Фрагмент коду

Запускаємо (Рис.1.3.17)

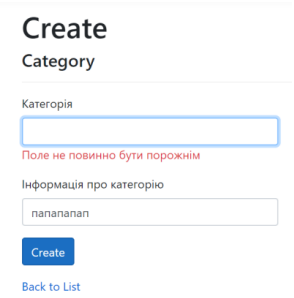


Рисунок 1.3.17 – Фрагмент створення категорії

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).

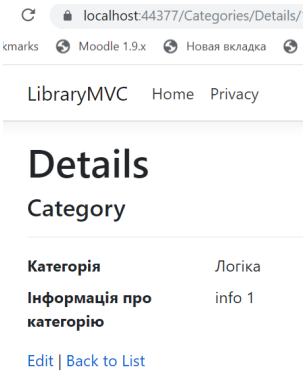


Рисунок 1.3.18 – Фрагмент вікна інформації про категорію

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

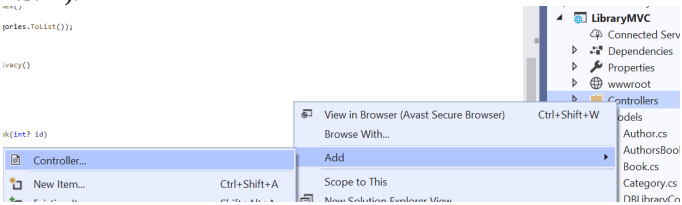


Рисунок 1.3.19 – Фрагмент вікна створення контролеру

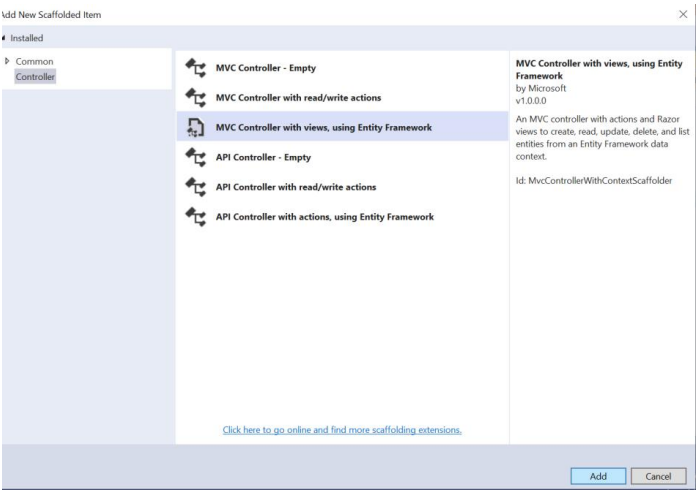


Рисунок 1.3.20 – Фрагмент вікна роботи з контролером

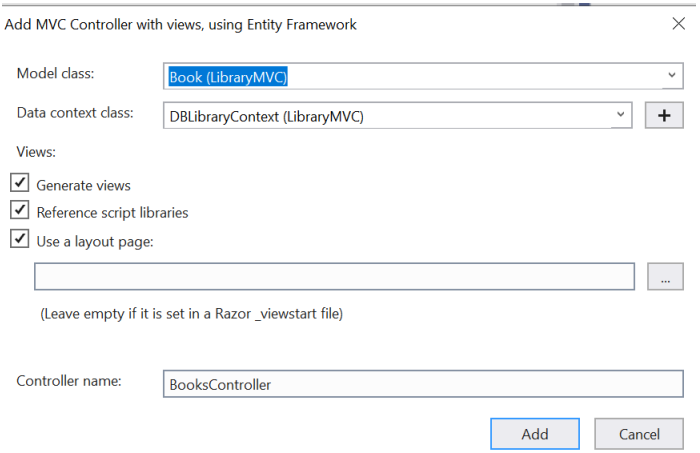


Рисунок 1.3.21 – Фрагмент вікна роботи з контролером

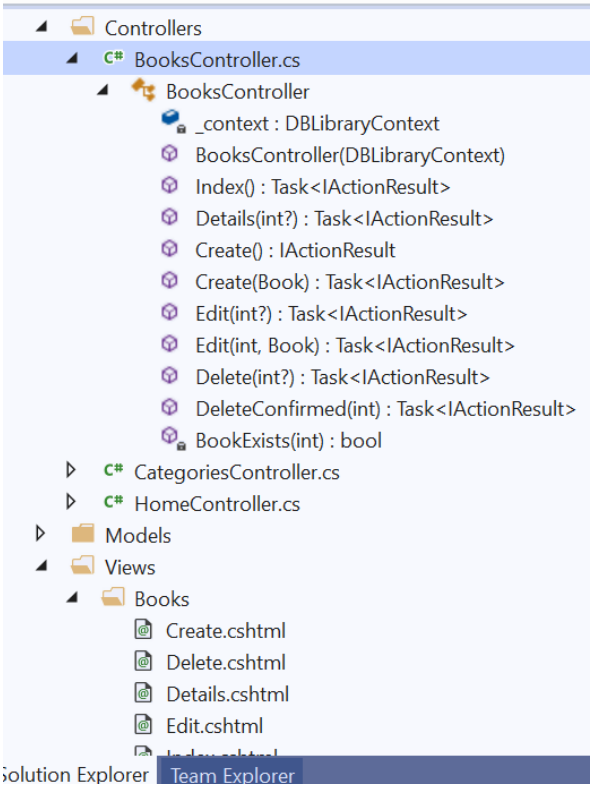


Рисунок 1.3.22 – Фрагмент вікна перегляду контролерів

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).

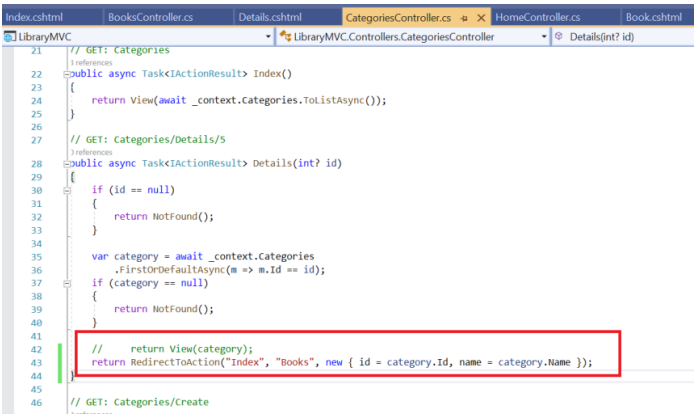


Рисунок 1.3.23 – Фрагмент коду

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

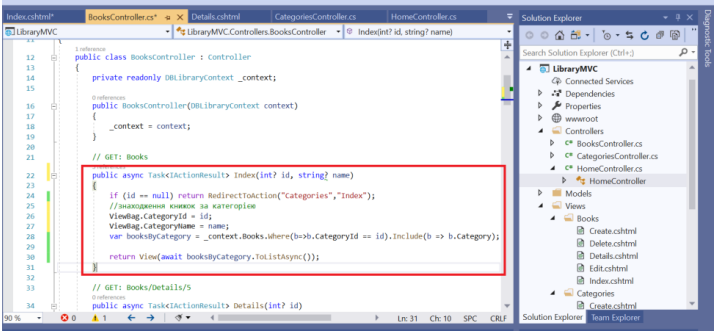


Рисунок 1.3.24 – Фрагмент коду

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).

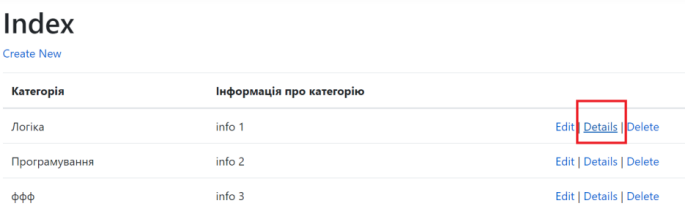


Рисунок 1.3.26 – Фрагмент вікна запуску програми

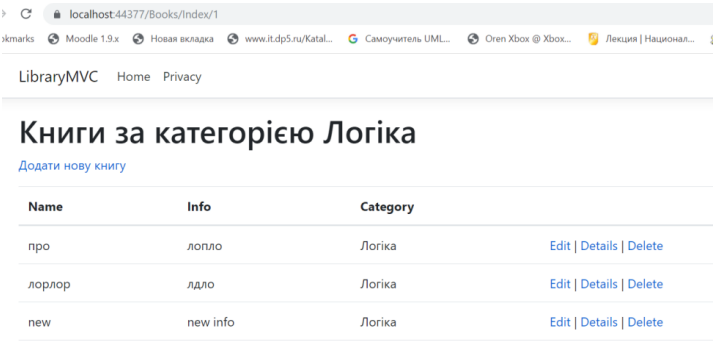


Рисунок 1.3.27 - Фрагмент вікна запуску програми

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)

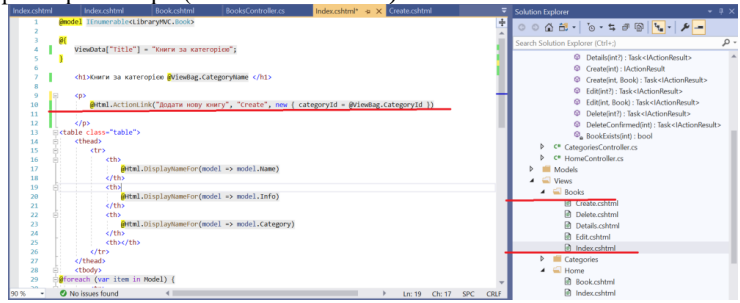


Рисунок 1.3.29 – Фрагмент вікна коду

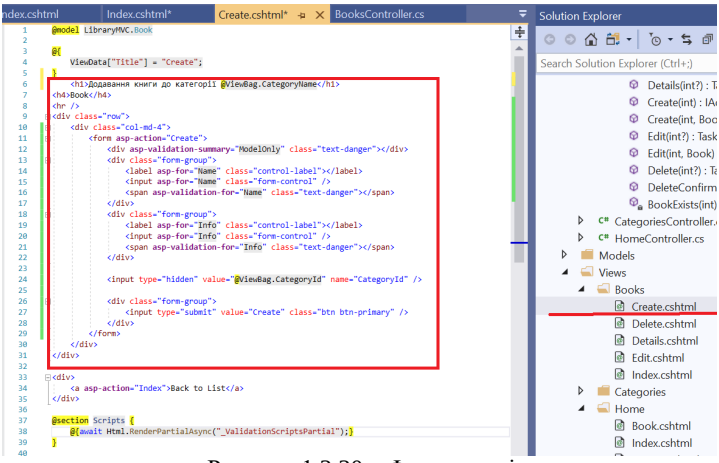


Рисунок 1.3.30 – Фрагмент вікна коду

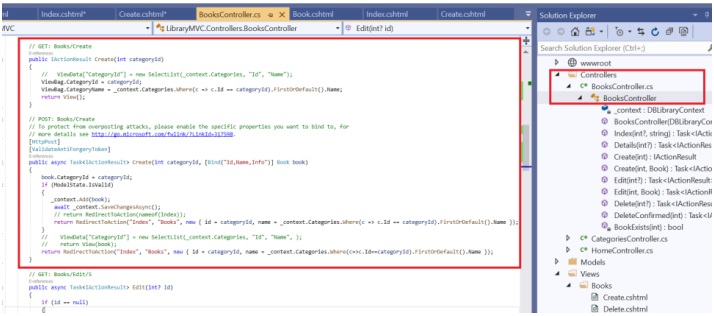


Рисунок 1.3.31 – Фрагмент вікна коду

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).

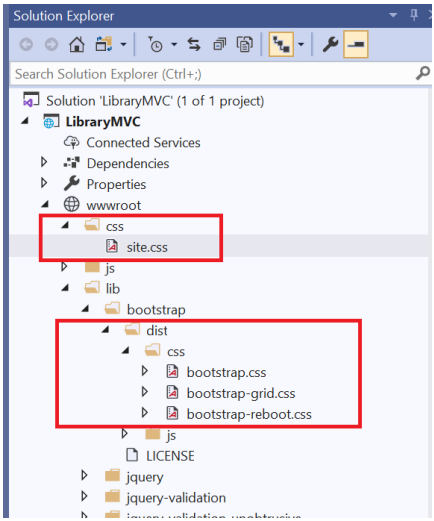


Рисунок 1.4.2 – Фрагмент вікна зміни стилів

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 опускали відповідь на питання про Layout*** **–** тому, використовується майстер-сторінка по замовчуванню**.**

Це прописано на рисунку 1.4.3.

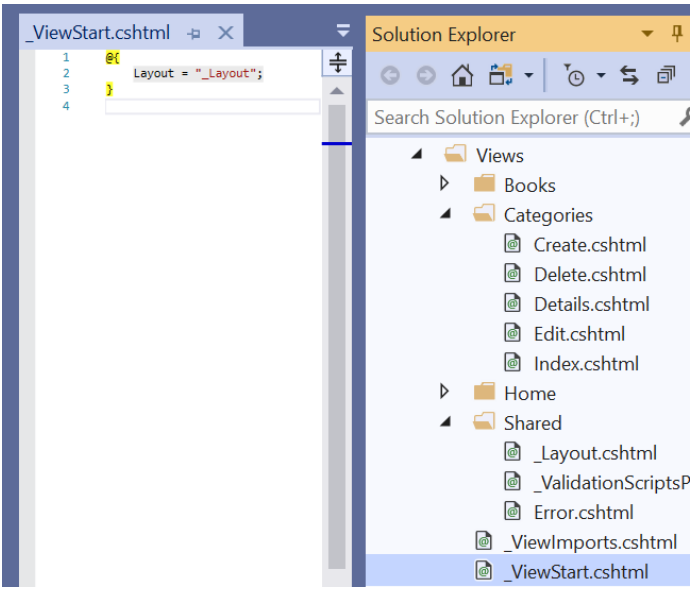


Рисунок 1.4.3 – Фрагмент вікна коду

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:

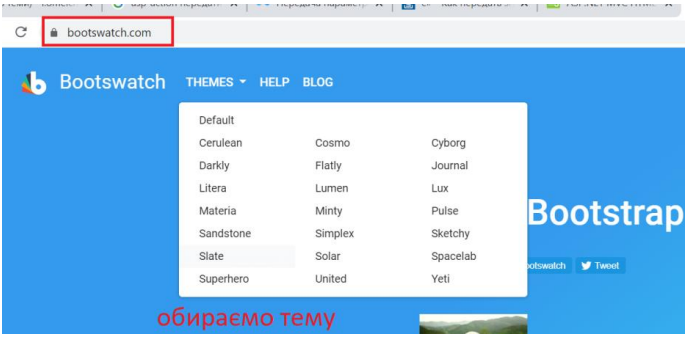


Рисунок 1.4.4 – Фрагмент вікна сайту

Завантажте потрібні файли (рис.1.4.5 – 1.4.6).

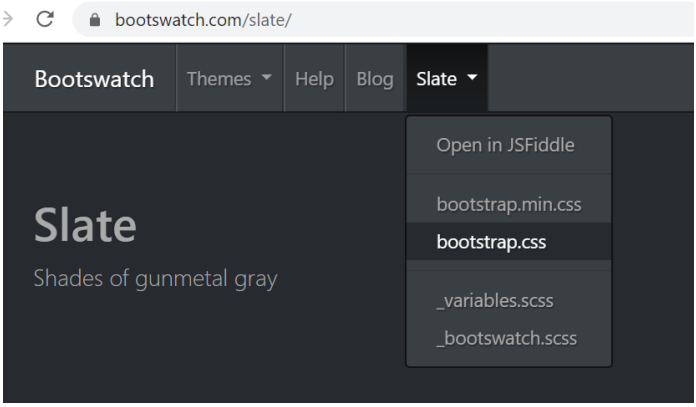


Рисунок 1.4.5 – Фрагмент вікна сайту

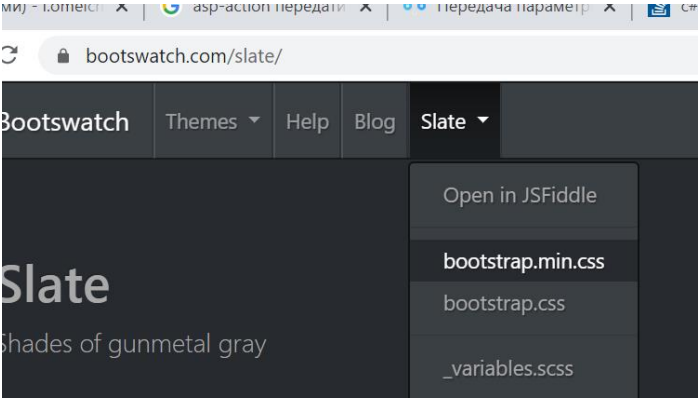


Рисунок 1.4.6 – Фрагмент вікна сайту

Для зручності переіменуйте їх (Рис. 1.4.7).

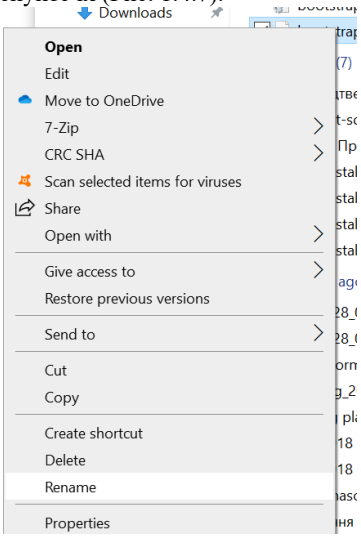


Рисунок 1.4.7 – Фрагмент вікна контекстного меню

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

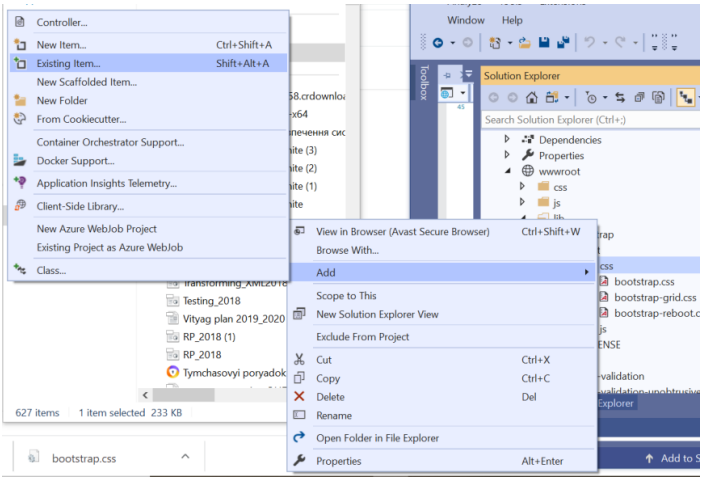


Рисунок 1.4.8 – Фрагмент вікна контекстного меню

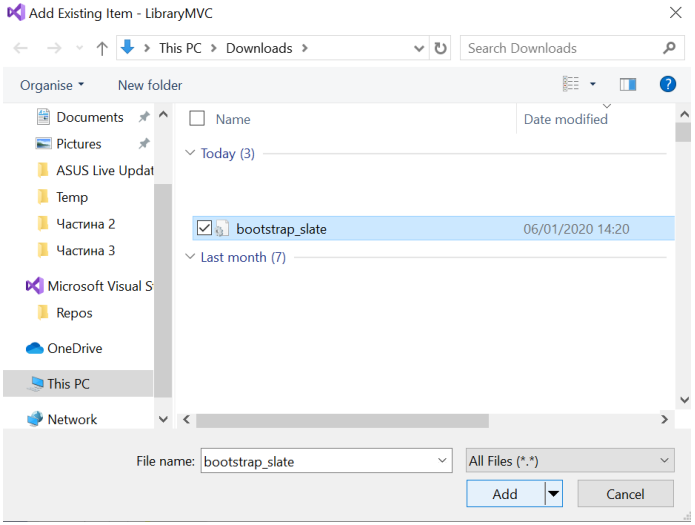


Рисунок 1.4.9 – Фрагмент вікна зміни імені

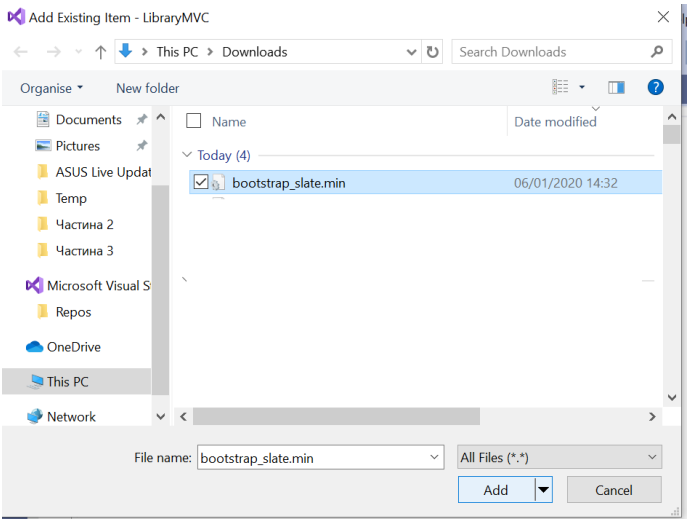


Рисунок 1.4.10 – Фрагмент вікна створення

Отримали:

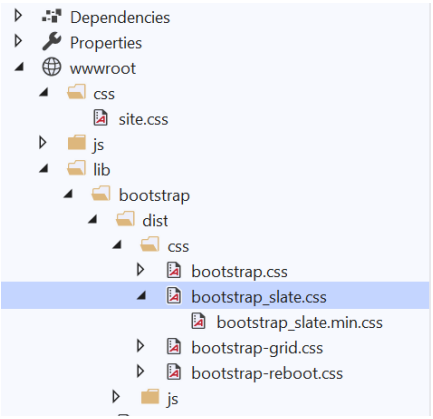


Рисунок 1.4.11 – Фрагмент вікна доступу до файлу

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

Рисунок 1.4.12– Фрагмент вікна зміни коду

Запускайте (Рис. 1.4.13).

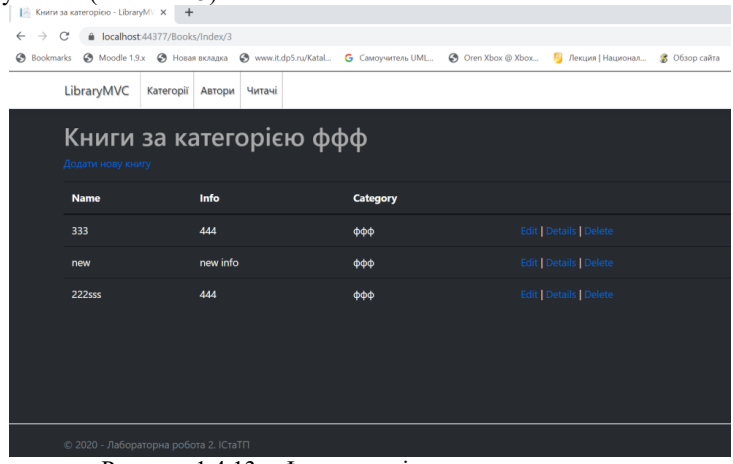


Рисунок 1.4.13 – Фрагмент вікна запуску програми

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

Тут буде використано бібліотеку призначену для створення діаграм - Google Charts (див. https://developers.google.com/chart).

Одним з найбільш поширених графічних завдань є побудова діаграм. Наприклад, на головній сторінці, яка відповідає за відображення категорій книжок нам потрібно додати діаграму, що показує співвідношення між кількостями книг в кожній категорії.

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).

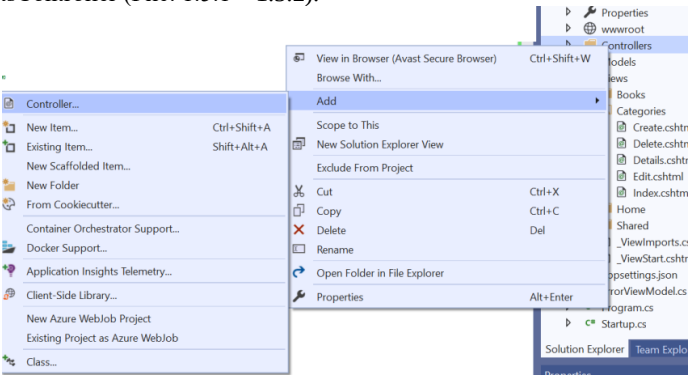
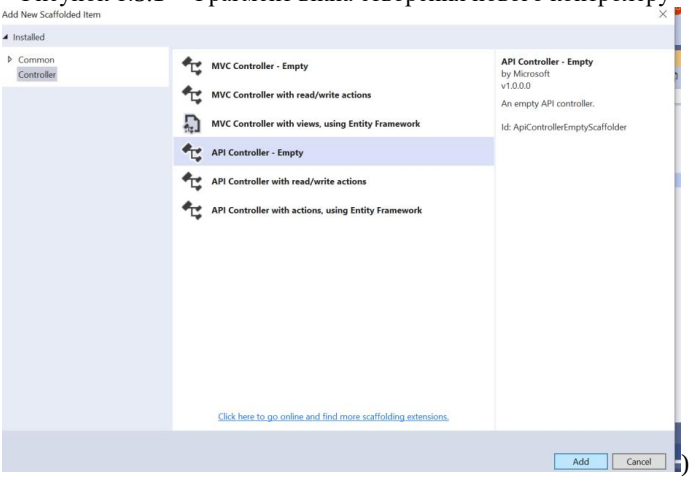


Рисунок 1.5.1 – Фрагмент вікна створення нового контролеру



)

Рисунок 1.5.2 – Фрагмент вікна створення контролеру

Визначимо в ньому метод, який повертає в якості JsonResult дані з назвами усіх категорій та кількостями книжок (Рис. 1.5.3).

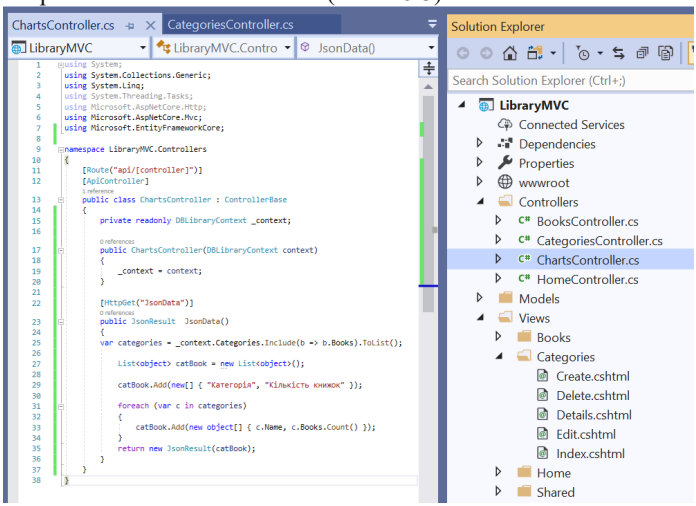


Рисунок 1.5.3 – Фрагмент вікна зміни методу

Тепер, залишилося викликати наш новий метод у відповідному представленні (в нашому випадку Index.cshtml папки Categories) та відобразити діаграму (Рис. 1.5.4 – 1.5.5).

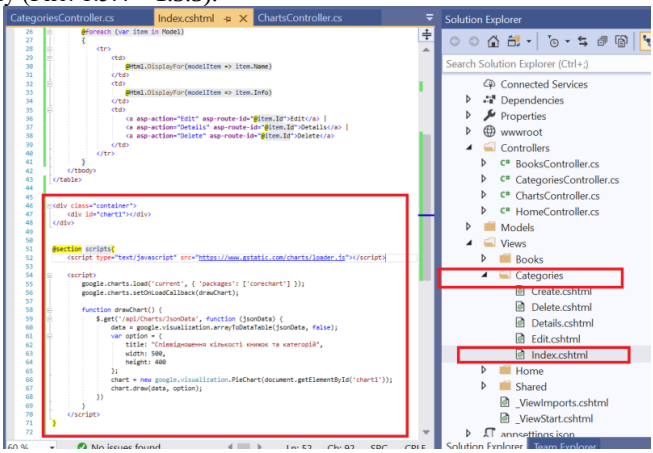


Рисунок 1.5.4 – Фрагмент вікна коду

Запускаємо

:

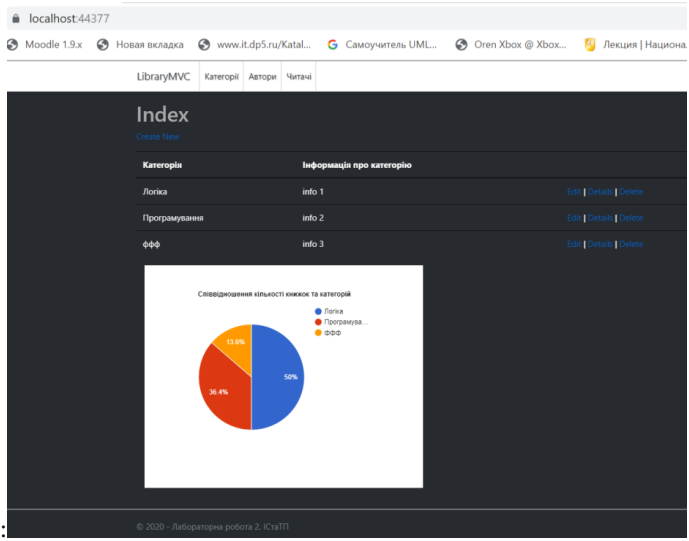


Рисунок 1.5.5 – Фрагмент вікна відображення побудованої діаграми

Крім PieChart, спробуйте ще LineChart, BarChart, ColumnChart та інші.

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

Постановка задачі

Задачею цього етапу буде одна з найбільш поширених проблем при розробці прикладного програмного забезпечення – формування та аналіз звітів.

Уявіть, що час від часу до вас надходить інформація у форматі Excel з даними для вашої програми (у нашому випадку в бібліотеку надходять нові книжки з описом та категорією і нам потрібно їх додати в БД). Крім того, ви повинні періодично формувати звіт аналогічного формату. Саме це і буде задачею даного етапу.

У випадку Вашої системи аналіз звітів може бути і більш інтелектуальним (наприклад, за описом книги підібрати категорію, або проаналізувати чи є у вас вже такі книги і збільшити кількість наявних книг, тощо – фантазуйте).

На сьогоднішній день існує досить багато готових рішень (бібліотек) для взаємодії з файлами Excel, щоб не працювати безпосередньо з системними GridView і DataTable. Розглянемо одну з таких бібліотек (ClosedXML) студент має право скористатися іншими бібліотеками для вирішення задачі цього етапу.

Нехай, у нас є наступний файл (назва листка - категорія книг, на листках назва книг, автори та інформація.) (Рис. 1.6.1)

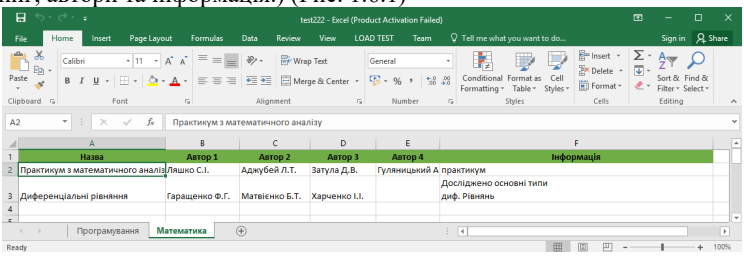


Рисунок 1.6.1 – Фрагмент вікна програми

Аналіз файлу

«Розпарсимо» цей файл. У випадку відсутності категорії та авторів додамо їх в БД, а у випадку їх наявності – прив’яжемо книги до існуючої інформації.

Відкриваємо Nuget Manager і додаємо пакет ClosedXML в проєкт (Рис. 1.6.2. – 1.6.4).

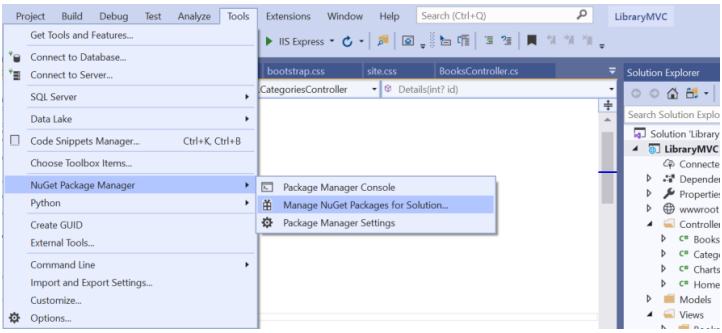


Рисунок 1.6.2 – Фрагмент вікна пакету ClosedXML

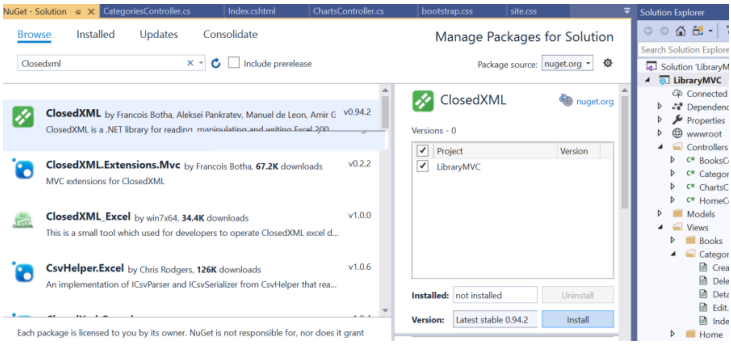


Рисунок 1.6.3 – Фрагмент вікна пакету ClosedXML

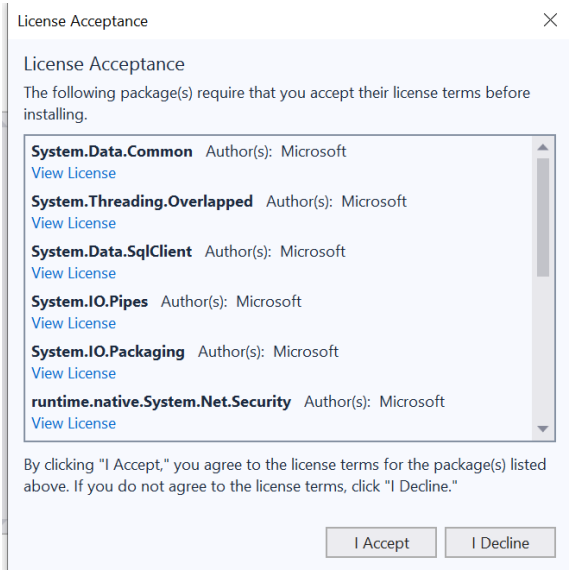


Рисунок 1.6.4 – Фрагмент вікна роботи з пакетом ClosedXML

Нехай, завантаження Excel-файлу буде здійснюватися на сторінці з категоріями книг. Тоді, у відповідному контролері (Categories) та у відповідному представленні (Index.cshtml з папки Categories) потрібно внести зміни (Рис.1.6.5 – 1.6.7).

```
        <div>
    <h3>Оберіть exel-файл для заватаження</h3>
    @using (Html.BeginForm("Import", "Categories", FormMethod.Post,
new { enctype = "multipart/form-data", id = "frm-excel" }))
    {
        <div>
            Завантажте Excel-файл:
            <input type="file" name="fileExcel" id="fileExcel" />
            <div>
                <input type="submit" value="Завантажити" />
            </div>
        </div>
    }
</div>
```

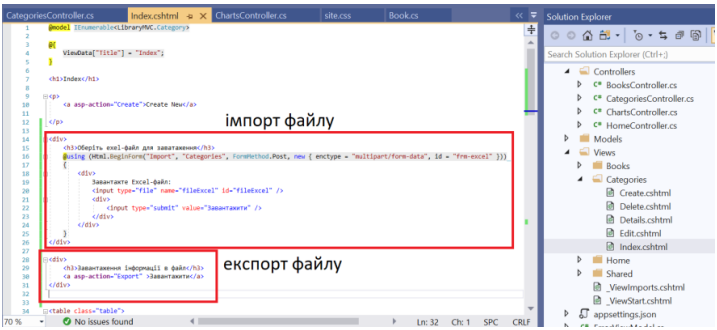


Рисунок 1.6.5 – Фрагмент вікна внесення змін у код

Ми звертаємося до методу Import в контролері Categories. Створимо цей метод:

```
    [HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Import(IFormFile fileExcel)
{
    if (ModelState.IsValid)
    {
        if (fileExcel != null)
        {
            using (var stream = new FileStream(fileExcel.FileName,
FileMode.Create))
            {
                await fileExcel.CopyToAsync(stream);
                using (XLWorkbook workBook = new XLWorkbook(stream,
XLEventTracking.Disabled))
                {
                    //перегляд усіх листів (в даному випадку категорій)
                    foreach (IXLWorksheet worksheet in workBook.Worksheets)
                    {
                        //worksheet.Name - назва категорії. Пробуємо знайти в
БД, якщо відсутня, то створюємо нову
                        Category newcat;
                        var c = (from cat in _context.Categories
                                 where cat.Name.Contains(worksheet.Name)
                                 select cat).ToList();
                        if (c.Count > 0)
                        {
                            newcat = c[0];
                        }
```

```
                    else
                    {
                        newcat = new Category();
                        newcat.Name = worksheet.Name;
                        newcat.Info = "from EXCEL";
                        //додати в контекст
                        _context.Categories.Add(newcat);
                    }
                    //перегляд усіх рядків
                    foreach (IXLRow row in worksheet.RowsUsed().Skip(1))
                    {
                        try
                        {
                            Book book = new Book();
                            book.Name = row.Cell(1).Value.ToString();
                            book.Info = row.Cell(6).Value.ToString();
                            book.Category = newcat;
                            _context.Books.Add(book);
                            //у разі наявності автора знайти його, у разі
відсутності - додати
                            for (int i = 2; i <= 5; i++)
                            {
                                if (row.Cell(i).Value.ToString().Length > 0)
                                {
                                    Author author;

                                    var a = (from aut in _context.Authors
                                             where
aut.Name.Contains(row.Cell(i).Value.ToString())
                                             select aut).ToList();
                                    if (a.Count > 0)
                                    {
                                        author = a[0];
                                    }
                                    else
                                    {
                                        author = new Author();
                                        author.Name = row.Cell(i).Value.ToString();
                                        author.Info = "from EXCEL";
                                        //додати в контекст
                                        _context.Add(author);
                                    }
                                    AuthorsBooks ab = new AuthorsBooks();
                                    ab.Book = book;
                                    ab.Author = author;
```

```
                        _context.AuthorsBooks.Add(ab);
                    }
                }
            }
            catch (Exception e)
            {
                //logging самостійно :)

            }
        }
    }
}
}

await _context.SaveChangesAsync();
}
return RedirectToAction(nameof(Index));
}
```

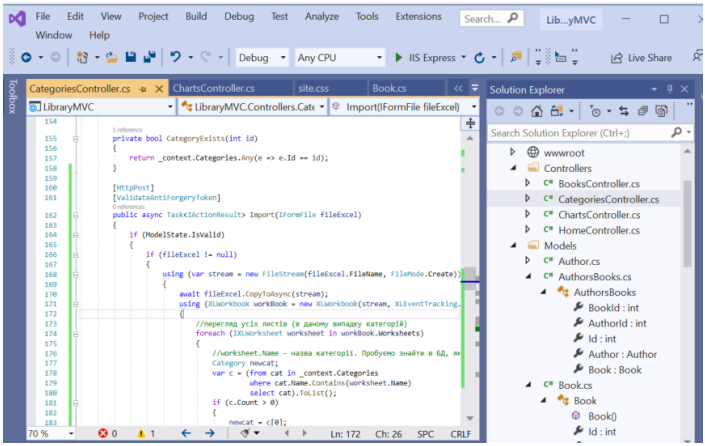


Рисунок 1.6.6 – Фрагмент вікна коду

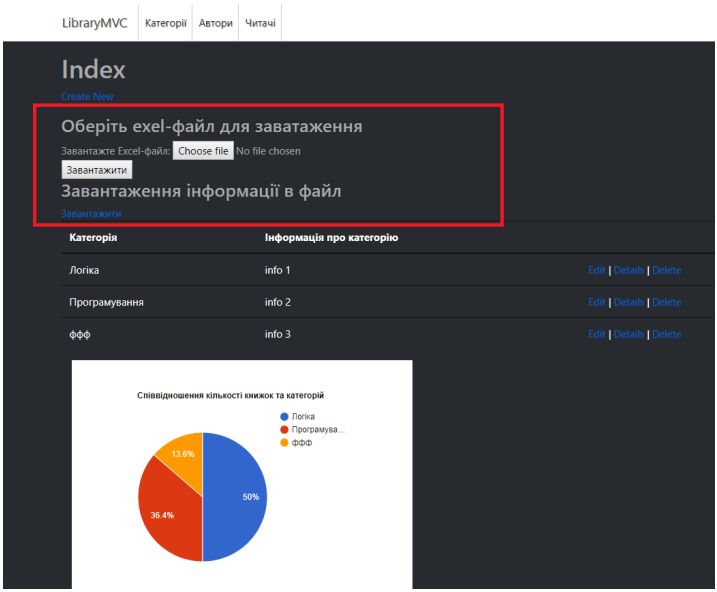


Рисунок 1.6.7 – Фрагмент вікна представлення діаграми

Генерація файлу

Також можлива інша ситуація, коли у нас немає вихідного Excel-файлу, замість цього вебдодаток повинен сформувати його динамічно, і користувач сайту зможе його завантажити.

Нехай, генерація Excel-файлу буде здійснюватися на сторінці з категоріями книг. Тоді, у відповідному контролері (Categories) та у відповідному представленні (Index.cshtml з папки Categories) потрібно внести зміни (Рис.1.6.8 – 1.6.11).

```
<div>
    <h3>Завантаження інформації в файл</h3>
    <a asp-action="Export" >Завантажити</a>
</div>
```

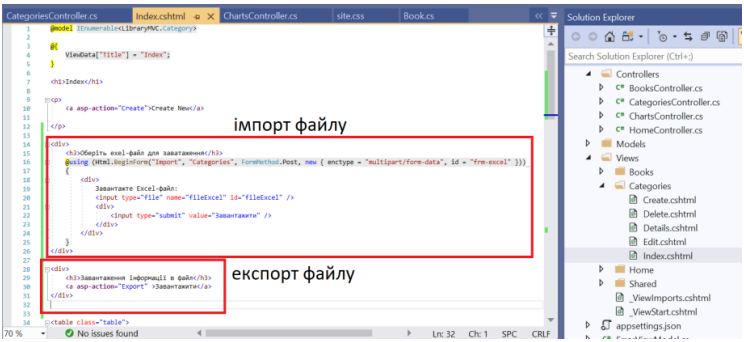


Рисунок 1.6.8 – Фрагмент вікна коду

Ми звертаємося до методу Export в контролері Categories. Створимо цей метод:

```
public ActionResult Export()
{
    using (XLWorkbook workbook = new XLWorkbook(XLEventTracking.Disabled))
    {
        var categories = _context.Categories.Include("Books").ToList();
        //тут, для прикладу ми пишемо усі книжки з БД, в своїх проєктах ТАК НЕ РОБИТИ (писати лише вибрані)
        foreach (var c in categories)
        {
            var worksheet = workbook.Worksheets.Add(c.Name);

            worksheet.Cell("A1").Value = "Назва";
            worksheet.Cell("B1").Value = "Автор 1";
            worksheet.Cell("C1").Value = "Автор 2";
            worksheet.Cell("D1").Value = "Автор 3";
            worksheet.Cell("E1").Value = "Автор 4";
            worksheet.Cell("F1").Value = "Категорія";
            worksheet.Cell("G1").Value = "Інформація";
            worksheet.Row(1).Style.Font.Bold = true;
            var books = c.Books.ToList();

            //нумерація рядків/стовпчиків починається з індекса 1 (не 0)
            for (int i = 0; i < books.Count; i++)
            {
                worksheet.Cell(i + 2, 1).Value = books[i].Name;
                worksheet.Cell(i + 2, 7).Value = books[i].Info;
```

```
                var ab = _context.AuthorsBooks.Where(a => a.BookId ==
books[i].Id).Include("Author").ToList();
                //більше 4-ох нікуди писати
                int j = 0;
                foreach (var a in ab)
                {
                    if (j < 5)
                    {
                        worksheet.Cell(i + 2, j + 2).Value = a.Author.Name;
                        j++;
                    }
                }

            }
        }
            using (var stream = new MemoryStream())
            {
                workbook.SaveAs(stream);
                stream.Flush();

                return new FileContentResult(stream.ToArray(),
                    "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet")
                {
                    FileDownloadName =
$"library_{DateTime.UtcNow.ToShortDateString()}.xlsx"
                };
            }
        }
    }
```

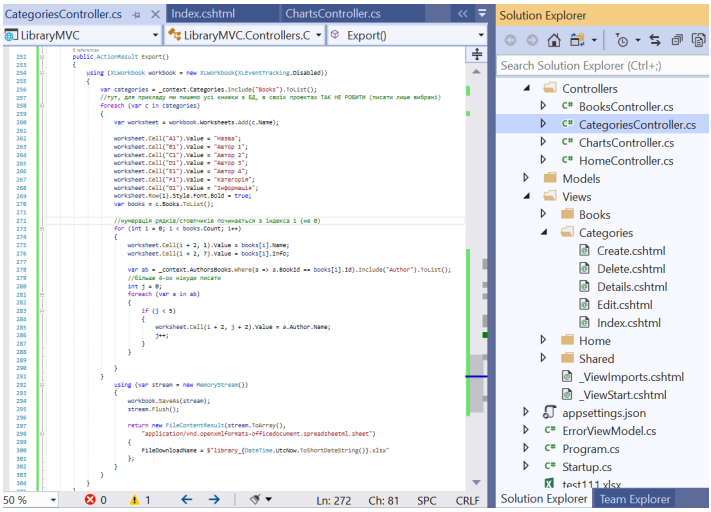


Рисунок 1.6.9 – Фрагмент вікна внесення змін до коду

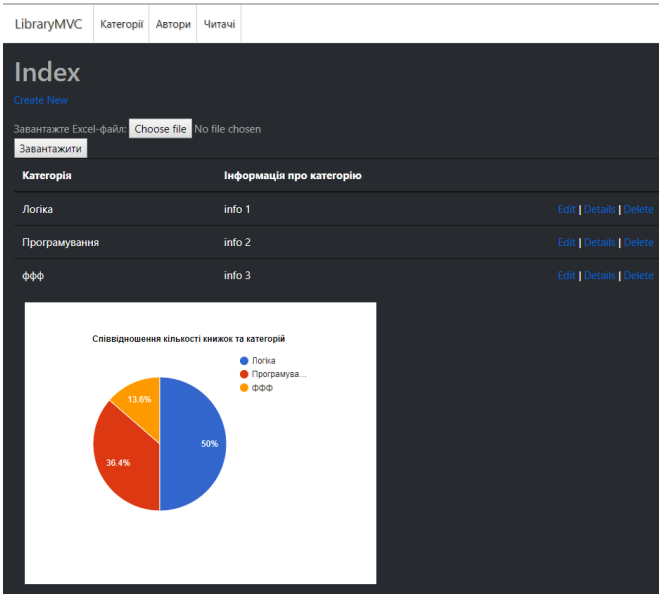


Рисунок 1.6.10 – Фрагмент вікна відображення діаграм

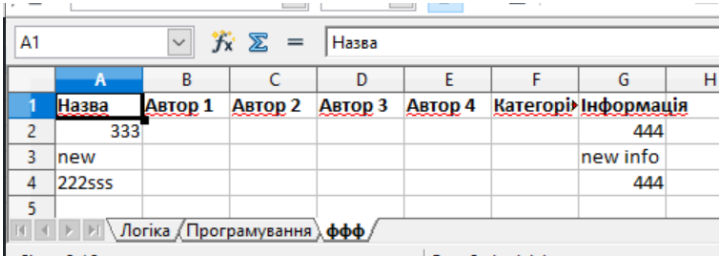


Рисунок 1.6.11 – Фрагмент вікна EXCEL-файлу

Для отримання балів за етап – реалізуйте в своєму проєкті аналіз та генерацію EXCEL-файлу з інформацією вашої предметної області. Складність файлу залежить він предметної області, але не нижча, ніж в запропонованому прикладі (бажано вища).

КРОКИ ВИКОНАННЯ ЕТАПУ 1.7

Додавання Identity в проєкт

ASP.NET Identity представляє вбудовану в ASP.NET систему аутентифікації і авторизації. Дана система дозволяє користувачам створювати облікові записи, аутентифицироваться, управляти обліковими записами або використовувати для входу на сайт облікові записи зовнішніх провайдерів, таких як Facebook, Google, Microsoft, Twitter та інших.

Додамо в папку Models новий клас User (Рис.1.7.1 -1.7.3).

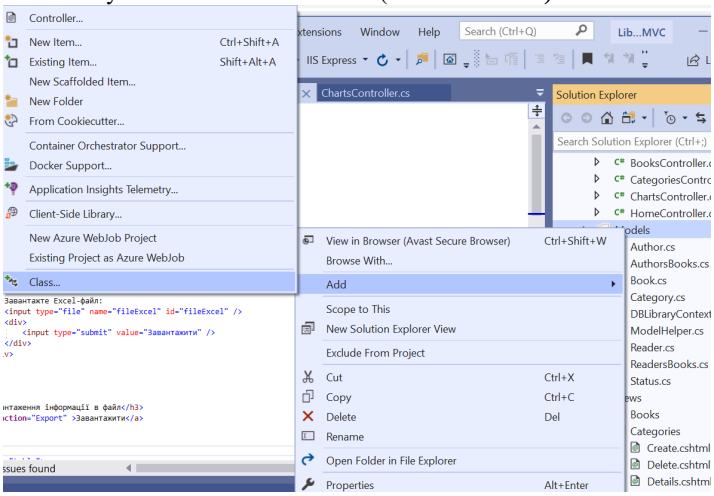


Рисунок 1.7.1 – Фрагмент вікна додавання Identity в проєкт

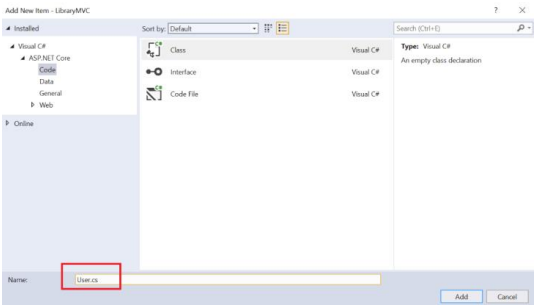


Рисунок 1.7.2 – Фрагмент вікна додавання Identity в проєкт

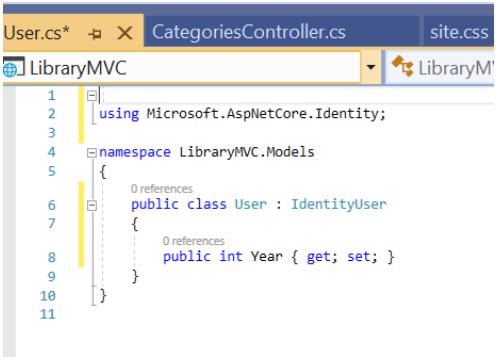


Рисунок 1.7.3 – Фрагмент вікна коду

```
using Microsoft.AspNetCore.Identity;

namespace LibraryMVC.Models
{
    public class User : IdentityUser
    {
        public int Year { get; set; }
    }
}
```

Клас User представляє користувача і успадковується від класу IdentityUser, переймаючи все його властивості. Крім того, тут додано властивість Year, яке буде представляти рік народження користувача. При бажанні можна визначити будь-які інші властивості.

Для взаємодії з MS SQL Server через ASP.NET Core Identity додамо в проєкт через Nuget пакети Microsoft.AspNetCore.Identity.EntityFrameworkCore (крім того, має бути встановлений і Microsoft.EntityFrameworkCore.SqlServer, але він у нас вже є).

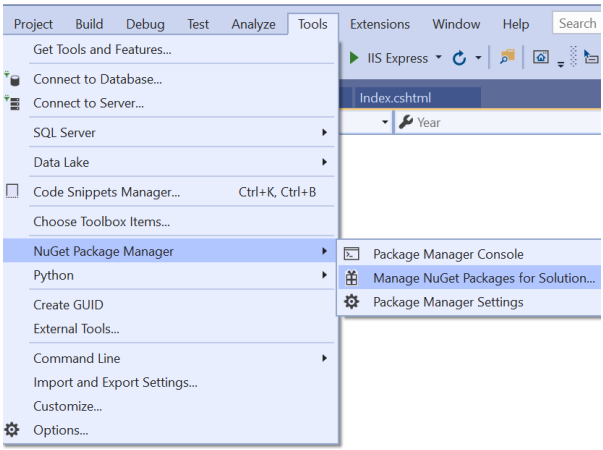


Рисунок 1.7.4 – Фрагмент вікна роботи з пакетом Microsoft.AspNetCore.Identity.EntityFrameworkCore

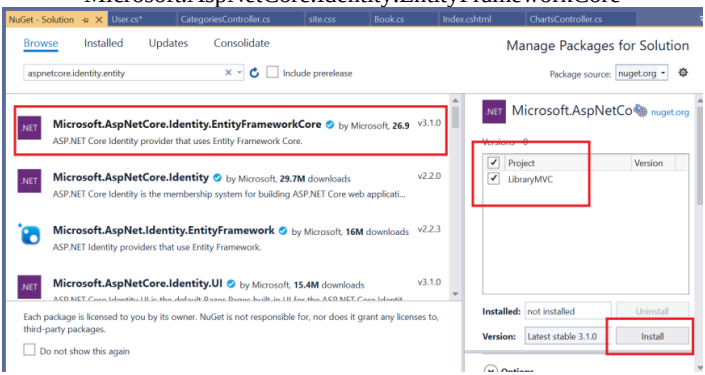


Рисунок 1.7.5 – Фрагмент вікна роботи з пакетом Microsoft.AspNetCore.Identity.EntityFrameworkCore

Так як у нас нова БД, то додамо новий клас контексту даних IdentityContext. Крім того, так як ми використовуємо Identity, то клас контексту даних буде успадковуватися не від DbContext, а від IdentityDbContext (Рис. 1.7.6 – 1.7.8).

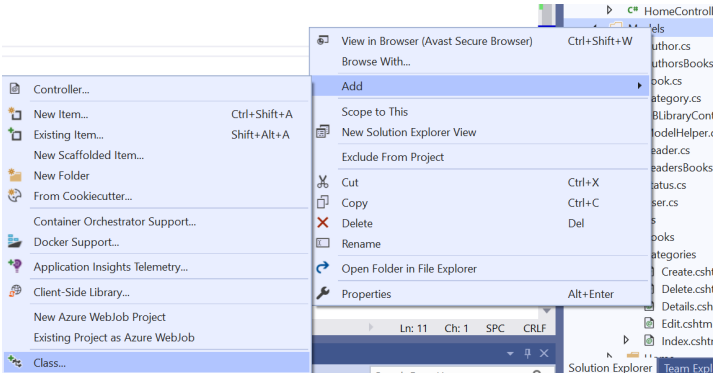


Рисунок 1.7.6 – Фрагмент вікна роботи з класом контексту даних IdentityContext

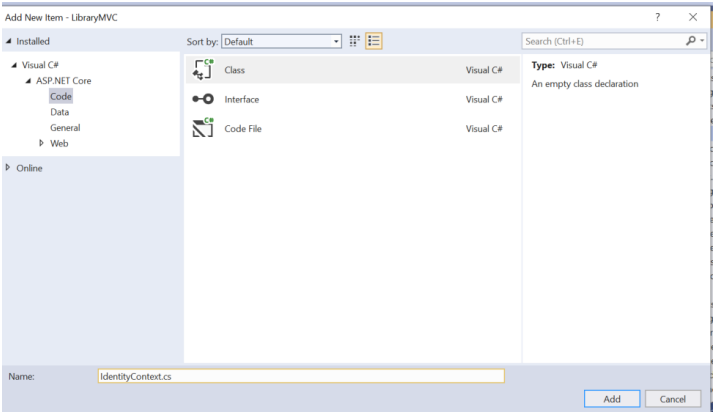


Рисунок 1.7.7 – Фрагмент вікна роботи з класом контексту даних IdentityContext

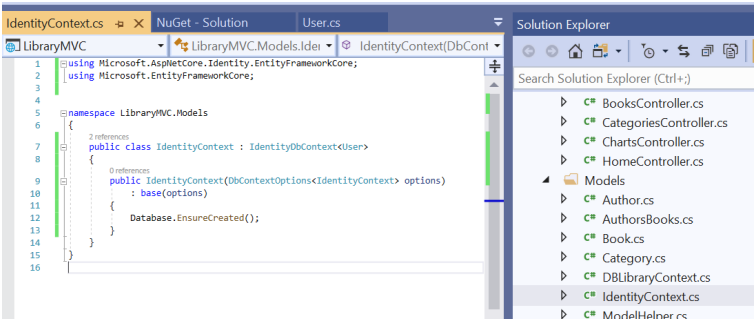


Рисунок 1.7.8 – Фрагмент вікна роботи з класом контексту даних IdentityContext

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace LibraryMVC.Models
{
    public class IdentityContext : IdentityDbContext<User>
    {
        public IdentityContext(DbContextOptions<IdentityContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }
    }
}
```

У нас є контекст і моделі, і тепер нам необхідна база даних, яка буде зберігати всі дані. Спочатку визначимо в файлі appsettings.json рядок підключення (Рис. 1.7.9).

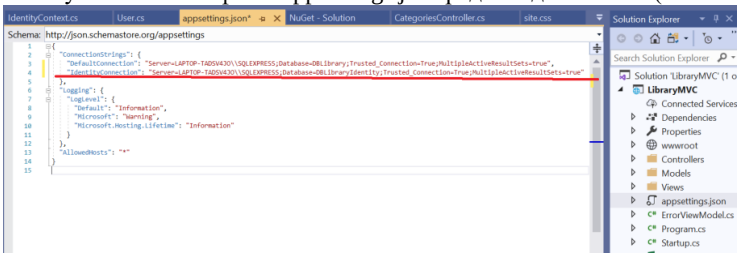


Рисунок 1.7.9 – Фрагмент вікна коду

"IdentityConnection": "Server=*LAPTOP-TADSV4JO\\SQLEXPRESS*; Database=*DBLibraryIdentity*; Trusted_Connection=True; MultipleActiveResultSets=true"

Зауваження: *параметри серверу беремо з рядку вище (див. етап 1).*

Внесемо зміни в клас Stuatup (Рис.1.7.10).

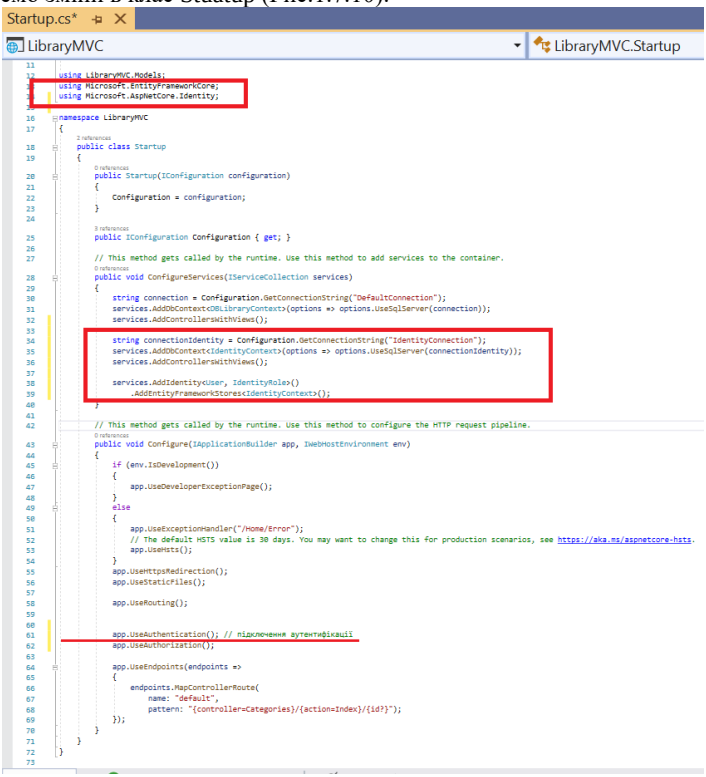


Рисунок 1.7.10 – Фрагмент вікна роботи з класом Stuatup

Зверніть увагу на змінені методи:

public void ConfigureServices(IServiceCollection services)
{
string connection = Configuration.GetConnectionString("DefaultConnection");
services.AddDbContext<DBLibraryContext>(options => options.UseSqlServer(connection));

```
            services.AddControllersWithViews();

            string connectionIdentity =
Configuration.GetConnectionString("IdentityConnection");
            services.AddDbContext<IdentityContext>(options =>
options.UseSqlServer(connectionIdentity));
            services.AddControllersWithViews();


            services.AddIdentity<User,
IdentityRole>()
                .AddEntityFrameworkStores<IdentityContext>();
        }


        public void Configure(IApplicationBuilder app, IWebHostEnvironment
env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                app.UseHsts();
            }
            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();


            app.UseAuthentication(); // підключення аутентифікації
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Categories}/{action=Index}/{id?}");
            });
        }
```

Крім того, додано:

using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Identity;

Метод AddIdentity () дозволяє встановити деяку початкову конфігурацію. Тут ми вказуємо тип користувача і тип ролі, які будуть використовуватися системою Identity. В якості типу користувача виступає створений нами вище клас User, а в якості типу ролі узятий стандартний клас IdentityRole, який знаходиться в просторі імен Microsoft.AspNetCore.Identity.EntityFrameworkCore.

Метод AddEntityFrameworkStores () встановлює тип сховища, яке буде застосовуватися в Identity для зберігання даних. В якості типу сховища тут вказується клас контексту даних.

Потім, щоб використовувати Identity, в методі Configure () встановлюється компонент middeware - UseAuthentication. Причому це middleware викликається перед app.UseEndpoints (), тим самим гарантуючи, що на час звернення до системи маршрутизації, контролерам і їх методам, куки належним чином оброблені і встановлені.

Тепер система Identity підключена до проєкту, і ми можемо з нею працювати.

Реєстрація та створення користувачів

Для роботи з обліковими записами користувачів додамо в папку Controllers новий контролер AccountController і визначимо в ньому метод для реєстрації користувачів:

Оскільки в класі Startup були додані сервіси Identity, то тут в контролері через конструктор ми можемо їх отримати.

За допомогою методу _userManager.CreateAsync користувач додається в базу даних. Як параметр передається сам користувач і його пароль.

Даний метод повертає об'єкт IdentityResult, за допомогою якого можна дізнатися успішність виконаної операції. Цілком можливо, що передані значення не задовольняють вимогам, і тоді користувача не буде додано до бази даних. У разі вдалого додавання за допомогою методу _signInManager.SignInAsync () встановлюємо аутентифікаційні куки для доданого користувача. У цей метод передається об'єкт користувача, що ідентифікує, і логічне значення, яке вказує, чи треба зберігати куки протягом тривалого часу. І далі виконуємо переадресацію на головну сторінку програми.

Якщо додавання пройшло невдало, то додаємо до стана моделі за допомогою методу ModelState всі виниклі при додаванні помилки, і відправлена модель повертається в представлення.

Створимо в проєкті нову папку ViewModels (Рис. 1.7.11-1.7.14).

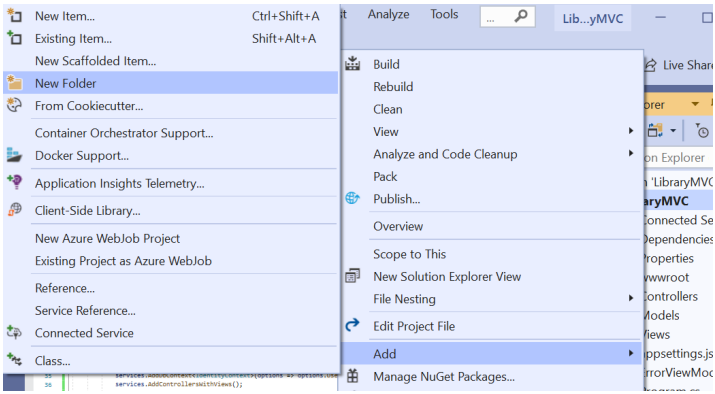


Рисунок 1.7.11 – Фрагмент вікна створення нової папки ViewModels

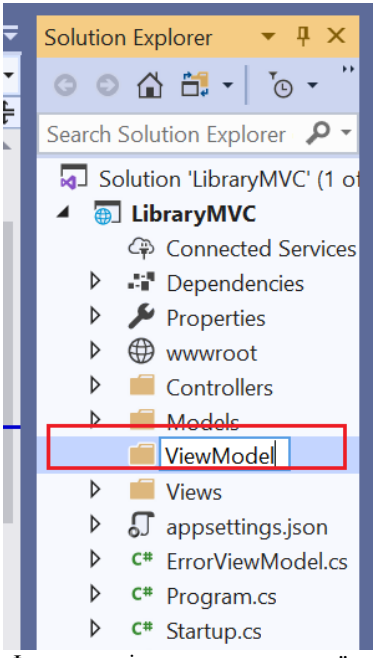


Рисунок 1.7.12 – Фрагмент вікна створення нової папки ViewModels

В яку додамо клас RegisterViewModel.cs:

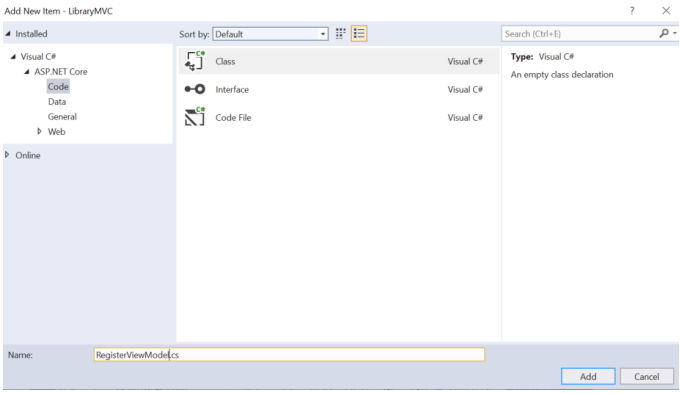


Рисунок 1.7.13 – Фрагмент вікна роботи з ViewModels

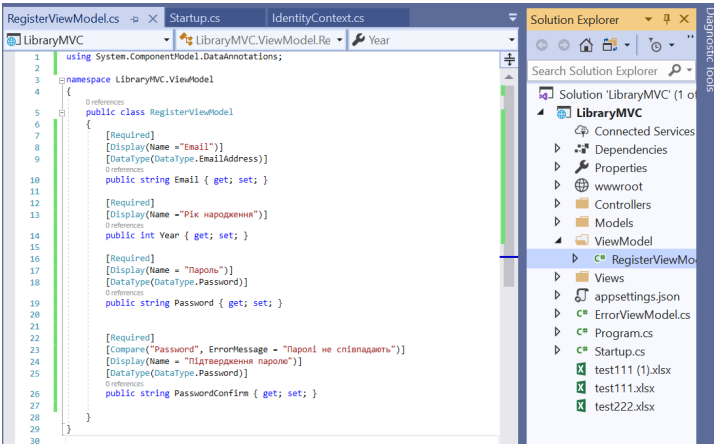


Рисунок 1.7.14 – Фрагмент вікна роботи з ViewModels

Для роботи з обліковими записами користувачів додамо в папку Controllers новий контролер AccountController і визначимо в ньому метод для реєстрації користувачів (Рис. 1.7.15 – 1.7.18).

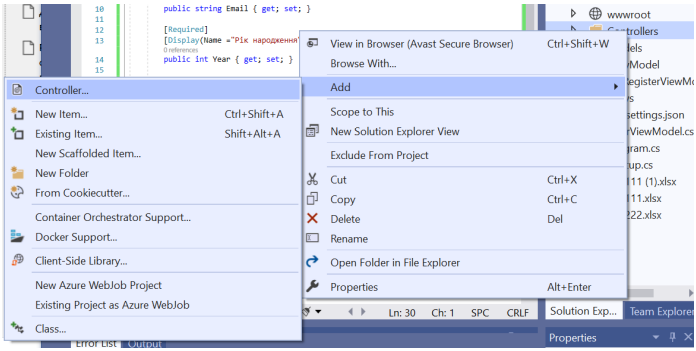


Рисунок 1.7.15 – Фрагмент вікна роботи з папкою Controllers

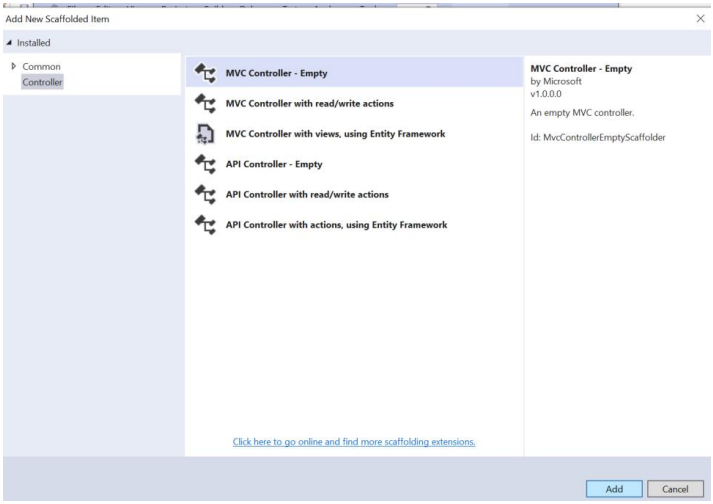


Рисунок 1.7.16 – Фрагмент вікна роботи з папкою Controllers

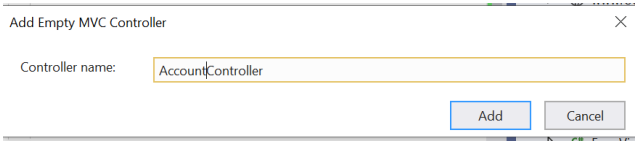


Рисунок 1.7.17 – Фрагмент вікна роботи з папкою Controllers

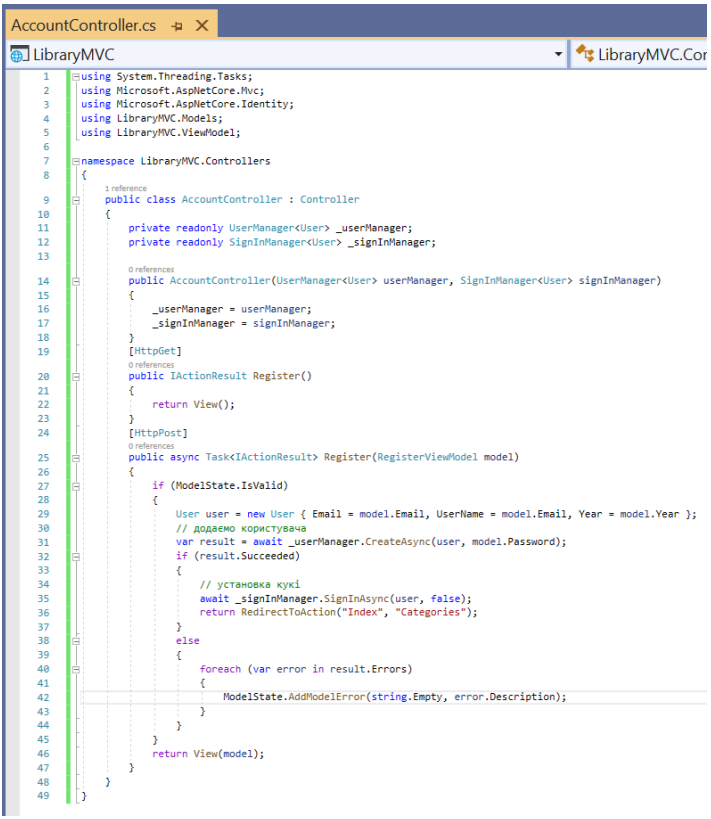


Рисунок 1.7.18 – Фрагмент вікна коду роботи з папкою Controllers

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Identity;
using LibraryMVC.Models;
using LibraryMVC.ViewModel;

namespace LibraryMVC.Controllers
{
    public class AccountController : Controller
    {
        private readonly UserManager<User> _userManager;
        private readonly SignInManager<User> _signInManager;
```

```
        public AccountController(UserManager<User> userManager,
SignInManager<User> signInManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
        }
        [HttpGet]
        public IActionResult Register()
        {
            return View();
        }
        [HttpPost]
        public async Task<IActionResult> Register(RegisterViewModel model)
        {
            if (ModelState.IsValid)
            {
                User user = new User { Email = model.Email, UserName =
model.Email, Year = model.Year };
                // додаємо користувача
                var result = await _userManager.CreateAsync(user, model.Password);
                if (result.Succeeded)
                {
                    // установка кукі
                    await _signInManager.SignInAsync(user, false);
                    return RedirectToAction("Index", "Categories");
                }
                else
                {
                    foreach (var error in result.Errors)
                    {
                        ModelState.AddModelError(string.Empty, error.Description);
                    }
                }
            }
            return View(model);
        }
    }
}
```

Для представлень цього контролера в каталозі Views визначимо підкаталог Account, в який додамо нове представлення Register.cshtml (Рис.1.7.19 – 1.7.22).

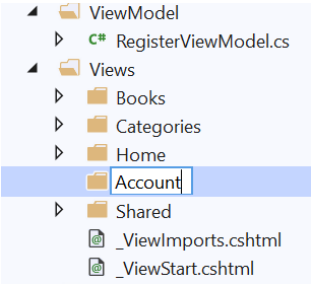


Рисунок 1.7.19 – Фрагмент вікна папки

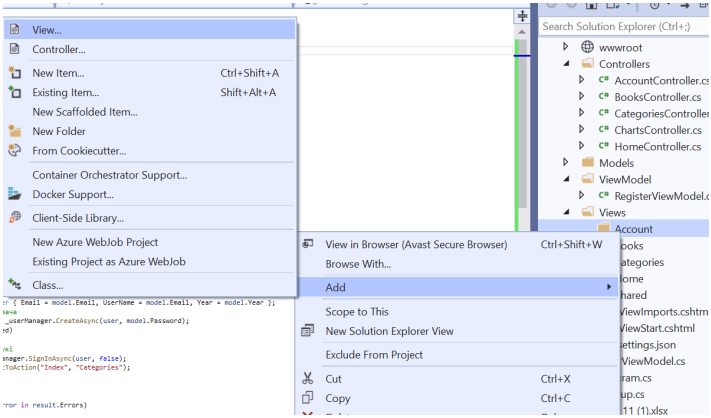


Рисунок 1.7.20 – Фрагмент створення Register.cshtml

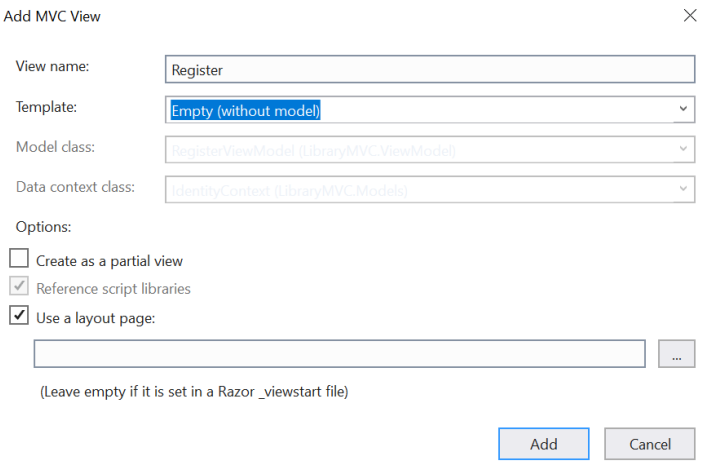


Рисунок 1.7.21 – Фрагмент створення Register.cshtml

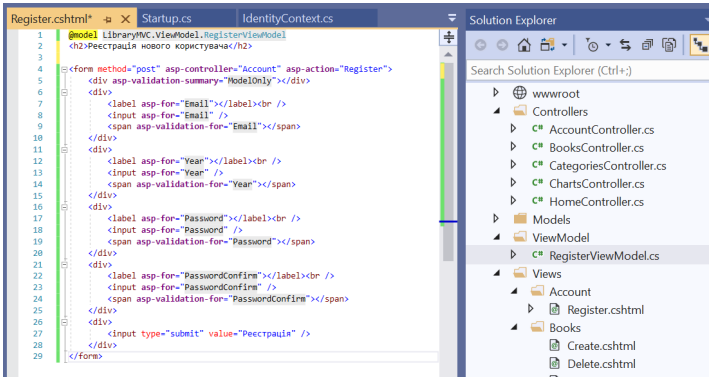


Рисунок 1.7.22 – Фрагмент вікна коду Register.cshtml

```
@model LibraryMVC.ViewModel.RegisterViewModel
<h2>Реєстрація нового користувача</h2>
<form method="post" asp-controller="Account" asp-action="Register">
    <div asp-validation-summary="ModelOnly"></div>
    <div>
        <label asp-for="Email"></label><br />
        <input asp-for="Email" />
        <span asp-validation-for="Email"></span>
```

```
    </div>
    <div>
        <label asp-for="Year"></label><br />
        <input asp-for="Year" />
        <span asp-validation-for="Year"></span>
    </div>
    <div>
        <label asp-for="Password"></label><br />
        <input asp-for="Password" />
        <span asp-validation-for="Password"></span>
    </div>
    <div>
        <label asp-for="PasswordConfirm"></label><br />
        <input asp-for="PasswordConfirm" />
        <span asp-validation-for="PasswordConfirm"></span>
    </div>
    <div>
        <input type="submit" value="Реєстрація" />
    </div>
</form>
```

Запускаємо (Рис.1.7.23).

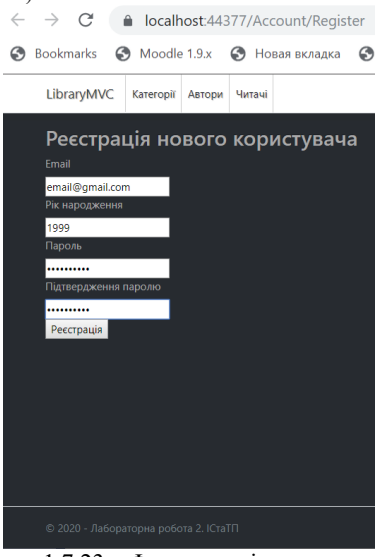


Рисунок 1.7.23 – Фрагмент вікна користувача

Зверніть увагу, що по замовчуванню до паролю висувається дуже багато вимог. Тепер можна перевірити нову базу даних (Рис.1.7.24)

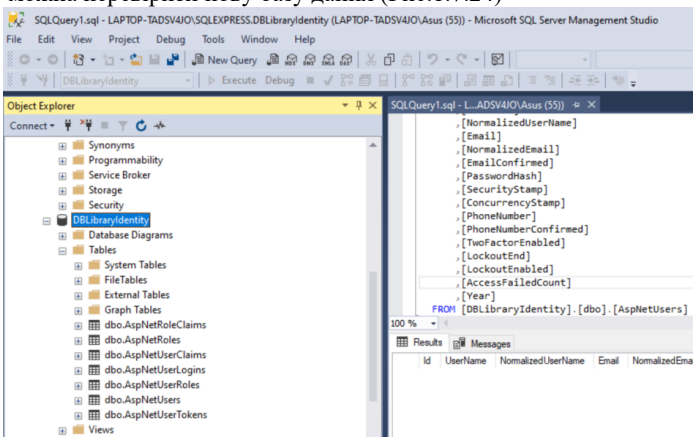


Рисунок 1.7.24 – Фрагмент вікна бази даних

Авторизація користувачів

Тепер додамо функціонал авторизації зареєстрованих користувачів.

Для створення механізму авторизації користувачів в додатку спочатку додамо в папку ViewModels модель LoginViewModel (Рис. 1.7.25 – 1.7.27).

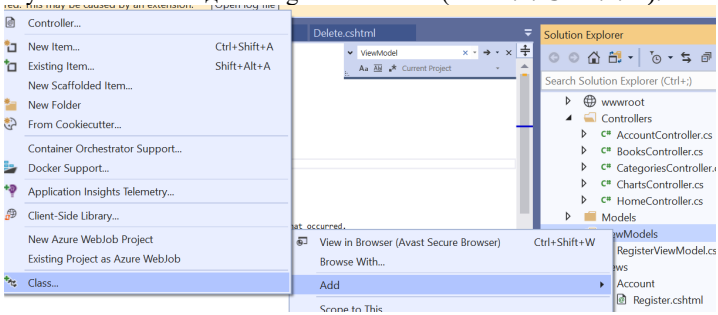


Рисунок 1.7.25 – Фрагмент вікна роботи з моделлю LoginViewModel

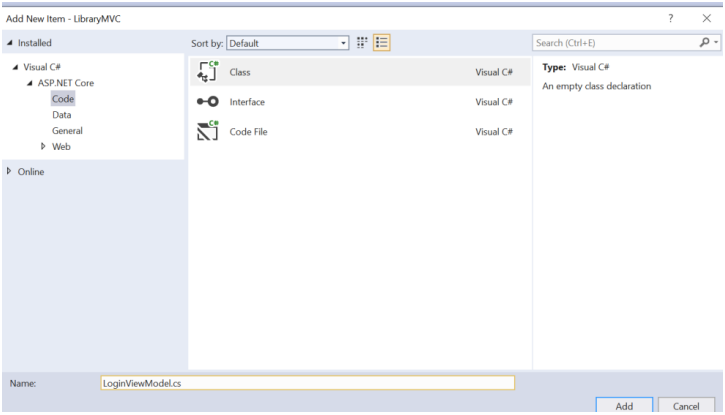


Рисунок 1.7.26 – Фрагмент вікна роботи з моделлю LoginViewModel

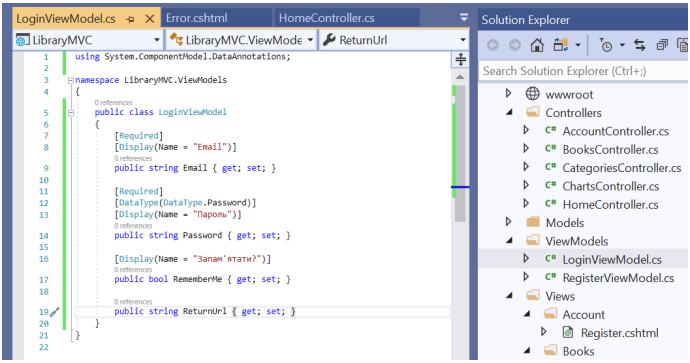


Рисунок 1.7.27 – Фрагмент вікна роботи з моделлю LoginViewModel

```
using System.ComponentModel.DataAnnotations;

namespace LibraryMVC.ViewModels
{
    public class LoginViewModel
    {
        [Required]
        [Display(Name = "Email")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
```

```
        [Display(Name = "Пароль")]
        public string Password { get; set; }

        [Display(Name = "Запам'ятати?")]
        public bool RememberMe { get; set; }

        public string ReturnUrl { get; set; }
    }
}
```

Додамо в AccountController три методи (Рис.1.7.28).

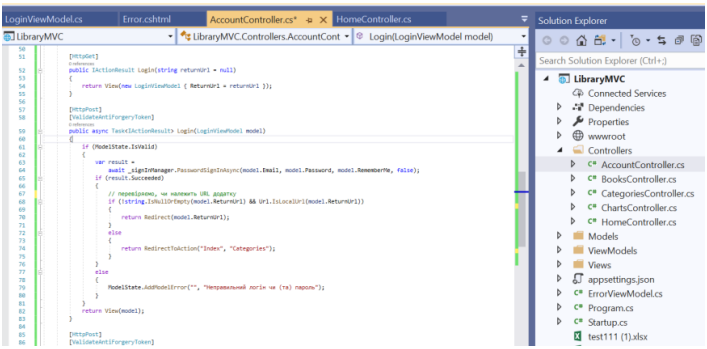


Рисунок 1.7.28 – Фрагмент вікна роботи з AccountController

```
 [HttpGet]
        public IActionResult Login(string returnUrl = null)
        {
            return View(new LoginViewModel { ReturnUrl = returnUrl });
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Login(LoginViewModel model)
        {
            if (ModelState.IsValid)
            {
                var result =
                    await _signInManager.PasswordSignInAsync(model.Email,
model.Password, model.RememberMe, false);
                if (result.Succeeded)
                {
                    // перевіряємо, чи належить URL додатку
                    if (!string.IsNullOrEmpty(model.ReturnUrl) &&
```

```
Url.IsLocalUrl(model.ReturnUrl))
                {
                    return Redirect(model.ReturnUrl);
                }
                else
                {
                    return RedirectToAction("Index", "Categories");
                }
            }
            else
            {
                ModelState.AddModelError("", "Неправильний логін чи (та)
пароль");
            }
        }
        return View(model);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Logout()
    {
        // видаляємо аутентифікаційні куки
        await _signInManager.SignOutAsync();
        return RedirectToAction("Index", "Categories");
    }
```

У Get-версії методу Login ми отримуємо адресу для повернення у вигляді параметра returnUrl і передаємо його в модель LoginViewModel.

В Post-версії методу Login отримуємо дані з представлення у вигляді моделі LoginViewModel. Всю роботу з аутентифікації користувача виконує метод signInManager.PasswordSignInAsync(). Цей метод приймає логін і пароль користувача. Третій параметр методу вказує, чи треба зберігати кукі на довгий час.

Даний метод також повертає IdentityResult, за допомогою якого можна дізнатися, чи завершилася аутентифікація успішно. Якщо вона завершилося успішно, то використовуємо властивість ReturnUrl моделі LoginViewModel для повернення користувача на попереднє місце. Для цього потрібно ще упевнитися, що адреса повернення належить додатком за допомогою методу Url.IsLocalUrl (). Це дозволить уникнути перенаправлень на небажані сайти. Якщо ж адреса повернення не встановлений або не належить додатком, виконуємо переадресацію на головну сторінку.

Третій метод - метод Logout виконує вихід користувача з програми. За вихід відповідає метод _signInManager.SignOutAsync(), який очищає аутентифікаційні куки.

Визначимо в проєкті в папці Views/Account представлення Login.cshtml, через яке буде здійснюватися вхід в додаток (Рис. 17.29 – 1.7.31).

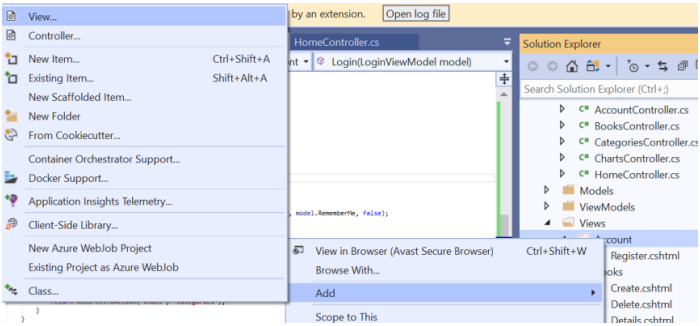


Рисунок 1.7.29 – Фрагмент вікна представлення Login.cshtml

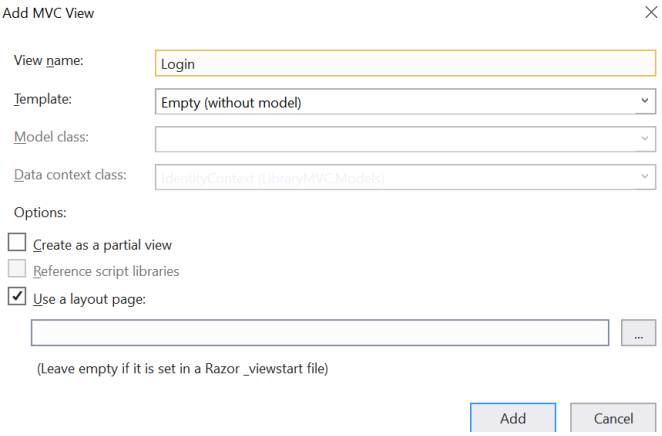


Рисунок 1.7.30 – Фрагмент вікна створення Login.cshtml

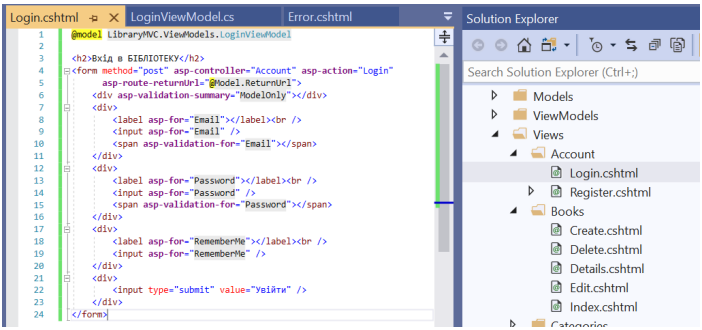


Рисунок 1.7.31 – Фрагмент вікна коду роботи з Login.cshtml

```
@model LibraryMVC.ViewModels.LoginViewModel

<h2>Вхід в БІБЛІОТЕКУ</h2>
<form method="post" asp-controller="Account" asp-action="Login"
      asp-route-returnUrl="@Model.ReturnUrl">
    <div asp-validation-summary="ModelOnly"></div>
    <div>
        <label asp-for="Email"></label><br />
        <input asp-for="Email" />
        <span asp-validation-for="Email"></span>
    </div>
    <div>
        <label asp-for="Password"></label><br />
        <input asp-for="Password" />
        <span asp-validation-for="Password"></span>
    </div>
    <div>
        <label asp-for="RememberMe"></label><br />
        <input asp-for="RememberMe" />
    </div>
    <div>
        <input type="submit" value="Увійти" />
    </div>
</form>
```

Внесемо зміни у створене по замовчуванню представлення Home/Index (Рис. 1.7.32 – 1.7.34).

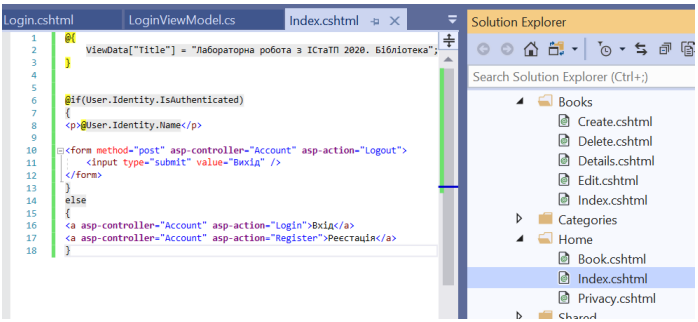


Рисунок 1.7.32 – Фрагмент вікна коду Home/Index

```
@{
    ViewData["Title"] = "Лабораторна робота з ІСтаТП 2020. Бібліотека";
}


@if(User.Identity.IsAuthenticated)
{
<p>@User.Identity.Name</p>

<form method="post" asp-controller="Account" asp-action="Logout">
    <input type="submit" value="Вихід" />
</form>
}
else
{
<a asp-controller="Account" asp-action="Login">Вхід</a>
<a asp-controller="Account" asp-action="Register">Реєстація</a>
}
```

Для входу по замовчуванню саме в Home/Index внесемо ще декілька змін:

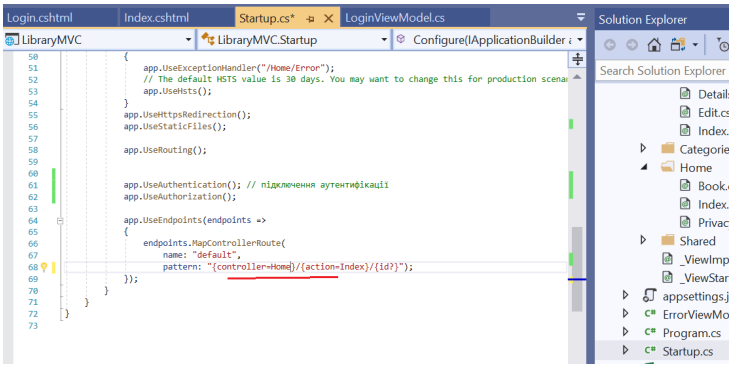


Рисунок 1.7.33 – Фрагмент вікна коду Home/Index


```
31              User user = new User { Email = model.Email, UserName = model.Email, Year = model.Year };
32              // додаємо користувача
33              var result = await _userManager.CreateAsync(user, model.Password);
34              if (result.Succeeded)
35              {
36                  // установка куки
37                  await _signInManager.SignInAsync(user, false);
38                  return RedirectToAction("Index", "Home");
39              }
40              else
41              {
42                  foreach (var error in result.Errors)
43                  {
44                      ModelState.AddModelError(string.Empty, error.Description);
45                  }
46              }
47          }
48          return View(model);
49      }
50
51      [HttpGet]
        0 references
52      public IActionResult Login(string returnUrl = null)
53      {
54          return View(new LoginViewModel { ReturnUrl = returnUrl });
55      }
56
57      [HttpPost]
58      [ValidateAntiForgeryToken]
        0 references
59      public async Task<IActionResult> Login(LoginViewModel model)
60      {
61          if (ModelState.IsValid)
62          {
63              var result =
64                  await _signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, false);
65              if (result.Succeeded)
66              {
67                  // перевіряємо, чи належить URL додатку
68                  if (!string.IsNullOrEmpty(model.ReturnUrl) && Url.IsLocalUrl(model.ReturnUrl))
69                  {
70                      return Redirect(model.ReturnUrl);
71                  }
72                  else
73                  {
74                      return RedirectToAction("Index", "Home");
75                  }
76              }
77              else
78              {
79                  ModelState.AddModelError("", "Неправильний логін чи (та) пароль");
80              }
81          }
82          return View(model);
83      }
84
85      [HttpPost]
86      [ValidateAntiForgeryToken]
        0 references
87      public async Task<IActionResult> Logout()
88      {
89          // видаляємо аутентифікаційні куки
90          await _signInManager.SignOutAsync();
91          return RedirectToAction("Index", "Home");
92      }
93
94  }
```

Рисунок 1.7.34 – Фрагмент вікна коду Home/Index

Можна запускати.

Визначення ролей
Залишилося розібратися з ролями користувачів.

Для роботи з ролями ASP.NET Core Identity надає клас IdentityRole. В цьому класі визначається кілька властивостей:

```
public virtual TKey Id { get; set; }
public virtual string Name { get; set; }
public virtual string NormalizedName { get; set; }
public virtual string ConcurrencyStamp { get; set; }
```

Властивість Name, зберігає назву ролі.

Якщо в БД буде роль «admin», то до контролерів можна прописувати властивість

[Authorize(Roles="admin")]

Ця властивість означає, що до зазначеного контролера доступ мають лише користувачі з роллю «admin».

Для адміністрування ролями скористаємося методами класів UserManager і RoleManager.

Спочатку додамо в папку ViewModels модель ChangeRoleViewModel (Рис. 1.7.35 – 1.7.37).

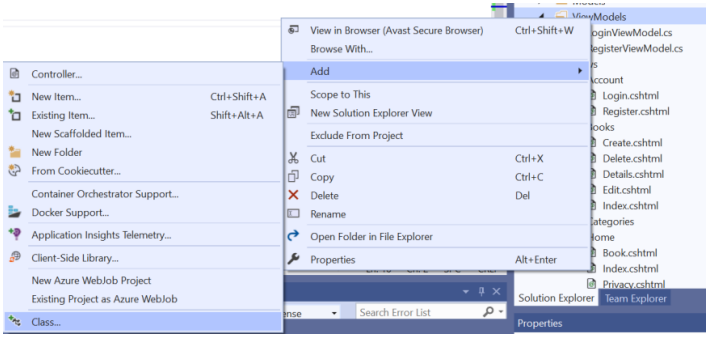


Рисунок 1.7.35 – Фрагмент вікна контекстного меню створення моделі

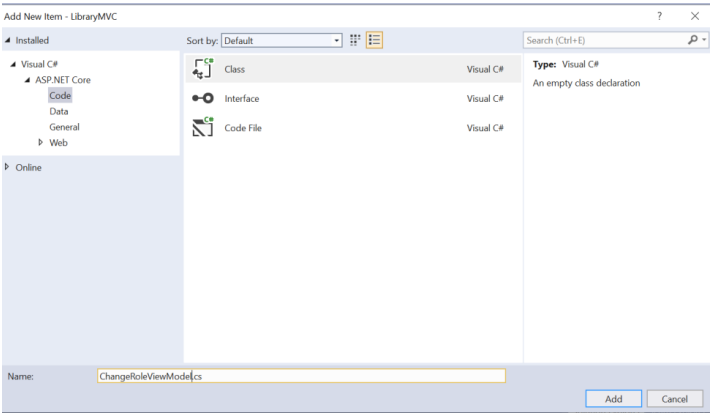


Рисунок 1.7.36 – Фрагмент вікна створення ChangeRoleViewModel

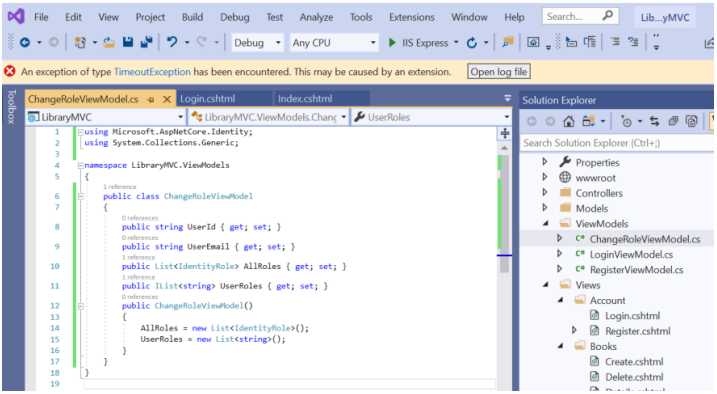


Рисунок 1.7.37 – Фрагмент вікна коду створення моделі ChangeRoleViewModel

```
using Microsoft.AspNetCore.Identity;
using System.Collections.Generic;

namespace LibraryMVC.ViewModels
{
    public class ChangeRoleViewModel
    {
        public string UserId { get; set; }
```

```
        public string UserEmail { get; set; }
        public List<IdentityRole> AllRoles { get; set; }
        public IList<string> UserRoles { get; set; }
        public ChangeRoleViewModel()
        {
            AllRoles = new List<IdentityRole>();
            UserRoles = new List<string>();
        }
    }
}
```

Ця модель дозволить керувати усіма ролями одного користувача (в ASP.NET Core Identity один користувач может мати багато ролей).

Спочатку додамо в папку Controllers новий контролер RolesController (Рис. 1.7.38 – 1.7.40).

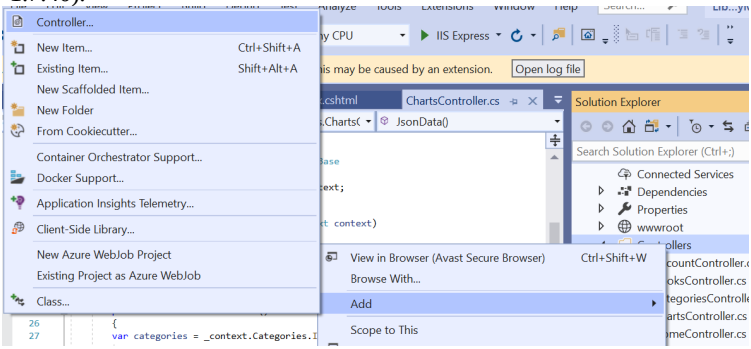


Рисунок 1.7.38 – Фрагмент вікна створення контролеру RolesController

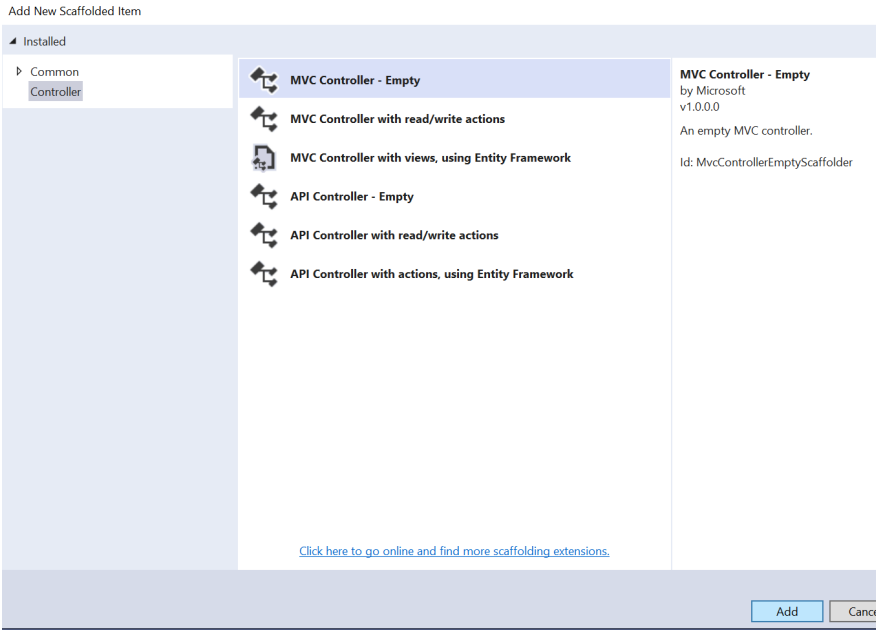


Рисунок 1.7.39 – Фрагмент вікна створення контролеру RolesController

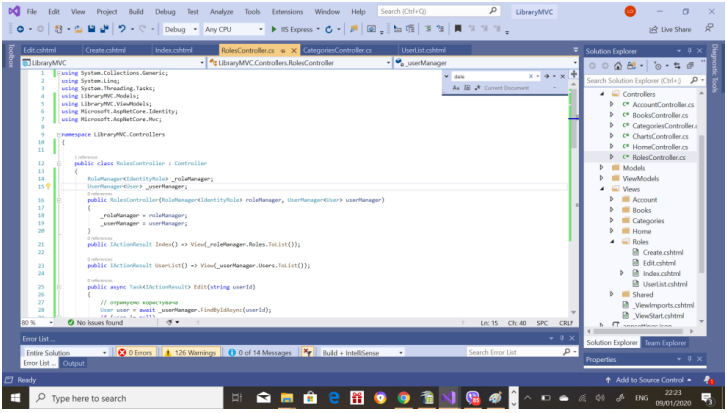


Рисунок 1.7.40 – Фрагмент вікна коду створення контролеру RolesController

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using LibraryMVC.Models;
using LibraryMVC.ViewModels;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace LibraryMVC.Controllers
{

    public class RolesController : Controller
    {
        RoleManager<IdentityRole> _roleManager;
        UserManager<User> _userManager;
        public RolesController(RoleManager<IdentityRole> roleManager,
UserManager<User> userManager)
        {
            _roleManager = roleManager;
            _userManager = userManager;
        }
        public IActionResult Index() => View(_roleManager.Roles.ToList());
        public IActionResult UserList() => View(_userManager.Users.ToList());

        public async Task<IActionResult> Edit(string userId)
        {
            // отримуємо користувача
            User user = await _userManager.FindByIdAsync(userId);
            if (user != null)
            {
                //список ролей користувача
                var userRoles = await _userManager.GetRolesAsync(user);
                var allRoles = _roleManager.Roles.ToList();
                ChangeRoleViewModel model = new ChangeRoleViewModel
                {
                    UserId = user.Id,
                    UserEmail = user.Email,
                    UserRoles = userRoles,
                    AllRoles = allRoles
                };
                return View(model);
            }

            return NotFound();
        }
```

```
        [HttpPost]
        public async Task<IActionResult> Edit(string userId, List<string> roles)
        {
            // отримуємо користувача
            User user = await _userManager.FindByIdAsync(userId);
            if (user != null)
            {
                // список ролей користувача
                var userRoles = await _userManager.GetRolesAsync(user);
                // получаем все роли
                var allRoles = _roleManager.Roles.ToList();
                // список ролей, які було додано
                var addedRoles = roles.Except(userRoles);
                // список ролей, які було видалено
                var removedRoles = userRoles.Except(roles);

                await _userManager.AddToRolesAsync(user, addedRoles);

                await _userManager.RemoveFromRolesAsync(user, removedRoles);

                return RedirectToAction("UserList");
            }

            return NotFound();
        }
    }
}
```

У контролері отримуємо об'єкти RoleManager і UserManager, які передаються через механізм впровадження залежностей.

Для представлень для даного контролера визначимо в папці Views каталог Roles (Рис. 1.7.41).

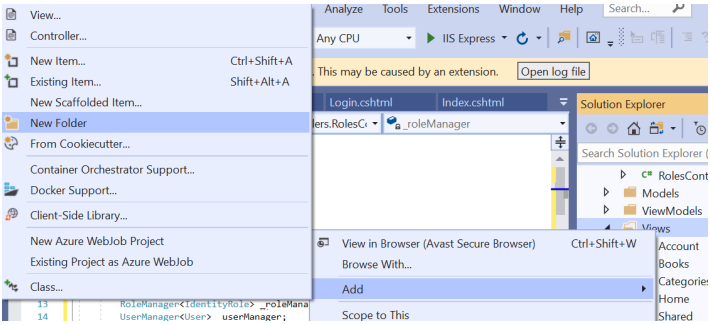


Рисунок 1.7.41 – Фрагмент вікна створення у Views каталогу Roles

Метод Index виводить список ролей. Для їх виведення додамо в каталог Views / Roles нове представлення Index.cshtml (Рис. 1.7.42 – 1.7.44).

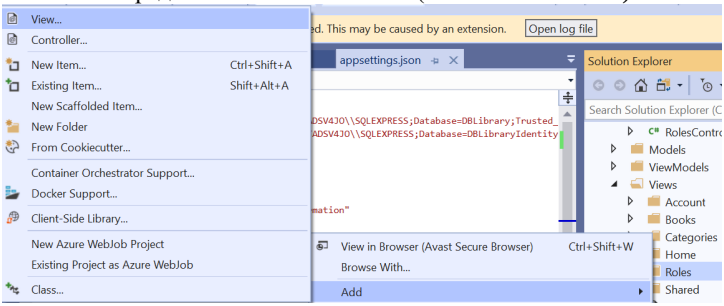


Рисунок 1.7.42 – Фрагмент вікна нового представлення Index.cshtml

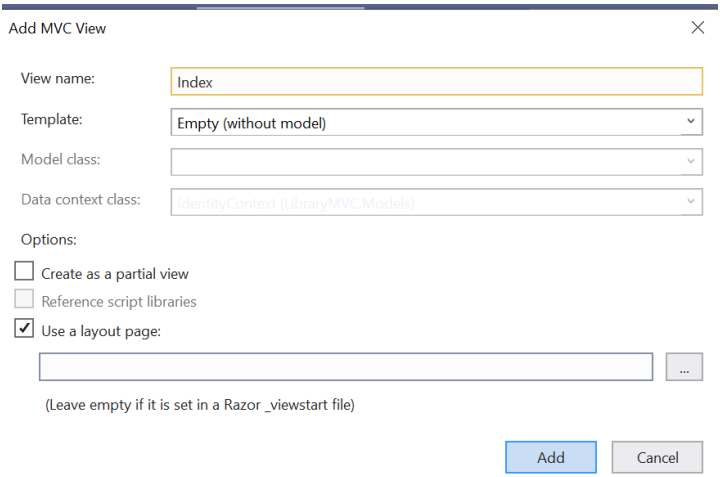


Рисунок 1.7.43 – Фрагмент вікна нового представлення Index.cshtml

Рисунок 1.7.44 – Фрагмент вікна коду Index.cshtml

```
@model IEnumerable<Microsoft.AspNetCore.Identity.IdentityRole>

<h2>Список ролей</h2>
<table class="table">
    @foreach (var role in Model)
    {
        <tr>
            <td>@role.Name</td>
            <td>
                <form asp-action="Delete" asp-route-id="@role.Id" method="post">
                    <button type="submit" class="btn btn-sm btn-danger">
```

```
                Видалити
              </button>
            </form>
          </td>
        </tr>
    }
</table>
<a asp-action="UserList">Список користувачів</a>
```

Для виведення списку користувачів визначено метод UserList. Для виведення користувачів додамо представлення UserList.cshtml (Рис. 1.7.45 – 1.7.47).

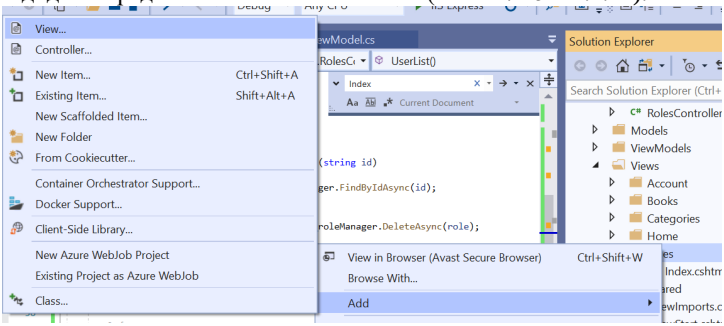


Рисунок 1.7.45 – Фрагмент вікна визначення методу UserList

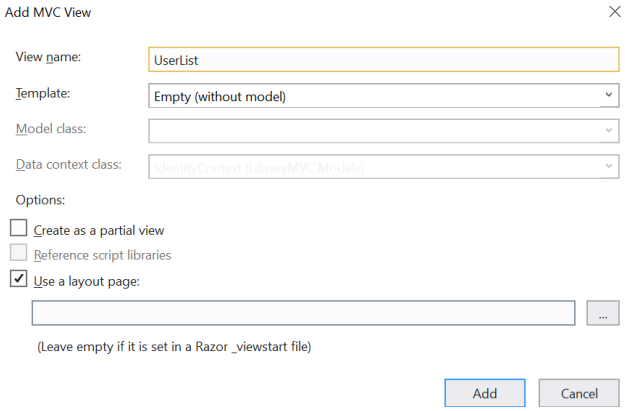


Рисунок 1.7.46 – Фрагмент вікна визначення методу UserList

Рисунок 1.7.47– Фрагмент вікна коду визначення методу UserList

```
@model IEnumerable<LibraryMVC.Models.User>

<h2>Список користувачів</h2>
<table class="table">
   @foreach (var user in Model)
   {
      <tr>
         <td>@user.Email</td>
         <td>
            <a class="btn btn-sm btn-primary" asp-action="Edit" asp-route-
userid="@user.Id">Права доступу</a>
         </td>
      </tr>
   }
</table>
```

При натисканні на посилання в цьому представленні в метод Edit передається id користувача. Потім з цього методу в представленя через спеціальну модель ChangeRoleViewModel передаються його id, email, ролі, а також список взагалі всіх ролей з бази даних.

Додамо для цього методу в папку Views / Roles представлення Edit.cshtml (Рис. 1.7.48 – 1.7.50).

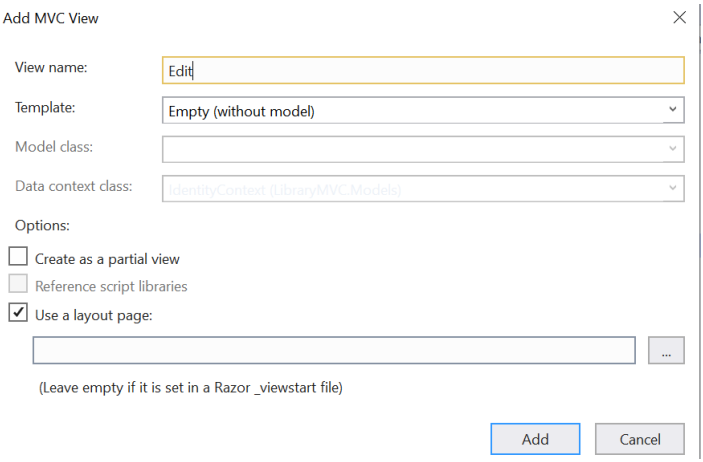


Рисунок 1.7.48 – Фрагмент вікна створення представлення Edit.cshtml

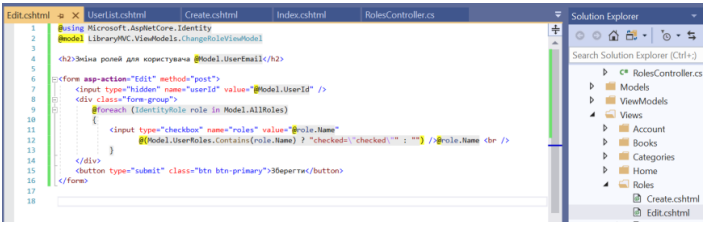


Рисунок 1.7.49 – Фрагмент вікна коду представлення Edit.cshtml

```
@using Microsoft.AspNetCore.Identity
@model LibraryMVC.ViewModels.ChangeRoleViewModel

<h2>Зміна ролей для користувача @Model.UserEmail</h2>

<form asp-action="Edit" method="post">
    <input type="hidden" name="userId" value="@Model.UserId" />
    <div class="form-group">
        @foreach (IdentityRole role in Model.AllRoles)
        {
            <input type="checkbox" name="roles" value="@role.Name"
                @(Model.UserRoles.Contains(role.Name) ? "checked=\"checked\"" :
"") />@role.Name <br />
        }
```

```
    </div>
    <button type="submit" class="btn btn-primary">Зберегти</button>
</form>
```

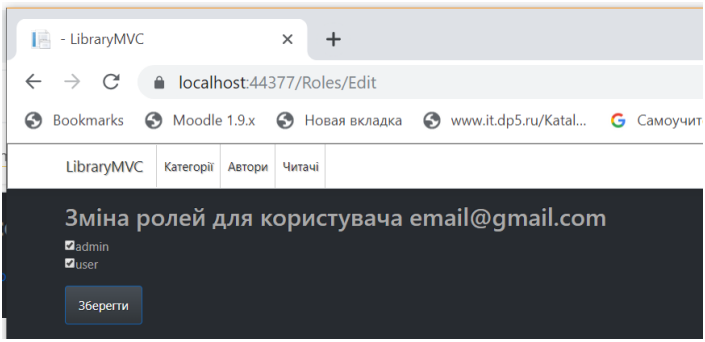


Рисунок 1.7.50 – Фрагмент вікна програми

Ініціалізація БД ролями і користувачами

Для коректної роботи проєкту нам потрібно здійснити початкову ініціалізацію БД ролями і користувачами, т.б. зробити так, щоб база даних до моменту першого звернення додатки вже містила початкові дані, наприклад, базові ролі і обліковий запис адміністратора.

Для ініціалізації бази даних початковими значеннями і користувачами визначимо новий клас RoleInitializer (Рис. 1.7.51 – 1.7.53).

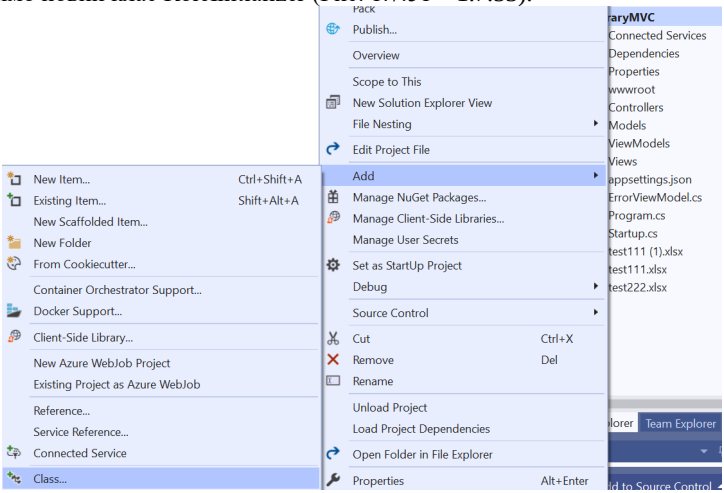


Рисунок 1.7.51 – Фрагмент вікна роботи з класом RoleInitializer

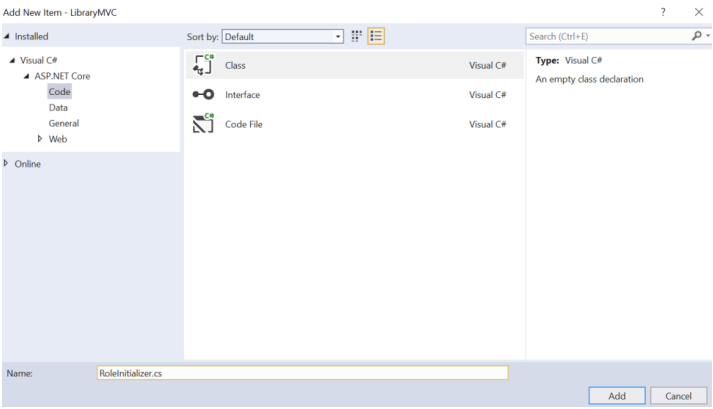


Рисунок 1.7.52 – Фрагмент вікна роботи з класом RoleInitializer

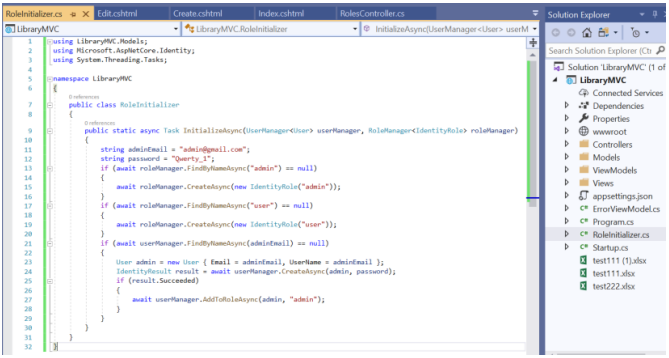


Рисунок 1.7.53 – Фрагмент вікна коду роботи з класом RoleInitializer

```
using LibraryMVC.Models;
using Microsoft.AspNetCore.Identity;
using System.Threading.Tasks;

namespace LibraryMVC
{
    public class RoleInitializer
```

```
    {
        public static async Task InitializeAsync(UserManager<User> userManager,
RoleManager<IdentityRole> roleManager)
        {
            string adminEmail = "aaa@gmail.com";
            string password = "Qwerty_1";
            if (await roleManager.FindByNameAsync("admin") == null)
            {
                await roleManager.CreateAsync(new IdentityRole("admin"));
            }
            if (await roleManager.FindByNameAsync("user") == null)
            {
                await roleManager.CreateAsync(new IdentityRole("user"));
            }
            if (await userManager.FindByNameAsync(adminEmail) == null)
            {
                User admin = new User { Email = adminEmail, UserName =
adminEmail };
                IdentityResult result = await userManager.CreateAsync(admin,
password);
                if (result.Succeeded)
                {
                    await userManager.AddToRoleAsync(admin, "admin");
                }
            }
        }
    }
}
```

У класі визначено метод InitializeAsync (), який додає в базу даних дві ролі - "admin" і "user", а також одного користувача - адміністратора. Для додавання тут застосовуються методи класів UserManager і RoleManager, які нам доступні через колекцію сервісів додатку.

Для ілюстрації всі дані (логін і пароль адміністратора) тут визначені в коді, але природно можна вказати значення і в файлі конфігурації і потім передавати в метод.

Використовуємо даних клас в класі Program при старті програми (Рис.1.7.54).

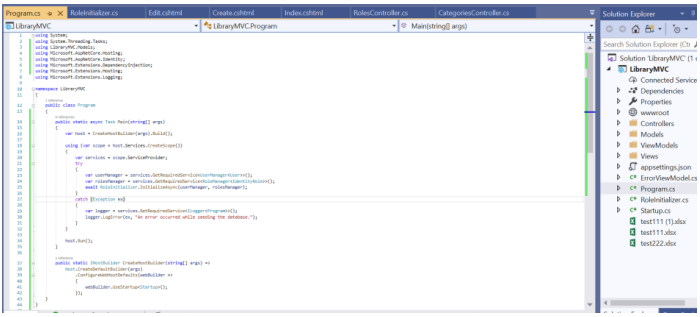


Рисунок 1.7.54 – Фрагмент вікна коду з класом Program

```
using System;
using System.Threading.Tasks;
using LibraryMVC.Models;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace LibraryMVC
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            var host = CreateHostBuilder(args).Build();

            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;
                try
                {
                    var userManager =
services.GetRequiredService<UserManager<User>>();
                    var rolesManager =
services.GetRequiredService<RoleManager<IdentityRole>>();
                    await RoleInitializer.InitializeAsync(userManager, rolesManager);
                }
                catch (Exception ex)
                {
```

```
                var logger = services.GetRequiredService<ILogger<Program>>();
                logger.LogError(ex, "An error occurred while seeding the
database.");
            }
        }

        host.Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
}
```

Тепер залишилося лише біля потрібних контролерів розставити атрибути, що відповідають за доступ для категорій користувачів та додати «using Microsoft.AspNetCore.Authorization;» (Рис. 1.7.55).

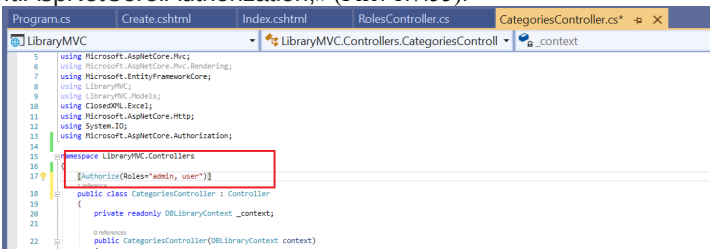


Рисунок 1.7.55 – Фрагмент вікна коду розстановки атрибутів

Запускаємо .

Лабораторна робота № 2
Розробка вебдодатку

Набір технологій:

- *Entity Framework Core (EF Core) Code First – ORM Framework*
- *Web API (ASP.NET Core 3.x)*
- *HTML, JavaScript*

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 та використання саме	**2**

			Code first (див. етап 3.1) .	
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://docs.microsoft.com/ru-ru/aspnet/core/client-side/spa/angular?view=aspnetcore-3.1&tabs=visual-	**3** **+ 2 (додаткові бали)**

			studio, https://metanit.com/sharp/aspnetcore/1.1.php, https://www.infoq.com/articles/Angular-Core-3/, …)).	
2.4	Не пізніше 18-го тижня від початку семестру	Unit-тестування (на 2 додаткових бали) https://metanit.com/sharp/aspnet5/22.3.php	За бажанням можете самостійно опанувати цю тему.	**+ 1 (додатковий бал)**

КРОКИ ВИКОНАННЯ ЕТАПУ 2.1

Створення проєкту

Web API представляє спосіб побудови програми ASP.NET, який спеціально заточений для роботи в стилі REST (Representation State Transfer). REST-архітектура передбачає застосування таких методів або типів запитів HTTP для взаємодії з сервером:

- GET
- POST
- PUT
- DELETE

Найчастіше REST-стиль особливо зручний при створенні Single Page Application, які нерідко використовують спеціальні javascript-фреймворки як то Angular, React або Vue.js. По суті Web API представляє собою вебслужбу, до якої можуть звертатися інші додатки. Причому ці програми можуть представляти будь-яку технологію і платформу – це можуть бути вебдодатки, мобільні або десктопні клієнти.

Структура додатку (Рис. 2.1.1).

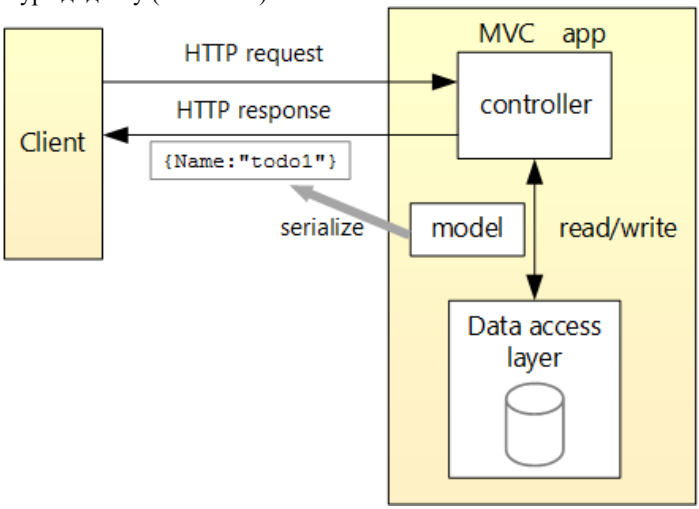


Рисунок 2.1.1 – Фрагмент вікна структури додатку

Створимо проєкт Web API. Для цього при створенні проєкту ASP.NET Core серед шаблонів виберемо Empty (Рис. 2.1.2 – 2.1.4).

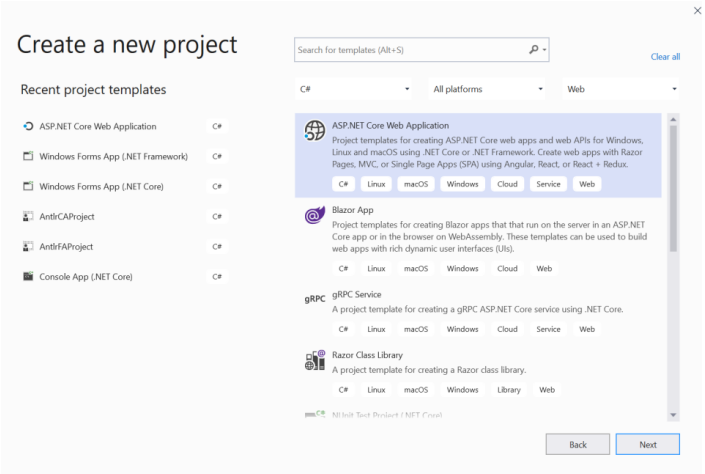


Рисунок 2.1.2 – Фрагмент вікна створення проєкту

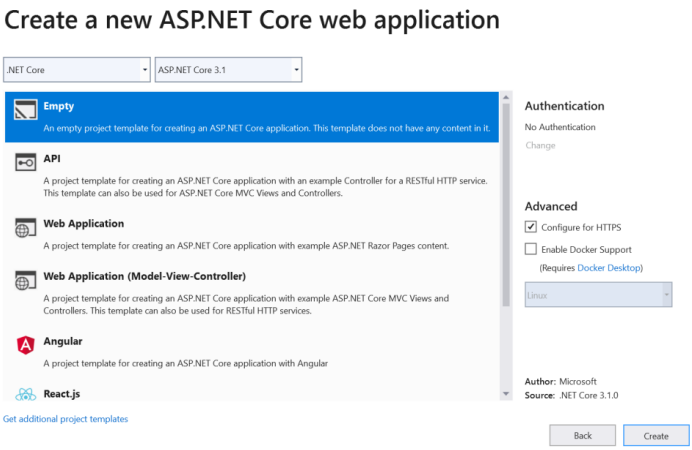


Рисунок 2.1.3 – Фрагмент вікна створення проєкту

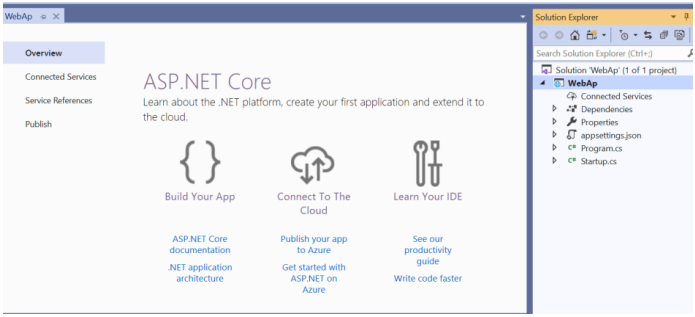


Рисунок 2.1.4 – Фрагмент вікна створення проєкту

Створення моделі та контексту даних

За замовчуванням всі типи сутностей, для яких визначені в контексті даних набори DbSet, включаються в модель і надалі зіставляються з таблицями в базі даних. Але крім того, в модель також включаються типи, на які є посилання в сутності, які вже включені в модель, наприклад, через властивості DbSet.

За замовчуванням кожна сутність зіставляється з таблицею, яка називається за іменем властивості DbSet <T> в контексті даних, що представляє дану сутність. Якщо в контексті даних подібного властивості не визначено, то для назви таблиці використовується ім'я класу сутності.

За замовчуванням кожна класу властивість зіставляється з однойменним стовпцем в таблиці.

За замовчуванням в якості ключа використовується властивість, яка називається Id або [ім'я_класу]Id.

Для встановлення властивості в якості первинного ключа за допомогою анотацій застосовується атрибут [Key].

За замовчуванням для властивостей первинних ключів, які представляють типи int або GUID і які мають значення за замовчуванням, генерується значення при вставці в базу даних. Для всіх інших властивостей значення за замовчуванням не генерується.

За замовчуванням властивість є необов'язковою, якщо воно допускає значення null. Це властивості, які мають, , такі типи як string, int?, byte [], об'єкти класів і т.д. Хоча ми також можемо налаштувати ці властивості як обов'язкові.

Властивість є обов'язковою, якщо значення null не є для неї коректним. Це властивості типів int, decimal, bool і т.д.

Якщо в якості цільової СУБД використовується MS SQL Server, то типи SQL Server і C# зіставляються наступним чином:

int: int

bit: bool
char: string
date: DateTime
datetime: DateTime
datetime2: DateTime
decimal: decimal
float: double
money: decimal
nchar: string
ntext: string
numeric: decimal
nvarchar: string
real: float
smallint: short
text: string
tinyint: byte
varchar: string

Наприклад, за замовчуванням Entity Framework для властивості з типом string буде використовувати стовпець з типом nvarchar.

У провадеров для інших СУБД система зіставлення типів може відрізнятися.

Для зв'язків між моделями в Entity Framework Core застосовуються зовнішні ключі і навігаційні властивості.

За замовчуванням при роботі з ланцюжками успадкування класів Entity Framework Core використовує підхід TPH (Table Per Hierarchy / Таблиця на одну ієрархію класів). На даний момент це єдина форма роботи з ієрархіями класів в Entity Framework Core. При використанні даного підходу TPH для всіх класів з однієї ієрархії в базі даних створюється одна таблиця. А щоб визначити, до якого саме класу належить рядок в таблиці, в цій же таблиці створюється додатковий стовпець - дискримінатор.

Каскадне видалення представляє автоматичне видалення залежної сутності після видалення головної. За замовчуванням для сутностей застосовується каскадне видалення, якщо наявність пов'язаної сутності є обов'язковим.

Для прикладу, нехай у нас буде бібліотека, тоді додамо в проєкт нову папку Models, а в неї помістимо нові класи моделі (Рис. 2.1.5 - 2.1.12).

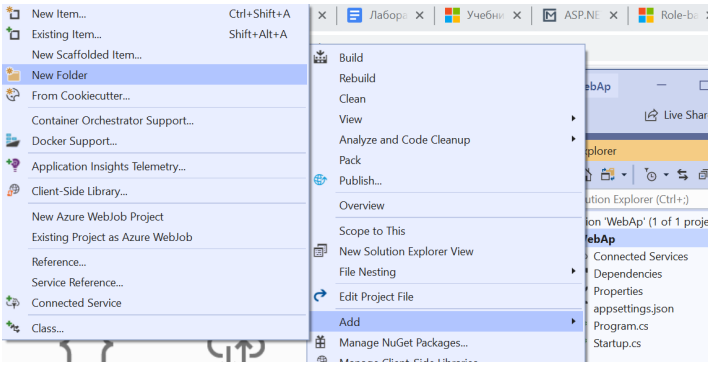


Рисунок 2.1.5 – Фрагмент вікна роботи з Models

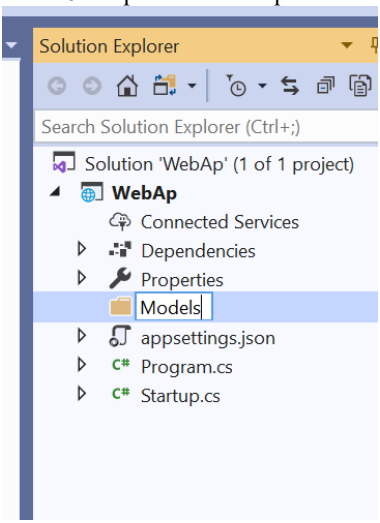


Рисунок 2.1.6 – Фрагмент вікна роботи з Models

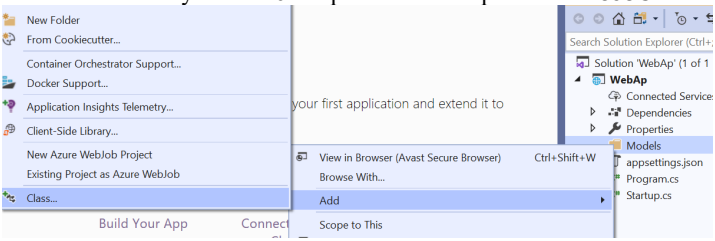


Рисунок 2.1.7 – Фрагмент вікна роботи з Models

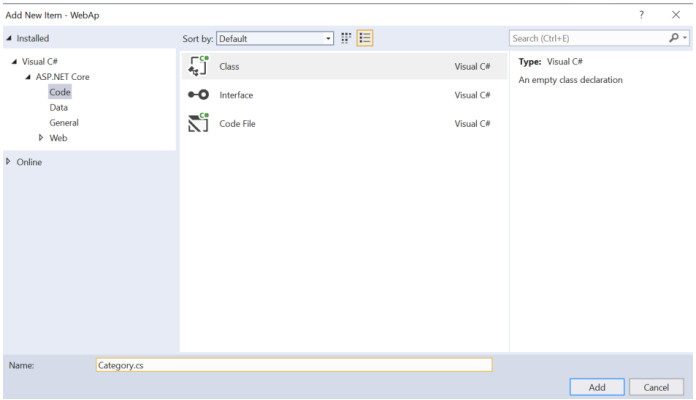


Рисунок 2.1.8 – Фрагмент вікна роботи з класами моделей

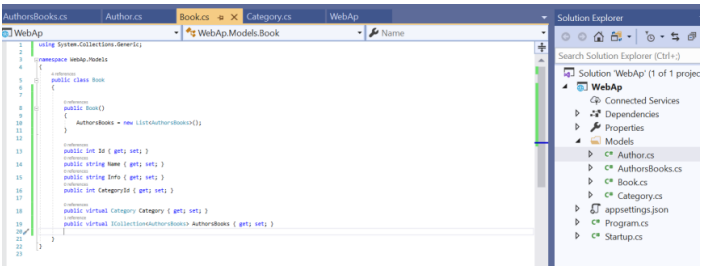


Рисунок 2.1.9 – Фрагмент вікна коду роботи з класами моделей

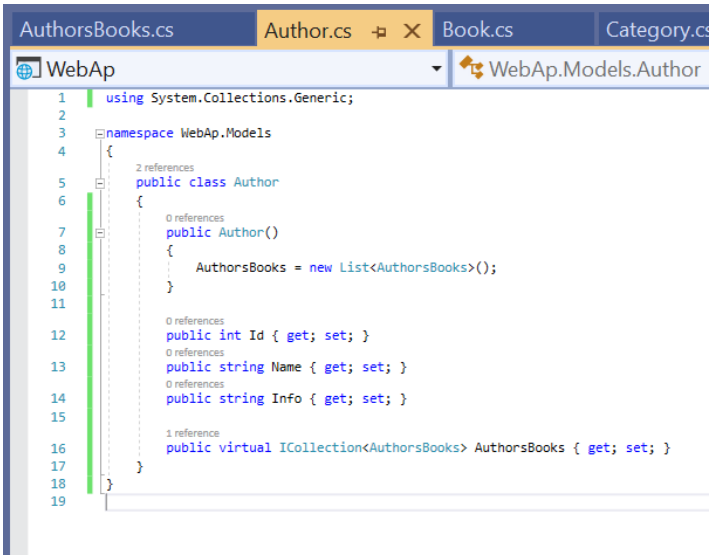


Рисунок 2.1.10 – Фрагмент вікна коду роботи з Models

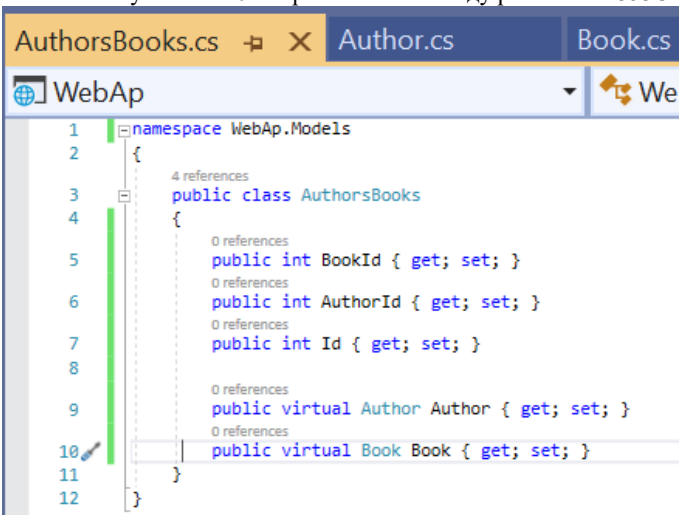


Рисунок 2.1.11 – Фрагмент вікна коду роботи з Models

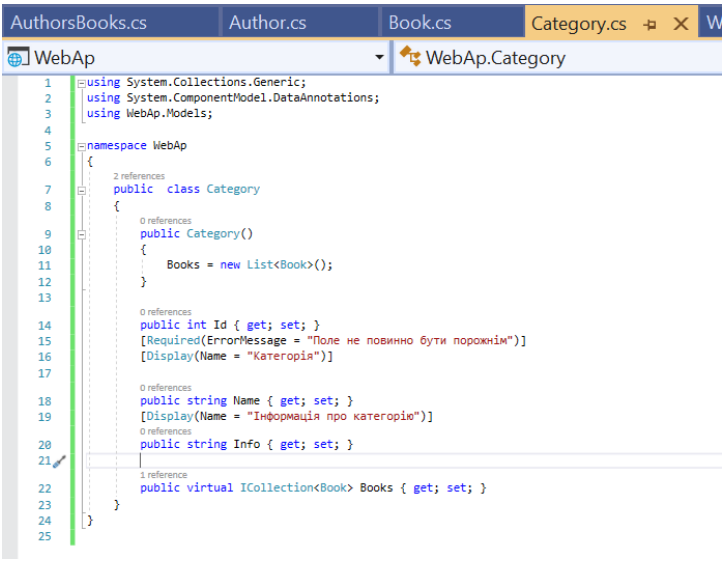


Рисунок 2.1.12 – Фрагмент вікна коду роботи з Models

Для взаємодії з MS SQL Server через Entity Framework через пакетний менеджер Nuget додамо в проєкт пакет Microsoft.EntityFrameworkCore.SqlServer (Рис. 2.1.13 – 2.1.16).

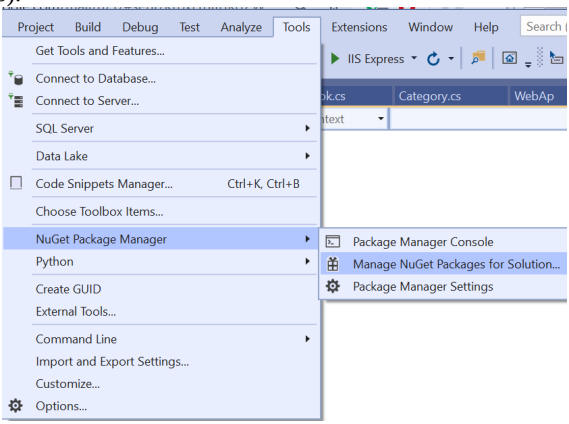


Рисунок 2.1.13 – Фрагмент вікна роботи з проєктом

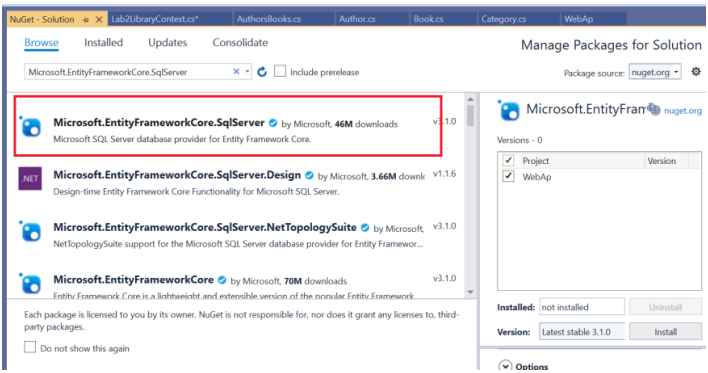


Рисунок 2.1.14 – Фрагмент вікна роботи з пакетом Microsoft.EntityFrameworkCore.SqlServer

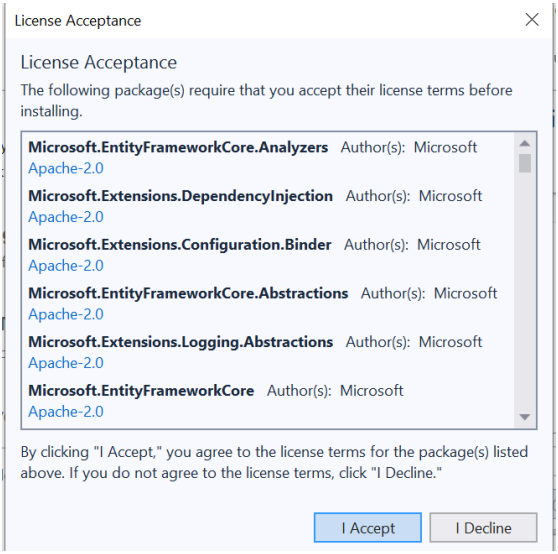


Рисунок 2.1.15 – Фрагмент вікна роботи з пакетом Microsoft.EntityFrameworkCore.SqlServer

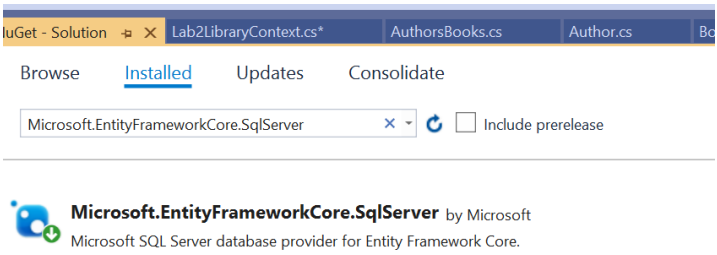


Рисунок 2.1.16 – Фрагмент вікна роботи з пакетом Microsoft.EntityFrameworkCore.SqlServer

В папку Models помістимо новий клас контексту даних (Рис. 2.1.17 – 2.1.19).

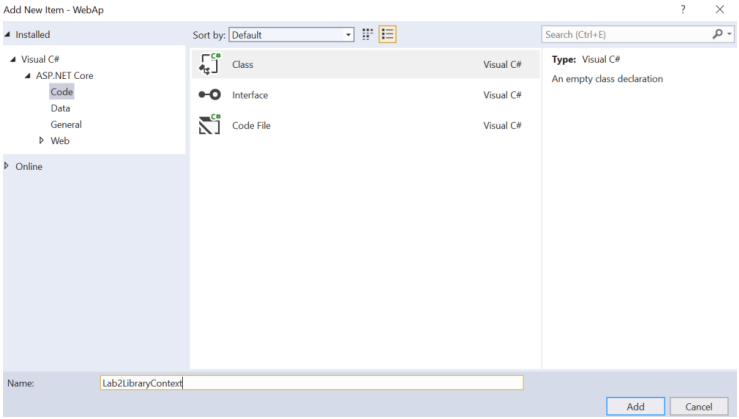


Рисунок 2.1.17 – Фрагмент вікна роботи з класом контексту даних

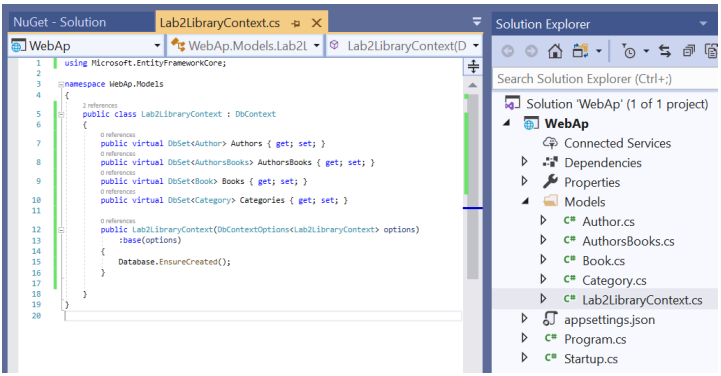


Рисунок 2.1.18 – Фрагмент вікна роботи з класом контексту даних

Змінимо appsetting.json (додамо рядок підключення):

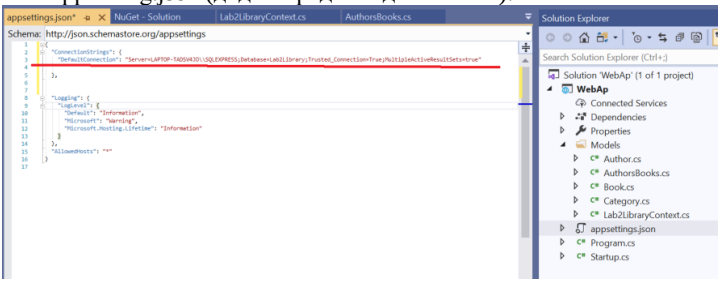


Рисунок 2.1.19 – Фрагмент вікна роботи з класом контексту даних

"ConnectionStrings": {
"DefaultConnection": "Server=*LAPTOP-TADSV4JO\\SQLEXPRESS*;
Database=*Lab2Library*;
Trusted_Connection=True;MultipleActiveResultSets=true"
},

Lab2Library – ім'я нової БД.

LAPTOP-TADSV4JO\\SQLEXPRESS – назва серверу, де буде розміщена Ваша БД (див. в SQL Server Management Studio)

Додамо пакет (Рис. 2.1.20).

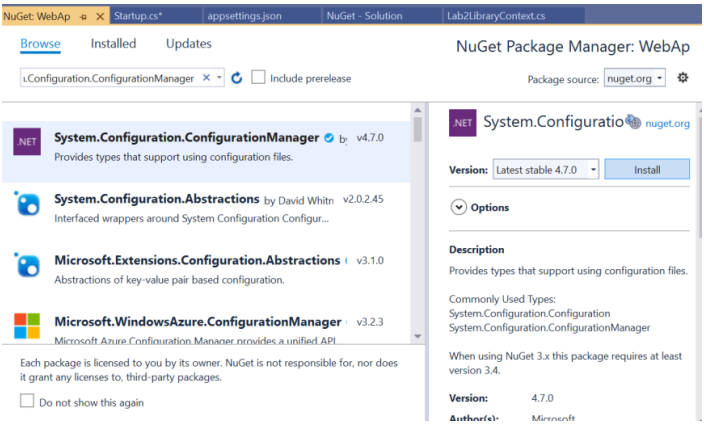


Рисунок 2.1.20 – Фрагмент вікна роботи з пакетом

Змінимо код класу Startup (Рис. 2.1.21).

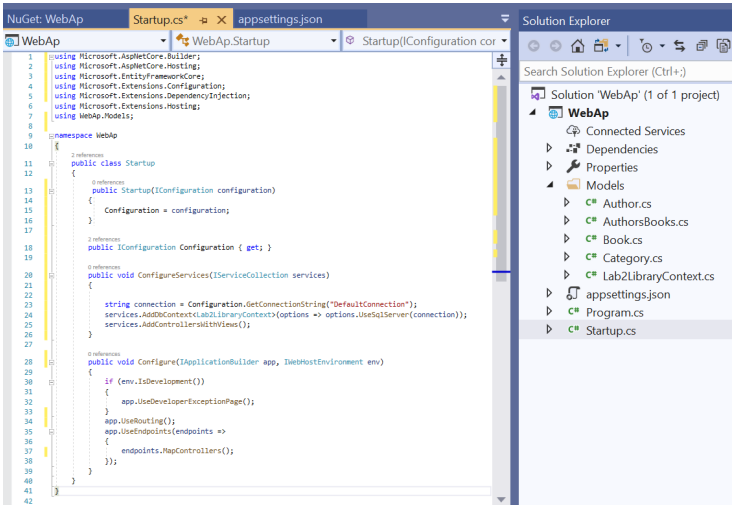


Рисунок 2.1.21 – Фрагмент вікна зміни коду класу Startup

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
using WebAp.Models;

namespace WebAp
{
    public class Startup
    {
         public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {

            string connection =
Configuration.GetConnectionString("DefaultConnection");
            services.AddDbContext<Lab2LibraryContext>(options =>
options.UseSqlServer(connection));
            services.AddControllers();
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            app.UseRouting();
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```

КРОКИ ВИКОНАННЯ ЕТАПУ 2.2

Створення контролерів

Далі додамо в проєкт нову папку Controllers, а в ній створимо новий api-контролер для кожного класу моделі. Для цього при додаванні нового елемента в проєкт можна використовувати шаблон API Controller Class (Рис.2.2.1 – 2.2.7).

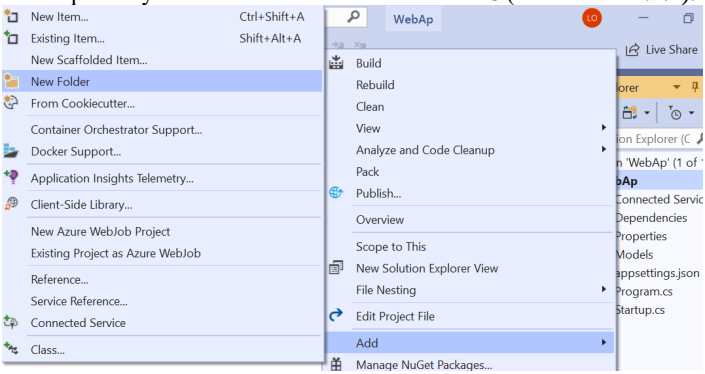


Рисунок 2.2.1 – Фрагмент вікна роботи з контролерами

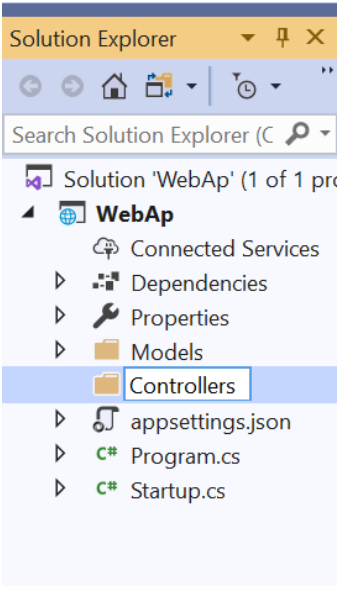


Рисунок 2.2.2 – Фрагмент вікна роботи з контролерами

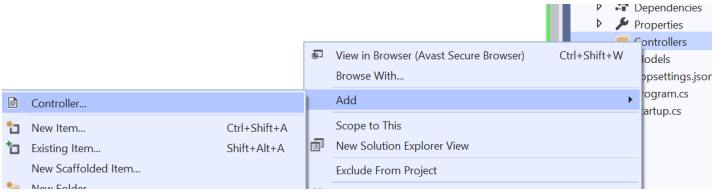


Рисунок 2.2.3 – Фрагмент вікна роботи з контролерами

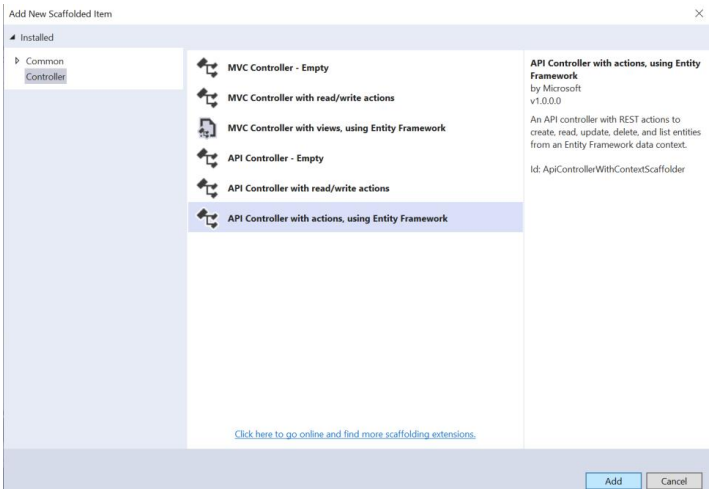


Рисунок 2.2.4 – Фрагмент вікна роботи з контролерами

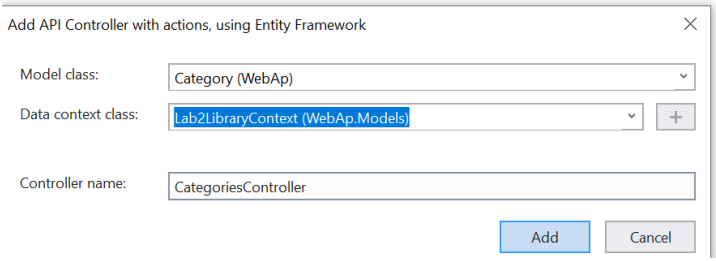


Рисунок 2.2.5 – Фрагмент вікна роботи з контролерами

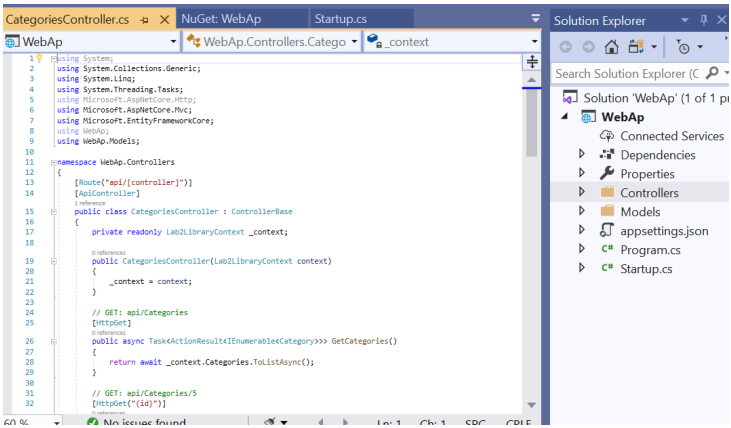


Рисунок 2.2.6 – Фрагмент вікна роботи з контролерами

До контролера застосовується атрибут [ApiController], який дозволяє використовувати низку додаткових можливостей, зокрема, в плані прив'язки моделі. Також до контролера застосовується атрибут маршрутизації, який вказує, як контролер буде зіставлятися з запитами.

У конструкторі контролера отримуємо контекст даних і використовуємо його для операцій з даними. Також в конструкторі контролера додаємо початкові дані.

Контролер API призначений переважно для обробки запитів протоколу HTTP: Get, Post, Put, Delete, Patch, Head, Options. В даному випадку для кожного типу запитів в контролері визначено свої методи. Так, метод Get () обробляє запити типу GET і повертає колекцію об'єктів з БД.

Якщо запит Get містить параметр id (ідентифікатор об'єкта), то він обробляється іншим методом - Get (int id), який повертає об'єкт по переданому id.

Запити типу Post обробляються методом Post, який отримує з тіла запиту відправлені дані і додає їх в базу даних.

Метод Put обробляє запити типу Put - отримує дані з запиту і змінює ними об'єкт в базі даних.

І метод Delete (int id) обробляє запити типу Delete, тобто запити на видалення - отримує із запиту параметр id і по цим ідентифікатором видаляє об'єкт з БД.

Аналогічно створюємо контролери для усіх класів моделі (за необхідності додаємо методи та/або змінюємо існуючі).

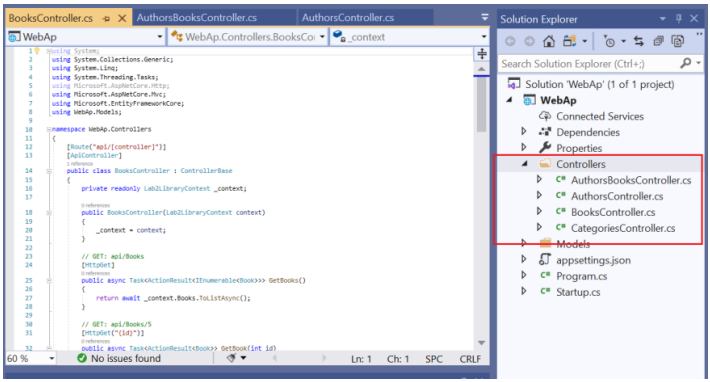


Рисунок 2.2.7 – Фрагмент вікна роботи з контролерами

Запускаємо (Рис.2.2.8).

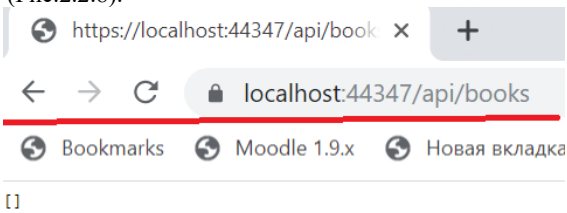


Рисунок 2.2.8 – Фрагмент вікна запуску програми

Ми здійснили звернення до БД, перевіримо її (Рис. 2.2.9).

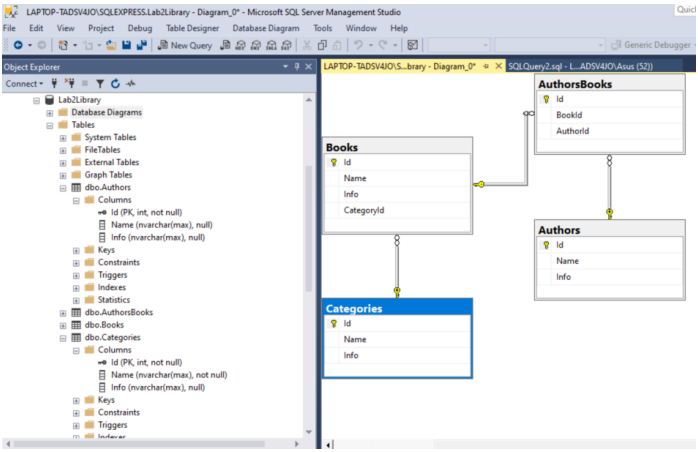


Рисунок 2.2.9 – Фрагмент вікна звернення до БД

Тестування контролерів

Ми створили контролери Web API, і протестували роботу методу GET контролера Books. Однак безпосередньо з рядка браузера крім запитів GET інші типи запитів ми протестувати не можемо. Звичайно, ми можемо створити клієнт у вигляді вебсторінки, мобільного додатка під якусь платформу або навіть графічного або консольного десктопного додатка, але створення клієнта може зайняти досить багато часу, тоді як нам просто треба протестувати обробку запитів.

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

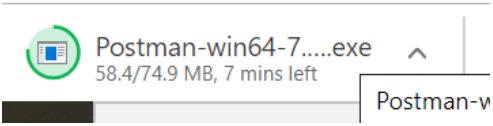


Рисунок 2.2.11 – Фрагмент вікна інсталяції програми

Запустіть вебдодаток.

Запустіть Postman (В File-> Settings відключіть параметр **SSL certificate verification** – після завершення роботи увімкніть знову).

Створіть новий запит (Рис. 2.2.12 – 2.2.15).

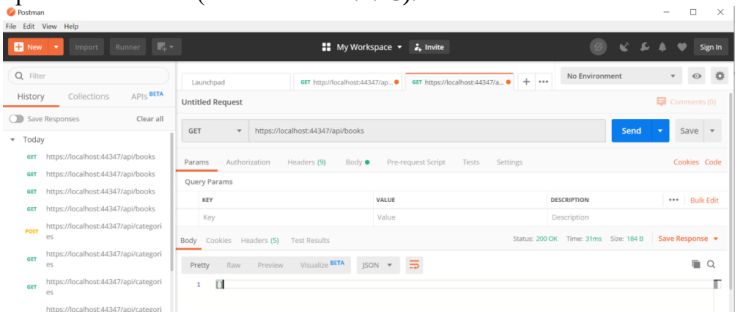


Рисунок 2.2.12 – Фрагмент вікна роботи з вебдодатком

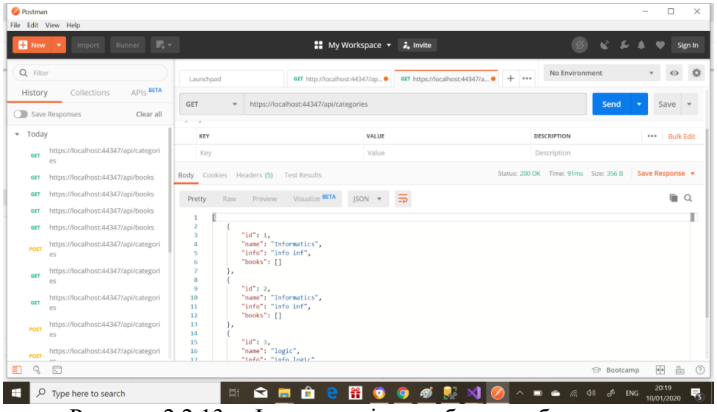


Рисунок 2.2.13 – Фрагмент вікна роботи з вебдодатком

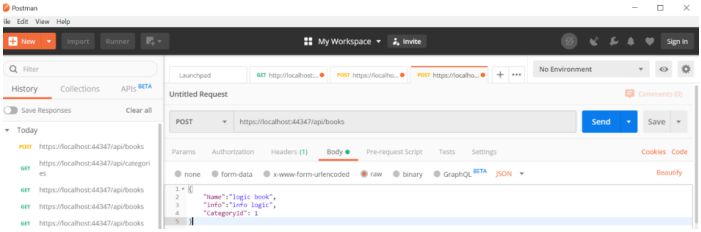


Рисунок 2.2.14 – Фрагмент вікна роботи з вебдодатком

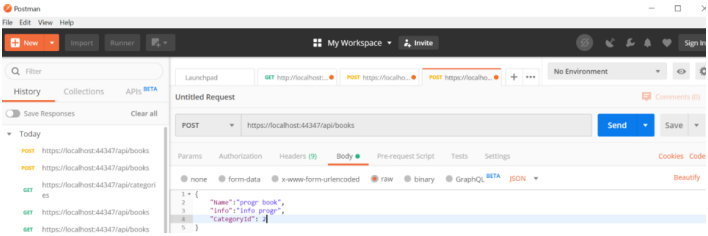


Рисунок 2.2.15 – Фрагмент вікна роботи з вебдодатком

Для здачі цього етапу підготуйте тестові приклади для усіх методів контролерів.

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://learn.javascript.ru/fetch).

Налаштуйте в додатку обслуговування статичних файлів і включіть зіставлення файлів за замовчуванням. Вставте в метод Configure в файлі Startup.cs наступний виділений код (Рис.2.3.1)

Рисунок 2.3.1 – Фрагмент вікна роботи з методом Configure

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

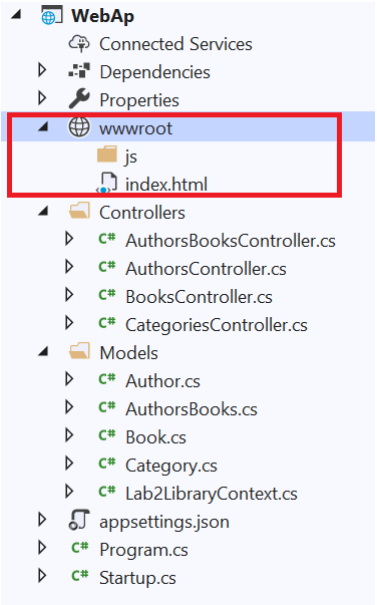


Рисунок 2.3.2 – Фрагмент вікна роботи з папкою wwwroot

Додайте HTML-файл index.html в папку wwwroot:

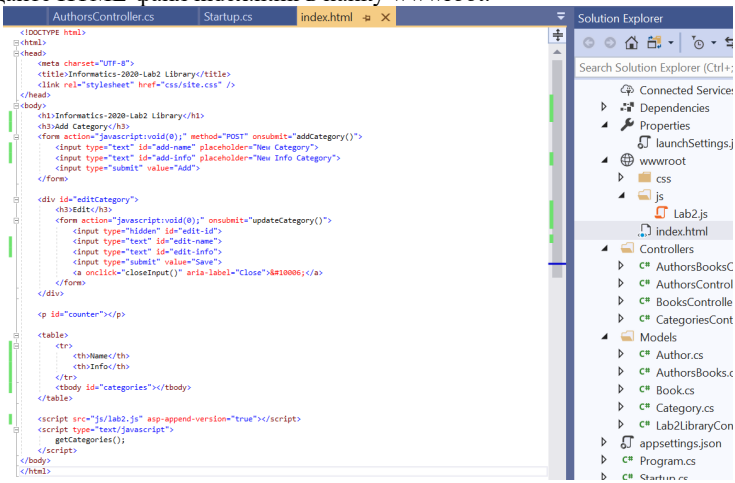


Рисунок 2.3.3 – Фрагмент вікна роботи з папкою wwwroot

```
<!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 JavaScript з іменем lab2.js в папку wwwroot/js (Рис.2.3.4).

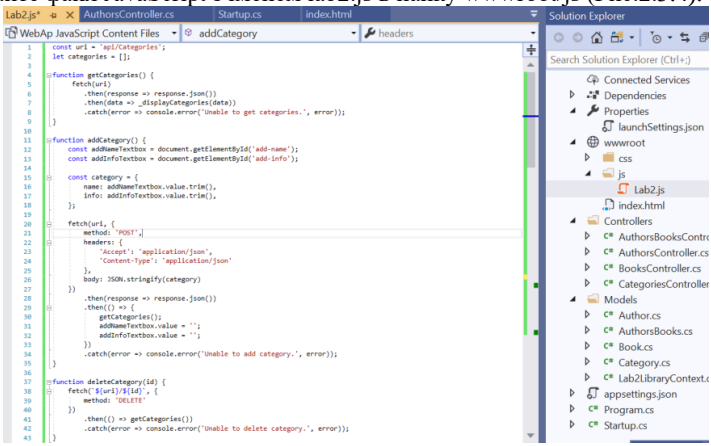


Рисунок 2.3.4 – Фрагмент вікна роботи з папкою wwwroot/js

```
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)

Можна запускати (Рис.2.3.5).

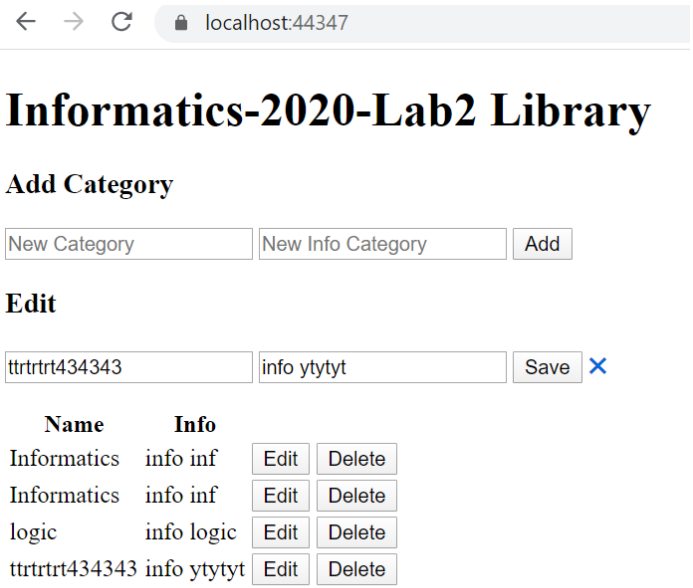


Рисунок 2.3.5 – Фрагмент вікна демонстрації етапу

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

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 програми, інтерфейси, класи, поля, …. Коли їх у програмі з десяток чи два, то вибір не є дуже принциповим питанням. Однак коли їхня кількість у системі сягає десятків, сотень, а то й тисяч, то, щоб не втратити контроль над системою, при виборі імен необхідно дотримуватись певної строгої дисципліни й порядку.

Доцільно використовувати осмислені імена для глобальних змінних і, за можливості, короткі – для локальних. Глобальні змінні, функції, класи та структури за означенням можуть з'являтись у довільному місці програми, тому найважливішою для них є інформативність, а не довжина. Краще використовувати глобальні імена так, щоб вони явно вказували на смисл понять, що ними подаються, тобто мали відповідну **мнемоніку**. Корисно опис кожної змінної, методу, класу, інтерфейсу супроводжувати коротким коментарем.

Однак коментарі не мають містити очевидної інформації на зразок того, що оператор `i++` збільшує змінну `i` тощо.

Для локальних змінних, навпаки, краще підходять короткі імена. Для використання всередині функції цілком підійдуть імена `i, j, n` тощо. При цьому 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. Ці основні конструкції програм слід писати 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 el in testList)
            {
                Console.WriteLine(el);
            }
        }
    }
```

Зрозумілість і стислість 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 boolVariable = GetBoolValue ();
if (boolVariable)
{
}
```

- 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:

```
/ * Blabla * /
```

- 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 IEnumerable { … }
Значення перелічного типу	Pascal Casing	enum SampleEnum {

Перелічний тип	Pascal Casing	FirstValue, SecondValue }

Делегат обробник події	Pascal Casing
	Закінчується суфіксом «EventHandler»
		public delegate void AnswerCreatedEventHandler(object sender, AnswerCreatedEventArgs e);
		public class AnswerCreatedEventArgs: EventArgs
		{
		public int CreatedId;
		public int ParentId;
		public string CreatorName;
		}

Клас-нащадок від EventArgs	Pascal Casing Закінчується суфіксом «EventArgs»	

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»
		[System.AttributeUsage(System.AttributeTargets.All, Inherited = false, AllowMultiple = true)]
		sealed class SampleAttribute:
		System.Attribute
		{
		public SampleAttribute()
		{
		}
		}

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

ЗРАЗОК ФРАГМЕНТУ КОНТРОЛЬНОЇ РОБОТИ 1

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

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

*Обов'язкове поле

1. Електронна адреса *

2. Напишіть своє прізвище та ім'я: *

3. Виберіть свою групу: *

Виберіть лише один варіант.

- К-24
- К-25
- К-26

4. Оберіть правильне визначення. Система керування базами даних (СКБД) – це * 1 бал

Виберіть лише один варіант.

- ◯ 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

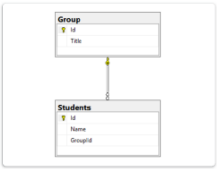
5. Яке ключове слово в SQL визначає умову пошуку, як перевірку належності діапазону значень * 1 бал

Виберіть лише один варіант.

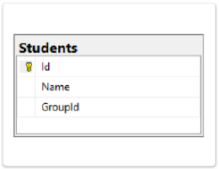
- ◯ IN
- ◯ BETWEEN
- ◯ LIKE
- ◯ IS
- ◯ GROUP BY
- ◯ JOIN

```
class StudentContext : DbContext
{
    public StudentContext()
    {
        Database.EnsureCreated();
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer("Server=.\\SQLEXPRESS; Database=StudentTestDB; Trusted_Connection=True");
    }

    public DbSet<Student> Students { get; set; }
}

public class Group
{
    public int Id { get; set; }

    public string Title { get; set; }
}
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public Group Group { get; set; }
}
```

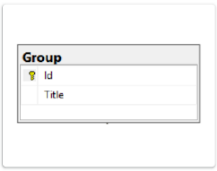
Виберіть лише один варіант.



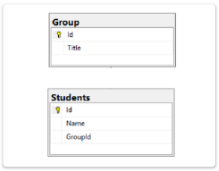
Варіант:



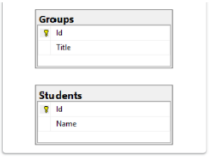
Варіант:



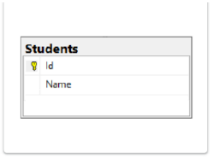
Варіант:



Варіант:



Варіант:



Варіант:

8. Який метод у класа DbTransaction "фіксує" транзакцію бази даних? * 1 бал

Виберіть лише один варіант.

- ( ) Rollback
- ( ) Transaction
- ( ) BeginTransaction
- ( ) Fixate
- ( ) Commit
- ( ) Refuse

9. Колекція OrderItems являє собою набір OrderItem з властивостями UnitPrice, Discount, і Quantity. Ви хочете, щоб запит, відфільтрував об'єкти, OrderItem знайшовши TotalPrice (UnitPrice * Quantity * Discount) з результатом більшим за 10000. Ви хочете сортувати TotalPrice, і включити повну ціну у ваш select. яке ключове слово ви можете використовувати для створення TotalPrice у запиті LINQ, при якому вам не доведеться повторити формулу тричі? * 1 бал

Виберіть лише один варіант.

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

16. Які з цих класів надають доступ до даних лише для читання в прямому напрямку на стороні сервера. * 1 бал

Виберіть усе, що підходить.

- [ ] для зміни конфігураційного файлу джере
- [ ] 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.

17. Для чого використовується DataAdapter? (виберіть усі правильні відповіді) * 1 бал

Виберіть усе, що підходить.

- [ ] для зміни конфігураційного файлу джерела даних
- [ ] для заповнення об'єкта DataSet
- [ ] для шифрування даних
- [ ] для модифікації
- [ ] вірних відповідей немає
- [ ] для створення з'єднання

19. Що буде в результаті виконання наступного фрагменту коду? * 2 бали

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

Виберіть лише один варіант.

- ◯ 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;

ЗРАЗОК ФРАГМЕНТУ КОНТРОЛЬНОІ РОБОТИ 2

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

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

1. Адрес электронной почты *

Інформація про студента

2. Напишіть своє прізвище та ім'я: *

3. Виберіть свою групу: *

Отметьте только один овал.

- К-24
- К-25
- К-26
- К-27

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

Отметьте только один овал.

- Зобов'язуюсь

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

```
@{
    var a = 1;

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

Отметьте только один овал.

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

6. Оберіть правильне твердження. Методи дій... * 1 балл

Отметьте только один овал.

- служать для відображення інтерфейсу програми
- здійснюють взаємодію з користувачем, обробляють дані, що вводяться та відповідає на дії користувача
- реалізують логіку даних програми
- обробляють певний запит за певним URL і повертають деякий результат обробки запиту

10. Чи можливо стандартними засобами ASP.NET MVC змінити правила маршрутизації? * 1 балл

Отметьте только один овал.

- Так
- Ні, стандартними не можна
- Ні, зовсім неможливо

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 балла

Отметьте все подходящие варианты.

- [ ] /BookController / BookDetails / 2
- [ ] /BookDetails / 2
- [ ] /BookController /BookDetails /? Id = 2
- [ ] / Book/BookDetails /? Id = 2
- [ ] / BookController / Book / 2
- [ ] /Book/BookDetails / 2

15. Оберіть правильні(-ий) тип(-и) Впровадження залежностей (Dependency Injection)? * 1 балл

Отметьте все подходящие варианты.

- [ ] Впровадження в конструктор (Constructor Injection);
- [ ] Впровадження у властивість (Setter Injection);
- [ ] Впровадження в параметр (Parameter Injection).
- [ ] Впровадження в проект (Project Injection).

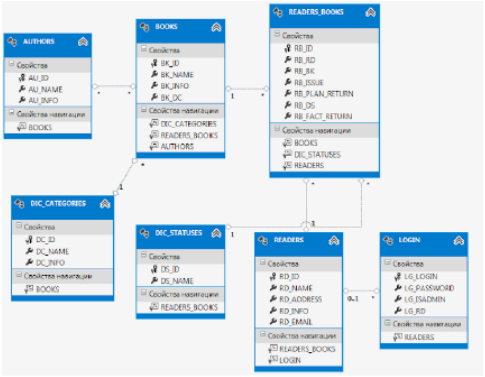
16. Які типи повертаємого значення можуть бути у стандартного асинхронного методу в стилі Task-based Asynchronous Pattern в C# 8? (відкрите, коротке) * 1 балл

17. Для яких методів Linq to Entities існують асинхронні аналоги? * 1 балл

Отметьте все подходящие варианты.

- [ ] Select
- [ ] Where
- [ ] OrderBy
- [ ] All
- [ ] Any
- [ ] Average

18. Написати метод API контролера для запиту Get, який за id читача (RD_ID) знаходить список книжок, які він брав з 01.09.2019 до 31.06.2020 (BOOKS). Контекст задано в класі контролера: private readonly LibraryContext _context;Передбачити можливість асинхронного виконання.Модель відповідає наступній схемі: * 3 балла



ПЕРЕЛІК ВЖИВАНИХ СКОРОЧЕНЬ

CLR	Common Language Runtime – вихідний файл виконання мов
FCL	Framework Class Library – стандартна бібліотека класів платформи .NET Framework
HTML	HyperText Markup Language – мова розмітки гіпертекстових документів
IDE	Integrated development environment – інтегроване середовище розробки
LINQ	Language Integrated Query – мова інтегрованих запитів
UML	Unified Modeling Language – уніфікована мова моделювання
XML	Extensible Markup Language – розширювана мова розмітки
АРМ	автоматизоване робоче місце
БД	база даних
БНФ	нотація Бекуса-Наура
ЖЦ	життєвий цикл
ООП	об’єктно-орієнтоване програмування
П.І.П.	прізвище, ім’я, по-батькові
ПС	програмна система

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проєктирования. Паттерны проєктирования = Design Patterns: Elements of Reusable Object-Oriented Software. — Addison-Wesley, «Питер», 1994. — Р. 395.
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/ru-ru/.
4. Об’єктно-орієнтоване програмування. Лабораторний практикум: навчальний посібник / Б.І. Бойко, Л.Л. Омельчук, Н.Г. Русіна – К.: 2016. – 90 с.
5. Лаврищева Е.М. Современные методы программирования: возможности и инструменты // Проблеми програмування. – 2006.– № 2-3. – С.60-74.
6. Буч Г., Якобсон А., Рамбо Дж. UML. Классика CS. 2-е изд. / Пер. с англ.; Под общей редакцией проф. С. Орлова — СПб.: Питер, 2006. — 736 с.
7. Крэг Ларман. Применение UML 2.0 и шаблонов проєктирования = Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development. — 3-е изд. — М.: Вильямс, 2006. — 736 с. Додаткова
8. Ставровский А.Б, Карнаух Т.О. Перші кроки програмування.-К.:Диалектика.-2005, с.389.
9. Кнут Д. Искусство программирования.Т.1,2,3. Изд 3-е, испр. - М. СбП. К.:Вильямс, 2001.
10. Вирт Н. Систематическое программирование. Введение.- М.:Мир,1987.с.184.
11. Вирт Н. Алгоритмы и структуры данных. - М.:Мир,1989.с. 263.
12. 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.
13. GitHub для пользователей Windows [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://habr.com/ru/post/313996/.
14. КАК ИСПОЛЬЗОВАТЬ GIT В MICROSOFT VISUAL STUDIO ENTERPRISE 2015 RC [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://itvdn.com/ru/blog/article/using-git-in-visual-studio-enterprise-2015.
15. GitHub Extension In Visual Studio 2017 [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/install-and-use-github-extension/.
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. Управление пользователями [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/16.7.php.
18. Изменение пароля [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/16.8.php.
19. Валидация пароля [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/16.9.php.
20. Валидация пользователя [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/16.10.php.
21. NET Core 3.1 SDK (v3.1.100) - Windows x64 Installer [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/download/dotnet-core/thank-you/sdk-3.1.100-windows-x64-installer.
22. Display live data on your site [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/chart.
23. Использование шаблона проекта Angular с ASP.NET Core [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/aspnet/core/client-side/spa/angular?view=aspnetcore-3.1&tabs=visual-studio.
24. Введение в ASP.NET Core и Angular Первый проект на ASP.NET Core с Angular [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://metanit.com/sharp/aspnetcore/1.1.php.
25. Angular & ASP.NET Core 3.0 - Deep Dive [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.infoq.com/articles/Angular-Core-3/.
26. Создание юнит-тестов [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/22.3.php.
27. Download Postman for Windows [Електронний ресурс] – Режим доступу до ресурсу: https://www.getpostman.com/downloads/.
28. Fetch [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://learn.javascript.ru/fetch.

Навчальне видання

Омельчук Людмила Леонідівна
Русіна Наталія Геннадіївна

ІНСТРУМЕНТАЛЬНІ СЕРЕДОВИЩА ТА ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ. ЛАБОРАТОРНИЙ ПРАКТИКУМ

Навчальний посібник

До 50-річчя кафедри
теорії та технології програмування
факультету комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка

Підписано до друку 19.10.2020 р.
Формат 64x90/16. Папір офс. Друк офс. Ум. друк арк. 10,23
Гарнітура Times New Roman. Наклад: 300 прим.,
Замовлення № 19/10/20

Виготовлювач: Типографія «Айс Принт»
Свідоцтво серія ДК № 3534 від 24.07.2009 р.
Тел: +38 (099) 192-00-33, +38 (048) 706-92-82
Site: www.ice-print.com.ua
E-mail:info@ice-print.com.ua