**Міністерство освіти і науки України**
**Луцький національний технічний університет**
**Факультет комп'ютерних та інформаційних технологій**
**Кафедра інженерії програмного забезпечення**

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

# РОЗРОБКА ВЕБСИСТЕМИ КЕРУВАННЯ ВІДВІДУВАЧАМИ ВИКОРИСТОВУЮЧИ ASP.NET CORE

# DEVELOPMENT OF A WEB-BASED VISITOR MANAGEMENT SYSTEM USING ASP.NET CORE

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-44
**Грисюк Дмитро Анатолійович**

Керівник:
**Денисюк Віктор Юрійович**

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2026 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____

Луцьк – 2026 року

**ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
ЗАВІДУВАЧ КАФЕДРИ
_____
«___» _____ 20__ р.

**ЗАВДАННЯ**

**НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ**

Грисюку Дмитру Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка вебсистеми керування відвідувачами використовуючи ASP.NET Core»*

Керівник роботи: *к.т.н., доцент Денисюк В. Ю.* _____
затверджені наказом закладу вищої освіти від *«31» грудня 2025 р.* *№ 541/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра____
*«29» травня 2026 р.*
3. Вихідні дані до роботи: *C#, ASP.NET Core MVC, Entity Framework Core, SQLite, Visual Studio, HTML, CSS, Bootstrap, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз сучасного стану проблем автоматизації обліку персоналу та управління задачами, постановка завдання, формування вимог до розроблюваної інформаційної системи, обґрунтування вибору технологій та засобів розробки, практична реалізація об'єкта проєктування, розробка структури бази даних, обґрунтування принципів захисту інформаційно-комп'ютерної системи, тестування та налагодження програмного забезпечення*.
5. Перелік графічного матеріалу: *17 рисунків, 4 лістинги, 4 таблиці.*
_____

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Аналіз предметної області | *Денисюк В. Ю.* | | |
| Специфікація вимог до розробленої системи | *Денисюк В. Ю.* | | |
| Розробка об’єкта проектування | *Денисюк В. Ю.* | | |
| Нормоконтроль | *Суринович О. М.* | | |
| Гарант ОП | *Ліщина Н. М.* | | |
| Показник запозичень тексту | ***___%*** | | |
| Академічна доброчесність | *Денисюк В. Ю.* | | |

7. Дата видачі завдання *«3» лютого 2026 р.*

| № | Назва етапів кваліфікаційної роботи бакалавра | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Огляд літературних джерел по темі кваліфікаційної роботи бакалавра | до 14.02.2026 р. | |
| 2 | Аналіз проблеми розробки та впровадження об’єкту проектування | до 03.03.2026 р. | |
| 3 | Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання | до 14.03.2026 р. | |
| 4 | Розробка функціонально-структурної схеми роботи об’єкта проектування та проектування бази даних | до 14.04.2026 р. | |
| 5 | Практична реалізація об’єкта проектування та розробка бази даних | до 20.04.2026 р. | |
| 6 | Тестування та налагодження об’єкта проектування | до 04.05.2026 р. | |
| 7 | Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру | до 29.05.2026 р. | |

Здобувач вищої освіти _____ Дмитро ГРИСЮК

Керівник кваліфікаційної роботи _____ Віктор ДЕНИСЮК

# АНОТАЦІЯ

Грисюк Д. А. Розробка вебсистеми керування відвідувачами використовуючи ASP.NET Core. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків.

У роботі розроблено вебсистему керування відвідувачами на основі ASP.NET Core. У першому розділі досліджено предметну область організації обліку відвідувачів, проаналізовано наявні програмні рішення та визначено доцільність створення власної системи. У другому розділі сформовано функціональні та нефункціональні вимоги, виконано проєктування структури програмного забезпечення, моделей даних і сценаріїв взаємодії користувачів із системою. У третьому розділі описано практичну реалізацію вебзастосунку, розробку бази даних, засоби захисту інформації, а також результати тестування та налагодження. Результатом роботи є програмний продукт, що забезпечує реєстрацію, облік і супровід процесів керування відвідувачами.

Ключові слова: вебсистема, керування відвідувачами, ASP.NET Core, база даних, автентифікація, інформаційна система, тестування.

# ABSTRACT

Hrysiuk D. A. Development of a Web-Based Visitor Management System Using ASP.NET Core. Manuscript. Qualification work for bachelor's degree in Software Engineering, specialty 121 Software Engineering. Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The work presents the development of a web-based visitor management system using ASP.NET Core. The first chapter examines the subject area of visitor registration and management, analyzes existing software solutions, and substantiates the need for a dedicated system. The second chapter defines functional and non-functional requirements, describes the software structure, data models, and user interaction scenarios. The third chapter presents the practical implementation of the web application, database development, information security mechanisms, and testing results. The result is a software product that supports visitor registration, accounting, and management workflows.

Keywords: web system, visitor management, ASP.NET Core, database, authentication, information system, testing.

# ЗМІСТ

# ВСТУП

Актуальність теми. Установи, що щоденно працюють із зовнішніми відвідувачами, потребують точного контролю запланованих і фактичних візитів, швидкого пошуку інформації про осіб, відповідальних співробітників, події та історію перебування в приміщенні. Паперові журнали або розрізнені електронні таблиці не забезпечують належної оперативності, ускладнюють формування звітності, створюють ризики дублювання даних і не дають змоги гнучко розмежовувати доступ між адміністраторами, операторами та працівниками. Використання вебтехнологій дає змогу централізувати ці процеси, забезпечити доступ через браузер, реалізувати перевірку даних, фільтрацію, облік статусів і формування звітів. Для побудови такої системи доцільним є застосування ASP.NET Core MVC, оскільки цей підхід підтримує розділення логіки обробки запитів, моделей даних і подань користувацького інтерфейсу [1]. Збереження даних у локальній базі SQLite через Entity Framework Core дає можливість поєднати реляційну модель даних із зручним механізмом доступу до сутностей програмного застосунку [2].

Мета роботи полягає у розробці вебсистеми керування відвідувачами з використанням ASP.NET Core, яка забезпечує облік відвідувачів, співробітників, запланованих візитів і подій, підтримує розмежування прав доступу, фіксацію входу та виходу, пошук, фільтрацію й формування звітної інформації.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– здійснити аналіз предметної області керування відвідувачами, визначити типові проблеми обліку візитів і обґрунтувати потребу у створенні вебсистеми;

– сформувати вимоги до програмного забезпечення з урахуванням ролей користувачів, сценаріїв реєстрації відвідувачів, планування візитів, роботи з подіями та формування звітів;

– спроєктувати структуру вебзастосунку, модель даних і логіку взаємодії основних сутностей системи;

– обґрунтувати вибір технологій ASP.NET Core MVC, Entity Framework Core та SQLite для реалізації поставленого завдання;

– реалізувати основні модулі системи, зокрема облік відвідувачів, працівників, візитів, подій, користувачів і звітності;

– перевірити працездатність програмного продукту, коректність обмеження доступу, обробки статусів візитів і збереження даних.

Об’єктом дослідження є процеси реєстрації, супроводу, контролю та аналізу відвідувань установи.

Предметом дослідження є методи, моделі та програмні засоби розробки вебсистеми керування відвідувачами на основі ASP.NET Core MVC, Entity Framework Core і SQLite.

Практична цінність розробки полягає у створенні вебзастосунку, який може використовуватися для централізованого ведення бази відвідувачів і співробітників, планування візитів, контролю входу та виходу, роботи із запрошеннями, обліку подій і підготовки звітів. Запропоноване рішення зменшує обсяг ручної роботи, підвищує точність обліку та дає змогу швидше отримувати інформацію про поточні й завершені відвідування.

# РОЗДІЛ 1
# АНАЛІЗ МЕТОДІВ РОЗРОБКИ СИСТЕМИ КЕРУВАННЯ ВІДВІДУВАЧАМИ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

## 1.1 Аналіз сучасного стану проблеми

Процес керування відвідувачами охоплює комплекс взаємопов'язаних операцій, серед яких попереднє планування візитів, реєстрація фактичного прибуття відвідувачів, визначення відповідального працівника або підрозділу, фіксація часу входу та виходу, контроль поточного статусу відвідування, а також формування звітної й аналітичної інформації. Належна організація цих процесів дає змогу підвищити рівень контролю за переміщенням відвідувачів та забезпечити ефективний облік усіх подій, пов'язаних із їх перебуванням на території організації.

У найпростішому випадку зазначені операції можуть виконуватися за допомогою паперового журналу або електронної таблиці. Проте такий підхід має низку суттєвих обмежень, зокрема не забезпечує швидкого пошуку необхідної інформації, підтримки рольового доступу до даних, ведення історії змін, централізованого аналізу накопичених відомостей та зручної роботи з подіями в реальному часі. Крім того, ручне ведення обліку підвищує ризик помилок і ускладнює контроль актуальності даних. Саме тому доцільним є використання спеціалізованих програмних засобів, які автоматизують процеси реєстрації, обліку та супроводу відвідувачів.

Одним із поширених рішень у цій сфері є Envoy Visitors. Система позиціонується як сучасна платформа для цифрової реєстрації відвідувачів, яка надає можливість налаштовувати форми та поля реєстрації відповідно до потреб організації, вести централізований журнал відвідувань, створювати перепустки, використовувати інтеграції з іншими інформаційними системами, надсилати автоматичні сповіщення та зберігати дані для подальшого аудиту й аналізу [3]. На рисунку 1.1 наведено фрагмент інтерфейсу Envoy Visitors, який демонструє

функціональність системи, пов’язану з попереднім погодженням та організацією відвідування.

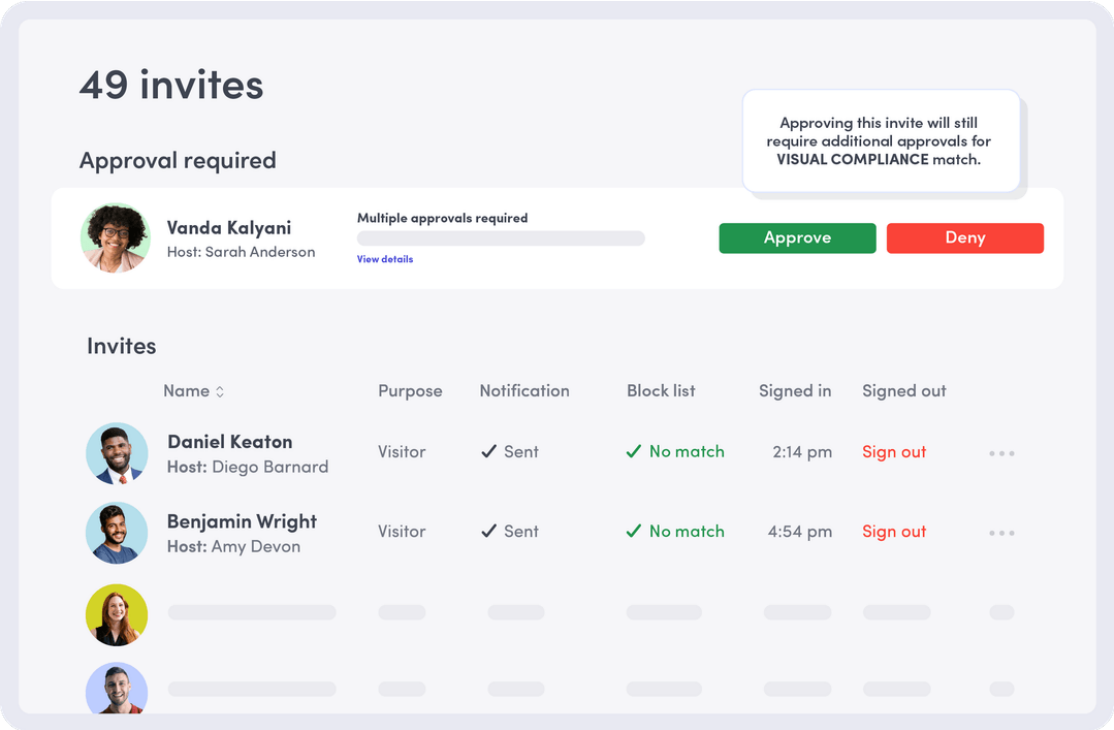



Рисунок 1.1 – Фрагмент інтерфейсу Envoy Visitors [3]

Перевагою Envoy Visitors є комплексність: система поєднує реєстрацію гостей, налаштування сценаріїв входу, сповіщення відповідальних осіб, цифрові документи, бейджі, інтеграції з іншими сервісами та централізований журнал. Недоліком для невеликої установи є орієнтація на хмарну корпоративну інфраструктуру, залежність від зовнішнього сервісу та надмірність частини функцій, які не завжди потрібні для локального обліку відвідувачів, працівників, візитів і подій.

Іншим аналогом є Sign In App. Його модуль Activity dashboard призначений для відображення всіх присутніх на об’єкті в реальному часі: відвідувачів, працівників, підрядників і студентів. У системі передбачено перегляд статусів, пошук, фільтри, попередні реєстрації, роботу з різними майданчиками та використання даних під час евакуації або інцидентів [4]. На рисунку 1.2 показано приклад панелі активності Sign In App.

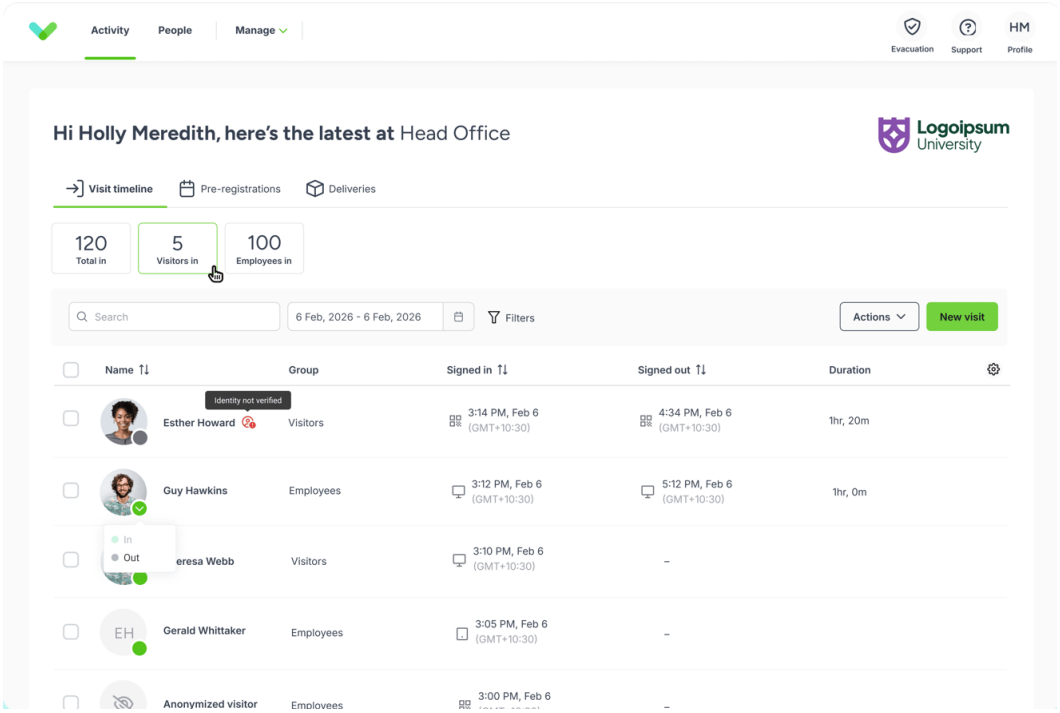



Рисунок 1.2 – Панель активності Sign In App [4]

Перевагою Sign In App є розвинена панель присутності, що дає змогу швидко побачити, хто перебуває на об'єкті, з якою метою та в якому статусі. Система також підтримує роботу з кількома локаціями, фільтрацію, попередню реєстрацію та сценарії безпеки. Водночас її функціональність виходить за межі вузького завдання обліку відвідувачів, оскільки охоплює працівників, підрядників, бронювання ресурсів, доставку та інші процеси. Для розроблюваної системи це означає, що доцільно зберегти ідею зручного журналу та статусів, але не ускладнювати програму зайвими модулями.

Ще одним прикладом є Eptura Visitor. Система орієнтована на керування повним циклом перебування відвідувача у будівлі: попередню реєстрацію, QR коди, інтеграцію з контролем доступу, перевірку за списками, формування бейджів, облік присутніх, аварійні сценарії та аналітику відвідувань [5]. Порівняно з попередніми аналогами Eptura Visitor сильніше пов'язана з фізичною інфраструктурою будівлі, тому є корисною для великих підприємств, але може бути складною для впровадження у невеликій організації без систем контролю доступу.

У таблиці 1.1 наведено порівняння розглянутих аналогів із розроблюваною вебсистемою керування відвідувачами.

Таблиця 1.1 – Порівняння аналогів систем керування відвідувачами

| Критерій | Envoy Visitors | Sign In App | Eptura Visitor | Розроблювана система |
|---|---|---|---|---|
| Основне призначення | Цифровий журнал відвідувачів, погодження, бейджі та інтеграції | Панель присутності для відвідувачів, працівників і підрядників | Корпоративне керування відвідувачами з інтеграцією контролю доступу | Веб-облік відвідувачів, працівників, візитів, подій і звітів |
| Переваги | Розвинені інтеграції, аудит, сповіщення, налаштування сценаріїв | Зручна панель реального часу, фільтри, підтримка кількох локацій | QR-коди, бейджі, аналітика, зв’язок із фізичною безпекою | Простота, локальна база SQLite, ролі, відповідність конкретним процесам |
| Недоліки | Залежність від хмарного сервісу та надмірність для малих установ | Широка платформа, частина модулів не стосується лише відвідувачів | Складність впровадження без інфраструктури контролю доступу | Менше корпоративних інтеграцій, але вища відповідність навчальній задачі |
| Корисні ідеї для розробки | Журнал візитів, статуси, підготовка до аудиту | Огляд присутніх, пошук і фільтрація | Попередня реєстрація та контроль безпеки | Модулі візитів, подій, ролей, звітності та експорту |

Аналіз аналогів показав, що сучасні системи керування відвідувачами найчастіше розвиваються у напрямі хмарних платформ із великою кількістю інтеграцій, мобільних сценаріїв, сповіщень, бейджів і засобів безпеки. Для кваліфікаційної роботи доцільно зосередитися на більш компактному рішенні, яке реалізує базові процеси установи: облік відвідувачів, працівників, запланованих і фактичних візитів, подій, учасників подій, статусів відвідування, ролей користувачів і звітності. Такий підхід дає змогу уникнути надмірної складності, але залишити головні переваги цифрового журналу: структурованість даних, пошук, контроль доступу та формування звітів.

## 1.2 Постановка завдання на кваліфікаційну роботу бакалавра

В результаті аналізу процесів реєстрації та супроводу відвідувачів було встановлено, що розроблювана система повинна не лише зберігати довідкову інформацію, а й підтримувати повний цикл роботи з візитом: від створення запису й визначення відповідального співробітника до фіксації фактичного входу, виходу та подальшого формування звітів. Оскільки система орієнтована на декілька груп користувачів, під час постановки завдання необхідно врахувати різні рівні доступу, вимоги до цілісності даних, зручність пошуку інформації та можливість роботи з подіями, у яких бере участь багато відвідувачів. На основі цього було сформовано наступні завдання:

– здійснити аналіз предметної області керування відвідувачами, визначити типові проблеми обліку візитів, зокрема дублювання записів, складність пошуку даних, відсутність єдиного журналу подій і недостатній контроль фактичного перебування відвідувачів у приміщенні, а також обґрунтувати потребу у створенні вебсистеми;

– сформувати вимоги до програмного забезпечення з урахуванням ролей адміністратора, оператора та співробітника, сценаріїв створення записів про відвідувачів, планування візитів, призначення відповідальних осіб, роботи з подіями, використання кодів запрошень, фільтрації інформації та формування звітів;

– спроєктувати структуру вебзастосунку, модель даних і логіку взаємодії основних сутностей системи, визначивши зв’язки між відвідувачами, співробітниками, візитами, подіями, учасниками подій і користувацькими обліковими записами;

– обґрунтувати вибір технологій ASP.NET Core MVC, Entity Framework Core та SQLite для реалізації поставленого завдання, врахувавши потребу у вебінтерфейсі, розмежуванні відповідальності між компонентами, роботі з реляційними даними та простому розгортанні системи;

– реалізувати основні модулі системи, зокрема облік відвідувачів, працівників, візитів, подій, учасників подій, користувачів і звітності, а також забезпечити автентифікацію, авторизацію за ролями, генерацію запрошень, пошук, фільтрацію й експорт звітних даних;

– перевірити працездатність програмного продукту, коректність обмеження доступу, обробки статусів візитів, фіксації часу входу та виходу, збереження даних у базі, формування звітів і роботи основних сценаріїв користувача.

# РОЗДІЛ 2
# СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

## 2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Під час аналізу предметної області встановлено, що система керування відвідувачами повинна підтримувати два взаємопов’язані процеси: індивідуальні візити до співробітників установи та участь відвідувачів у запланованих подіях.

У першому випадку центральним об’єктом є візит, що пов’язує відвідувача, відповідального співробітника, мету, запланований час, фактичний час входу й виходу та поточний статус перебування. Такий запис має відображати не лише факт запланованого відвідування, а й подальші зміни його стану під час реального проходження відвідувача через установу.

У другому випадку система має зберігати інформацію про подію, організатора, місце проведення, часові межі, максимальну кількість учасників і окремі записи участі кожного відвідувача.

Основними користувачами системи визначено адміністратора, оператора та співробітника.

Адміністратор відповідає за ведення співробітників і облікових записів, оператор виконує поточну реєстрацію відвідувачів, індивідуальних візитів та учасників подій, а співробітник переглядає пов’язані з ним візити та події.

Такий розподіл дає змогу не відкривати всі дані кожному користувачу, але зберегти достатню гнучкість для щоденної роботи установи. Крім того, рольова модель допомагає чітко відокремити адміністративні дії, операції реєстрації та перегляд персонально пов’язаних записів.

Основні варіанти використання системи для адміністратора, оператора та співробітника наведено на рисунку 2.1.

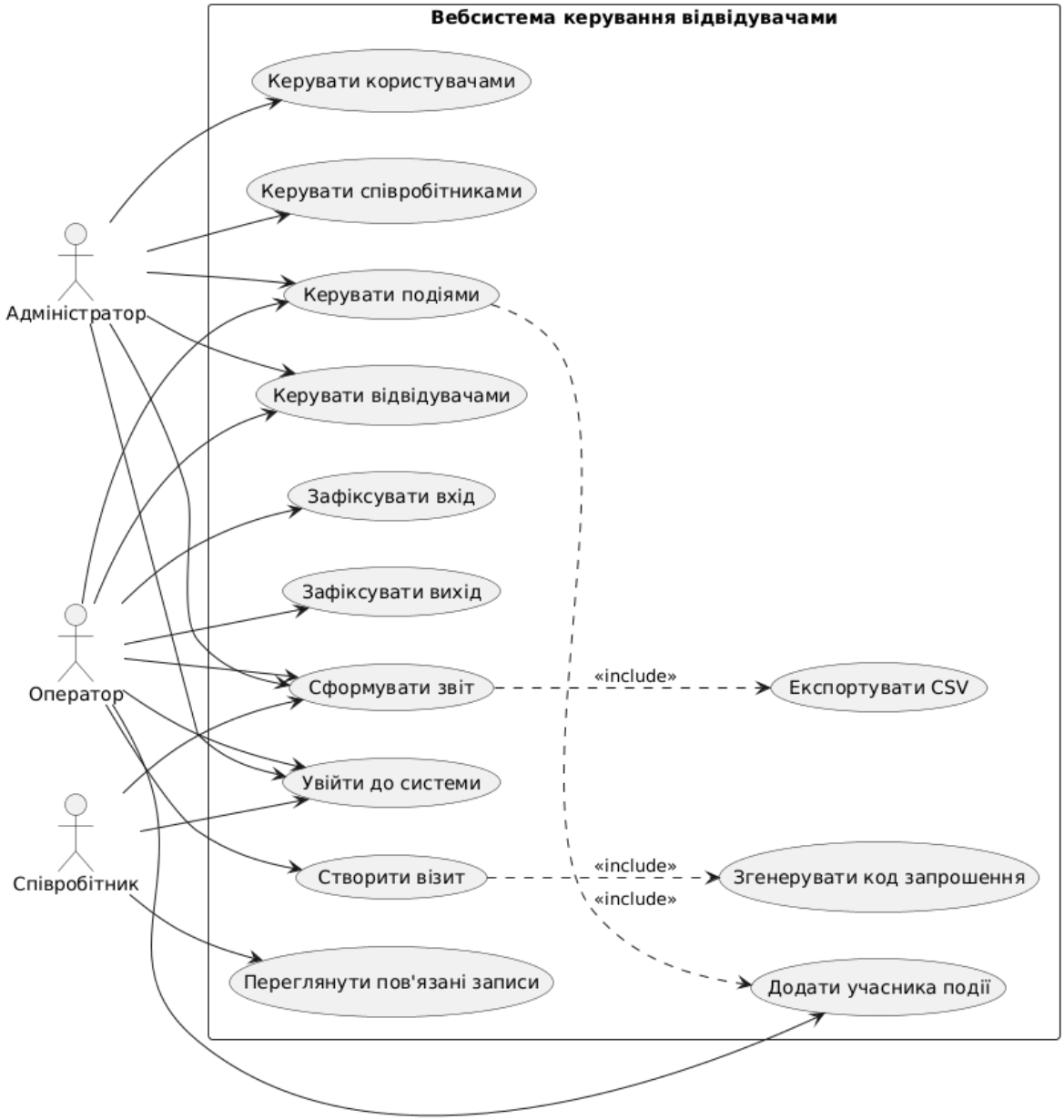



Рисунок 2.1 – UML-діаграма варіантів використання вебсистеми керування відвідувачами

З урахуванням визначених сценаріїв сформовано функціональні вимоги до програмного забезпечення. Система повинна надавати доступ лише після введення коректного логіна і пароля активного користувача, а також розмежовувати права адміністратора, оператора та співробітника під час відкриття модулів і виконання операцій. Для адміністратора передбачається керування співробітниками, користувачами та основними довідниками, для

оператора – реєстрація відвідувачів, візитів і учасників подій, для співробітника – перегляд пов’язаних із ним записів.

До основних функціональних можливостей системи належить облік відвідувачів із підтримкою створення, перегляду, редагування, видалення і пошуку записів. Для співробітників система повинна зберігати ПІБ, відділ, посаду, телефон та електронну адресу. Окремий блок вимог стосується планування візитів: система має створювати візит із прив’язкою до відвідувача, відповідального співробітника, мети та запланованої дати, а також фіксувати фактичний час входу і виходу зі зміною поточного статусу візиту.

Для роботи з подіями система повинна підтримувати створення подій, визначення організатора, місця проведення, часових меж, статусу і максимальної кількості учасників. Для кожного учасника події має створюватися окремий запис із кодом запрошення та можливістю фіксації прибуття і вибуття. Також система повинна забезпечувати пошук і фільтрацію візитів та подій за статусом, датою, відповідальною особою і текстовим запитом, а у звітній частині – відображати статистику за візитами й подіями та експортувати відібрані дані у CSV формат.

Крім функціональних можливостей, для системи визначено нефункціональні вимоги. Інтерфейс повинен бути доступним через браузер і містити окремі сторінки для основних довідників, операцій та звітів. Для забезпечення цілісності даних система має перевіряти наявність пов’язаних записів, коректність статусів, дат входу і виходу, а також не допускати дублювання учасника в межах однієї події.

Вимоги безпеки передбачають контроль доступу до сторінок за ролями та зберігання паролів користувачів у вигляді хешів. Для зручності роботи основні списки повинні підтримувати пошук, фільтрацію та сортування без ручного перегляду всіх записів. Підтримуваність системи забезпечується поділом логіки між моделями, контролерами, поданнями, сервісами та контекстом бази даних. Розгортання має залишатися простим: система повинна працювати з локальною реляційною базою SQLite без потреби в окремому сервері баз даних.

Проєктування програмного забезпечення виконано на основі архітектурного підходу MVC, який передбачає відокремлення моделей предметної області, контролерів обробки запитів і подань користувацького інтерфейсу [1]. У межах розроблюваної системи моделі описують сутності відвідувачів, співробітників, візитів, подій, учасників подій і користувачів; контролери реалізують бізнеслогіку відповідних модулів; подання формують HTML сторінки для взаємодії з користувачем.

Для збереження даних використано Entity Framework Core з провайдером SQLite, що дає змогу працювати з таблицями бази даних через класи C# і контекст ApplicationDbContext [2]. У контексті визначено набори Visitors, Employees, Visits, AppUsers, Events та EventParticipants. Така структура забезпечує пряме відображення основних об’єктів предметної області на таблиці бази даних і спрощує виконання запитів з урахуванням зв’язків між ними.

Структурно-функціональна схема розроблюваної системи наведена на рисунку 2.2. Вона показує взаємодію ролей користувачів із вебінтерфейсом, контролерами MVC, службовими класами, механізмами захисту та базою даних.

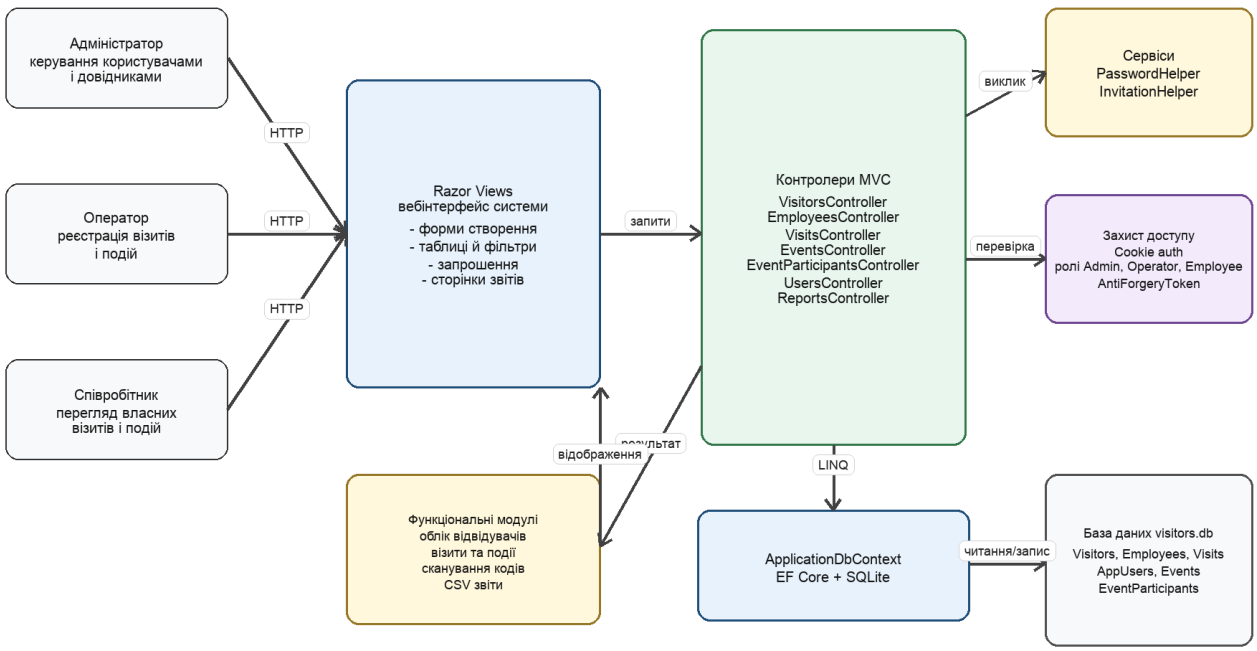



Рисунок 2.2 – Структурно-функціональна схема вебсистеми керування відвідувачами

Зі схеми видно, що користувачі працюють із системою через Razor подання, а основна бізнеслогіка зосереджена в контролерах. Контролери застосовують рольові обмеження, викликають допоміжні сервіси для хешування паролів і генерації запрошень, а для збереження та вибірки даних використовують ApplicationDbContext.

Основні сутності проєктної моделі наведено в таблиці 2.1.

Таблиця 2.1 – Основні сутності проєктної моделі системи

| Сутність | Призначення | Ключові дані |
|---|---|---|
| Visitor | зберігає інформацію про особу, яка відвідує установу | ПІБ, телефон, email, номер документа, організація, примітки |
| Employee | описує співробітника установи, до якого може бути призначено візит або подію | ПІБ, відділ, посада, телефон, email |
| Visit | фіксує індивідуальний візит відвідувача до співробітника | відвідувач, співробітник, мета, запланована дата, час входу, час виходу, статус, код запрошення |
| Event | описує подію, у якій можуть брати участь декілька відвідувачів | назва, опис, місце, початок, завершення, організатор, статус, код події, ліміт учасників |
| EventParticipant | пов’язує відвідувача з конкретною подією та фіксує його участь | подія, відвідувач, код учасника, статус участі, час входу, час виходу, примітки |
| AppUser | містить обліковий запис користувача системи | логін, хеш пароля, ПІБ, роль, активність, прив’язаний співробітник |

Зв’язки між основними сутностями предметної області показано на UML-діаграмі класів, наведеній на рисунку 2.3.

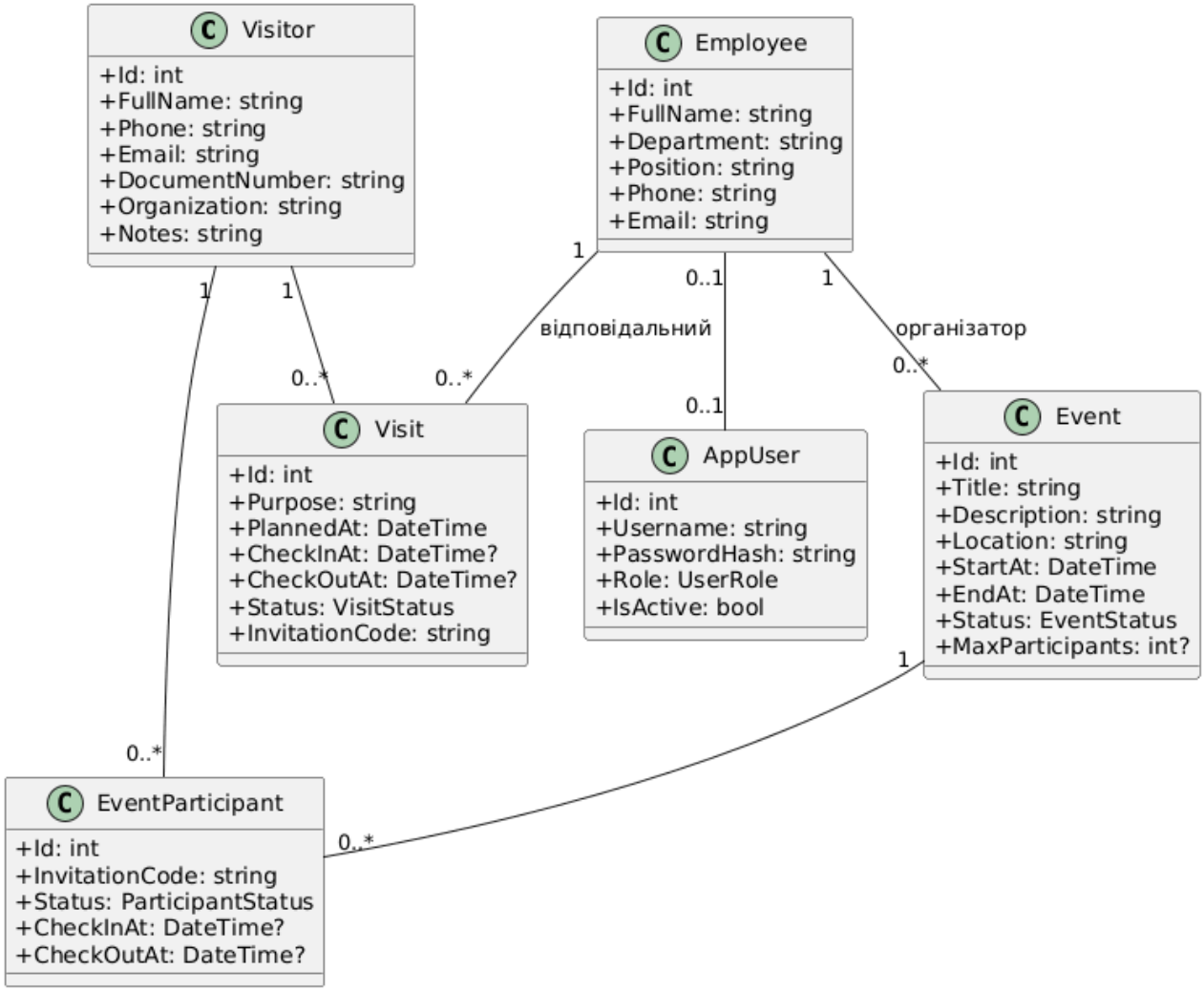



Рисунок 2.3 – UML-діаграма класів предметної області системи

Логіка доступу побудована так, щоб адміністратор мав повний контроль над користувачами та співробітниками, оператор виконував реєстраційні операції, а співробітник працював лише з тими візитами і подіями, які належать йому як відповідальній особі або організатору. Такий підхід дає змогу розмежувати службові, операційні та переглядові функції системи, не ускладнюючи щоденну роботу користувачів.

Для цього під час виконання запитів контролери додатково перевіряють ідентифікатор співробітника, пов’язаний з поточним обліковим записом. Якщо користувач не має прав на перегляд або зміну певного запису, система не повинна надавати доступ до відповідної дії.

Сценарій роботи з індивідуальним візитом передбачає створення запису, автоматичну генерацію коду запрошення, перегляд запрошення, фіксацію входу після прибуття відвідувача та фіксацію виходу після завершення перебування. На етапі створення візиту система пов’язує між собою відвідувача, відповідального співробітника, мету відвідування і запланований час прибуття. Згенерований код запрошення використовується як зручний ідентифікатор, за яким оператор може швидко знайти потрібний запис без ручного перегляду всього списку візитів.

Якщо користувач вводить або сканує код запрошення, система знаходить відповідний візит і, за умови наявності прав доступу, встановлює час входу та статус перебування. Після цього запис відображає фактичний початок перебування відвідувача в установі. Після фіксації виходу система встановлює час завершення перебування, а запис переходить у завершений стан.

Послідовність фіксації входу відвідувача за кодом запрошення наведено на рисунку 2.4.

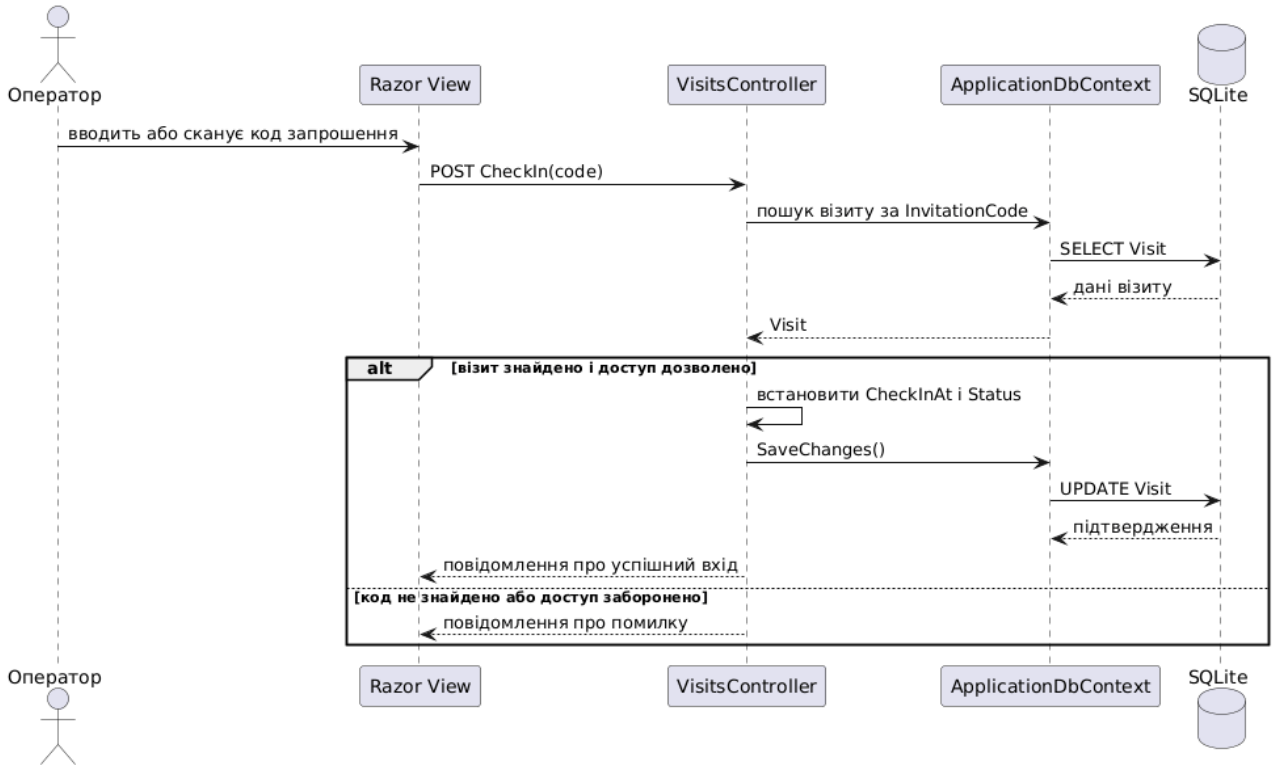



Рисунок 2.4 – UML-діаграма послідовності фіксації входу відвідувача

Життєвий цикл індивідуального візиту від створення запису до завершення або скасування показано на рисунку 2.5.

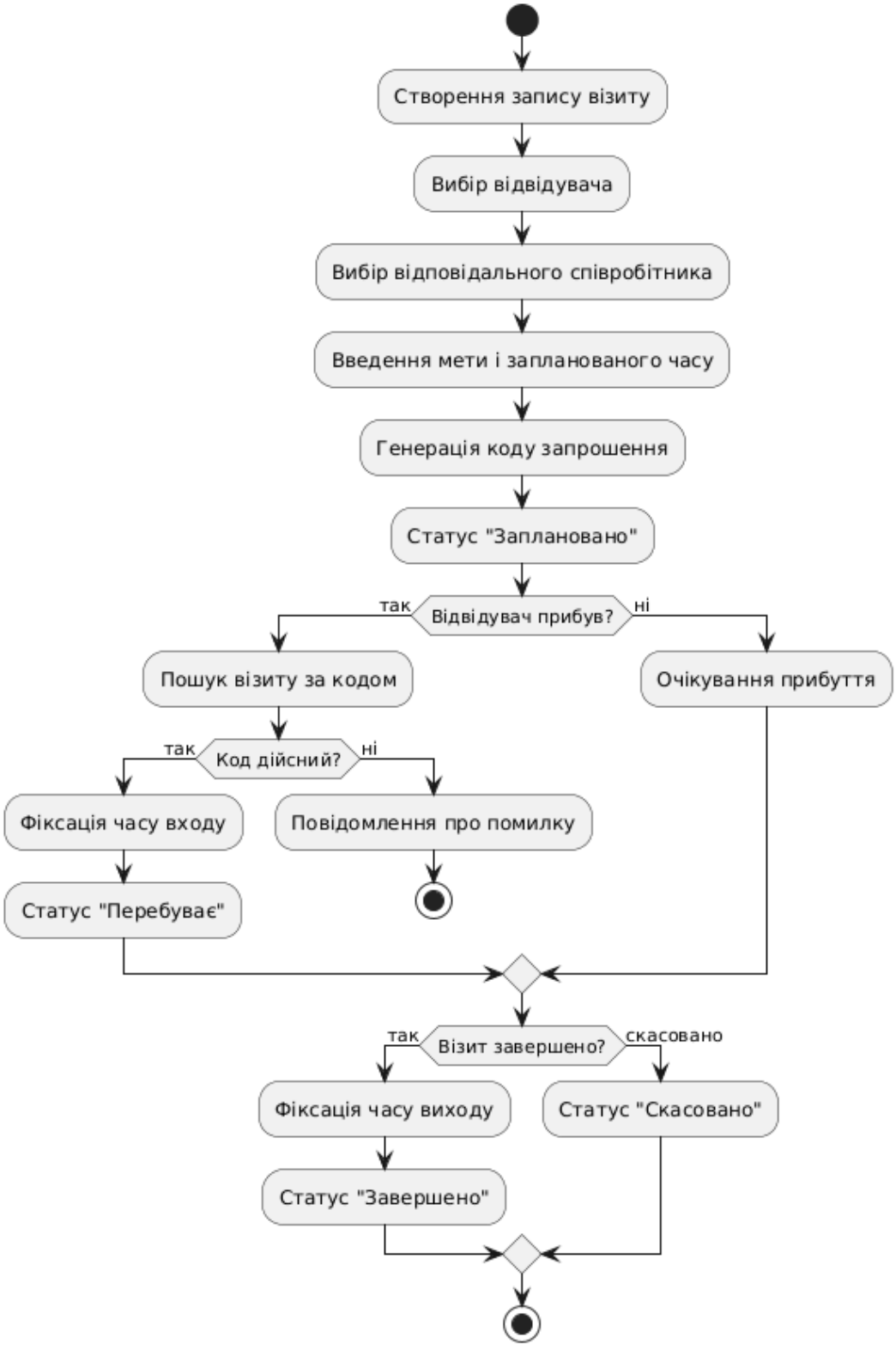



Рисунок 2.5 – UML-діаграма активності життєвого циклу візиту

Сценарій роботи з подією відрізняється тим, що одна подія може містити багато учасників. Для кожного учасника створюється окремий запис EventParticipant з власним кодом запрошення і статусом участі. Під час додавання учасника система перевіряє, чи не був цей відвідувач уже доданий до події, а також контролює максимальну кількість учасників, якщо такий ліміт задано.

Звітна частина системи призначена для швидкого аналізу накопичених даних. Для візитів передбачено фільтрацію за датами, співробітником, статусом і пошуковим рядком; для подій – за датами, організатором, статусом і текстовим запитом. У звітах обчислюються підсумкові показники, зокрема кількість запланованих, активних, завершених і скасованих записів, а також кількість учасників подій і прибулих осіб. Для подальшого опрацювання дані експортуються у CSV файл.

Отже, у межах підрозділу визначено ролі користувачів, функціональні та нефункціональні вимоги, основні сутності предметної області й загальну архітектуру програмного забезпечення. Запропонована структура забезпечує відповідність поставленим завданням: система зберігає дані про відвідувачів і співробітників, підтримує життєвий цикл візитів, дає змогу працювати з подіями, обмежує доступ за ролями та формує звітну інформацію.

## 2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Вибір засобів реалізації вебсистеми керування відвідувачами здійснювався з урахуванням визначених у підрозділі 2.1 вимог: наявності браузерного інтерфейсу, роботи з реляційними даними, підтримки ролей користувачів, фільтрації, формування звітів, генерації кодів запрошень і простого локального розгортання. Для такої системи важливо, щоб технологічний стек був достатньо надійним для серверної обробки даних, але не вимагав складної інфраструктури.

Для побудови серверної частини розглянуто декілька підходів у межах ASP.NET Core: MVC, Razor Pages і Minimal APIs. ASP.NET Core MVC забезпечує розділення моделей, контролерів і подань, що є зручним для системи з багатьма сутностями та повторюваними операціями створення, перегляду, редагування, видалення, пошуку і фільтрації [1]. Razor Pages орієнтовані на сторінкові сценарії та можуть бути продуктивними для простіших вебінтерфейсів [6]. Minimal APIs добре підходять для компактних HTTP API, але в розроблюваній системі основним є не лише обмін JSON даними, а повноцінний вебінтерфейс із поданнями, формами й рольовими сторінками [7].

Порівняння підходів до побудови вебзастосунку наведено в таблиці 2.2.

Таблиця 2.2 – Порівняння підходів до побудови вебзастосунку

| Підхід | Переваги | Недоліки щодо поставленого завдання | Висновок |
|---|---|---|---|
| ASP.NET Core MVC | чітке розділення контролерів, моделей і подань; зручність для CRUD модулів; підтримка маршрутизації та авторизації | потребує більшої кількості файлів, ніж найпростіший сторінковий підхід | обрано як основний підхід для системи |
| Razor Pages | зручні для сторінок із локальною логікою; менше структурних файлів для простих сценаріїв | для великої кількості пов’язаних сутностей може бути менш наочним розподіл відповідальності | доцільні для простіших сторінкових застосунків |
| Minimal APIs | компактний опис HTTP кінцевих точок; зручність для сервісних API | не орієнтовані на класичний інтерфейс з HTML поданнями і формами | не є основним варіантом для цієї роботи |

Для доступу до даних обрано Entity Framework Core, оскільки він дозволяє працювати з таблицями бази даних через класи предметної області, виконувати запити до пов’язаних сутностей і використовувати міграції під час розвитку структури бази. У системі це особливо важливо для сутностей Visitor, Employee, Visit, Event, EventParticipant і AppUser, між якими є зв’язки один до багатьох та перевірки належності записів поточному користувачу.

На рівні системи керування базою даних розглянуто SQLite, SQL Server LocalDB і PostgreSQL, оскільки всі ці засоби можуть використовуватися з .NET та реляційною моделлю даних. Остаточний вибір здійснено з урахуванням простоти розгортання, підтримки Entity Framework Core, достатності для локальної навчальної системи та можливості подальшого розвитку проєкту.

SQLite використовується через провайдер Entity Framework Core і не потребує окремого серверного процесу, що спрощує запуск навчального проєкту та перенесення бази разом із застосунком [2]. SQL Server LocalDB надає ширші можливості для розробки у середовищі Microsoft, але створює жорсткішу прив’язку до локальної інсталяції SQL Server. PostgreSQL доцільний для багатокористувацьких промислових систем, однак для поставленої задачі його розгортання є надмірним.

Під час вибору засобу зберігання даних розглянуто SQLite, SQL Server LocalDB і PostgreSQL. SQLite є найдоцільнішим варіантом для поточної версії системи, оскільки зберігає базу даних у локальному файлі, просто розгортається разом із застосунком, підтримується через Entity Framework Core і є достатнім для навчальної вебсистеми керування відвідувачами. Основним обмеженням SQLite є менша придатність до роботи з великою кількістю одночасних користувачів, однак для локальної версії системи це обмеження не є критичним.

SQL Server LocalDB також може бути використаний для локальної розробки, оскільки добре інтегрується з екосистемою Microsoft і має потужніші механізми керування даними. Водночас його застосування потребує встановлення відповідного компонента SQL Server, а перенесення проєкту на інший комп’ютер стає складнішим. Тому SQL Server LocalDB доцільно

розглядати як можливий варіант майбутнього розширення системи, але не як основне рішення для поточної реалізації.

PostgreSQL забезпечує високу надійність, масштабованість і розвинені засоби адміністрування, що робить його придатним для серверного розгортання та роботи з великою кількістю користувачів. Проте для розроблюваної локальної системи такий варіант є надмірним, оскільки потребує окремого сервера баз даних і додаткового адміністрування. З огляду на вимоги простого запуску, локального зберігання та мінімальної інфраструктури для реалізації обрано SQLite.

Для організації доступу користувачів використано cookie-автентифікацію ASP.NET Core [8]. Такий підхід відповідає класичному вебсценарію, у якому після успішного входу користувач отримує автентифікаційний cookie, а подальші запити виконуються з урахуванням його ідентичності та ролі. У системі використано ролі Admin, Operator і Employee, що відповідає підходу рольової авторизації в ASP.NET Core MVC [9]. Контролери застосовують рольові обмеження для відкриття відповідних сторінок, а для співробітника додатково перевіряється зв’язок облікового запису з конкретним EmployeeId. Використання повного ASP.NET Core Identity для цієї роботи не є обов’язковим, оскільки система має просту власну модель AppUser і не потребує реєстрації зовнішніх користувачів, підтвердження email чи складної політики профілів.

Для побудови користувацького інтерфейсу застосовано HTML подання Razor, синтаксис яких дозволяє поєднувати HTML розмітку з C# кодом на стороні сервера [10]. У межах розроблюваної системи Razor подання використовуються для формування сторінок відвідувачів, співробітників, візитів, подій, користувачів і звітів. Через такі подання користувач працює з таблицями, формами створення та редагування записів, кнопками дій, фільтрами, повідомленнями про результат операції та сторінками запрошень. У проєкті також використовується Bootstrap як готовий набір стилів і компонентів для адаптивних вебсторінок [11]. Це дає змогу швидше сформувати зрозумілі

таблиці, форми, кнопки, повідомлення й навігаційні елементи без розроблення власної системи стилів з нуля.

Під час реалізації логіки системи використано набір прикладних алгоритмів. Для пошуку і фільтрації даних формується запит до відповідної таблиці з урахуванням текстового рядка, статусу, дати або відповідального співробітника. Для працівника з роллю Employee запит додатково обмежується його ідентифікатором, що запобігає перегляду чужих візитів і подій. Для формування запрошення використовується генерація короткого унікального коду з перевіркою його відсутності в наявних записах. Такий код застосовується під час відкриття запрошення, ручного введення або сканування для фіксації фактичного прибуття.

Алгоритм фіксації входу і виходу побудований на перевірці поточного стану запису. Якщо для візиту ще не встановлено час входу, система записує поточний час і переводить візит у стан перебування в приміщенні. Якщо час входу вже є, але час виходу відсутній, після завершення візиту записується час виходу і встановлюється завершений стан. Подібний підхід використано для учасників подій, де статус участі залежить від наявності часу входу та виходу. Завдяки цьому система не лише зберігає заплановані записи, а й відображає фактичний перебіг відвідування.

Для звітності застосовано алгоритм послідовного звуження набору даних. Спочатку обирається базова таблиця візитів або подій, потім до неї додаються фільтри за датами, статусом, відповідальною особою та текстовим запитом. Після отримання результатів обчислюються підсумкові показники: загальна кількість записів, кількість запланованих, активних, завершених і скасованих елементів, а також кількість учасників подій. Для експорту формується CSV файл у кодуванні UTF-8; загальний формат CSV описано в RFC 4180 [12]. Це дає змогу відкривати результати у табличних редакторах і використовувати їх для подальшого аналізу.

Узагальнення обраних засобів і їх призначення наведено в таблиці 2.3.

Таблиця 2.3 – Обрані засоби реалізації вебсистеми

| Засіб | Призначення в системі |
|---|---|
| ASP.NET Core MVC | побудова серверної частини, маршрутизації, контролерів, моделей і подань |
| Entity Framework Core | доступ до реляційних даних через класи предметної області |
| SQLite | локальне збереження даних про відвідувачів, співробітників, візити, події, учасників і користувачів |
| Cookie authentication | підтримка входу користувача, збереження сеансу і рольового доступу |
| Razor views | формування HTML сторінок, таблиць, форм і повідомлень |
| Bootstrap | базове оформлення інтерфейсу та адаптивних елементів сторінок |
| CSV експорт | передавання звітних даних у форматі, придатному для подальшого аналізу |

Отже, для реалізації вебсистеми керування відвідувачами обрано ASP.NET Core MVC, Entity Framework Core, SQLite, cookie-автентифікацію, Razor подання, Bootstrap і CSV експорт. Такий набір засобів відповідає функціональним вимогам системи, забезпечує достатню простоту розгортання, зрозумілу структуру коду, підтримку рольового доступу та можливість подальшого розширення програмного продукту.

# РОЗДІЛ 3
# РОЗРОБКА СИСТЕМИ КЕРУВАННЯ ВІДВІДУВАЧАМИ

## 3.1 Практична реалізація об’єкта проєктування

Практична реалізація вебсистеми керування відвідувачами виконана як ASP.NET Core MVC застосунок з розділенням на моделі, контролери, подання, сервіси та контекст доступу до бази даних. Така структура дозволяє окремо підтримувати опис предметних сутностей, обробку HTTP запитів, побудову сторінок інтерфейсу і службові операції, пов’язані з авторизацією, запрошеннями та звітністю.

У процесі розроблення було створено проєкт VisitorManagementSystem, у якому реалізовано модулі відвідувачів, співробітників, візитів, подій, учасників подій, користувачів, профілю та звітів. Фрагмент роботи з вихідним кодом у середовищі Visual Studio показано на рисунку 3.1.

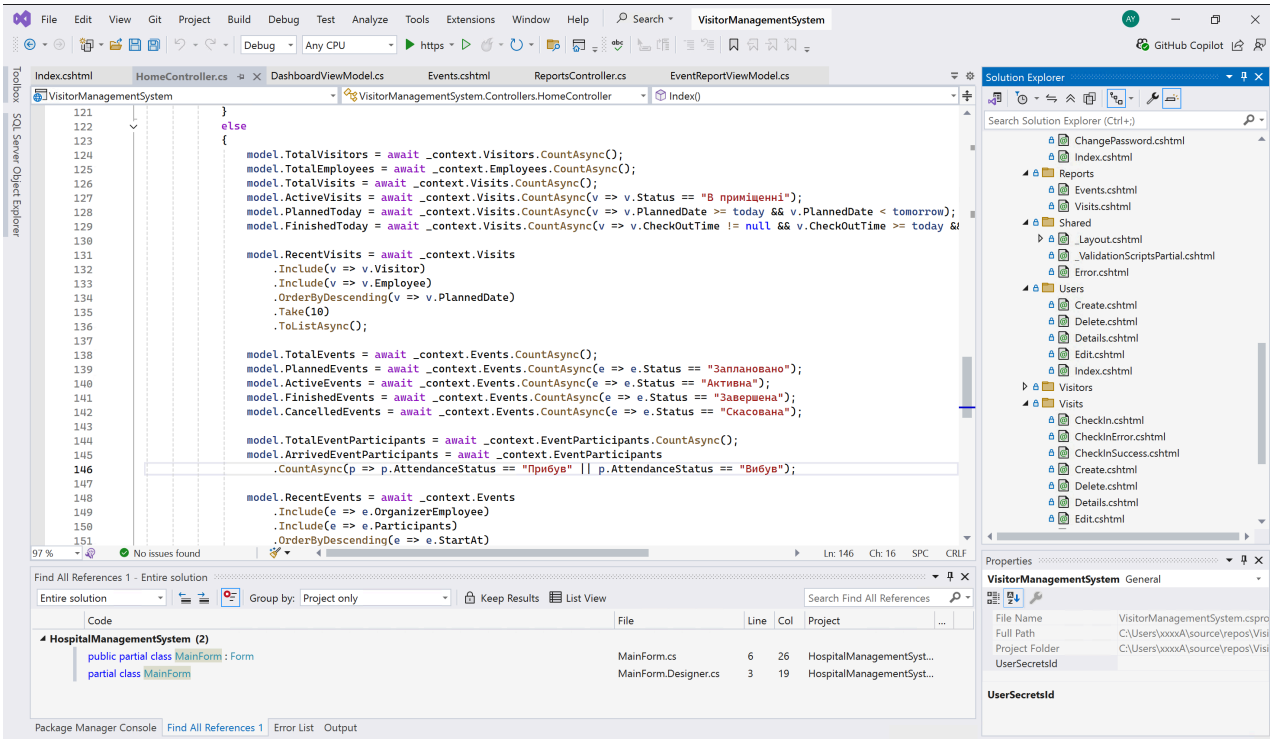



Рисунок 3.1 – Фрагмент розроблення вебсистеми керування відвідувачами

На рівні серверної частини застосунок побудовано за схемою MVC. Контролери приймають запити користувача, звертаються до ApplicationDbContext, виконують перевірки доступу та передають підготовлені дані у Razor подання. Моделі Visitor, Employee, Visit, Event, EventParticipant і AppUser описують основні сутності системи, а подання відповідають за відображення таблиць, форм створення, редагування, деталей і сторінок підтвердження.

У системі реалізовано рольову модель доступу. Адміністратор має повний доступ до користувачів, довідників, візитів, подій і звітів; оператор працює з реєстрацією відвідувачів, візитами та подіями; співробітник бачить лише ті візити або події, які пов’язані з ним.

Такий поділ реалізовано через атрибути Authorize у контролерах і додаткові перевірки поточного EmployeeId у методах перегляду, редагування та звітування.

Основою збереження даних є клас ApplicationDbContext. Він наслідує DbContext і містить набори сутностей, які відповідають основним таблицям системи. Саме через цей клас контролери отримують доступ до відвідувачів, співробітників, візитів, користувачів, подій та учасників подій.

Робота з базою даних виконується через Entity Framework Core. Контролери не формують SQL запити вручну, а використовують LINQ вирази, Include для завантаження пов’язаних сутностей та асинхронні методи ToListAsync, FirstOrDefaultAsync і SaveChangesAsync. Це спрощує супровід коду і дозволяє однаково працювати з різними таблицями предметної області.

Під час отримання списків візитів і подій система спочатку формує базовий IQueryable запит, а потім послідовно додає умови пошуку, фільтрації за статусом, датою та відповідальним співробітником. Такий підхід зручний тим, що один і той самий метод може обробити як повний список записів, так і звужений набір даних для конкретного користувача або звіту.

Фрагмент реалізації контексту бази даних наведено у лістингу 3.1.

Лістинг 3.1 – Опис контексту бази даних системи

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options) : base(options) {
    }

    // Набори сутностей, що відповідають таблицям бази даних.
    public DbSet<Visitor> Visitors { get; set; }
    public DbSet<Employee> Employees { get; set; }
    public DbSet<Visit> Visits { get; set; }
    public DbSet<AppUser> AppUsers { get; set; }
    public DbSet<Event> Events { get; set; }
    public DbSet<EventParticipant> EventParticipants { get; set; }
}
```

кінець лістингу 3.1

Для створення запрошень використано окремий службовий клас InvitationHelper. Він генерує короткий код з набору символів, який не містить візуально схожих літер і цифр. Для підвищення випадковості застосовано RandomNumberGenerator, після чого кожен байт перетворюється на символ майбутнього коду.

Код запрошення використовується у двох основних сценаріях: для індивідуального візиту та для участі у події. Під час створення запису система перевіряє, чи вже існує такий код у відповідній таблиці, і за потреби генерує новий. Це дає змогу використовувати код як короткий ідентифікатор для сторінки запрошення, QR переходу або ручного введення на сторінці сканування.

Запрошення містить не лише код, а й дані про відвідувача, відповідального співробітника, дату, мету візиту або назву події. У поданнях ці дані подаються у зручному для друку та демонстрації вигляді, а сам код може бути використаний для швидкої фіксації прибуття.

Фрагмент генерації коду запрошення наведено у лістингу 3.2.

Лістинг 3.2 – Генерація коду запрошення

```
public static string GenerateCode(int length = 8)
{
    const string chars = "ABCDEFGHJKLMNPQRSTUVWXYZ23456789";
    var result = new StringBuilder();
    var bytes = new byte[length];
    // Криптографічний генератор зменшує ризик повторення кодів.
    using var rng = RandomNumberGenerator.Create();
    rng.GetBytes(bytes);
    foreach (var b in bytes)
        result.Append(chars[b % chars.Length]);
    return result.ToString();
}
```

кінець лістингу 3.2

Окрема частина практичної реалізації пов’язана зі скануванням або введенням коду запрошення. Контролер приймає текст, за потреби витягує параметр code з URL, знаходить відповідний візит, перевіряє доступ співробітника і встановлює час входу. Розширений фрагмент цієї логіки подано у додатку А.

Сценарій реєстрації входу передбачає, що оператор або співробітник відкриває сторінку сканування, вводить код або передає повне посилання з параметром code. Контролер нормалізує отримане значення, знаходить відповідний запис у базі даних і, якщо відвідувач ще не був зареєстрований, встановлює CheckInTime та змінює статус на В приміщенні.

Для виходу використовується окрема дія CheckOut. Вона перевіряє, що візит уже має час входу, після чого встановлює CheckOutTime і переводить запис у завершений стан. Якщо спробувати виконати вихід без попереднього входу або

повторити вже виконану операцію, система не змінює дані й повертає користувачу повідомлення про помилку.

Основну частину обробки відсканованого коду наведено у лістингу 3.3.

Лістинг 3.3 – Обробка відсканованого коду візиту

```
public async Task<IActionResult> ProcessScannedCode(string? code)
{
    if (string.IsNullOrWhiteSpace(code))
        return View("CheckInError");
    code = code.Trim();
    var visit = await _context.Visits
        .Include(v => v.Visitor)
        .Include(v => v.Employee)
        .FirstOrDefaultAsync(v => v.InvitationCode == code);
    if (visit == null)
        return View("CheckInError");
    if (visit.CheckInTime == null)
    {
        visit.CheckInTime = DateTime.Now;
        visit.Status = "В приміщенні";
        await _context.SaveChangesAsync();
    }
    return View("CheckInSuccess", visit);
}
```

кінець лістингу 3.3

Для адміністративної частини реалізовано звіти за візитами та подіями. Звіти формуються на основі запитів Entity Framework Core з фільтрами за датою, співробітником, статусом і текстом пошуку. Результати можуть бути експортовані у CSV файл, що дозволяє відкрити їх у табличному редакторі.

Звіти за візитами містять кількість запланованих, активних, завершених і скасованих записів, а звіти за подіями додатково враховують кількість учасників

та кількість осіб, які фактично прибули. Для співробітника набір даних у звітах автоматично обмежується його власними візитами або подіями, тому один і той самий модуль звітності працює з урахуванням ролі користувача.

CSV експорт реалізовано як формування текстового вмісту з табличними даними. Для кожного значення застосовується екранування лапок, а у відповідь додається UTF8 BOM, щоб українські символи коректно відкривалися у табличних редакторах. Ім’я файла містить тип звіту, дату і час формування, що полегшує зберігання декількох експортованих результатів.

Фрагмент формування рядка CSV з екрануванням значень наведено у лістингу 3.4.

Лістинг 3.4 – Формування CSV рядків звіту

```
var sb = new StringBuilder();

sb.AppendLine("Відвідувач;Співробітник;Мета;Заплановано;Вхід;Вихід;Статус
");

foreach (var visit in visits)
{
    sb.AppendLine(string.Join(";",
        EscapeCsv(visit.Visitor?.FullName),
        EscapeCsv(visit.Employee?.FullName),
        EscapeCsv(visit.Purpose),
        EscapeCsv(visit.PlannedDate.ToString("dd.MM.yyyy HH:mm")),
        EscapeCsv(visit.CheckInTime?.ToString("dd.MM.yyyy HH:mm") ?? ""),
        EscapeCsv(visit.CheckOutTime?.ToString("dd.MM.yyyy HH:mm") ??
""),
        EscapeCsv(visit.Status)
    ));
}
```

кінець лістингу 3.4

Наведені фрагменти демонструють основні практичні складові розробки: опис моделі даних, генерацію запрошень, фіксацію факту входу за кодом і підготовку звітів. У сукупності ці модулі забезпечують роботу вебсистеми від створення облікових записів і візитів до контролю присутності та отримання підсумкової інформації.

Інтерфейс користувача реалізовано за допомогою Razor подань і спільного макета. У верхньому меню розміщено переходи до основних модулів, а всередині сторінок використовуються таблиці, форми, кнопки дій, фільтри та повідомлення про результат операції. Для повторюваних сценаріїв застосовано TempData повідомлення, які інформують користувача про успішне створення, оновлення, видалення або помилку виконання дії.

Після входу користувач потрапляє на інформаційну панель, де коротко показано основні показники системи: кількість відвідувачів, візитів, подій, користувачів і поточний стан записів, як показано на рисунку 3.2.

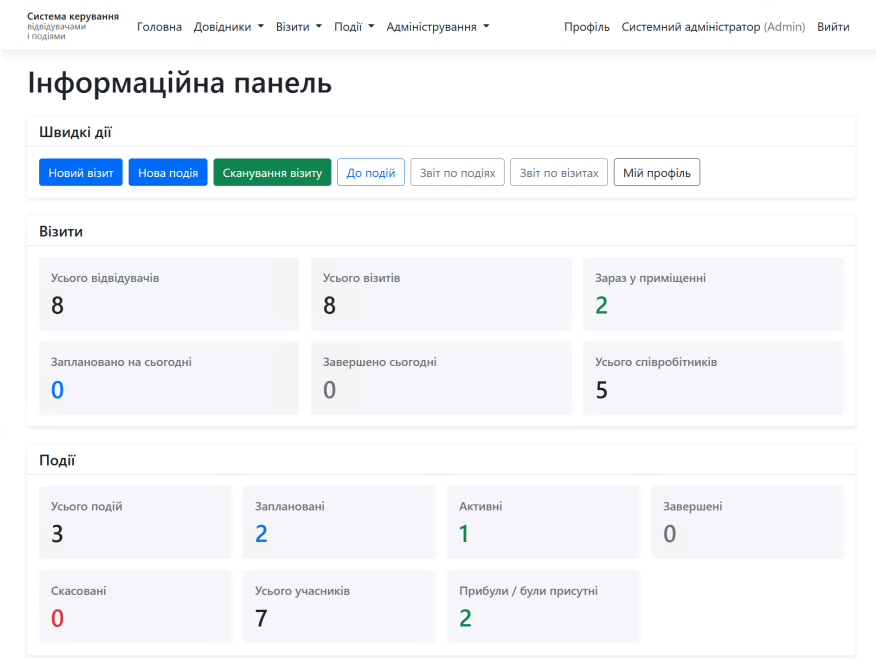



Рисунок 3.2 – Інформаційна панель вебсистеми керування відвідувачами

Операційна робота з індивідуальними візитами виконується через журнал, у якому доступні пошук, фільтрація за статусом і кнопки дій для перегляду,

редагування, входу, виходу та запрошення. Приклад сторінки візитів наведено на рисунку 3.3.

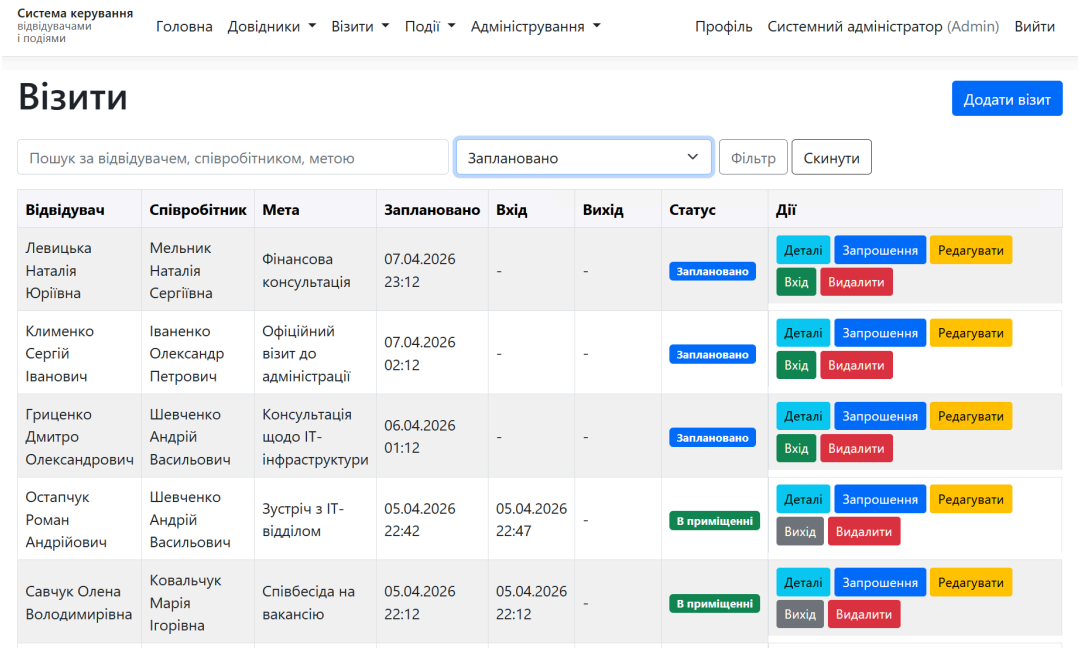



Рисунок 3.3 – Журнал візитів із фільтрацією та діями над записами

Для подій система формує сторінку запрошення з даними події та QR кодом, який може бути використаний для швидкої реєстрації учасника. Приклад такого запрошення наведено на рисунку 3.4.

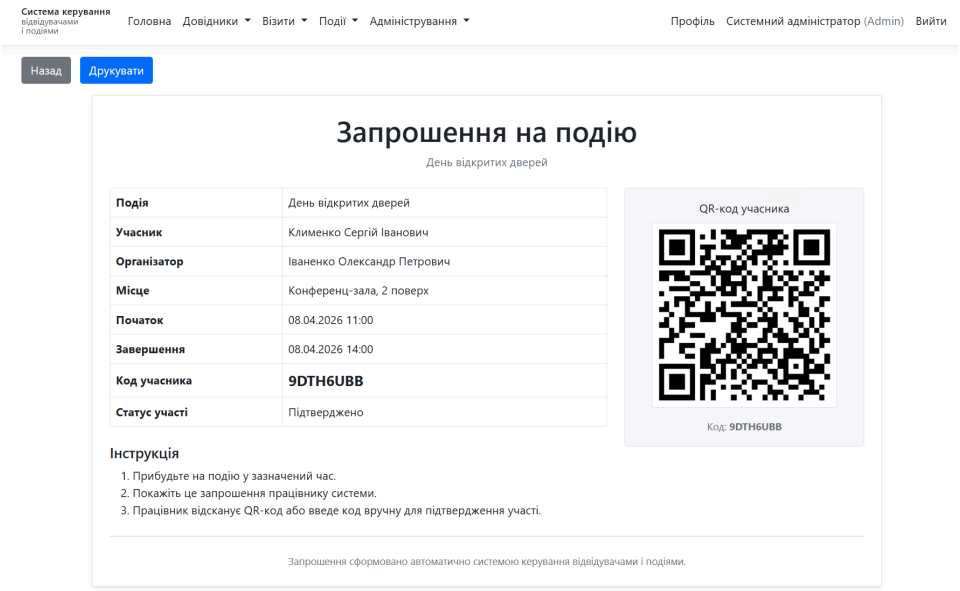



Рисунок 3.4 – Сторінка запрошення учасника події з QR кодом

Таким чином, практична реалізація охоплює не лише окремі CRUD сторінки, а повний робочий цикл системи: адміністрування користувачів, ведення довідників відвідувачів і співробітників, планування візитів, створення подій, запрошення учасників, фіксацію входу та виходу, обмеження доступу за ролями і формування звітів для подальшого аналізу. Усі ці функції об’єднані в межах єдиного вебзастосунку, тому користувач може виконувати основні операції без переходу між різними програмами або ручного ведення окремих журналів. Реалізовані модулі взаємодіють між собою через спільну базу даних, що забезпечує узгодженість інформації про відвідувачів, співробітників, візити, події та облікові записи.

### 3.2 Розробка бази даних

База даних вебсистеми керування відвідувачами реалізована засобами Entity Framework Core і зберігається у SQLite файлі visitors.db. Її структура відповідає предметній області системи: окремо зберігаються відвідувачі, співробітники, заплановані візити, користувачі системи, події та учасники подій. Такий поділ дає змогу не дублювати інформацію про одну й ту саму особу в різних модулях і пов’язувати записи через зовнішні ключі. Наприклад, один відвідувач може мати декілька індивідуальних візитів або бути учасником різних подій, а один співробітник може відповідати за прийом багатьох відвідувачів і водночас виступати організатором подій.

У проєкті структура бази даних описана класами моделей і класом ApplicationDbContext. Після створення або зміни моделей Entity Framework Core формує міграції, які використовуються для послідовного оновлення структури бази даних відповідно до змін у моделі [13]. У результаті логічна модель предметної області безпосередньо відображається у фізичну структуру бази даних. Це спрощує підтримку проєкту, оскільки зміни в сутностях системи можна послідовно переносити до бази даних без ручного створення SQL-

скриптів. Крім того, використання ORM дає змогу працювати з даними на рівні C# класів, зберігаючи при цьому реляційні зв’язки між основними таблицями.

Структуру основних таблиць і зв’язків бази даних розробленої системи наведено на рисунку 3.5.

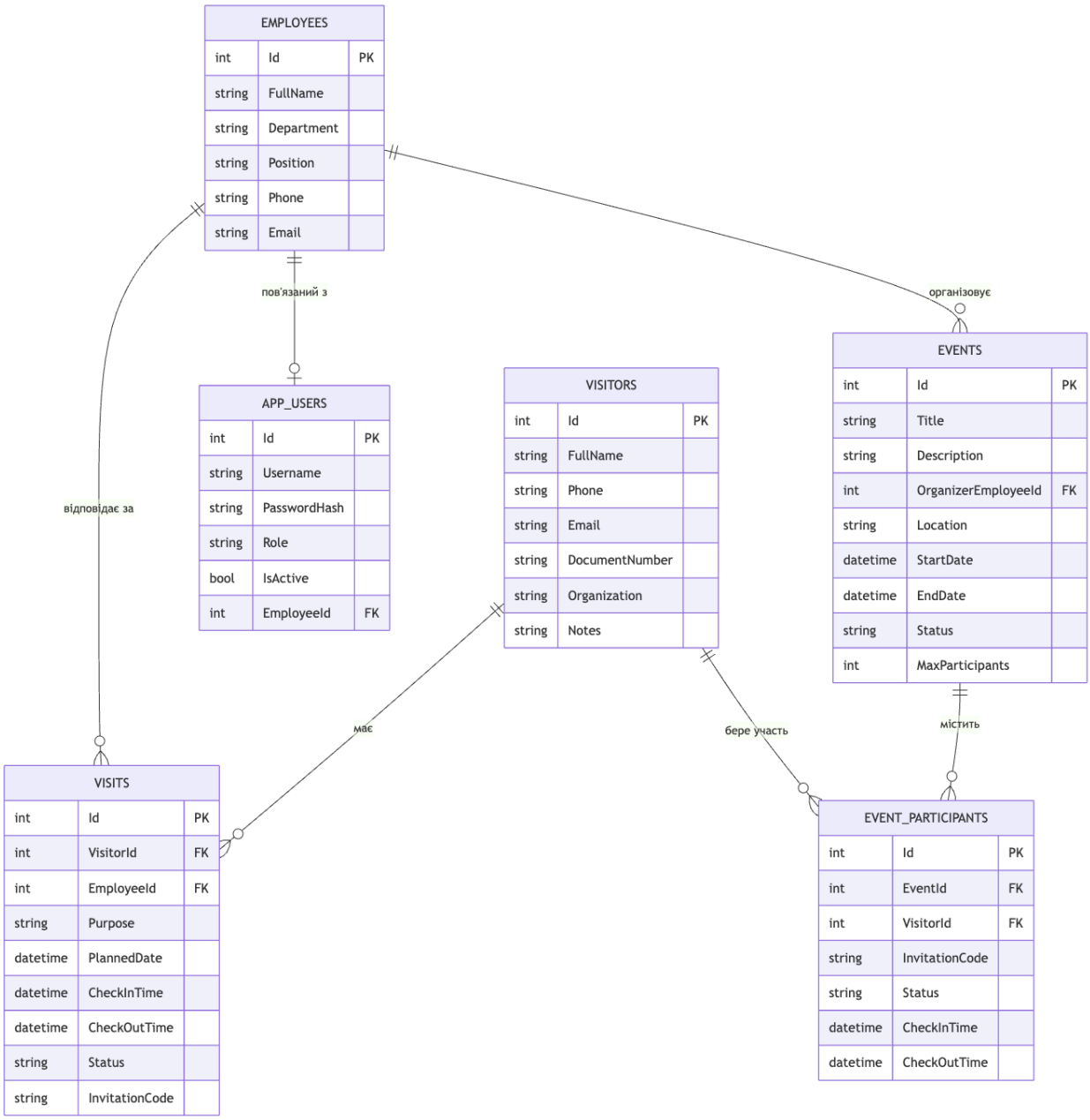



Рисунок 3.5 – Схема бази даних вебсистеми керування відвідувачами

Основу бази даних становлять таблиці, що відповідають ключовим сутностям предметної області. Таблиця Visitors призначена для зберігання даних

про відвідувачів, які можуть мати індивідуальні візити або брати участь у подіях. У ній зберігаються ідентифікатор запису, ПІБ, телефон, електронна адреса, номер документа та назва організації. Завдяки цьому інформація про відвідувача не дублюється в кожному окремому візиті або записі участі в події.

Таблиця Employees використовується для зберігання інформації про співробітників установи, відповідальних за прийом відвідувачів або організацію подій. Основними полями цієї таблиці є ідентифікатор, ПІБ, відділ, посада, телефон та електронна адреса. Ці дані застосовуються під час призначення відповідального співробітника для індивідуального візиту або організатора події.

Таблиця Visits фіксує заплановані та фактичні індивідуальні візити. Вона містить посилання на відвідувача і співробітника, заплановану дату, час входу, час виходу, статус та код запрошення. Саме ця таблиця забезпечує облік життєвого циклу візиту: від створення запису і формування запрошення до фіксації входу, виходу та завершення перебування.

Для зберігання облікових записів користувачів застосовується таблиця AppUsers. Вона містить логін, хеш пароля, ПІБ користувача, роль, ознаку активності та, за потреби, зв’язок із записом співробітника. Через цю таблицю реалізується автентифікація користувачів і розмежування доступу між адміністратором, оператором та співробітником.

Таблиця Events призначена для зберігання інформації про події. У ній фіксуються назва події, час початку і завершення, відповідальний організатор, статус та код запрошення. Така структура дає змогу планувати групові відвідування, пов’язувати подію з конкретним співробітником і контролювати її стан у процесі проведення.

Таблиця EventParticipants зберігає список учасників подій і стан їхньої присутності. Вона містить посилання на подію та відвідувача, код запрошення, статус участі, час входу і час виходу. Завдяки цій таблиці одна подія може мати багато учасників, а для кожного учасника можна окремо відстежувати факт прибуття, перебування та вибуття.

Зовнішні ключі використовуються для підтримки цілісності даних. Поля VisitorId і EmployeeId у таблиці Visits вказують на відповідні записи Visitors і Employees. Поле OrganizerEmployeeId у таблиці Events вказує на співробітника, який організовує подію. У таблиці EventParticipants поля EventId і VisitorId пов’язують учасника з конкретною подією та конкретним відвідувачем. Поле EmployeeId у AppUsers є необов’язковим, оскільки адміністратор або оператор можуть не бути прив’язаними до запису співробітника.

Для полів, що зберігають короткі текстові значення, у моделях задано обмеження довжини. Наприклад, FullName має довжину до 100 символів, статуси зберігаються у полях до 50 символів, а опис події може містити до 1000 символів. Такі обмеження допомагають контролювати розмір даних і запобігати внесенню надмірно довгих значень у поля, які використовуються в таблицях та формах інтерфейсу.

Статуси візитів і подій зберігаються як текстові значення. Для візиту використовуються стани Заплановано, В приміщенні, Завершено і Скасовано. Для учасника події використовуються стани Запрошено, Прибув і Вибув. Такий підхід спрощує відображення статусів у таблицях, фільтрах і звітах, а перевірка допустимих значень виконується на рівні контролерів.

Коди запрошень зберігаються у таблицях Visits, Events і EventParticipants. Це дозволяє створювати окремі запрошення для індивідуального візиту, для події загалом і для конкретного учасника події. Перед збереженням нового коду система перевіряє його унікальність у відповідній таблиці, що зменшує ризик помилкового відкриття чужого запису під час сканування.

Отже, база даних побудована як реляційна модель, у якій довідкові сутності Visitors і Employees використовуються повторно, а операційні таблиці Visits, Events і EventParticipants фіксують конкретні дії користувачів.

Така структура підтримує планування візитів, контроль входу та виходу, організацію подій, рольовий доступ і формування звітів без дублювання основних даних.

### 3.3 Захист інформаційно-комп’ютерної системи

Захист інформації у вебсистемі реалізовано на рівні автентифікації, авторизації та обмеження доступу до окремих дій контролерів. Користувач повинен увійти в систему через форму авторизації, після чого сервер створює cookie сесію і зберігає основні claims: ідентифікатор користувача, логін, ПІБ та роль.

Форма входу є першою точкою перевірки користувача. Вона приймає логін і пароль, передає їх у AccountController, після чого пароль хешується і порівнюється зі значенням PasswordHash у таблиці AppUsers. Тестове вікно авторизації показано на рисунку 3.6.

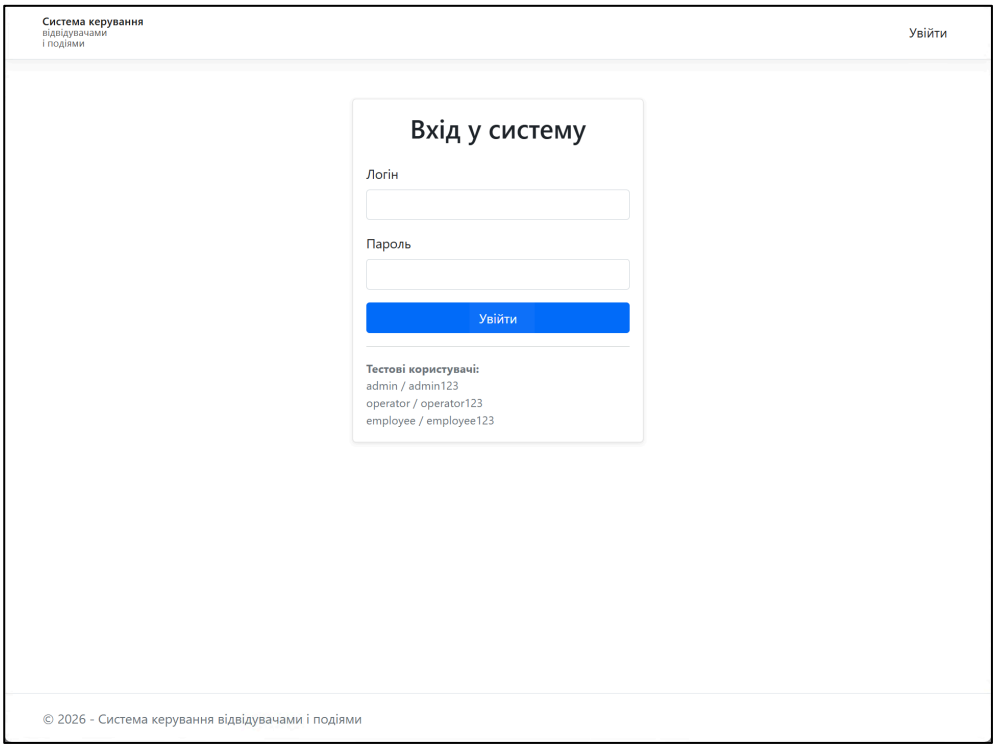



Рисунок 3.6 – Вікно авторизації користувача

У системі передбачено три основні ролі: Admin, Operator і Employee. Адміністратор має доступ до керування користувачами та довідниками, оператор виконує реєстраційні операції, а співробітник працює лише з власними візитами та подіями. На рівні коду доступ обмежено атрибутами Authorize із зазначенням

ролей, наприклад для сторінок користувачів дозволено лише роль Admin, а для перегляду звітів допускаються Admin, Operator і Employee.

Керування обліковими записами винесено в окремий модуль користувачів. На цій сторінці адміністратор бачить логіни, ПІБ, ролі, прив'язку до співробітника та стан активності облікового запису. Приклад сторінки керування користувачами наведено на рисунку 3.7.

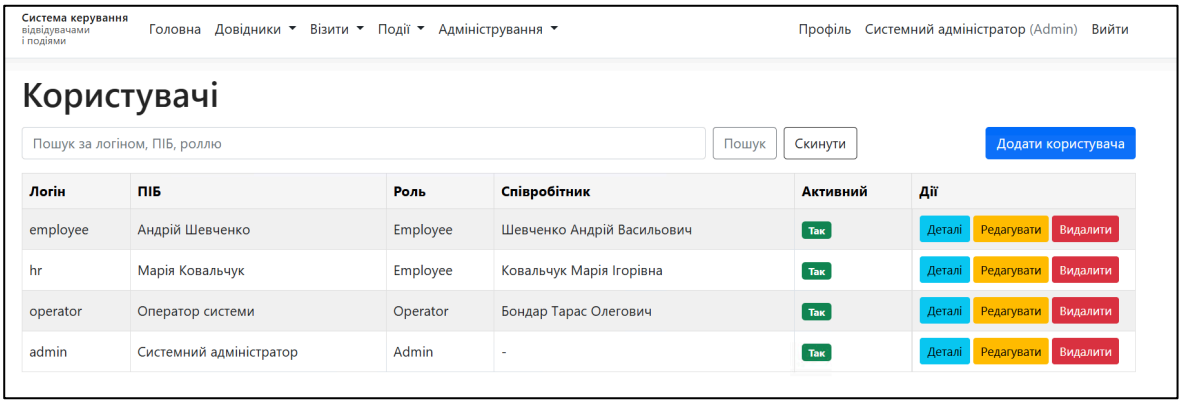



Рисунок 3.7 – Сторінка керування користувачами та ролями доступу

Додатковий контроль доступу реалізовано для ролі Employee. Навіть якщо співробітник відкриває загальний контролер візитів або подій, запит до бази даних звужується за його EmployeeId. Якщо обліковий запис не пов'язаний зі співробітником або користувач намагається переглянути чужий запис, система повертає заборону доступу.

Захист від випадкової або підробленої зміни даних забезпечується використанням ValidateAntiForgeryToken у POST методах. Це стосується створення, редагування, видалення, фіксації входу, фіксації виходу та обробки відсканованих кодів. Крім того, перед збереженням даних контролери перевіряють коректність пов'язаних сутностей, статусів, часу входу й виходу та наявність обов'язкових полів.

Отже, захист системи поєднує автентифікацію користувача, рольову авторизацію, перевірку приналежності записів конкретному співробітнику, захист POST запитів і валідацію даних перед збереженням. Такий підхід є

достатнім для локальної вебсистеми обліку відвідувачів і зменшує ризик несанкціонованого перегляду або редагування інформації.

### 3.4 Тестування та налагодження інформаційно-комп'ютерної системи

Тестування вебсистеми виконувалося за основними користувацькими сценаріями: вхід у систему, перегляд інформаційної панелі, створення та редагування довідникових записів, планування візитів, створення подій, додавання учасників, сканування запрошень, фіксація входу та виходу, фільтрація даних і експорт звітів. Окремо перевірялися сценарії з різними ролями користувачів, оскільки від них залежить набір доступних сторінок і дій.

Під час функціонального тестування модуля візитів перевірялося, що система коректно створює запис, генерує код запрошення, відображає його на сторінці запрошення, приймає код на сторінці сканування і переводить візит у стан В приміщенні. Сторінка сканування коду, яка використовувалася під час перевірки цього сценарію, наведена на рисунку 3.8.

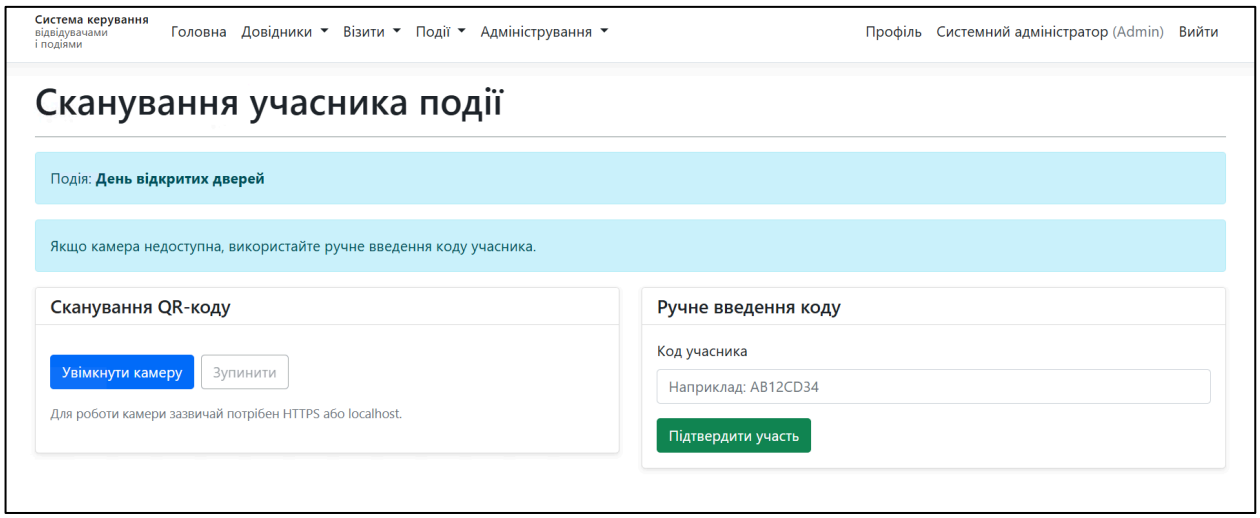



Рисунок 3.8 – Сторінка сканування або ручного введення коду запрошення

Для перевірки UI та UX оцінювалася зручність основних таблиць, зрозумілість кнопок дій, наявність фільтрів, коректність повідомлень про помилки та успішні операції. Особливу увагу приділено тому, щоб користувач

міг швидко знайти потрібний запис за статусом, датою або пошуковим рядком, а також повернутися до пов’язаних сторінок без зайвих переходів.

Звітний модуль тестувався через фільтрацію за датами, організатором, статусом і текстовим пошуком. Після застосування фільтрів перевірялися підсумкові лічильники, таблиця результатів і кнопка експорту. Приклад сторінки звіту по подіях наведено на рисунку 3.9.

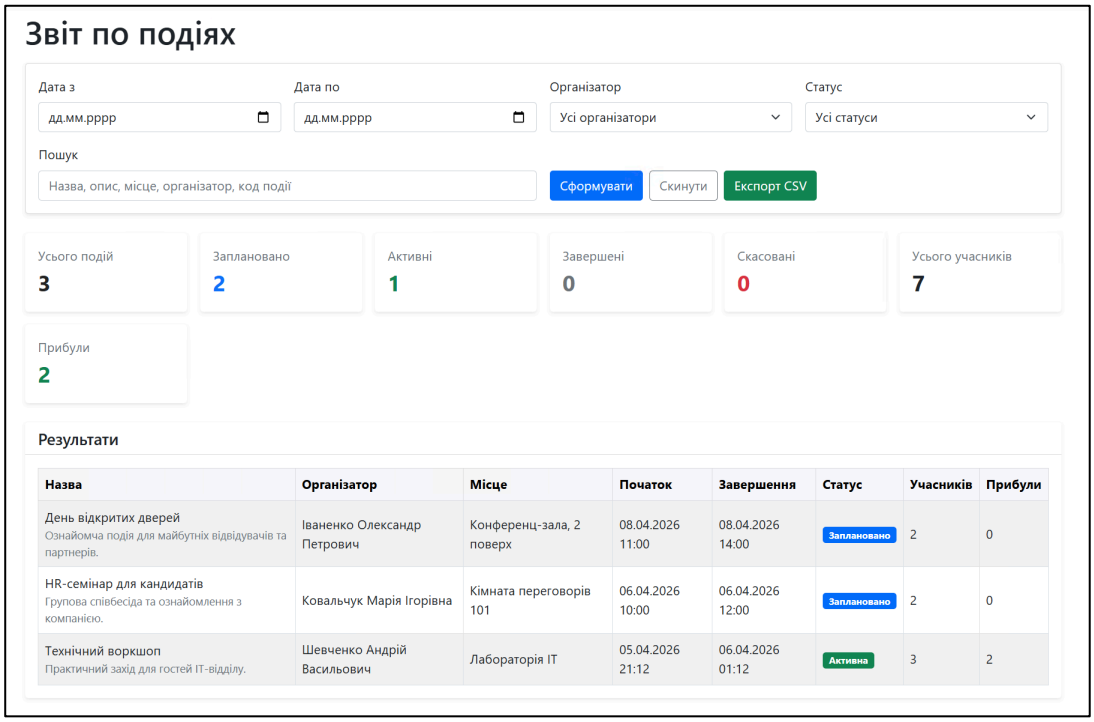
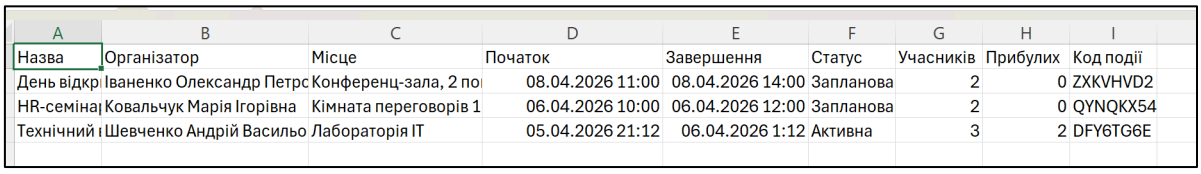



Рисунок 3.9 – Перевірка фільтрації та підсумкових показників у звіті по подіях

Експорт у CSV перевірявся шляхом формування файлів звітів і відкриття результатів у табличному редакторі. Перевірялося кодування українських символів, порядок колонок, наявність заголовків і відповідність рядків даним, що були показані на сторінці звіту. Приклад відкритого CSV файлу наведено на рисунку 3.10.



| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Назва | Організатор | Місце | Початок | Завершення | Статус | Учасників | Прибулих | Код події |
| 2 | День відкр | Іваненко Олександр Петрс | Конференц-зала, 2 по | 08.04.2026 11:00 | 08.04.2026 14:00 | Запланова | 2 | 0 | ZXKVHVD2 |
| 3 | HR-семінар | Ковальчук Марія Ігорівна | Кімната переговорів 1 | 06.04.2026 10:00 | 06.04.2026 12:00 | Запланова | 2 | 0 | QYNQKX54 |
| 4 | Технічний | Шевченко Андрій Василько | Лабораторія IT | 05.04.2026 21:12 | 06.04.2026 1:12 | Активна | 3 | 2 | DFY6TG6E |

Рисунок 3.10 – Перевірка результату експорту звіту у CSV файл

аПід час тестування ролей перевірялося, що адміністратор може керувати користувачами, оператор має доступ до реєстраційних операцій, а співробітник не бачить чужих візитів і подій. Також перевірено обробку помилкових дій: введення порожнього коду запрошення, повторну фіксацію входу, спробу виходу без попереднього входу, відкриття неіснуючого запису та збереження форми з некоректними даними.

Результати тестування показали, що основні сценарії роботи виконуються коректно: дані зберігаються у базі, статуси змінюються відповідно до дій користувача, обмеження доступу працюють згідно з ролями, фільтри й пошук звужують набір записів, а CSV експорт формує придатні для подальшого аналізу файли. Виявлені під час перевірки неточності інтерфейсу були усунені шляхом уточнення підписів, повідомлень і розміщення кнопок дій.

# ВИСНОВКИ

У кваліфікаційній роботі розроблено вебсистему керування відвідувачами з використанням ASP.NET Core, Entity Framework Core та SQLite. Створений програмний продукт поєднує облік відвідувачів, співробітників, візитів, подій, учасників подій, користувачів і звітної інформації в єдиному вебзастосунку.

Здійснено аналіз предметної області керування відвідувачами. У результаті визначено, що для ефективного супроводу візитів необхідно зберігати не лише персональні дані відвідувачів, а й мету візиту, відповідального співробітника, запланований час, фактичний час входу та виходу, статус перебування, а також інформацію про події й учасників.

Сформовано вимоги до програмного забезпечення. Передбачено функції створення та редагування записів, пошуку, фільтрації, розмежування доступу за ролями, формування звітів, експорту даних у CSV, роботи з кодами запрошень і фіксації фактичного перебування відвідувачів у приміщенні.

Спроєктовано структуру вебзастосунку та модель даних. Основними сутностями системи визначено Visitor, Employee, Visit, Event, EventParticipant і AppUser, між якими встановлено логічні зв’язки для підтримки довідникових, операційних і звітних процесів.

Обґрунтовано вибір технологій реалізації. ASP.NET Core MVC використано для побудови серверної частини й користувацьких подань, Entity Framework Core – для доступу до даних і роботи з міграціями, SQLite – для збереження інформації в локальній реляційній базі даних.

Реалізовано основні модулі програмного продукту: керування відвідувачами, співробітниками, візитами, подіями, учасниками подій, користувачами, профілем і звітами. Додано механізми автентифікації, авторизації за ролями, генерації кодів запрошень, фіксації входу та виходу, а також експорту звітних даних.

Перевірено працездатність ключових функцій системи. Результати виконання роботи підтверджують досягнення поставленої мети, а розроблена

вебсистема може бути використана як основа для автоматизації обліку відвідувачів в установах, де важливими є контроль доступу, історія візитів і швидке отримання звітної інформації.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Overview of ASP.NET Core MVC. Microsoft Learn. URL: https://learn.microsoft.com/aspnet/core/mvc/overview (дата звернення: 10.02.2026).

2. SQLite Database Provider – EF Core. Microsoft Learn. URL: https://learn.microsoft.com/ef/core/providers/sqlite/ (дата звернення: 10.02.2026).

3. Visitor management system & check in software. Envoy. URL: https://envoy.com/products/visitors (дата звернення: 12.02.2026).

4. Activity dashboard with real time monitoring. Sign In App. URL: https://signinapp.com/features/activity-dashboard (дата звернення: 14.02.2026).

5. Eptura Visitor. Eptura. URL: https://eptura.com/our-platform/eptura-visitor/ (дата звернення: 07.03.2026).

6. Razor Pages architecture and concepts in ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/aspnet/core/razor-pages/ (дата звернення: 12.03.2026).

7. Minimal APIs quick reference. Microsoft Learn. URL: https://learn.microsoft.com/aspnet/core/fundamentals/minimal-apis (дата звернення: 25.03.2026).

8. Use cookie authentication without ASP.NET Core Identity. Microsoft Learn. URL: https://learn.microsoft.com/aspnet/core/security/authentication/cookie (дата звернення: 11.04.2026).

9. Role-based authorization in ASP.NET Core MVC. Microsoft Learn. URL: https://learn.microsoft.com/aspnet/core/mvc/security/authorization/roles (дата звернення: 20.04.2026).

10. Razor syntax reference for ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/aspnet/core/mvc/views/razor (дата звернення: 22.04.2026).

11. Introduction. Bootstrap. URL: https://getbootstrap.com/docs/5.0/getting-started/introduction/ (дата звернення: 28.04.2026).

12. RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC Editor. URL: https://www.rfc-editor.org/info/rfc4180 (дата звернення: 02.05.2026).

13. Managing Migrations. EF Core. Microsoft Learn. URL: https://learn.microsoft.com/ef/core/managing-schemas/migrations/managing (дата звернення: 10.05.2026).

# ДОДАТКИ

## Додаток А

## Фрагмент контролера обробки запрошення

```
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Admin,Operator,Employee")]
public async Task<IActionResult> ProcessScannedCode(string? code)
{
    if (string.IsNullOrWhiteSpace(code))
        return View("CheckInError");

    code = code.Trim();

    // Якщо замість коду передано повне посилання, система витягує
параметр code.
    if (Uri.TryCreate(code, UriKind.Absolute, out var uri))
    {
        var query = QueryHelpers.ParseQuery(uri.Query);
        if (query.TryGetValue("code", out var values))
            code = values.FirstOrDefault() ?? code;
    }

    var visit = await _context.Visits
        .Include(v => v.Visitor)
        .Include(v => v.Employee)
        .FirstOrDefaultAsync(v => v.InvitationCode == code);

    if (visit == null)
        return View("CheckInError");

    if (visit.CheckInTime == null)
    {
        visit.CheckInTime = DateTime.Now;
        visit.Status = "В приміщенні";
        await _context.SaveChangesAsync();
    }

    return View("CheckInSuccess", visit);
}
```