МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

Кафедра програмної інженерії

# КУРСОВА РОБОТА

|  |  |  |
| --- | --- | --- |
| з | Бази даних | |
|  | (назва дисципліни) | |
| на тему: | | «Розробка бази даних інформаційної системи «Агентство нерухомості» з використанням СУБД Apache Ignite» |
|  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студента | | 2 | | курсу, групи | 6.1219-2 |
| спеціальності | | | 121 інженерія програмного забезпечення | | |
|  | | | (шифр і назва спеціальності) | | |
| Д. П. Наумова | | | | | |
| (ініціали та прізвище) | | | | | |
| Керівник | доцент кафедри програмної інженерії, кандидат фізико-математичних наук Кудін О. В. | | | | |
|  | (посада, вчене звання, науковий ступінь, прізвище та ініціали) | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Національна шкала: | |  | | |
| Кількість балів: |  | | Оцінка ECTS: |  |
|  |  | |  |  |
| Члени комісії: |  |  | |  | |
|  | (підпис) |  | | (ініціали та прізвище) | |
|  |  |  | |  | |
|  |  |  | |  | |
|  | (підпис) |  | | (ініціали та прізвище) | |
|  |  |  | |  | |
|  |  |  | |  | |
|  | (підпис) |  | | (ініціали та прізвище) | |

Запоріжжя – 2021

|  |  |  |
| --- | --- | --- |
| **МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  **ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ** | | |
| Факультет | математичний | |
| Кафедра | програмної інженерії | |
| Дисципліна | | бази даних |
| Спеціальність | | 121 інженерія програмного забезпечення |
| Освітня програма | | (шифр і назва)  програмна інженерія |

**ЗАВДАННЯ**

**на курсову роботу студентові (студентці)**

|  |
| --- |
| Наумову Дмитру Павловичу |

(прізвище, ім’я та по-батькові)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. Тема роботи | Розробка бази даних інформаційної системи «Агентство | | | | | |
|  | нерухомості» з використанням СУБД Apache Ignite | | | | | |
| 2. Строк здачі студентом закінченої роботи | | | | 20.05.2021 | | |
| 3. Вихідні дані до роботи | | | 1. Постановка задачі. | | | |
| 2. Перелік літератури. | | | | | | |
| 3. Практична реалізація коду | | | | | | |
| 4. Зміст роботи | | | | |  | |
| 1. Постановка задачі. | | | | | | |
| 2. Основні теоретичні відомості. | | | | | | |
| 3. Система управління базою даних Apache Ignite | | | | | | |
| 5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) | | | | | |  |
| 6. Дата видачі завдання | | | 01.03.2021 | | | | | |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Назва етапів курсової роботи** | **Термін виконання етапів роботи** | **Примітка** |
| 1. | Розробка плану роботи | 01.03.2021 |  |
|  |  |  |  |
| 2. | Збір вихідних даних | 10.03.2021 |  |
|  |  |  |  |
| 3. | Обробка методичних джерел | 25.03.2021 |  |
|  |  |  |  |
| 4. | Розробка першого розділу | 17.04.2021 |  |
|  |  |  |  |
| 5. | Розробка другого розділу | 29.04.2021 |  |
|  |  |  |  |
| 6. | Розробка третього розділу | 09.05.2021 |  |
|  |  |  |  |
| 7. | Оформлення курсової роботи | 17.05.2021 |  |
|  |  |  |  |
| 8. | Захист курсової роботи | 20.05.2021 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | |  |  | Д. П. Наумов |
|  |  | (підпис) |  | (ініціали та прізвище) |
|  |  | | | |
| Керівник роботи | |  |  | О. В. Кудін |
|  |  | (підпис) |  | (ініціали та прізвище) |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

**РЕФЕРАТ**

Курсова робота «Розробкa бази даних інформаційної системи «Агентство нерухомості» з використанням СУБД Apache Ignite»: 59с., 3 рис., 2 табл., 22 джерела, 3 додатки.

БАЗА ДАНИХ, APACHE IGNITE, KEY-VALUE, NOSQL.

Об’єкт дослідження – процес розробки бази даних інформаційної системи «Агентство нерухомості».

Предмет дослідження – СУБД Apache Ignite.

Мета роботи – розробити базу даних та клієнтський додаток інформаційної системи «Агентство нерухомості».

Методи дослідження – методи програмної інженерії, системний аналіз.

У роботі розроблено базу даних інформаційної системи «Агентство нерухомості» з використанням СУБД Apache Ignite та клієнтський web-додаток з використанням ASP.NET Core.

Результати роботи можуть бути використані для подальшого застосування у сфері проектування та розробки баз даних.

**ЗМІСТ**

Завдання на курсову роботу 2

Реферат 4

Вступ 6

[1 ПРОЕКТУВАННЯ БАЗИ ДАНИХ 8](#_Toc72362485)

[1.1 Аналіз існуючих СУБД 8](#_Toc72362486)

[1.2 Опис предметної області 11](#_Toc72362487)

[1.3 Побудова ER-діаграм 13](#_Toc72362488)

[Висновки до розділу 1 17](#_Toc72362489)

[2 РЕАЛІЗАЦІЯ БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ «АГЕНТСТВО НЕРУХОМОСТІ» З ВИКОРИСТАННЯМ СУБД APACHE IGNITE 18](#_Toc72362490)

[2.1 Теоретичні відомості 18](#_Toc72362491)

[2.2 Налаштування серверу та клієнта БД 19](#_Toc72362492)

[2.3 Програмна реалізація сутностей 21](#_Toc72362493)

[2.4 Забезпечення цілісності відношень 27](#_Toc72362494)

[2.5 Розробка запитів до бази даних 30](#_Toc72362495)

[Висновки до розділу 2 32](#_Toc72362496)

[3 РОЗРОБКА КЛІЄНТСЬКОГО ДОДАТКУ 33](#_Toc72362497)

[3.1 Теоретичні відомості 33](#_Toc72362498)

[3.2 Програмна реалізація додатку 35](#_Toc72362499)

[Висновки до розділу 3 37](#_Toc72362500)

[Висновки 38](#_Toc72362501)

[Перелік посилань 40](#_Toc72362502)

[Додаток А Створення бази даних 43](#_Toc72362503)

[Додаток Б Взаємодія з базою даних 52](#_Toc72362505)

[Додаток В Розробка клієнтського додатку 56](#_Toc72362507)

# ВСТУП

Сучасний розвиток інформаційних систем призводить до того, що останнім необхідні засоби для ефективної організації та збереження великих обсягів даних.

Прикладом інформаційних систем може бути агентство нерухомості.

Агентство займається оформленням контрактів з нерухомості. Ведеться облік продавців, агентів та клієнтів. До кожного об’єкту нерухомості наводиться його розгорнутий опис. За згодою покупця і продавця агентство укладає угоду. Заробітна плата агентів обчислюється залежно від кількості укладених угод.

Отже, актуальною задачею є розробка баз даних для автоматизації обліку в інформаційних системах.

Метою курсової роботи є розробка бази даних та клієнтського додатку інформаційної системи «Агентство нерухомості».

Задачі, які необхідно розв’язати для досягнення поставленої мети:

* проаналізувати існуючі системи керування базами даних;
* проаналізувати предметну область;
* сформулювати вимоги до інформаційної системи;
* спроектувати схему даних;
* спроектувати макет клієнтського програмного застосунку;
* реалізувати базу даних засобами обраної СУБД;
* реалізувати клієнтський програмний застосунок;
* протестувати інформаційну систему.

Об’єкт дослідження – процес розробки інформаційної системи «Агентство нерухомості».

Предмет дослідження – СУБД Apache Ignite.

Методи дослідження – методи програмної інженерії, системний аналіз.

Структурно робота складається з трьох розділів. У першому розділі наведено аналіз існуючих СУБД, спроектовано базу даних, виходячи з опису предметної області та поставлених вимог до інформаційної системи. У другому розділі базу даних реалізовано з використанням СУБД Apache Ignite. У третьому розділі розроблено клієнтський web-додаток.

# 1 ПРОЕКТУВАННЯ БАЗИ ДАНИХ

**1.1 Аналіз існуючих СУБД**

За використовуваною моделлю даних СУБД поділяються на такі типи:

– реляційні;

– документ-орієнтовані;

– мережеві;

– «ключ-значення»;

– комбіновані.

Реляційна модель даних є найбільш поширеною; відомими СУБД з цією моделлю є MySQL, SQLite, PostgeSQL, Microsoft SQL Server. Модель була запропонована в 1970р. Едгаром Коддом [1, c. 97]. Дані зберігаються у вигляді *n*-них відношень, а окремі записи є кортежами довжини *n*, де *n –* кількість атрибутів сутності. Елементи кортежу можуть набувати значень із домену атрибута – множини його допустимих значень [2].

Для візуального сприйняття використовуються терміни «таблиця», «рядок», «стовпець». Таблиця є множиною рядків, тому повторення останніх не допускається. Усі дані одного стовпця мають бути однакового типу та належати до множини допустимих значень. [1]

Реляційні СУБД дозволяють створювати таблиці, використовуючи попередньо визначену множину типів даних. Крім того, реляційна модель надає можливість використання первинних ключів, які забезпечують унікальність атрибута і рядка в цілому, та зовнішніх ключів, які дозволяють встановлювати логічні зв’язки між сутностями та обмежити домен атрибута. [1].

Для пошуку даних використовуються SQL-запити, до основного функціоналу яких належить вибір даних за певними критеріями, поєднання даних з декількох таблиць, упорядкування та групування рядків, обчислення підсумкових функцій (максимальне, мінімальне, середнє значення, сума тощо). Операції додавання, зміни та видалення даних також реалізуються у вигляді SQL-запитів [1].

Для пришвидшення доступу до даних використовуються індекси [3].

Сучасні версії реляційних СУБД мають широкий набір функцій [3], тому майже будь-яку операцію можна виконати на боці сервера бази даних, що забезпечує мінімальне мережеве навантаження. Це є критично важливим, якщо клієнтське ПЗ і сервер бази даних знаходяться на різних вузлах мережі.

До недоліку реляційних СУБД, на прикладі MySQL, можна віднести неможливість безпосереднього отримання запису за його ключем [3] – така функціональність може знадобитися при вирішенні певних задач. Доводиться застосовувати SQL-запити пошуку за значенням ключа.

Ще одним недоліком даного типу СУБД є фіксована схема даних для кожної таблиці. Її можна змінювати в процесі роботи з базою даних, але всі записи мають одночасно відповідати визначеному шаблону. [3]

Далі розглянемо документ-орієнтовану модель. Відомою СУБД з такою моделлю даних є MongoDB. На відміну від реляційної моделі, документ-орієнтована модель не вимагає чіткої структури записів. Допускаються записи в одній колекції з абсолютно різними наборами полів [4].

MongoDB не має зовнішніх ключів [4], тому перевірку цілісності відношень (якщо такі є) необхідно здійснювати вручну.

Так само як в реляційних базах даних, MongoDB підтримує індекси для швидкого доступу до даних [4].

Для взаємодії з базою даних MongoDB передбачає використання функцій з синтаксисом Javascript, натомість дана СУБД зовсім не підтримує мову SQL [4].

До переваг MongoDB відноситься її гнучкість при зберіганні даних, синтаксис Javascript, що робить її особливо зручною для знавців даної мови, можливість створення збережуваних функцій [4].

Недоліками MongoDB є відсутність зовнішніх ключів та SQL, що робить неможливим наслідування реляційних баз даних.

Мережева модель є розвитком ієрархічної моделі даних та дозволяє відношення «багато до багатьох» між записами. Різновидом мережевої моделі є графова модель, що передбачає зберігання даних у вигляді графу.

Однією з відомих графових СУБД є Neo4j.

Перевагою графової моделі є можливість реалізації складних відношень між сутностями [5].

В СУБД Neo4j використовується мова запитів Cypher [5], що відрізняється від інших мов своїм синтаксисом. Запити можуть охоплювати велику кількість сутностей та відношень.

Синтаксис використовуваної в Neo4j мови запитів може бути складним для початківців, це можна вважати недоліком даної СУБД.

Модель «ключ-значення» можна вважати однією з найбільш простих моделей даних. В якості прикладу можна розглянути СУБД Apache Ignite.

В даній моделі сутності розглядаються як пара примітивних даних:

– ключ, який є унікальним ідентифікатором запису;

– значення даного запису.

В залежності від СУБД, може бути присутньою додаткова функціональність, як-от SQL-надбудова в Apache Ignite, що дозволяє виконувати запити з використанням мови SQL [6].

Якщо брати до уваги Apache Ignite, вона реалізує SQL-запити та SQL-таблиці, наслідуючи реляційні СУБД, але не має зовнішніх ключів.

Отже, вище було наведено класифікацію систем управління базами даних, розглянуто особливості, переваги та недоліки кожного типу.

**1.2 Опис предметної області**

В даній роботі розглядається предметна область «Агентство нерухомості». Агентство є посередником купівлі-продажу об’єктів нерухомості. Ведеться облік нерухомості різних типів, як-от будинки, квартири, земельні ділянки. Клієнти можуть переглядати колекцію об’єктів нерухомості, публікувати свої об’єкти або замовлення з певним переліком вимог. Працівники агентства за згодою продавця і покупця укладають договори купівлі-продажу. Заробітна плата працівників залежить від вартості укладених угод.

Для підвищення рівня доступності та конкурентоспроможності, а також для автоматизації обліку на підприємстві, необхідно розробити автоматизовану інформаційну систему. Система має забезпечувати наступні функції:

– перегляд наявних об’єктів нерухомості;

– пошук об’єктів за певними критеріями;

– публікація, редагування і видалення об’єктів нерухомості;

– додавання об’єкта нерухомості в закладки;

– публікація замовлень з описом вимог до об’єкта;

– перегляд, редагування та скасування замовлень;

– запит на відвідування об’єкта для детального ознайомлення;

– укладання договору купівлі-продажу;

– модерація оголошень та облікових записів користувачів;

– створення, редагування та видалення облікових записів;

– підбір об’єктів нерухомості на основі вподобань користувачів.

Для забезпечення наведеного вище функціоналу необхідно реалізувати наступний набір сутностей (табл. 1.1).

Таблиця 1.1 – Набір сутностей предметної області.

|  |  |
| --- | --- |
| Назва сутності | Опис |
| Person | Особа – містить інформацію про ім’я, прізвище, номер телефону, адресу електронної пошти, адресу проживання, дату реєстрації. |
| Credential | Облікові дані – до них належить пароль, стан облікового запису (нормальний, деактивований або заблокований) та рівень доступу до системи (звичайний користувач, агент, модератор, адміністратор). |
| Agent | Агент – містить інформацію про загальну кількість угод, кількість угод за останній місяць, заробітну плату. |
| EstateObject | Об’єкт нерухомості – містить загальний опис, ціну, адресу, дату публікації, ідентифікатор продавця, актуальність, тип об’єкта, стан, теги, посилання на фотографії. |
| House | Будинок – описова сутність, містить характерні для будинку атрибути: площа земельної ділянки та житлова площа (одиниця виміру – квадратний метр), кількість поверхів, кількість кімнат. |
| Flat | Квартира – описова сутність, містить атрибути, характерні для квартири: житлова площа (одиниця виміру – квадратний метр), кількість кімнат, номер поверху. |
| Landplot | Земельна ділянка – описова сутність, містить інформацію про площу ділянки (одиниця виміру – ар). |
| Location | Локація, або район місцезнаходження – містить інформацію про область, населений пункт та район (якщо є поділ на райони). Вважається, що агентство працює лише на території України, тому назва країни не вказується. |
| ClientWish | Побажання клієнта – містить ідентифікатор клієнта, актуальність, дату публікації, інформацію про тип об’єкта, бажане місцезнаходження, максимальну ціну, необхідний стан об’єкта, теги. |
| Order | Заявка на огляд об’єкта – містить дату подання, ідентифікатор клієнта, ідентифікатор об’єкта та ідентифікатор агента, який погодиться супроводжувати об’єкт нерухомості. |

Кінець таблиці 1.1

|  |  |
| --- | --- |
| Назва сутності | Опис |
| Match | Співпадіння – сутність для опису відношення між побажаннями клієнта та об’єктами нерухомості, що полягає у відповідності характеристик об’єкта вимогам, наведеним у побажанні клієнта. Містить ідентифікатор об’єкта та ідентифікатор клієнтського побажання. Передбачається, що користувач зможе переглядати загальний перелік співпадінь на головній сторінці або окремий перелік для кожного опублікованого побажання. |
| Bookmark | Помітка, або закладка – містить ідентифікатор клієнта та ідентифікатор об’єкта. |
| Deal | Договір – містить дату укладання, ідентифікатори об’єкта нерухомості, продавця, покупця, агента, фінальну ціну об’єкта. |

Отже, вище було розглянуто інформаційну систему «Агентство нерухомості», сформовано перелік вимог до програмного забезпечення даної системи та визначено набір сутностей, що будуть використовуватися в даній системі.

**1.3 Побудова ER-діаграм**

ER-діаграма є графічним представленням сутностей і відношень предметної області [14]. Діаграма дозволяє відобразити сутності, їх атрибути, відношення та характеристики відношень (тип відношення, кратність). В даному випадку сутності зображуються у вигляді прямокутників, а відношення – у вигляді ліній з назвою відношення посередині лінії та зазначенням кратності відношення з обох боків від назви. Якщо до діаграми включаються переліки атрибутів сутностей, то лініями відношень з’єднуються атрибути, які є ключовими (як правило, зовнішній ключ однієї сутності-учасника відношення поєднується з первинним ключем іншої сутності).

Можливими варіантами кратності відношень є:

– 1 : 1 (один до одного);

– 1 : N (один до багатьох);

– M : N (багато до багатьох) [14].

Прикладом відношення 1 : 1 може бути відношення «обліковий запис має ідентифікаційний номер телефону». За одним обліковим записом закріплено один номер, і один номер може використовуватися лише в одному обліковому записі.

Прикладом відношення 1 : N може бути відношення «користувач продає об’єкт нерухомості». Як правило, користувач може продати декілька об’єктів нерухомості, або не продати жодного, але об’єкт може мати лише одного продавця.

Відношення «користувач додає об’єкт нерухомості в закладки» є типовим відношенням кратності M : N. Як правило, один користувач може додати в закладки декілька об’єктів нерухомості, і один об’єкт може бути в закладках у різних користувачів. Такі відношення зазвичай реалізуються з використанням додаткових сутностей. У наведеному вище прикладі відношення M : N буде поділено на 2 відношення: «Користувач має закладку» та «Закладка пов’язана з об’єктом нерухомості» – обидва з кратністю 1 : N.

Згідно з переліком сутностей (див. табл. 1.1) визначимо відношення предметної області (див. табл. 1.2).

Таблиця 1.2 –Відношення предметної області «Агентство нерухомості»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Учасник відношення 1 | Учасник відношення 2 | Назва | Кратність | Пояснення |
| 1 | 2 | 3 | 4 | 5 |
| Person | Agent | is a | 1:1 | Особа може бути агентом, одному обл. запису відповідає не більше ніж один обліковий запис агента. |
| Person | Credential | has | 1:1 | Особа має облікові дані, одній особі відповідає один набір облікових даних. |

Продовження таблиці 1.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Person | Location | lives in | N:1 | Декілька осіб можуть проживати в одній локації. |
| Person | ClientWish | has | 1:N | Особа може опублікувати декілька побажань. |
| ClientWish | Location | attached to | N:1 | Декілька побажань клієнта можуть бути закріпленими за однією локацією. |
| Person | EstateObject | sells | 1:N | Одна особа може продавати декілька об’єктів нерухомості, у кожного опублікованого об’єкта нерухомості може бути лише 1 продавець. |
| EstateObject | House | is a | 1:1 | Об’єкт нерухомості може бути будинком. Одному об’єкту нерухомості може відповідати не більше ніж один опис будинку. |
| EstateObject | Flat | is a | 1:1 | Об’єкт нерухомості може бути квартирою. Одному об’єкту нерухомості може відповідати не більше ніж один опис квартири. |
| EstateObject | Landplot | is a | 1:1 | Об’єкт нерухомості може бути земельною ділянкою. Одному об’єкту нерухомості може відповідати не більше ніж один опис земельної ділянки. |
| EstateObject | Location | placed in | N:1 | Декілька об’єктів нерухомості можуть знаходитися в одній локації. |
| EstateObject | Deal | sold by | 1:1 | Об’єкт може бути проданий за договором. Одному об’єкту відповідає не більше ніж один договір купівлі-продажу. |
| Person | Deal | buys | 1:N | Особа може купити декілька об’єктів нерухомості. Одному договору відповідає один покупець. |
| Person | Deal | sells | 1:N | Особа може продати декілька об’єктів нерухомості. Одному договору відповідає один продавець. |
| Agent | Deal | serves | 1:N | Агент може обслуговувати декілька договорів. Одному договору відповідає один агент. |

Кінець таблиці 1.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Person | Order | has | 1:N | Одна особа може зробити декілька запитів на огляд різних об’єктів нерухомості. Одному запиту відповідає одна особа, що подала запит. |
| Agent | Order | serves | 1:N | Один агент може супроводжувати декілька замовлень. Одному замовленню відповідає один агент. |
| EstateObject | Order | has | 1:N | Один об’єкт нерухомості може бути замовлений для огляду декілька разів. Одному замовленню відповідає один об’єкт нерухомості. |
| Person | Bookmark | has | 1:N | Один користувач може зробити деілька закладок. Одній закладці відповідає один користувач. |
| EstateObject | Bookmark | has | 1:N | Один об’єкт нерухомості може бути доданий у закладки декілька разів. Одна закладка може посилатися на один об’єкт. |
| ClientWish | Match | has | 1:N | Одне побажання клієнта може мати декілька співпадінь з опублікованими об’єктами нерухомості. Одне співпадіння відповідає одному побажанню клієнта. |
| EstateObject | Match | has | 1:N | Один об’єкт нерухомості може відповідати різним побажанням клієнтів. Одне співпадіння відповідає одному об’єкту нерухомості. |

Далі, з урахуванням опису сутностей (див. табл. 1.1) визначимо їх атрибути та побудуємо ER-діаграму (див. рис. 1.1). Необхідно враховувати, що далі буде використовуватися модель даних key-value, тому кожна сутність повинна мати один ключовий артибут.

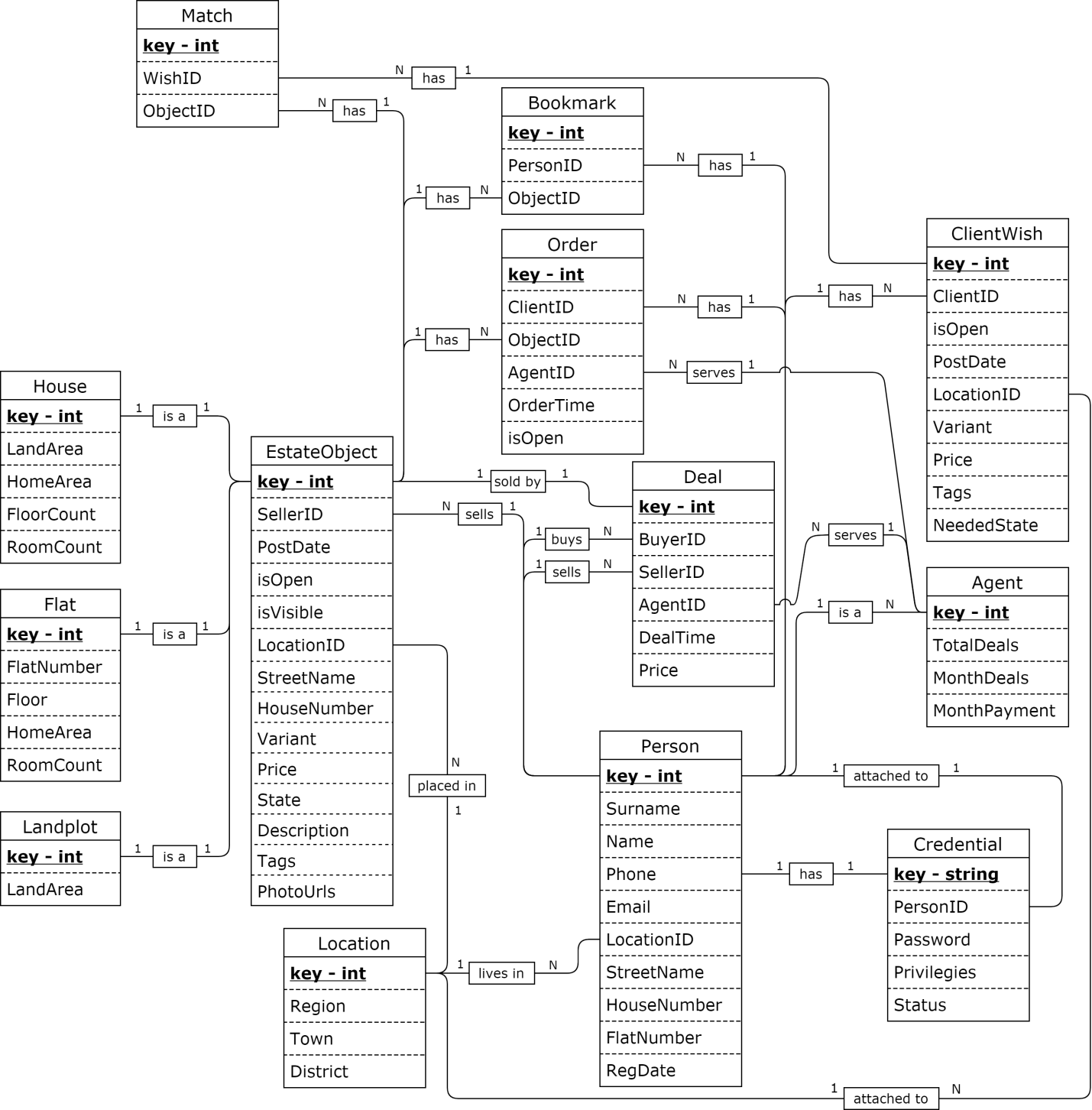


Рисунок 1.1 – ER-діаграма «Агентство нерухомості»

**Висновки до розділу 1**

В даному розділі було розглянуто класифікацію систем управління базами даних, проведено аналіз предметної області, складено перелік вимог до інформаційної системи. Побудовано ER-діаграму для відображення сутностей і відношень предметної області. ER-діаграма буде використовуватися при подальшій розробці інформаційної системи.

# 2 РЕАЛІЗАЦІЯ БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ «АГЕНТСТВО НЕРУХОМОСТІ» З ВИКОРИСТАННЯМ СУБД APACHE IGNITE

**2.1 Теоретичні відомості**

Apache Ignite (далі – Ignite) є розподіленою СУБД, що використовує модель даних key-value. Написана з використанням мови програмування Java. СУБД допускає як одиничну, так і багатовузлову конфігурацію мережі, а дані можуть зберігатися розподілено або в режимі реплікації. При розподіленому режимі кожен вузол мережі зберігає окрему порцію даних; в режимі реплікації кожен вузол містить копію всіх наявних даних. Багатовузлова конфігурація використовується при великих навантаженнях на інформаційну систему, тоді як на початкових етапах розробки і тестування доцільно використовувати одиничну конфігурацію [6].

За замовченням Ignite зберігає дані виключно в оперативній пам’яті [7]. Це означає, що інформація буде втрачена при збої процесу або вимкненні комп’ютера. Для збереження даних на постійному носії необхідно застосувати файл конфігурації з відповідними параметрами (див. розділ 2.2) при запуску вузла.

Дана СУБД використовує модель key-value, але для маніпуляцій з даними дозволяється використання мови SQL [8]. Оскільки мова SQL є досить широко використовуваною, перехід від іншої бази даних до Ignite значно полегшується.

Ignite надає можливість застосування транзакцій для забезпечення стійкості бази даних [9]. Транзакції є доцільними при виконанні декількох послідовних логічно пов’язаних запитів до бази даних (наприклад, перевірити наявність запису з ключем N, якщо немає, то додати новий запис).

Ignite надає програмні інтерфейси для різних середовищ і мов програмування, наприклад, Java, C#/.NET, C++, Python, Node.js [10]. Крім того, існує можливість вбудованого використання Ignite у програмі [11], але такий підхід уповільнює запуск програми, оскільки при кожному її виконанні запускається вузол Ignite.

Програмні інтерфейси Ignite є зручними у використанні тому, що застосовуються технології відповідної мови програмування і допускається застосування користувацьких типів даних. Тобто СУБД самостійно вирішує задачу розпізнавання і приведення типів [12]. Щодо технологій, якщо взяти до уваги платформу .NET, Ignite дозволяє використовувати LINQ – типовий для платформи інтерфейс взаємодії з даними [13].

**2.2 Налаштування серверу та клієнта БД**

В даній роботі буде використовуватися клієнт-серверний підхід до розгортання СУБД. Передбачається виконання роботи на ОС Windows. Для початку, встановимо необхідне ПЗ:

– Java runtime environment 8;

– Apache Ignite.

Оскільки за замовченням Ignite не зберігає дані на диску, створимо файл конфігурації (config.xml), в якому зазначимо необхідні опції (див. додаток А.1). Файл бажано помістити в директорію з виконавчими скриптами Ignite (bin). Для запуску серверного вузла застосовується команда:

ignite.bat config.xml

За замовченням вузол запускається в незахищеному режимі – при підключенні ім’я користувача і пароль не потрібні. Крім того, мережа вузлів при першому запуску знаходиться в неактивному стані. Для активації необхідно застосувати утиліту control.bat або відповідний метод в коді програми.

Після активації СУБД є повністю готовою до роботи. Для взаємодії з СУБД можна використовувати самостійні утиліти (наприклад, sqlline) або програмні інтерфейси.

Створимо новий проект з використанням мови C#, платформи .NET Core 3.1. Підключимо бібліотеку Apache.Ignite.Core.dll – файл бібліотеки повинен знаходитися в директорії IGNITE\_HOME/platforms/dotnet/bin (тут і далі IGNITE\_HOME – коренева директорія Ignite).

Приклад налаштування клієнтського вузла:

IIgniteClient client;

client = Ignition.StartClient (new IgniteClientConfiguration

{Endpoints = new[] {"127.0.0.1:10800"}}

);

IIgniteClient є клієнтським інтерфейсом Ignite, який є посередником у взаємодії програми користувача з сервером СУБД [15].

Метод Ignition.StartClient() створює новий екземпляр клієнта та здійснює підключення до серверного вузла.

До методу Ignition.StartClient() в якості параметра передається конфігурація клієнтського вузла. В даному випадку зазначається лише IP-адреса серверного вузла та номер використовуваного порту.

Після виконання вказаних вище операцій, якщо підключення вдалося, то до змінної client буде записано екземпляр клієнтського інтерфейсу. В іншому випадку отримаємо System.AggregateException з детальним описом помилки.

Як було зазначено вище, при першому запуску мережа вузлів Ignite знаходиться в неактивному стані, тому її необхідно активувати наступним методом:

client.GetCluster().SetActive(true);

Щоб закрити створене підключення до серверу СУБД, необхідно викликати метод Dispose() клієнтського інтерфейсу.

client.Dispose();

Отже, вище було розглянуто процес налаштування серверного вузла та клієнта СУБД Apache Ignite. Дана СУБД вимагає незначної кількості дій, що значно полегшує її використання.

**2.3 Програмна реалізація сутностей**

Як було зазначено в розділі 2.1, Ignite надає можливість зберігання користувацьких типів даних. Це спрощує розробку програмного забезпечення тому, що код можна використовувати повторно.

Для кращого розуміння процесу реалізації сутностей необхідно розглянути практичні приклади. Тут і далі будемо використовувати мову програмування C#.

Отримавши доступ до СУБД, можна почати реалізацію сутностей. Користуючись ER-діаграмою (рис. 1.1), для прикладу створимо клас сутності Location:

public class Location {

public string Region;

public string Town;

public string District;

}

Ignite зберігає дані у вигляді структури даних «кеш» (ICache, ICacheClient), яка є множиною пар ключ-значення [15]. В мові C# існує схожа структура даних під назвою Dictionary.

ICache та ICacheClient є аналогами таблиць, використовуваних в реляційній моделі даних. Різниця між ICache та ICacheClient полягає в тому, що інтерфейс ICache застосовується в серверних вузлах, запущених у користувацькій програмі, а ICacheClient – у вузлах-клієнтах [15].

Щоб створити кеш без SQL-надбудови, застосовується метод клієнтського інтерфейсу CreateCache() з параметром типу string:

ICacheClient LocationCache = client.CreateCache<int, Location>("location");

В даному випадку створюється кеш з ключами типу int та значеннями типу Location. В методі CreateCache() зазначаються типи ключів та значень, а в якості параметра передається назва кешу.

Наведений вище підхід є найпростішим варіантом створення кешу. Допускається при реалізації сутностей, які не мають відношень з іншими сутностями, і якщо не передбачається застосування SQL-запитів.

Щоб мати змогу виконувати SQL-запити, необхідно зазначити атрибут QuerySqlField у полях класу сутності або задати конфігурацію кешу, в якій зазначені поля SQL [16].

Розглянемо перший варіант:

public class Location {

[QuerySqlField] public string Region;

[QuerySqlField] public string Town;

[QuerySqlField] public string District;

}

CacheClientConfiguration locationCfg = new CacheClientConfiguration {

GroupName = "estateagency",

Name = "location",

AtomicityMode = CacheAtomicityMode.Transactional,

QueryEntities = new[] {

new QueryEntity {

TableName = "Locations",

KeyType = typeof(int),

ValueType = typeof(Location)

}

}

};

ICacheClient LocationCache = client.CreateCache<int, Location>(locationCfg);

В даному випадку в кожному полі класу було зазначено атрибут QuerySqlField, що робить поле доступним для SQL-запитів. Далі було створено конфігурацію кешу, де зазначено назву групи, власну назву кешу, транзакційний режим роботи, набір сутностей SQL. В даному випадку кеш містить лише одну сутність SQL з назвою таблиці «Locations», ключем типу int та значенням типу Location. Поля SQL автоматично отримуються з класу Location.

Робити всі поля класу видимими для SQL необов’язково. Достатньо зазначити атрибут QuerySqlField в одному полі класу. Якщо таких полів не буде і в конфігурації кешу буде зазначено поле QueryEntities, то при створенні кешу станеться помилка.

Другий варіант передбачає визначення полів SQL безпосередньо в конфігурації кешу [16]. Може бути корисним, якщо з певних причин клас сутності неможливо змінити.

CacheClientConfiguration locationCfg = new CacheClientConfiguration {

GroupName = "estateagency",

Name = "location",

AtomicityMode = CacheAtomicityMode.Transactional,

QueryEntities = new[] {

new QueryEntity {

TableName = "Locations",

KeyType = typeof(int),

ValueType = typeof(Location),

Fields = new[] {

new QueryField ("Region", typeof(string)),

new QueryField ("Town", typeof(string)),

new QueryField ("District", typeof(string))

}

}

}

};

ICacheClient LocationCache = client.CreateCache<int, Location>(locationCfg);

В даному варіанті в об’єкті QueryEntity, що відповідає сутності Location, зазначається поле Fields, що містить поля, які мають бути доступними для SQL. При конфігурації кешу автоматично встановлюється відповідність між SQL-полями та полями класу з однаковими назвами.

Окрім визначення полів, досить часто доводиться визначати індекси для пришвидшення доступу до даних. Так само як при визначенні SQL-полів, індекси можна визначити двома способами: в класі сутності або в конфігурації кешу [17].

При першому способі необхідно зазначити атрибут QuerySqlField з опцією IsIndexed на потрібному полі класу. Розглянемо приклад сутності Location:

public class Location {

**[QuerySqlField (IsIndexed = true)]** public string Region;

[QuerySqlField] public string Town;

[QuerySqlField] public string District;

}

При другому способі, в об’єкті QueryEntity, що відповідає сутності Location, необхідно визначити колекцію Indexes. Колекція містить індекси з зазначенням використовуваних полів класу.

Indexes = new QueryIndex[]

{

new QueryIndex("Region")

}

Після створення кешу, його можна отримати:

а) за допомогою методу IIgniteClient.GetCache(string).

ICacheClient LocationCache = client.GetCache<int, Location>("location");

б) за допомогою більш універсальних методів:

– IIgniteClient.GetOrCreateCache(CacheClientConfiguration);

– IIgniteClient.GetOrCreateCache(string).

ICacheClient LocationCache = client.GetOrCreateCache<int, Location>("location");

Для видалення існуючого кешу застосовується метод IIgniteClient.DestroyCache(string).

client.DestroyCache ("location");

Для отримання переліку назв існуючих кешів застосовується метод IIgniteClient.GetCacheNames().

ICollection<string> names = client.GetCacheNames();

Особливість Apache Ignite полягає в тому, що СУБД може зберігати різні типи даних в одному кеші [17]. Допускається поєднання будь-яких користувацьких та системних типів даних, але найбільш доцільним є зберігання типів даних, що пов’язані наслідуванням. Якщо сутності використовують ключ одного типу, то даний ключ є спільним для всіх сутностей. Мається на увазі, що при існуванні в одному кеші сутностей типу А і типу В, що мають ключ типу *a*, то значення *x* ключа може зустрітися лише один раз. Якщо існує запис типу A з ключем *x*,то не може існувати запис типу В з таким же значенням ключа.

Розглянемо приклад сутностей EstateObject, House з предметної області «Агентство нерухомості» (див. табл. 1.1, рис. 1.1). Сутність House походить від сутності EstateObject. Спочатку визначимо класи сутностей:

public class EstateObject {

[QuerySqlField] public int SellerID;

[QuerySqlField] public DateTime PostDate;

[QuerySqlField] public bool isOpen;

[QuerySqlField] public bool isVisible;

[QuerySqlField] public int LocationID;

[QuerySqlField] public string StreetName;

[QuerySqlField] public string HouseNumber;

[QuerySqlField] public byte Variant;

[QuerySqlField] public int Price;

[QuerySqlField] public byte State;

[QuerySqlField] public ICollection<string> Tags;

[QuerySqlField] public ICollection<string> PhotoUrls;

}

public class House: EstateObject {

[QuerySqlField] public float LandArea;

[QuerySqlField] public float HomeArea;

[QuerySqlField] public short FloorCount;

[QuerySqlField] public short RoomCount;

}

Далі необхідно задати конфігурацію кешу та створити його.

CacheClientConfiguration objectCfg = new CacheClientConfiguration {

GroupName = "estateagency",

Name = "estateobject",

AtomicityMode = CacheAtomicityMode.Transactional,

QueryEntities = new[] {

new QueryEntity {

TableName = "EstateObjects",

KeyType = typeof(int),

ValueType = typeof(EstateObject)

},

new QueryEntity {

TableName = "Houses",

KeyType = typeof(int),

ValueType = typeof(House)

}

}

};

ICacheClient ObjectCache = client.CreateCache<int, EstateObject>(objectCfg);

ICacheClient HouseCache = client.GetCache<int, House>("estateobject");

Відповідно до наведених вище прикладів, необхідно реалізувати інші сутності предметної області. Програмний код наведено в додатку А.2.

Отже, вище було розглянуто підключення до серверу БД та програмну реалізацію сутностей в моделі даних key-value із SQL-надбудовою. Реалізація сутностей включає в себе такі етапи:

– Визначення класу сутності;

– Конфігурація кешу;

– Створення кешу.

**2.4 Забезпечення цілісності відношень**

На відміну від реляційної моделі даних, модель key-value не забезпечує цілісність відношень. Важливість цієї функції полягає в захисті від появи записів, які не мають змісту при відсутності запису, на який встановлено посилання.

Наприклад, сутність об’єкта нерухомості має зберігати інформацію про продавця. Замість того, щоб зберігати всю інформацію, об’єкт нерухомості містить лише ID продавця – посилання на запис особи з відповідним ключем. Якщо такий запис не існує, то інформація про об’єкт нерухомості є неповною і, відповідно, некоректною, бо не містить інформації про особу-продавця.

Перевірка цілісності відношень має здійснюватися при додаванні, редагуванні та видаленні записів. В реляційній моделі даних перевірка здійснюється автоматично на рівні серверу БД, якщо в таблицях визначені зовнішні ключі. СУБД Apache Ignite не має функціональності зовнішніх ключів [18], тому перевірку відношень необхідно виконувати на рівні клієнтської програми.

Розглянемо приклад сутності EstateObject з предметної області (див. табл. 1.1, табл. 1.2, рис. 1.1). Дана сутність містить посилання на запис особи-продавця (SellerID) та локації (LocationID). При операції додавання необхідно перевірити наявність записів в кешах Person і Location. З урахуванням того, що база даних може працювати в багатокористувацькому режимі, застосовуються транзакції. Будемо вважати, що існує кеш LocationCache з записами типу Location, PersonCache з записами типу Person, ObjectCache з записами типу EstateObject.

public static bool PutObject (int key, EstateObject obj){

using (var tx = client.GetTransactions().TxStart()){

if (!PersonCache.ContainsKey(obj.SellerID)){

tx.Commit();

return false;

}

if (!LocationCache.ContainsKey(obj.LocationID)){

tx.Commit();

return false;

}

ObjectCache.Put(key, obj);

return true;

}

}

Наведений вище метод передбачає прямий запис об’єкта нерухомості за ключем key, незалежно від наявності заданого ключа. Щоб уникнути перезапису вже існуючих об’єктів, можна додати перевірку наявності ключа або створити кеш, у якому зберігається лічильник, і автоматично визначати ключі для нових записів.

При зміні даних запису EstateObject (ключ не змінюється), необхідно перевірити наявність тих же записів, що при операції додавання. Зміна ключа може відбутися, але при реальній експлуатації інформаційної системи така необхідність виникає дуже рідко. Для сутності EstateObject це не потрібно взагалі, але, наприклад, ключ сутності Credential може змінюватися. Тоді буде необхідно змінити поле Phone сутності Person, яке посилається на ключ сутності Credential.

При видаленні запису EstateObject порушується зміст тих сутностей, які містять поля з посиланням на ключ даного запису:

– Deal (ключ);

– Order (поле ObjectID);

– Bookmark (поле ObjectID);

– Match (поле ObjectID).

При видаленні запису можливі такі рішення:

– видалити записи з посиланням;

– замінити значення посилань на NULL;

– заборонити видалення;

– нічого не робити.

Deal є важливою сутністю, тому видалення об’єкта нерухомості забороняється, якщо існує Deal з посиланням на даний об’єкт. В іншому випадку видалення можна дозволити, при цьому будуть видалятися записи Order, Bookmark, Match, що містили посилання на вилучений об’єкт нерухомості.

Приклад методу для видалення запису EstateObject:

public static bool DeleteObject (int key){

if (DealCache.ContainsKey(key)){

return false;

}

OrderCache.Query (new SqlFieldsQuery

($"delete from Orders where ObjectID={key};"));

BookmarkCache.Query (new SqlFieldsQuery

($"delete from Bookmarks where ObjectID={key};"));

MatchCache.Query (new SqlFieldsQuery

($"delete from Matches where ObjectID={key};"));

ObjectCache.Remove (key);

return true;

}

В даному випадку застосовується як функціональна взаємодія з інтерфейсом кешу, так і SQL-запити. SQL дозволяє перекласти навантаження на серверний вузол БД, зменшити кількість мережевих взаємодій і пришвидшити виконання операції. Такий спосіб забезпечення цілісності відношень у будь-якому випадку буде працювати повільніше, ніж в реляційній моделі даних з підтримкою зовнішніх ключів. Цілісність відношень не є абсолютно необхідною, але її варто забезпечити для уникнення беззмістовних записів у базі даних. Можна досягти цілісності відношень іншими способами – наприклад, розробити клієнтську програму з обмеженою функціональністю та можливістю вибирати значення полів даних з підібраних заздалегідь.

Отже, вище було розглянуті типові рішення для забезпечення цілісності відношень в моделі даних key-value. На відміну від реляційної моделі, модель key-value не має такого функціоналу, тому за цілісність відношень відповідає клієнтське програмне забезпечення. Перевірку цілісності доцільно вбудовувати в методи для додавання, зміни та видалення записів.

**2.5 Розробка запитів до бази даних**

В розділі 2.4 було розглянуто взаємодію з базою даних з використанням програмних методів Get, Put, Remove. Такий спосіб взаємодії підходить для одиничних маніпуляцій.

Apache Ignite дозволяє використання SQL-запитів для пошуку та маніпуляції великих об’ємів даних з використанням складних критеріїв. Як і в реляційних базах даних, можна використовувати ключові слова (select, update, delete, insert, from, join, where, order by, limit і т.д). Підтримується базовий набір вбудованих функцій, як-от sum, avg, count, max, min, year, day, dateadd, а також тригонометричні функції [18].

В якості прикладу розглянемо запити для вирішення таких задач:

а) пошук будинків за заданими критеріями.

б) отримання стислого опису об’єктів, що знаходяться в переліку закладок користувача;

SQL-запит для задачі (а):

select

\_key, Price, LocationID, PostDate

from

"estateobject".Houses

where

LocationID=0 and

Price<16000 and

isOpen=true

order by

PostDate desc;

Запити виконуються в утиліті sqlline. Для початку переглянемо кількість записів у таблиці Houses, потім виконаємо запит. Результат виконання наведено в додатку Б.1.1.

Для задачі (б) SQL-запит має такий вигляд:

select

A.\_key,

Price,

Region,

Town,

District,

Description

from

(

select

\_key,

Price,

LocationID,

left (Description, 60) as Description

from

(

select

ObjectID,

PersonID

from "bookmark".Bookmarks

where PersonID=0

) as B

join

"estateobject".Houses as H

on B.ObjectID=H.\_key

) as A

join

"location".Locations as L

on A.LocationID=L.\_key;

Недоліком таких запитів є необхідність ручного редагування коду при зміні критеріїв пошуку. При взаємодії з базою даних через користувацьку програму, запит можна помістити у функцію, а критерії пошуку передавати в якості параметрів.

Приклад функції для задачі (а):

public static IList<IList<object>> GetHouses (int location, int maxprice)

{

string query = $"select \_key, Price, LocationID, PostDate, Description from \"estateobject\".Houses where LocationID={location} and Price<={maxprice} order by PostDate desc;";

return ObjectCache.Query(new SqlFieldsQuery(query)).GetAll();

}

Розглянемо приклад запиту, що маніпулює даними. Запит має вирішити наступну задачу: видалити записи будинків з ціною менше 1000 або описом, що складається менше ніж з 30 символів.

delete from "estateobject".Houses

where

Price<1000 or

Price is null or

length(Description)<30 or

Description is null;

Результат виконання запиту наведено в додатку Б.1.2.

В даному підрозділі було розглянуто використання SQL-запитів для взаємодії з базою даних. SQL-запити дозволяють шукати записи в базі даних за складними критеріями, об’єднувати дані з декількох таблиць, використовувати вбудовані функції для обробки даних, проводити маніпуляції з великими об’ємами даних – додавання нових записів, редагування і видалення за певними критеріями.

**Висновки до розділу 2**

Отже, в даному розділі було наведено стислий опис можливостей СУБД Apache Ignite та розглянуто процес реалізації бази даних, що включає такі етапи:

– налаштування серверу та клієнта БД;

– програмна реалізація сутностей;

– забезпечення цілісності відношень;

– розробка SQL-запитів та програмних функцій для обробки даних.

3 РОЗРОБКА КЛІЄНТСЬКОГО ДОДАТКУ

3.1 Теоретичні відомості

При розробці клієнтського додатку, основною задачею якого є взаємодія з базою даних, буде використовуватися мова програмування С# та фреймворк ASP.NET Core.

Перша версія мови C#, разом з платформою .NET Framework 1.0, була випущена компанією Microsoft у 2002 році. Починаючи з першої версії, C# є об’єктно-орієнтованою мовою програмування [19].

Можливості сучасних версій мови C#:

– статична типізація;

– класи з динамічними і статичними полями, методами, властивостями;

– наслідування;

– інтерфейси;

– абстрактні типи даних;

– асинхронні функції;

– перевантаження методів та операторів;

– лямбда-вирази [19].

ASP.NET Core – фреймворк для розробки веб-додатків з використанням технологій .NET Core. Включає в себе інструменти для розробки простих веб-сторінок, складних веб-додатків з використанням MVC (Model-view-controller), а також веб-інтерфейсів (Web API). Усі вищенаведені підходи можна комбінувати в одному проекті [20].

Технологія Razor Pages, що використовується в ASP.NET Core для динамічної генерації HTML-сторінок, надає можливість додавання коду C# у шаблон сторінки та комбінації програмного коду з HTML у будь-якому порядку [21]. Такий підхід дозволяє зробити генерацію сторінки більш лаконічною, у порівнянні з власноручним написанням складних функцій з великою кількістю операцій додавання текстових рядків.

Для реалізації логіки складних веб-додатків та для взаємодії з зовнішніми компонентами доцільно використовувати контролери, що входять до MVC.

В загальному вигляді, MVC працює наступним чином. Контролер приймає запит клієнта, виконує деякі маніпуляції з даними, що передбачувані логікою контролера та створює модель – дані у вигляді, прийнятному для обчислювальної техніки. За допомогою виду (View) утворюється зрозуміле для користувача представлення моделі. Отримане представлення надсилається клієнтові у відповідь на запит [22].

MVC забезпечує розділення обов’язків між частинами додатку. Контролер взаємодіє з клієнтами, виконує маніпуляції з даними. Модель використовується для зберігання даних. Вид подає дані у зручному для користувача вигляді. Але, з урахуванням того, що у виді може бути присутній програмний код, обов’язки контролера і виду можуть перетинатися.

В даній роботі створюється веб-додаток тому, що:

– веб-додатки є доступними та крос-платформними: їх можна відкрити у браузері як на персональному комп’ютері, так і на мобільному пристрої, користувачу не потрібно завантажувати додаткове програмне забезпечення;

– мова HTML дозволяє швидко та лаконічно створити графічний інтерфейс користувача;

– клієнт не матиме доступу до програмної логіки тому, що всі операції виконуються на сервері – це робить інформаційну систему більш захищеною;

– клієнт не матиме безпосереднього доступу до бази даних.

Отже, вище було наведено основні відомості про мову програмування C# та фреймворк ASP.NET Core, який буде використовуватися при розробці веб-додатка інформаційної системи «Агентство нерухомості». Також наведено стислий опис архітектури MVC та переваги веб-додатків над іншими типами програмного забезпечення.

3.2 Програмна реалізація додатку

Для реалізації веб-додатку створимо новий проект із використанням ASP.NET Core MVC. Підключимо раніше розроблений проект з визначенням сутностей та методами для взаємодії з базою даних. Призначенням попереднього проекту є реалізація сутностей, що є спільною для бази даних та клієнтських додатків, та реалізація часто виконуваних операцій. Призначенням нового проекту є реалізація варіантів використання системи, доступних для простих користувачів.

В якості прикладу розглянемо один з варіантів використання системи – створення облікового запису. Зовнішнім учасником процесу є користувач.

Орієнтовний перебіг подій:

Користувач надсилає запит на отримання сторінки для створення облікового запису. Контролер приймає запит і надсилає користувачу сторінку з формою для введення облікових даних. Користувач заповнює форму і надсилає її на сервер. Контролер приймає запит та перевіряє значення полів на коректність. Якщо є недопустимі значення, контролер знову надсилає форму для даних з повідомленням про помилку. Якщо локальна перевірка пройшла успішно, перевіряється наявність облікових записів у базі даних з таким номером телефону. Якщо такі записи вже існують, контролер знову надсилає форму для даних з повідомленням про помилку. Якщо таких записів немає, то надані користувачем облікові дані вважаються правильними. До бази даних додається нові записи Person та Credential (див. табл. 1.1, рис. 1.1). Користувач переадресовується на сторінку для входу, що містить повідомлення про успішне створення облікового запису.

Побудуємо діаграму діяльності для даного варіанту використання (див. рис. 3.1).

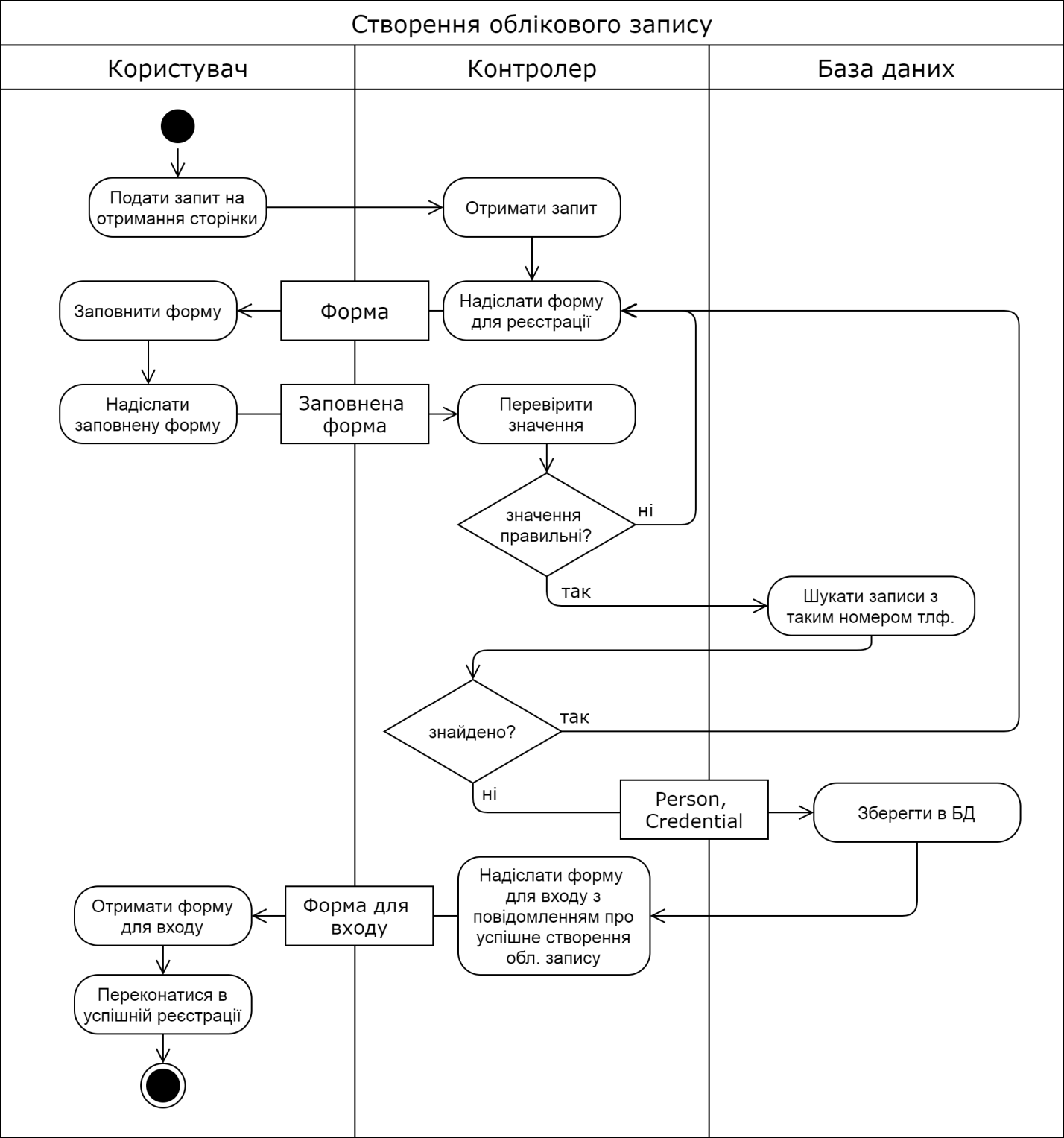


Рисунок 3.1 – Діаграма діяльності «Створення облікового запису»

Діаграма діяльності надає графічне представлення того, як дії розподіляються між учасниками процесу. Далі, використовуючи опис перебігу подій, діаграму діяльності (див. рис. 3.1) та опис сутностей (див. рис. 2.1), спробуємо реалізувати даний варіант використання.

Для цього розробимо в класі контролера:

– метод SignupGet, що буде надсилати користувачу форму для реєстрації;

– метод Signup, для обробки заповненої користувачем форми, перевірки даних та додавання запису в БД;

– метод LoginGet, що буде надсилати користувачу форму для входу в обліковий запис.

Окрім методів в класі контролера, необхідно розробити сторінку з формою для реєстрації та сторінку з формою для входу.

Фрагмент програмного коду контролера наведено в додатку В.1. Приклад коду сторінки для реєстрації наведено в додатку В.2. Приклад зовнішнього вигляду сторінки (див. рис. В.1).

Отже, вище було розглянуто варіант використання «Створення облікового запису» інформаційної системи «Агентство нерухомості». Для даного варіанту використання складено опис перебігу подій та побудовано діаграму діяльності, що відображає послідовність дій та розподіл обов’язків між учасниками процесу.

Висновки до розділу 3

В даному розділі було наведено стислий опис мови програмування C#, фреймворку ASP.NET Core, архітектури MVC. Наведено переваги використання веб-додатків. Наведено приклад розробки веб-додатку для варіанту використання «Створення облікового запису» інформаційної системи «Агентство нерухомості».

**Висновки**

В даній роботі було наведено процес розробки бази даних інформаційної системи «Агентство нерухомості». При виконанні роботи використано СУБД Apache Ignite.

В першому розділі наведено класифікацію існуючих СУБД, переваги і недоліки використовуваних моделей даних. Далі було проведено аналіз предметної області, складено перелік вимог до інформаційної системи. З урахуванням вимог складено перелік сутностей, що будуть використовуватися в даній інформаційній системі. Побудовано ER-діаграму для графічного представлення сутностей і відношень інформаційної системи.

В другому розділі наведено теоретичні відомості про СУБД Apache Ignite, приклади реалізації сутностей, забезпечення цілісності відношень, приклади взаємодії з базою даних через методи програмного інтерфейсу та SQL-запити. Сутності інформаційної системи реалізовано з використанням моделі «ключ-значення» та SQL-надбудови. Взаємодія з базою даних відбувається з одночасним використанням двох підходів:

**–** методи програмного інтерфейсу для одиничного доступу до записів;

**–** SQL-запити для обробки великої кількості записів.

В третьому розділі розроблено клієнтський веб-додаток для взаємодії з базою даних інформаційної системи. Додаток розроблено з використанням мови програмування C#, фреймворку ASP.NET Core та архітектури MVC. Наведено приклад реалізації варіанту використання «Створення облікового запису».

Використана в роботі СУБД Apache Ignite є базою даних типу «ключ-значення».

Переваги застосування даної СУБД:

**–** розширені можливості взаємодії з даними;

**–** підтримка користувацьких типів;

– зберігання різних типів даних в одному кеші;

**–** можливість одночасної роботи з об’єктами та полями SQL;

**–** зручний програмний інтерфейс;

**–** можливість розподіленої роботи;

– зберігання даних в оперативній пам’яті з можливістю запису на постійний носій;

Недоліки даної СУБД:

– відсутність зовнішніх ключів;

– модель даних створює перешкоди при реалізації деяких сутностей;

– неможливість повної відмови від SQL при вирішенні задач;

– за замовченням дані зберігаються лише в оперативній пам’яті.

СУБД Apache Ignite доцільно використовувати у великих інформаційних системах, що передбачають зберігання значної кількості даних та доступ до даних з великою швидкістю.

Використання Apache Ignite в малих системах з намаганням наслідувати реляційну модель не є дуже доцільним, але є альтернативним варіантом реалізації бази даних, що порушує монополію реляційних СУБД.

Apache Ignite перебуває в стані активної розробки. Можна припускати, що в майбутніх версіях набір функцій Apache Ignite буде розширено, що надасть можливість повноцінної розробки баз даних із застосуванням реляційної моделі.

**Перелік посилань**

1. Дейт К. Дж. Введение в системы баз данных, 7-е издание. : Пер. с англ. – М. : Издательский дом «Вильямс», 2001. – 1072 с. : ил.
2. E. F. Codd. A Relational Model of Data for Large Shared Data Banks. URL: <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf> (дата звернення: 03.04.2021)
3. MySQL :: MySQL 8.0 Reference manual. URL: [https://dev.mysql.com/doc/refman/8.0/en/](https://dev.mysql.com/doc/refman/8.0/en/%20) (дата звернення: 04.04.2021)
4. The MongoDB 4.4 Manual – MongoDB Manual. URL: [https://docs.mongodb.com/manual/](https://docs.mongodb.com/manual/%20) (дата звернення: 08.04.2021)
5. The Neo4j Getting Started Guide v4.2. URL: <https://neo4j.com/docs/getting-started/current/> (дата звернення 08.04.2021)
6. Apache Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/index> (дата звернення: 04.05.2021)
7. Ignite Persistence | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/persistence/native-persistence> (дата звернення: 05.05.2021)
8. Working with SQL | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/SQL/sql-introduction> (дата звернення: 05.05.2021)
9. Performing transactions | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/key-value-api/transactions> (дата звернення: 05.05.2021)
10. Thin Clients Overview | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/thin-clients/getting-started-with-thin-clients> (дата звернення: 05.05.2021)
11. Ignite quick start guide for .NET | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/quick-start/dotnet> (дата звернення: 06.05.2021)
12. Ignite .NET and Platform interoperability. URL: <https://ignite.apache.org/docs/latest/net-specific/net-platform-interoperability> (дата звернення: 06.05.2021)
13. Apache Ignite .NET LINQ Provider | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/net-specific/net-linq> (дата звернення: 06.05.2021)
14. Peter Pin-Shan Chen. The Entity-Relationship model – Toward a unified view of data. URL: <https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.523.6679> (дата звернення: 01.05.2021)
15. .NET Thin Client | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/thin-clients/dotnet-thin-client> (дата звернення: 07.05.2021)
16. SQL API | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/SQL/sql-api> (дата звернення 08.05.2021)
17. Defining indexes | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/SQL/indexes> (дата звернення 09.05.2021)
18. SQL Conformance | Ignite Documentation. URL: <https://ignite.apache.org/docs/latest/sql-reference/sql-conformance> (дата звернення 10.05.2021)
19. The history of C# - C# Guide | Microsoft Docs. URL: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history> (дата звернення: 13.05.2021)
20. Introduction to ASP.NET Core | Microsoft Docs. URL: <https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0> (дата звернення 13.05.2021)
21. Views in ASP.NET Core MVC | Microsoft Docs. URL: <https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview> (дата звернення 13.05.2021)
22. Overview of ASP.NET Core MVC | Microsoft Docs. URL: <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0> (дата звернення 13.05.2021)

# Додаток А

# Створення бази даних

**А.1 Конфігурація кешу**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="ignite.cfg" class="org.apache.ignite.configuration.IgniteConfiguration">

<!-- Enabling Apache Ignite Persistent Store. -->

<property name="dataStorageConfiguration">

<bean class="org.apache.ignite.configuration.DataStorageConfiguration">

<!--property name="storagePath" value="/storage" /-->

<property name="walSegmentSize" value="#{64 \* 1024 \* 1024}"/>

<property name="walSegments" value="2" />

<property name="walHistorySize" value="2" />

<property name="defaultDataRegionConfiguration">

<bean class="org.apache.ignite.configuration.DataRegionConfiguration">

<property name="persistenceEnabled" value="true"/>

</bean>

</property>

</bean>

</property>

<!-- Explicitly configure TCP discovery SPI to provide a list of initial nodes. -->

<property name="discoverySpi">

<bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">

<property name="ipFinder">

<bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.multicast.TcpDiscoveryMulticastIpFinder">

<property name="addresses">

<list>

<value>127.0.0.1:47500..47502</value>

</list>

</property>

</bean>

</property>

</bean>

</property>

</bean>

</beans>

**А.2 Реалізація сутностей**

using System;

using System.Collections;

using System.Collections.Generic;

using Apache.Ignite.Core.Cache.Configuration;

using System.Text.RegularExpressions;

namespace EstateAgency.Entities

{

public class Credential: IValidable

{

public int PersonID;

public string Password;

public byte Privilegies;

public byte Status;

public ValidationResult Validate

{

get

{

if (Password == null || Password.Length < 8 || Password.Length > 32)

return new ValidationResult (

"Password should be from 8 to 32 characters long.",

"Password"

);

if (Status!=(byte)'n' && Status!=(byte)'d' && Status!=(byte)'b')

return new ValidationResult (

"Status can be only 'n' - normal, 'd' - deactivated or 'b' - banned.",

"Status"

);

if (Privilegies!=(byte)'c' && Privilegies!=(byte)'a' && Privilegies!=(byte)'m' && Privilegies!=(byte)'r')

return new ValidationResult (

"Privilegies can be only 'c' - client, 'a' - agent, 'm' - moderator or 'r' - root.",

"Credential"

);

return ValidationResult.Success;

}

}

}

public class Person: IValidable

{

[QuerySqlField] public string Surname;

[QuerySqlField] public string Name;

[QuerySqlField] public string Phone;

[QuerySqlField] public string Email;

[QuerySqlField] public int LocationID;

[QuerySqlField] public string StreetName;

[QuerySqlField] public string HouseNumber;

[QuerySqlField] public short FlatNumber;

[QuerySqlField] public DateTime RegDate;

public ValidationResult Validate

{

get

{

if(Surname == null || Surname.Length < 2 || Surname.Length > 64)

return new ValidationResult (

"Surname must have from 2 to 64 letters.",

"Surname"

);

if(Name == null || Name.Length < 2 || Name.Length > 64)

return new ValidationResult (

"Name must have from 2 to 64 letters.",

"Name"

);

if(Phone == null || Phone.Length < 10 || Phone.Length > 16)

return new ValidationResult (

"Phone number must be from 10 to 16 digits long.",

"Phone"

);

foreach (char i in Surname)

{

if (!(char.IsLetter(i) || i=='-' || i==' '))

return new ValidationResult (

"Surname can only consist of alphabetic letters or dash.",

"Surname"

);

}

foreach (char i in Name)

{

if (!(char.IsLetter(i) || i=='-' || i==' '))

return new ValidationResult (

"Name can only consist of alphabetic letters or dash.",

"Name"

);

}

for (int i=0; i<Phone.Length; i++)

{

if (Phone[i]=='+' && i==0) continue;

if (!(char.IsDigit(Phone[i])))

return new ValidationResult (

"Phone must consist only of digits or '+' at the begin.",

"Phone"

);

}

if (Email == null || Email.Length<8 || Email.Length>64)

return new ValidationResult (

"Email must be from 8 to 64 characters long.",

"Email"

);

string mch = Regex.Match(Email, @"\b[a-zA-Z0-9.\_-]+@[a-zA-Z0-9][a-zA-Z0-9.-]{0,61}[a-zA-Z0-9]\.[a-zA-Z.]{2,6}\b").Value;

if (mch != Email)

return new ValidationResult (

$"Regex: {mch} - Email must consist only of latin alphabetic letters, digits, '@' symbol, dot or dash. If your email is real and doesn't correspond to this criteria, we are sorry.",

"Email"

);

if (StreetName == null)

return new ValidationResult (

"Street name must not be empty.",

"StreetName"

);

foreach (char i in StreetName)

{

if (!(char.IsLetter(i) || char.IsDigit(i) || i=='.' || i=='-' || i==' '))

return new ValidationResult (

"Street name must contain only alphabetic letters, digits, dot or dash.",

"Email"

);

}

if (HouseNumber.Length==0 || HouseNumber.Length>5 || HouseNumber[0]=='0' ||

((HouseNumber.Length - Regex.Match(HouseNumber, "\\d+").Value.Length) > 1) ||

(Regex.Match(HouseNumber, "\\d+").Index!=0))

return new ValidationResult (

"Invalid house number.",

"HouseNumber"

);

if (FlatNumber > 1000)

return new ValidationResult (

"Invalid flat number.",

"FlatNumber"

);

if(RegDate > DateTime.UtcNow)

return new ValidationResult (

"Signup date can not be after current date and time.",

"RegDate"

);

return ValidationResult.Success;

}

}

}

public class Agent: IValidable

{

[QuerySqlField] public short TotalDeals;

[QuerySqlField] public short MonthDeals;

[QuerySqlField] public int MonthPayment;

public ValidationResult Validate

{

get

{

if (TotalDeals < 0 || MonthDeals < 0 || MonthPayment < 0 || TotalDeals < MonthDeals) return ValidationResult.Fail;

return ValidationResult.Success;

}

}

}

public class Location: IValidable

{

[QuerySqlField] public string Region;

[QuerySqlField] public string Town;

[QuerySqlField] public string District;

public ValidationResult Validate

{

get

{

if (Region == null || Region.Length < 5 || Region.Length > 30)

return new ValidationResult(

"Region name length should be from 5 to 30 characters.",

"Region"

);

if (Town == null || Town.Length < 2 || Town.Length > 30)

return new ValidationResult(

"Town name length should be from 2 to 30 characters.",

"Region"

);

if (District==null || District=="")

return ValidationResult.Success;

if (District.Length > 30 || District.Length < 2)

return new ValidationResult(

"District name length should be from 2 to 30 characters.",

"Region"

);

return ValidationResult.Success;

}

}

}

public class EstateObject: IValidable

{

[QuerySqlField] public int SellerID;

[QuerySqlField] public DateTime PostDate;

[QuerySqlField] public bool isOpen;

[QuerySqlField] public bool isVisible;

[QuerySqlField] public int LocationID;

[QuerySqlField] public string StreetName;

[QuerySqlField] public string HouseNumber;

[QuerySqlField] public byte Variant;

[QuerySqlField] public int Price;

[QuerySqlField] public byte State;

[QuerySqlField] public string Description;

[QuerySqlField] public ICollection<string> Tags;

[QuerySqlField] public ICollection<string> PhotoUrls;

public virtual ValidationResult Validate

{

get

{

if (StreetName == null)

return new ValidationResult (

"Street name must not be empty.",

"StreetName"

);

foreach (char i in StreetName)

{

if (!(char.IsLetter(i) || char.IsDigit(i) || i=='.' || i=='-' || i==' '))

return new ValidationResult (

"Street name must contain only alphabetic letters, digits, dot or dash.",

"Email"

);

}

if (HouseNumber.Length==0 || HouseNumber.Length>5 || HouseNumber[0]=='0' ||

((HouseNumber.Length - Regex.Match(HouseNumber, "\\d+").Value.Length) > 1) ||

(Regex.Match(HouseNumber, "\\d+").Index!=0))

return new ValidationResult (

"Invalid house number.",

"HouseNumber"

);

if (Variant!=(byte)'o')

return new ValidationResult (

"Variant of basic Estate object can only be 'o' - other.",

"Variant"

);

if (State < 0 || State > 5)

return new ValidationResult (

"State can be only from 0 to 5.",

"State"

);

if (Description == null || Description.Length > 2000 || Description.Length < 10)

return new ValidationResult (

"Description of object must be from 10 to 2000 characters long.",

"Description"

);

if (Tags!=null && Tags.Count > 20)

return new ValidationResult (

"Only 20 tags are allowed.",

"Tags"

);

if (PhotoUrls == null || PhotoUrls.Count < 1 || PhotoUrls.Count > 10)

return new ValidationResult (

"You should provide from 1 to 10 photos.",

"PhotoUrls"

);

return ValidationResult.Success;

}

}

}

public class House: EstateObject, IValidable

{

[QuerySqlField] public float LandArea;

[QuerySqlField] public float HomeArea;

[QuerySqlField] public short FloorCount;

[QuerySqlField] public short RoomCount;

public override ValidationResult Validate

{

get

{

var v = base.Validate;

if (!(v.isValid) && v.FieldName!="Variant") return v;

if (Variant!='h')

return new ValidationResult ("Variant must be 'h' for a house.", "Variant");

if (LandArea < 1 || LandArea > 1000F)

return new ValidationResult ("Land area must be in [1..1000] range.", "LandArea");

if (HomeArea < 1 || HomeArea > 40000F)

return new ValidationResult ("Home area must be in [1..40000] range.", "HomeArea");

if (FloorCount < 1 || FloorCount > 10)

return new ValidationResult ("Floor count must be in [1..10] range.", "FloorCount");

if (RoomCount < 1 || RoomCount > 100)

return new ValidationResult ("Room count must be in [1..100] range.", "RoomCount");

return ValidationResult.Success;

}

}

}

/// <summary>

/// Inherits basic estate object entity, contains specific information about flat.

/// </summary>

public class Flat: EstateObject, IValidable

{

[QuerySqlField] public float HomeArea;

[QuerySqlField] public short Floor;

[QuerySqlField] public short RoomCount;

[QuerySqlField] public short FlatNumber;

public override ValidationResult Validate

{

get

{

var v = base.Validate;

if (!(v.isValid) && v.FieldName!="Variant") return v;

if (Variant!='f')

return new ValidationResult ("Variant must be 'f' for a flat.", "Variant");

if (HomeArea < 1 || HomeArea > 400F)

return new ValidationResult ("Home area must be in [1..40000] range.", "HomeArea");

if (Floor < 0 || Floor > 100)

return new ValidationResult ("Floor number must be in [1..100] range.", "FloorCount");

if (RoomCount < 1 || RoomCount > 10)

return new ValidationResult ("Room count must be in [1..10] range.", "RoomCount");

if (FlatNumber < 1 || FlatNumber > 1000)

return new ValidationResult ("Flat number must be in [1.1000] range.", "FlatNumber");

return ValidationResult.Success;

}

}

}

public class Landplot: EstateObject, IValidable

{

[QuerySqlField] public float LandArea;

public override ValidationResult Validate

{

get

{

var v = base.Validate;

if (!(v.isValid) && v.FieldName!="Variant") return v;

if (Variant!='l')

return new ValidationResult ("Variant must be 'l' for landplot.", "Variant");

if (LandArea < 1 || LandArea > 1000F)

return new ValidationResult ("Land area must be in [1..1000] range.", "LandArea");

return ValidationResult.Success;

}

}

}

public class ClientWish: IValidable

{

[QuerySqlField] public int ClientID;

[QuerySqlField] public DateTime PostDate;

[QuerySqlField] public bool isOpen;

[QuerySqlField] public byte Variant;

[QuerySqlField] public int LocationID;

[QuerySqlField] public int Price;

[QuerySqlField] public ICollection<string> Tags;

[QuerySqlField] public byte NeededState;

public ValidationResult Validate

{

get

{

if (PostDate > DateTime.UtcNow)

return new ValidationResult ("Post date must not be after now.", "PostDate");

if (!(Variant==(byte)'o' || Variant==(byte)'h' || Variant==(byte)'f' || Variant==(byte)'l'))

return new ValidationResult ("Variant can be only 'o' - other, 'h' - house, 'f' - flat or 'l' - landplot.", "Variant");

if (Price < 1)

return new ValidationResult ("Price must be greater than 0.", "Price");

if (Tags!=null && Tags.Count > 20)

return new ValidationResult ("No more than 20 tags are allowed.", "Tags");

if (NeededState < 0 || NeededState > 5)

return new ValidationResult ("Needed state must be from 0 to 5.", "Needed state");

return ValidationResult.Success;

}

}

}

public class Bookmark

{

[QuerySqlField] public int PersonID;

[QuerySqlField] public int ObjectID;

}

public class Match

{

[QuerySqlField] public int WishID;

[QuerySqlField] public int ObjectID;

}

public class Order

{

[QuerySqlField] public int ClientID;

[QuerySqlField] public int ObjectID;

[QuerySqlField] public int AgentID;

[QuerySqlField] public DateTime OrderTime;

[QuerySqlField] public bool isOpen;

}

public class Deal

{

[QuerySqlField] public int BuyerID;

[QuerySqlField] public int SellerID;

[QuerySqlField] public int AgentID;

[QuerySqlField] public DateTime DealTime;

[QuerySqlField] public int Price;

}

}

# Додаток Б

# Взаємодія з базою даних

**Б.1 Приклади SQL-запитів**

**Б.1.1 Запит для пошуку даних**

0: jdbc:ignite:thin://127.0.0.1> select

. . . . . . . . . . . semicolon> \_key,

. . . . . . . . . . . semicolon> Price,

. . . . . . . . . . . semicolon> LocationID,

. . . . . . . . . . . semicolon> PostDate

. . . . . . . . . . . semicolon> from

. . . . . . . . . . . semicolon> "estateobject".Houses

. . . . . . . . . . . semicolon> ;

+------+--------+------------+-----------------------------+

| \_KEY | PRICE | LOCATIONID | POSTDATE |

+------+--------+------------+-----------------------------+

| 1 | 12333 | 0 | 2021-05-16 16:27:33.0478344 |

| 3 | 10200 | 0 | 2021-05-16 16:36:15.9647435 |

| 4 | 12333 | 2 | 2021-05-16 16:42:14.7152629 |

| 5 | 12333 | 2 | 2021-05-16 16:48:08.9595245 |

| 6 | 123333 | 0 | 2021-05-16 17:24:13.6913401 |

| 7 | 12333 | 0 | 2021-05-16 17:33:57.0137042 |

| 8 | 123333 | 0 | 2021-05-16 17:39:23.7553928 |

| 9 | 80000 | 4 | 2021-05-17 10:56:50.6588875 |

| 10 | 26100 | 7 | 2021-05-19 09:45:39.8163004 |

| 47 | 1001 | null | null |

+------+--------+------------+-----------------------------+

10 rows selected (0.031 seconds)

0: jdbc:ignite:thin://127.0.0.1> select

. . . . . . . . . . . semicolon> \_key,

. . . . . . . . . . . semicolon> Price,

. . . . . . . . . . . semicolon> LocationID,

. . . . . . . . . . . semicolon> PostDate

. . . . . . . . . . . semicolon> from

. . . . . . . . . . . semicolon> "estateobject".Houses

. . . . . . . . . . . semicolon> where

. . . . . . . . . . . semicolon> LocationID=0 and

. . . . . . . . . . . semicolon> Price<16000 and

. . . . . . . . . . . semicolon> isOpen=true

. . . . . . . . . . . semicolon> order by

. . . . . . . . . . . semicolon> PostDate desc,

. . . . . . . . . . . semicolon> Price;

+------+-------+------------+-----------------------------+

| \_KEY | PRICE | LOCATIONID | POSTDATE |

+------+-------+------------+-----------------------------+

| 7 | 12333 | 0 | 2021-05-16 17:33:57.0137042 |

| 3 | 10200 | 0 | 2021-05-16 16:36:15.9647435 |

| 1 | 12333 | 0 | 2021-05-16 16:27:33.0478344 |

+------+-------+------------+-----------------------------+

3 rows selected (0.046 seconds)

0: jdbc:ignite:thin://127.0.0.1>

**Б.1.2 Запит для видалення даних**

0: jdbc:ignite:thin://127.0.0.1> insert into "estateobject".Houses (

. . . . . . . . . . . . . . . )> \_key, Price, Description)

. . . . . . . . . . . semicolon> values

. . . . . . . . . . . semicolon> (45, 1200, 'too short'),

. . . . . . . . . . . semicolon> (46, 999, 'Description example description example description'),

. . . . . . . . . . . semicolon> (47, 1001, 'Description example, description example, example description');

3 rows affected (0.231 seconds)

0: jdbc:ignite:thin://127.0.0.1> !outputformat vertical

0: jdbc:ignite:thin://127.0.0.1> insert into "estateobject".Houses (

. . . . . . . . . . . . . . . )> \_key, Description)

. . . . . . . . . . . semicolon> values

. . . . . . . . . . . semicolon> (48, 'too short'),

. . . . . . . . . . . semicolon> (50, 'Description example description example description'),

. . . . . . . . . . . semicolon> (49, 'Description example, description example, example description');

3 rows affected (0.047 seconds)

0: jdbc:ignite:thin://127.0.0.1> select \_key, Price, Description, length(Description) as Length from "estateobject".Houses;

\_KEY 1

PRICE 12333

DESCRIPTION Here should be description, but I am lazy x2 to write it. Sorry x2.

LENGTH 67

\_KEY 3

PRICE 10200

DESCRIPTION aaaaaaa aaaaaaaaaa aaaaaaaaaa aaa aaaaaaaaaaaaa aaaaa aaaaaaaaaaaaa aaaaa aaaaaaaaaaaaa aaaaaa aaaaaaaaaaaaa aaaaa aaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaa

LENGTH 149

\_KEY 4

PRICE 12333

DESCRIPTION aa aaaaaa aaaaaaaa aaaaaa aaaaaaaaaaaaaa aa aaa

LENGTH 47

\_KEY 5

PRICE 12333

DESCRIPTION aaa aaaaa aaaaaaaaaaa aa aaaaaaaaaaaaa aaa aa ffafaf eafafeefaaef

LENGTH 65

\_KEY 6

PRICE 123333

DESCRIPTION aaaa aaaaaaaaaa aaaaaaaaaaaaaaaaaaaaa aaaaaaaa aaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaa aa aaaaaaaaaaaaaa

LENGTH 109

\_KEY 7

PRICE 12333

DESCRIPTION sdfsdddfdfsa dsadfsadfdsdasf dasffffffffffffff sdafdaaff asfsfsdfssdffsfsd sfdsaffasfsa

LENGTH 87

\_KEY 8

PRICE 123333

DESCRIPTION aaaaaaaa aaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaa aa aaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaa

LENGTH 94

\_KEY 9

PRICE 80000

DESCRIPTION Будинок площею 92 кв.м в Олександрівському районі

LENGTH 49

\_KEY 10

PRICE 26100

DESCRIPTION 2-поверховий будинок в с.Біленьке. 2 поверхи. 8 кімнат. Житлова площа 189 кв.м. Стан відмінний. Ремонт зроблено в травні 2020 року. Є вода, газ, електропостачання.

LENGTH 164

\_KEY 45

PRICE 1200

DESCRIPTION too short

LENGTH 9

\_KEY 46

PRICE 999

DESCRIPTION Description example description example description

LENGTH 51

\_KEY 47

PRICE 1001

DESCRIPTION Description example, description example, example description

LENGTH 61

\_KEY 48

PRICE null

DESCRIPTION too short

LENGTH 9

\_KEY 49

PRICE null

DESCRIPTION Description example, description example, example description

LENGTH 61

\_KEY 50

PRICE null

DESCRIPTION Description example description example description

LENGTH 51

15 rows selected (0.056 seconds)

0: jdbc:ignite:thin://127.0.0.1> delete from "estateobject".Houses

. . . . . . . . . . . semicolon> where

. . . . . . . . . . . semicolon> Price<1000 or

. . . . . . . . . . . semicolon> Price is null or

. . . . . . . . . . . semicolon> length(Description)<30 or

. . . . . . . . . . . semicolon> Description is null;

5 rows affected (0.055 seconds)

0: jdbc:ignite:thin://127.0.0.1> select \_key, Price, Description, length(Description) as Length from "estateobject".Houses;

\_KEY 1

PRICE 12333

DESCRIPTION Here should be description, but I am lazy x2 to write it. Sorry x2.

LENGTH 67

\_KEY 3

PRICE 10200

DESCRIPTION aaaaaaa aaaaaaaaaa aaaaaaaaaa aaa aaaaaaaaaaaaa aaaaa aaaaaaaaaaaaa aaaaa aaaaaaaaaaaaa aaaaaa aaaaaaaaaaaaa aaaaa aaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaa

LENGTH 149

\_KEY 4

PRICE 12333

DESCRIPTION aa aaaaaa aaaaaaaa aaaaaa aaaaaaaaaaaaaa aa aaa

LENGTH 47

\_KEY 5

PRICE 12333

DESCRIPTION aaa aaaaa aaaaaaaaaaa aa aaaaaaaaaaaaa aaa aa ffafaf eafafeefaaef

LENGTH 65

\_KEY 6

PRICE 123333

DESCRIPTION aaaa aaaaaaaaaa aaaaaaaaaaaaaaaaaaaaa aaaaaaaa aaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaa aa aaaaaaaaaaaaaa

LENGTH 109

\_KEY 7

PRICE 12333

DESCRIPTION sdfsdddfdfsa dsadfsadfdsdasf dasffffffffffffff sdafdaaff asfsfsdfssdffsfsd sfdsaffasfsa

LENGTH 87

\_KEY 8

PRICE 123333

DESCRIPTION aaaaaaaa aaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaa aa aaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaa

LENGTH 94

\_KEY 9

PRICE 80000

DESCRIPTION Будинок площею 92 кв.м в Олександрівському районі

LENGTH 49

\_KEY 10

PRICE 26100

DESCRIPTION 2-поверховий будинок в с.Біленьке. 2 поверхи. 8 кімнат. Житлова площа 189 кв.м. Стан відмінний. Ремонт зроблено в травні 2020 року. Є вода, газ, електропостачання.

LENGTH 164

\_KEY 47

PRICE 1001

DESCRIPTION Description example, description example, example description

LENGTH 61

10 rows selected (0.041 seconds)

0: jdbc:ignite:thin://127.0.0.1>

# Додаток В

# Розробка клієнтського додатку

**В.1 Приклад програмного коду контролера**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using EstateAgencyWeb.Models;

using EstateAgency.Entities;

using EstateAgency.Database;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Session;

using System.IO;

namespace EstateAgencyWeb.Controllers

{

public class HomeController : Controller

{

[HttpPost("/Login")]

public IActionResult LoginPost(string phone, string password)

{

if (phone==null || password==null)

{

ViewData["ErrorMessage"] = "Phone or password fields were empty.";

return View ("Login");

}

switch (DbAdvanced.CheckCredentials(phone, password))

{

case 'n':

HttpContext.Session.SetString("Phone", phone);

Credential cr = DbClient.CredentialCache.Get(phone);

HttpContext.Session.SetInt32("PersonID", cr.PersonID);

HttpContext.Session.SetString("Privilegies", ((char)cr.Privilegies).ToString());

ViewData["LoggedIn"]=true;

return RedirectToAction ("Explore");

case 'p':

ViewData["ErrorMessage"] = "Password is wrong. Try again.";

break;

case 'a':

ViewData["ErrorMessage"] = $"Account with phone '{phone}' was not found.";

break;

case 'b':

ViewData["ErrorMessage"] = $"Account with phone '{phone}' was permanently banned.";

break;

case 'd':

ViewData["ErrorMessage"] = $"Account with phone '{phone}' was deactivated.";

break;

default:

ViewData["ErrorMessage"] = "Unknown error occured. Try again.";

break;

}

return View("Login");

}

[HttpGet("/Signup")]

public ActionResult SignupGet()

{

return View("Signup");

}

[HttpPost("/Signup")]

public ActionResult Signup()

{

Person p = new Person{

Phone = Request.Form["phone"],

Email = Request.Form["email"],

Name = Request.Form["name"],

Surname = Request.Form["surname"],

LocationID = int.Parse(Request.Form["location"]),

StreetName = Request.Form["street"],

HouseNumber = Request.Form["housenumber"],

RegDate = DateTime.UtcNow

};

short flatnumber;

if (short.TryParse(Request.Form["flatnumber"], out flatnumber))

p.FlatNumber = flatnumber;

var v = p.Validate;

if(!v.isValid)

{

ViewData["ErrorMessage"] = v.Message;

return View ("Signup");

}

try

{

DbAdvanced.CreateAccount(p, Request.Form["password"]);

ViewData["ErrorMessage"] = "You have successfully created an account. Now log in.";

return View("Login");

}

catch (ReferentialException ee)

{

ViewData["ErrorMessage"] = ee.ReadableMessage;

return View ("Signup");

}

}

**B.2 Приклад View**

Файл Signup.cshtml:

@using EstateAgency.Database;

@{

Layout = "\_Basic";

ViewData["Title"] = "Sign up";

ViewData["script"] = "site-signup.js";

}

<h2>Create new account</h2>

<form asp-action="Signup" asp-controller="Home" method="post" class="customcard container20em">

<label>Номер телефону</label>

<input type="tel" name="phone" />

<label>Пароль</label>

<input type="text" name="password" maxlength="32" />

<label>Email</label>

<input type="email" name="email" maxlength="64" />

<label>Ім'я</label>

<input type="text" name="name" maxlength="32" />

<label>Прізвище</label>

<input type="text" name="surname" maxlength="32" />

<label>Область</label>

<select id="regionselect">

@foreach (string i in DbAdvanced.Regions)

{

<option value="@i">@i </option>

}

</select>

<label>Населений пункт</label>

<select id="townselect">

</select>

<label>Район (необов'язково)</label>

<select name="location" id="districtselect">

</select>

<label>Вулиця</label>

<input type="text" name="street" maxlength="40" />

<label>Номер будинку</label>

<input type="text" name="housenumber" maxlength="5" />

<label>Номер квартири (необов'язково)</label>

<input type="text" name="flatnumber" maxlength="4" />

<br />

<p style='color: darkred; font-size:80%; font-weight: bold; line-height: 200%;'>@ViewData["ErrorMessage"] &ensp;</p>

<button type="submit">Продовжити</button>

</form>

**В.3 Приклад вигляду сторінки**

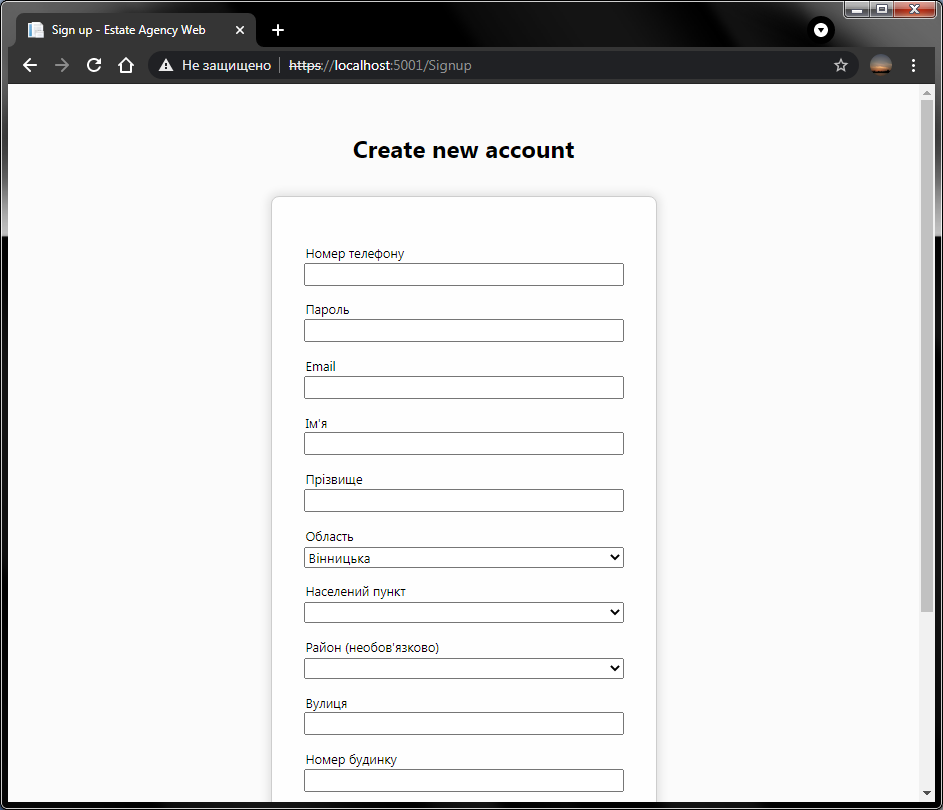


Рисунок В.1 – Вигляд сторінки «Sign up» у браузері