Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет математики та інформатики

Кафедра математичного моделювання

Курсова робота

на тему: Розробка веб-додатку для комунікації волонтерів та людей, що потребують допомоги

Студента (ки) 3 курсу 301 групи
Спеціальності 122 Комп'ютерні науки
Козельської Марини Анатоліївни
(прізвище та ініціали)
Керівник доцент, канд. фіз.-мат. наук
Піддубна Л.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____

Кількість балів: ____Оцінка: ECTS ____

_____ ____Піддубна Л.А.____
(підпис керівника) (прізвище та ініціали)

Чернівці – 2022

Анотація

Розроблено частину веб-додатку для комунікації волонтерів та людей, що потребують допомоги. Додане відображення оголошень на допомогу та списку волонтерів, що готові прийти на поміч.

При розробці було використано наступні технології: мова програмування Java, Spring Framework, мова розмітки html, css, фреймворк Angular.

Зміст

Вступ

У сучасному світі існує багато проблем, які потребують вмілого та ефективного рішення. Однією з таких проблем є необхідність допомогти людям у складних життєвих ситуаціях. Волонтерство є одним з найкращих способів надання допомоги тим, хто її потребує. Проте, часто буває важко знайти відповідну інформацію про можливості допомоги та знайти людей, які могли би надати її.

У даній курсовій роботі спробуємо розв'язати цю проблему шляхом створення веб-додатку, який дозволить волонтерам та людям, які потребують допомоги, знаходити одне одного та співпрацювати.

Даний додаток має на меті зробити процес знаходження допомоги та взаємодії між волонтерами та людьми, які потребують допомоги, максимально простим та зручним. Зокрема, в додатку буде можливість додавати та переглядати оголошення про потребу в допомозі, а також знаходити людей, які можуть надати допомогу в конкретній сфері.

Розділ 1 Постановка задачі

Метою курсової роботи є створення веб-додатку для комунікації між волонтерами та людьми, які потребують допомоги. Додаток повинен допомогти людям знайти необхідну допомогу та зв'язатися з волонтерами, які готові допомогти у різних сферах.

Для досягнення поставленої мети спочатку необхідно вирішити наступні задачі:

1. Розробити веб-додаток, який має зручний та інтуїтивно зрозумілий інтерфейс для користувачів. Додаток повинен мати зручну навігацію.

2. Розробити систему розміщення оголошень про потребу в допомозі. Оголошення повинні містити короткий опис проблеми та потреби, а також контактну інформацію особи, яка потребує допомоги.

3. Провести тестування додатку та внести необхідні корективи для покращення його роботи.

Після вирішення цих задач очікується, що веб-додаток буде допомагати людям знайти допомогу в різних сферах життя, включаючи медичну допомогу, допомогу з побутовими питаннями та багато іншого. Крім того, додаток може стати зручним інструментом для волонтерів, які готові допомагати людям у своєму місті або районі. Вони зможуть знайти потреби інших в своїй області діяльності.

Ці задачі є першими кроками у створенні повноцінного веб-додатку, де люди зможуть самостійно додавати оголошення, а волонтери реагувати на них, та комунікувати з людьми, що потребують допомоги.

Розділ 2 Опис використаних технологій

2.1 Azure

2.1.1 Опис Azure. Їх послуги

Azure – це платформа хмарних обчислень, розроблена компанією Microsoft. Ця платформа надає широкий спектр різноманітних послуг, які можуть використовуватися для розробки, впровадження та керування різними типами додатків та сервісів.

Однією з головних переваг Azure є те, що вона дозволяє користувачам зосередитися на розробці додатків, не витрачаючи багато часу на керування інфраструктурою. Платформа Azure надає можливість використовувати послуги, які включають в себе такі можливості, як зберігання даних, обробку даних, розгортання додатків, машинне навчання, інтернет речей та багато іншого.

Основні послуги Azure можна розділити на наступні категорії:

Обчислення:

- Віртуальні машини: можливість запуску та управління віртуальними машинами на базі різних операційних систем, включаючи Windows та Linux.
- Контейнери: можливість розгортання, керування та масштабування контейнерів Docker.
- Автоматичні масштабованість та резервне копіювання: можливість автоматичного масштабування додатків та забезпечення резервного копіювання даних.

Зберігання:

- Azure Blob Storage: можливість зберігати великі обсяги неструктурованих даних, таких як зображення, відео та інші файли.
- Azure Table Storage: можливість зберігання структурованих даних в режимі ключ-значення.
- Azure SQL Database: можливість зберігання та управління реляційними базами даних.

Мережі:

- Azure Virtual Network: можливість створювати та налаштовувати віртуальні мережі.

- Azure Load Balancer: можливість балансувати трафік між різними вузлами додатку.

- Azure Traffic Manager: можливість керувати трафіком між різними географічно розташованими вузлами додатку.

Аналітика та інтелектуальний аналіз даних:

- Azure Stream Analytics: можливість відслідковувати та аналізувати в реальному часі великі потоки даних.

- Azure Data Lake Analytics: можливість обробки великих обсягів даних у режимі пакетного аналізу.

- Azure Machine Learning: можливість розробки та навчання моделей машинного навчання.

Розгортання та управління додатками:

- Azure App Service: можливість швидко розгорнути та керувати веб-додатками на базі .NET, Java, PHP та Node.js.

- Azure Kubernetes Service: можливість розгортання та керування контейнерами Kubernetes.

- Azure Functions: можливість запускати код у відповідь на різноманітні події.

Безпека та ідентифікація:

- Azure Active Directory: можливість керувати доступом користувачів до різних ресурсів на базі ролей та прав доступу.

- Azure Key Vault: можливість зберігання та керування конфіденційними ключами та секретами.

- Azure Security Center: можливість моніторингу безпеки різних ресурсів в Azure та рекомендацій щодо покращення безпеки.

Крім того, Azure надає можливість інтегрувати різноманітні послуги та створювати власні рішення за допомогою Azure API та різноманітних інструментів розробки, таких як Visual Studio, Visual Studio Code та Azure CLI.

В цілому, Azure є потужною та розширеною платформою хмарних обчислень, яка надає широкий спектр різноманітних послуг для розробки та управління різними типами додатків та сервісів [1].

2.1.2 Опис CosmosDB

CosmosDB – це глобальна, масштабована, розподілена база даних з декількома API, яку розробляє та підтримує компанія Microsoft. Вона дозволяє розробникам працювати з даними різної природи, такими як документи, графи, ключі та значення, а також забезпечує низьку латентність та високу пропускну здатність завдяки розподіленому збереженню даних.

CosmosDB пропонує мульти-API, які включають DocumentDB, Cassandra, MongoDB, Graph API та Tabular Data Stream (TDS) API. Це означає, що розробники можуть використовувати CosmosDB для роботи з даними будь-якої форми, не залежно від їх структури.

Однією з особливостей CosmosDB є можливість глобальної реплікації даних. База даних CosmosDB може зберігати копії даних в різних географічних регіонах, що дозволяє забезпечувати доступ до даних з будь-якої точки світу. Більше того, CosmosDB забезпечує гарантовану консистентність даних незалежно від розташування користувачів та взаємодії з даними.

Для того, щоб забезпечити швидкий доступ до даних, CosmosDB використовує декілька оптимізацій та механізмів розподіленого зберігання даних. Для кожного API CosmosDB використовує різні механізми зберігання та обробки даних. Наприклад, для підтримки графової моделі даних CosmosDB використовує графову базу даних, яка дозволяє зберігати взаємозв'язані дані та здійснювати ефективний пошук великих графів.

CosmosDB дозволяє легко розширювати ресурси для забезпечення високої пропускної здатності та обробки великих обсягів даних. Крім того, CosmosDB підтримує гнучкість схеми даних, що дозволяє зберігати дані різної структури та формату.

Окрім того, CosmosDB має вбудовані механізми для забезпечення безпеки даних. CosmosDB забезпечує автоматичне шифрування даних, використовуючи ключі, що зберігаються в Azure Key Vault. Крім того, CosmosDB забезпечує механізми автентифікації та авторизації, що дозволяють контролювати доступ до даних.

Ще однією з ключових переваг CosmosDB є її інтеграція з іншими сервісами та інструментами Azure. CosmosDB може інтегруватись з Azure Functions, Azure Stream Analytics, Azure Search та іншими сервісами для реалізації різних сценаріїв використання даних.

Відповідно до моделі розподіленого зберігання даних, CosmosDB розподіляє дані на фізичних серверах. Це дозволяє забезпечувати високу доступність та надійність даних. Крім того, CosmosDB може автоматично мігрувати дані між фізичними серверами, що дозволяє забезпечувати балансування навантаження та високу доступність даних.

Узагальнюючи, CosmosDB – це потужна розподілена база даних з підтримкою різних API та механізмів зберігання даних. Вона забезпечує високу доступність, надійність та безпеку даних, а також має гнучку схему даних та можливості для розширення. Крім того, CosmosDB інтегрується з іншими сервісами Azure та надає розробникам багато можливостей для розробки різноманітних додатків та сервісів з високою пропускною здатністю та масштабованістю [2].

Щоб створити базу даних в CosmosDB, необхідно мати аккаунт Azure та доступ до порталу Azure. Ось кроки, які потрібно виконати для створення бази даних в CosmosDB:

1. Спочатку потрібно увійти в портал Azure та перейти до розділу "Сторінки Azure". Знайти цей розділ можна у лівій частині екрана.

2. Натиснути кнопку "Створити ресурс" в правому верхньому куті екрана та вибрати категорію "Бази даних".

3. Вибрати "Azure Cosmos DB" в списку доступних баз даних та натиснути кнопку "Далі".

4. Заповнити форму створення облікового запису CosmosDB. Необхідно вказати ім'я облікового запису, ресурсну групу, місце зберігання та API для роботи з базою даних.

5. Після заповнення форми, потрібно натиснути кнопку "Попередній перегляд та створення". Натиснути кнопку "Створити".

6. Після створення облікового запису CosmosDB перейти до розділу "Ключі" та скопіювати рядок з'єднання з відповідного розділу.

7. Відкрити, наприклад, Visual Studio або будь-який інший редактор коду та підключитись до бази даних CosmosDB, використовуючи рядок з'єднання, який був скопійованим на попередньому кроці.

8. Після підключення до бази даних CosmosDB ви можете створювати контейнери та документи, що дозволить вам зберігати та отримувати дані з бази даних.

Отже, створення бази даних в CosmosDB є простим та займе всього кілька хвилин. Після створення бази даних ви можете легко підключитись до неї та розпочати роботу з даними [3].

2.2 Мова програмування Java

Java – об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems».

Основна логіка додатку написана саме за допомогою Java [4].

2.2.1 Spring Framework

Spring Framework є одним з найпопулярніших Java фреймворків, що використовується для створення веб-додатків та інших програмних проектів. Фреймворк забезпечує багато корисних інструментів та функцій, що допомагають розробникам зосередитися на розробці функціоналу додатку, замість витрачання часу на написання коду, який повторюється.

Spring Framework забезпечує підхід "інверсії керування" (Inversion of Control - IoC) та "впровадження залежностей" (Dependency Injection - DI), що

дозволяє розробникам визначати, які об'єкти використовуватимуться в проекті, і при цьому не потрібно створювати їх самостійно. Це спрощує процес розробки, оскільки розробник може сконцентруватися на розробці функціональності, не звертаючи увагу на деталі роботи з об'єктами.

Spring Framework також має вбудовану підтримку багатьох інших технологій, таких як JDBC, ORM (Hibernate, JPA), AOP (Aspect-Oriented Programming), інтеграція зі Spring Security, Spring Data, Spring Boot і т.д. Це дозволяє розробникам ефективно використовувати ці технології в своїх проектах без необхідності займатися важким конфігуруванням та налаштуванням [5].

2.3 Postman

Postman – це програма для тестування і розробки REST API. Вона дозволяє розробникам легко створювати, тестувати, документувати та дебажити API. Postman надає користувачам можливість створювати HTTP-запити до веб-сервера, перевіряти відповіді на запити та аналізувати дані, що повертаються сервером.

Postman дозволяє розробникам:

- Створювати HTTP-запити до веб-серверів за допомогою різних методів HTTP, таких як GET, POST, PUT, DELETE, PATCH та інших.
- Отримувати відповіді від серверів у різних форматах, таких як JSON, XML, HTML та інших.
- Дебажити запити та відповіді для виявлення помилок та проблем.
- Автоматизувати тестування API та створювати колекції запитів для тестування та документування API.

Цей додаток використовувався при тестуванні програми для курсової Роботи [6].

2.4 HTML, CSS, Angular

HTML (Hypertext Markup Language) – це стандартна мова розмітки веб-сторінок. Вона використовується для створення структури веб-сторінки та вказує, які елементи мають бути на сторінці та як вони повинні

відображатися. HTML використовує теги для опису вмісту сторінки, такі як заголовки, параграфи, списки, зображення та багато іншого [7].

CSS (Cascading Style Sheets) – це мова стилів, яка використовується для оформлення веб-сторінок. Вона дозволяє визначати зовнішній вигляд елементів, таких як кольори, шрифти, розміри, відступи, рамки та інше. CSS дозволяє відокремити стиль веб-сторінки від її вмісту, що робить її більш підтримуваною та змінюваною [8].

Angular – це фреймворк для розробки веб-додатків, який використовує мову програмування TypeScript. Angular дозволяє створювати веб-додатки з високою рівнем складності та взаємодії з користувачем. Він забезпечує компонентну модель, яка дозволяє розбити додаток на невеликі, повторно використовувані блоки. Angular також надає сервіси, які дозволяють взаємодіяти з сервером та зберігати дані на клієнтському боці [9].

Розділ 3 Опис розробленого проекту

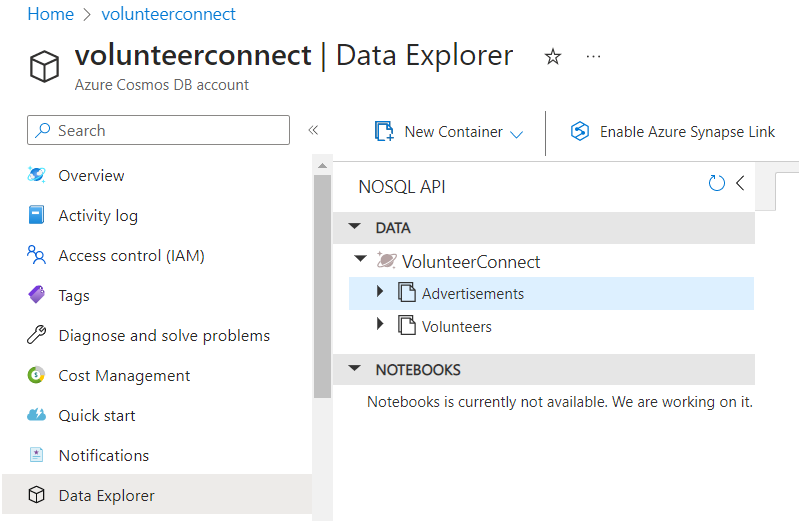
3.1 Опис проєкту VolunteerConnect

Проєкт VolunteerConnect має на меті забезпечити зручну платформу для комунікації між волонтерами та людьми, які потребують допомоги. Його основною задачею є створення зручного веб-додатку, який дозволить людям, що потребують допомоги, легко створювати оголошення про потрібні речі, а волонтерам швидко знаходити запити, які відповідають їхнім навичкам.

Для досягнення цієї мети потрібен набір інформації про волонтерів, їхні навички, а також про потреби людей (дані про них ті оформляють у запит). Окрім цього, необхідно розробити зручний інтерфейс, який дозволить швидко знаходити оголошення, волонтерів та контактну інформацію для комунікації.

3.2 Створення бази даних

Для цього проєкту є зручною для використання база даних CosmosDB. За допомогою Azure порталу я створила новий ресурс з використанням API Azure Cosmos DB для NoSQL. Після цього було додано нову базу даних VolunteerConnect.



3.3 Створення проєкту

Для створення backend частини проєкту використано мову програмування Java та Spring Framework. Таким чином, за допомогою

фреймворку, з’явилась можливість встановити зв'язок з базою даних та розробити простий REST додаток.

Для цього у файлі pom.xml (для збірки проєкту використовувався інструмент Maven) потрібно додати низку залежностей. Для зручності можна використати Spring Initializr, таким чином

```
     <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>com.azure.spring</groupId>
    <artifactId>spring-cloud-azure-starter-data-
cosmos</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Конфігурація фреймворку відбувається за допомогою окремо створених класів у пакеті config. Ще необхідно додати налаштування для підключення до бази даних CosmosDB у файлі application.properties.

Для створення ж frontend частини проєкту використовувався фреймворк Angular.

3.4 Сутності

Перш за все, варто створити модель даних, яку необхідно зберігати в базі даних CosmosDB. Наприклад, можна створити клас з анотацією

@Container, яка вказує, що цей клас повинен бути збережений в контейнері з вказаним ім’ям.

Для мого проєкту потрібно кілька моделей даних. Спочатку, я створила клас Volunteer, що наслідує клас User. Це робиться для того, щоб було зручно додавати інші класи користувачів, такі як Customer.

```
 public class User {
    @Id
    @GeneratedValue
    String id;
    String name;
    String surname;
    @Email(regexp = "^(?=.{1,256}$)[a-zA-Z0-9!#$%&'*+/=?^_`{|}~-
]+(?:\\.[a-zA-Z0-9!#$%&'*+/=?^_`{|}~-]+)*@[a-zA-Z0-9](?:[a-zA-
Z0-9-]{0,64}[a-zA-Z0-9])?(?:\\.[a-zA-Z]{2,})$")
    String email;
    @Pattern(regexp = "^[+]{1}(?:[0-9\\-
\\(\\)\\/\\.]\\s?){6,15}[0-9]{1}$")
    String phone;

    public User(String name, String surname, String email,
String phone) {
        this.name = name;
        this.surname = surname;
        this.email = email;
        this.phone = phone;
    }

    public User() {
    }
}
```

Ще один клас який потрібний для розробки – це Advertisement. Частина коду цього класу виглядає так:

```
     @Container(containerName = "Advertisements")
public class Advertisement {
    @Id
    @GeneratedValue
    String id;
         @NotNull
    String name;
    @Enumerated(EnumType.STRING)
    Category category;
    @JsonProperty("isOpen")
    Boolean isOpen;
         @NotNull
    @Valid
    Customer customer;
         @NotNull
    Description description;
    ArrayList<Volunteer> responded;

    public Advertisement(String name, Category category, Boolean
isOpen, Customer customer, Description description,
ArrayList<Volunteer> responded) {
        this.name = name;
        this.category = category;
        this.isOpen = isOpen;
        this.customer = customer;
        this.description = description;
        this.responded = responded;
    }
     }
```

3.5 ENUM

У Java ENUM є типом даних, що містить константи заранні значення. ENUM-и використовуються для створення зручних та читабельних списків констант з додатковими функціональними можливостями.

ENUM можна використовувати для опису станів деякого об'єкта, у випадку мого проєкту, їх було зручно використати для опису категорії оголошення та вмінь волонтерів. Реалізовано це було таким чином:

```
    public enum Category {
    CLOTHES, FOOD, DRIVING, EQUIPMENT
}
```

```
    public enum Skill {
    COOKING, DRIVING, FUNDING, SHOPPING
}
```

3.6 Контролери і репозиторії

Контролер та репозиторій – це дві ключові складові патерну MVC (Model-View-Controller) в програмуванні. Контролер відповідає за обробку запитів та передачу даних між моделлю та відображенням, тоді як репозиторій забезпечує доступ до даних та зберігання їх у базі даних.

У випадку з CosmosDB, ми можемо створити репозиторій, який розширює CosmosRepository, щоб забезпечити доступ до бази даних CosmosDB.

Для реалізації проєкту мені достатньо два контролера: AdvertisementController та VolunteerController.

Анотація RequestMapping у Java використовується для вказання URL-адреси, за якою буде доступний деякий метод або контролер у веб-додатку, що побудований на фреймворку Spring.

Ця анотація дозволяє пов'язати HTTP-запит з певним методом або контролером, який повинен обробити цей запит. Використовуючи анотацію RequestMapping, можна вказати такі параметри як HTTP-метод, шлях до ресурсу, типи параметрів, що передаються у запиті та інші опції.

Для AdvertisementContoller я вказую шлях за замовчуванням, адже список оголошень буде відображатись на головній сторінці сайту.

```
     @Controller
@RequestMapping("/")
public class AdvertisementController {
    AdvertisementRepository advertisementRepository;

    @Autowired
    public AdvertisementController(AdvertisementRepository
advertisementRepository) {
        this.advertisementRepository = advertisementRepository;
    }
     …
```

Репозиторій AdvertisementRepository розширює CosmosRepository, тому, створений репозиторій надає можливість виконання простих операцій таких як додавання і видалення об’єктів, отримання даних про них, адже наслідує їх з інтерфейсу CosmosRepository. AdvertisementRepository має такий вигляд:

```
     @Repository
public interface AdvertisementRepository extends
CosmosRepository<Advertisement, String> {
}
```

Аналогічні речі відбуваються з VolunteerController та VolunteerRepository.

3.7 Data Transfer Objects

DTO (Data Transfer Object) – це об'єкт, який використовується для передачі даних між різними частинами програмного забезпечення. DTO дозволяє розділити модель даних на дві частини: доменну модель та представлення (presentation layer).

Після створення ENUM ми стикаємось з проблемою при бажанні вивести інформацію про волонтера або оголошення, адже категорія водіння буде виводитись як “DRIVING”. Це мене не влаштовує, адже сайт має бути повністю українською мовою. Для того, щоб на фронтенд відправлявся український текст, буде зручно використати DTO – Data Transfer Object.

Реалізовано це було створенням AdvertisementDTO та VolunteerDTO.

```
     @Data
public class AdvertisementDTO {
    String id;
    String name;
    String category;
    Boolean isOpen;
    Customer customer;
    Description description;
    ArrayList<Volunteer> responded;

    public AdvertisementDTO(String id, String name, Category
category, Boolean isOpen, Customer customer, Description
description, ArrayList<Volunteer> responded) {
        this.id = id;
        this.name = name;
        this.category = categoryToString(category);
        this.isOpen = isOpen;
        this.customer = customer;
        this.description = description;
        this.responded = responded;
    }

    private String categoryToString(Category category) {
        return switch (category) {
            case CLOTHES -> "ОДЯГ";
            case FOOD -> "ЇЖА";
            case DRIVING -> "ВОДІННЯ";
            case EQUIPMENT -> "ТЕХНІКА";
        };
    }
}
```

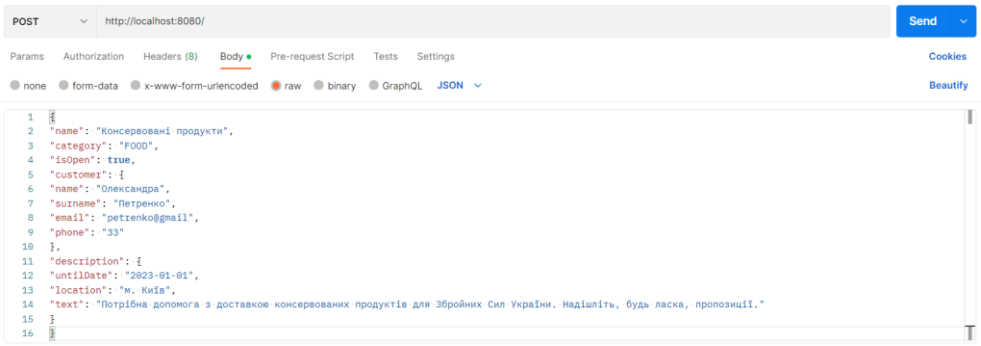
3.8 Валідація

У коді вище можна помітити, що доволі часто використовуються анотації для валідації полів. Вона використовується у контролері, при доданні нового елемента, а виглядає це так:

```
    @PostMapping
public ResponseEntity<Advertisement> addAdvertisements(@Valid
@RequestBody Advertisement advertisement) {
    Set<ConstraintViolation<Advertisement>> violations =
validator.validate(advertisement);
    if (!violations.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

    Advertisement newAdvertisement =
advertisementRepository.save(advertisement);
    return new ResponseEntity<>(newAdvertisement,
HttpStatus.CREATED);
}
```

Тут перевіряється чи є поля, які порушують встановлені нами правила, і якщо такі є, то повертаю http статус про поганий запит. На даному етапі протестувати валідацію можна за допомогою Postman:



Після введення неправильного номеру телефону замовника та його email адреси, у відповідь отримуємо помилку:

Більш детальну інформацію про неї можна отримати у консолі.

3.9 Створення проєкту Angular

Створення проєкту Angular можна розділити на декілька кроків:

1. Встановлення необхідних компонентів. Перш за все, необхідно встановити Node.js та npm (Node Package Manager). Для цього можна завантажити інсталятор з офіційного сайту https://nodejs.org. Також можна встановити Angular CLI (Command Line Interface) з використанням команди:

npm install -g @angular/cli

2. Створення нового проєкту за допомогою команди: ng new project-name

3. Запуск проєкту за допомогою команди: ng serve

Ця команда запустить проєкт та відкриє його у браузері за адресою http://localhost:4200/.

3.10 Компоненти Angular

У Angular компоненти – це основні будівельні блоки веб-додатка, які містять логіку та представлення веб-сторінок. Компоненти в дозволяють розділити веб-сторінку на незалежні блоки з власними функціональними можливостями та розміткою.

Кожен компонент має свої властивості та методи, які дозволяють взаємодіяти з користувачем та маніпулювати даними на веб-сторінці. Також в Angular можна передавати дані між компонентами та взаємодіяти з іншими компонентами на веб-сторінці.

Для шапки та підвалу сторінки створюються окремі компоненти та використовуються в app.component.html, щоб бути присутніми на кожній сторінці:

```
    <app-top-bar></app-top-bar>

<div class="container">
  <router-outlet (activate)="resetPosition();"></router-outlet>
</div>

<app-footer></app-footer>
```

Так само й для кожної сторінки створюється компонент, або якоїсь її частини. Наприклад, у компоненті main використовується компонент request з параметрами:

```
    <div                                class="request-container">
  <div  class="request-item"  *ngFor="let  ad  of  advertisements">
    <app-request
      [id]="ad.id"
      [isOpen]="ad.isOpen"
      [name]="ad.name"
      [category]="ad.category"
      [customer]="ad.customer"
      [description]="ad.description"
    ></app-request>
  </div>
</div>
```

Файл typescript у компоненті використовується для опису логіки компонента на мові програмування TypeScript. TypeScript - це розширення JavaScript, яке додає до мови деякі нові функції, які дозволяють зробити код більш організованим та ефективним.

У файлі typescript компонента можна визначати змінні, методи та властивості, які використовуються в шаблоні.

```
    export class RequestComponent {
  @Input() isOpen: boolean;
  @Input() id: string;
  @Input() name: string;
  @Input() category: string;
  @Input() customer: Customer;
```

```
  @Input() description: Description;

  constructor() {
  }

  ngOnInit(): void {
  }
}
```

3.11 Класи в Angular

У Angular класи використовуються для опису об'єктів та їхньої логіки, які використовуються в компонентах, сервісах, директивах та інших частинах веб-додатка. Класи містять поля, методи та конструктори, які використовуються для опису структури об'єктів та їхньої логіки. Класи можуть містити публічні та приватні поля та методи, які використовуються в компонентах для маніпулювання даними та взаємодії з іншими компонентами веб-додатка.

Класи в Angular дозволяють створювати сервіси, які надають спільний функціонал для декількох компонентів, директив та інших частин веб-додатка.

Для реалізації додатку було достатньо було створити наступні класи: advertisement, customer, description, volunteer. Приклад реалізації такого класу:

```
    export class Advertisement {
  id: string;
  isOpen: boolean;
  name: string;
  category: string;
  customer: Customer;
  description: Description;
}
```

3.12 Сервіси в Angular

Сервіси (services) в Angular – це класи, які можуть бути використані для зберігання та обробки даних в рамках додатку. Вони є важливим елементом будь-якого Angular-додатку та можуть бути використані для взаємодії з сервером, зберігання стану додатку та додаткових налаштувань.

У цьому додатку є два сервіси: RequestService та VolunteerService. RequestService виглядає так:

```
    @Injectable({
  providedIn: 'root'
})
export class RequestService {
  private baseURL = "http://localhost:8080/";

  constructor(private http: HttpClient) {
  }

  getAdsList(): Observable<Advertisement[]> {
    return this.http.get<Advertisement[]>(`${this.baseURL}`);
  }

  getAd(id: String): Observable<Advertisement> {
    return
this.http.get<Advertisement>(`${this.baseURL}request/${id}`);
  }
}
```

Він використовується для взаємодії з сервером для отримання списку оголошень або окремого оголошення за його ідентифікатором.

Перший рядок коду @Injectable ({ providedIn: 'root' }) вказує Angular, що цей сервіс повинен бути створений один раз, і інжектуватись в кореневий модуль додатку (AppModule). Це дозволяє сервісу бути доступним для використання в будь-якому місці додатку.

Клас містить приватне поле baseURL, яке містить базову адресу сервера, що буде використовуватися для взаємодії з API.

Конструктор класу приймає екземпляр HttpClient, який буде використовуватися для виконання HTTP запитів до сервера.

Висновок

У результаті написання курсової роботи було розроблено веб-додаток VolunteerConnect, який дозволяє волонтерам переглядати оголошення людей, що потребують допомоги та викладати свої дані, щоб інші могли зв’язуватись з ними.

Для написання коду використовувалась Java, що пропонує різноманітні фреймворки та бібліотеки, що допомагають значно скоротити час розробки веб-додатків та підвищити їхню ефективність та безпеку. Один з найпопулярніших фреймворків для розробки веб-додатків на Java – Spring Framework, був використаний у даній курсовій роботі.

Для збереження інформації була використана база даних Azure CosmosDB. Під час її використання та аналізу, були виявлені такі переваги та недоліки:

- вона дозволяє масштабувати базу даних горизонтально, що дозволяє збільшувати її розмір та продуктивність за необхідності.
- вона дозволяє зберігати різноманітні типи даних (структуровані, напівструктуровані, невпорядковані) у різних форматах (JSON, BSON, XML), що дозволяє працювати з даними в зручний для розробника формат.

Серед недоліків:

- у порівнянні з іншими рішеннями, Azure CosmosDB може бути досить дорогим, особливо для малих проектів з обмеженим бюджетом.
- вона має обмежену підтримку запитів, що може ускладнити розробку додатків та додати складнощів до виконання запитів до бази даних.

Враховуючи переваги та недоліки, Azure CosmosDB може бути добрим вибором для великих та складних проєктів з високими вимогами до продуктивності та доступності даних, але може бути неоптимальним для менших проектів з обмеженим бюджетом та простими вимогами до бази даних.

Узагальнюючи, написання курсової роботи дало змогу познайомитись з популярними технологіями та інструментами для розробки веб-додатків, проаналізувати їх переваги та недоліки, а також здобути практичні навички їх використання.

Список джерел та літератури

1. Microsoft Azure [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/azure/cloud-adoption-framework/get-started/what-is-azure
2. Microsoft Azure [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-gb/products/cosmos-db
3. Microsoft [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/quickstart-portal
4. Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/en/download/help/whatis_java.html
5. Spring Docs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html
6. JavaTPoint [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/postman
7. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/HTML
8. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/CSS
9. Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/guide/architecture

Додатки

Пакет org.example.config

CosmosProperties.java

Клас властивостей для налаштування підключення до бази даних Azure Cosmos DB.

```java
package org.example.config;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties(prefix = "azure.cosmos")
public class CosmosProperties {
    private String uri;

    private String key;

    private String secondaryKey;

    private String database;

    private boolean queryMetricsEnabled;

    private boolean responseDiagnosticsEnabled;


    public String getUri() {
        return uri;
    }

    public void setUri(String uri) {
        this.uri = uri;
    }

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

    public String getSecondaryKey() {
        return secondaryKey;
    }

    public void setSecondaryKey(String secondaryKey) {
        this.secondaryKey = secondaryKey;
    }

    public void setDatabase(String database) {
        this.database = database;
    }

    public String getDatabase() {
        return database;
    }

    public boolean isQueryMetricsEnabled() {
        return queryMetricsEnabled;
    }
```

```
    public void setQueryMetricsEnabled(boolean enableQueryMetrics) {
        this.queryMetricsEnabled = enableQueryMetrics;
    }

    public boolean isResponseDiagnosticsEnabled() {
        return responseDiagnosticsEnabled;
    }

    public void setResponseDiagnosticsEnabled(boolean
enableResponseDiagnostics) {
        this.responseDiagnosticsEnabled = enableResponseDiagnostics;
    }
}
```

CosmosSpringConfiguration.java

Це файл з класом конфігурації Spring, який використовується для налаштування підключення до бази даних Azure Cosmos DB.

```
package org.example.config;

import com.azure.cosmos.CosmosClientBuilder;
import com.azure.cosmos.DirectConnectionConfig;
import com.azure.cosmos.GatewayConnectionConfig;
import com.azure.spring.data.cosmos.config.AbstractCosmosConfiguration;
import com.azure.spring.data.cosmos.config.CosmosConfig;
import com.azure.spring.data.cosmos.core.ResponseDiagnostics;
import com.azure.spring.data.cosmos.core.ResponseDiagnosticsProcessor;
import
com.azure.spring.data.cosmos.repository.config.EnableCosmosRepositories;
import
com.azure.spring.data.cosmos.repository.config.EnableReactiveCosmosRepositori
es;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import
org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.lang.Nullable;
import
org.springframework.validation.beanvalidation.LocalValidatorFactoryBean;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import javax.validation.Validator;

@Configuration
@EntityScan
@EnableConfigurationProperties(CosmosProperties.class)
@EnableCosmosRepositories(basePackages = "org.example.repository")
@EnableReactiveCosmosRepositories
@PropertySource("classpath:application.properties")
public class CosmosSpringConfiguration extends AbstractCosmosConfiguration
implements WebMvcConfigurer {
    private static final Logger logger =
LoggerFactory.getLogger(CosmosSpringConfiguration.class);

    private CosmosProperties properties;
```

```
    CosmosSpringConfiguration(CosmosProperties properties) {
        this.properties = properties;
    }

    @Bean
    public CosmosClientBuilder cosmosBuildClient() {
        DirectConnectionConfig directConnectionConfig =
DirectConnectionConfig.getDefaultConfig();

        //use this for gateway connection
        GatewayConnectionConfig gatewayConnectionConfig =
GatewayConnectionConfig.getDefaultConfig();

        return new CosmosClientBuilder()
                .endpoint(properties.getUri())
                .key(properties.getKey())
                .directMode(directConnectionConfig);
    }

    @Bean
    public CosmosConfig cosmosConfig() {
        return CosmosConfig.builder()
                .responseDiagnosticsProcessor(new
ResponseDiagnosticsProcessorImplementation(this.properties))
                .enableQueryMetrics(properties.isQueryMetricsEnabled())
                .build();
    }

    @Bean
    public Validator localValidatorFactoryBean() {
        return new LocalValidatorFactoryBean();
    }

    private static class ResponseDiagnosticsProcessorImplementation
implements ResponseDiagnosticsProcessor {

        private CosmosProperties properties;

        ResponseDiagnosticsProcessorImplementation(CosmosProperties
properties) {
            this.properties = properties;
        }

        @Override
        public void processResponseDiagnostics(@Nullable ResponseDiagnostics
responseDiagnostics) {
            if (this.properties.isResponseDiagnosticsEnabled()) {
                logger.info("Response Diagnostics {}", responseDiagnostics);
            }
        }
    }

    @Override
    protected String getDatabaseName() {
        return properties.getDatabase();
    }

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("http://localhost:4200")
                .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
                .allowedHeaders("Origin", "Access-Control-Allow-Origin",
```

```
"Content-Type",
                        "Accept", "Authorization", "Origin, Accept", "X-
Requested-With",
                        "Access-Control-Request-Method", "Access-Control-
Request-Headers")
                .exposedHeaders("Origin", "Content-Type", "Accept",
"Authorization",
                        "Access-Control-Allow-Origin", "Access-Control-Allow-
Origin", "Access-Control-Allow-Credentials")
        ;
    }
}
```

SpringMvcServletInitializer.java

Використовується для ініціалізації та налаштування додатку, який побудований з використанням Spring MVC. Він відповідає за реєстрацію фронт-контролера додатку, який обробляє HTTP-запити від клієнтів, та налаштування інших компонентів, які необхідні для роботи додатку.

```
package org.example.config;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;

import org.springframework.web.filter.HiddenHttpMethodFilter;
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherSer
vletInitializer;

public class SpringMvcServletInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{CosmosSpringConfiguration.class};
    }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }

    @Override
    public void onStartup(ServletContext aServletContext) throws
ServletException {
        super.onStartup(aServletContext);
        registerHiddenFieldFilter(aServletContext);
    }

    private void registerHiddenFieldFilter(ServletContext aContext) {
        aContext.addFilter("hiddenHttpMethodFilter",
                new HiddenHttpMethodFilter()).addMappingForUrlPatterns(null,
true, "/*");
    }
}
```

Пакет org.example.controller

В цьому пакеті розміщені класи-контролери, що оброблюють HTTP-запити веб-додатка і повертають HTTP-відповіді.

AdvertisementController.java

```
package org.example.controller;

import org.example.dto.AdvertisementDTO;
import org.example.entity.Advertisement;
import org.example.entity.Customer;
import org.example.entity.Volunteer;
import org.example.repository.AdvertisementRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import javax.validation.ConstraintViolation;
import javax.validation.Valid;
import javax.validation.Validator;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.Set;

@Controller
@RequestMapping("/")
public class AdvertisementController {
    AdvertisementRepository advertisementRepository;
    private Validator validator;

    @Autowired
    public AdvertisementController(AdvertisementRepository
advertisementRepository, Validator validator) {
        this.advertisementRepository = advertisementRepository;
        this.validator = validator;
    }

    @GetMapping
    public ResponseEntity<List<AdvertisementDTO>> getAllAdvertisements() {
        List<Advertisement> advertisements = new ArrayList<>();
        advertisementRepository.findAll().forEach(advertisements::add);

        List<AdvertisementDTO> advertisementDTOS = advertisements
                .stream()
                .map(this::toDTO)
                .toList();

        return new ResponseEntity<>(advertisementDTOS, HttpStatus.OK);
    }

    @GetMapping("request/{id}")
    public ResponseEntity<AdvertisementDTO>
getAdvertisement(@PathVariable("id") String id) {
        Optional<Advertisement> ad = advertisementRepository.findById(id);
        if (ad.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        AdvertisementDTO advertisementDTO = toDTO(ad.get());
        return new ResponseEntity<>(advertisementDTO, HttpStatus.OK);
```

```
    }

    @PostMapping
    public ResponseEntity<Advertisement> addAdvertisements(@Valid
@RequestBody Advertisement advertisement) {
        Set<ConstraintViolation<Advertisement>> violationsAd =
validator.validate(advertisement);
        Set<ConstraintViolation<Customer>> violationsCustomer =
validator.validate(advertisement.getCustomer());
        if (!violationsAd.isEmpty() || !violationsCustomer.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }

        Advertisement newAdvertisement =
advertisementRepository.save(advertisement);
        return new ResponseEntity<>(newAdvertisement, HttpStatus.CREATED);
    }

    @DeleteMapping("request/{id}")
    public ResponseEntity<Volunteer> deleteAdvertisements(@PathVariable("id")
String id) {
        if (advertisementRepository.findById(id).isEmpty()) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
        advertisementRepository.deleteById(id);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    private AdvertisementDTO toDTO(Advertisement advertisement) {
        return new AdvertisementDTO(advertisement.getId(),
                advertisement.getName(),
                advertisement.getCategory(),
                advertisement.getIsOpen(),
                advertisement.getCustomer(),
                advertisement.getDescription(),
                advertisement.getResponded());
    }
}
```

VolunteerController.java

```
package org.example.controller;

import org.example.dto.VolunteerDTO;
import org.example.entity.Volunteer;
import org.example.repository.VolunteerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import javax.validation.ConstraintViolation;
import javax.validation.Valid;
import javax.validation.Validator;
import java.util.*;

@Controller
@RequestMapping("/volunteers")
public class VolunteerController {
    private VolunteerRepository volunteerRepository;
    private Validator validator;
```

```
    @Autowired
    public VolunteerController(VolunteerRepository volunteerRepository,
Validator validator) {
        this.volunteerRepository = volunteerRepository;
        this.validator = validator;
    }

    @GetMapping
    public ResponseEntity<List<VolunteerDTO>> getAllVolunteer() {
        List<Volunteer> volunteers = new ArrayList<>();

volunteerRepository.findAll().iterator().forEachRemaining(volunteers::add);

        List<VolunteerDTO> volunteerDTOS = volunteers
                .stream()
                .map(this::toDTO)
                .toList();

        return new ResponseEntity<>(volunteerDTOS, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<VolunteerDTO> getVolunteer(@PathVariable("id")
String id) {
        Optional<Volunteer> volunteer = volunteerRepository.findById(id);
        if (volunteer.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
        VolunteerDTO volunteerDTO = toDTO(volunteer.get());
        return new ResponseEntity<>(volunteerDTO, HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<Volunteer> addVolunteer(@Valid @RequestBody
Volunteer volunteer) {
        Set<ConstraintViolation<Volunteer>> violations =
validator.validate(volunteer);
        if (!violations.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }

        Volunteer newVolunteer = volunteerRepository.save(volunteer);
        return new ResponseEntity<>(newVolunteer, HttpStatus.CREATED);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Volunteer> deleteVolunteer(@PathVariable("id")
String id) {
        if (volunteerRepository.findById(id).isEmpty()) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
        volunteerRepository.deleteById(id);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    private VolunteerDTO toDTO(Volunteer volunteer) {
        return new VolunteerDTO(volunteer.getId(),
                volunteer.getName(),
                volunteer.getSurname(),
                volunteer.getEmail(),
                volunteer.getPhone(),
                volunteer.getLocation(),
```

```
                volunteer.getSkills(),
                volunteer.getAbout()
        );
    }
}
```

Пакет org.example.dto

В цьому пакеті зібрані DTO для сутностей. Data Transfer Object - це об'єкт, який використовується для передачі даних між різними частинами програмного забезпечення.

AdvertisementDTO.java

```
package org.example.dto;

import lombok.Data;
import org.example.entity.Customer;
import org.example.entity.Description;
import org.example.entity.Volunteer;
import org.example.enums.Category;

import java.util.ArrayList;

@Data
public class AdvertisementDTO {
    String id;
    String name;
    String category;
    Boolean isOpen;
    Customer customer;
    Description description;
    ArrayList<Volunteer> responded;

    public AdvertisementDTO(String id, String name, Category category,
Boolean isOpen, Customer customer, Description description,
ArrayList<Volunteer> responded) {
        this.id = id;
        this.name = name;
        this.category = categoryToString(category);
        this.isOpen = isOpen;
        this.customer = customer;
        this.description = description;
        this.responded = responded;
    }

    private String categoryToString(Category category) {
        return switch (category) {
            case CLOTHES -> "ОДЯГ";
            case FOOD -> "ЇЖА";
            case DRIVING -> "ВОДІННЯ";
            case EQUIPMENT -> "ТЕХНІКА";
        };
    }

 }
```

VolunteerDTO.java

```
package org.example.dto;

import lombok.Data;
import org.example.enums.Skill;

import java.util.ArrayList;

@Data
public class VolunteerDTO {
    String id;
    String name;
    String surname;
    String email;
    String phone;
    String location;
    ArrayList<String> skills;
    String about;

    public VolunteerDTO(String id, String name, String surname, String email,
String phone, String location, ArrayList<Skill> skills, String about) {
        this.id = id;
        this.name = name;
        this.surname = surname;
        this.email = email;
        this.phone = phone;
        this.location = location;
        this.skills = new ArrayList<>(
                skills
                        .stream()
                        .map(this::skillToString)
                        .toList()
        );
        this.about = about;
    }

    private String skillToString(Skill skill) {
        return switch (skill) {
            case COOKING -> "ПРИГОТУВАННЯ ЇЖІ";
            case DRIVING -> "ВОДІННЯ";
            case FUNDING -> "ФІНАНСУВАННЯ";
            case SHOPPING -> "ПОКУПКИ";
        };
    }
}
```

Пакет org.example.entity

В цьому пакеті зібрані сутності, які потрібні для збереження та передачі даних.

Advertisement.java

```
package org.example.entity;

import com.azure.spring.data.cosmos.core.mapping.Container;
import com.azure.spring.data.cosmos.core.mapping.GeneratedValue;
import com.fasterxml.jackson.annotation.JsonProperty;
import org.example.enums.Category;
import org.springframework.data.annotation.Id;

import javax.persistence.EnumType;
```

```
import javax.persistence.Enumerated;
import javax.validation.Valid;
import javax.validation.constraints.NotNull;
import java.sql.Timestamp;
import java.time.LocalDateTime;
import java.util.ArrayList;

@Container(containerName = "Advertisements")
public class Advertisement {
    @Id
    @GeneratedValue
    String id;
    @NotNull
    String name;
    @Enumerated(EnumType.STRING)
    Category category;
    @JsonProperty("isOpen")
    Boolean isOpen;
    @NotNull
    @Valid
    Customer customer;
    @NotNull
    Description description;
    ArrayList<Volunteer> responded;

    public Advertisement(String name, Category category, Boolean isOpen,
Customer customer, Description description, ArrayList<Volunteer> responded) {
        this.name = name;
        this.category = category;
        this.isOpen = isOpen;
        this.customer = customer;
        this.description = description;
        this.responded = responded;
    }

    public Advertisement(String id, String name, Category category, Boolean
isOpen, Customer customer, Description description, ArrayList<Volunteer>
responded) {
        this.id = id;
        this.name = name;
        this.category = category;
        this.isOpen = isOpen;
        this.customer = customer;
        this.description = description;
        this.responded = responded;
    }

    public Advertisement() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
```

```
        this.id = id;
    }

    public Category getCategory() {
        return category;
    }

    public void setCategory(Category category) {
        this.category = category;
    }

    public Boolean getIsOpen() {
        if
(this.description.getUntilDate().before(Timestamp.valueOf(LocalDateTime.now()
))) {
            this.isOpen = false;
        }
        if (isOpen == null) {
            isOpen = true;
        }
        return isOpen;
    }

    public void setIsOpen(Boolean open) {
        isOpen = open;
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public Description getDescription() {
        return description;
    }

    public void setDescription(Description description) {
        this.description = description;
    }

    public ArrayList<Volunteer> getResponded() {
        return responded;
    }

    public void setResponded(ArrayList<Volunteer> responded) {
        this.responded = responded;
    }
}
```

Customer.java

```
package org.example.entity;


public class Customer extends User {

    public Customer(String id, String name, String surname, String email,
String phone) {
        super(name, surname, email, phone);
```

```
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

Description.java

```
package org.example.entity;

import java.util.Date;

public class Description {
    Date untilDate;
    String location;
    String text;

    public Description(Date untilDate, String location, String text) {
        this.untilDate = untilDate;
        this.location = location;
        this.text = text;
    }

    public Description() {
    }
```

```
    public Date getUntilDate() {
        return untilDate;
    }

    public void setUntilDate(Date untilDate) {
        this.untilDate = untilDate;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}
```

User.java

```
package org.example.entity;

import com.azure.spring.data.cosmos.core.mapping.GeneratedValue;
import org.springframework.data.annotation.Id;

import javax.validation.constraints.Email;
import javax.validation.constraints.Pattern;

public class User {
    @Id
    @GeneratedValue
    String id;
    String name;
    String surname;
    @Email(regexp = "^(?=.{1,256}$)[a-zA-Z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\.[a-
zA-Z0-9!#$%&'*+/=?^_`{|}~-]+)*@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,64}[a-zA-Z0-
9])?(?:\\.[a-zA-Z]{2,})$")
    String email;
    @Pattern(regexp = "^[+]{1}(?:[0-9\\-\\(\\)\\/\\.]\\s?){6,15}[0-9]{1}$")
    String phone;

    public User(String name, String surname, String email, String phone) {
        this.name = name;
        this.surname = surname;
        this.email = email;
        this.phone = phone;
    }

    public User() {
    }
}
```

Volunteer.java

```
package org.example.entity;

import com.azure.spring.data.cosmos.core.mapping.Container;
import org.example.enums.Skill;

import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import java.util.ArrayList;

@Container(containerName = "Volunteers")
public class Volunteer extends User {

    String location;
    @Enumerated(EnumType.STRING)
    ArrayList<Skill> skills;
    String about;

    public Volunteer(String name, String surname, String email, String phone,
String location, ArrayList<Skill> skills, String about) {
        super(name, surname, email, phone);
        this.location = location;
        this.skills = skills;
        this.about = about;
    }

    public Volunteer() {
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
```

```
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public ArrayList<Skill> getSkills() {
        return skills;
    }

    public void setSkills(ArrayList<Skill> skills) {
        this.skills = skills;
    }

    public String getAbout() {
        return about;
    }

    public void setAbout(String about) {
        this.about = about;
    }
}
```

Пакет org.example.enums

В цьому пакеті зібрані enum, дані, що містить константні значення.

Category.java

```
package org.example.enums;

public enum Category {
    CLOTHES, FOOD, DRIVING, EQUIPMENT
}
```

Skill.java

```
package org.example.enums;

public enum Skill {
    COOKING, DRIVING, FUNDING, SHOPPING
}
```

Пакет org.example.repository

В цьому пакеті зібрані репозиторії, що забезпечують доступ до даних та зберігання їх у базі даних. Вони розширюють CosmosRepository, щоб забезпечити доступ до бази даних CosmosDB.

AdvertisementRepository.java

```java
package org.example.repository;

import com.azure.spring.data.cosmos.repository.CosmosRepository;
import org.example.entity.Advertisement;
import org.springframework.stereotype.Repository;

@Repository
public interface AdvertisementRepository extends
CosmosRepository<Advertisement, String> {
}
```

VolunteerRepository.java

```java
package org.example.repository;

import com.azure.spring.data.cosmos.repository.CosmosRepository;
import org.example.entity.Volunteer;
import org.springframework.stereotype.Repository;

@Repository
public interface VolunteerRepository extends CosmosRepository<Volunteer,
String> {
}
```

Пакет org.example

Main.java

```java
package org.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Main {
    public static void main(String[] args) {
        SpringApplication.run(Main.class, args);
    }
}
```

Resources

Application.properties

Це файл з параметрами підключення до бази даних Azure Cosmos DB, які використовуються у додатках на Spring Framework для забезпечення з'єднання з базою даних.

```
azure.cosmos.uri=https://volunteerconnect.documents.azure.com:443/
azure.cosmos.database=VolunteerConnect
azure.cosmos.key=qDlQj4gjS9T3Qud9Ns8nPE7NSuaJsUnJ0og1Ptv4DuClbxZOg42C532oeN3s
04xW7BzWyhz26OyeACDbP0klGg==
azure.cosmos.queryMetricsEnabled=true
azure.cosmos.responseDiagnosticsEnabled=true
```

pom.xml

Файл pom.xml використовується для керування версіями залежностей, містить інформацію про проект, таку як залежності, налаштування збірки, плагіни.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <groupId>org.example</groupId>
    <artifactId>finalProject</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>19</maven.compiler.source>
        <maven.compiler.target>19</maven.compiler.target>
        <java.version>17</java.version>
        <spring-cloud-azure.version>4.3.0</spring-cloud-azure.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>com.azure.spring</groupId>
            <artifactId>spring-cloud-azure-starter-data-cosmos</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>

        <dependency>
            <groupId>javax.persistence</groupId>
            <artifactId>javax.persistence-api</artifactId>
            <version>2.2</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/log4j/log4j -->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
```

```
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.azure.spring</groupId>
                <artifactId>spring-cloud-azure-dependencies</artifactId>
                <version>${spring-cloud-azure.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```