Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут комп’ютерних систем
Кафедра інформаційних систем

Пояснювальна записка
до курсової роботи з дисципліни
«Об’єктно-орієнтоване програмування»
на тему: «Система для організації виставок»

Виконала:
Студентка групи АІ-233
Унтілова Є. О.
Перевірив:
Годовиченко М.А.

Одеса 2025

Вступ

У сучасних умовах розвитку індустрії організації виставок спостерігається динамічне зростання попиту на цифровізацію та автоматизацію процесів підготовки, проведення й супроводу заходів. Це обумовлено необхідністю оперативно координувати роботу великої кількості учасників, забезпечувати ефективну комунікацію між організаторами, експонентами та відвідувачами, а також вести точний облік ресурсів і фінансових операцій. Виставкова діяльність охоплює широкий спектр подій — від тематичних виставок і ярмарків до масштабних міжнародних експозицій, що вимагає гнучких і надійних інструментів управління.

Організація виставок передбачає вирішення низки комплексних завдань: планування концепції заходу, підбір і оформлення експозицій, розробку сценаріїв, реєстрацію учасників, маркетинговий супровід, підготовку інформаційних матеріалів, а також технічне забезпечення й контроль за виконанням усіх етапів підготовки. Традиційні методи обліку та координації, засновані на паперових документах чи неструктурованих електронних таблицях, вже не відповідають сучасним вимогам до швидкості, прозорості та якості роботи.

Впровадження сучасної інформаційної системи для організації виставок дозволяє автоматизувати ключові бізнес-процеси: електронну реєстрацію учасників і відвідувачів, управління заявками, формування розкладу подій, моніторинг виконання завдань, аналітику ефективності заходу та забезпечення високого рівня сервісу для всіх категорій користувачів. Така система має підтримувати інтеграцію з іншими інформаційними платформами, гарантувати захист персональних даних, бути масштабованою й адаптивною до потреб різних форматів виставкової діяльності.

Метою цієї роботи є розробка ефективної, надійної та гнучкої системи для організації виставок, яка дозволить автоматизувати основні етапи підготовки та проведення заходів, підвищити якість обслуговування учасників і відвідувачів, а

також забезпечити прозорість і контроль усіх процесів. У пояснювальній записці розглядаються основні етапи створення такого програмного забезпечення: аналіз предметної області, проєктування архітектури, реалізація функціональних модулів і оцінка ефективності впровадження системи.

1. Аналіз предметної області

Система для організації виставок розроблена з метою автоматизації ключових процесів у сфері проведення мистецьких, культурних та тематичних заходів. Основна задача такої системи — забезпечити ефективну взаємодію між організаторами, митцями, експонатами, локаціями та відвідувачами, а також оптимізувати облік ресурсів, планування подій і аналітику. У системі зберігається та обробляється інформація про митців, експонати, виставки, локації та перегляди, що дозволяє підвищити якість організації заходів, зменшити ризики помилок і забезпечити прозорість усіх етапів підготовки.

Ключовими об’єктами системи є митці, експонати, виставки, локації та записи участі експонатів у виставках (ExhibitionEntry). Митець — це учасник, що створює експонати та може брати участь у багатьох виставках, зберігається інформація про його ім’я та країну. Експонат — унікальний об’єкт мистецтва чи експозиції, який має назву, опис, автора (митця) та рік створення. Виставка — це подія з визначеними датами початку й завершення, що проходить у певній локації. Локація містить дані про назву та адресу місця проведення. Запис участі експоната у виставці (ExhibitionEntry) дозволяє відстежувати, які експонати представлені на конкретних заходах, а також переміщення об’єктів між різними локаціями.

Життєвий цикл організації виставки в системі включає кілька основних етапів. Спочатку адміністратор додає або оновлює інформацію про митців і експонати. Далі створюється виставка із зазначенням локації, дат проведення та тематичного спрямування. Після цього експонати призначаються до виставки, що дозволяє формувати експозицію відповідно до концепції заходу. Система забезпечує можливість переміщення експонатів між локаціями, ведення історії участі експонатів у різних виставках, а також облік переглядів і статистики відвідуваності.

Функціональні можливості системи охоплюють автоматизацію додавання, оновлення та видалення даних про митців, експонати, виставки й локації,

призначення експонатів на виставки, отримання списків експонатів певного року, митців за країною, поточних виставок, а також генерацію різноманітних аналітичних звітів. Звіти включають статистику по виставках, популярність локацій, аналіз участі митців та експонатів, що сприяє стратегічному плануванню і підвищенню якості організації заходів.

Для адміністраторів система надає зручний інтерфейс для управління всіма сутностями: митцями, експонатами, виставками та локаціями, а також інструменти для аналітики та контролю ефективності проведення заходів. Організатори можуть оперативно оновлювати інформацію, призначати експонати на виставки, отримувати статистику щодо участі митців та відвідуваності, що дозволяє приймати обґрунтовані рішення щодо майбутніх подій. Система також дозволяє відстежувати експонати без виставки, локації з найбільшою кількістю заходів та інші важливі показники.

Загалом, впровадження системи для організації виставок забезпечує прозорість і контроль усіх етапів підготовки й проведення заходів, зменшує вплив людського фактора, підвищує лояльність учасників і відвідувачів завдяки зручному та швидкому сервісу, а також сприяє підвищенню ефективності роботи організаторів і розвитку виставкової діяльності в цілому.

2. Проєктування системи

2.1 Опис сутностей

У системі управління орендою техніки визначено п’ять основних сутностей, які відображають ключові об’єкти предметної області:

Artist (Митець) — представляє автора експонатів, які беруть участь у виставках. Кожен митець має унікальний ідентифікатор (id), ім’я (name) та країну походження (country).

Exhibit (Експонат) — унікальний об’єкт мистецтва або експозиції, який може бути представлений на виставках. Містить ідентифікатор (id), назву (title), опис (description), посилання на автора (artist) та рік створення (yearCreated).

Exhibition (Виставка) — подія, під час якої експонати демонструються відвідувачам у певній локації протягом визначеного періоду. Містить ідентифікатор (id), назву (title), дату початку (startDate), дату завершення (endDate) та посилання на локацію проведення (location).

Location (Локація) — місце проведення виставки. Містить унікальний ідентифікатор (id), назву локації (name) та адресу (address).

ExhibitionEntry (Запис участі у виставці) — зв’язує конкретний експонат із певною виставкою, фіксуючи його участь у цій події. Містить ідентифікатор (id), посилання на виставку (exhibition) та експонат (exhibit).

2.2 Опис архітектури застосунку

Застосунок побудований за трирівневою архітектурою, яка включає три основні рівні:

Контролер (Controller) — цей рівень відповідає за прийом і обробку HTTP-запитів від клієнтів. Контролер отримує параметри запиту, виконує виклики до бізнес-логіки, що реалізована у сервісному шарі, і повертає результати у вигляді JSON-відповідей. Він виступає посередником між інтерфейсом користувача та внутрішніми компонентами системи.

Сервіс (Service) — тут реалізована основна бізнес-логіка застосунку. На цьому рівні відбуваються всі необхідні перевірки, обробка і трансформація даних, а також координація виконання операцій. Сервісний шар взаємодіє з репозиторіями для отримання або збереження інформації в базі даних і повертає дані у вигляді спеціальних об’єктів передачі даних (DTO), що забезпечує відокремлення внутрішньої структури даних від зовнішніх представлень.

Репозиторій (Repository) — цей рівень відповідає за роботу з базою даних. Використовуючи Spring Data JPA, репозиторії забезпечують абстракцію від безпосереднього написання SQL-запитів, надаючи готові методи для створення, оновлення, видалення та пошуку даних у сховищі.

Взаємодія між рівнями організована послідовно: контролер приймає запити від користувача і передає їх сервісу для обробки; сервіс виконує бізнес-логіку і звертається до репозиторіїв для роботи з даними; репозиторії безпосередньо взаємодіють із базою даних, забезпечуючи збереження та отримання інформації. Така структура сприяє розділенню відповідальностей, підвищує масштабованість і підтримуваність застосунку.

2.3 Опис REST API

Опис REST-запитів для системи організації виставок:

1. Додати митця — створює нового митця в системі, зберігає його ім’я та країну.
2. Отримати митців — повертає перелік усіх митців, що зберігаються у базі даних.
3. Оновити дані митця — дозволяє змінити інформацію про конкретного митця (наприклад, ім’я або країну).
4. Видалити митця — видаляє митця з системи за його унікальним ідентифікатором.
5. Додати експонат — створює новий експонат, вказуючи його назву, опис, автора та рік створення.
6. Отримати експонати — повертає список усіх експонатів у системі.
7. Оновити експонат — змінює дані про експонат, наприклад його назву, опис чи автора.
8. Видалити експонат — видаляє експонат із бази даних за ідентифікатором.
9. Додати виставку — створює нову виставку із зазначенням назви, дат проведення та локації.
10. Отримати всі виставки — повертає перелік усіх виставок, що були створені в системі.
11. Оновити виставку — дозволяє редагувати інформацію про виставку (наприклад, змінити дати або локацію).
12. Видалити виставку — видаляє виставку за її ідентифікатором.
13. Додати локацію — створює нову локацію з назвою та адресою для проведення виставок.
14. Отримати локації — повертає список усіх локацій, що доступні для проведення виставок.
15. Оновити локацію — змінює інформацію про локацію (наприклад, адресу або назву).
16. Призначити експонат на виставку — додає експонат до конкретної виставки, створюючи запис участі.
17. Видалити експонат з виставки — видаляє зв’язок між експонатом і виставкою, тобто експонат більше не бере участі у цій події.

18. Отримати експонати виставки — повертає перелік експонатів, що представлені на певній виставці.
19. Отримати виставки митця — повертає список виставок, у яких брав участь конкретний митець.
20. Отримати експонати певного року — дозволяє знайти експонати, створені у вказаному році.
21. Отримати статистику по виставках — формує аналітичну інформацію щодо проведених виставок (наприклад, кількість експонатів, митців, відвідуваність).
22. Отримати митців за країною — повертає список митців, що представляють певну країну.
23. Отримати локації з найбільшою кількістю виставок — формує рейтинг локацій за кількістю проведених у них виставок.
24. Отримати експонати без виставки — повертає перелік експонатів, які наразі не призначені до жодної виставки.
25. Отримати поточні виставки — повертає список виставок, що тривають на даний момент.

3. Реалізація програмного продукту

3.1 Моделі даних

Для відображення основних сутностей предметної області у програмному коді створено відповідні моделі (класи), які відповідають таблицям у базі даних. Кожна модель містить поля, що відображають атрибути сутності, а також зв’язки між сутностями (наприклад, зв’язок "один-до-багатьох" між митцем та експонатами). У проєкті реалізовані такі основні моделі: Artist, Exhibit, Exhibition, ExhibitionEntry та Location.

Клас Artist представляє митця, який створює експонати та бере участь у виставках. Модель містить основні атрибути митця, а також зв’язок із експонатами, що дозволяє відстежувати всі роботи, створені конкретним автором.

java-код для класу Artist:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Artist {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String country;

    @Column(length = 2000)
    private String bio;

    @OneToMany(mappedBy = "artist", cascade = CascadeType.ALL)
    private List<Exhibit> exhibits = new ArrayList<>();
}
```

Клас Exhibit описує експонат, який може бути представлений на виставках. Модель містить інформацію про назву, опис, тип, рік створення, а також зв’язок із митцем, який є автором експоната. Додатково реалізовано зв’язок із записами участі у виставках.

java-код для класу Exhibit:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Exhibit {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String description;
    private String type;
    private Integer yearCreated;

    @ManyToOne
    @JoinColumn(name = "artist_id")
    private Artist artist;

    @OneToMany(mappedBy = "exhibit", cascade = CascadeType.ALL)
    private List<ExhibitionEntry> entries = new ArrayList<>();
}
```

Клас Exhibition представляє виставку, яка проходить у певній локації протягом визначеного періоду. Модель містить атрибути, що описують назву, дати початку та завершення, а також зв’язок із локацією та записами участі експонатів.

java-код для класу Exhibition:

```
@Entity
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Exhibition {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private LocalDate startDate;
    private LocalDate endDate;

    @ManyToOne
    @JoinColumn(name = "location_id")
    private Location location;

    @OneToMany(mappedBy = "exhibition", cascade = CascadeType.ALL)
    private List<ExhibitionEntry> entries = new ArrayList<>();
}
```

Клас Location описує локацію, в якій проводяться виставки. Містить основні атрибути — назву, адресу, місткість, а також зв’язок із виставками, які проходили у цій локації.

java-код для класу Location:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Location {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String address;
    private int capacity;

    @OneToMany(mappedBy = "location", cascade = CascadeType.ALL)
    private List<Exhibition> exhibitions = new ArrayList<>();
}
```

Клас ExhibitionEntry є проміжною сутністю, що зв’язує експонат із виставкою та фіксує участь експоната у певній події. Додатково зберігає інформацію про автора та розміщення експоната на виставці.

java-код для класу ExhibitionEntry:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class ExhibitionEntry {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "exhibit_id")
    private Exhibit exhibit;

    @ManyToOne
    @JoinColumn(name = "exhibition_id")
    private Exhibition exhibition;
```

```
    @ManyToOne
    @JoinColumn(name = "artist_id")
    private Artist artist;

    private String placement;
}
```

Також для кожної з моделей було створено DTO класи. Нижче будуть наведені приклади DTO класів:

```
public class ArtistDTO {
    private Long id;
    private String name;
    private String bio;
    private String country;
}
public class ExhibitDTO {
    private Long id;
    private String title;
    private String description;
    private String type;
    private Integer yearCreated;
}
public class ExhibitionDTO {
    private Long id;
    private String title;
    private LocalDate startDate;
    private LocalDate endDate;
}
public class LocationDTO {
    private Long id;
    private String name;
    private String address;
    private int capacity;
```

```
}
public class ExhibitionEntryDTO {
    private Long id;
    private Long artistId;
    private Long exhibitId;
    private Long exhibitionId;
    private String placement;
}
```

3.2 Репозиторії

Для взаємодії з базою даних у системі організації виставок використовується набір репозиторіїв, реалізованих на основі інтерфейсу JpaRepository фреймворку Spring Data JPA. Репозиторії забезпечують зручний доступ до даних, дозволяють виконувати стандартні CRUD-операції (створення, читання, оновлення, видалення), а також містять методи для виконання специфічних запитів, необхідних для бізнес-логіки застосунку.

Основні репозиторії проєкту:

ArtistRepository

Інтерфейс ArtistRepository відповідає за збереження, пошук, оновлення та видалення даних про митців. Додатково реалізовано метод для пошуку митців за країною походження.

```
public interface ArtistRepository extends JpaRepository<Artist,
Long> {
    List<Artist> findByCountry(String country);
}
```

findByCountry(String country) — повертає список митців, які представляють задану країну.

ExhibitRepository

Інтерфейс ExhibitRepository забезпечує доступ до даних про експонати. Окрім базових CRUD-операцій, містить методи для пошуку експонатів за роком створення та для отримання експонатів, які не беруть участі у жодній виставці.

```
public interface ExhibitRepository extends JpaRepository<Exhibit,
Long> {
    List<Exhibit> findByYearCreated(Integer yearCreated);
    List<Exhibit> findByEntriesIsEmpty();
}
```

findByYearCreated(Integer yearCreated) — повертає всі експонати, створені у вказаному році.

findByEntriesIsEmpty() — повертає експонати, які наразі не призначені до жодної виставки.

ExhibitionRepository

Інтерфейс ExhibitionRepository реалізує роботу з виставками. Додатково містить метод для отримання поточних виставок, які тривають на даний момент.

```
public interface ExhibitionRepository extends
JpaRepository<Exhibition, Long> {
    List<Exhibition>
findByStartDateBeforeAndEndDateAfter(LocalDate now1, LocalDate
now2);
}
```

findByStartDateBeforeAndEndDateAfter(LocalDate now1, LocalDate now2) — повертає список виставок, які є активними на поточну дату.

LocationRepository

Інтерфейс LocationRepository забезпечує доступ до даних про локації проведення виставок. Містить метод для отримання локацій із найбільшою кількістю проведених виставок.

```
public interface LocationRepository extends
JpaRepository<Location, Long> {

    @Query("SELECT l FROM Location l LEFT JOIN l.exhibitions e
GROUP BY l.id ORDER BY COUNT(e) DESC")
    List<Location> findTopLocationsByExhibitionCount();
}
```

findTopLocationsByExhibitionCount() — повертає список локацій, відсортований за кількістю проведених у них виставок.

ExhibitionEntryRepository

Інтерфейс ExhibitionEntryRepository відповідає за роботу із записами участі експонатів у виставках. Дозволяє отримувати всі записи для конкретної виставки або для певного митця.

```
public interface ExhibitionEntryRepository extends
JpaRepository<ExhibitionEntry, Long> {
    List<ExhibitionEntry> findByExhibitionId(Long exhibitionId);
    List<ExhibitionEntry> findByArtistId(Long artistId);
}
```

findByExhibitionId(Long exhibitionId) — повертає всі записи експонатів, що беруть участь у заданій виставці.

findByArtistId(Long artistId) — повертає всі записи участі експонатів певного митця у виставках.

UserRepository

Інтерфейс UserRepository використовується для роботи з користувачами системи (наприклад, для автентифікації). Дозволяє знаходити користувача за іменем користувача (username).

```
    public interface UserRepository extends
JpaRepository<ExhibitionUser, Long> {

        Optional<ExhibitionUser> findByUsername(String username);

    }
```

findByUsername(String username) — повертає користувача за його логіном, якщо такий існує.

3.3 Сервіси

Сервісний шар у системі організації виставок реалізує бізнес-логіку застосунку, забезпечує перевірку даних, трансформацію між сутностями та DTO, а також координує взаємодію з репозиторіями. Кожен сервіс відповідає за окрему функціональність: управління митцями, експонатами, виставками, локаціями, записами участі у виставках та користувачами.

ArtistService

Сервіс ArtistService відповідає за виконання операцій над митцями. Він дозволяє отримувати список усіх митців, знаходити митця за ідентифікатором, створювати нових митців, оновлювати та видаляти їх, а також здійснювати пошук за країною. Для передачі даних використовується об’єкт ArtistDTO, який відокремлює внутрішню структуру сутності від зовнішнього представлення:

```
@Service
@RequiredArgsConstructor
public class ArtistService {

    private final ArtistRepository artistRepository;

    public List<ArtistDTO> getAllArtists() { ... }
    public ArtistDTO getArtistById(Long id) { ... }
    public ArtistDTO createArtist(ArtistDTO dto) { ... }
    public ArtistDTO updateArtist(Long id, ArtistDTO dto) { ... }
    public void deleteArtist(Long id) { ... }
    public List<ArtistDTO> getArtistsByCountry(String country) {
... }

    // Методи для перетворення між Entity та DTO
}
```

ExhibitService

Сервіс ExhibitService реалізує бізнес-логіку для роботи з експонатами. Дозволяє отримувати всі експонати, додавати нові, редагувати та видаляти їх, а також здійснювати пошук за роком створення та знаходити експонати, які не беруть участі у жодній виставці:

```
@Service
@RequiredArgsConstructor
public class ExhibitService {

    private final ExhibitRepository exhibitRepository;

    public List<ExhibitDTO> getAllExhibits() { ... }
    public ExhibitDTO getExhibitById(Long id) { ... }
    public ExhibitDTO createExhibit(ExhibitDTO dto) { ... }
```

```
    public ExhibitDTO updateExhibit(Long id, ExhibitDTO dto) { ...
}
    public void deleteExhibit(Long id) { ... }
    public List<ExhibitDTO> getExhibitsByYear(Integer year) { ...
}
    public List<ExhibitDTO> getExhibitsWithoutExhibition() { ... }

    // Методи для перетворення між Entity та DTO
}
```

ExhibitionService

Сервіс ExhibitionService відповідає за управління виставками. Дозволяє отримувати всі виставки, додавати нові, редагувати та видаляти їх, а також отримувати список поточних (активних) виставок. Для передачі даних використовується ExhibitionDTO:

```
@Service
@RequiredArgsConstructor
public class ExhibitionService {

    private final ExhibitionRepository exhibitionRepository;

    public List<ExhibitionDTO> getAllExhibitions() { ... }
    public ExhibitionDTO getExhibitionById(Long id) { ... }
    public ExhibitionDTO createExhibition(ExhibitionDTO dto) { ...
}
    public ExhibitionDTO updateExhibition(Long id, ExhibitionDTO
dto) { ... }
    public void deleteExhibition(Long id) { ... }
    public List<ExhibitionDTO> getCurrentExhibitions() { ... }

    // Методи для перетворення між Entity та DTO
}
```

ExhibitionEntryService

Сервіс ExhibitionEntryService реалізує логіку роботи із записами участі експонатів у виставках. Дозволяє створювати, оновлювати, видаляти записи, отримувати всі записи, а також здійснювати пошук за виставкою або митцем. Для передачі інформації використовується ExhibitionEntryDTO:

```
@Service
@RequiredArgsConstructor
```

```
public class ExhibitionEntryService {

    private final ExhibitionEntryRepository entryRepository;
    private final ExhibitRepository exhibitRepository;
    private final ExhibitionRepository exhibitionRepository;
    private final ArtistRepository artistRepository;

    public List<ExhibitionEntryDTO> getAllEntries() { ... }
    public ExhibitionEntryDTO getEntryById(Long id) { ... }
    public ExhibitionEntryDTO createEntry(ExhibitionEntryDTO dto)
{ ... }
    public ExhibitionEntryDTO updateEntry(Long id,
ExhibitionEntryDTO dto) { ... }
    public void deleteEntry(Long id) { ... }
    public List<ExhibitionEntryDTO> getEntriesByExhibition(Long
exhibitionId) { ... }
    public List<ExhibitionEntryDTO> getEntriesByArtist(Long
artistId) { ... }

    // Методи для перетворення між Entity та DTO
}
```

LocationService

Сервіс LocationService відповідає за управління локаціями проведення виставок. Дозволяє отримувати всі локації, додавати нові, редагувати та видаляти їх, а також формувати рейтинг локацій за кількістю проведених виставок:

```
@Service
@RequiredArgsConstructor
public class LocationService {

    private final LocationRepository locationRepository;

    public List<LocationDTO> getAllLocations() { ... }
    public LocationDTO getLocationById(Long id) { ... }
    public LocationDTO createLocation(LocationDTO dto) { ... }
    public LocationDTO updateLocation(Long id, LocationDTO dto) {
... }
    public void deleteLocation(Long id) { ... }
    public List<LocationDTO> getTopLocationsByExhibitionCount() {
... }

    // Методи для перетворення між Entity та DTO
```

```
}
```

AppService та AppDetailsService

Сервіси AppService та AppDetailsService реалізують функціонал реєстрації, автентифікації користувачів та інтеграцію з системою безпеки Spring Security. AppService відповідає за реєстрацію нових користувачів, кодування паролів та збереження ролей. AppDetailsService реалізує інтерфейс UserDetailsService і використовується для завантаження користувача за іменем під час автентифікації:

```
@Service
@RequiredArgsConstructor
public class AppService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public void registerUser(String username, String password,
String role) { ... }
}

@Component
public class AppDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException { ... }
}
```

CustomOAuth2Service

Сервіс CustomOAuth2Service реалізує інтеграцію з OAuth2-провайдерами для авторизації користувачів через зовнішні сервіси (наприклад, Google). Якщо користувача з таким email ще не існує, він створюється автоматично з роллю "USER":

```
@Service
public class CustomOAuth2Service implements
OAuth2UserService<OAuth2UserRequest, OAuth2User> {
```

```
    private final UserRepository userRepository;

    @Override
    public OAuth2User loadUser(OAuth2UserRequest request) { ... }
}
```

3.4 Контролери

Контролери у системі організації виставок реалізують шар взаємодії з користувачем, приймають HTTP-запити, викликають відповідні методи сервісного шару та повертають відповіді у форматі JSON або відображають сторінки інтерфейсу. Для REST-інтерфейсу використовуються анотації @RestController та @RequestMapping, для веб-інтерфейсу — @Controller. Контролери відповідають за маршрутизацію запитів, валідацію вхідних даних, обробку помилок та формування відповідей.

ArtistController

Контролер ArtistController забезпечує повний CRUD-функціонал для роботи з митцями. Дозволяє отримувати список усіх митців, знаходити митця за ідентифікатором, створювати, оновлювати та видаляти записи, а також здійснювати пошук митців за країною:

```
    @RestController
    @RequestMapping("/api/artists")
    @RequiredArgsConstructor
    public class ArtistController {

        private final ArtistService artistService;

        @GetMapping
        public ResponseEntity<List<ArtistDTO>> getAllArtists() {
... }

        @GetMapping("/{id}")
        public ResponseEntity<ArtistDTO> getArtist(@PathVariable
Long id) { ... }

        @PostMapping
```

```
        public ResponseEntity<ArtistDTO> createArtist(@RequestBody
ArtistDTO dto) { ... }


        @PutMapping("/{id}")
        public ResponseEntity<ArtistDTO>
updateArtist(@PathVariable Long id, @RequestBody ArtistDTO dto) {
... }


        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deleteArtist(@PathVariable
Long id) { ... }


        @GetMapping("/by-country/{country}")
        public ResponseEntity<List<ArtistDTO>>
getArtistsByCountry(@PathVariable String country) { ... }
    }
```

ExhibitController

Контролер ExhibitController реалізує REST-інтерфейс для управління експонатами. Дозволяє виконувати базові CRUD-операції, а також отримувати експонати за роком створення та експонати, які не беруть участі у жодній виставці:

```
    @RestController
    @RequestMapping("/api/exhibits")
    @RequiredArgsConstructor
    public class ExhibitController {


        private final ExhibitService exhibitService;


        @GetMapping
        public ResponseEntity<List<ExhibitDTO>> getAllExhibits() {
... }


        @GetMapping("/{id}")
        public ResponseEntity<ExhibitDTO> getExhibit(@PathVariable
Long id) { ... }


        @PostMapping
```

```
        public ResponseEntity<ExhibitDTO>
createExhibit(@RequestBody ExhibitDTO dto) { ... }


        @PutMapping("/{id}")
        public ResponseEntity<ExhibitDTO>
updateExhibit(@PathVariable Long id, @RequestBody ExhibitDTO dto)
{ ... }


        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deleteExhibit(@PathVariable
Long id) { ... }


        @GetMapping("/year/{year}")
        public ResponseEntity<List<ExhibitDTO>>
getByYear(@PathVariable Integer year) { ... }


        @GetMapping("/without-exhibition")
        public ResponseEntity<List<ExhibitDTO>>
getWithoutExhibition() { ... }
    }
```

ExhibitionController

Контролер ExhibitionController відповідає за управління виставками. Реалізує стандартні CRUD-операції, а також надає можливість отримати список поточних (активних) виставок:

```
    @RestController
    @RequestMapping("/api/exhibitions")
    @RequiredArgsConstructor
    public class ExhibitionController {

        private final ExhibitionService exhibitionService;

        @GetMapping
        public ResponseEntity<List<ExhibitionDTO>>
getAllExhibitions() { ... }

        @GetMapping("/{id}")
        public ResponseEntity<ExhibitionDTO>
getExhibition(@PathVariable Long id) { ... }

        @PostMapping
```

```
        public ResponseEntity<ExhibitionDTO>
createExhibition(@RequestBody ExhibitionDTO dto) { ... }

        @PutMapping("/{id}")
        public ResponseEntity<ExhibitionDTO>
updateExhibition(@PathVariable Long id, @RequestBody ExhibitionDTO
dto) { ... }

        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deleteExhibition(@PathVariable
Long id) { ... }

        @GetMapping("/current")
        public ResponseEntity<List<ExhibitionDTO>>
getCurrentExhibitions() { ... }
    }
```

ExhibitionEntryController

Контролер ExhibitionEntryController дозволяє керувати записами участі експонатів у виставках. Доступні операції створення, оновлення, видалення, отримання всіх записів, а також пошук записів за виставкою чи митцем:

```
    @RestController
    @RequestMapping("/api/entries")
    @RequiredArgsConstructor
    public class ExhibitionEntryController {

        private final ExhibitionEntryService entryService;

        @GetMapping
        public ResponseEntity<List<ExhibitionEntryDTO>>
getAllEntries() { ... }

        @GetMapping("/{id}")
        public ResponseEntity<ExhibitionEntryDTO>
getEntry(@PathVariable Long id) { ... }

        @PostMapping
        public ResponseEntity<ExhibitionEntryDTO>
createEntry(@RequestBody ExhibitionEntryDTO dto) { ... }
```

```
        @PutMapping("/{id}")
        public ResponseEntity<ExhibitionEntryDTO>
updateEntry(@PathVariable Long id, @RequestBody ExhibitionEntryDTO
dto) { ... }


        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deleteEntry(@PathVariable Long
id) { ... }


        @GetMapping("/by-exhibition/{exhibitionId}")
        public ResponseEntity<List<ExhibitionEntryDTO>>
getEntriesByExhibition(@PathVariable Long exhibitionId) { ... }


        @GetMapping("/by-artist/{artistId}")
        public ResponseEntity<List<ExhibitionEntryDTO>>
getEntriesByArtist(@PathVariable Long artistId) { ... }
    }
```

LocationController

Контролер LocationController реалізує REST-інтерфейс для управління локаціями проведення виставок. Дозволяє виконувати CRUD-операції, а також отримувати топ-локації за кількістю проведених виставок:

```
    @RestController
    @RequestMapping("/api/locations")
    @RequiredArgsConstructor
    public class LocationController {

        private final LocationService locationService;

        @GetMapping
        public ResponseEntity<List<LocationDTO>> getAllLocations()
{ ... }

        @GetMapping("/{id}")
        public ResponseEntity<LocationDTO>
getLocation(@PathVariable Long id) { ... }

        @PostMapping
        public ResponseEntity<LocationDTO>
createLocation(@RequestBody LocationDTO dto) { ... }

        @PutMapping("/{id}")
        public ResponseEntity<LocationDTO>
updateLocation(@PathVariable Long id, @RequestBody LocationDTO
dto) { ... }
```

```
        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deleteLocation(@PathVariable
Long id) { ... }

        @GetMapping("/top")
        public ResponseEntity<List<LocationDTO>>
getTopLocationsByExhibitionCount() { ... }
    }
```

JwtController

Контролер JwtController відповідає за REST-реєстрацію та автентифікацію користувачів із використанням JWT-токенів. Дозволяє реєструвати нових користувачів і отримувати токен при успішному вході:

```
    @RestController

    @RequestMapping("/api")

    public class JwtController {



        @PostMapping("/register")

        public ResponseEntity<?> register(@RequestBody AuthRequest
request) { ... }



        @PostMapping("/login")

        public ResponseEntity<?> login(@RequestBody AuthRequest
request) { ... }

    }
```

AuthController

Контролер AuthController реалізує веб-інтерфейс для реєстрації, входу та відображення сторінок для користувачів. Забезпечує інтеграцію з Spring Security, обробку форм реєстрації та входу, а також перенаправлення користувачів залежно від ролі:

```
    @Controller
    public class AuthController {
```

```
        @GetMapping("/register")
        public String registerForm(Model model) { ... }

        @PostMapping("/register")
        public String registerSubmit(@ModelAttribute("user")
ExhibitionUser user, @RequestParam String role, Model model) { ...
}

        @GetMapping("/login")
        public String loginForm() { ... }

        @PostMapping("/form-login-token")
        public String formLoginToken(@RequestParam String
username, @RequestParam String password, Model model) { ... }

        @GetMapping("/success")
        public String success(Authentication authentication) { ...
}
    }
```

AdminController та UserController

Контролери AdminController та UserController відображають відповідні сторінки для адміністратора та звичайного користувача після входу:

```
    @Controller
    public class AdminController {
        @GetMapping("/admin")
        public String admin() { return "admin"; }
    }

    @Controller
    public class UserController {
        @GetMapping("/user")
        public String user() { return "user"; }
    }
```

3.5 Обробка помилок і валідація

У системі організації виставок особлива увага приділяється коректній обробці помилок і валідації вхідних даних для забезпечення надійності, безпеки

та зручності користувачів. Валідація та обробка помилок реалізовані на кількох рівнях застосунку:

Валідація вхідних даних:

Валідація на рівні DTO

Для передачі даних між клієнтом і сервером використовуються об’єкти DTO (Data Transfer Object). Вони містять анотації валідації (наприклад, @NotNull, @Size, @Pattern), що дозволяє автоматично перевіряти коректність полів під час отримання HTTP-запитів. Це запобігає збереженню некоректних або неповних даних у базі.

Валідація в контролерах

Контролери приймають вхідні дані у вигляді DTO, які проходять валідацію за допомогою механізму Spring Validation. У разі виявлення помилок валідації контролер повертає клієнту відповідь із кодом помилки (наприклад, HTTP 400 Bad Request) та детальним описом проблем.

Перевірка у сервісному шарі

Додаткові бізнес-правила та логічні перевірки виконуються у сервісах. Наприклад, перевірка унікальності імені користувача під час реєстрації, коректності дат виставки (дата початку не повинна бути пізнішою за дату завершення), наявності пов’язаних сутностей перед оновленням чи видаленням.

Обробка помилок:

Глобальний обробник помилок

Для централізованої обробки виключень у застосунку реалізований глобальний обробник помилок (наприклад, через @ControllerAdvice), який перехоплює винятки, що виникають у контролерах і сервісах. Це дозволяє формувати уніфіковані відповіді з інформацією про помилки, логувати їх та уникати витоку внутрішньої інформації.

Обробка специфічних виключень:
В сервісах застосунку передбачена обробка типових помилок, таких як:

EntityNotFoundException — коли шуканий запис у базі не знайдений;

DataIntegrityViolationException — порушення цілісності даних (наприклад, спроба видалити митця, який має експонати);

UsernameNotFoundException — при спробі автентифікації неіснуючого користувача;

RuntimeException з повідомленнями про бізнес-логіку (наприклад, дублікати користувачів при реєстрації).

Повернення коректних HTTP-статусів:
Контролери повертають відповідні HTTP-коди залежно від результату обробки запиту:

200 OK — успішна операція;

201 Created — успішне створення ресурсу;

204 No Content — успішне видалення;

400 Bad Request — помилки валідації;

404 Not Found — ресурс не знайдено;

409 Conflict — конфлікти даних (наприклад, дублікати);

500 Internal Server Error — несподівані помилки серверу.

Безпека та захист від некоректних дій:

Аутентифікація та авторизація

Вхідні запити проходять перевірку прав доступу, що запобігає несанкціонованим діям (наприклад, редагуванню чи видаленню даних без відповідних ролей).

Обробка помилок аутентифікації

При невірному введенні логіна або пароля користувач отримує інформативне повідомлення без розкриття деталей безпеки.

4. Реєстрація та автентифікація

У системі організації виставок реалізовано комплексний підхід до забезпечення безпеки, який включає автентифікацію, авторизацію, генерацію та валідацію JWT-токенів, а також інтеграцію з OAuth2. Основні компоненти безпеки описані нижче.

Основні компоненти безпеки

— Модель користувача (ExhibitionUser) — містить поля username, password, role та реалізує інтерфейс UserDetails для інтеграції зі Spring Security.

— Репозиторій користувачів (AuthRequest) — забезпечує збереження та пошук користувачів у базі даних.

— Сервіси безпеки (AppDetailsService, AppService, CustomOAuth2Service) — відповідають за завантаження користувача, реєстрацію, кодування паролів, підтримку OAuth2.

— JWT-компоненти (AuthTokenGenerator, AuthTokenFilter) — генерують, перевіряють та обробляють JWT-токени для захисту REST API.

— CustomOAuth2Service — сервіс для інтеграції з OAuth2-провайдерами, автоматичне створення користувача при першому вході.

Реєстрація користувача

Реєстрація доступна як через REST API, так і через форму:

REST API:

```
POST /api/register
{
  "username": "user@example.com",
  "password": "password"
}
```

https://system-for-organizing-exhibitions.onrender.com/login

Якщо ім’я користувача вже зайняте — повертається помилка 400. Пароль зберігається у зашифрованому вигляді.

Форма:

```
GET /register — повертає сторінку реєстрації.
POST /register — обробляє дані форми, кодує пароль, зберігає користувача з вибраною роллю.
```

Вхід та автентифікація

REST API:

```
POST /api/login
{
  "username": "user@example.com",
  "password": "password"
}
```

У відповідь повертається JWT-токен, який потрібно використовувати для доступу до захищених ресурсів (у заголовку Authorization: Bearer ...).

Форма:

```
GET /login — повертає сторінку входу.
POST /form-login-token — автентифікує користувача, повертає токен для подальшої роботи.
```

OAuth2: підтримується вхід через Google та інші OAuth2-провайдери. Якщо користувач входить перший раз — створюється відповідний запис у базі.

Авторизація та захист ресурсів

Доступ до більшості маршрутів дозволений лише автентифікованим користувачам.

— Доступ до /admin — лише для користувачів з роллю ADMIN.

— Доступ до /user — лише для користувачів з роллю USER.

— Доступ до /api/register, /api/login, /register, /login, /oauth2/** — відкритий для всіх.

— Для REST API використовується JWT-фільтр, який перевіряє токен у кожному запиті.

Обробка помилок безпеки

— При невірному логіні або паролі повертається відповідне повідомлення про помилку.

— При спробі доступу до захищених ресурсів без токена або з недійсним токеном — статус 401 Unauthorized.

— При спробі доступу до маршруту без достатніх прав — статус 403 Forbidden.

5. Тестування програмного продукту

Для перевірки правильності роботи функціоналу обробки зображень у системі було проведено ручне тестування через Postman та автоматичне тестування за допомогою юніт-тестів. Тестування охоплювало: завантаження файлів, перегляд, видалення, перевірку формату файлу та його розміру, перевірку авторизації доступу.

Реєстрація нового користувача

Метод: POST

URL: http://localhost:8080/api/register

Очікування: Користувач реєструється в системі із вказаним логіном, паролем та роллю. Пароль зберігається в зашифрованому вигляді (наприклад, BCrypt), за замовчуванням призначається роль ROLE_USER, а також автоматично встановлюються прапори безпеки (акаунт активний, не заблокований тощо).

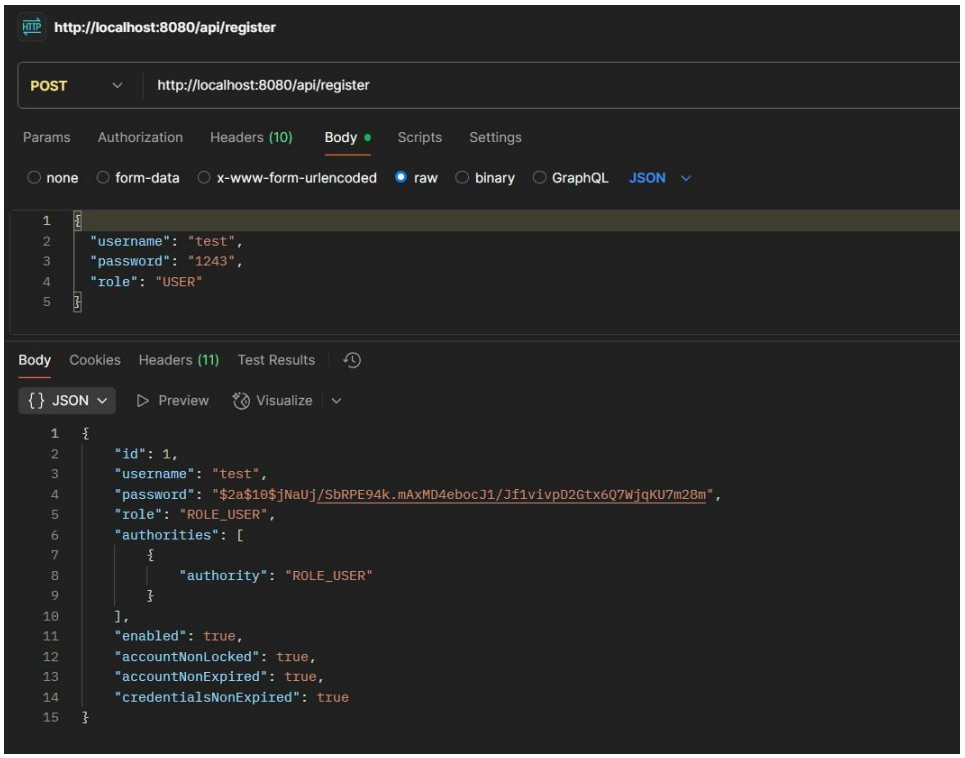

Рисунок 1 – Успішна реєстрація нового користувача через Postman

Авторизація користувача (JWT login)

Метод: POST

URL: http://localhost:8080/api/login

Очікування: У разі успішної авторизації користувача із вказаними логіном і паролем, у відповіді сервер повертає JWT-токен, який надалі використовується для виконання авторизованих запитів.

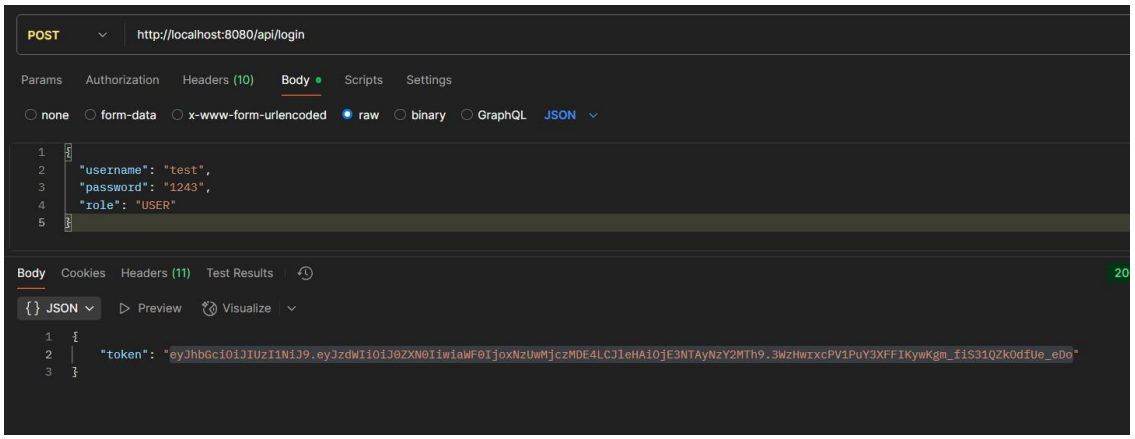

Рисунок 2 – Успішна авторизація та отримання JWT токена

Додати митця

Метод: POST

URL: http://localhost:8080/api/artists

Очікування: До бази даних додається новий художник із зазначенням імені, біографії та країни. У відповідь повертається створений об'єкт із присвоєним унікальним id.

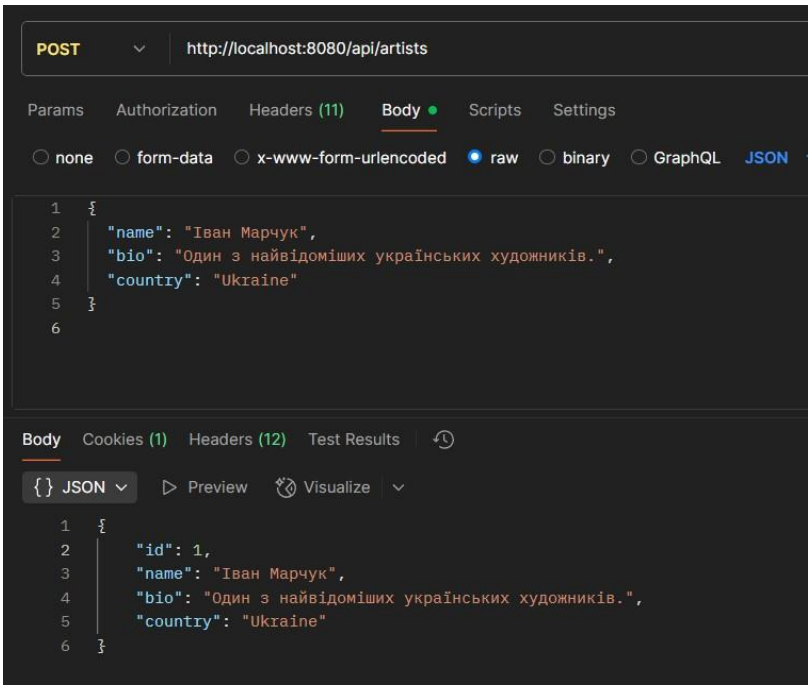

Рисунок 3 – Успішне створення запису художника

Отримати митців
Метод: GET

URL: http://localhost:8080/api/artists

Очікування: Після надсилання GET-запиту очікується, що сервер поверне список художників у форматі JSON. Кожен об’єкт містить такі поля: id, name, bio, country. Запит не потребує тіла або параметрів — лише виконується звернення до ендпоінту.

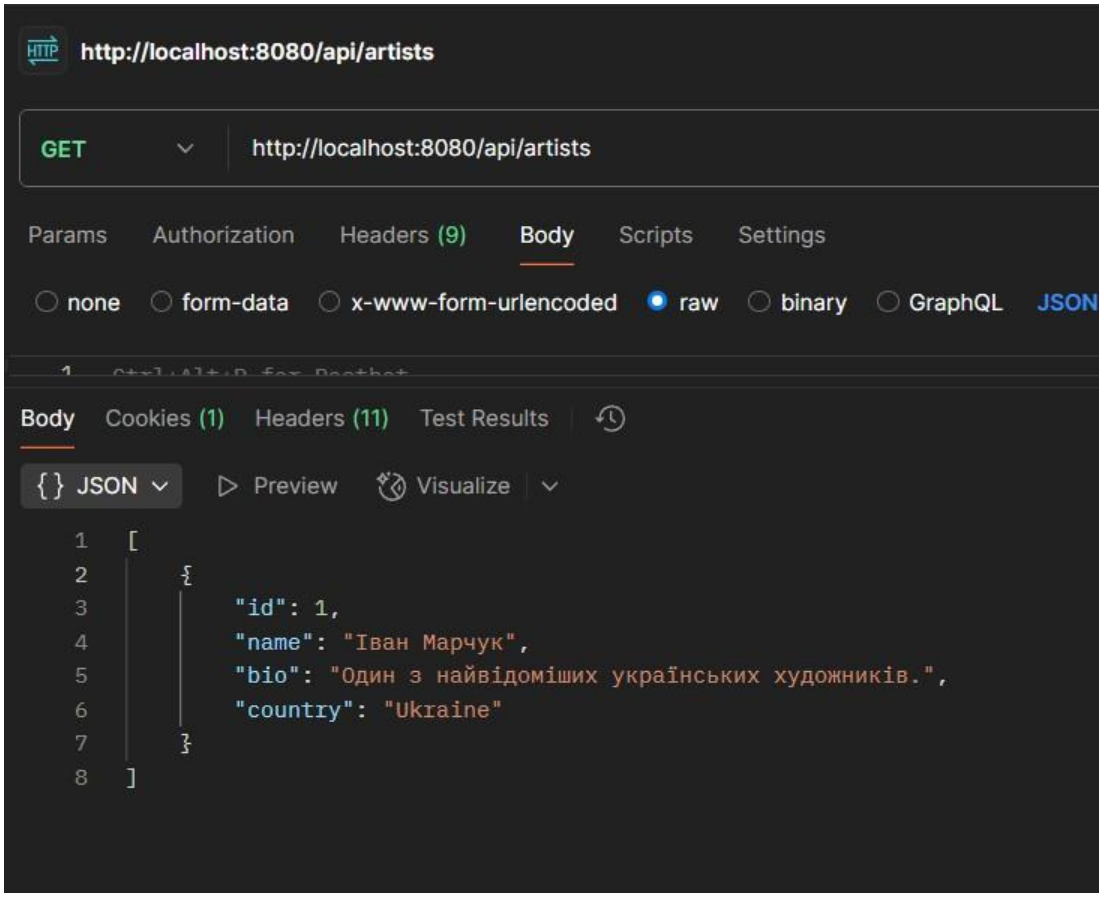

Рисунок 4 – Успішне отримання списку художників

Оновити дані митця

Метод: PUT

URL: http://localhost:8080/api/artists/1

Очікування: Оновлення даних художника з `id = 1`. У тілі запиту передаються нові значення полів `name` та `bio`. Після успішного виконання запиту очікується, що сервер поверне оновлений об’єкт художника з внесеними змінами.

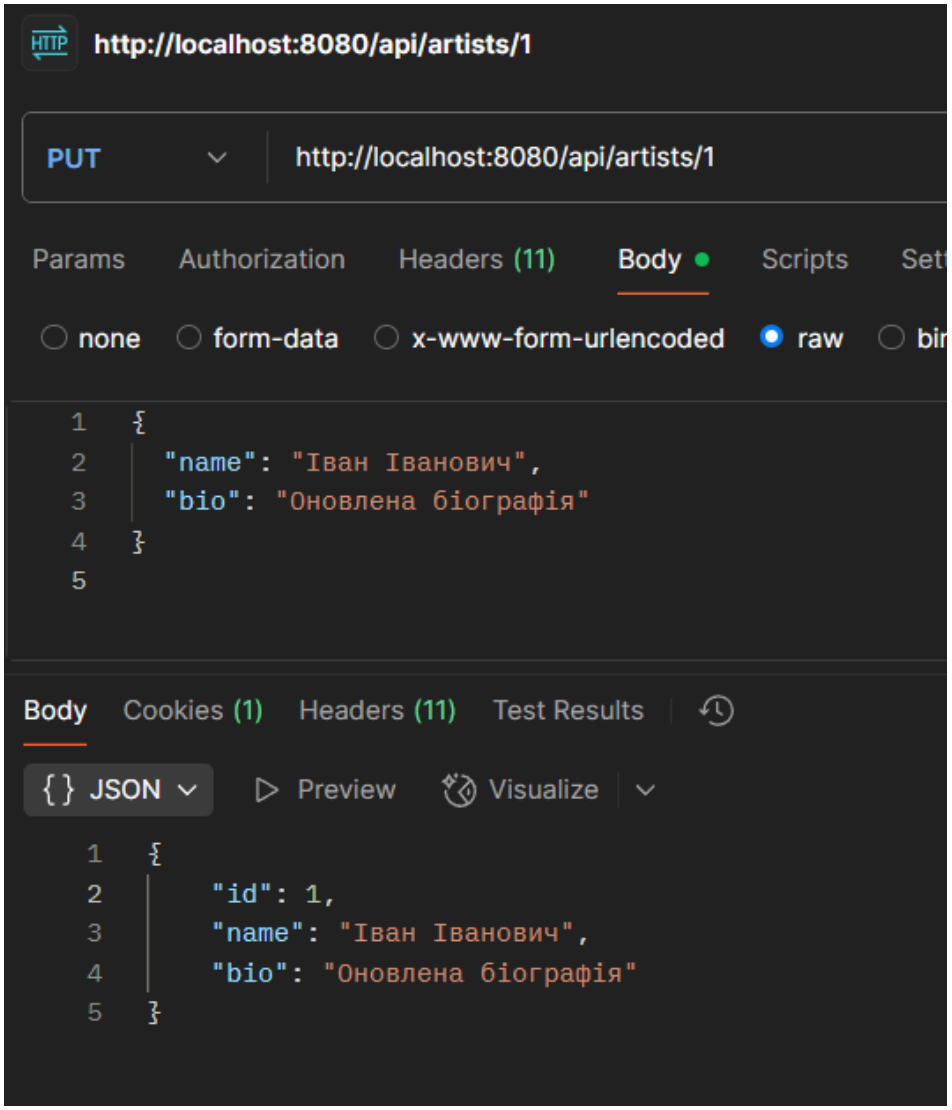

Рисунок 5 – Успішне оновлення даних художника

Видалити митця

Метод: DELETE

URL: http://localhost:8080/api/artists/1

Очікування: Видалення художника з бази даних за `id = 1`. Запит не потребує тіла — лише вказується ідентифікатор у URL. Після успішного виконання очікується, що сервер поверне підтвердження видалення або статус 200/204 без тіла.

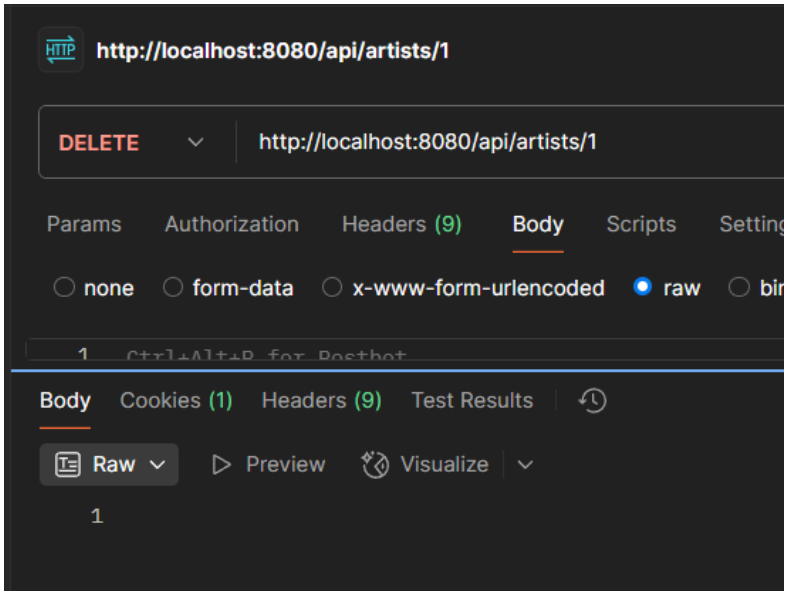

Рисунок 6 – Успішне видалення художника

Вивести список художників з певної країни
Метод: GET

URL: http://localhost:8080/api/artists/by-country/Ukraine

Очікування: У відповідь повертається список художників, що походять з вказаної країни (в даному випадку — Ukraine). Кожен художник містить ідентифікатор, ім’я, біографію та країну.

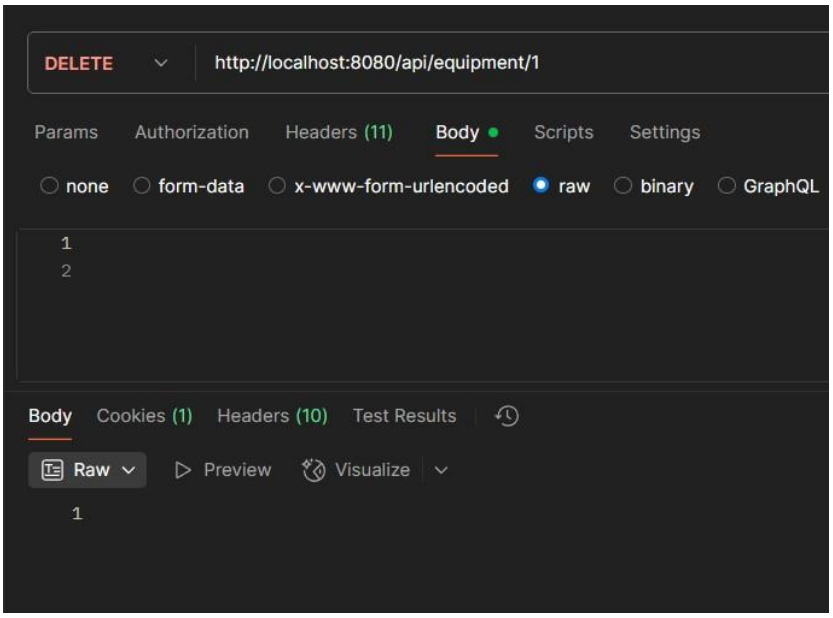

Рисунок 7 – Отримання списку художників з країни Ukraine

Додати експонат

Метод: POST

URL: http://localhost:8080/api/exhibits

Очікування: У базу даних додається новий експонат із вказаними атрибутами: назва, опис, тип, рік створення. У відповіді повертається створений об’єкт з автоматично згенерованим `id`.

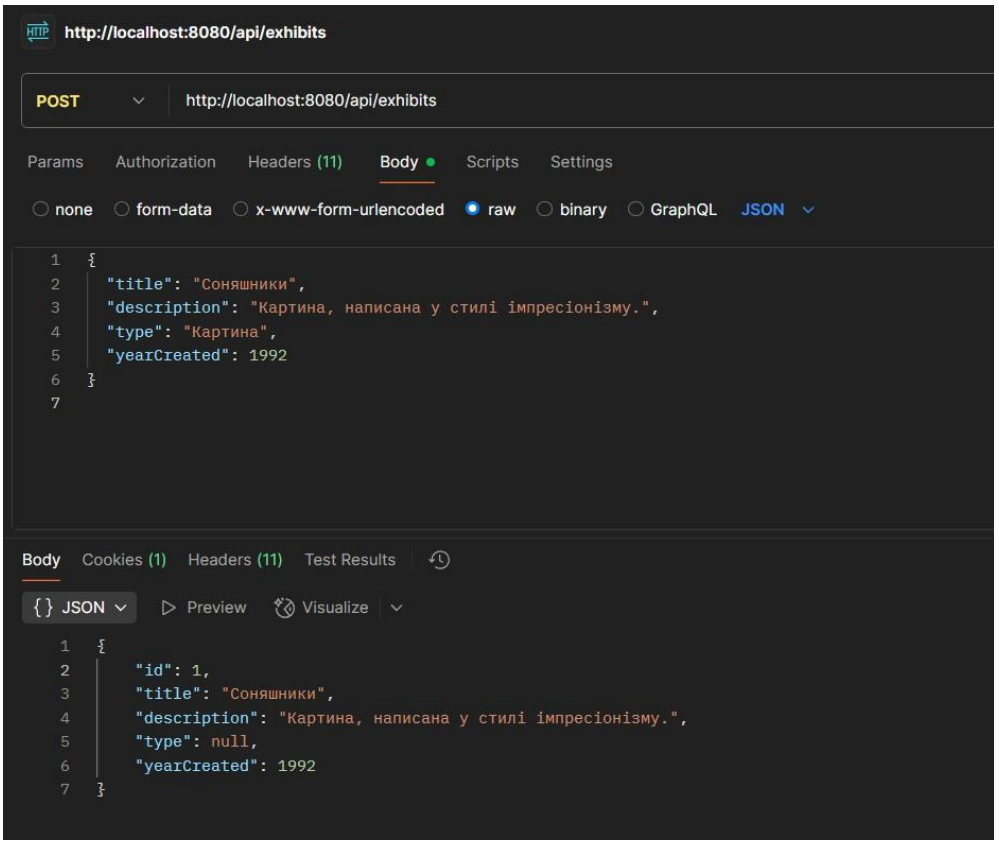

Рисунок 8 – Успішне створення експоната

Оновити експонат

Метод: PUT

URL: http://localhost:8080/api/exhibits/1

Очікування: Оновлення існуючого експоната з ідентифікатором `1`. Змінюється назва, опис, тип і рік створення. У відповідь надсилається оновлений об'єкт експоната у форматі JSON.

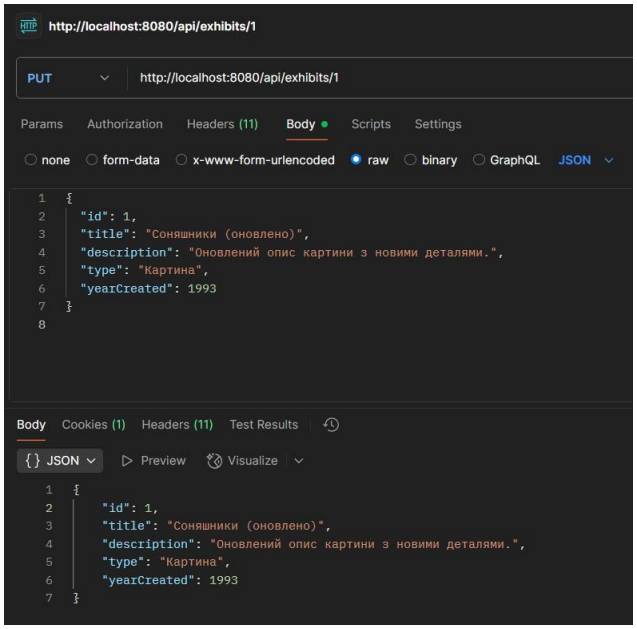

Рисунок 9 – Успішне оновлення експоната

Додати виставку

Метод: POST

URL: http://localhost:8080/api/exhibitions

Очікування: Створення нової виставки з вказаними назвою та датами. У відповідь надсилається об’єкт виставки з автоматично згенерованим id.

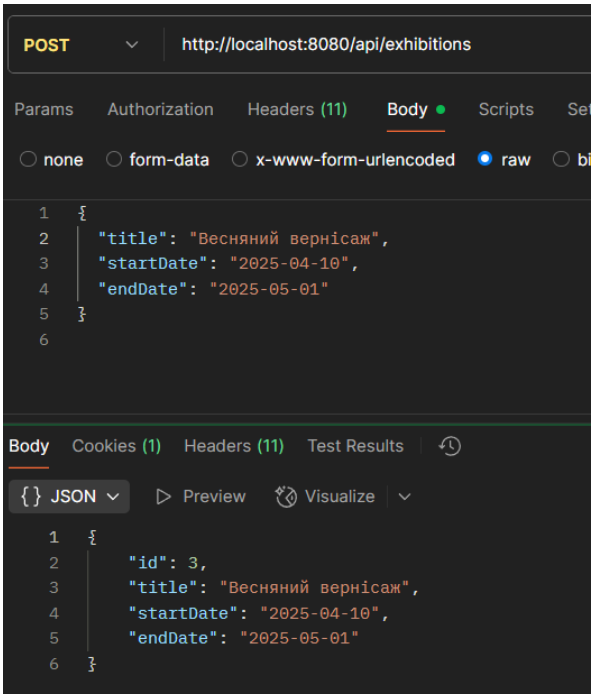

Рисунок 10 – Успішне створення виставки

Отримати всі виставки
Метод: GET

URL: http://localhost:8080/api/exhibitions

Очікування: У відповідь повертається список виставок у форматі JSON. Кожен об’єкт містить унікальний ідентифікатор, назву виставки, дату початку та дату завершення.

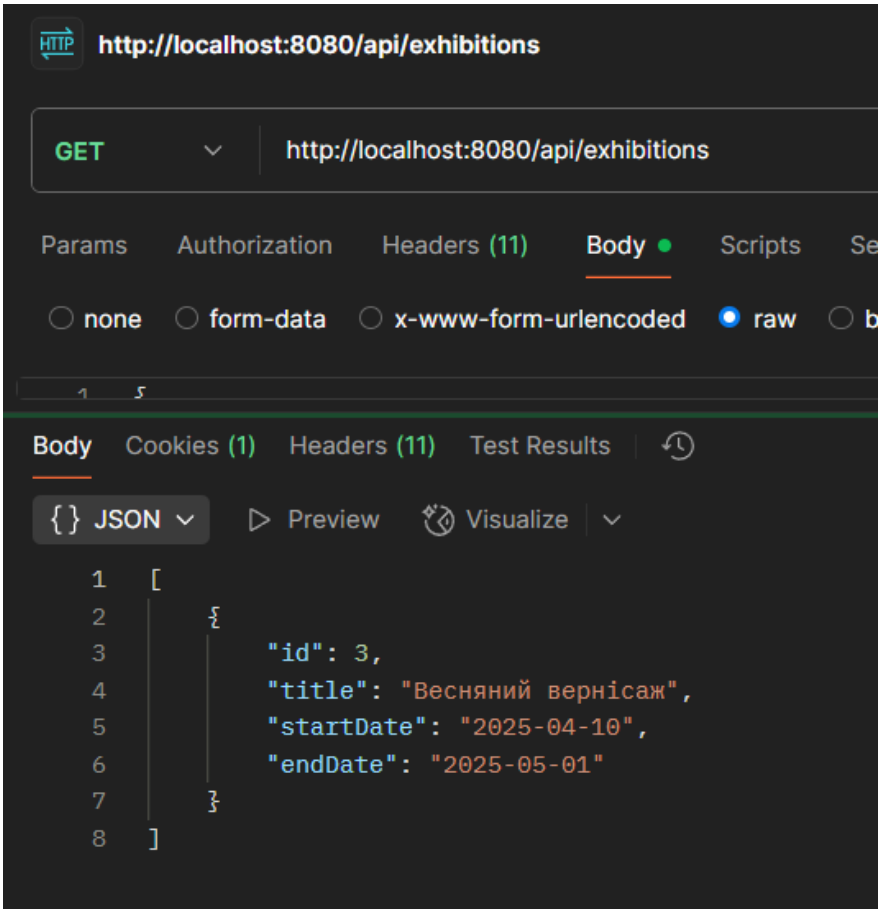

Рисунок 11 – Отримання списку виставок

Оновити виставку

Метод: PUT

URL: http://localhost:8080/api/exhibitions/3

Очікування: Виставку з ідентифікатором `id = 3` оновлено. Назва, дата початку та дата завершення змінюються згідно з переданими даними. У відповіді повертається оновлений об’єкт.

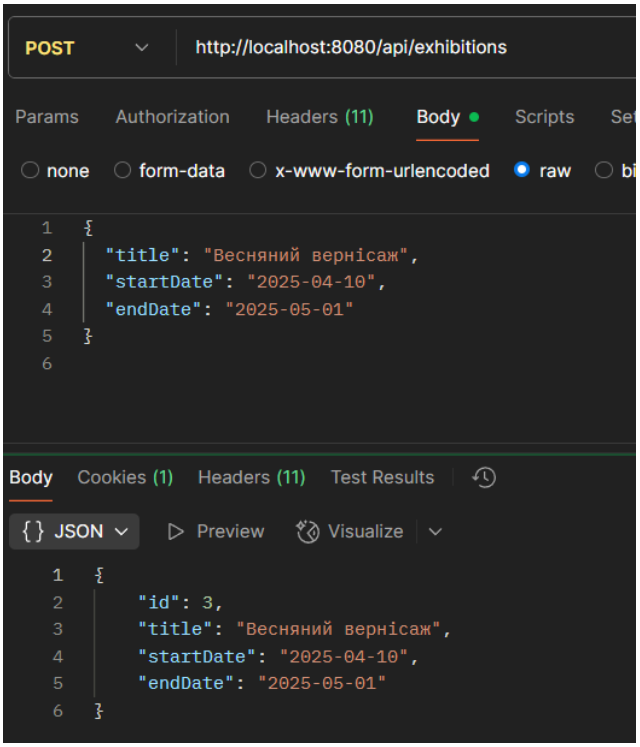

Рисунок 12 – Оновлення даних виставки

Видалити виставку

Метод: DELETE

URL: http://localhost:8080/api/exhibitions/3

Очікування: Виставка з ідентифікатором `id = 3` успішно видаляється з бази даних. У відповідь сервер повертає підтвердження видалення (наприклад, значення `1`).

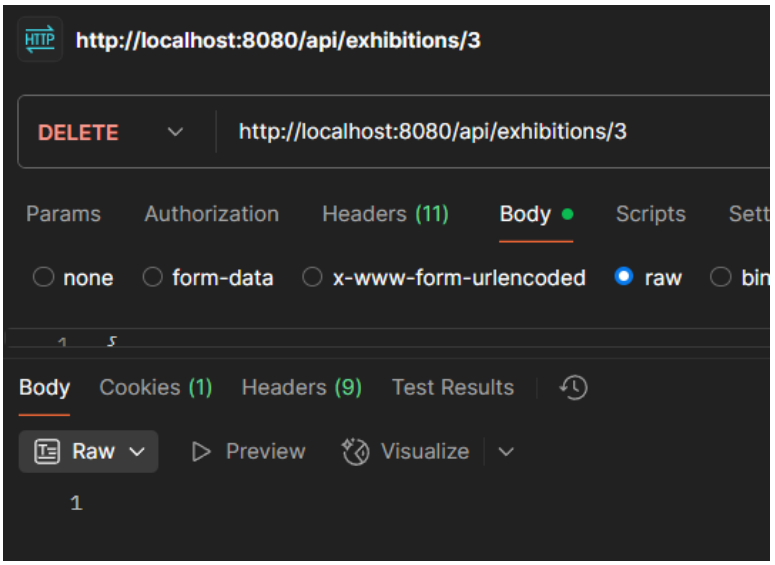

Рисунок 13 – Видалення виставки

Додати локацію

Метод: POST

URL: http://localhost:8080/api/locations

Очікування: Створюється нова локація з вказаною назвою та адресою. У відповіді повертається створений об'єкт з унікальним ідентифікатором, а також автоматично заповнене поле `capacity` зі значенням 0.

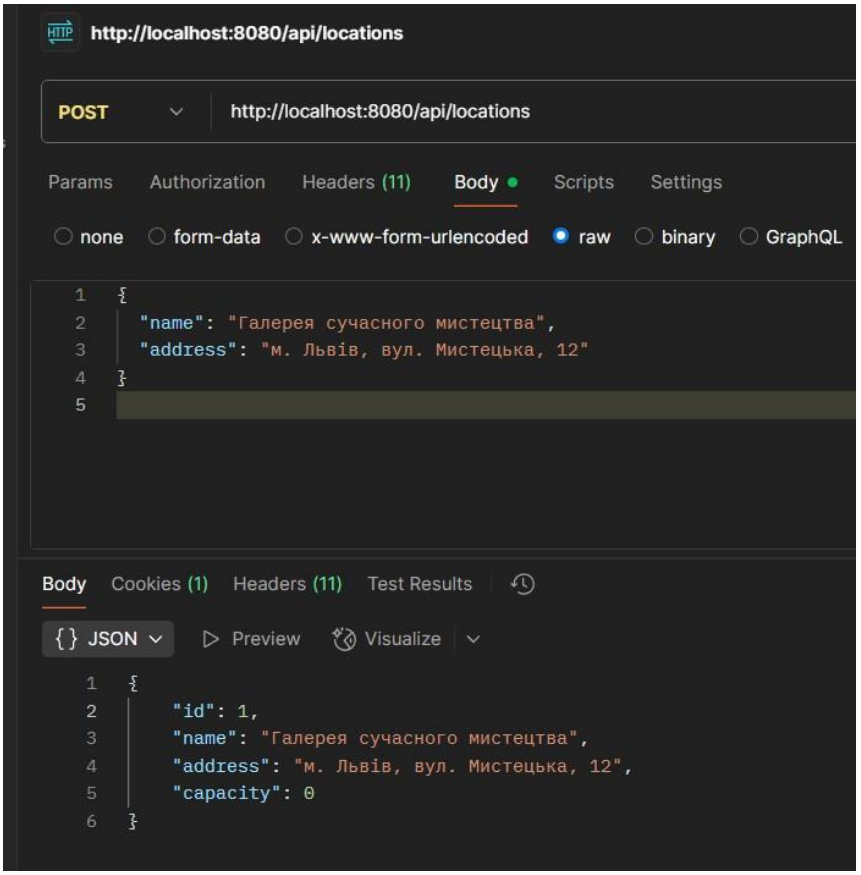

Рисунок 14 – Успішне створення нової локації

Отримати локації

Метод: GET

URL: http://localhost:8080/api/locations

Очікування: Отримується список усіх доступних локацій, що зберігаються в базі даних. У відповіді повертається масив об’єктів з детальною інформацією про кожну локацію: унікальний ідентифікатор, назва, адреса та місткість.

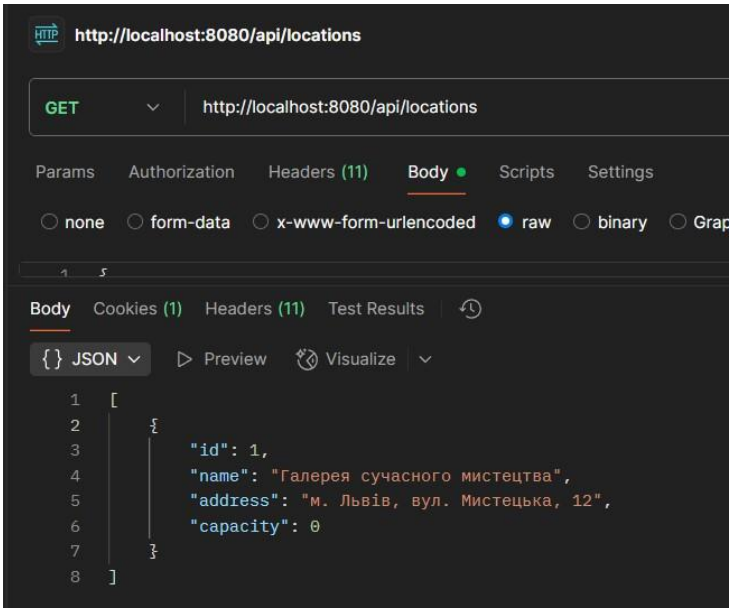

Рисунок 15 – Отримання списку локацій

Оновити локацію
Метод: PUT

URL: http://localhost:8080/api/locations/1

Очікування: Оновлюється інформація про локацію з ідентифікатором `1`. Змінюється назва та адреса локації. У відповіді повертається оновлений об’єкт з актуальними даними.

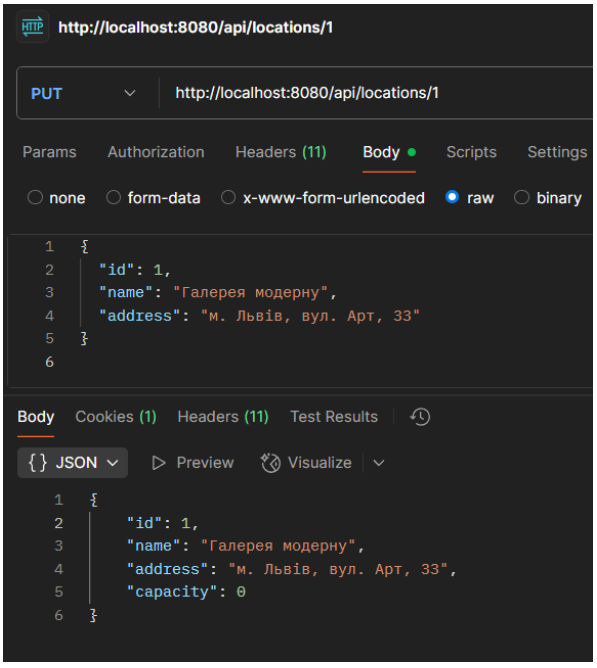

Рисунок 16 – Успішне оновлення даних локації

Призначити експонат на виставку
Метод: POST

URL: http://localhost:8080/api/entries

Очікування: Додається новий запис (entry) для виставки. Об’єкт містить ідентифікатори художника (artistId), експонату (exhibitId) та виставки (exhibitionId), а також місце розміщення (placement). У відповідь сервер повертає об’єкт з тими ж полями, доповнений автоматично згенерованим id.

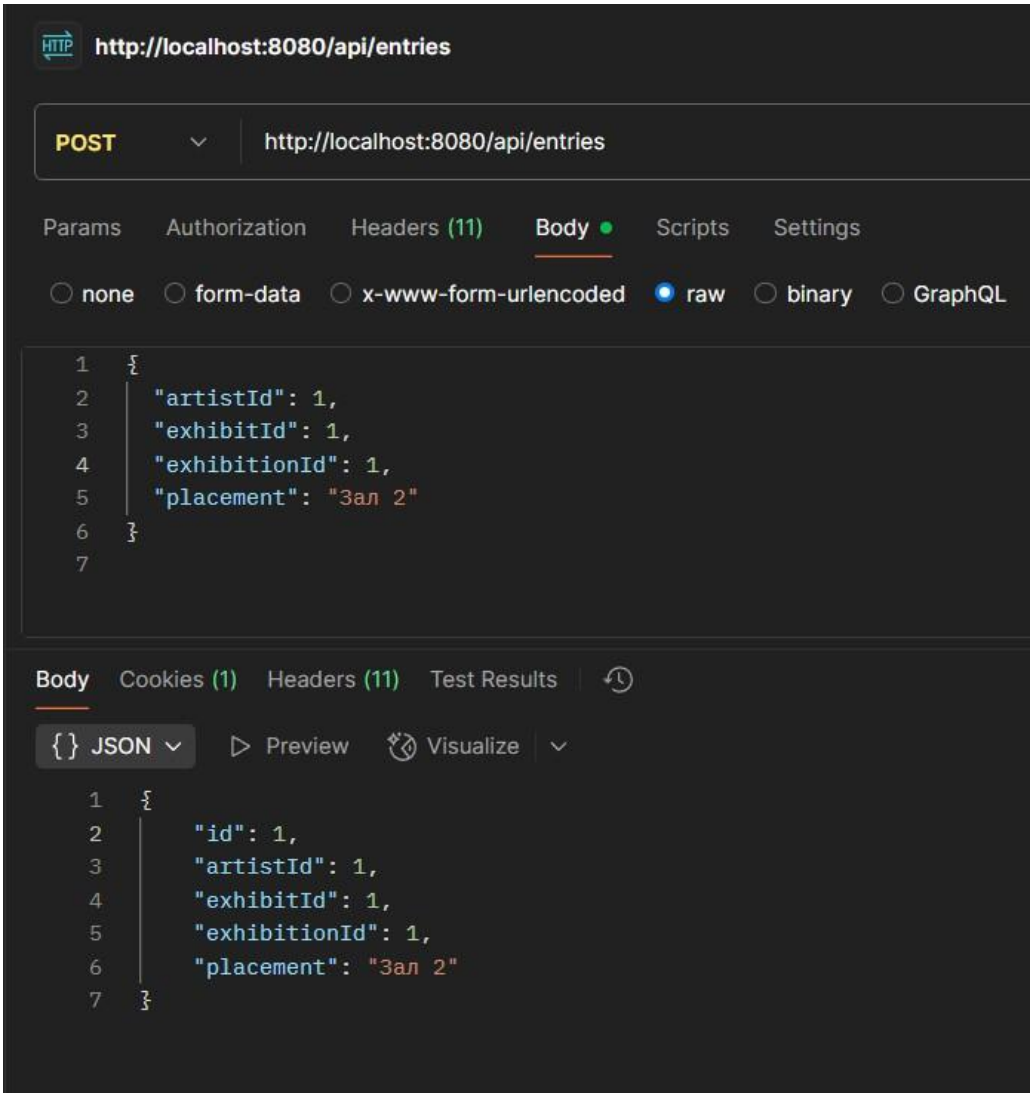

Рисунок 17 – Успішне додавання запису про участь експоната у виставці

Видалити експонат з виставки
Метод: DELETE

URL: http://localhost:8080/api/entries/5

Очікування: Видалення запису з ідентифікатором 5 з бази даних. Якщо запис існує — він успішно видаляється, а у відповідь може повертатись порожнє тіло або статус з підтвердженням.

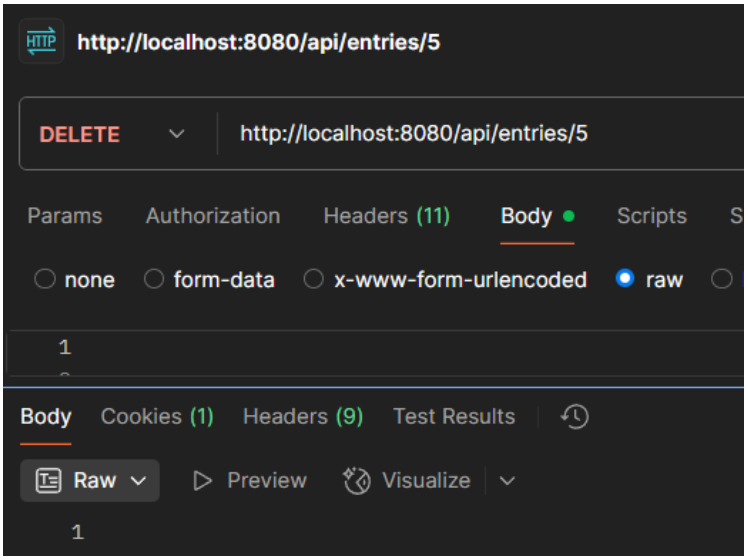

Рисунок 18 – Успішне видалення запису про участь експоната у виставці

Отримати експонати виставки
Метод: GET

URL: http://localhost:8080/api/entries/by-exhibition/1

Очікування: Отримання списку всіх записів (entries), що пов’язані з виставкою з ідентифікатором 1. У відповідь повертається масив JSON-об’єктів із детальною інформацією про кожен запис.

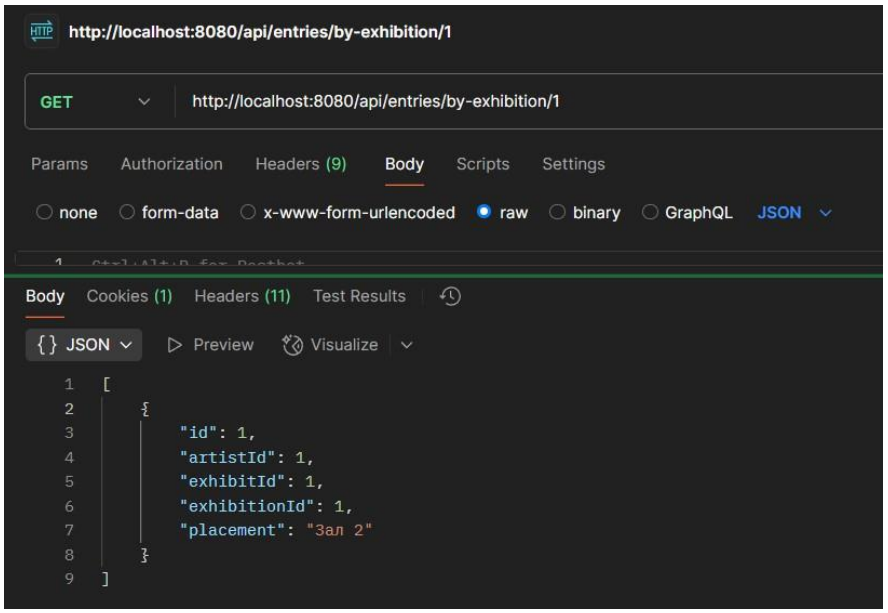

Рисунок 19 – Успішне отримання записів для виставки

Отримати виставки митця
Метод: GET

URL: http://localhost:8080/api/entries/by-artist/1

Очікування: Повертається перелік записів, пов’язаних із художником із `id = 1`. У відповіді міститься інформація про відповідність твору експозиції, виставці та розміщення в залі.

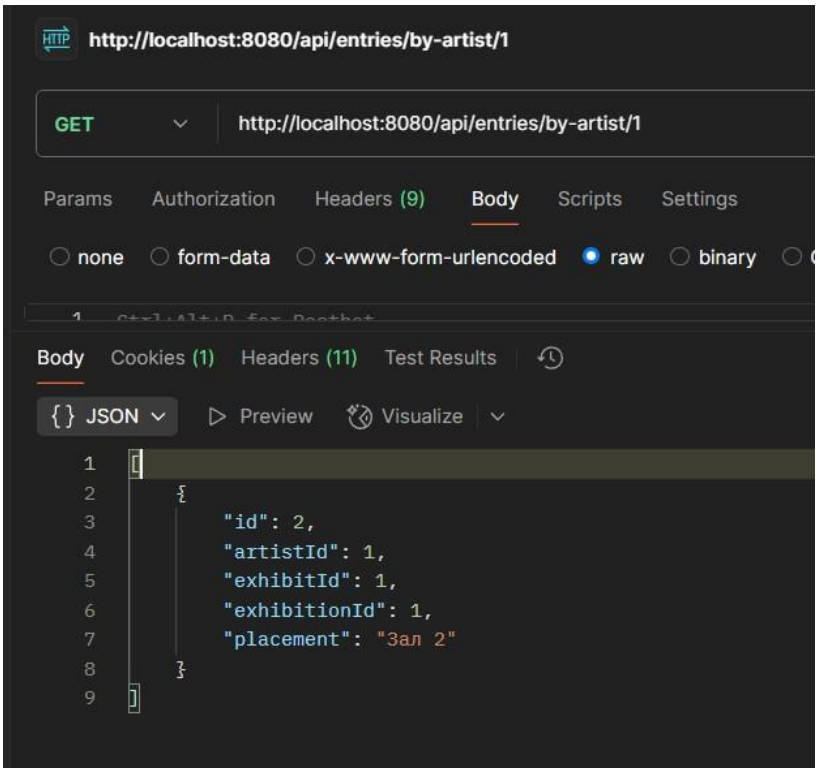

Рисунок 20 – Отримання запису твору за художником

Отримати експонати певного року
Метод: GET

URL: http://localhost:8080/api/exhibits/year/2010

Очікування: Повертається список експонатів, створених у 2010 році. Кожен об'єкт містить ідентифікатор, назву, опис, тип та рік створення.

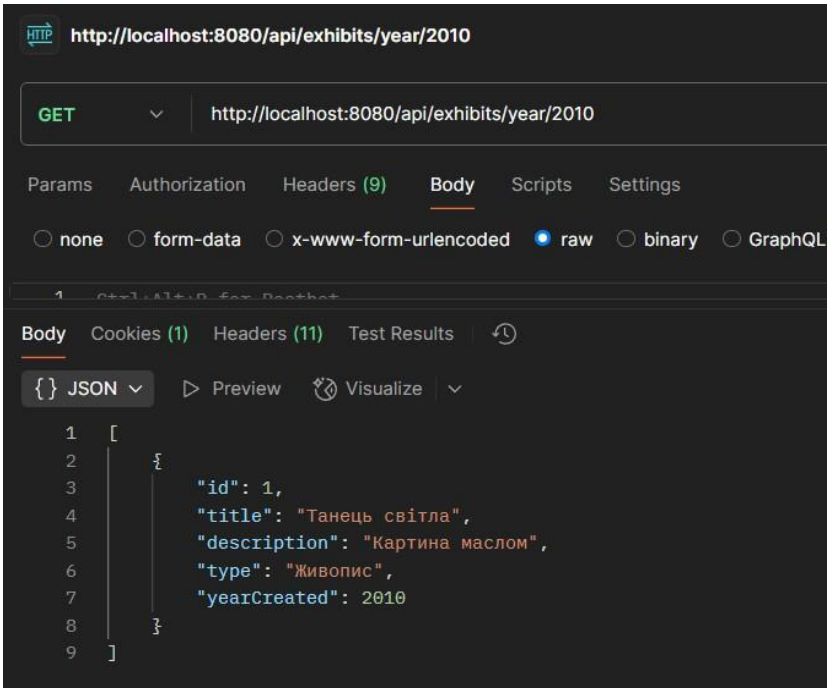

Рисунок 21 – Отримання експонатів за роком створення

Отримати митців за країною
Метод: GET

URL: http://localhost:8080/api/artists/by-country/Spain

Очікування: Повертається список художників, що походять із вказаної країни (в даному випадку — Іспанія).

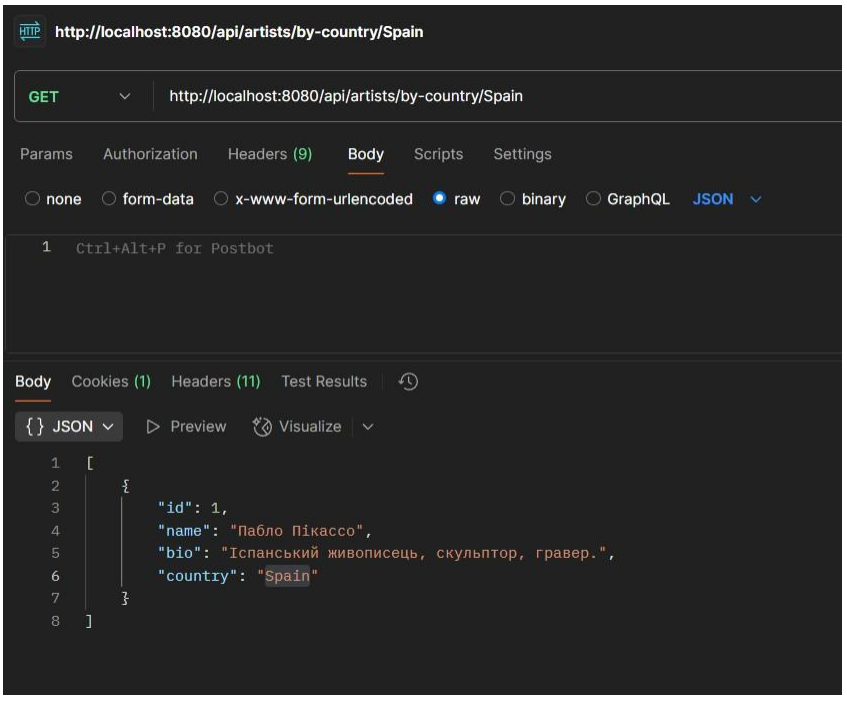

Рисунок 22 – Отримання художників за країною походження

Отримати локації з найбільшою кількістю виставок

Метод: GET

URL: http://localhost:8080/api/locations/top

Очікування: Отримання списку топових локацій з їх назвами, адресами та місткістю. Повертається масив об'єктів у форматі JSON.

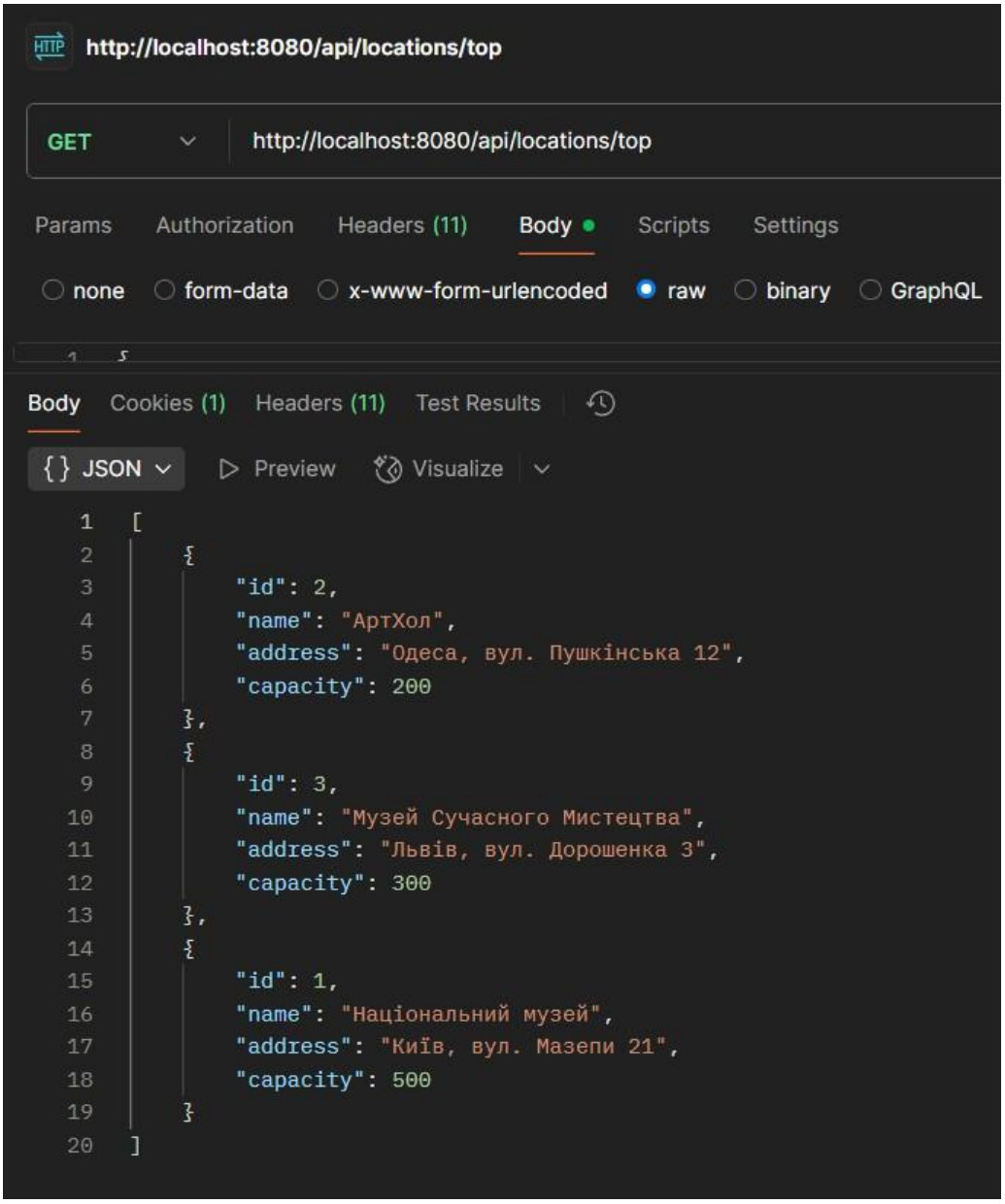

Рисунок 23 – Успішне отримання топових локацій

Отримати експонати без виставки

Метод: GET

URL: http://localhost:8080/api/exhibits/without-exhibition

Очікування: Отримання списку експонатів, які не закріплені за жодною виставкою. Повертається масив об'єктів у форматі JSON з інформацією про експонати: назва, опис, тип та рік створення.

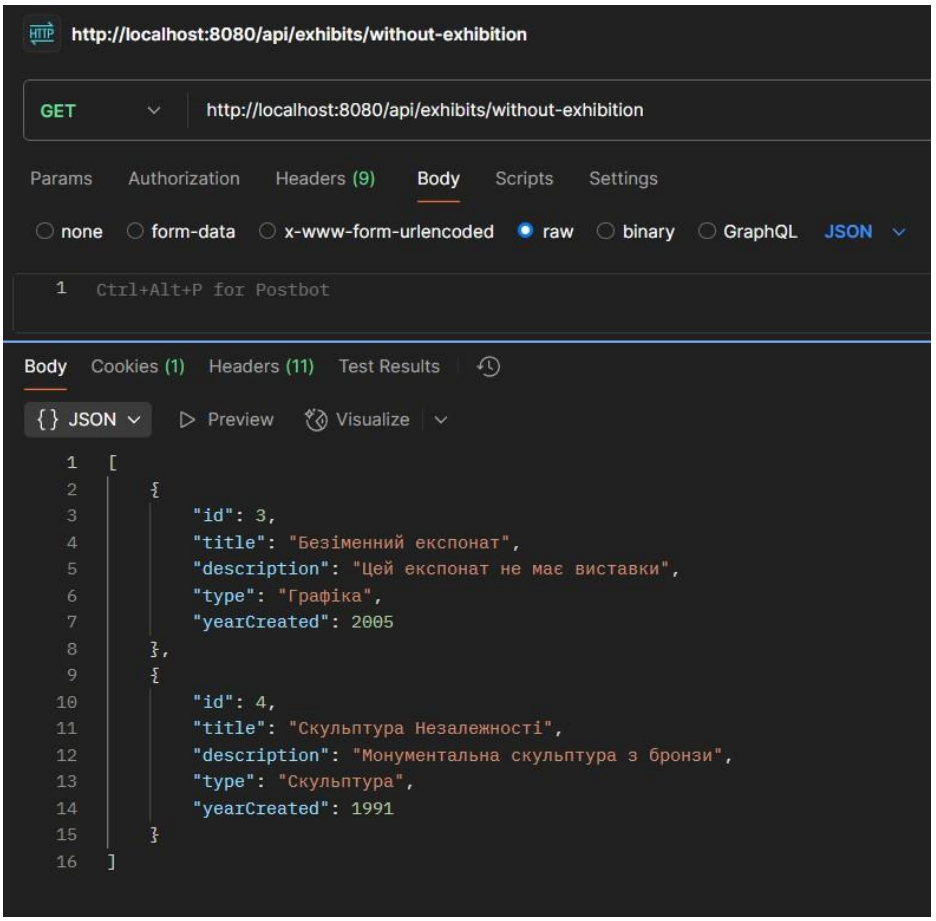

Рисунок 24 – Успішне отримання експонатів без виставки

Отримати поточні виставки

Метод: GET

URL: http://localhost:8080/api/exhibitions/current

Очікування: Отримання інформації про поточні виставки, що тривають на даний момент. Повертається масив об’єктів у форматі JSON з назвою виставки, датами початку та завершення.

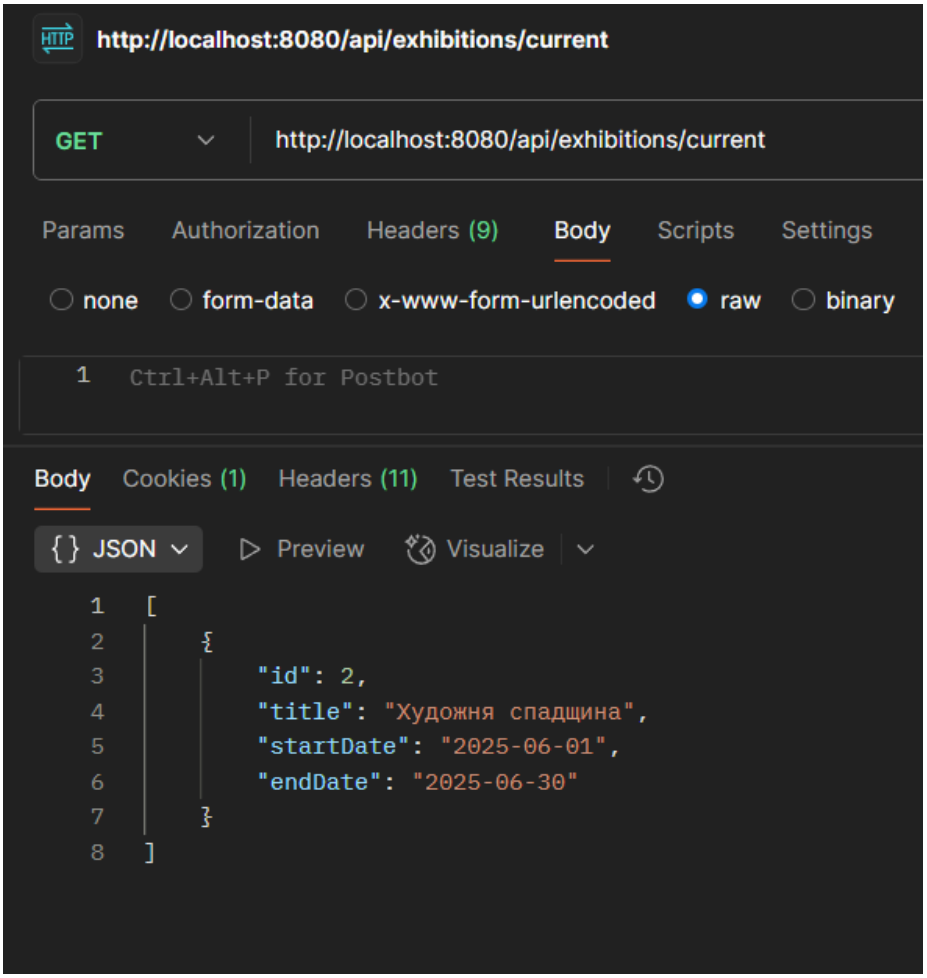

Рисунок 25 – Успішне отримання поточної виставки

Висновок

У результаті виконання курсової роботи було розроблено інформаційну систему для організації виставок, що автоматизує основні бізнес-процеси у сфері управління митцями, експонатами, виставками та локаціями проведення заходів. Реалізований застосунок дозволяє ефективно керувати даними про учасників, формувати експозиції, контролювати участь експонатів у виставках та аналізувати статистичні показники.

Система підтримує повний набір CRUD-операцій для основних сутностей, забезпечує пошук і фільтрацію за різними параметрами, ведення історії участі експонатів, а також інтеграцію з механізмами безпеки, включаючи автентифікацію, авторизацію та захист API за допомогою JWT. Для підвищення надійності застосовано валідацію вхідних даних і централізовану обробку помилок.

Архітектура системи побудована за трирівневим принципом із розділенням на контролери, сервіси і репозиторії, що реалізовано з використанням Spring Boot, Spring Data JPA та Spring Security. Застосовано DTO для безпечного обміну даними між клієнтом і сервером, а також інтегровано OAuth2 для авторизації через зовнішні провайдери.

Функціональність системи була протестована за допомогою інструментів для тестування REST API, що підтвердило стабільність, коректність та зручність використання розробленого програмного продукту.

Отже, поставлені завдання були успішно виконані: створено гнучку, масштабовану та безпечну систему, яка здатна підвищити ефективність організації виставок, покращити взаємодію між учасниками та організаторами, а також забезпечити прозорість і контроль усіх процесів. Отримані під час розробки знання та навички сприятимуть подальшому професійному розвитку у сфері розробки сучасних інформаційних систем.

Список використаних джерел

1. Spring Security Reference Documentation. OAuth 2.0 Resource Server JWT. URL: https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html (дата звернення: 19.06.2025)
2. Habr. JWT-аутентификация при помощи Spring Boot 3 и Spring Security 6. URL: https://habr.com/ru/articles/784508/ (дата звернення: 19.06.2025)
3. DevOps Blog. Spring Boot 3 + Spring Security 6 With JWT Authentication and Authorization [New 2024]. URL: https://blog.devops.dev/spring-boot-3-spring-security-6-with-jwt-authentication-and-authorization-new-2024-989323f29b84 (дата звернення: 19.06.2025)
4. Dzen. Java 1305. Как работает аутентификация и авторизация в Spring Security с использованием JWT токена? URL: https://dzen.ru/a/ZRp9AI9FaRPAV9cE (дата звернення: 19.06.2025)
5. YouTube – SPRING SECURITY 6 with JWT Authentication: Secure Your App in MINUTES! URL: https://www.youtube.com/watch?v=B1SUyu98HvQ (дата звернення: 19.06.2025)
6. YouTube – JWT-аутентификация для нативных приложений - Spring Security. URL: https://www.youtube.com/watch?v=iURuqbGeRNc (дата звернення: 19.06.2025)