Artwork Management System: Project Report

1. Introduction

The Artwork Management System is a backend project designed to manage digital artwork data efficiently and securely. The project consists of two major components:

1. **Backend API** implemented in **Java Spring Boot**
2. **Data ingestion and processing** implemented in **Python**

The system provides a secure RESTful API for CRUD operations on artwork data, ensuring proper access control and protection of sensitive data.

2. Choice of Technologies

2.1 Python for Data Ingestion

Python was chosen to connect to the Artwork API and upload data to MongoDB. Key reasons:

- **Ease of use and readability:** Python allows rapid development of scripts to retrieve and process JSON data.
- **Rich library ecosystem:** Libraries like `requests` (for HTTP requests) and `pymongo` (for MongoDB operations) make integration straightforward.
- **Cross-platform compatibility:** Scripts run on Windows, macOS, and Linux without modification.
- **Rapid prototyping:** Python allows quick iteration for data ingestion tasks.
- **JSON handling:** Python natively supports JSON operations, which is crucial for working with API data.

Using Python ensures data from the external API can be fetched, transformed, and stored efficiently in MongoDB.

2.2 Java Spring Boot for Backend

Spring Boot was chosen for the backend API due to:

- **Rapid application development:** Pre-configured templates and starter dependencies simplify API creation.
- **Robust ecosystem:** Seamless integration with Spring Data (MongoDB) and Spring Security reduces boilerplate.
- **Scalability and maintainability:** Strong typing and modular project structure improve maintainability.
- **Security support:** Built-in support for JWT authentication and role-based authorization.
- **Industry-standard:** Spring Boot is widely used in enterprise environments.

3. Backend Architecture

The backend follows a **layered architecture**:

1. **Controller Layer:** Handles HTTP requests and maps endpoints to service methods.
2. **Service Layer:** Contains business logic, such as creating, updating, and validating artworks.
3. **Repository Layer:** Performs CRUD operations on MongoDB via Spring Data.

Example:

```
public Artwork createArtwork(Artwork artwork) {
    if (artworkRepository.existsById(artwork.getId())) {
        throw new ResponseStatusException(HttpStatus.CONFLICT, "Artwork already exists"
    }
    return artworkRepository.save(artwork);
}
```

This separation of concerns improves maintainability and testability.

4. Security Implementation

4.1 JWT-Based Authentication

Security is implemented using **JSON Web Tokens (JWT)**:

1. Users register or log in via `/api/v1/auth/register` or `/api/v1/auth/login`.
2. Server generates a JWT containing username and roles (e.g., `ROLE_USER` or `ROLE_ADMIN`).
3. Client includes the JWT in the `Authorization` header for subsequent requests:

```
Authorization: Bearer <token>
```

4. Spring Security validates the token using a custom `JwtAuthenticationFilter` and sets the user's identity in the security context.

4.2 Security Configuration

The `SecurityConfig` class enforces endpoint access:

```
.authorizeHttpRequests(auth -> auth
    .requestMatchers("/api/v1/auth/**").permitAll()
    .requestMatchers(HttpMethod.GET, "/api/v1/artworks/**").hasAnyRole("USER","ADMIN")
    .requestMatchers(HttpMethod.POST, "/api/v1/artworks/**").hasRole("ADMIN")
    .requestMatchers(HttpMethod.PUT, "/api/v1/artworks/**").hasRole("ADMIN")
    .requestMatchers(HttpMethod.DELETE, "/api/v1/artworks/**").hasRole("ADMIN")
)
```

- **Public endpoints:** `register` and `login` are open to all users.
- **USER role:** Can perform GET requests.
- **ADMIN role:** Can perform GET, POST, PUT, DELETE requests.
- **Stateless sessions:** JWT ensures each request carries authentication info.

This ensures **role-based access control** and secure API endpoints.

5. MongoDB Integration

MongoDB stores artwork and user data. Benefits:

- **Flexible schema:** Artwork fields can vary without database migrations.
- **High performance:** Efficient read/write operations, ideal for batch ingestion via Python.
- **Easy integration:** Spring Data MongoDB maps Java models directly to collections.

Python scripts use `pymongo` to fetch and upload data from the Artwork API.

6. Advantages of This Setup

- Rapid backend development with Spring Boot
- Python ETL scripts for easy API data ingestion
- Secure stateless authentication with JWT
- Role-based access control for sensitive endpoints
- Flexible, high-performance MongoDB storage
- Maintainable project structure with clear separation of layers

7. Conclusion

This project demonstrates a robust, secure, and maintainable backend system for artwork management. Python was selected for **data ingestion** due to simplicity and strong library support, while Spring Boot provided a **reliable backend API**. JWT authentication ensures authorized access, and MongoDB provides flexible storage for artwork data.

The project exemplifies **modern software engineering practices**, combining efficient data processing, secure API design, and role-based access control.