**Software Architect Practical Test**

You must create a Python API following the best practices of Hexagonal Architecture, Clean Architecture, and Domain-Driven Design (DDD). The main goal is to evaluate your ability to structure a robust, maintainable application, clearly applying these concepts.

**Problem Context**

Create an API that allows users to provide their opinions and personal ratings for movies. The API will consolidate this user-provided data with additional information from the public OMDb API (The Open Movie Database - <http://www.omdbapi.com/>).

**Functional Requirements**

* Allow users to submit personal opinions and ratings for movies based on the IMDb identifier.
* Retrieve consolidated details for movies, enriching them with data from the OMDb API (such as genre, director, main actors, IMDb rating, and synopsis) along with **all user-provided opinions and ratings**.
* Perform searches on movies with filtering options by title and year.

**Example JSON Payloads**

**User Opinion Submission Example:**

{

"imdb\_id": "tt1375666",

"user\_opinion": "Excellent plot with stunning visuals and a thought-provoking story.",

"user\_rating": 9

}

**Consolidated Movie Details Response Example:**

{

"title": "Inception",

"year": 2010,

"imdb\_id": "tt1375666",

"genre": "Action, Adventure, Sci-Fi",

"director": "Christopher Nolan",

"actors": [

{ "name": "Leonardo DiCaprio" },

{ "name": "Joseph Gordon-Levitt" },

{ "name": "Elliot Page" }

],

"imdb\_rating": "8.8",

"plot": "A thief who steals corporate secrets through dream-sharing technology is tasked with planting an idea into a CEO's mind.",

"reviews": [

{

"user\_opinion": "Excellent plot with stunning visuals and a thought-provoking story.",

"user\_rating": 9

},

{

"user\_opinion": "A bit complex, but visually amazing!",

"user\_rating": 8

}

]

}

**Movie Search Request Example:**

{

"title": "Inception",

"year": 2010

}

**Movie Search Response Example:**

[

{

"title": "Inception",

"year": 2010,

"imdb\_id": "tt1375666",

"genre": "Action, Adventure, Sci-Fi",

"imdb\_rating": "8.8",

"reviews": [

{

"user\_opinion": "Excellent plot with stunning visuals and a thought-provoking story.",

"user\_rating": 9

},

{

"user\_opinion": "A bit complex, but visually amazing!",

"user\_rating": 8

}

]

}

]

**Technical Requirements**

* **Clean Architecture & Hexagonal Architecture:**
  + Clearly structure application layers: Application, Domain, Infrastructure, and Adapters.
  + Clearly define ports (interfaces) for communication between layers.
* **Domain-Driven Design (DDD):**
  + Model entities, value objects, aggregates, and domain services.
  + Implement a rich domain model with validations and business rules.
* **Integration Patterns:**
  + Use **Adapter Pattern** to convert external API data formats to the domain model (e.g., OMDbMovieProvider).
  + Implement an **Anti-Corruption Layer (ACL)** to isolate and transform data from the OMDb API into a format compatible with the domain, using a separate translator (e.g., convert\_omdb\_to\_movie function).
* **Design Patterns Required:**
  + Template Method
    - Suggested use: Define a standard algorithm for OMDb API integration, allowing subclasses to specialize specific steps like error handling, response parsing, and fallback mechanisms.
  + Factory Method
    - Suggested use: Instantiate different adapters or services responsible for obtaining external or internal data dynamically.
  + Strategy
    - Suggested use: Allow switching between different movie search strategies, such as simple searches by title or more complex searches using multiple criteria.
* **API Gateway:**
  + Use Nginx as an API Gateway to route requests to the application.

**Expected Deliverables**

* Complete source code available in a GitHub. The deploy process will be base on a Jenkins pipeline that will use your docker compose to deploy your solution inside one.
* Sufficient unit and/or integration tests to ensure functionality and facilitate code maintenance.

**Evaluation Criteria**

* Clarity and adequacy in applying Hexagonal Architecture and Clean Architecture principles.
* Correct and effective use of DDD concepts.
* Appropriate and resilient integration patterns implementation.
* Proper and clear implementation of required design patterns.
* Quality, organization, readability, and testability of the code.

**Technical Suggestions**

* Language: Python.
* Suggested Framework: FastAPI.
* Suggested local database: PostgreSQL or MongoDB.
* Use Docker with Docker Compose for local infrastructure.

**External API for Integration**

* **OMDb API** (<http://www.omdbapi.com/>) – Free, easy setup, limited free key available.

**Automation and Deployment**

* Deployment will be automated via Jenkins on our Virtual Machine (VM).
* Provide Dockerfile and docker-compose files completely and correctly configured for deployment.
* We will configure the Jenkins pipeline for automated deployment using the docker-compose you provide.

**Additional Automated Evaluation**

* Besides your provided tests, we will execute an additional suite of automated tests to validate implemented functionalities, stability, and integration with the external API.

**Endpoint:**

* The endpoint must be:
  + [post] <http://architecture-test/create-movie:90/>
  + [get] <http://architecture-test/search-movie:90/>

**These endpoints are MANDATORY using those description above.**