AS03: Refactoring & Application Deployment 📦

Design Patterns and Application Deployment

Introduction:

- This assignment builds upon the previous two and focuses on refactoring the `musicFinder` application, and deploying the final version using Docker.
- You will apply relevant design patterns to improve the maintainability and scalability of the application.
- The goal is to ensure the application follows modern software engineering principles while maintaining a fully functional CI/CD pipeline.

▼ Task 3.1: Refactoring with Design Patterns [25 marks]

Goal:

The objective of this task is to refactor specific parts of the `musicFinder` application using design patterns to improve code readability, maintainability, and scalability.

> *The skeleton codes are provided in the repository.*

a) Singleton with Dependency Injection [5 marks]

Scenario:
Implement a `Logger` class for the application, ensuring that only one instance exists throughout the app. Refactor the `Logger` to use Spring's **Dependency Injection (DI)** for cleaner code and better testability.

Instructions:

1. **Complete the Singleton Logger Class:**

    - Implement the `Logger` class using the Singleton pattern to track search requests and errors.
    - Use `private static` to ensure only one instance exists, but **don't call it manually.**
2. **Refactor Logger to Use Spring's DI:**
    - Use Spring's `@Component` annotation to register `Logger` as a bean.
    - `@Autowired` the `Logger` instance in the `MusicFinderController` to track search requests.

---

b) Abstract Factory for Search Providers [5 marks]

Scenario:
Complete the
Abstract Factory Pattern to handle different types of search providers (e.g., YouTube and Lyrics providers).

- You can refer to the existing API calls for each provider.
- They offer a different type of search, but they should follow a common interface.

Instructions:

1. **Complete the Search Provider Interface:**
    - Implement a common interface for search providers, e.g., `SearchProvider`.
    - Each provider (YouTube, Lyrics) will implement this interface.
2. **Implement the Concrete Factories:**
    - Create concrete classes like `YouTubeSearchProvider` and `LyricsSearchProvider`, implementing the interface methods.
    - Add logic to fetch the correct results from the APIs.
3. **Complete the Abstract Factory:**
    - Implement an abstract factory `SearchProviderFactory` that provides methods like `createProvider()`.
    - Create subclasses like `YouTubeSearchProviderFactory` and `LyricsSearchProviderFactory` to instantiate specific search providers.

---

c) Decorator with Caching [5 marks]

Scenario:
Implement the
Decorator Pattern to add caching functionality to the search results.

→ The first time a search is executed, the result should be fetched from the API, but subsequent requests should be served from the cache.

Instructions:

1. **Complete the Cache Decorator:**
    - Implement a `CacheDecorator` that wraps the search provider class.
    - Check if the search result exists in the cache before making a new API request.
2. **Implement the Caching Mechanism:**
    - Store the search results in a `Map` or any suitable caching solution (`CacheService)`.
    - When a search query is repeated, retrieve the result from the cache instead of hitting the API.
    - Additional Notes:

        > Use the `CacheService` to cache the search results, and to check if the search results are already cached

        > Use `"Cached Result:"` as a prefix for the cached results to differentiate them from the direct fetch of uncached search results

---

d) Strategy Pattern for Search Algorithm [10 marks]

Scenario:
The app should support multiple search algorithms. Implement the
Strategy Pattern to switch between different search algorithms (e.g., fuzzy search vs. exact search).

Instructions:

1. **Define the Search Strategy Interface:**

   - Create a `SearchStrategy` interface with a method `search()`, taking query parameters as input.

2. **Implement Different Strategies:**
   - Implement different strategies: `ExactSearchStrategy` and `FuzzySearchStrategy`.
   - `ExactSearchStrategy` will perform a straightforward match.
   - `FuzzySearchStrategy` will allow partial matches.
   - Additional notes:

     > The search strategy implementation can be abstract (i.e., you can simplify it to return different messages representative of "hypothetical" searches.

     > 🚫 overkill solution necessary !

3. **Bonus:**
   - Combine this with the caching decorator from the previous challenge, so that the search results are cached regardless of the strategy used.

💡

Submissions:

- Ensure the **refactored code** is committed to your GitHub repository.
- Ensure there are meaningful **commits** showing your refactoring process.

▼ Task 3.2: Application Deployment [5 marks]

Goal:

Finalise the CI/CD pipeline and deploy the fully refactored version of the `musicFinder` application. Ensure that the pipeline is capable of building, testing, and deploying the Dockerized version of the application.

Instructions:

1. **Add a CI/CD pipeline**:
    - Create a new `.github/workflows/ci.yml` file to include Docker build and deployment steps.
    - Ensure the pipeline:
        - **Builds** the application using Maven.
        - **Deploys** the application inside a Docker container.
    - Ensure that the application can be accessed locally via `http://localhost:8080`.

Tips:

- Test the pipeline manually before submitting to ensure everything runs smoothly.
- Ensure the Docker image is correctly configured to expose port 8080.
- **Helpful Links**:
    - GitHub Actions for Docker

💡

Submissions:

- Ensure your **GitHub repository** contains an updated `.github/workflows/ci.yml` file.
- The pipeline must be triggered automatically on every push.

Disclaimer:

This assignment will be evaluated using **GitHub Actions**, which will automatically run checks on your repository. Please ensure that your pipeline passes all required checks before the deadline.

- **Automated Testing —** Each push will trigger GitHub Actions to validate your work based on the CI/CD pipeline, refactored code, and Docker deployment.

- **Monitoring Progress —** Check the **Actions tab** in your repository to view the status of your submission.