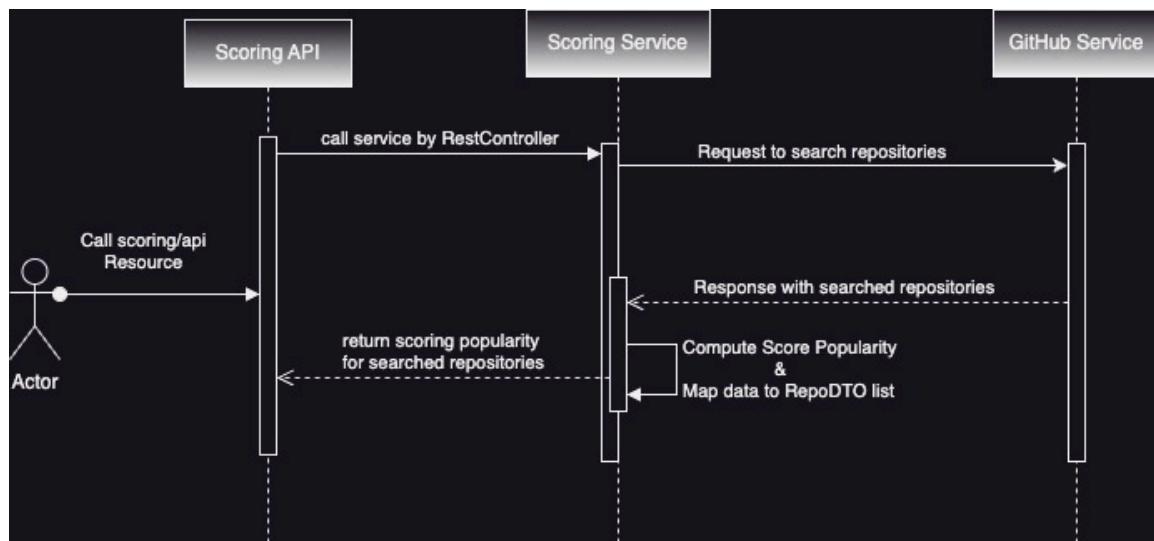


Project Report: Scoring Repository Application

1. Introduction

This document provides a comprehensive overview of the Repository Scoring Application. The application fetches repositories from GitHub based on user-defined parameters such as creation date and language, and assigns popularity scores based on stars and other metrics. Here is the sequence flow to present the overview of the user activity to access the API.

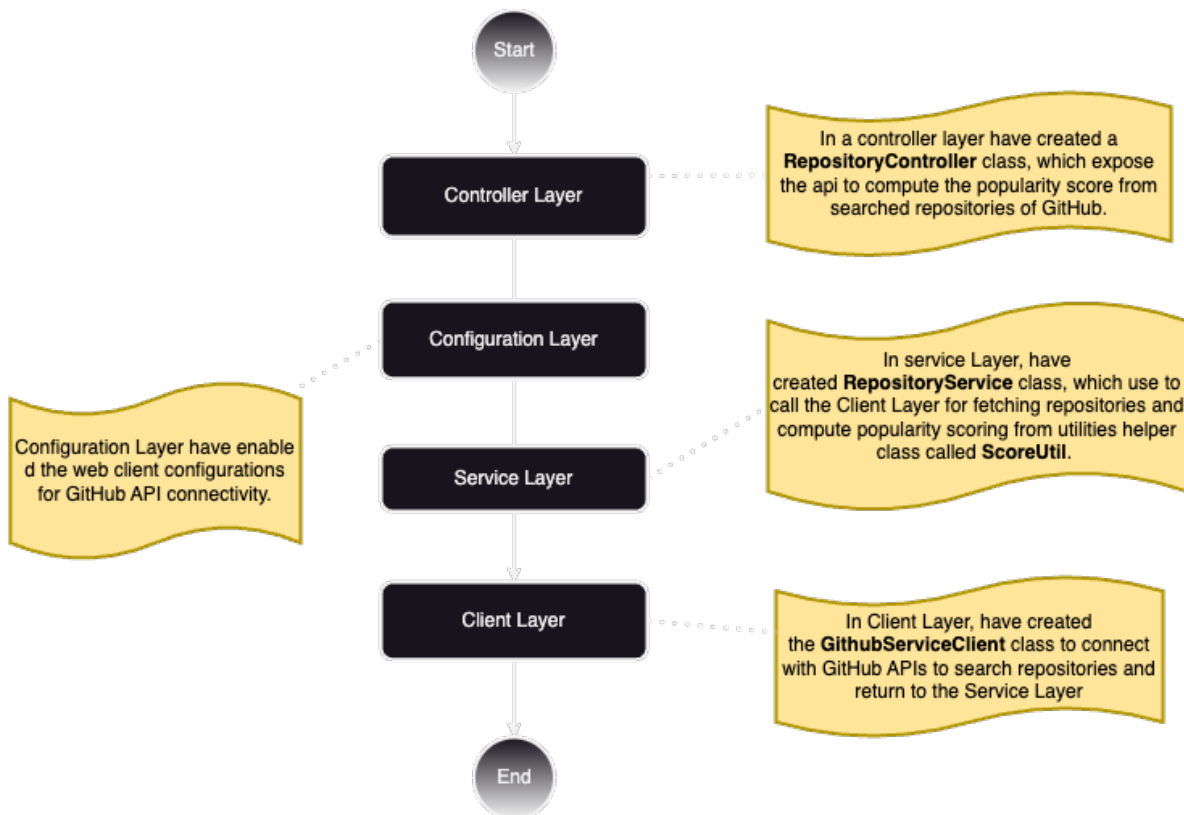


2. Architecture

The application follows a modular architecture with the following key components:

- **Controller Layer:** Handles API requests from users.
- **Service Layer:** Implements business logic and scoring calculations.
- **Client Layer:** Manages communication with the GitHub API.
- **Configuration Layer:** Configures external services such as the GitHub API.

Here is the diagram for the layer views.



The application uses Spring Boot for rapid development and dependency injection, and follows REST principles for API communication.

3. Design Patterns Used

The following design patterns are utilized in the application:

- **Builder Pattern:** Used in RestTemplateBuilder for constructing HTTP clients.
- **Singleton Pattern:** Ensures a single instance of the 'RestTemplate' is reused.
- **Factory Pattern:** Spring's @Bean mechanism acts as a factory for dependency injection.

4. Popularity of Scoring mechanism

The popularity score calculated on the basis of normalized stars, forks and Recency of update. And it use to assign the computed score to each repository. It use some internal calculation formulas as follows.

1. **StarsScore** = Stars / MaxStars
2. **ForksScore** = Forks / MaxForks
3. **RecencyScore** = $\log(1 + x)$ where $x = (\text{maxDays} - \text{daysSinceUpdate}) / \text{maxDays}$
4. **PopularityScore** = (W1 * StarsScore) + (W2 * ForksScore) + (W3 * RecencyScore)

Popularity Score computation by assign weights to each of these three factors. i.e. stars, fork and recency which is use to calculate individual popularity score of each repository. where the sum of the weights for stars, forks, and recency, W1, W2, and W3, is 1.

6. Test report

The Scoring Application followed the test driven development, therefore implemented unit and integration tests. Here is overview of tests included in the project.

Test Cases

[\[Summary\]](#) [\[Package List\]](#) [\[Test Cases\]](#)


RepositoryServiceTest

	testFetchAndScoreRepositories_Success	0,272 s
---	---------------------------------------	---------

GithubServiceClientTest

	testGetSearchRepositories_Success	0,483 s
	testGetSearchRepositories_OtherException	0,018 s
	testGetSearchRepositories_HttpClientErrorException	0,012 s

ApplicationExceptionHandlerTest

	handleGitHubApiException_ShouldReturnBadGateway	0,015 s
	handleRateLimitException_ShouldReturnTooManyRequests	0,002 s
	handleGenericException_ShouldReturnInternalServerError1	0,004 s

RepositoryControllerTest

	testGetRepositoriesDefaultParams	0,442 s
	testGetRepositoriesCustomParams	0,033 s

ScoringApplicationTests

	contextLoads	1,049 s
---	--------------	---------

RepositoryControllerIntegrationTest

	testGetRepositoriesHavingCustomParams	2,709 s
	testGetRepositoriesHavingNoParams	0,667 s

7. Created Operation Test via Postman

The application created the endpoint for computing popularity score of the searched repositories via connecting the GitHub API. Here is the manual tested response of the developed endpoint.

The screenshot shows a Postman interface with a GET request to `http://localhost:8080/v1/repositories/score?createdAfter=2000-01-02&language=Java&page=10&itemsPerPage=3`. The request is successful, returning a 200 OK status with a response time of 2.90 s and a body size of 579 B. The response body is displayed in JSON format, showing a list of three repositories with their popularity scores.

Key	Value	Description
<input checked="" type="checkbox"/> createdAfter	2000-01-02	
<input checked="" type="checkbox"/> language	Java	
<input checked="" type="checkbox"/> page	10	

```
1 [
2   {
3     "name": "netty",
4     "owner": "netty",
5     "stars": 33643,
6     "forks": 15988,
7     "lastUpdated": "2025-01-05T11:49:29Z",
8     "popularityScore": 0.7
9   },
10  {
11    "name": "AndroidUtilCode",
12    "owner": "Blankj",
13    "stars": 33389,
14    "forks": 10694,
15    "lastUpdated": "2025-01-05T15:53:45Z",
16    "popularityScore": 0.5976430525343255
17  },
18  {
19    "name": "spring-boot-demo",
20    "owner": "xkcoding",
21    "stars": 33280,
22    "forks": 10917,
23    "lastUpdated": "2025-01-05T14:56:09Z",
24    "popularityScore": 0.600531480298837
25  }
26 ]
```

8. Getting code from GitHub repository.

To get a local copy of the current code, clone it using git and run it as follows.

```
$ git clone https://github.com/eastechsystem/scoring-  
repositories.git  
$ cd scoring-repositories  
$ mvn clean spring-boot:run
```

9. Conclusion

The Scoring Application is designed for scalability, maintainability, and performance. It leverages Spring Boot's powerful ecosystem to provide a robust and modular backend solution.

10. Reference:

Reference for stated formulas in above section which was use to calculate score of stars, fork, and recency as a source.

1. [Decay-based ranking](#)
2. [Exponentially decaying -> Math to the rescue](#)
3. [Discussion about time decay for ranking content](#)
4. [Scoring Popularity in GitHub](#)
5. [GitHub Search repositories Rest API](#)