Below is a tailored project design for your specific use case: a **Dart-based tool** integrated with a **Spring Boot application** to handle business requirements in the equities and QPS (Query Per Second) trading domains. The tool, named **Trading Requirement Impact Analyzer (TRIA)**, will:

1. Accept input from Jira (via API) or manual text related to trading business requirements.
2. Search Confluence/Wiki for similar historical requirements.
3. Generate an integration design document detailing impacted systems, classes, and configuration changes.

This solution leverages Dart for analysis and document generation, with Spring Boot serving as the entry point to trigger the process.

**Project Name: Trading Requirement Impact Analyzer (TRIA)**

**Objective**

Develop a Dart-based tool, callable from a Spring Boot application, that:

* Takes business requirements from Jira or manual text input in the equities and QPS trading domains.
* Identifies similar past requirements from Confluence.
* Produces a detailed integration design document listing impacted trading applications, classes, and configuration changes.

**Use Case**

A trader or analyst submits a requirement via Jira (e.g., "Enhance order matching speed for equities trading") or text input. TRIA retrieves similar past requirements from Confluence, analyzes impacts on trading systems (e.g., Order Management System, QPS Engine), and generates a document with technical details.

**Project Structure**

**Dart Component (tria\_dart/)**

text

WrapCopy

tria\_dart/

├── lib/

│ ├── models/

│ │ ├── requirement.dart

│ │ ├── trading\_system.dart

│ │ └── impact.dart

│ ├── integrations/

│ │ ├── jira\_client.dart

│ │ └── confluence\_client.dart

│ ├── analyzers/

│ │ └── requirement\_analyzer.dart

│ ├── utils/

│ │ └── document\_generator.dart

├── bin/

│ └── tria.dart

├── config/

│ └── trading\_ecosystem.json # Defines trading systems and classes

└── pubspec.yaml

**Spring Boot Component (tria\_spring/)**

text

WrapCopy

tria\_spring/

├── src/

│ ├── main/

│ │ ├── java/com/example/tria/

│ │ │ ├── TriaApplication.java

│ │ │ ├── controller/

│ │ │ │ └── TriaController.java

│ │ │ └── service/

│ │ │ └── TriaService.java

│ │ └── resources/

│ │ └── application.properties

├── tria\_dart/ # Compiled Dart executable

│ └── tria

└── pom.xml

**Step-by-Step Implementation**

**Step 1: Define Dart Models**

**lib/models/requirement.dart:**

dart

WrapCopy

class Requirement {

final String id;

final String description;

final String source;

Requirement({required this.id, required this.description, required this.source});

Map<String, dynamic> toJson() => {'id': id, 'description': description, 'source': source};

}

**lib/models/trading\_system.dart:**

dart

WrapCopy

class TradingSystem {

final String name;

final List<String> classes;

final List<String> configs;

TradingSystem({required this.name, required this.classes, required this.configs});

}

**lib/models/impact.dart:**

dart

WrapCopy

class Impact {

final String systemName;

final List<String> classes;

final List<String> configs;

final String reason;

Impact({required this.systemName, required this.classes, required this.configs, required this.reason});

@override

String toString() => 'System: $systemName, Classes: $classes, Configs: $configs (Reason: $reason)';

}

**Step 2: Integrate with Jira and Confluence**

**lib/integrations/jira\_client.dart:**

dart

WrapCopy

import 'package:http/http.dart' as http;

import 'dart:convert';

import '../models/requirement.dart';

class JiraClient {

final String baseUrl = 'https://your-jira-instance.atlassian.net';

final String apiToken = 'your-jira-api-token';

final String username = 'your-email';

Future<List<Requirement>> fetchRequirements(String projectKey) async {

final url = '$baseUrl/rest/api/3/search?jql=project=$projectKey';

final response = await http.get(

Uri.parse(url),

headers: {

'Authorization': 'Basic ${base64Encode(utf8.encode('$username:$apiToken'))}',

'Content-Type': 'application/json',

},

);

if (response.statusCode == 200) {

final data = jsonDecode(response.body);

return (data['issues'] as List).map((issue) => Requirement(

id: issue['key'],

description: issue['fields']['summary'],

source: 'Jira',

)).toList();

}

throw Exception('Failed to fetch Jira issues');

}

}

**lib/integrations/confluence\_client.dart:**

dart

WrapCopy

import 'package:http/http.dart' as http;

import 'dart:convert';

import '../models/requirement.dart';

class ConfluenceClient {

final String baseUrl = 'https://your-confluence-instance.atlassian.net';

final String apiToken = 'your-confluence-api-token';

final String username = 'your-email';

Future<List<Requirement>> searchSimilarRequirements(String query) async {

final url = '$baseUrl/rest/api/content/search?cql=type=page%20and%20text~$query';

final response = await http.get(

Uri.parse(url),

headers: {

'Authorization': 'Basic ${base64Encode(utf8.encode('$username:$apiToken'))}',

'Content-Type': 'application/json',

},

);

if (response.statusCode == 200) {

final data = jsonDecode(response.body);

return (data['results'] as List).map((page) => Requirement(

id: page['id'],

description: page['title'],

source: 'Confluence',

)).toList();

}

throw Exception('Failed to search Confluence');

}

}

**Step 3: Analyze Requirements**

**lib/analyzers/requirement\_analyzer.dart:**

dart

WrapCopy

import 'dart:convert';

import 'dart:io';

import '../models/requirement.dart';

import '../models/trading\_system.dart';

import '../models/impact.dart';

class RequirementAnalyzer {

late List<TradingSystem> systems;

RequirementAnalyzer() {

final configFile = File('config/trading\_ecosystem.json');

final config = jsonDecode(configFile.readAsStringSync());

systems = (config['systems'] as List).map((sys) => TradingSystem(

name: sys['name'],

classes: List<String>.from(sys['classes']),

configs: List<String>.from(sys['configs']),

)).toList();

}

final Map<String, Map<String, List<String>>> keywordMappings = {

'order': {

'Equities': ['OrderMatcher', 'TradeExecutor'],

'QPS': ['QueryEngine'],

'configs': ['order\_processing.yml']

},

'payment': {

'Equities': ['PaymentProcessor'],

'QPS': [],

'configs': ['payment\_config.properties']

},

'speed': {

'Equities': ['PerformanceOptimizer'],

'QPS': ['QpsBalancer'],

'configs': ['performance\_tuning.conf']

},

};

List<Impact> analyze(Requirement requirement) {

final words = requirement.description.toLowerCase().split(' ');

final impacts = <Impact>[];

for (final word in words) {

if (keywordMappings.containsKey(word)) {

final mapping = keywordMappings[word]!;

for (final system in systems) {

final affectedClasses = mapping[system.name] ?? [];

final configs = mapping['configs'] ?? [];

if (affectedClasses.isNotEmpty || configs.isNotEmpty) {

impacts.add(Impact(

systemName: system.name,

classes: affectedClasses.where((c) => system.classes.contains(c)).toList(),

configs: configs.where((c) => system.configs.contains(c)).toList(),

reason: 'Keyword "$word" matched',

));

}

}

}

}

return impacts;

}

**config/trading\_ecosystem.json:**

json

WrapCopy

{

"systems": [

{"name": "Equities", "classes": ["OrderMatcher", "TradeExecutor", "PaymentProcessor", "PerformanceOptimizer"], "configs": ["order\_processing.yml", "payment\_config.properties", "performance\_tuning.conf"]},

{"name": "QPS", "classes": ["QueryEngine", "QpsBalancer"], "configs": ["performance\_tuning.conf"]}

]

}

**Step 4: Generate Integration Design Document**

**lib/utils/document\_generator.dart:**

dart

WrapCopy

import 'dart:io';

import '../models/requirement.dart';

import '../models/impact.dart';

class DocumentGenerator {

void generateDesignDocument(Requirement req, List<Requirement> similarReqs, List<Impact> impacts, String outputPath) {

final file = File(outputPath);

final sink = file.openWrite();

sink.writeln('# Integration Design Document');

sink.writeln('\*\*Generated on:\*\* ${DateTime.now()}');

sink.writeln();

sink.writeln('## New Requirement');

sink.writeln('- \*\*ID\*\*: ${req.id}');

sink.writeln('- \*\*Source\*\*: ${req.source}');

sink.writeln('- \*\*Description\*\*: ${req.description}');

sink.writeln();

sink.writeln('## Similar Historical Requirements');

if (similarReqs.isEmpty) {

sink.writeln('No similar requirements found.');

} else {

for (final simReq in similarReqs) {

sink.writeln('- \*\*${simReq.id}\*\*: ${simReq.description}');

}

}

sink.writeln();

sink.writeln('## Impacted Systems and Changes');

if (impacts.isEmpty) {

sink.writeln('No impacts identified.');

} else {

for (final impact in impacts) {

sink.writeln('### ${impact.systemName}');

sink.writeln('- \*\*Classes\*\*: ${impact.classes.join(', ') or 'None'}');

sink.writeln('- \*\*Configs\*\*: ${impact.configs.join(', ') or 'None'}');

sink.writeln('- \*\*Reason\*\*: ${impact.reason}');

sink.writeln();

}

}

sink.close();

}

}

**Step 5: Dart Main Logic**

**bin/tria.dart:**

dart

WrapCopy

import 'package:tria\_dart/integrations/jira\_client.dart';

import 'package:tria\_dart/integrations/confluence\_client.dart';

import 'package:tria\_dart/analyzers/requirement\_analyzer.dart';

import 'package:tria\_dart/models/requirement.dart';

import 'package:tria\_dart/utils/document\_generator.dart';

import 'dart:convert';

void main(List<String> arguments) async {

if (arguments.length < 2) {

print('Usage: dart run bin/tria.dart <mode> <input> [output]');

print('Modes: jira <projectKey>, text "<requirement>"');

return;

}

final mode = arguments[0];

final input = arguments[1];

final outputPath = arguments.length > 2 ? arguments[2] : 'design\_doc.md';

Requirement newReq;

final confluenceClient = ConfluenceClient();

final analyzer = RequirementAnalyzer();

final generator = DocumentGenerator();

if (mode == 'jira') {

final jiraClient = JiraClient();

final reqs = await jiraClient.fetchRequirements(input);

newReq = reqs.first; *// Take first issue for simplicity*

} else if (mode == 'text') {

newReq = Requirement(id: 'TEXT-1', description: input, source: 'Manual');

} else {

print('Invalid mode.');

return;

}

*// Search similar requirements*

final similarReqs = await confluenceClient.searchSimilarRequirements(newReq.description);

final impacts = analyzer.analyze(newReq);

*// Generate document*

generator.generateDesignDocument(newReq, similarReqs, impacts, outputPath);

print('Design document generated: $outputPath');

}

**Step 6: Spring Boot Integration**

**pom.xml:**

xml

WrapCopy

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

**src/main/java/com/example/tria/service/TriaService.java:**

java

WrapCopy

package com.example.tria.service;

import org.springframework.stereotype.Service;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.List;

@Service

public class TriaService {

public String generateDesignDoc(String mode, String input) throws Exception {

String dartExe = "./tria\_dart/tria";

String outputPath = "design\_doc\_" + System.currentTimeMillis() + ".md";

ProcessBuilder pb = new ProcessBuilder(dartExe, mode, input, outputPath);

pb.redirectErrorStream(true);

Process process = pb.start();

List<String> output = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()))) {

String line;

while ((line = reader.readLine()) != null) {

output.add(line);

}

}

int exitCode = process.waitFor();

if (exitCode != 0) throw new RuntimeException("Dart execution failed: " + exitCode);

return outputPath;

}

}

**src/main/java/com/example/tria/controller/TriaController.java:**

java

WrapCopy

package com.example.tria.controller;

import com.example.tria.service.TriaService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/api/tria")

public class TriaController {

@Autowired

private TriaService triaService;

@PostMapping("/generate")

public String generateDesignDoc(@RequestParam String mode, @RequestBody String input) throws Exception {

return triaService.generateDesignDoc(mode, input);

}

}

**src/main/java/com/example/tria/TriaApplication.java:**

java

WrapCopy

package com.example.tria;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class TriaApplication {

public static void main(String[] args) {

SpringApplication.run(TriaApplication.class, args);

}

}

**Step 7: Compile and Deploy**

1. **Dart**:

bash

WrapCopy

cd tria\_dart

dart pub get

dart compile exe bin/tria.dart -o tria

cp tria ../tria\_spring/tria\_dart/

1. **Spring Boot**:

bash

WrapCopy

cd tria\_spring

mvn clean install

mvn spring-boot:run

**Step 8: Test the Application**

* **Via Curl**:

bash

WrapCopy

curl -X POST "http://localhost:8080/api/tria/generate?mode=text" -H "Content-Type: text/plain" -d "Enhance order matching speed for equities trading"

* **Output**: A file like design\_doc\_1677654321.md with impacted systems (Equities, QPS), classes, and configs.

**Sample Output (design\_doc.md)**

markdown

WrapCopy

# Integration Design Document

**\*\*Generated on:\*\*** 2025-02-21 23:06:00

## New Requirement

- **\*\*ID\*\***: TEXT-1

- **\*\*Source\*\***: Manual

- **\*\*Description\*\***: Enhance order matching speed for equities trading

## Similar Historical Requirements

- **\*\*12345\*\***: Improve trading speed for equities (Confluence)

## Impacted Systems and Changes

### Equities

- **\*\*Classes\*\***: OrderMatcher, PerformanceOptimizer

- **\*\*Configs\*\***: order*\_processing.yml, performance\_*tuning.conf

- **\*\*Reason\*\***: Keyword "order" matched

### QPS

- **\*\*Classes\*\***: QpsBalancer

- **\*\*Configs\*\***: performance*\_tuning.conf*

*-* ***\*\*Reason\*\*****: Keyword "speed" matched*

**Enhancements**

* **Similarity Scoring**: Use cosine similarity or NLP for better Confluence matching.
* **Config Validation**: Check if configs exist in the system.
* **PDF Output**: Use pdf Dart package for richer documents.
* **Async Processing**: Implement Spring Boot async calls for large-scale analysis.

This solution meets your needs for trading applications in equities and QPS domains, integrating seamlessly with Jira, Confluence, and Spring Boot. Let me know if you need further refinements!

### Detailed Design Document

#### 1. Overview

The **TRIA** system is designed to:

* Accept business requirements from Jira or manual text input related to equities and QPS trading domains.
* Search Confluence for similar historical requirements.
* Generate an integration design document detailing impacted systems, classes, and configuration changes.
* Be triggered via a Spring Boot REST API.

#### 2. Functional Requirements

* **Input Sources**:
  + Jira: Fetch requirement summaries from a specified project.
  + Manual: Accept text input from the Spring Boot API.
* **Similarity Search**: Query Confluence for pages with similar content to the new requirement.
* **Impact Analysis**: Identify affected trading systems (Equities, QPS), classes, and configuration files.
* **Output**: Generate a Markdown-based integration design document.

#### 3. Non-Functional Requirements

* **Performance**: Process a single requirement in under 10 seconds.
* **Scalability**: Support fetching multiple Jira issues or Confluence pages if extended.
* **Security**: Use API tokens for Jira/Confluence access, stored securely (e.g., environment variables).
* **Maintainability**: Modular design for easy updates to trading system configurations.

#### 4. System Components

##### 4.1 Spring Boot Application (tria\_spring)

* **Purpose**: Acts as the entry point, exposing a REST API to trigger the Dart tool.
* **Modules**:
  + **Controller (TriaController)**:
    - Endpoint: POST /api/tria/generate?mode=<mode>
    - Input: mode (jira/text), body (requirement text or project key).
    - Output: Path to the generated design document.
  + **Service (TriaService)**:
    - Executes the Dart executable as a subprocess.
    - Captures output and returns the document path.
* **Dependencies**: Spring Web, Java 17.

##### 4.2 Dart Tool (tria\_dart)

* **Purpose**: Core logic for requirement processing, analysis, and document generation.
* **Modules**:
  + **Integrations**:
    - JiraClient: Fetches requirements from Jira using REST API.
    - ConfluenceClient: Searches Confluence for similar requirements.
  + **Models**:
    - Requirement: Represents a requirement with ID, description, and source.
    - TradingSystem: Defines trading apps (Equities, QPS) with classes and configs.
    - Impact: Captures analysis results (system, classes, configs, reason).
  + **Analyzer (RequirementAnalyzer)**:
    - Loads trading ecosystem from trading\_ecosystem.json.
    - Maps keywords to systems, classes, and configs.
    - Analyzes requirement text for impacts.
  + **Utils (DocumentGenerator)**:
    - Generates Markdown document with requirement details, similar requirements, and impacts.
  + **Main (tria.dart)**:
    - Orchestrates input processing, analysis, and document generation.

##### 4.3 Configuration (trading\_ecosystem.json)

* **Purpose**: Defines the trading ecosystem for impact analysis.
* **Structure**:

json

WrapCopy

{

"systems": [

{

"name": "Equities",

"classes": ["OrderMatcher", "TradeExecutor", "PaymentProcessor", "PerformanceOptimizer"],

"configs": ["order\_processing.yml", "payment\_config.properties", "performance\_tuning.conf"]

},

{

"name": "QPS",

"classes": ["QueryEngine", "QpsBalancer"],

"configs": ["performance\_tuning.conf"]

}

]

}

#### 5. Workflow

1. **Input Handling**:
   * Spring Boot receives a POST request with mode (jira/text) and input.
   * For "jira" mode, Dart fetches issues from Jira.
   * For "text" mode, Dart uses the provided text directly.
2. **Similarity Search**:
   * Dart queries Confluence with keywords from the new requirement.
   * Retrieves a list of similar historical requirements.
3. **Impact Analysis**:
   * Dart analyzer splits requirement text into keywords.
   * Matches keywords against trading system mappings.
   * Identifies impacted systems, classes, and configs.
4. **Document Generation**:
   * Dart generates a Markdown file with:
     + New requirement details.
     + Similar historical requirements.
     + Impacted systems with classes/configs.
5. **Output Delivery**:
   * Spring Boot returns the path to the generated document.

#### 6. Technical Specifications

* **Languages**: Java (Spring Boot), Dart.
* **Dependencies**:
  + Spring Boot: spring-boot-starter-web.
  + Dart: http package for API calls.
* **APIs**:
  + Jira REST API: https://<instance>.atlassian.net/rest/api/3/.
  + Confluence REST API: https://<instance>.atlassian.net/rest/api/.
* **Authentication**: Basic Auth with API token (username:token base64-encoded).
* **Output Format**: Markdown (extensible to PDF with additional libraries).

#### 7. Data Flow

* **Input**: String (text or Jira project key).
* **Processing**:
  + Jira → Requirement object.
  + Text → Requirement object.
  + Confluence → List of similar Requirement objects.
  + Analyzer → List of Impact objects.
* **Output**: Markdown file (design\_doc\_<timestamp>.md).

#### 8. Error Handling

* **Jira/Confluence Failure**: Return HTTP 500 with error message.
* **Dart Execution Failure**: Throw exception in Spring Boot, log error.
* **Invalid Input**: Return HTTP 400 with validation message.

#### 9. Assumptions

* Jira and Confluence instances are accessible with API tokens.
* Trading ecosystem is predefined in trading\_ecosystem.json.
* Requirements are concise and keyword-driven for analysis.

### Architectural Diagram

Below is a textual representation of the architecture that you can convert into a visual diagram using tools like PlantUML (code provided at the end).

#### Diagram Description

text

WrapCopy

+-------------------+ +-------------------+

| Client (User) | | Jira |

| (Browser/Postman) | (Requirement Source) |

+-------------------+ +-------------------+

| |

| POST /api/tria/generate |

v |

+-------------------+ +-------------------+

| Spring Boot App |<----->| Confluence |

| (REST API) | | (Historical Data) |

| - TriaController | +-------------------+

| - TriaService |

+-------------------+

|

| Executes

v

+-------------------+

| Dart Tool (TRIA) |

| - JiraClient |

| - ConfluenceClient|

| - RequirementAnalyzer |

| - DocumentGenerator |

+-------------------+

|

| Generates

v

+-------------------+

| Design Document |

| (design\_doc.md) |

+-------------------+

#### Key Interactions

1. **Client → Spring Boot**: User sends requirement via POST request.
2. **Spring Boot → Jira**: (If Jira mode) Dart fetches requirement data.
3. **Spring Boot → Confluence**: Dart searches for similar requirements.
4. **Dart → Internal Analysis**: Analyzes impacts using trading\_ecosystem.json.
5. **Dart → File System**: Writes design document.
6. **Spring Boot → Client**: Returns document path.

#### PlantUML Code (for Diagram)

plantuml

WrapCopy

@startuml

actor Client

participant "Spring Boot App" as Spring

participant "Jira" as Jira

participant "Confluence" as Confluence

participant "Dart Tool (TRIA)" as Dart

participant "Design Document" as Doc

Client -> Spring: POST /api/tria/generate (mode, input)

Spring -> Dart: Execute tria <mode> <input> <output>

alt mode = "jira"

Dart -> Jira: Fetch requirements (projectKey)

Jira --> Dart: List of issues

end

Dart -> Confluence: Search similar requirements

Confluence --> Dart: List of pages

Dart -> Dart: Analyze impacts (trading\_ecosystem.json)

Dart -> Doc: Generate design\_doc.md

Dart --> Spring: Output path

Spring --> Client: Return document path

@enduml

* Paste this into a PlantUML tool (e.g., <https://www.plantuml.com/plantuml>) to generate the diagram.

### Detailed Design Highlights

* **Modularity**: Spring Boot and Dart components are loosely coupled, connected via process execution.
* **Extensibility**: Add new trading systems or keywords in trading\_ecosystem.json and keywordMappings.
* **Scalability**: Spring Boot can handle multiple requests; Dart can be optimized with compiled binaries.
* **Reusability**: Historical data from Confluence aids in decision-making.

This design meets your needs for trading applications in equities and QPS domains, integrating with Jira and Confluence, and providing a comprehensive integration design document. Let me know if you need further elaboration or adjustments!