SOFTWARE ARCHITECTURE DOCUMENT

Concordia University
Department of Computer Science and Software Engineering
SOEN 6441
Advanced Programming Practices

Project

Team Members:

Name: Jatin Arora
Applicant ID: 40225129

Name: Rahul Singh
Applicant ID: 40228461

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

This document specifies a broad architectural overview of the system, by means of a number of different architectural views to illustrate various aspects of the system.

1.2 Scope

This document provides an architectural overview of a system, that fetches data from Cricbuzz Cricket API, stores the data in a local sqlite database, and fetches the data from the database based on user input.

1.3 Github

This project is available at:
https://github.com/jatin51997/Project

2. Architectural Representation

This document presents the architecture as a series of diagrams; Class Diagram, Relational Diagram, along with various object relational structural patterns, data source architectural patterns and design patterns that have been have been incorporated as a part of the system.

3. Use Case View

The use cases incorporated in the system are:

3.1 Select Team

The user can select any team from the dropdown menu. Based on the selected team, the dropdown for players gets populated.

3.2 Select Player

The user can select any player from the dropdown menu.

3.3 View stats

The user can view the stats of the selected player.

3.4 Select Format

The user can select any cricket format.

3.5 View Rankings

The user can view the rankings, based on the selected format.

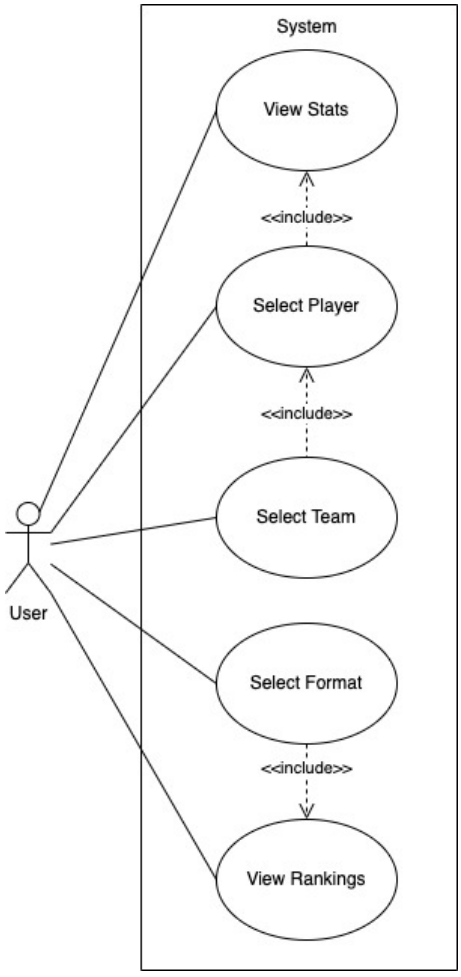

Fig.1 Use Case Diagram

4. Logical View

The logical view is presented using the class diagram. We have five classes namely, TeamInfo, PlayerInfo, BattingStats, BowlingStats, and Rankings. We have two interfaces namely Average and FormatInfo. The attributes and methods of each class have been indicated in the diagram.

The relationship between each of the classes and interfaces is as below:

- 1 to many composition relationship between the class TeamInfo and PlayerInfo.
- 1 to many composition relationship between the class TeamInfo and Rankings.
- 1 to many composition relationship between the class PlayerInfo and BattingStats.
- 1 to many composition relationship between the class PlayerInfo and BowlingStats.
- 1 to many composition relationship between the class Rankings and TeamInfo.
- Class BattingStats and Class BowlingStats implement the interface formatInfo and Average.
- Class Rankings Implement the interface formatInfo.

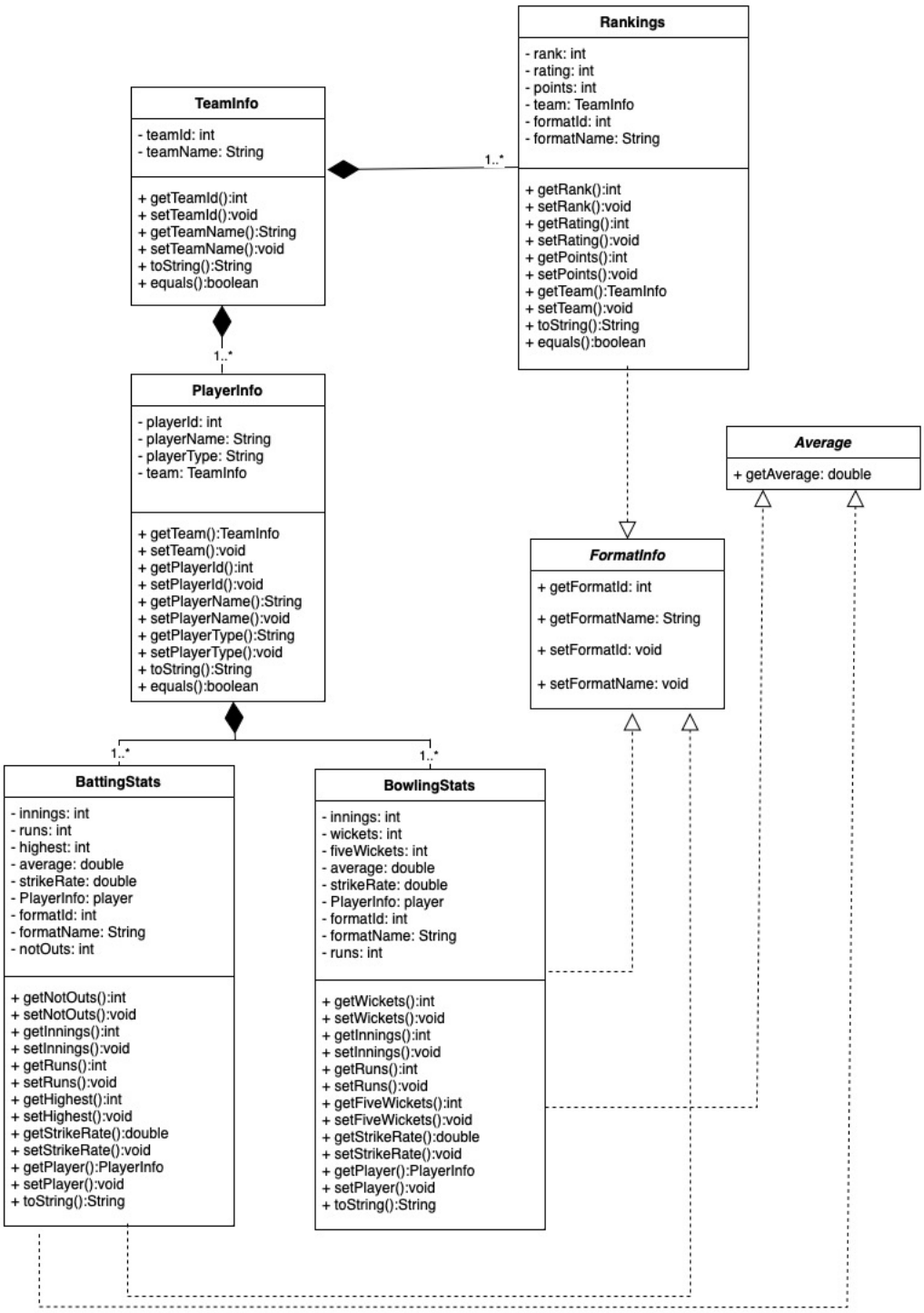

Fig.2 Class Diagram

5. Data View

The data view is illustrated using the relational diagram. We have maintained six tables in our database, namely TEAMINFO, PLAYERINFO, BATTINGSTATS, BOWLINGSTATS, RANKINGS, and FORMATINFO. The columns for each table have been indicated in the diagram, along with their primary and foreign keys.

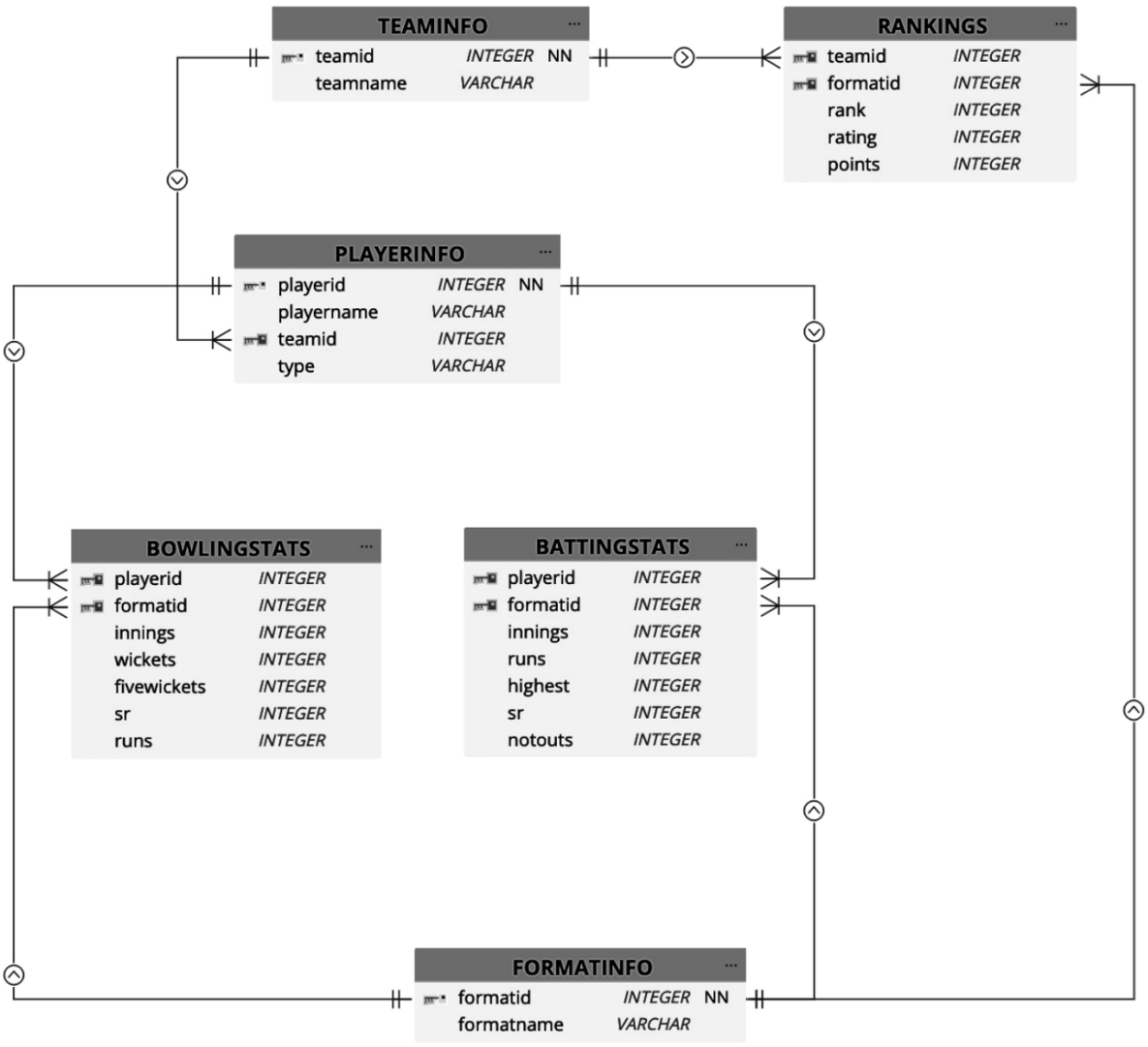

Fig.3 Relational Diagram

6. Patterns

6.1 Object Relational Structural Pattern

The system design is based on Table-per-class inheritance mapping (Vertical Mapping). Each class in our system has its own table, and each attribute of the class is mapped to its own column in the table.

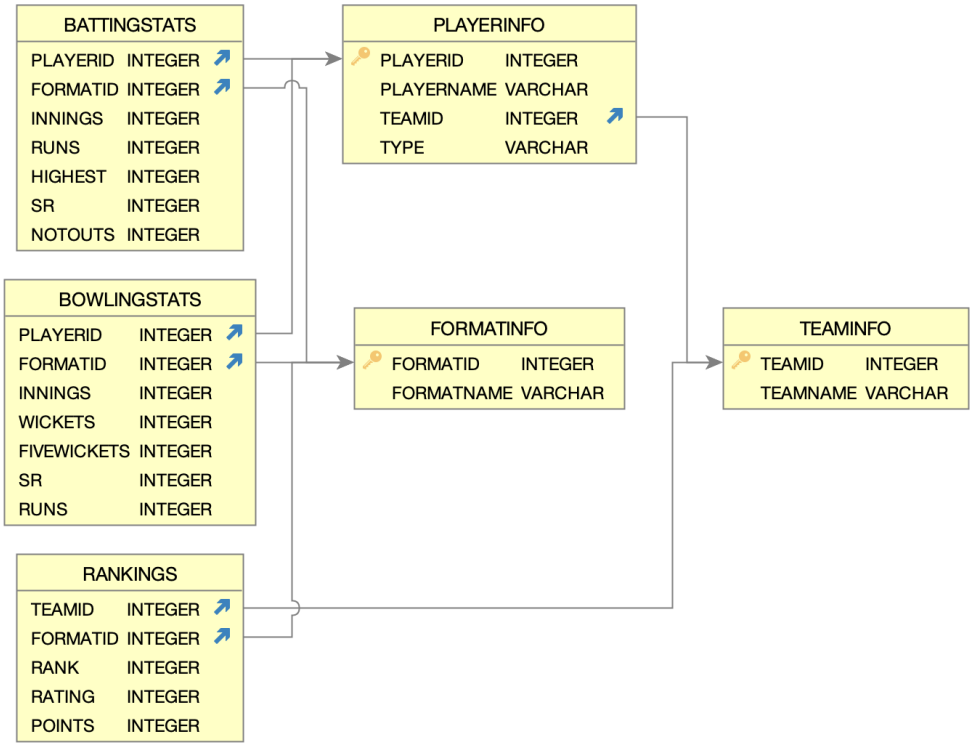

Fig.4 Table per class inheritance mapping

6.2 Data Source Architectural Pattern

The user interacts with the system at the front end using the UI. The call from the front end to the backend is handled inside the servlet class.

The servlet class redirects the call to the mapper class which acts like a Data Mapper and is responsible for creating domain objects and passing them to the TDG class.

The TDG class acts like a Table Data Gateway and contains all the SQL code. It is responsible for mapping the input parameters to the SQL call and executing it against the database. In our system the TDG handles the select and insert queries.

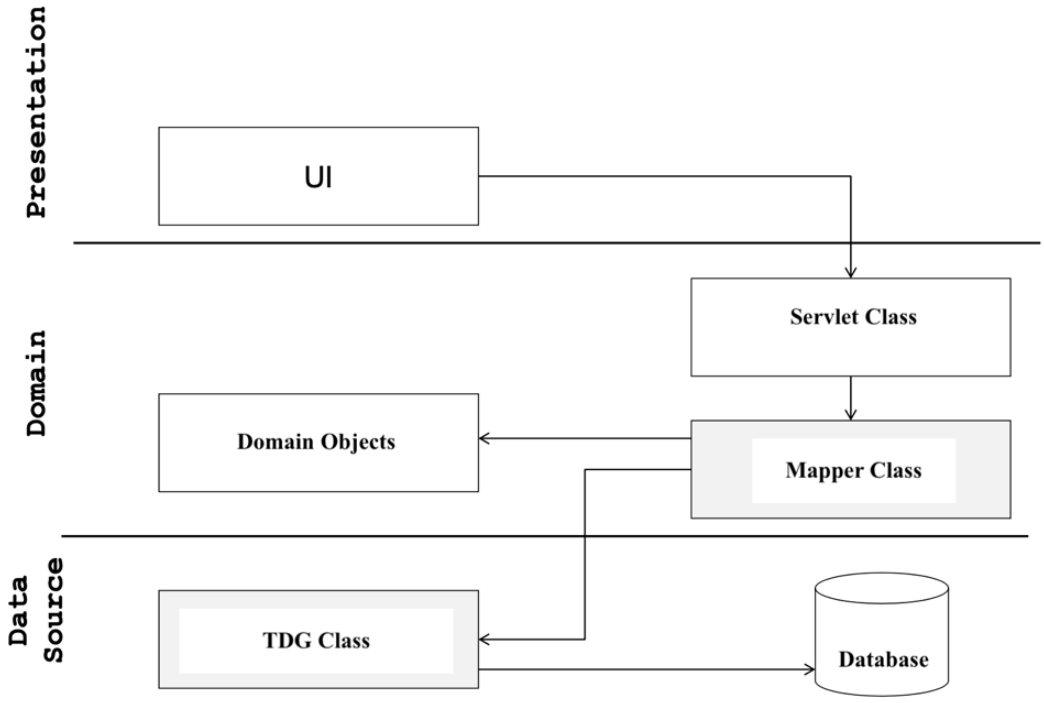

Fig.5 Data Source Architectural Pattern

6.3 Design Patterns

6.3.1 Singleton pattern

It is a kind of creational design pattern. This system incorporates the Singleton pattern for two classes, the TDG class and the Mapper class.

The singleton pattern for TDG class ensures a single DB connection shared by multiple objects.

The object of each of these classes are instantiated only once. This has been achieved using a private constructor that is implicitly called once upon the request to obtain an instance of the class. There exists a getInstance() method that checks if an instance of the class has already been instantiated or not.

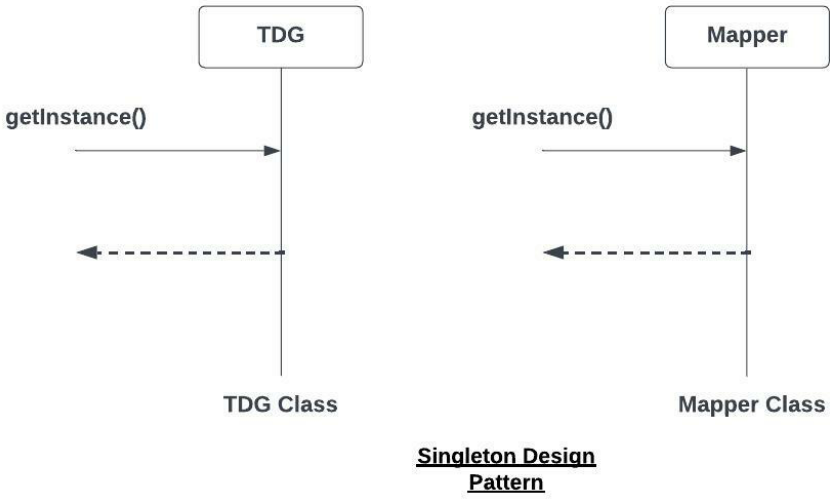

Fig.6 Singleton Design Pattern

6.3.2 Strategy Pattern

It is a kind of Behavioral design pattern. In Strategy pattern, a class behavior can be changed at run time. In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

The system has an interface called Average which has a method getAverage. The classes BattingStats and BowlingStats implement the interface and provide their own version of the method getAverage. We have a class called context which is responsible for determining the runtime behavior of getAverage method.

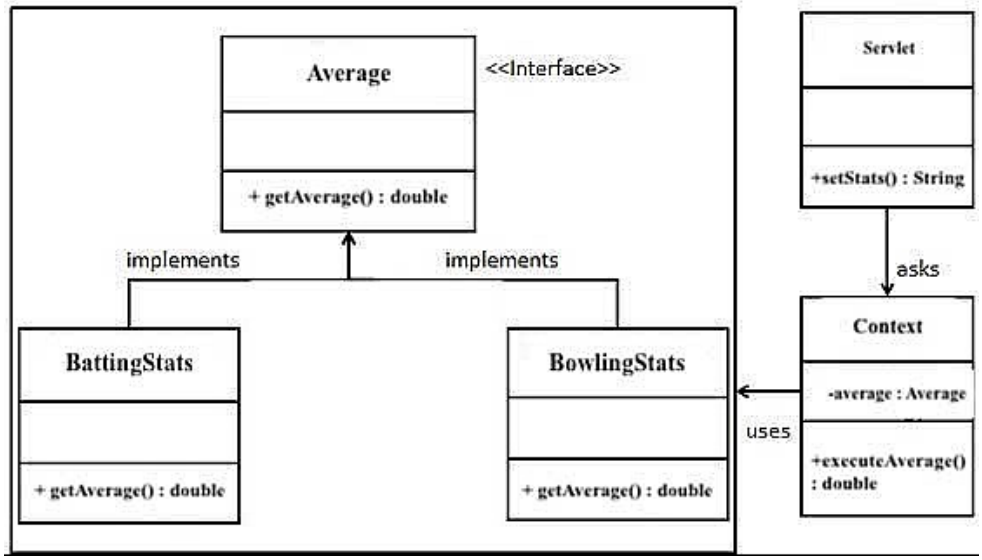

Fig.7 Strategy Pattern

7. Refactoring

We have incorporated the following refactoring techniques, and comments have been added in the code to represent the same.

7.1 Extract Method
7.2 Extract Class
7.3 Decompose Conditional
7.4 Consolidate Duplicate Conditional
7.5 Assertions
7.6 Replace inheritance with delegation

8. Testing

The testing of the system has been performed on Junit5 framework. The unit test cases of the system are as follows:

S.No	Test Case	Description
1	checkStatusCode200	Check if each endpoint of API return status code 200.
2	checkContentTypeJSON	Check if each endpoint of API returns content of type JSON.
3	checkAppropriateJSONkeys	Check if response of each API contains appropriate JSON key value pairs.
4	checkDataBaseConnection	Check if database connection is working.
5	checkDataBaseQueries	Check if inserted records in database are correct.

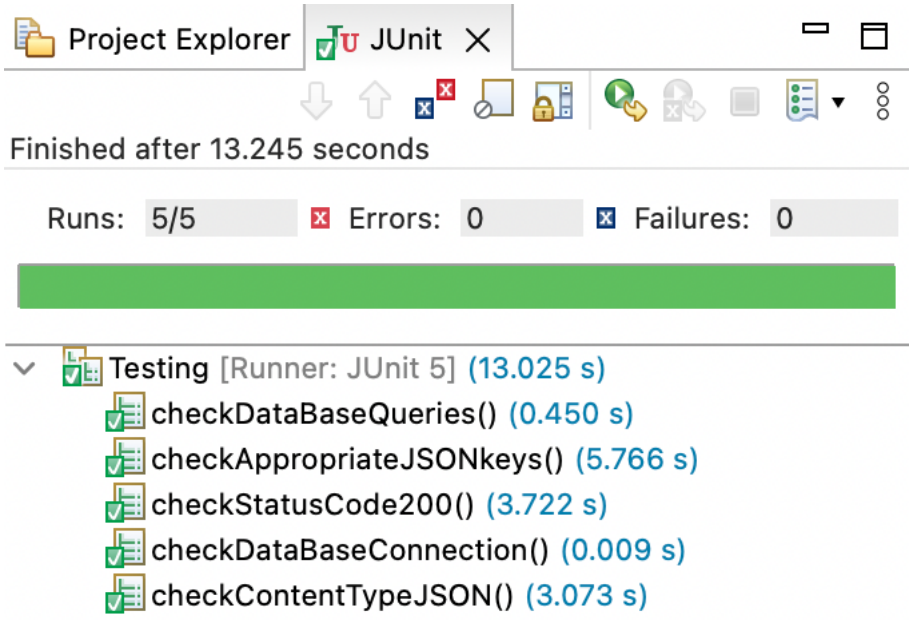

Fig.8 Testing results