Laboratory ReportIntroduction

We designed *CalcioTrends* with the objective of creating a website that offers a different point of view on soccer statistics from what the market offers. We tried to distinguish ourselves by creating a general evaluation for players that measures their impact on a game. This kind of evaluation allows us to summarize a variety of statistics for a single player into one simple *"Impact Score"* making it easy even for novice users to understand the world of soccer.
It's important to note that the Impact Score was created based on the data provided, it soon becomes apparent that the *score* suffers from a bias towards attacking players, ideally with more complete data on the actions of each player the evaluation will be more complete and accurate.

Technical tasks:

1. Data Processing

Task:
Process the data in order to avoid errors when using it in both the analysis and the databases. Modify datasets with the objective of avoiding processing null or erroneous values.

Solution:
We created a series of python scripts to set null values to easily recognizable ones that wouldn't disrupt their processing (eg. setting null integers to -1), for ease of accessibility and to improve the performance of our requests we decided to append the calculated *Impact scores* to the players database. By doing so we managed to reduce the number of requests by 50% dramatically improving our performance.

Issues:
Originally the scripts were made to convert both the "errors" and to convert the specified dataset into json, however some of the team's machines had issues loading the new datasets both through the API and GUI tools such as IntelliJ Idea. To fix these issues new versions of the scripts were made to simply correct the errors while not changing the file format.

2. Creating the Springboot/Postgres database

Task:
Following the lectures we used the SpringBoot framework to manage our postgres database.
Solution:
We first created our database using pgAdmin4.

Since we were already provided with numerous examples and starting points during the lectures we started by implementing the “Players” table taking for reference the “scorer” example from Lecture 16.

Issues:

While the Spring Boot server implementation aims for universality and machine independence, some of us encountered issues during compilation. Specifically, there were challenges in identifying primary keys and correctly indexing the generated databases. Our resolution came with the switch to Java 22, which effectively resolved the issue.

3. Creating the Mongo database

Task:

We used mongoDB’s Compass to create the database, manage tables and test aggregations before implementing them in our Express server.

Solution:

For the first initialization of the database we decided to perform some basic tests such as connection from IntelliJ’s database page and some basic aggregations to ensure that all the data was properly loaded on a single machine, repeating the process on all of the team's computers.

Issues:

Unlike Sprigboot we didn’t encounter any particular problem with setting up this database, providing the csv files to Compass automatically created the schemas and uploaded the data with the appropriate data types.

It was briefly considered to use an external service for hosting the database in order to remove the possibility of introducing data discrepancy but due to time constraints we decided to not proceed with this option.

4. Populating the databases

Task:

Once the databases were created we needed to populate them with the data from the datasets.

Solution:

We decided to use the functionalities provided by IntelliJ and Compass to populate the databases, this allowed us to easily populate the databases with the data from the datasets.

Issues:

We initially created an API to populate the Springboot database, however, we soon realized that due to the previously mentioned issues using the API was not viable. We decide to keep the API as a proof of concept for possible future use.

5. Designing the interface

Task:

Under the suggestion of some colleagues we opted to use BootstrapStudio to rapidly design the websites’ interface.

Solution:

While we previously designed an interface through the use of Figma we weren't able to export the design to a format that could be easily implemented in the website. BootstrapStudio allowed us to create a more functional and visually appealing interface in a fraction of the time.

Issues:

As with most tools the learning curve was steep, however, the team managed to overcome this by using the tutorials provided by the software and the community.

6. Creating the Express servers

Task:

We used Express to create both the server that would interact with the MongoDB database and the main server that would serve the website.

Solution:

We started by implementing a simple server that would serve the website, allowing us to implement a page in its entirety before moving on to making requests to the databases .We chose the Players page since it needs data from both mongoDB and Springboot databases.

Once we fully understood the process of making requests to the databases we started implementing the rest of the pages.

Issues:

In the first moment we chose to export our design to html and css files, however, we soon realized that this would make it impossible not to break cross-origin policies. To fix this we decided to move to EJS, due to our initial commitment to the html and css files we decided to keep the "static" approach to the website.

Another issue we encountered was the lack of knowledge on how to properly make requests to the databases, however, after some research and testing we managed to overcome this issue.

7. Data analysis

Task:

After implementing the main webpages we started working on what possible graphs we'd want to add in a possible future to the website. With this in mind we settled on developing easy to understand graphs that could be used to show the data we were given.

Solution:

Following a series of preliminary tests, our team opted to retain the maximum amount of data attainable, regardless of its level of completeness. This decision was grounded in the observation that the impact of incomplete data could be substantially offset by the sheer volume of the dataset. In our endeavor to create visual representations that are both accessible and meaningful, we embarked on developing a new metric: the Impact Score. This metric was designed to facilitate the evaluation of individual players' performances comprehensively. To compute the Impact Score, we aggregated the values of several key performance indicators, namely goals, assists, shots, passes, and tackles. Subsequently, we normalized the

Impact Score to a scale ranging from 0 to 100, ensuring a standardized assessment across players.

Issues:

While it started as an ideal simplified metrics to evaluate a single player the Impact Score soon started to show its limitations. The first issue we encountered was due to the nature of the data in fact, within the datasets provided there is little to no data regarding saves or defensive actions such as tackles thus making it impossible to accurately evaluate players who do not strictly participate in attacking actions. This transformed the Impact Score into a glorified goals and assists metric which is not what we were aiming for. Another major issue was how sparse the dataset was, with many players having only a few games worth of data, this made it difficult to evaluate players in an accurate manner resulting in rookies with good performance in very few games being rated higher than veterans with a consistent performance over the season. Still with a more complete and less sparse dataset the Impact Score could be a very useful metric to evaluate a player's overall contribution to the team.

Conclusion:

Throughout the development of this project, we recognized the effectiveness of utilizing repetitive patterns in managing requests to the two servers. In this regard, we found Github Copilot invaluable for expediting certain implementations. A well-structured division of labor was crucial in swiftly developing each section of the site,

Areas for Improvement in Our Site:

We allocated less emphasis to the graphical aspect, which could undoubtedly be enhanced to make the site more appealing to potential users.

Division of work:

We believe the division of work was equitable. Our team fostered a high level of collaboration, emphasizing cooperation and parallel work. To synchronize efforts, we collectively developed the 'Players' page to address potential errors, after which each team member autonomously managed their respective tasks.

Specifically, Edoardo focused primarily on the 'Teams' and 'Championships' pages and their specific details. Alice managed 'Specific Player', 'Latest Matches', and 'Specific Match' pages. Enrico handled 'Home', 'Chat', and managed the graphic design of the site to resolve any visual artifacts.

Throughout the site's development, challenges arose, which were often resolved through team-wide participation to find optimal solutions and streamline timelines. In particular, Edoardo served as the main contact for Spring Boot server-related issues, Alice supervised and supported MongoDB server management, while Enrico managed the client-server aspect.

Bibliography:

Use of Github Copilot to implement repetitive requests on the site and address potential errors.

Utilization of Stack Overflow to understand how to resolve certain issues.

Use of ChatGPT for more complex MongoDB functions, which were used as examples to create new ones.