TDT4250 - Model-Driven Software Development
Assignment 1

M. H. Johannessen

13.09.2024

Contents

Figures

Tables

Chapter 1

Introduction

The Football Tournament Management Application (FTA) was developed as part of the course IDATG1002 - Software Engineering[1] at NTNU Gjøvik. A central objective of this course is to provide students with practical experience in the design and implementation of complex software systems. This was achieved through hands-on projects that required the application of key software engineering principles, including modular design, data persistence, and user interface development, in a real-world setting. In 2022, the specific challenge assigned was to create a tournament management application for a sport of the student's choice.

This document is part of the coursework for TDT4250 - Model-Driven Software Engineering[2] at NTNU Trondheim. The goal of this assignment is to reason about variability, features, and variation points in a familiar software system. On that occasion, this document provides an overview of the FTA, including a thorough system description, an analysis of the current features and variability mechanisms, and a discussion on possible variability extensions.

[1]IDATG1002: https://www.ntnu.edu/studies/courses/IDATG1002/2020
[2]TDT4250: https://www.ntnu.edu/studies/courses/TDT4250

Chapter 2

System description

The FTA is a desktop application developed in Java using JavaFX for the user interface an Maven for dependency management. It is designed to facilitate the organization and management of football tournaments, and does so by supporting various tournament formats, like single-elimination, double-elimination, and round-robin tournaments. The application provides functionalities for creating and managing teams, scheduling matches, tracking scores, and visualizing the tournament bracket.

Architectural-wise the application is structured according to the *Model-View-Controller* (MVC) pattern, which separates the system into three distinct components: the model, the view, and the controller [1].

The model layer comprises several classes that represent the fundamental data structures of the application. For instance, the *Team.java* class manages the details of individual teams. The *Match.java* class is responsible for representing the details of each match, such as the participating teams, scores, and match outcomes. The *Bracket.java* class organizes the progression of matches within the tournament, while the *Tournament.java* class oversees the overall flow of the tournament.

The view layer is implemented using FXML files, which define the user interface layout. JavaFX is utilized to render these interfaces. Key FXML files include *BracketView.fxml*, which defines the layout for visualizing the tournament bracket; *MainMenuView.fxml*, which serves as the primary navigation hub for users; and *TournamentMakerView.fxml*, which provides the interface for setting up new tournaments.

The controller layer acts as the intermediary between the model and the view layers, processing user inputs, executing the necessary business logic, and updating the user interface accordingly. For example, *BracketController.java* handles the logic for updating and displaying the tournament bracket based on match results. Similarly, *TournamentMakerController.java* manages the user interactions during the tournament setup process.

Class	Attributes*	Relationships
Team	- name - manager - totalGoals - goalsThisRound	1..* Teams participate in 1..* Matches 1 Team belongs to 1 Tournament
Player	- playerName - playerStats	1 Player belongs to 1 Team
Match	- team1 - team2 - scoreTeam1 - scoreTeam2 - winner	1..* Matches belong to 1 Bracket 2 Teams participate in 1 Match
Bracket	- bracketSide - rounds - matches	1 Bracket contains 1..* Matches 1 Bracket belongs to 1 Tournament
Tournament	- instance - leftBracketMatches - leftBracketTeams - rightBracketMatches - rightBracketTeams - tournamentType	1 Tournament manages 1..* Teams 1 Tournament contains 1 Bracket
User**	- username - role (Admin, Coach)	1 User manages 1..* Tournaments
TournamentFileHandler	- saveTournamentData - loadTournamentData	Handles 1..* Tournaments

Table 2.1: Representation of the Classes, Attributes, and Relationships in the FTA
*Attributes differ slightly from the implemented ones, but the content is consistent.
**User-rights separation is not implemented.

Chapter 3

Current features

The system's functionalities include several essential features. The FTA's user interface is driven primarily by MainMenuView.fxml, with interactions managed by MainMenuController.java. The main menu serves as the central hub for all major functionalities, including starting new tournaments, loading existing ones, and accessing additional resources like the FAQ section. The FAQ section, implemented through FAQView.fxml and FAQController.java, provides users with guidance on using the application (See Appendix A for a User Manual).

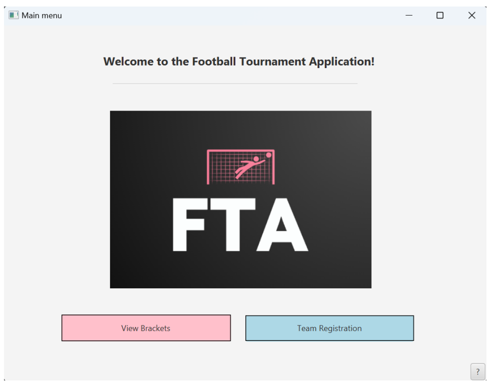
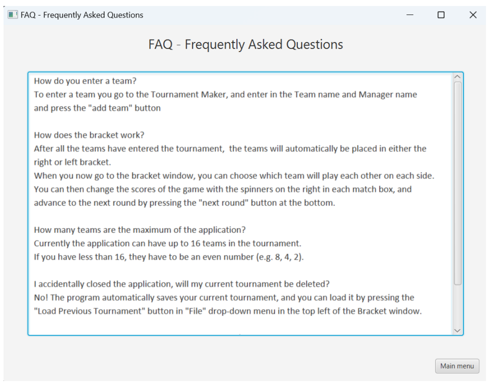


Figure 3.1: Main Menu



FAQ - Frequently Asked Questions

How do you enter a team?
To enter a team you go to the Tournament Maker, and enter in the Team name and Manager name and press the "add team" button

How does the bracket work?
After all the teams have entered the tournament, the teams will automatically be placed in either the right or left bracket.
When you now go to the bracket window, you can choose which team will play each other on each side.
You can then change the scores of the game with the spinners on the right in each match box, and advance to the next round by pressing the "next round" button at the bottom.

How many teams are the maximum of the application?
Currently the application can have up to 16 teams in the tournament.
If you have less than 16, they have to be an even number (e.g. 8, 4, 2).

I accidentally closed the application, will my current tournament be deleted?
No! The program automatically saves your current tournament, and you can load it by pressing the "Load Previous Tournament" button in "File" drop-down menu in the top left of the Bracket window.

Figure 3.2: FAQ

FTA is managed using Maven. Maven handles the project's dependencies, build process, and overall configuration. This ensures that all necessary libraries are included and up-to-date, and that the project can be easily built and deployed. Maven's integration with Java and its use in managing large projects is particularly beneficial in maintaining the FTA's modular structure and ensuring that all components work together.

Team management is a core feature of the FTA. It allows users to create and manage teams participating in a tournament. Each team is represented by the

Team.java class, which is designed to encapsulate the attributes and behaviors of a football team, such as the team's name and its players. Although the original design intended for the system to allow full customization of teams — including player details — the current implementation restricts customization to the team level only. The ability to add or modify individual player information is not yet supported, highlighting a potential area for future enhancement.

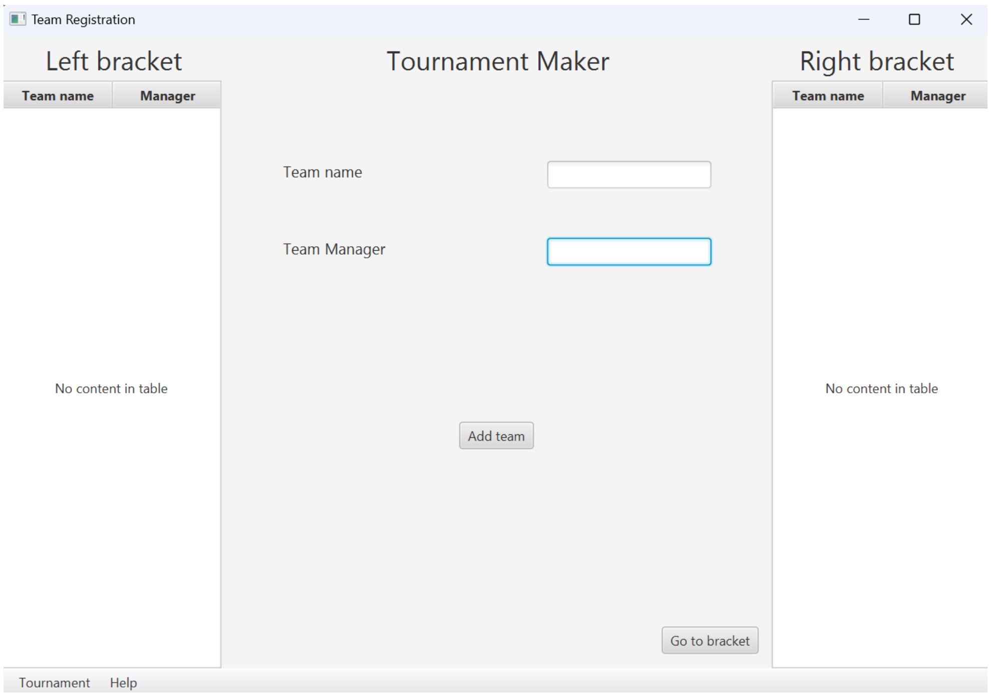


Figure 3.3: Team Registration

A comprehensive match scheduling and management system is included in the FTA. The Match.java class handles the details of individual matches, including the teams involved, the match scores, and the outcomes. While the system does not enforce a specific tournament format, it provides the flexibility to manually select and manage matches. This approach allows users to adapt to the tournament structure to their specific needs.

The key feature of the FTA is its dynamic bracket visualization. This feature, defined in BracketView.fxml and managed by BracketController.java, provides a real-time visual representation of the tournament's progress. As matches are completed and results are entered, the bracket is automatically updated to reflect the advancing teams. This visual feedback is crucial for both users and administrators, as it offers an immediate overview of the tournament's status. The separation of concerns, achieved through the MVC architecture, ensures that the visual elements remain decoupled from the underlying logic [1].

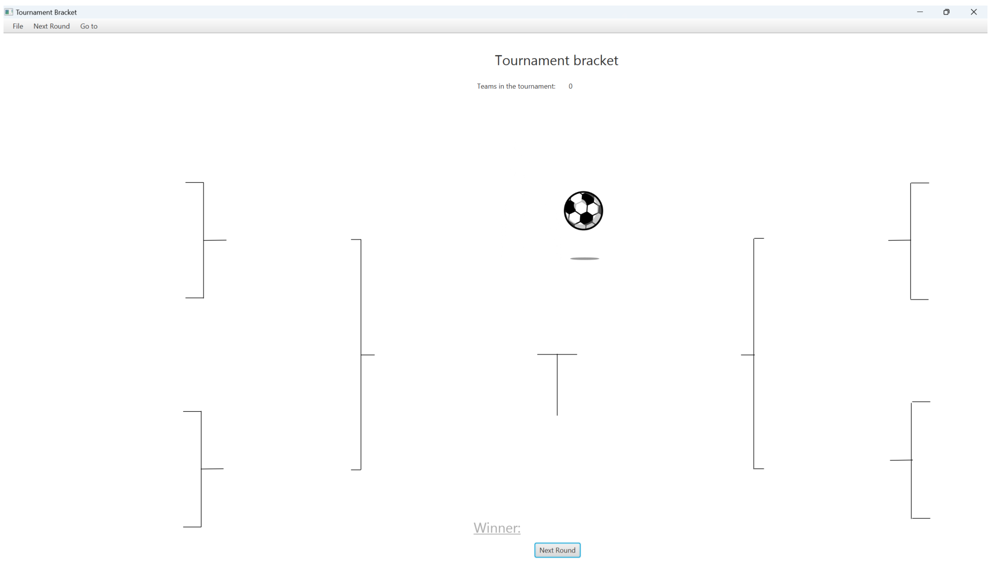


Figure 3.4: Tournament Bracket - New File

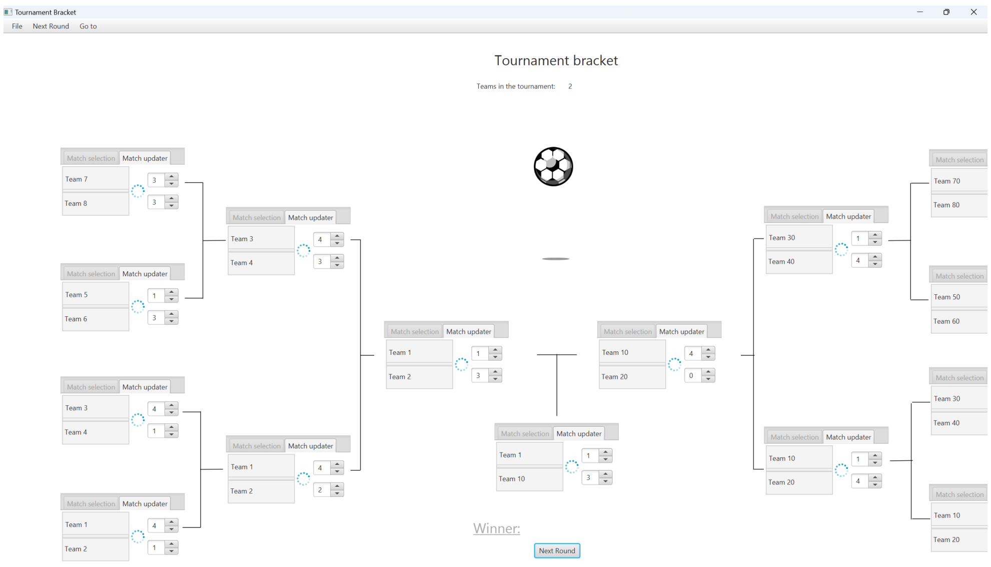


Figure 3.5: Tournament Bracket - Loaded File

Data persistence is another important component of the FTA. The TournamentFile-Handler.java class is responsible for saving and restoring tournament data across sessions. This is achieved through serialization, where the application's state is stored in a storable format, and deserialized when loading saved tournaments. This functionality ensures that users can pause and resume their tournaments without losing progress - a feature that is particularly important for larger tournaments that may span several days or weeks.

Chapter 4

Current variability mechanisms

Variability is a core concept in software engineering, especially in the context of Software Product Lines (SPLs). Variability allows a software system to be adaptable and flexible, enabling the creation of different versions or configurations of the system from a common set of artifacts [1] [2]. The FTA implements several foundational features essential for managing football tournaments. However, the current state of its variability mechanisms indicates areas for potential enhancement.

At present, the system does not explicitly support multiple tournament structures, such as single-elimination, double-elimination, or round-robin formats. The Tournament.java and Bracket.java classes manage the overall flow and match progression within a tournament, but these are implemented in a static manner. They lack distinct methods or subclasses that would allow users to dynamically select and manage different tournament formats [2]. Thus, the system does not currently exhibit variability in tournament structures.

The Match.java class is responsible for managing individual match outcomes, including the participating teams and the final results. Currently, the system supports basic win/lose scenarios but lacks the capability to handle more complex outcomes, such as ties, extra time, or penalty shootouts. In its current state, the system assumes that users will manually resolve these situations by adding a goal for the winning team. While this manual intervention introduces a limited form of user-driven variability, the system itself does not automate or provide built-in mechanisms for handling these variations.

The Team.java class within the FTA manages the basic details of teams, such as team names and players. However, it lacks support for variability in team configurations, such as differing team sizes or specific player attributes. The system currently applies a one-size-fits-all approach to team management, which means there is no inherent variability in how teams can be configured.

Data persistence in the FTA is managed by the TournamentFileHandler.java class, which is responsible for saving and loading tournament data. While the system handles basic data serialization and deserialization, it does not currently support variability in data formats. The system uses a single format for data persistence, so no variability is present in this aspect of the application.

Overall, the FTA exhibits minimal and largely static forms of variability. The system currently handles basic scenarios in a uniform way without providing significant flexibility or adaptability in terms of tournament structures, match result handling, team configurations, or data formats. Enhancing the system to incorporate more dynamic and automated variability mechanisms would greatly improve its capacity [2]. This potential for improvement will be discussed in greater detail in the next chapter.

Chapter 5

Possible variability extensions

To enhance the FTA and expand its utility across various contexts, several variability extensions can be introduced. These extensions would address the current limitations but also make the system more adaptable, flexible, and capable of meeting a broader range of user needs.

5.1 Generalization of tournament types

Currently, the FTA does not support multiple tournament structures. However, the system could be extended to include formats like single-elimination, double-elimination, and round-robin. Also, more complex structures. For instance, adding support for group stages followed by knockout rounds would enable users to model tournaments with preliminary rounds leading into elimination stages[1].

To achieve this, the system could introduce new classes or configuration options, allowing users to define custom tournament formats. These formats could be stored in configuration files or managed through a flexible interface.

Using the Strategy Pattern or Template Method Pattern would allow different tournament types to share common logic while customizing specific rules and progression paths. For instance, an abstract TournamentFormat class could define methods like initializeBracket(), advanceTeams(), and finalizeResults(). Subclasses such as SingleEliminationTournament, DoubleEliminationTournament, and GroupStageKnockoutTournament would implement the specific behaviors for each format [3].

Additionally, employing a Domain-Specific Language (DSL) could allow users to define tournament structures in a high-level, declarative manner. The DSL would enable tournament organizers to specify group sizes, advancement criteria, and tie-breaking rules, which the system would then parse to generate the appropriate structure dynamically [4]. This approach aligns with the principles of MDSE,

[1]A format common in football and many other sports.

where design patterns play an important role in managing variability [1].

5.2 Customizable match rules

The current system handles basic win/lose scenarios but lacks variability in match rules, such as custom scoring systems, time limits, or tie-breaking criteria. Adding configurable match rules would improve flexibility. Also, it could make the system applicable to a wider range of sports or user preferences.

This could be achieved by incorporating a rule engine or strategy objects that define various match scenarios. The Match.java class could be extended to support custom rules, with methods that can be overridden or selected based on the match type or user criteria. This would allow the system to adjust match outcomes and scoring dynamically.

Using the Strategy Pattern would encapsulate match rules in separate strategy objects. For instance, a MatchRules interface could define methods like calculateScore(), applyTimeLimit(), and determineOutcome(). Different implementations of this interface could be created for various sports or user-defined rules [3].

Additionally, integrating a rule engine like Drools or Jess could manage complex match rules. Rule engines separate business logic from the core application, which allows rules to be defined, modified, and managed independently [5]. Users could specify rules for scoring, fouls, penalties, and time limits, which the rule engine would apply during match execution.

5.3 Support for multiple sports

The FTA is currently designed specifically for football tournaments. Within this context it focus on managing teams, matches, and brackets.

To extend the system for use in multiple sports, the concept of a "team" and "match" would need to be abstracted to apply universally. This could involve creating abstract base classes or interfaces for Team, Match, and other key entities. These would then be extended or implemented for specific sports. Attributes and methods unique to football could be generalized or encapsulated in sport-specific subclasses. This would enable the system to manage tournaments for other sports, as well as customized rules and team configurations of each sport.

Aspect-Oriented Programming (AOP) could manage cross-cutting concerns that apply to multiple sports. For example, logging, security, and persistence. AOP separates these concerns from the core business logic. This allows for independent development and modification of sport-specific features without affecting shared system aspects [6].

5.4 Player statistics

Introducing player statistics would be a valuable extension to the FTA. Currently, the system focuses on team-level management. Adding the ability to display individual player statistics would significantly enhance its utility. This could involve creating new classes or extending existing ones to manage player statistics across matches and tournaments. The data could then be displayed in customizable views or reports.

To achieve this, a PlayerStatistics class could be introduced. This class would track various metrics like goals scored, assists, fouls, and playing time. It could be associated with the Player class, with methods to update and retrieve statistics during matches. Additionally, it would be beneficial to abstract different types of players, such as goalkeepers and forwards, since each position require tracking of different statistics. For example, a goalkeeper need metrics for saves and clean sheets, while a forward would focus on goals and assists.

The Observer Pattern could be employed to update player statistics in real-time as events occur. For instance, whenever a goal is scored, the PlayerStatistics class would be notified to update the relevant player's stats. This pattern ensures the system remains responsive and up-to-date without requiring manual intervention [3].

5.5 Enhanced data management

Data persistence is currently managed by the TournamentFileHandler.java class. While it supports basic serialization and deserialization, flexibility could be improved by adding support for various data storage backends.

Implementing a Data Access Layer (DAL) would abstract the storage mechanism, providing a unified interface for data operations. This approach is well-established in enterprise application architecture [7]. It allows users to choose their preferred storage option, with specific implementations handling interactions with different databases or cloud services.

Dependency Injection (DI) could be used to inject the selected storage backend at runtime. This can be based on user preferences or configuration files. DI frameworks like Spring could manage these dependencies, ensuring the system remains flexible and easy to extend [1].

5.6 User roles and permissions

Introducing user roles and permissions would enable different levels of access to tournament management features, enhancing both security and usability. Currently, the system does not differentiate between users.

A role-based access control (RBAC) system could be implemented to define roles such as admin, coach, or viewer, each with specific permissions. For example, an admin could have full access to create, modify, and delete tournaments, while a coach might only manage team configurations and match results. RBAC can be enforced using Access Control Lists (ACLs) or a Policy-Based Access Control (PBAC) framework. These frameworks ensure that user actions are appropriately restricted based on their roles [8].

5.7 Software product line vision

The FTA has the potential to evolve into a SPL capable of managing tournaments for various sports. By adopting a Feature-Oriented Programming (FOP) approach, the FTA can be structured to represent core features such as team management and match scheduling, while also incorporating optional and alternative features. These optional features could include support for different sports, customizable match rules, and flexible data storage solutions. The flexibility provided by FOP would ensure that the SPL could accommodate new sports and tournament types with minimal adjustments to the core system [1].

One of the key benefits of transforming the FTA into an SPL is the ability to generate tailored tournament management systems that align with specific user needs. For example, a user focused on organizing football tournaments might prioritize features like dynamic bracket generation and detailed player statistics, while another user organizing tennis tournaments might need different scoring systems and player ranking features. By using Feature Models and Diagrams, the system can visually represent the relationships between mandatory, optional, and alternative features. This visual representation simplifies the process of customization, enabling users to configure their system by selecting only the features they require, thereby reducing complexity and avoiding unnecessary functionality [4].

In addition to enhancing user experience, this SPL approach also provides significant benefits in terms of system maintainability and scalability. As new features or sports are introduced, they can be integrated into the SPL as additional modules or extensions, rather than requiring a complete overhaul of the existing system. This modularity ensures that the FTA remains flexible and adaptable, capable of evolving alongside the needs of its users. Furthermore, the use of feature models allows for clear documentation of the system's variability. Ultimately, making it easier for developers to understand the impact of changes and to manage the system over time [1] [2].

Chapter 6

Conclusion

This assignment provided an opportunity to apply software engineering principles within the context of a familiar system, the FTA. Through a detailed analysis of the system's current features, variability mechanisms, and potential extensions, it became evident that the FTA serves its primary function well. Despite this, it can still greatly benefit from enhancements in variability.

The exploration of possible extensions, such as the generalization of tournament types and customizable match rules, demonstrates the potential for the FTA to evolve into a more flexible and powerful tool. Moreover, adopting a SPL approach would position the FTA as a versatile platform capable of catering to a broader audience.

In conclusion, the proposed improvements and extensions are aligned with the principles of MDSE and SPLs. Implementing these changes would not only enhance the system's functionality but also ensure its scalability and relevance in diverse contexts.

Bibliography

[1] M. Brambilla, J. Cabot and M. Wimmer, *Model-Driven Software Engineering in Practice: Second Edition*, 2nd. Morgan & Claypool Publishers, 2017, ISBN: 1627057080.

[2] L. Montecchi, *Variability implementation*, Presentation slides, Presented at the TDT4250 - Model-Driven Software Engineering course, NTNU Trondheim, 2024.

[3] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1994.

[4] M. Fowler, *Domain-specific languages*. Addison-Wesley Professional, 2010.

[5] Red Hat, *Drools Documentation*. 2015, Accessed: 2024-08-29. [Online]. Available: https://docs.jboss.org/drools/release/6.2.0.Final/drools-docs/html_single/.

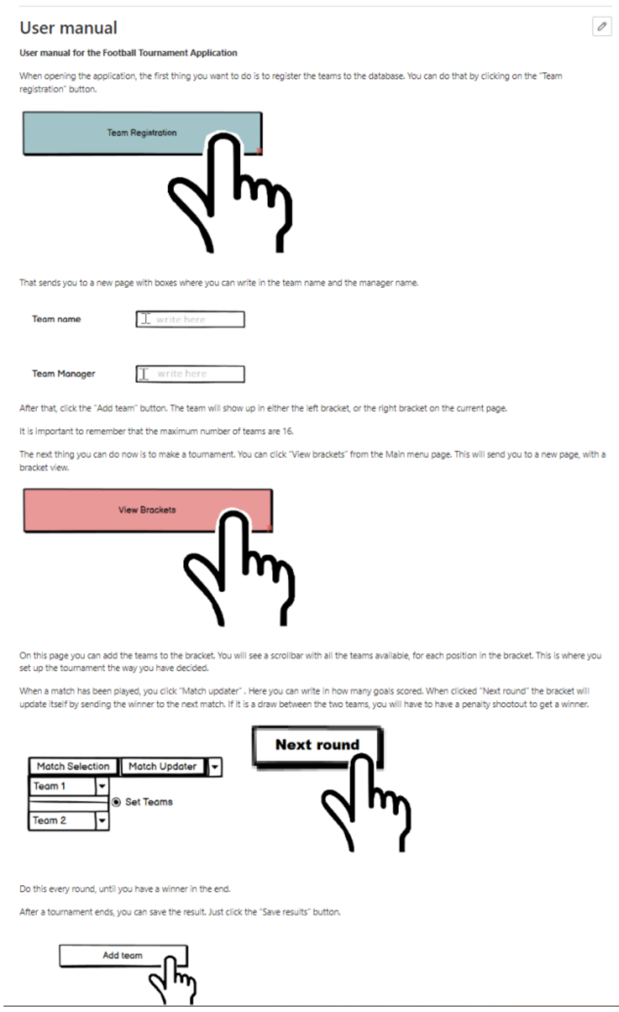
[6] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier and J. Irwin, 'Aspect-oriented programming,' in *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97)*, Berlin, Heidelberg: Springer-Verlag, 1997, pp. 220–242. DOI: 10.1007/BFb0053381.

[7] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[8] W. Stallings and L. Brown, *Computer Security: Principles and Practice*, 4th. Pearson, 2017.

Appendix A

User Manual



User manual

User manual for the Football Tournament Application

When opening the application, the first thing you want to do is to register the teams to the database. You can do that by clicking on the "Team registration" button.

That sends you to a new page with boxes where you can write in the team name and the manager name.

After that, click the "Add team" button. The team will show up in either the left bracket, or the right bracket on the current page.

It is important to remember that the maximum number of teams are 16.

The next thing you can do now is to make a tournament. You can click "View brackets" from the Main menu page. This will send you to a new page, with a bracket view.

On this page you can add the teams to the bracket. You will see a scrollbar with all the teams available, for each position in the bracket. This is where you set up the tournament the way you have decided.

When a match has been played, you click "Match updater" . Here you can write in how many goals scored. When clicked "Next round" the bracket will update itself by sending the winner to the next match. If it is a draw between the two teams, you will have to have a penalty shootout to get a winner.

Do this every round, until you have a winner in the end.

After a tournament ends, you can save the result. Just click the "Save results" button.