POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering 2
Design Document

Author(s): **Ballabio Giacomo - 10769576**
Benelle Francesco - 10727489
Cavallotti Alberto - 10721275

Academic Year: 2023-2024
Version: 1.0
Release date: 07/01/2024

Contents

1 | Introduction

1.1. Purpose

The purpose of this document is to present a detailed description of CodeKataBattle. It is addressed to the developers who have to implement the requirements and could be used as an agreement between the customer and the contractors.
The document is also intended to provide the customer with a clear and unambiguous description of the system's functionalities and constraints, allowing the customer to validate the requirements and to verify if the system meets the expectations.

1.2. Scope

CodeKataBattle (CKB) is a new platform that aims to help Students to improve their software development skills by participating in Tournaments and Battles, in which they will train on code katas, programming exercises that contain some test cases to be passed. Educators will create Tournaments and Battles in order to challenge the Students that will be asked to create groups and compete with their code.

1.3. Definition, Acronyms, Abbreviations

Acronyms	Definition
DD	Design Document
RASD	Requirements Analysis & Specification Document
ST	Student
ED	Educator
STG	Student Group
CKB	CodeKataBattle
GH	GitHub
User	All STs and EDs
API	Application Programming Interface
RX	Requirement X
CMP	Component

Table 1.1: Acronyms used in the document.

1.4. Revision History

Version 1.0 - 07/01/2024

1.5. Reference Documents

- Specification Document Assignment

1.6. Document Structure

The document is structured in seven sections, as described below.

Introduction. In the first section, the chapter elucidates the significance of the Design Document, providing comprehensive definitions and explanations of acronyms and abbreviations. Additionally, it recalled the scope of the CodeKateBattle system.

Architectural Design. The second section shows the main components of the system and their relationships. This section also focuses on design choices and architectural styles, patterns and paradigms.

User Interface Design. The next section, the third, describes the user interface of the

system, providing mockups and explanations of the main pages.

Requirements Traceability. The fourth section describes the requirements of the system, showing how they are satisfied by the design choices.

Implementation, Integration and test Plan. This fifth part provides an overview of the implementation of the various components of the system, it also shows how they are integrated and it gives a plan for testing them all.

Effort Spent. In the sixth section are included information about the number of hours each group member has worked for this document.

References. The last section contains the list of the documents used to redact this Design Document.

2 | Architectural Design

2.1. Overview

Here we represents an overview of how the entire CKB architecture is composed of:

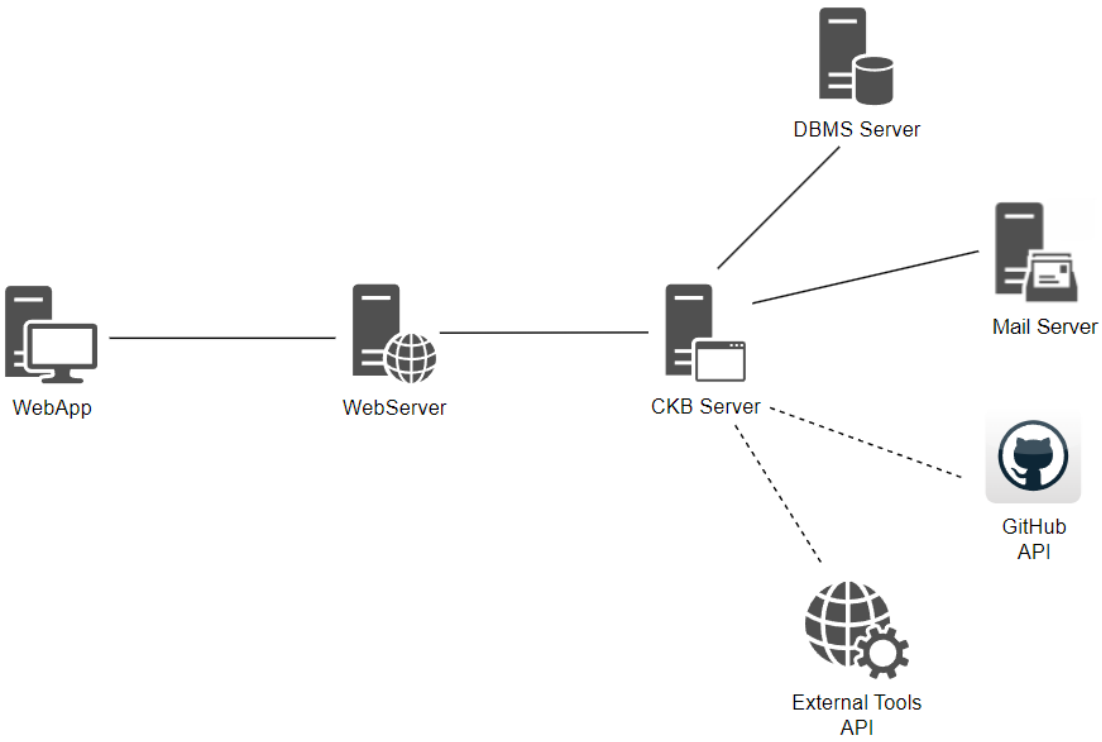


Figure 2.1: CKB Overview.

Client side:

- WebApp: serves as the User interface, allowing all Users to connect to CKB. It enables Users to perform operations such as registration, login, creating or joining Tournaments and Battles, creating or modifying Badges and searching for other Users.

Server side:

- **Web Server:** handles communication with Users, receiving and processing their inputs. Additionally, it provides load balancing for requests, distributing them among various replicas of the CKB Server.
- **CKB Server:** is the central component where interfaces are located, facilitating communication between the Web Server and databases/APIs. It serves as the primary server for the entire website and is replicated across multiple machines to handle a high volume of requests.
- **DBMS Server:** stores data related to Users, Tournaments, and Battles. It acts as the repository for essential information.
- **Mail Server:** is responsible for sending confirmation eMails when a new User registers on CKB, enhancing the User registration process.
- **GitHub API:** is utilized for communication with GitHub, facilitating the creation of the Battle repository and allowing STGs to fork the repository to push their code.
- **External Tools API:** used to automatically test the STG code when a new push is made. It is also used to retrieve the results of the tests and update the Battle dashboard.

2.2. Component View

2.2.1. High Level Diagram

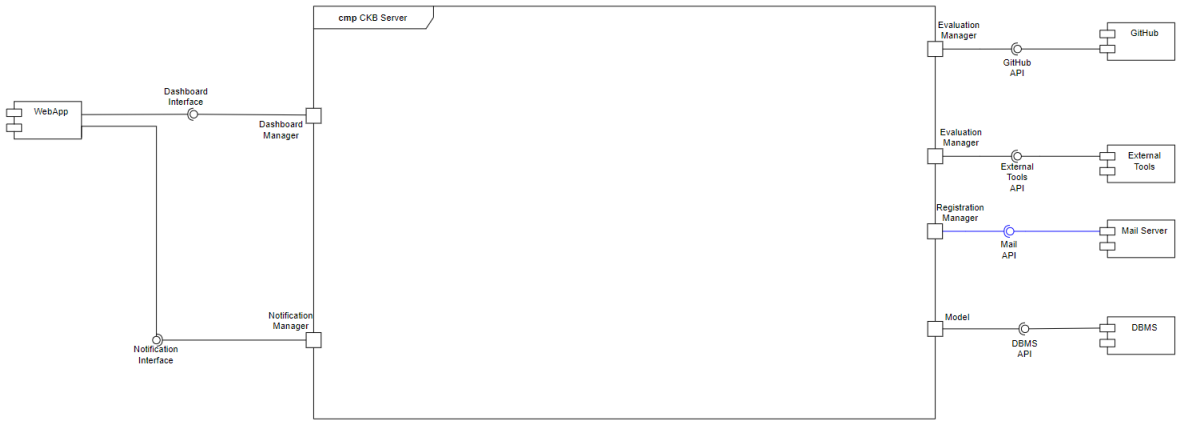


Figure 2.2: High Level Diagram.

In the figure above is the high level component diagram of CKB where it's represented the external components of CKB and how they communicate with the CKB server, in particular:

- **WebApp**: serves as the external access point for Users, allowing communication with the CKB Server through the Dashboard Interface—the sole means for Client-Server interaction from the User side. The CKB Server can relay notifications, such as Tournament or Battle creation, to Users through the Notification Interface.
- **DBMS:** is the storage repository for all User, Tournament, and Battle data. It communicates with the CKB Server via the DBMS API, which is connected to the Model component.
- **Mail Server:** responsible for sending registration confirmation eMails, the Mail Server communicates with the CKB Server using the Mail API interface. This interface is linked to the Registration Manager component, which oversees the User registration process..
- **External Tools:** external application used for testing the code submitted by STGs on GitHub. It communicates with the CKB Server through the External Tools API, connecting to the Evaluation Manager component. The Evaluation Manager handles the evaluation process for STG-submitted code.
- **GitHub:** external website used to create repositories for the code katas of Battles. Each STG, after forking the main repository, pushes their code for evaluation. GitHub communicates with the CKB Server through the GitHub API, linked to the Evaluation Manager.

2.2.2. Low Level Diagram

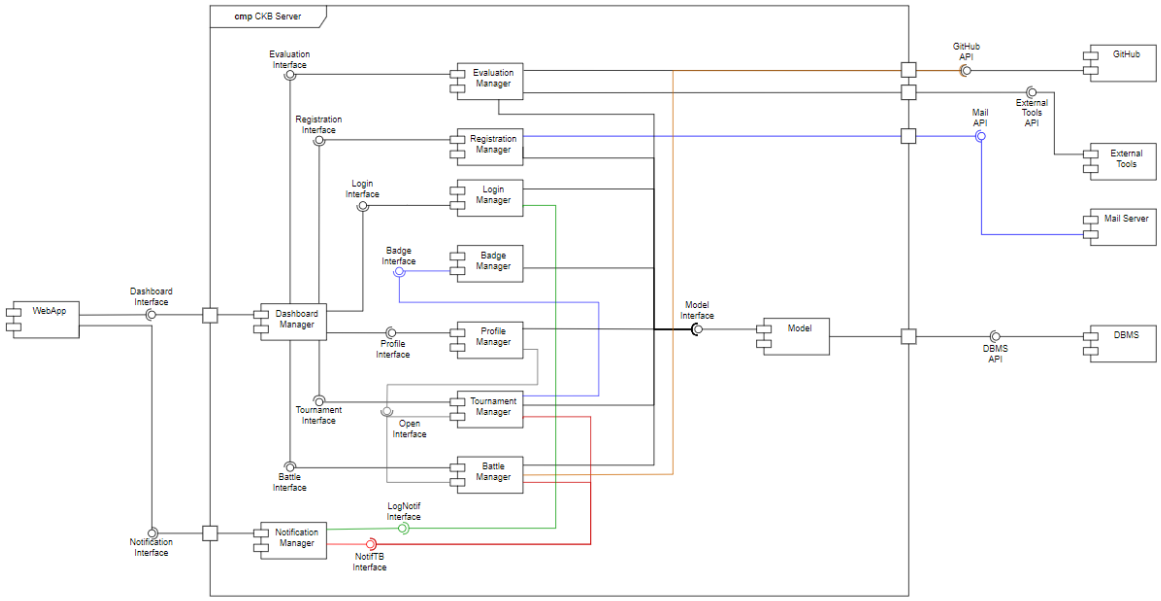


Figure 2.3: Low Level Diagram.

The figure above represents the complete architecture of CKB website with each components inside the CKB Server:

- **Dashboard Manager:** pivotal component that orchestrates all communication between users and the CKB website. Users interact with CKB through the Dashboard Interface, and the Dashboard Manager directs requests to the appropriate components. It serves as the central hub for user interactions.
- **Model:** This high-level component represents the data on the server and acts as an interface to the database server. It acts as a mask to the database server, and every component needs to interface with it to access data from the DBMS through the DBMS API.
- **Evaluation Manager:** component that handles the code evaluation both when a new push is made by a STG on GH or when an ED wants to manually evaluate a STG code during the consolidation stage of a Battle, more detailed information in the following section in Figure 2.4. This component communicates with the Dashboard manager through the Evaluation Interface, with the Model component through the Model Interface to add and modify the STG evaluation in the DBMS, with GitHub through the GitHub API when a new push is made by a STG and

with the External Tools through the External Tools API to automatically evaluate a STG code.

- **Registration Manager:** component that handles the registration of a new User. When a new User wants to create an account on CKB system it communicates with the Dashboard Manager that forwards the request to the Registration Manager through the registration interface. Then the Registration Manager handles the request and communicates through the Mail API to the Mail Server, to send a confirmation mail to the new User, and through the Model Interface to the Model component to add the new User's information to the DBMS. This component also gives the permission to the User that registered as an Educator to create Tournaments, Battles and Badges.
- **Login Manager:** component that handles the login process for registered Users. When a User attempts to log in, the Dashboard Manager forwards the request to the Login Manager through the Login Interface. The Login Manager communicates with the Model component through the Model interface to retrieve the User's data from the DBMS.
- **Badge Manager:** component that handles the Badges creation and modification when a new Tournament is created. When the ED creates a new Tournament, the Badge Manager component receives a request from the Tournament Manager through the badge interface and lets the ED create new Badges or modify existing ones, more details in Figure 2.5. The new Badges are added in the DBMS through the communication between the Badge Manager and the Model component through the Model Interface.
- **Profile Manager:** component that allows User profile search and profile open in both Tournament and Battle dashboard. When a User initiates a search, the Dashboard Manager forwards the request to the Profile Manager using the Profile Interface. The Profile Manager communicates with the Model component through the Model Interface to retrieve relevant information from the DBMS.This component also manages the open profile operation within a Tournament or Battle dashboard, more detail in Figure 2.6.
- **Tournament Manager:** component that manages all user actions related to Tournaments; more detailed information in Figure 2.7. It communicates with the Dashboard Manager through the Tournament Interface, with the Notification Manager through the NotifTB Interface when a new Tournament is created and with the Model component through the Model Interface to add or retrieve data from the

DBMS.

- **Battle Manager:** component that manages all user actions related to Battles; more detailed information in Figure 2.8. It communicates with the Dashboard Manager through the battle interface, with the Notification Manager through the NotifTB Interface when a new Battle is created or when a ST invites other STs to join his STG and with the Model component through the Model Interface to add or retrieve data from the DBMS. It also communicates with GitHub through the GitHub API to create repositories and upload code katas for new Battles.
- **Notification Manager:** component that handles each notification that has to be sent to the Users in particular when a new Tournament is created it sends a notification to all STs registered in CKB, when a new Battle is created it sends a notification to all the STs that have joined that specific Tournament, when a ST invites another ST to join his STG for a battle a notification is sent to the second ST and when a Tournaments is closed and the score are updated it sends a notification to all the STs that have joined the Tournament. All the communication from the Battle Manager and the Tournament Manager with the Notification manager is made through the NotifTB interface and the communication with the WebApp is made through the Notification Interface.

2.2.3. Evaluation Manager

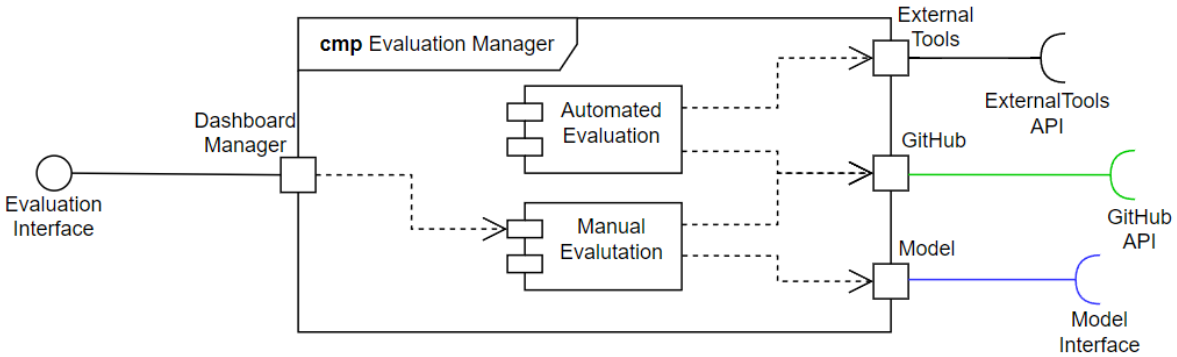


Figure 2.4: Evaluation Manager.

The Evaluation Manager is composed by two other sub-components that handles the two different evaluation method of a STG code:

- **Automated Evaluation component:** utilized by the CKB system to automati-

cally evaluate STG code whenever a new push is made to the STG's forked repository on GitHub. When a code update occurs, GitHub communicates with the Automated Evaluation component through the GitHub API. Subsequently, the Automated Evaluation component sends the code to External Tools via the External Tools API, where the code undergoes testing and evaluation. Upon receiving the evaluation results, the Automated Evaluation component, through the Model interface, communicates with the Model component. The Model component utilizes the DBMS API to update the new score in the relevant DBMS section associated with the corresponding Battle.

- **Manual Evaluation component:** comes into play when an ED wishes to manually assess an STG code. The process begins with the WebApp, which, through the Dashboard Interface, requests the STG code from the Dashboard Manager. The Dashboard Manager then communicates this request to the Manual Evaluation component through the Evaluation Interface. Subsequently, the Manual Evaluation component communicates with the GitHub API to retrieve the source code from the STG's forked repository, allowing the ED to analyze it. Once the evaluation is complete and the ED decides to update the score, the Manual Evaluation component, through the Model Interface, communicates with the Model component. The Model component, utilizing the DBMS API, updates the score in the DBMS, ensuring the manual evaluation results are recorded appropriately.

2.2.4. Badge Manager

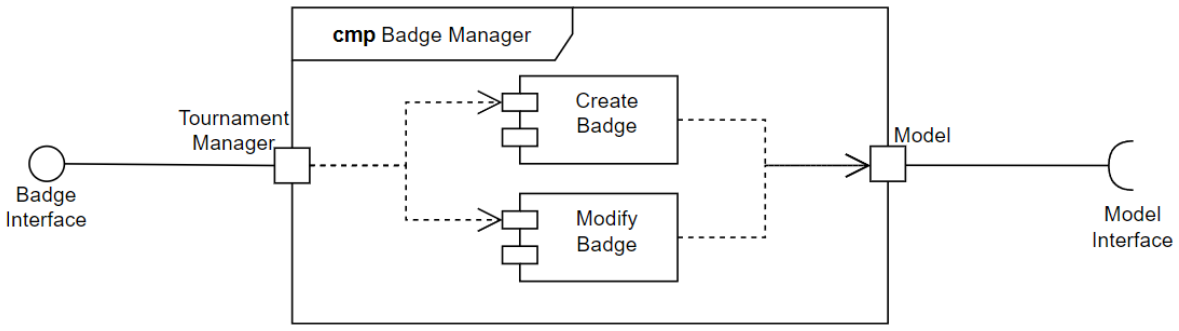


Figure 2.5: Badge Manager.

The Badge Manager is the component used by the CKB system to handle the creation and the modification of the Badges during the Tournament creation:

- **Create Badge component:** activated when an ED intends to create a new Badge for a newly generated Tournament. The initiation of this process is through the Create Tournament component, a sub-component of the Tournament Manager. The Create Tournament component, via the Badge Interface, sends a request to the Create Badge component, allowing the ED to define the Badge with its specific settings and parameters, such as the criteria STs must fulfill to obtain it. Following the Badge creation, the Create Badge component, through the Model Interface, communicates with the Model component, ensuring the newly created Badge is added to the DBMS.
- **Modify Badge component:** engaged when an ED aims to modify an existing Badge that was previously created for another Tournament. The process is initiated by the Create Tournament component, a sub-component of the Tournament Manager. The Create Tournament component, via the Badge Interface, sends a request to the Modify Badge component, allowing the ED to adjust parameters associated with an existing Badge, specifying new criteria for STs to fulfill. Once the Badge is successfully modified, the Modify Badge component, through the Model Interface, communicates with the Model component. This communication ensures that the updated Badge information is reflected in the DBMS.

2.2.5. Profile Manager

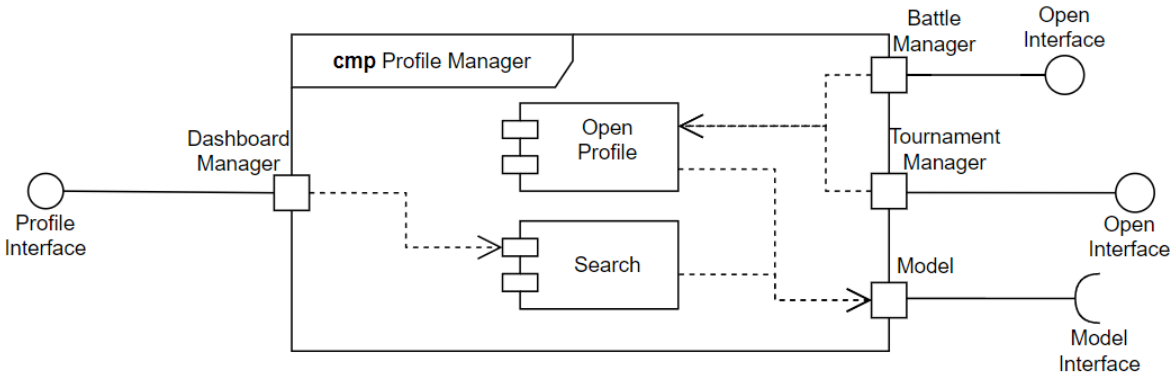


Figure 2.6: Profile Manager.

The Badge Manager is the component used by the CKB system to handle the research of a User's profile and the visualization of another User's profile when the User clicks on a nickname within the Tournament or Battle dashboard:

- **Search component:** responsible for managing the profile search process when a User enters a nickname or keyword into the search bar across various CKB pages. When a User initiates a search by entering another User's nickname or a relevant keyword, the Dashboard Manager communicates with the Search component through the Profile Interface. The Search component then forwards the search request to the Model component via the Model Interface. Subsequently, the Model component retrieves the profile information from the DBMS. The retrieved information is then presented to the User, allowing them to visualize the searched User's profile.
- **Open Profile component:** manages the retrieval of a User's profile when a User clicks on a nickname within a Tournament or Battle dashboard. When a User clicks on another User's nickname in a Tournament or Battle dashboard, the Dashboard Manager communicates with the View component within the Tournament or Battle Manager. The View component forwards the request to the Open Profile component through the Open Interface. The Open Profile component, in turn, communicates with the Model component via the Model Interface. This communication with the Model component facilitates the retrieval of the User's profile information from the DBMS. The retrieved profile information is then presented to the User, allowing them to view the selected User's profile.

2.2.6. Tournament Manager

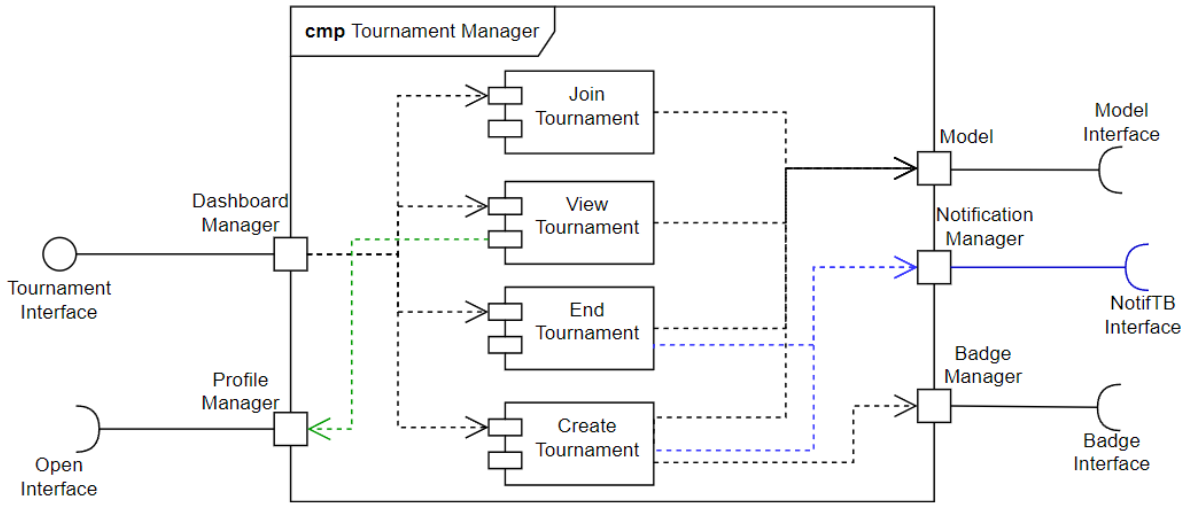


Figure 2.7: Tournament Manager.

The Tournament Manager is the component used by the CKB system to handle every aspect of the Tournament, from the creation to the end going through the Join and the View sub-components:

- **Create Tournament component:** utilized when an ED initiates the creation of a new Tournament. The ED communicates with the Dashboard Manager through the Dashboard Interface, which then directs the request to the Create Tournament component. This component sends back a creation form for the ED to fill. Once completed, the Create Tournament component communicates the data to the Model component through the Model Interface, adding the information to the DBMS via the DBMS API. If the ED grants permissions to other EDs, the Create Tournament component communicates with the Notification Manager through the NotifTB Interface to notify the specified EDs. Additionally, notifications are sent to all registered STs via the Notification Manager, informing them of the new tournament. Throughout the tournament creation process, the Create Tournament component also communicates with the Badge Manager through the Badge Interface, allowing the ED to create or modify Badges associated with the tournament.
- **Join Tournament component:** activated when an ST wishes to join a tournament. The ST communicates with the Dashboard Manager through the Dashboard Interface, and the request is forwarded to the Join Tournament component. This component communicates through the Model Interface with the Model component to add the ST to the participant list in the DBMS through the DBMS API.
- **View Tournament component:** used by the CKB system to let the User visualize the Tournament page with all the information, like the available Battles and the Dashboard with the STs score . When a User wants to search a Tournament it writes the Tournament name or a keyword of it in search bar and it communicates with the Dashboard Manager through the Dashboard Interface that forwards the request to the View Tournament component that communicates with the Model component through the model interface to retrieve all the information from the DBMS through the DBMS API and let the User visualize the Tournament page. the same communication is made when a User clicks on a Tournament name in another User's profile or in his main dashboard page. This component also manages the open profile operation when a User wants to visualize another User's profile from the Tournament dashboard; when a User clicks on another User's nickname in the Tournament dashboard the Dashboard Manager communicates through the Dashboard Interface with the View Tournament component that forwards the request to the the Profile Manager through the Open Interface.

- **End Tournament component:** triggered when an ED decides to close a Tournament, preventing further ST participation and ED creation of Battles within it. The ED communicates with the Dashboard Manager through the Dashboard Interface, which forwards the request to the Close Tournament component. The Close Tournament component communicates with the Model component through the Model Interface, modifying the Tournament's status to make it non-joinable in the DBMS through the DBMS API. Additionally, the Close Tournament component communicates through the NotifTB Interface to the Notification Manager, which sends notifications to all STs who participated in the Tournament, informing them that final scores are ready for viewing.

2.2.7. Battle Manager

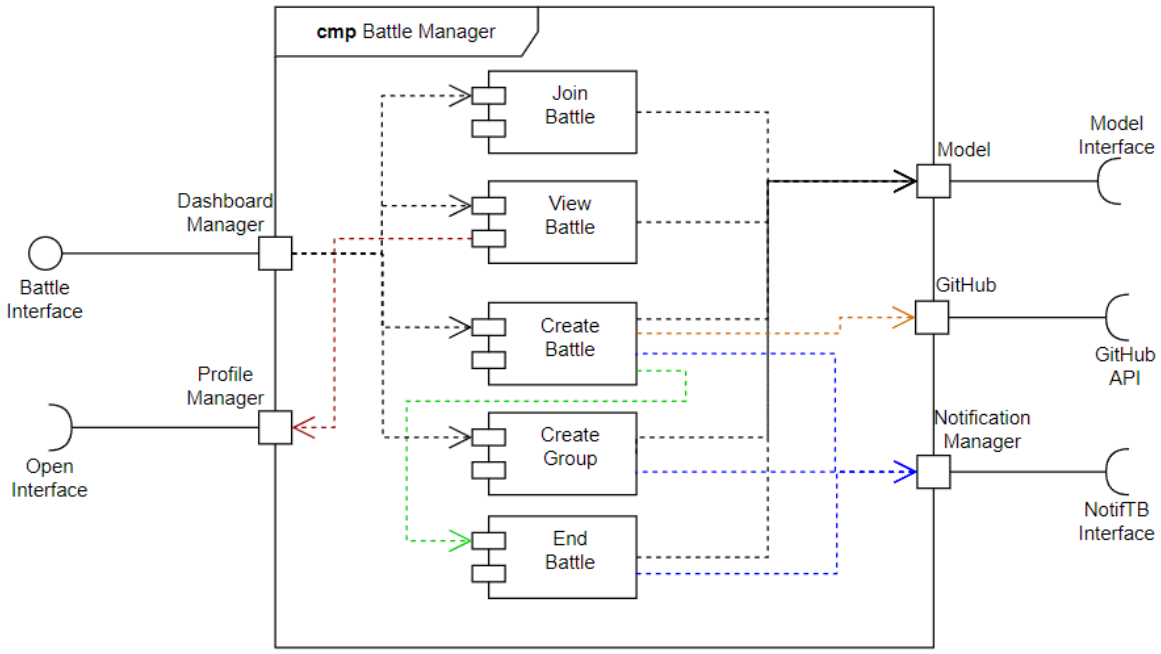


Figure 2.8: Battle Manager.

The Battle Manager is the component used by the CKB system to handle every aspect of the Battle, from the creation to the end going through the Join, the View and the Create Group sub-components:

- **Create Battle component:** engaged when an ED wishes to create a new Battle within a Tournament. The ED communicates with the Dashboard Manager through

the Dashboard Interface, forwarding the request to the Create Battle component. The Create Battle component responds by sending a creation form to be filled by the ED. Upon completion, the component communicates the data to the Model component through the Model Interface, adding the information to the DBMS through the DBMS API. Additionally, the Create Battle component communicates with the Notification Manager through the NotifTB Interface, notifying all STs who have joined the relevant Tournament about the new Battle. It also collaborates with the End Battle component to create a timer, ensuring that after the consolidation stage concludes, the End Tournament component notifies all STs about the availability of final grades. The Create Battle component also communicates with GitHub through the GitHub API to create a new repository and upload the code kata for the battle. This repository is later forked by all STGs to submit their code.

- **Join Battle component:** activated when an ST intends to join a Battle. The ST communicates with the Dashboard Manager through the Dashboard Interface, and the request is forwarded to the Join Battle component. The Join Battle component, through the Model Interface, communicates with the Model component, adding the ST to the participant list of the battle in the DBMS through the DBMS API.
- **View Battle component:** used by the CKB system to let the User visualize the Battle page with the dashboard including all the STGs score. When a User wants to visualize the Battle page it communicates with the Dashboard Manager through the Dashboard Interface that forwards the request to the View Battle component that communicates with the Model component through the Model Interface to retrieve all the information from the DBMS through the DBMS API and finally let the User visualize the Battle page. This component also manages the open profile operation when a User wants to visualize another User's profile from the Battle dashboard; when a User clicks on another User's nickname in the Battle dashboard the Dashboard Manager communicates through the Dashboard Interface with the View Tournament component that forwards the request to the the Profile Manager through the Open Interface.
- **Create Group component** engaged when STs want to create a new STG for a Battle. After joining a Battle before the registration deadline expires, an ST can create an STG to participate in the battle. The ST communicates with the Dashboard Manager through the Dashboard Interface, initiating a request to create a new STG. The request is forwarded to the Create Group component through the Battle Interface, allowing the ST to decide the STG name and invite other STs. Notifications are sent through the Notification Manager, which communicates with

the Battle Manager through the NotifTB interface and with the WebApp through the notification interface. When the STG is confirmed, the Create Group component communicates through the Model Interface with the Model component to add the newly created STG to the DBMS through the DBMS API.

- **End Battle component:** activated to notify all STGs that the final scores of the battle are available. When the consolidation stage concludes, and the timer created by the Create Battle component expires, the End Battle component communicates through the Model Interface with the Model component, updating the scores in the DBMS through the DBMS API. It also notifies all participating STs through the Notification Manager, communicating through the NotifTB Interface, that the final scores are accessible on the Battle page. Communication between the Notification Manager and the WebApp is facilitated through the Notification Interface.

2.3. Deployment View

2.4. Runtime View

2.5. Component Interfaces

2.6. Selected Architectural Styles and Patterns

CodeKataBattle will be developed over a **3-Tier architecture**, which is a software application architecture that organizes applications into three logical and physical computing tiers. Each tier runs on its own infrastructure, so the complexity of the system is reduced, its flexibility and scalability is enhanced. Each tier may be developed simultaneously by separate teams of developers and can be updated or scaled as needed without impacting the other tiers.
The system is modularized over three independent tiers:

- Presentation Tier: this is the top-level tier, that directly interacts with the users. Its goal will be to collect the users' inputs and show them the outputs produced by the lower tiers.

- Application Tier: this is the middle level tier. The application tier processes the users' requests that arrive from the presentation tier, perform the necessary computation, even retrieving or storing data on the Data tier, and computes that data in order to complete specific tasks requested by the users. This tier needs to handle

several requests by different users simultaneously, while guaranteeing the security and integrity of the stored data.

- Data Tier: this is the lowest level tier. The Data tier stores, retrieves and manages data used or produced by the application tier by the DataBase System's APIs.

For a more detailed description of the tiers and their components, please refer to *section 2.3*.
The behavior of the system will be mostly as a **Client-Server architecture**, in which the Client represents the front-end user interface, as it is the connection point between the final users of the system and the system itself, meanwhile the Server represents the backend platform, as it elaborates the users' requests, computes and shows the answers. In a pure Client-Server architecture, the server is always the one to make the request, while the server is a passive element, which handles and elaborates the requests of the client and returns the answers to them. In some cases, as when the process of sending the Notifications or the interaction with the external tools, the server needs to perform operations without being invoked by the client first, which makes it an active component and having a moer Event Driven behavior.
The Client-Server interactions are performed through **REST APIs**, a set of commands commonly used in the context of web transactions. The *REpresentational State Transfer* consists in the use of a stateless Server, that means that the Client is the component that needs to communicate the state of the transaction to the Server. This approach allows the Server to be more scalable and helps with handling many requests that concern different states simultaneously. The components communicate by transferring a representation of a resource in a format that the server is able to process, this allows the usage of HTTP protocol to share data and to encode those using the JSON format.
The implementation of CKB will follow the **Model-View-Controller** pattern, which is a software design pattern that splits the software into three elements interconnected with each other:

- Model: contains the methods that manage the data. It provides methods for saving, retrieving and manipulating data from the database.
- View: the view is the part that concerns the visual representation of the data for the final user.
- Controller: acts as a connection point between the view and the model. The controller handles the users' interactions with the view and processes the operations.

2.7. Other Design Decisions

2.7.1. Availability

The introduction of load balancing and replication mechanisms significantly enhances the reliability and availability of our system. Load balancing optimizes resource utilization by distributing requests evenly across servers, preventing performance bottlenecks. Concurrently, replication ensures fault tolerance by duplicating essential data and services. This redundancy minimizes the impact of potential failures, fortifying our system to maintain consistent data management and service availability even in challenging scenarios.

2.7.2. Scalability

Microservices architecture, at its core, is meticulously crafted to be both independently deployable and inherently scalable. This design philosophy enables each microservice to be deployed autonomously, facilitating swift updates and modifications without disrupting the entire system. Beyond this, the scalability aspect of microservices empowers our system to gracefully handle increased demands by efficiently scaling specific services.

2.7.3. Notification Handling

Notification management plays a pivotal role in the CKB system, influencing every stage from Tournament, Battle or STG creation to their conclusion. Notifications in CKB are orchestrated to reach users upon login or, if a user is already logged in, immediately when they become available. This approach guarantees that users are promptly informed about key events, such as Tournament, Battle or STG creation, maximizing user engagement and system transparency.

2.7.4. Ease of Deployment

The adoption of microservices architecture introduces a notable advantage in ease of deployment. This methodology empowers the implementation of changes to individual services independently, allowing for seamless deployment without impacting the entire system. In the event of issues, the microservices approach facilitates swift identification, isolation, and correction, eliminating the need to halt the entire system. This deployment flexibility not only accelerates the release of updates but also enhances the system's resilience and agility, enabling efficient troubleshooting and maintenance processes.

2.7.5. Data Storage

To enhance operational efficiency and simplify data management, we've chosen a unified DBMS containing information about Users, Tournaments, and Battles. This approach reduces the time and complexity associated with retrieving and updating data, leveraging interconnected relationships among these components.

3 | User Interface Design

4 | Requirements Traceability

LoginManager:

- R2: CKB allows registered EDs to login
- R3: CKB allows registered STs to login

RegistrationManager:

- R1: CKB allows unregistered Users to sign up
- R46: CKB shall communicate with the mailing system in order to allow Users to register their account

CreateTournament:

- R4: CKB allows EDs to create Tournaments
- R5: CKB allows EDs to grant the permissions of a Tournament to other EDs
- R15: CKB allows EDs to choose which Badges to award in a certain Tournament

CreateBattle:

- R6: CKB allows EDs to create Battles
- R7: CKB allows EDs to uploads the code kata of a Battle
- R8: CKB allows EDs to set the minimum and the maximum number of STs per group of a Battle
- R9: CKB allows EDs to set a registration deadline of a Battle
- R10: CKB allows EDs to set a submission deadline of a Battle
- R11: CKB allows EDs to set additional configuration for the scoring system of a Battle
- R12: CKB allows EDs to set functional aspects for the scoring system of a Battle

- R30: CKB creates a GH repository of the code kata when the registration deadline for the Battle expires
- R31: CKB sends the link of the GH repository to every STG that participates in the Battle

CreateGroup:

- R22: CKB allows STs to create a new STG
- R23: CKB allows STs to join a STG
- R24:CKB allows STs to invite other STs in their STG

OpenProfile:

- R18: CKB allows EDs to visualize the profile of another User
- R19: CKB allows STs to visualize the profile of another User

JoinTournament:

- R20: CKB allows STs to join a Tournament

ViewTournament:

- R41: CKB allows STs to check the Leaderboard of a Tournament
- R42: CKB allows EDs to check the Leaderboard of a Tournament

CloseTournament:

- R17: CKB allows EDs to close a Tournament
- R49: CKB shall assign the Badges to all STs that fulfill their requirements

CreateBadge:

- R13: CKB allows EDs to create new Badges

ModifyBadge:

- R14: CKB allows EDs to choose the rules related to the awarding of Badges

JoinBattle:

- R21: CKB allows STs to join a Battle
- R47: STs need to fork the GH repository of the Battle they are participating in

ViewBattle:

- R35: CKB allows STs to check the Leaderboard of a Battle
- R36: CKB allows EDs to check the Leaderboard of a Battle

EndBattle:

- R16: CKB allows EDs to assign a score manually during the consolidation stage

ManualEvaluation:

- R16: CKB allows EDs to assign a score manually during the consolidation stage
- R37: CKB allows EDs to analyze the code of a STG

AutomatedEvaluation:

- R32: CKB evaluates the STG's work every time a push is made on GH and calculates Battle score for the STG
- R33: CKB updates the Battle Leaderboard once a new score is registered
- R34: CKB updates the Tournament Leaderboard once a new score is registered
- R44: CKB shall communicate with the GH API in order to calculate a new score every time a push action is made by a STG
- R45: CKB shall communicate with the external tool in order to calculate the score of a STG

Model:

- R25: CKB stores the informations about the Users
- R26: CKB shall ensure security of data

DashBoardManager:

- R39: CKB allows STs to check the list of ongoing Tournaments
- R40: CKB allows EDs to check the list of ongoing Tournaments

NotificationManager:

- R27: CKB sends notifications to every ST when a new Tournament is created
- R28: CKB sends notifications when a new Battle is created to every ST which is participating in the Tournament that the Battle is part of
- R29: CKB sends notifications to a ST when he receives an invitation to be part of STG
- R38: CKB sends notifications to every STs participating in the Battle once the consolidation stage ends
- R43: CKB sends notifications to every ST involved in a Tournament when the Tournament is closed and the final ranks are available
- R50: CKB sends notifications to the ED when he receives the permission to create Battles in a Tournament

5 | Implementation, Integration and Test Plan

5.1. Overview and Implementation Plan

In this last chapter, it will be described the Implementation of the system, the Integration and the Test Strategy that has to be followed. In general, the method that will be followed is a Bottom-Up strategy.
By adopting this strategy, the implementation will start from the leaves of the 'uses' hierarchy, starting from the small functionalities that do not require other functionalities to work. These modules will require a Driver each that will be developed in order to be tested. Once a new module is developed and tested, it may be integrated into the system and replace a previously existing Driver, but the new module will require a new Driver in order to be tested. In this way several working subsystems are created, which will be eventually integrated into the final one. Bottom-Up strategy promotes an incremental integration, that makes it easier to track bugs and errors, given that the testing is going to be done on a reduced part of the system at the beginning and on every module once it is ready. This strategy also allows independent development teams to work in parallel on different functions.

5.2. Features Identification

[F1] Login and Registration Features. These are the basic features of CKB, that will be needed by every User, both Educators and Students. Though this set of features will require the least amount of time to be implemented, its role will be crucial to the proper functioning of the entire web app. Since they are required for the correct workflow of the following features, they will also be the first to be implemented.

[F2] Creation Features. This set of features includes every creation feature, such as the creation of Tournaments, Battles, Groups or Badge, whatever represents the cre-

ation of a new Bean or a write operation on the database. These features are required for the following features, since without them it wouldn't be possible to visualize a Battle or Tournament.

[F3] View Features. These features include the possibility to open the page of a Tournament, a Battle, a User Profile, or the Home Page. They need the correspondent F2 feature in order to work and are essential for the Search and Join Features.

[F4] Search Features. These features include the use of the search bar on the website in order to retrieve the list of the ongoing Tournaments, Battles or Users. These features require that F2 and F3 are implemented but aren't needed by other features.

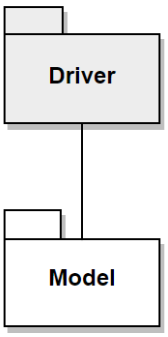
[F5] Join Features. Include the operations that permit joining a Tournament, a Battle, or a Group. This set of features need the view and creation features to exist, before getting implemented.

[F6] Evaluation Features. Features that include Manual and Automated Evaluation using the External Tools.

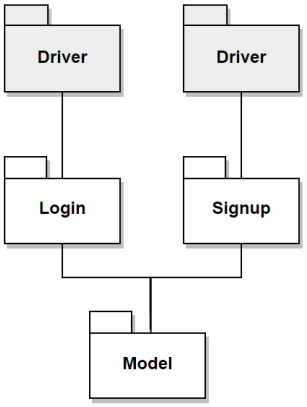
[F7] Notification Features. This is the last possible set of features to be developed, since the proper functioning of this feature requires that every other kind of feature properly functions too.

5.3. Integration Strategy

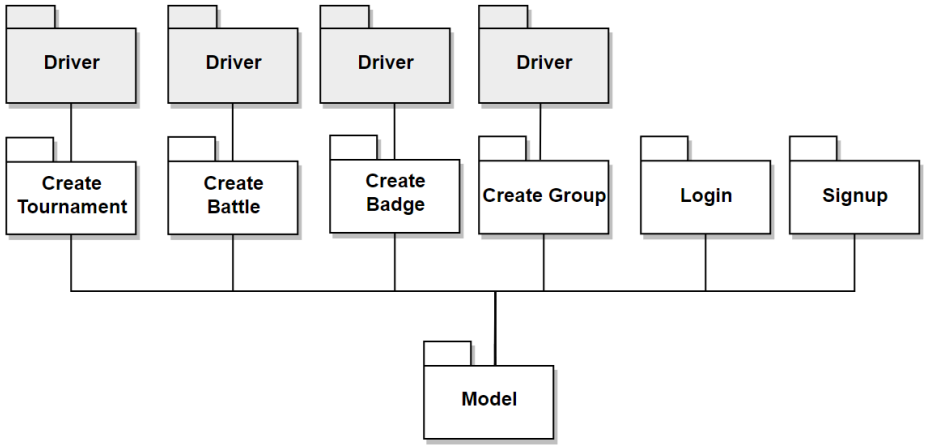
The integration of components and the testing of the system should start as soon as the DBMS and the host server are ready. The connections with the Mailing System, GitHub and the External Tools are not required since the starting moment, but will be necessary once the corresponding features will be integrated. As explained before the integration will follow a bottom-up approach.
Starting from the model, which will be tested alongside a proper driver,



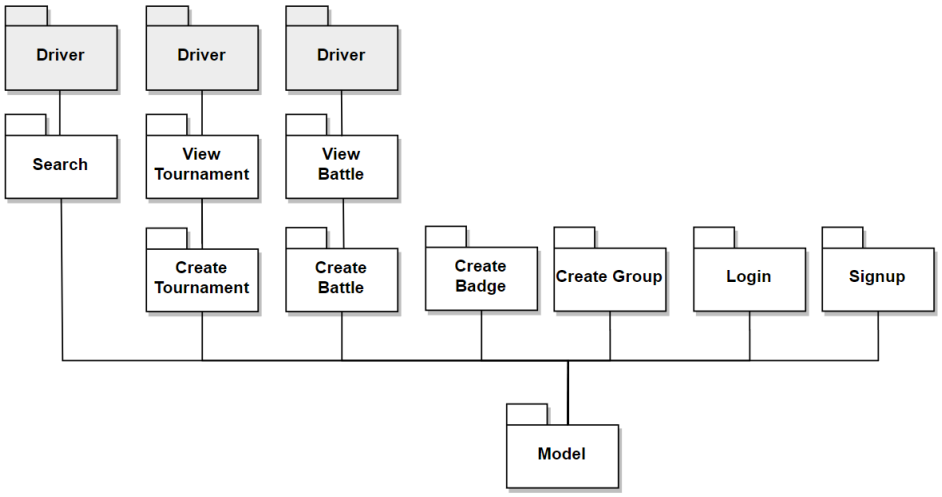
the integration of components will proceed with the Login and Registration features:



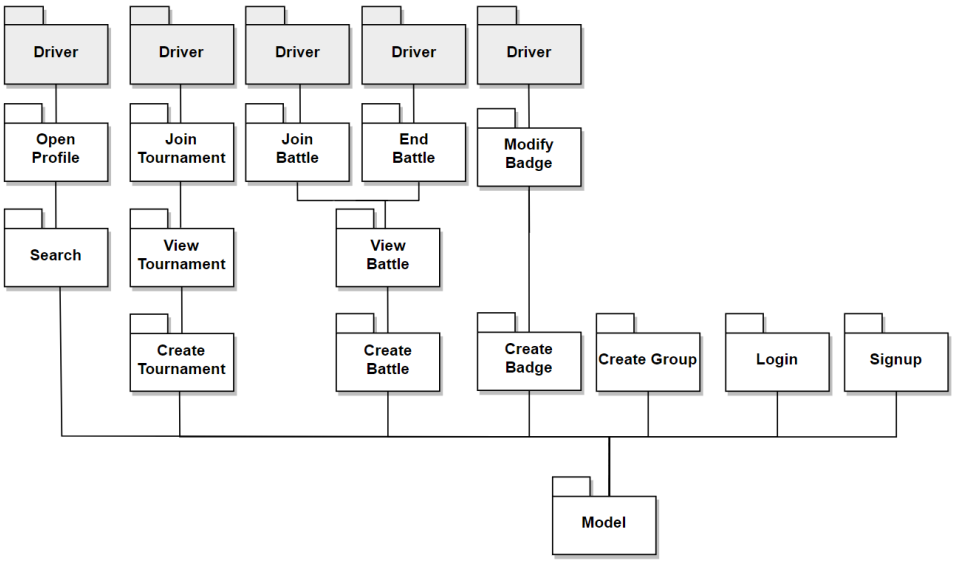
Once they will be developed and tested, it will be the turn of the components that perform a creation feature and their drivers:



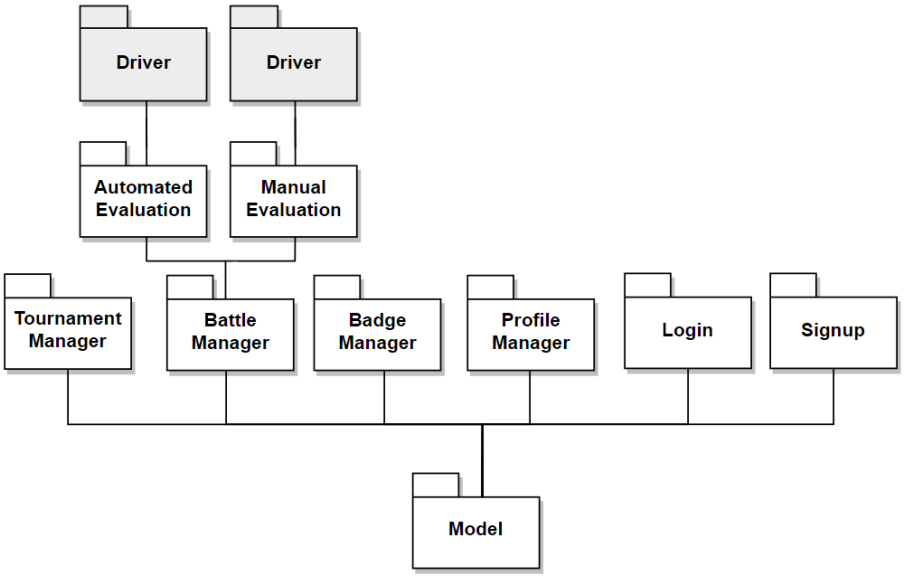
Now it is possible to create tournaments, battles, groups and badges functions to view those elements' pages or the search components can be integrated.



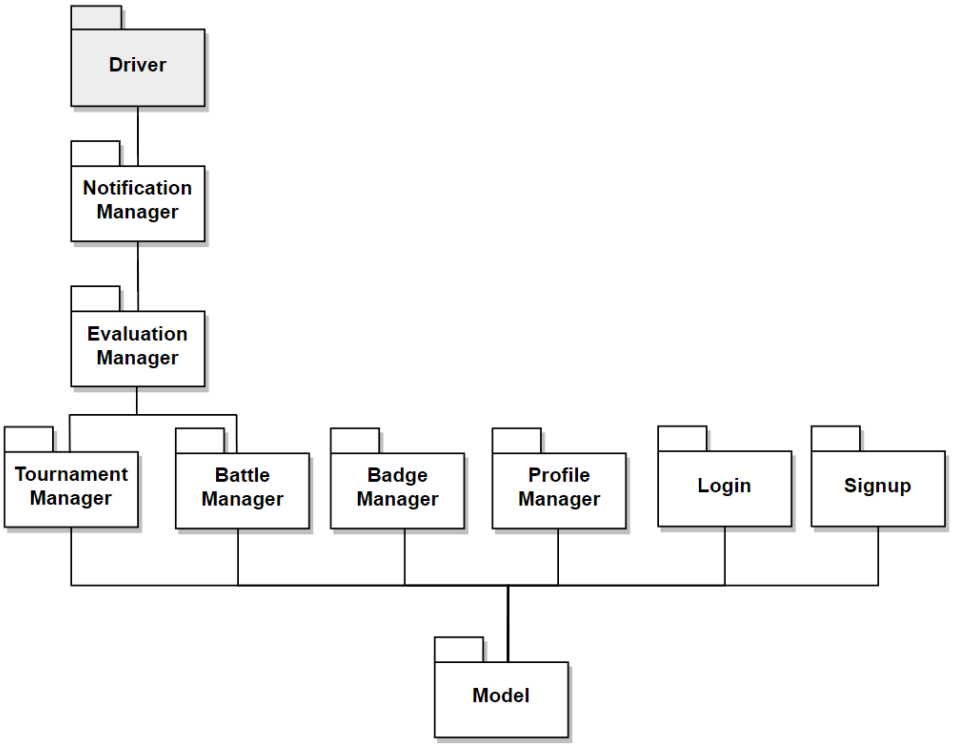
The last missing components in the context of tournaments and battles are the Join features, which will be integrated in parallel with the possibility to open users' profiles and to modify badges parameters for the Educators.



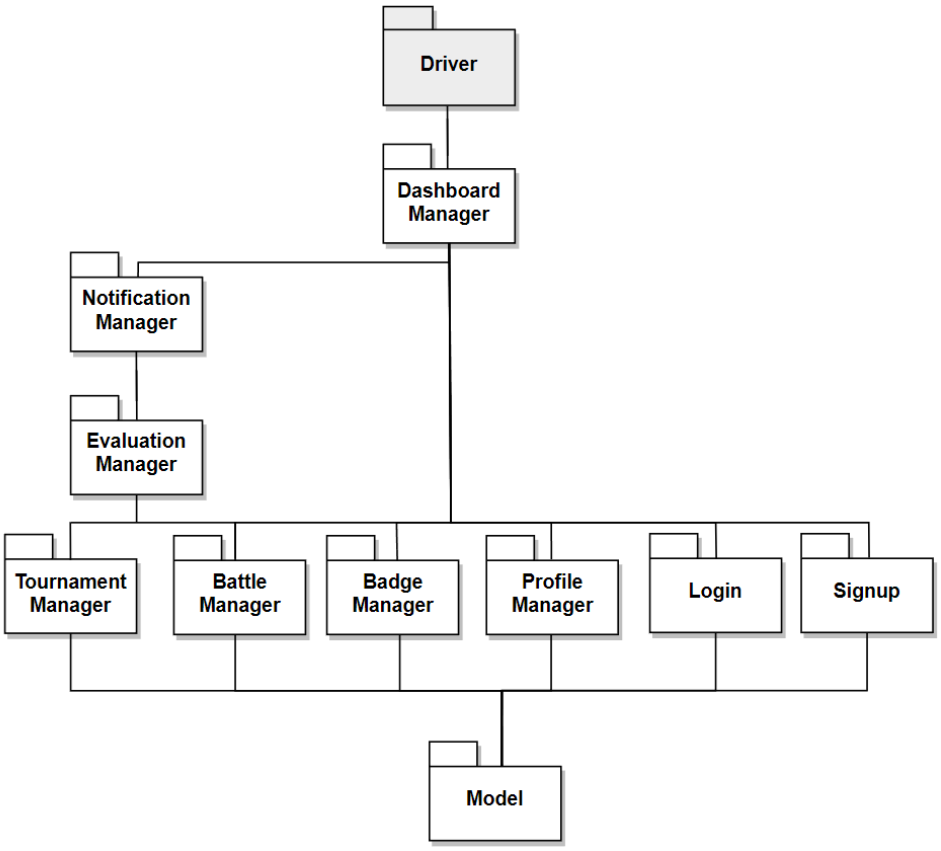
For the sake of simplicity, the previously integrated components are represented grouped in their Manager component (let the 'Tournament Manager' contain every Tournament-related component, and so on). Evaluation features will come next.



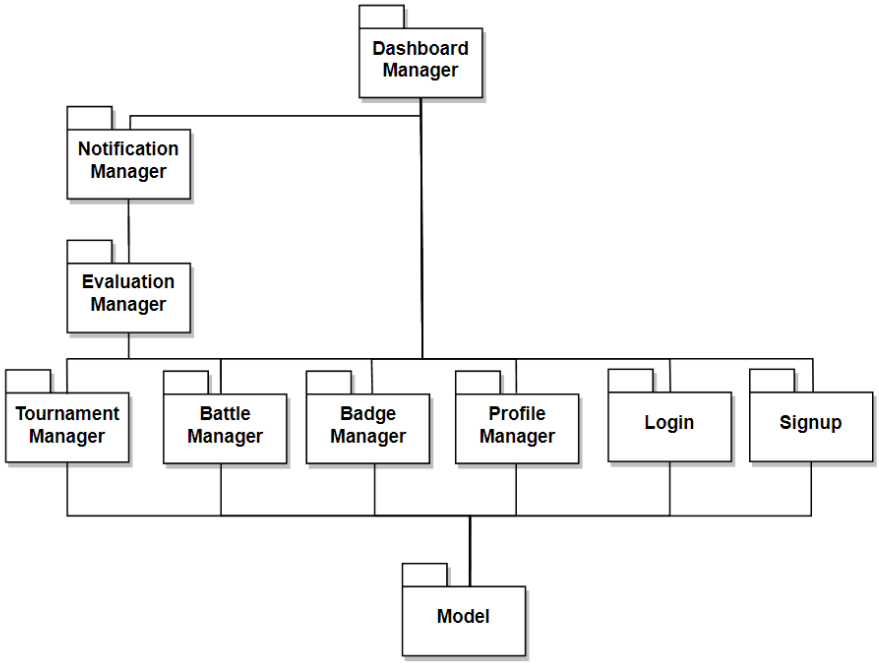
Once the Evaluation features are integrated, they will be considered as the same in the 'Evaluation Manager'. Then Notification Manager will be the next to be integrated and tested.



The last component that has to be integrated is the dashboard Manager, that is essential for the correct workflow of the user interface.



After the removal of the Dashboard Manager's Driver, the final system is as follows.



5.4. System Testing Strategy

It is necessary to check that each newly developed component works properly before integrating it with the system by the use of Drivers. Once a new component is integrated to the system, it must be checked with a new Driver that the new component is properly integrated, that the modules' properties still hold and that the integrated system follows the correct workflow. Once every component is integrated, it will be needed to test the system as a whole to ensure the proper workflow is followed and the absence of bugs. In order to do so the following kinds of testing will be applied.

- **Functional Testing:** Functional Testing will be performed on the system to guarantee that the workflow is correct and consistent with the functionalities described in the RASD document, checking the fulfillment of goals, requirements and use cases and the possibility to correctly simulate the described scenarios.
- **Load Testing:** Load Testing is useful in order to find eventual memory leaks, buffer overflows and bad management of memory.
- **Performance Testing:** The system will undergo this kind of testing in order to identify bottlenecks and to observe the resilience of the system under heavy workload, keeping in mind that the system shall support many users working simultane-

ously keeping response times as low as possible, following what is stated in the *RASD document, section 3.3*. This will also help to identify optimization possibilities in the software's algorithms.

- **Stress Testing:** In order to make sure that the system is capable of recovering itself after a failure, Stress Testing will be adopted, by simulating lots of concurrent users or reducing the system's computational resources.
- **User Interface Testing:** It is important that the system correctly works on different kinds of devices, and different browsers as stated during the Requirements Analysis, testing the usability and accessibility of the web app on every possible platform, for both kinds of users: EDs and STs.

6 Effort Spent

Member of group	Effort spent	
Ballabio Giacomo	Introduction	1.5h
	Architectural Design	13h
	User Interface Design	6h
	Requirements Traceability	0.5h
	Implementation Integration Test Plan	2h
	Reasoning	4h
Benelle Francesco	Introduction	1.5h
	Architectural Design	13h
	User Interface Design	5.5h
	Requirements Traceability	1h
	Implementation Integration Test Plan	7h
	Reasoning	4h
Cavallotti Alberto	Introduction	2h
	Architectural Design	28h
	User Interface Design	3.5h
	Requirements Traceability	0.5h
	Implementation Integration Test Plan	3h
	Reasoning	4h

Table 6.1: Effort spent by each member of the group.

7 References

7.1. References

- The Requirement Engineering and Design Project specification document A.Y. 2023–2024.

7.2. Used Tools

- GitHub for project versioning and sharing.
- LaTeX and *Visual Studio Code* as editor for writing this document.
- *sequencediagram.org* for the sequence diagrams' design.
- *draw.io* for the other diagrams' design.
- *Google Documents* for collaborative writing, notes and reasoning.

List of Figures

List of Tables