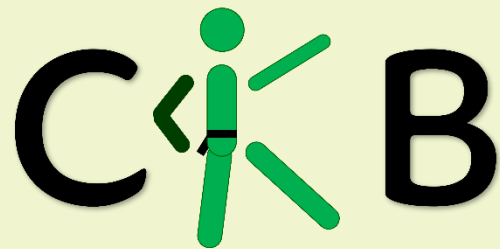


ITD:

Implementation & Testing Document



AY 2023/2024

Alessandro Fornara

Simone Colecchia

Prof. Matteo Rossi

INDICE

1. INTRODUCTION	3
A. Purpose	3
B. Definitions, acronyms, abbreviations	3
B.1. Definitions	3
B.2. Acronyms	4
C. Revision history	5
D. References	5
E. Document structure	6
2. DEVELOPMENT	7
A. Implemented functionalities	7
B. Adopted development frameworks	8
B.1. Backend	8
B.1.1 Spring MVC	8
B.1.2 Spring Data MongoDB	10
B.1.3 Spring Security	10
B.2. Frontend	10
B.2.1 Vue.js	10
B.2.2 Bootstrap	13
B.3. External communications	13
3. SOURCE CODE	14
A. Backend	14
B. Frontend	15
C. Packaging	17
4. TESTING	19
5. INSTALLATION AND USAGE	21
A. INSTALLATION	21
A.1. Prerequisites	21
A.2. Installation steps	21
B. USAGE	23
6. EFFORT SPENT	25

1. INTRODUCTION

A. Purpose

This document aims to provide an exhaustive overview of the process of developing and testing of CKB platform.

It details the key aspects, methodologies, and considerations in both development and testing phases, highlighting the advantages and disadvantages of the chosen frameworks and the structure of the developed source code.

Moreover, the last section details a step-by-step guide for installing the prototype of the CKB platform and test its main functionalities.

B. Definitions, acronyms, abbreviations

B.1. *Definitions*

- User:
It is a generic user of the platform, who wants to teach (Educator), learn, or improve coding (Student).
- Battle:
It is an event with fixed starting and ending time, consisting of a fight among multiple teams, which have to solve a Code Kata and provide their solution.
- Tournament:
It is a championship in which Students compete and it is composed of several sequential Battles.
- Public Tournament:
It is a tournament that is visible and open for all the Students subscribed to CKB.
- Private Tournament:
It is a tournament that is open only to a group of Students, who have been given the permission to visualize it and subscribe.
- Code Kata:
A kata is an exercise in karate where you repeat a form many, many times, making little improvements in each. Code Kata is an attempt to bring this

element of practice to software development. See more at <https://codekata.com/>.

- Tournament and Battle Final Rank:
The final rank for a tournament or battle is the rank that is computed and finalized at the end of them. It will be definitive, and no changes will be applied to it.
- Team:
It is a group of Students created to participate to a single Battle. Each team is composed by a certain number of Students between a minimum and a maximum value (fixed by the creator of the Battle).
- Keyword:
It is a unique alphanumeric string which is automatically generated by the system to preserve privacy and security. It is used to protect access to Private Tournaments or Teams.
- Automated Workflow:
A set of actions performed by an actor to automatically retrieve and push/send data whenever some changes are observed.
- Automatic Evaluation:
It is an automatic computation of a team score performed by the system, based on the provided solution for a Code Kata and considering some criteria (set by the creator of the Battle).
- Manual Evaluation:
It is a personal additional evaluation for a team score, which is performed manually by an Educator with respect to each member of the team and according to his discretion.
- GitHub:
Hosting at <https://github.com/>, it is the most famous platform for software development and version control, allowing users to store, manage and share their code.

B.2. Acronyms

- CKB: CodeKataBattle, the name of the platform
- MVC: Model-View-Controller
- DTO: Data Transfer Object, architectural pattern used for transferring data between software application subsystems. It is an object that defines the structure of data to be sent or received by/from servers.
- DAO: Data Access Object, architectural pattern for persistency management. It is an interface used to communicate with the DBMS.

- JWT: Json Web Token
- JS: JavaScript
- HTML: Hyper Textual Modeling Language
- DOM: Document Object Model, a programming interface for web documents. It represents the web page and programs can access it to change the document structure, style, and content.
- VDOM: Virtual DOM.
- SPA: Single Page Application, a web application or website that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from the server.
- JSON: JavaScript Object Notation, universal standard for the exchange of object-like information.
- YAML: Yet Another Markup Language, most famous markup language.
- GUI: Graphic User Interface
- CDN: Content Delivery Network, distributed server network delivering web resources such as assets, icons, CSS and so on.
- RASD: Requirement Analysis and Specification Document
- DD: Design Document

C. Revision history

- ✓ **V1.0** – First version of the document. Revised on 3rd February 2024.

D. References

- “Assignment IT AY 2023-2024”
- “RASD” by Alessandro Fornara, Simone Colecchia
- “DD” by Alessandro Fornara, Simone Colecchia
- Spring Boot documentation - <https://spring.io/projects/spring-boot>
- Vue.js documentation - <https://vuejs.org/guide/introduction.html>
- MongoDB documentation - <https://www.mongodb.com/docs/>
- GitHub Actions documentation - <https://docs.github.com/en/actions>
- GitHub API documentation - <https://docs.github.com/en/rest?apiVersion=2022-11-28>

E. Document structure

- **Section 1: Introduction**
This section offers a brief introduction to the document that is here presented, including all the definitions, acronyms and abbreviations that will be found reading it.
- **Section 2: Development**
This section offers a detailed view on all the aspects concerning the developing process. In particular, the implemented functionalities are described, together with all the adopted frameworks, which are described taking into consideration advantages and disadvantages.
- **Section 3: Source Code**
This section offers a concise but comprehensive overview of the structure and organization of the source code for the CKB platform, describing the roles of the main modules of the application.
- **Section 4: Testing**
This section provides an exploration of the testing methodologies applied during the development of the CKB platform.
- **Section 5: Installation and Usage**
This section is addressed to those involved in installing, configuring, and using the prototype of the CKB platform. It includes a comprehensive installation guide with step-by-step instructions, ensuring a smooth setup process.

2. DEVELOPMENT

A. Implemented functionalities

All the functionalities presented in the RASD and DD (listed below) have been developed and implemented as described in the other documents.

- Create Tournament
- Close Tournament
- Promote to Moderator
- Create Battle
- Automated Evaluation
- Evaluate
- Close Consolidation Stage
- Subscribe to Tournament
- Create Team and join Team

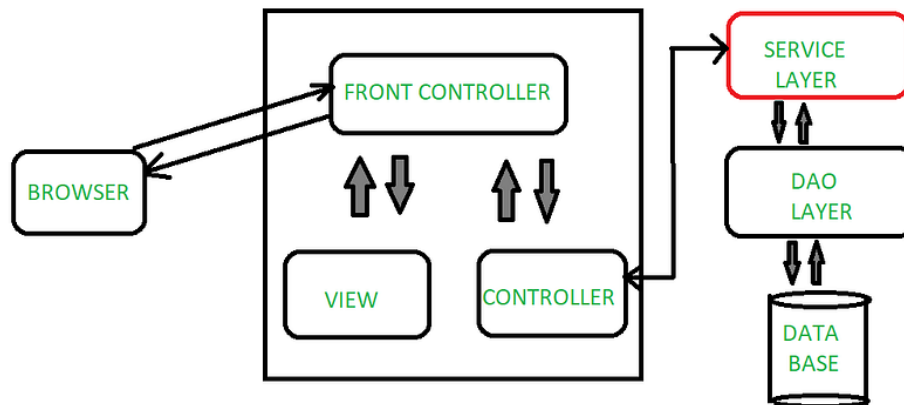
However, GitHub-related functionalities have been realized in a simpler way more suitable for a prototype. The released version of the application does not include a login through GitHub functionality, but it uses a single GitHub account on which the automated repository creation process takes place. Given that the GitHub API has some limitations on the number of requests receivable from a single client hourly, this could represent an issue for a market-ready product.

Finally, as requested to 2-member groups, the possibility to perform a static analysis of the pushed code has not been included. Educators can exploit the automated evaluation based on timeliness aspects and correctness of the outputs, which are computed and sent by GitHub Actions directly to the server.

B. Adopted development frameworks

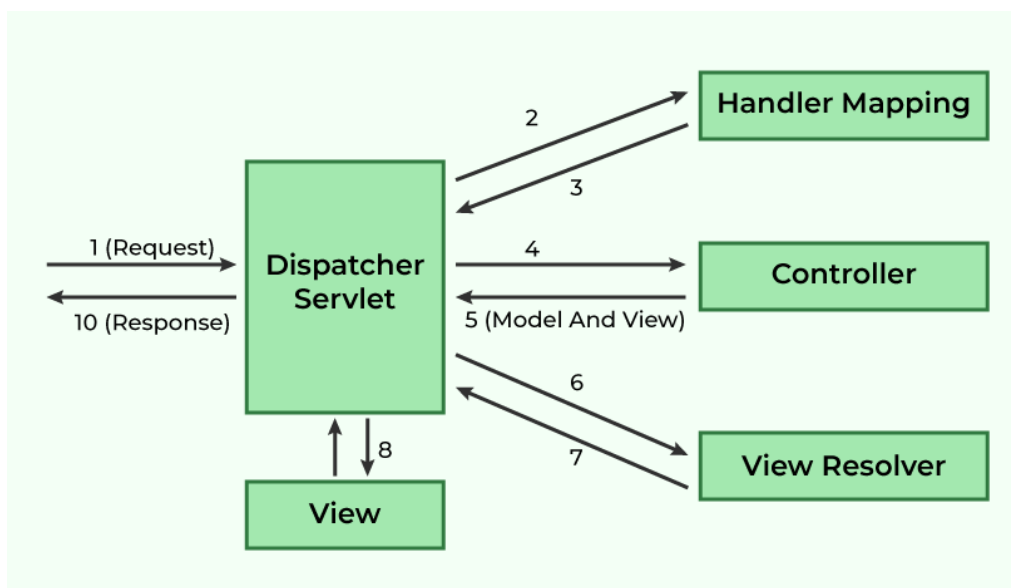
B.1. Backend

B.1.1 Spring MVC



Spring MVC Framework follows the Model-View-Controller architectural design pattern which works around the Front Controller i.e. the Dispatcher Servlet. The Dispatcher Servlet handles and dispatches all the incoming HTTP requests to the appropriate controller. It uses *@RestController* and *@RequestMapping* as default request handlers. The *@RestController* annotation defines that a particular class is a controller, while the *@RequestMapping* annotation maps web requests to Spring Controller methods. The controllers elaborate requests through services that interface with the database using DAOs.

Spring Model-View-Controller Flow Diagram



1. Initial Request Handling:
The DispatcherServlet acts as the primary controller. It is responsible for intercepting all incoming requests, effectively serving as a front controller.
2. Handler Mapping:
Once a request is received, the DispatcherServlet consults an XML configuration file to find the appropriate handler mapping. This mapping directs the request to the specific controller designated to handle it.
3. Controller Processing:
The chosen controller then processes the request. After completing its processing, the controller returns a ModelAndView object. This object encapsulates both the data and the view that should be rendered.
4. View Resolution:
The DispatcherServlet then refers back to the XML configuration file to identify the correct view resolver. Based on this information, it invokes the appropriate view component to render the response.

Advantages of Spring MVC Framework

- Lightweight Container Use:
The framework utilizes a lightweight servlet container for the development and deployment of applications, enhancing efficiency.
- Facilitates Rapid and Parallel Development:
Its design allows for quick development processes and enables multiple developers to work simultaneously without significant conflict.
- Ease of Updates:
Updating applications built with Spring MVC is generally more straightforward compared to other frameworks.

Disadvantages of Spring MVC Framework

- Complexity:
Developing applications with Spring MVC can be complex, especially for those new to the pattern.

B.1.2 Spring Data MongoDB

Spring Data MongoDB is a module of the Spring Data family which provides an easy-to-use framework for integrating Spring applications with MongoDB. It simplifies the use of MongoDB by providing a high-level abstraction for database interactions, reducing the amount of code required for data access.

B.1.3 Spring Security

Spring Security is a component of the Spring MVC framework, dedicated to managing the security of applications. It operates on a role-based access control system and utilizes various annotations to restrict access to certain controllers, ensuring that only authorized individuals can interact with them.

In the context of CodeKataBattle, Spring Security is specifically tasked with handling REST authentication, which is implemented using the JSON Web Token (JWT) pattern. JWT defines a concise and self-contained method for securely transmitting information as a JSON object. In CKB, this involves the exchange of tokens that authorize each client request. Therefore, clients are required to append this token to the header of every request they make to the server, ensuring secure and authenticated communication. For the sake of simplicity, automated workflows created through GitHub Actions don't need any token to access the associated endpoint.

B.2. Frontend

B.2.1 Vue.js

Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative and component-based programming model that helps in developing either simple or complex user interfaces.

This framework is built upon Node.js, which is responsible for the management of the creation and the boot process of a Vue project: it provides all the needed modules for its proper initialization and build, setting the directories and the main files of the application such as *index.html*, *main.js*, together with all the configuration files.

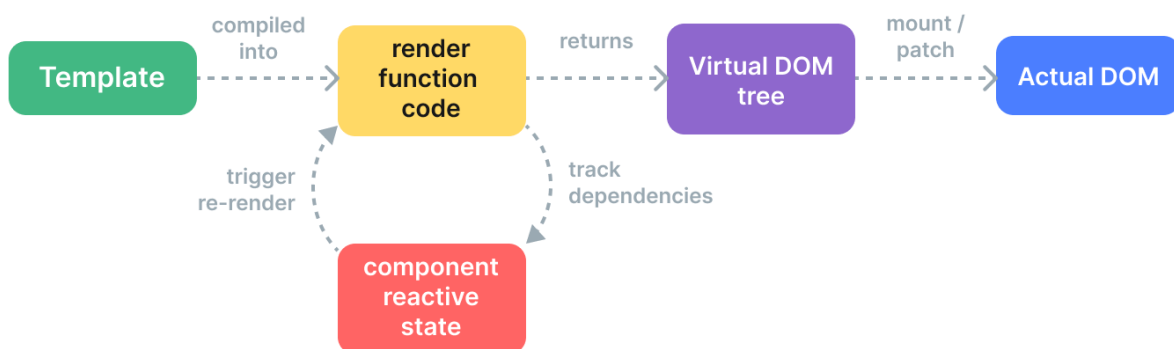
Furthermore, thanks to *npm*, Node packet manager, it is possible to start a local development server (`npm run serve`) to run the application and visualize on the screen all changes applied on the code in real time. In fact, when this server is on, all the components are transcribed and reorganized by Node into HTML and JS modules every time an update in the code is perceived, and the DOM is updated.

Vue.js rendering mechanism

Vue.js exploits the VDOM paradigm as its rendering mechanism.

In other words, an ideal, or "virtual," representation of the UI is kept in memory and synchronized with the actual DOM.

In Vue.js, the virtual DOM tree consists of plain JavaScript objects known as "virtual nodes" (*vnodes*). These *vnodes* represent elements in the UI and contain all the necessary information to create the actual DOM elements. The virtual DOM tree is traversed by the runtime renderer to construct the real DOM tree, a process referred to as "mounting."



Vue.js code structure

Vue source code is organized into Vue components (.vue files) which interact with each other to exchange data and cooperate. Each one consists of three sections:

- `<template>`

This section is dedicated to the definition of the HTML template of the corresponding page (or frame). Here, besides of pure html tags, some Vue directives (such as *v-model* or *v-bind* etc.) can be included in order to define reactive data binding between the template and the script. Thanks to the VDOM compliance, it is also possible to define dynamic behaviors for the html components, whose properties are dynamically updated alongside the changes in the script.

- `<script>`

In this section, the script of the Vue component is defined. Here data, methods and properties are specified. Vue offers some pre-defined methods that can be

overwritten to define the lifecycle of the component, i.e. what actions to be performed at the creation of the page, at the update of data and so on.

- `<style>`

This section is dedicated to the style specifications. Here CSS code is included to define the stylistic aspects of the template.

Advantages of Vue.js

- Flexibility and reactivity
Vue.js offers a high degree of flexibility, allowing developers to adopt it incrementally in existing projects or use specific parts of the framework as needed. This adaptability makes it suitable for a wide range of applications. Additionally, Vue.js is inherently reactive, meaning that changes in the underlying data are automatically reflected in the user interface. This reactive nature simplifies the development process, as developers don't need to manually update the DOM when the data changes.
- Two-way data binding
Vue.js includes a two-way data binding system, enabling the synchronization between the model and the view. When data in the model changes, the corresponding view is automatically updated, and vice versa. This bidirectional data flow reduces the amount of code needed to manage the state of an application, leading to cleaner and more maintainable code.
- Component-based architecture
Vue.js follows a component-based architecture, promoting modularity and reusability. Each component, as described just above, encapsulates their own logic, template and styles, making it easier to manage and scale complex applications. This allows developers to work on isolated components and assemble them to build the overall application.
- Ease of learning and Intuitiveness
The framework's syntax is clear and concise, making it accessible for developers who are new to Vue.js or frontend development in general. The intuitiveness of Vue.js simplifies the integration process, enabling developers to quickly grasp the key concepts and start building applications.

In general, Vue.js is a very smart and versatile framework and so it adapts to all kinds of UI projects. But given its features, it is particularly well-suited for SPAs, i.e. applications that loads a single HTML page and dynamically update content as users interact.

Since CKB platform has been thought to be developed as a SPA, Vue.js may be the best choice for developing the GUI of CKB.

B.2.2 Bootstrap

Bootstrap is an open-source front-end framework that streamlines the development of responsive and visually appealing web pages. Built on HTML, CSS, and JavaScript, Bootstrap provides a wide range of pre-defined and pre-styled components such as modals, navigation bars, forms, and more. These are included just by referencing the names of HTML classes and imported by adding the link to the CDN server to the main HTML file (*index.html*).

B.3. External communications

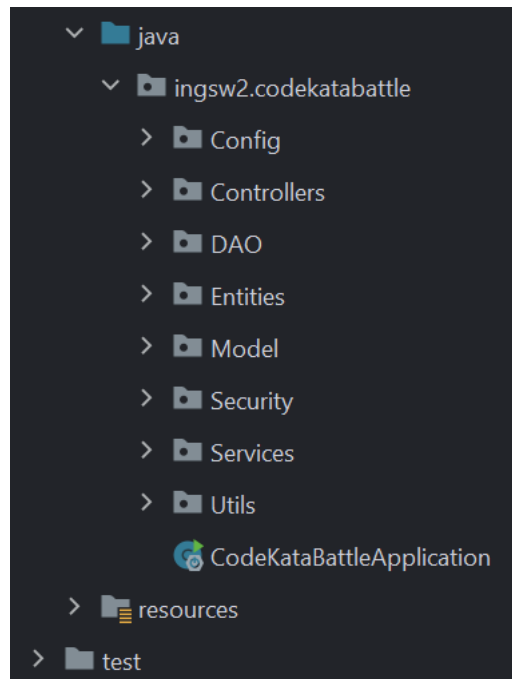
- GitHub API Integration:
CodeKataBattle uses the GitHub API for automation in repository creation when battle is initiated. Using the Java class *RestTemplate*, the CKB server initiates a request to the GitHub API. This request includes essential information such as a token and the associated GitHub account username linked to the provided token. Subsequently, the server deploys the files provided by the Educator who created the battle.
- GitHub Actions for Automated Evaluation:
CodeKataBattle integrates with GitHub Actions, allowing Students to automate the evaluation process. Students can configure workflows (for example through a YAML file) using GitHub Actions to send outputs associated with their code submissions.
- SMTP Server for Notifications:
To keep CodeKataBattle users informed, the server employs an SMTP server to dispatch notifications in the form of emails. A dedicated Google account (*codekatabattle@gmail.com*) has been created. This ensures that users receive timely updates and important information related to CodeKataBattle.

3. SOURCE CODE

A. Backend

The backend is developed using the Spring Boot framework merged with the Maven framework.

The code is structured as follows:



- Config:
This package contains the configuration classes of Spring framework, which manage security and general application configuration.
- Controllers:
This package contains the controllers of the application according to the MVC paradigm.
- DAO:
This package contains the DAOs which are used to query the DBMS.
- Entities:
This package contains the classes that represent database entities, such as tournaments, users and battles.
- Model:
According to the Spring framework, this package contains all DTOs used by the client to send information to the server and vice versa.

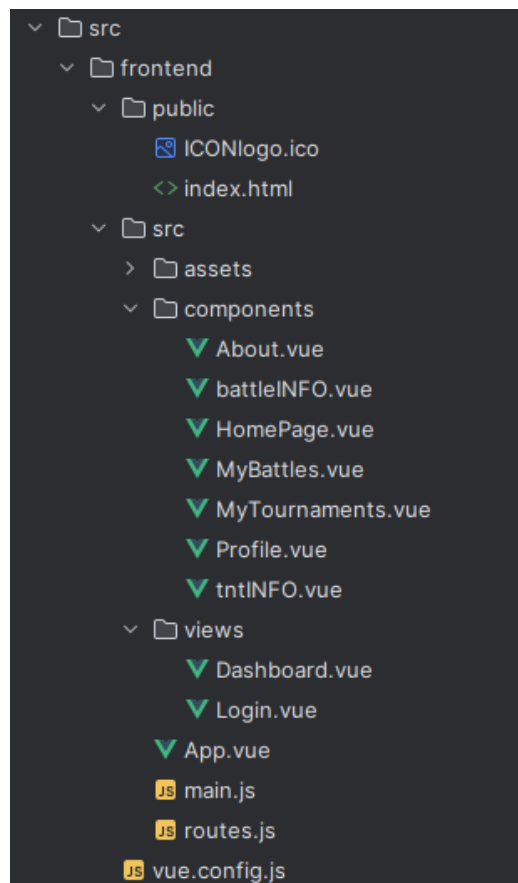
- Security:
This package contains the classes used of to secure the application, including the JWT logic and a filter which makes sure that users can access only content they're allowed to.
- Services:
This package contains the services which act as intermediators between controllers and DAOs.
- Utils:
This package contains classes which define methods used often by the other modules.

Refer to the in-code documentation for more detailed explanations regarding the functionalities and implementations within each module.

B. Frontend

For what has been previously explained in section 2.B.2.1, the frontend source code pushed in the GitHub repository consists only of the VUE and JS files which are behind the build of the node project.

The code is structured as follows:



- *Index.html*
This is the core HTML file of the whole frontend application. Here a `<link>` to bootstrap CDN server is included to make the dependency possible, and all the generated files are injected (see 2.B.2)
- *main.js* and *App.vue*
These two files are responsible for the boot of the VUE application. *App.vue* displays the view dispatched by the Router and *main.js* creates and mounts the application.
- *Routes.js*
This is the configuration file for the Vue Router. Here all the main sub-routes of the application are defined and configured.
- *Views*
Including the two views of the application: *Login.vue* and *Dashboard.vue*.
- *Components*
This directory includes all the components of the application, which are loaded by the views according to their route definition.
- *Assets*
This directory contains all the icons, logos and images loaded by the components when booted.

C. Packaging

In order to create the JAR executable of CodeKataBattle application, the backend and frontend source codes have been merged using the Maven framework:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <image>
          <builder>paketobuildpacks/builder-jammy-
base:latest</builder>
        </image>
      </configuration>
    </plugin>

    <plugin>
      <groupId>com.github.eirslett</groupId>
      <artifactId>frontend-maven-plugin</artifactId>
      <version>1.7.6</version>

      <executions>
        <execution>
          <id>Install node and npm</id>
          <goals>
            <goal>install-node-and-npm</goal>
          </goals>
          <phase>generate-resources</phase>
          <configuration>
            <nodeVersion>${node.version}</nodeVersion>
            <npmVersion>${npm.version}</npmVersion>
          </configuration>
        </execution>

        <execution>
          <id>npm install</id>
          <goals>
            <goal>npm</goal>
          </goals>
          <phase>generate-resources</phase>
          <configuration>
            <arguments>install</arguments>
          </configuration>
        </execution>

        <execution>
          <id>npm build</id>
          <goals>
            <goal>npm</goal>
          </goals>
          <phase>generate-resources</phase>
          <configuration>
            <arguments>run build</arguments>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
  <configuration>
    <nodeVersion>${node.version}</nodeVersion>
```

```
        <workingDirectory>src/frontend</workingDirectory>
    </configuration>

</plugin>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <executions>
        <execution>
            <id>Copy Vue frontend into Spring Boot target static
folder</id>
            <phase>process-resources</phase>
            <goals>
                <goal>copy-resources</goal>
            </goals>
            <configuration>
                <outputDirectory>target/classes/static</outputDirectory>
                <resources>
                    <resource>
                        <directory>src/frontend/dist</directory>
                        <filtering>true</filtering>
                    </resource>
                </resources>
            </configuration>
        </execution>
    </executions>
</plugin>
</plugins>
</build>
```

These plugins build the frontend by installing all necessary packages with the `npm run build` command, which compiles and minifies the Vue.js application for production, placing the output in a *dist* folder. Subsequently, the generated code is copied into the static folder of the Spring Boot source code, which is finally packaged into a JAR executable file.

4. TESTING

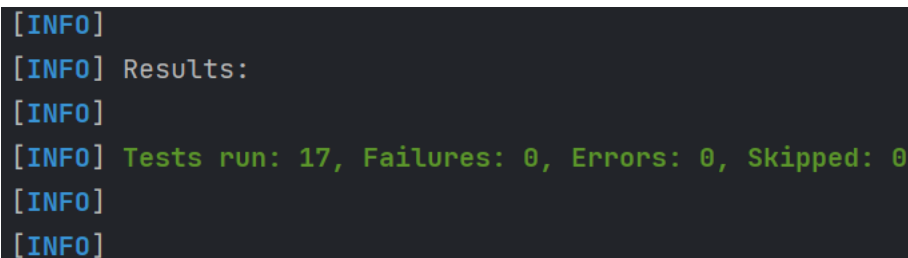
The application was tested through JUnit test cases. These tests were designed to simulate requests to various server endpoints using Spring Security's test support.

All methods contained in the controllers have been tested. This approach facilitated the testing of the underlying services and DAOs. As elaborated in 2.B.1, the integration of these components is crucial for constructing responses in the controllers. This strategy ensured that not only the front-facing components but also the backend functionalities were thoroughly evaluated for reliability and performance.

TEST CASES

- **AuthorizationControllerTest**
This class checks the correctness of the register and login functionalities and the exchange of the Json-Web-Token.
- **BattleControllersTest**
This class checks the correctness of all battle-related functionalities by first registering one Educator and two Students.
- **GitHubControllerTest**
This class checks the correctness of GitHub-related functionalities after creating the necessary scenario using the other functionalities.
- **StudentTournamentControllerTest**
This class checks the correctness of the subscribe functionality.
- **TournamentManagementControllerTest**
This class check the correctness of the all tournament-related functionalities for Educators.

The report detailing the outcomes of all previously described tests is depicted in the image below.



```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 17, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
```

TESTING THE AUTOMATED EVALUATION

Since it would be difficult to create a complete Junit test case for the automated evaluation, this functionality has been also tested manually, using GitHub Actions to emulate a real process of setting up a GitHub workflow and of pushing code.

INTEGRATION WITH FRONTEND

In addition to the backend testing, the application underwent a process of integration testing with the frontend to ensure seamless cooperation.

This involved validating that:

- All requests coming from the frontend are correctly forwarded to the application server, containing the token for the authorization in the *header* and, if involved, the correct JSON object in the *body*.
- All the server functionalities are accessible from the UI, which should react correctly either to positive responses or to errors.
- All the information received from the server are displayed correctly, i.e. they should be coherent with what stored in the database and with the local time zone (e.g. for what concerns dates).

5. INSTALLATION AND USAGE

A. INSTALLATION

A.1. *Prerequisites*

- ✓ Java SE JDK 17: Required to run the CKB application.

A.2. *Installation steps*

1) CKB.jar

- Location: Found in the delivery folder of our GitHub repository <https://github.com/AlessandroFornara/FornaraColecchia>.
- Installation: No installation required. Simply download the CKB.jar file.

2) MongoDB

- Optional: The CKB application can connect to an online cluster, so local installation of MongoDB is not mandatory.
- Local Installation (if needed):
 - Download the MongoDB Community Server from [MongoDB's official website](https://www.mongodb.com/try/download/community) (<https://www.mongodb.com/try/download/community>).
 - Run the downloaded installer and follow the on-screen instructions.
- Enabling Transactions:
 - For our cluster: No action needed; configuration is pre-set.
 - For local database: Install *mongosh* and follow the guide to convert to a replica set, available [here](https://www.mongodb.com/docs/manual/tutorial/convert-standalone-to-replica-set/) (<https://www.mongodb.com/docs/manual/tutorial/convert-standalone-to-replica-set/>).

3) Ngrok

- Purpose: Creates external access to a local server port.
- Download: Available at [Ngrok's website](https://ngrok.com/download) (<https://ngrok.com/download>).
- Registration: Not required. Use the provided key in our delivery folder on GitHub.
- Setup:
 - Run Ngrok.
 - Add the authentication token with:

```
ngrok config add-authtoken token
```

(token available in the delivery folder)

4) GitHub Token Generation

- Purpose:
To access the automated evaluation functionality within the CKB application, a GitHub token is required. This token allows the application to automatically create repositories when a battle is about to start.
- Steps to Create a GitHub Token:
 1. *Account Creation*:
 - Visit GitHub's website and sign up for an account if you don't already have one.
 2. *Access Settings*:
 - After logging in, navigate to your account settings.
 3. *Developer Settings*:
 - In your account settings, find and click on 'Developer settings'.
 4. *Personal Access Tokens*:
 - Select 'Personal access tokens' under 'Developer settings'.
 5. *Token Selection*:
 - Choose 'Tokens (classic)' for the token type.
 6. *Generate New Token*:
 - Click on 'Generate new token (classic)'.
 7. *Set Scope*:
 - Ensure to select the “repo” scope which provides necessary permissions.
 8. *Token Generation*:
 - Follow the prompts to create the token.
- Storage:
 - **Important**: Save the generated token securely, as you will need it in the subsequent steps outlined in the usage section of our documentation.

B. USAGE

1) Starting the Server

- Command:

```
java -jar CKB.jar --spring.profiles.active=cluster --  
gitHub.accessToken=token --gitHub.username=username
```

- Location: Execute the command in the directory where CKB.jar is located.
- Note:
 - Replace `cluster` with `local` if using a local MongoDB database.
 - Replace `token` with the GitHub token generated in the installation steps.
 - Replace `username` with your GitHub username that is associated with the token.

2) Accessing the Application

- URL: The application will be accessible at <http://localhost:8080/>.

3) Using GitHub Functionalities with Ngrok

- Start Ngrok: Run the Ngrok application.
- Create Tunnel: Use the command `ngrok http 8080`.
- Obtain Link: Ngrok provides a link (*'Forwarding'*) to the local server port (8080), which is used to integrate with GitHub Actions for automated evaluation.
- Further Instructions: Detailed instructions on how to proceed are available in the delivery folder in our GitHub repository.

4) Database Access

For a comprehensive view of the database data, we recommend installing MongoDB on your device. This installation includes MongoDB Compass, an intuitive graphical interface that simplifies the connection and management of your database. Users accessing the database via an online cluster should note that direct visualization of data is only possible with MongoDB Compass. During installation, online cluster users can skip the transaction

enabling step. To connect to the online cluster database, utilize the link provided in the delivery folder.

If you're working with a local database, MongoDB Compass facilitates the importation of prepared collections. These collections are accessible in our GitHub repository's delivery folder. Follow these steps:

1. Launch MongoDB Compass.
2. To create a new database, click the "+" symbol near "Databases". Name this database "CKB" (this is crucial).
3. In your new "CKB" database, create a collection. Name it to match the import file (for instance, if the file is "CKB.Battle," name the collection "Battle").
4. Navigate to "Collection," select "Import Data," and choose the relevant file.

5) GitHub Actions

Upon reaching the registration deadline, when a battle is set to commence, our server automatically initiates the creation of a public repository. This repository can be found on the GitHub account linked to the token and username specified at the application's startup. Within the delivery folder of our GitHub repository, you will find **further instructions and practical examples**. These resources are designed to facilitate your utilization of the automated evaluation feature, including the activation of the GitHub Actions workflow and other related functionalities.

For any doubts or issues you can contact us:

- alessandro1.fornara@mail.polimi.it
- simone.colecchia@mail.polimi.it

6. EFFORT SPENT

Section	Fornara	Colecchia	Together	TOTAL
Section 1	0	1	0	1
Section 2	2	2	0	4
Section 3	1	1	0	2
Section 4	1	0	0	1
Section 5	2	0	0	2
Revision&Restyling	0	2	1	3
TOTAL	6	6	1	13