

Project Report – Course: “Introduction to Web Programming”

Group ID: 03 - De Toffoli Alessandro

Introduction

The project consists of two web applications: **MatchWeb** and **REST_App**.

MatchWeb is the main web application; it allows users to register and play betting slips for one of the following sports: Baseball, Cricket, or Padel.

However, the only sport actually supported by the platform is **Cricket**, for which the **REST_App** application provides a list of fictitious teams, a random schedule, and aleatory results for each match.

The application also assigns each user one of the following roles: **ADMIN#03**, **USER#03**, and **MODERATOR#03**. The functionalities offered vary depending on the assigned role.

MatchWeb - Implementation

Throughout the project, I aimed to maintain a modular and coherent structure, in which **@Beans** manage specific aspects of the business logic.

A consistent pattern was followed: **@Controllers** use **@Services**, which in turn use **@Components** and **@Repositorys**.

The application is divided into a public and a private section:

1. Public Section – Services

The public section is accessible to any user without authentication:

1.1 @Controller

The public pages are managed by two **@Controllers**:

- **PublicController**
- **SecurityController**

1.1.1 PublicController

Handles the views related to public services and information pages outside of login and signup, such as **/index**, **/sponsor**, **/ratings**, etc.

1.1.2 SecurityController

Manages user registration and authentication.

It uses the **Signup** service to register new users and insert the relevant data into the database.

Credential validation during login, as well as session invalidation during logout, is handled by **Spring Security**.

Once authenticated, the controller uses the **Authentication** interface to determine the user's role and returns the appropriate dashboard (adminDashboard or userDashboard).

1.2 @Service

1.2.1 NewsService

The News section is handled client-side via `fetch()` requests to a REST endpoint `/news`, managed by **PublicController**, which uses the **NewsService** to return news items.

Initially, the news is stored in a text file `news.txt`.

Upon application startup, **NewsService** extracts, shuffles, and cyclically returns the news in an `ArrayList<String>`.

All news is temporarily stored in the browser's `SessionStorage` to allow easy access via JS on page changes.

1.2.2 RatingService

This service interfaces with the **RatingRepository** (see 3.3) and returns an `ArrayList<Rating>` containing all saved comments from the database.

1.2.3 Signup

A service that validates registration data on the backend and inserts the user into the database using **UserRepository** and **ScoreboardRepository** (see 3.1, 3.2).

It uses a **Validator** component to check that the password, sport, and date of birth meet the project's requirements.

2. Private Section – Services

The private section provides different dashboards depending on whether the authenticated user is an ADMIN or a USER/MODERATOR.

Both roles can view profile data and change their password.

**(Not explicitly required for ADMIN in the assignment but implemented anyway.)*

2.1 @Controller

- **UserController** – manages USER and MODERATOR actions
- **AdminController** – manages ADMIN actions

2.1.1 UserController

Handles all actions and pages available to users with USER or MODERATOR roles.
Uses **FinalScoreService** and **UserActionService**.

2.1.2 AdminController

Manages all ADMIN-exclusive pages and actions.
Uses **AdminActionService** and **AssegnaPremi** to manage users and assign prizes.

2.2 @Service

2.2.1 UserActionService

Handles actions available to users: writing reviews, viewing/editing profile, and checking game eligibility.
Uses **UserRepository**, **ScoreboardRepository**, and **RatingRepository**.

2.2.2 FinalScoreService

Connects to **REST_App** through **ExternalMatchService** (see 5.0) to fetch match results.
Provides methods to update user scores and return a **List<Integer>** with match results, available through a REST endpoint.

2.2.3 AdminActionService

Used by **AdminController** to manage admin actions.
Interfaces with **UserRepository** and provides methods to promote users or retrieve a list of registered users.

**(Not strictly necessary, but adds modularity and consistency.)*

2.2.4 AssignPrizesService

Used by **AdminController** to manage the podium and prize assignment through **ScoreboardRepository**.
Can retrieve the top 3 users, assign them prizes, and reset the leaderboard.

3. Database

The database uses 7 tables; including those required by Spring Security:

- **USERDATA**: user personal data
- **RATINGS**: reviews, authors, dates, scores
- **SCOREBOARD**: user rankings
- **GIORNATE**: user activity data
- **PRIZES**: prizes

Three `@Repository` classes are used to access the database.

**A `schema.sql` file is provided to initialize the database and insert fake users for testing, with both ADMIN and USER credentials.*

3.1 UserRepository

Handles user-related data (role, password, personal info) and is used by various services.

3.2 ScoreboardRepository

Handles ranking and game-related data (**SCOREBOARD** and **GIORNATE** tables).

Maintains an in-memory `ArrayList<User>` sorted by score, refreshed with each score update.

Also provides methods to initialize new users' game data.

3.3 RatingRepository

Used by `RatingService` and `UserActionService` to insert/read reviews using `addRating()` and `getRatings()` methods.

4. ExternalMatchService

MatchWeb communicates with **REST_App** through an OpenFeign interface `MatchProxy`, implemented in `ExternalMatchService`.

This service is used to fetch teams (e.g., during signup), match schedules, and results.

5. POJOs

Plain Java objects are used to transfer data efficiently.

Besides `SecurityUser` (used by Spring), the following classes are implemented:

- **Match**: used with OpenFeign to parse JSON data from `REST_App`
- **Rating**: holds review info (author, score, date, comment)

5.1 Users

Five different user classes exist depending on context:

- **BasicUser**: abstract class with minimal data and common methods
- **InfoUser**: adds public profile info (used in admin user list)
- **ScoreboardUser**: adds ranking data (used in leaderboard)
- **SignupUser**: used only during registration
- **FullUser**: combines InfoUser and ScoreboardUser (used in user dashboard)

This class setup adds some business logic complexity but improves code readability by avoiding long constructors or RowMappers.

The redundancy in FullUser is due to a design mistake – Java, unlike C++, doesn't support multiple inheritance.

6. Views

All HTML pages use **Thymeleaf** as the template engine and **Bootstrap** for styling. Reusable fragments (e.g., navbar, footer) are included via `fragment.html`.

Custom MatchWeb logos were created using Canva.

Sponsor logos are AI-generated and fictitious to avoid copyright issues.

7. Security

Private pages/services are protected by Spring's Security Filter Chain requiring authentication and role-based access.

All SQL queries are parameterized to prevent SQL injection.

Thymeleaf's `th:text` escapes HTML by default, mitigating XSS and SSTI risks.

CSRF tokens are enabled and handled automatically by Spring/Thymeleaf for forms and manually added in `fetch()` headers for JavaScript.

REST_App - Implementation

This secondary app provides match information. It includes:

- **@RestController** – MainController
- **@Service** – CalendarManager

1. CalendarManager

Handles the match calendar with three **ArrayLists**: calendar, teams, and results.

Teams are loaded from **teams.txt**, paired randomly, and assigned a match date (yesterday, today, or tomorrow).

At least one match is always scheduled for today.

Random results are generated in the **[0, 2]** range.

(It uses the same **Match class as MatchProxy in Team03_MatchWeb)*

2. MainController

A **@RestController** that exposes endpoints returning JSON data via CalendarManager.

Challenges and Future Improvements

The main issue faced was adding a "show password" toggle button in the forms.

Although the **togglePwVisibility()** function exists in **form-validation.js**, it was not used because the button conflicted with Bootstrap labels and caused layout bugs.

For future improvements, since OpenFeign data is quite static, we could implement data caching and a fallback service to prevent crashes when Team03_REST is unavailable.

Additional Features

To replace numerical scores with star ratings, we used a JS + CSS library downloaded and slightly modified from:

<https://github.com/pryley/star-rating.js>

Lessons Learned

Beyond the topics covered in the course, this being my first project-based exam, I learned how to plan and consistently develop a medium-complexity web application within a fixed deadline.

[DISCLAIMER]

ChatGPT and/or other AI tools were used for:

- *Writing descriptions and informational texts*
- *Creating fake names and logos*
- *Advanced search*
- *Quick debugging and interpreting error messages and exceptions*