Università
degli Studi
di Ferrara

Soccer Activities Management

Progetto del corso di Ingegneria del Software Avanzata

Michele Vaccari - Matricola 121955

13 settembre 2022

Università degli studi di Ferrara
Corso di laurea magistrale in Ingegneria Informatica e dell'Automazione
AA 2021-2022

Indice

Processo di ingegneria dei requisiti

Descrizione

Il sito *Soccer Activities Management* è una piattaforma per la gestione delle attività sportive attraverso una comunità di utenti.

Caratteristiche i

1. Possibilità di creare gli utenti per i due ruoli principali: arbitri e gestori di squadra
2. Possibilità di creare i diversi tornei, di disegnarne lo schema. Ogni torneo ha: nome, descrizione e tipo (all'italiana o ad eliminazione)
3. Un torneo è costituito da diversi gironi o fasi, in base alla tipologia di torneo. Un girone ha un nome (ad esempio: girone 1) ed è costituito da un insieme di gare. Una fase ha un nome (ad esempio: semifinale 1) ed è costituita da una sola gara
4. Una gara è descritta da: data e ora, luogo, nomi delle squadre che si sfidano, un arbitro
5. Per ogni torneo è associata una classifica generale che viene aggiornata sulla base dei referti di gara compilati dagli arbitri

Caratteristiche ii

6. Possibilità per il gestore di una squadra di creare una squadra con una rosa di massimo 36 giocatori. Ogni squadra ha un nome e può avere uno sponsor. Per ogni giocatore il gestore può specificare: nome, cognome, luogo e data di nascita, numero di maglia e foto
7. Per ogni gara cui la squadra è assegnata, il gestore della squadra deve fornire la formazione della squadra composta da nome e cognome dei giocatori e rispettivi ruoli
8. A gara terminata l'arbitro dovrà compilare un referto in cui annoterà: l'orario effettivo di inizio e di fine della gara, risultato finale, numero di reti con i rispettivi giocatori che li hanno realizzati, i giocatori espulsi per ogni squadra e i giocatori ammoniti per ogni squadra

Caratteristiche iii

9. Possibilità di selezionare il torneo desiderato e visualizzare:
    - La pagina relativa ad una gara con: nomi delle squadre, formazioni e referti
    - La pagina relativa ad una squadra con la rosa dei giocatori e il calendario delle partite
    - La pagina relativa ad un giocatore con le statistiche di gioco (punti realizzati, espulsioni e ammonizioni) per quel torneo

Utenti del sistema

Il sistema prevede che le categorie di utenti sia così rappresentata:

Amministratori Possono effettuare i punti dal 1 al 5 compresi

Gestori di squadra Possono effettuare i punti dal 6 al 7 compresi

Arbitri Possono effettuare solamente il punto 8

Utenti pubblici Possono effettuare solamente il punto 9

Demo

Demo del sistema

Rileggiamo le specifiche e vediamo l'implementazione del sistema sviluppato utilizzando i dati demo.

Tabella 1: Credenziali e ruoli degli utenti demo del sistema

Email	Password	Tipo	Squadra
m.vaccari@sam.com	Password01	Admin	
u.cairo@sam.com	Password01	Team Manager	Torino
w.mattioli@sam.com	Password01	Team Manager	S.P.A.L. 2013
m.setti@sam.com	Password01	Team Manager	Hellas Verona
j.saputo@sam.com	Password01	Team Manager	Bologna
m.busacca@sam.com	Password01	Referee	
p.collina@sam.com	Password01	Referee	

Licenza

Licenza MIT

Si è scelto di distribuire il software sviluppato tramite licenza *MIT*.
La licenza MIT è una licenza permissiva, molto semplice e breve, che permette di fare ciò che si vuole col codice a patto che la licenza venga re-distribuita nel codice che si utilizza.

Componenti del sistema

Architettura software

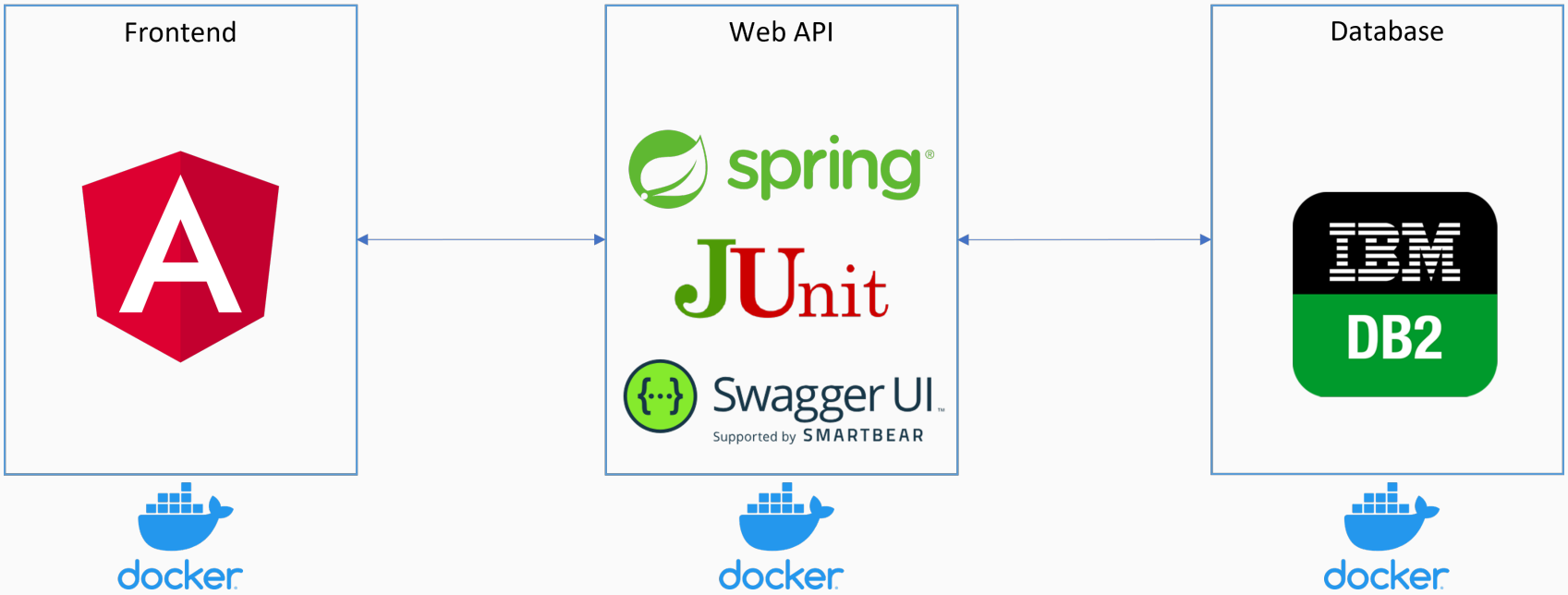

Figura 1: Architettura del sistema

Frontend

- **Linguaggio:** TypeScript
- **Framework:** Angular
- **Gestore delle dipendenze:** npm (vedi *frontend/package.json*)
- **Building system:** Node.js

Le fasi di *Build* e *Deploy* vengono gestite utilizzando docker (vedi *frontend/Dockerfile*)

Database i

- **Linguaggio:** SQL (IBM Db2)

Le fasi di *Build* e *Deploy* vengono gestite utilizzando docker (vedi *database/Dockerfile*)
È possible utilizzare dei dati demo per vedere l'applicazione in funzione oppure fare un'installazione pulita (vedi *database/initialization-scripts*)

Database ii

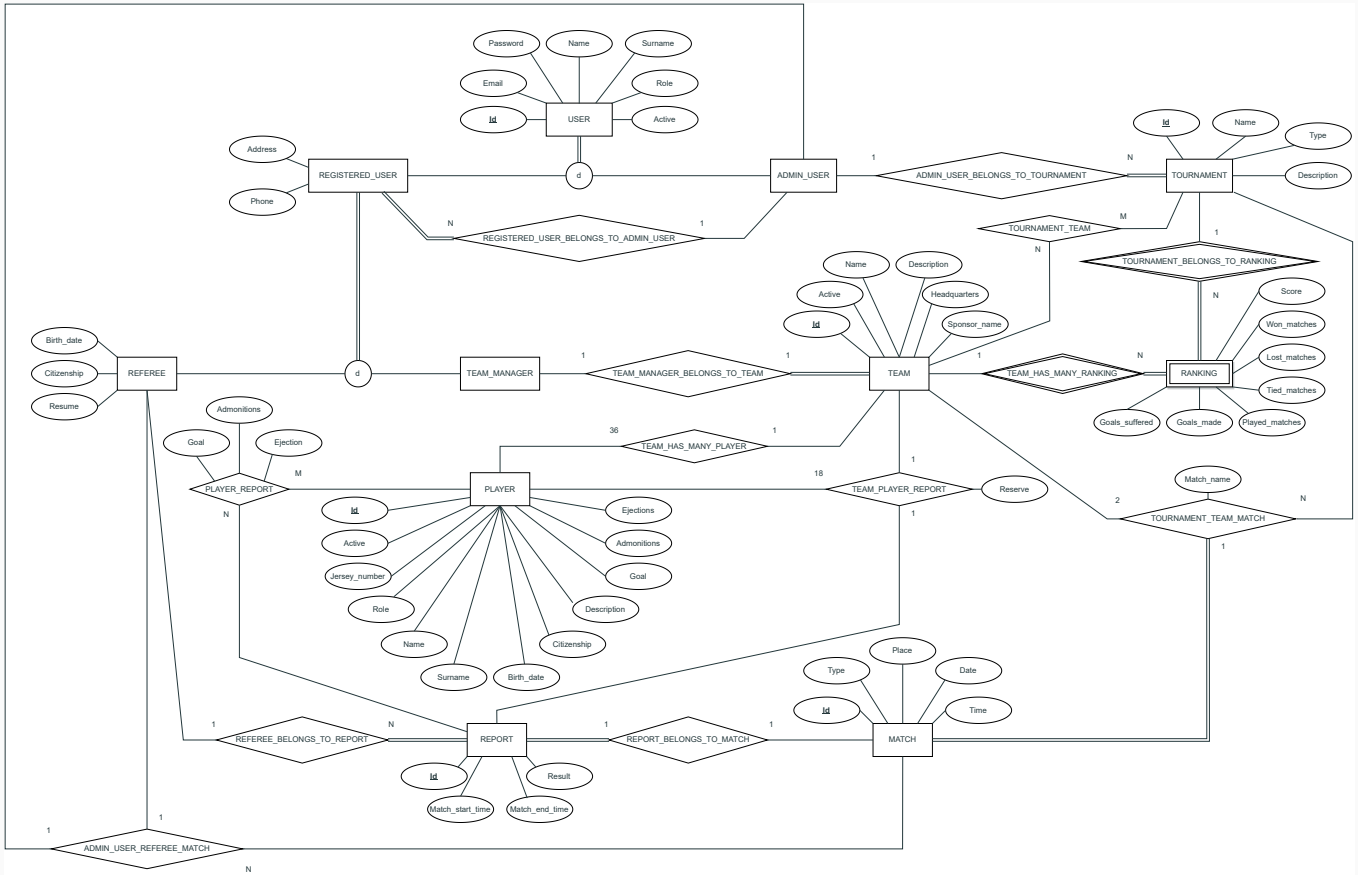

Figura 2: Schema ER

Web API i

- **Linguaggio:** Java
- **Framework:** Spring Boot
- **Gestore delle dipendenze:** maven (vedi *web-api/pom.xml*)
- **Target system:** OpenJDK 11

Le fasi di *Build* e *Deploy* vengono gestite utilizzando docker (vedi *web-api/Dockerfile*)

Web API ii

Una volta avviato il container *Web API* è disponibile la documentazione delle API REST

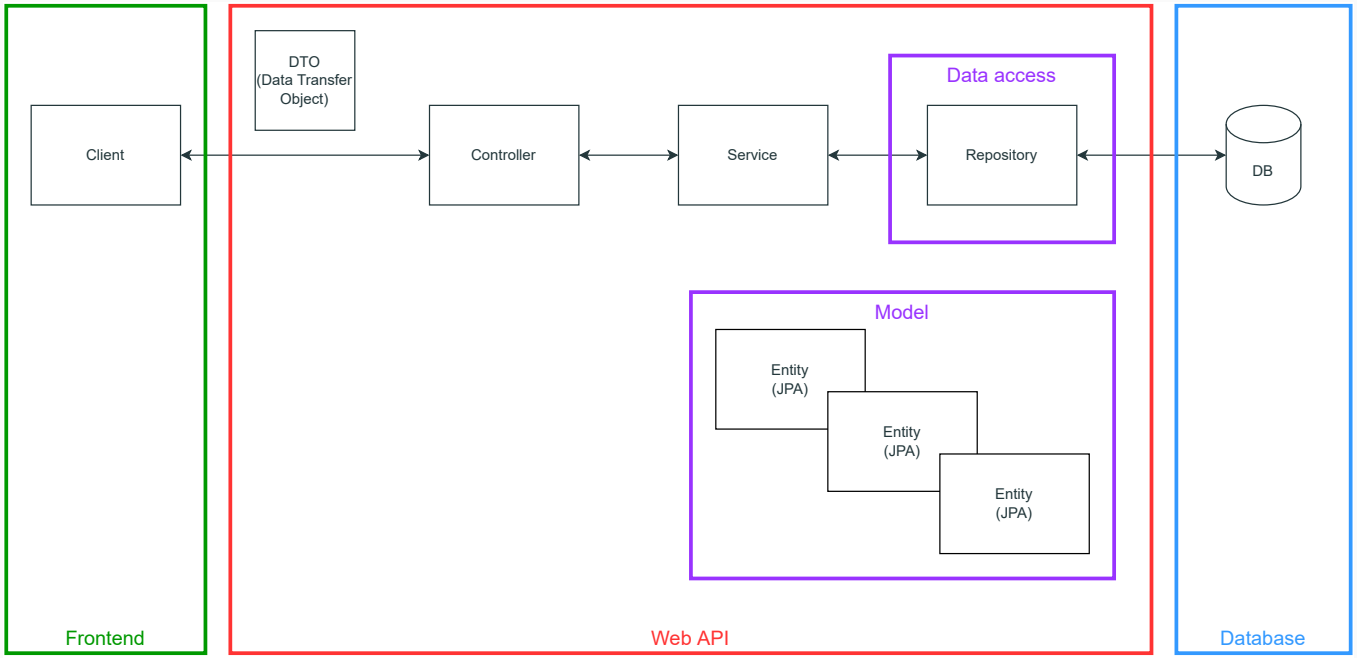

Figura 3: Web API data flow

Web API iii

Test White Box

Unit test (approccio bottom up):

- Dto (esempio AdminDtoTest.java / AdminDto.java)
- Model (esempio AdminUserTest.java / AdminUser.java)
- Service (esempio AdminServiceImplTest.java / AdminServiceImpl.java)
- Controller (esempio AdminControllerTest.java / AdminController.java)

Difetti:

- Copertura legata all'implementazione: una modifica dell'implementazione richiede di rivalutare i casi di test.
- Non rilevano la mancata implementazione di funzionalità

Web API iv

Test Black Box (basati sulla specifica di un modulo, anziché sulla sua implementazione)

- Come specifica del modulo utilizzare la documentazione delle API REST
- Utilizzo del software Postmann per creare una suite di test

Tipi di processo e strumenti usati

Tipi di processi usatii i

Si sono utilizzati prevalentemente processi agili, in particolare si è cercato di:

- Sviluppare e consegnare per incrementi: testare e valutare ogni incremento
- Mantenere la semplicità: limitare la complessità del sistema sia in fase di sviluppo che in fase di progettazione
- Limitare l'uso di duplicazioni di logica e l'uso di commenti [2]

Durante lo sviluppo si è seguito un tipo di processo pianificato-agilmente (uso del metodo dei *proiettili traccianti*[1]): si traccia una linea guida delle attività da seguire pianificandole in anticipo. Durante l'analisi e lo sviluppo si possono sperimentare e modificare le tecnologie a disposizione (plugin e librerie) ma non i requisiti.

Tipi di processi usatii ii

Extreme Programming

- *Incremental planning e user story:* non esiste un grande piano per il sistema e i requisiti vanno discussi incrementalmente con un rappresentante del cliente. I requisiti prendono anche il nome di *user story* e sono determinati dal tempo disponibile e dalle diverse priorità. Dalle story nascono le funzionalità da implementare
- *Small release (o meglio small feature sul branch develop):* rilasci frequenti. Si sviluppa prima un insieme minimo di funzionalità, per avere un business, poi si integrano con release successive
- *Continuous integration:* integrazione continua. Appena il codice è pronto, questo viene integrato nel sistema e quindi nasce una nuova versione. Tutti i test di unità devono essere eseguiti in maniera automatica e devono avere successo affinché una nuova versione del software sia accettata

Tipi di processi usatii iii

- *Refactoring:* miglioramento della struttura e dell'efficienza del codice per agevolarne la manutenzione

Tipi di processi usatii iv

Integrazione e configurazione

Svantaggi (vedi caso dell'uso di Spring Security e validazione del JWT):

- Non sempre il software riusabile è sufficientemente flessibile per adattarsi completamente ai requisiti con la sola configurazione (e valutare ciò a priori è difficile)
- Molti framework rendono banale implementare gran parte dei requisiti, ma difficile implementare i rimanenti; il risultato è che spesso si decide di lasciare non implementati i requisiti problematici
- Non si ha controllo sull'evoluzione del software riutilizzato
- È necessario adattarsi alla filosofia progettuale del software riusato

Modello di Gitflow usato

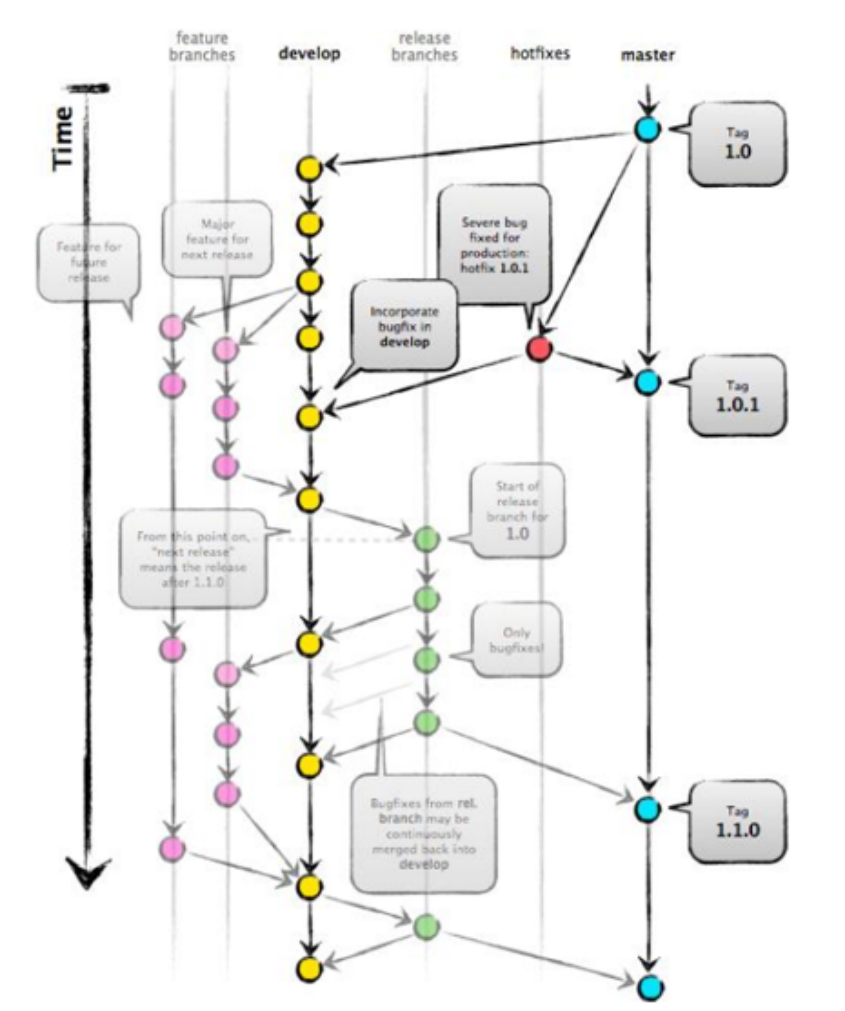

Figura 4: Git flow

Convalida e Verifica dei componenti

Frontend

- Code walkthroughs (esecuzioni manuali)
- Code reviews (prima di mergiare una PR)

Web API

- Approccio White Box
- Apporccio Black Box
- Code walkthroughs (esecuzioni manuali)
- Code reviews (ispezioni alla ricerca di errori o regole di codifica non rispettate)

Database

- Code walkthroughs (esecuzioni manuali)
- Code reviews (prima di mergiare una PR)

Continuous Integration

Continuous integration (CI): non appena lo sviluppatore completa un task, le modifiche vengono integrate nel sistema (push nel repository remoto). Dopo l'integrazione il sistema viene compilato (build del sistema) e vengono eseguiti tutti i test in maniera automatica. Il codice può essere poi integrato nel branch principale (main, master o altro nome).

GitHub actions
Vedi:

- CI Frontend *.github/workflows/frontend.yml*
- CI Web API *.github/workflows/web-api.yml*

Conclusioni

Possibili sviluppi

- Implementare Continuos Delivery (branch main $\rightarrow$ stable, branch release $\rightarrow$ beta, branch develop $\rightarrow$ alpha)
- Implementare Continuos Deployment (identificare cloud provider dove deployare gli artefatti prodotti)
- Campagna di beta tester
- Potenziare la parte di test automatici
- Tool per tracciare le dipendenze in modo eterogeneo, utile anche per notifica e valutazione delle vulnerabilità di sicurezza (ad esempio Dependency Track
- Introduzione di uno strumento per code coverage (ad esempio SonarQube)
- Integrazione nella pipeline della generazione di documentazione software (preferibilmente redatta da un technical wiriter, utilizzando un tool come Sphinx)

Considerazioni

- Semplicità vs. Astrazione (duplicazione e logica vs generalizzazione)
- Più un codice è generico e più test automatici chiameranno quel codice
- Test automatici vs test manuali
- Performance vs manutenibilità

Bibliografia i

A. Hunt and D. Thomas.
Il Pragmatic Programmer: Guida per manovali del software che vogliono diventare maestri.
Maestri di programmazione. Apogeo, 2018.

R. Martin.
Clean Code: Guida per diventare bravi artigiani nello sviluppo agile di software.
Maestri di programmazione. Feltrinelli Editore, 2018.

Grazie per l'attenzione