REPORT

Design and development of a scalable and reliable service:

courtsight

Performed by:
Leonardo Baraldi, Riccardo Barbieri
Leonardo Ruberto, Gabriele Tassinari

Date: July 2024

Indice

1 Introduzione

Questo documento illustra la realizzazione del progetto di *Scalable e Reliable Services*.

L'obiettivo è stato quello sviluppare un servizio per la previsione degli esiti delle partite di NBA, accessibile tramite un'applicazione web. Inoltre, l'applicazione realizzata è stata adibita per fornire informazioni statistiche relative a partite, squadre e giocatori, incluse le quote di scommessa per i mercati head-to-head e spread (head-to-head con handicap).

1.1 Architettura

Di seguito viene descritta l'architettura del sistema, inclusi tutti i componenti e le interazioni tra essi.

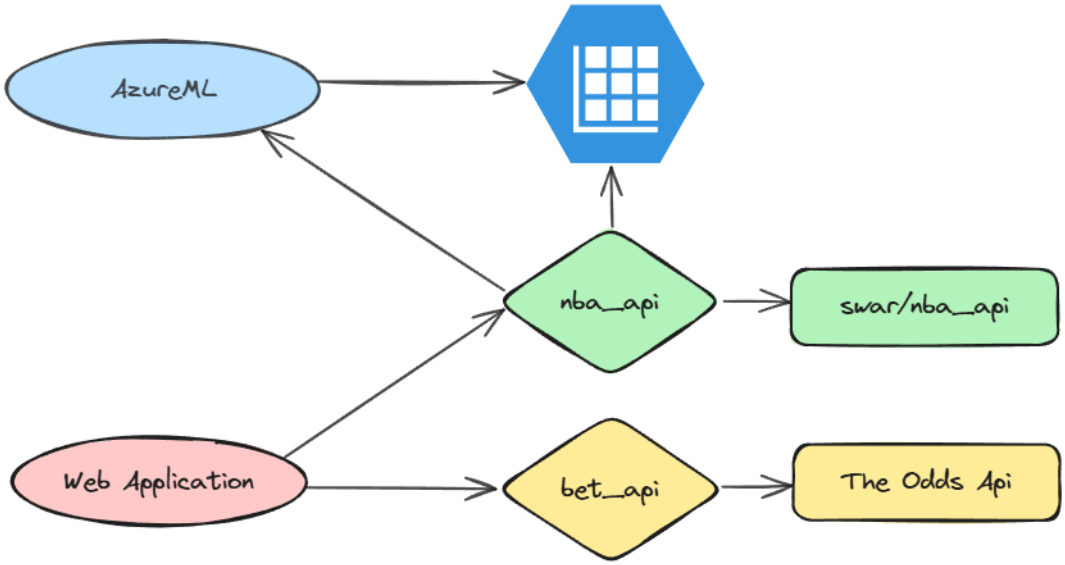


Figura 1: Architettura del sistema

Ai margini dell'architettura, rappresentati come rettangoli, troviamo due servizi esterni: `swar/nba_api`, libreria in Python utilizzata per ottenere informazioni statistiche sulla NBA, e `The Odds Api`, un servizio REST API che fornisce le quote di scommessa aggiornate in tempo reale.

Per facilitare l'uso di questi servizi, sono stati sviluppati due middleware sotto forma di REST API server. Questi middleware offrono endpoint mirati e semplificati, rispetto alle funzioni esposte dai servizi esterni, per l'interazione con quest'ultimi.

In azzurro, è rappresentato un modello di Machine Learning implementato sulla piattaforma Azure. Questo modello è stato progettato per prevedere l'esito delle partite NBA ed addestrato utilizzando un database popolato con dati storici delle partite, estrapolati dal middleware `nba_api` direttamente dalla libreria `swar/nba_api`.

Infine, in rosa, troviamo l'applicazione web che interagisce con i due middleware per ottenere le informazioni necessarie da visualizzare e, anche se non direttamente, con il modello di Machine Learning per ottenere la predizione dell'esito di una determinata partita.

Nei capitoli successivi verranno analizzati nel dettaglio tutti i componenti descritti. Inizieremo con i middleware, per poi passare al modello di Machine Learning e infine all'applicazione web.

Nella fase conclusiva, verrà descritto il processo di deployment del servizio in cloud sulla piattaforma Azure, con una spiegazione delle risorse utilizzate, dei costi sostenuti e di una panoramica sul load balancing.

2 Middleware

L'obiettivo del nostro progetto è stato quindi quello di realizzare un servizio web per fornire statistiche sui match NBA e provare a predirne i risultati. Come anticipato, abbiamo esteso questa idea creando un'applicazione web (analizzata in seguito) che permettesse non solo di prevedere l'esito di un match NBA, ma anche di fornire tutte le informazioni necessarie per valutare personalmente quale squadra avesse maggiori probabilità di vincere, includendo anche le quote offerte dai bookmakers. In pratica, il nostro progetto si è avvicinato alla realizzazione di un sito di scommesse con l'aggiunta di informazioni e previsioni sui match, ovviamente limitato al contesto NBA.

Per realizzare questa applicazione web, abbiamo avuto bisogno di due tipi di informazioni principali:

- dati statistici sulle partite, sulle squadre e sui giocatori della NBA;
- informazioni sulle quote delle scommesse head-to-head e spread.

2.1 Swagger OAS 3.0

Abbiamo deciso di implementare i nostri middleware come REST API server, essendo uno standard affermato per la creazione di servizi web che offre benefici come flessibilità, scalabilità e facilità di manutenzione. Per implementare e documentare le nostre API in modo efficiente, abbiamo scelto di utilizzare l'Open API Specification 3.0 (OAS 3.0).

La specifica OAS 3.0 ci ha permesso di descrivere in dettaglio tutti gli endpoint disponibili, inclusi i parametri, le risposte (sia di successo che di errore) e i tipi di oggetti restituiti, con esempi dettagliati. Abbiamo utilizzato i servizi offerti da Swagger Editor per generare automaticamente un'interfaccia utente che ci ha permesso di testare le nostre API con facilità, inizialmente basandoci solo sulla specifica testuale e ottenendo risposte fittizie, e infine utilizzando il progetto completo per ottenere risposte reali basate sulla logica di backend presente dietro ciascuno degli endpoint.

Questo approccio ci ha permesso di facilitare notevolmente lo sviluppo e la coordinazione tra i membri del team: è stato sufficiente definire la specifica testuale per permettere a chi sviluppava l'applicazione web e a chi implementava i middleware di lavorare in modo indipendente e parallelo, basandosi semplicemente sulla specifica OAS definita in precedenza. Inoltre, Swagger ci ha permesso di generare gli scheletri dei REST API server in modo automatizzato, prendendo in input solamente la specifica testuale in formato YAML. Questo ci ha dato un enorme vantaggio sia iniziale, per realizzare il progetto senza partire da zero, sia in termini di estensibilità futura. Infatti, per aggiungere nuovi endpoint, possiamo farli generare dallo stesso servizio e copiare nel nostro progetto solo le parti "nuove", con uno sforzo di implementazione minimo.

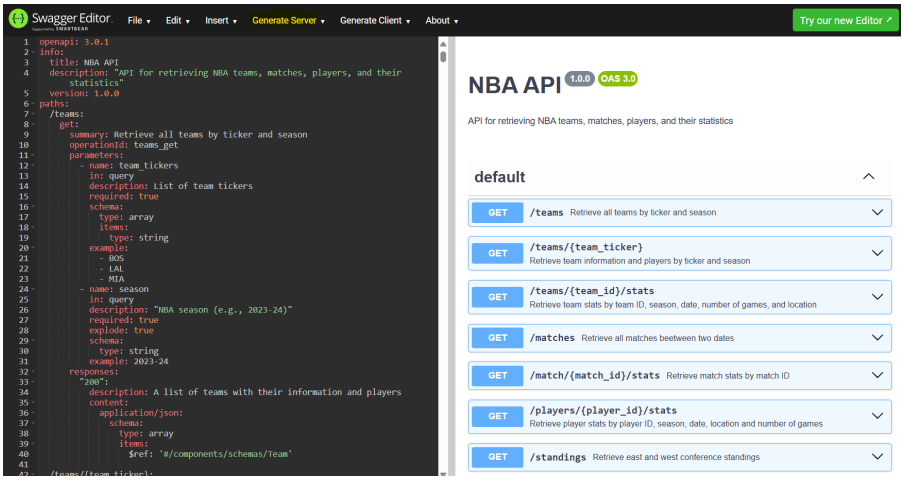


Figura 2: Interfaccia utente Swagger UI per testing e generazione server

2.2 Bet API

Partiamo dall'analisi della `bet_api`, ovvero del servizio che permette alla `web app` di ottenere tutte le informazioni necessarie relative alle quote delle partite. Alla base di questa API è presente un'ulteriore API, che analizza tutti i siti dei bookmakers di varie regioni del mondo e restituisce le quote di ogni partita, relative a diversi mercati: The Odds API.

Il concetto di middleware entra in gioco qui: avremmo potuto utilizzare direttamente l'API già esistente per ottenere tutte le informazioni di nostro interesse, ma così facendo abbiamo potuto semplificare gli endpoint, limitandoli alle sole informazioni di nostro interesse e richiedendo solo i parametri di input strettamente necessari. Inoltre, abbiamo centralizzato la gestione della sicurezza, sollevando la web app dal memorizzare e inviare ogni volta la `API key` per autenticarsi presso il servizio esterno.

`The Odds API` fornisce le quote non solo per la NBA, ma per qualsiasi sport presente su un sito di scommesse. Questo ci ha permesso di estendere ulteriormente le funzionalità della nostra applicazione e di realizzare di conseguenza il middleware `bet_api`. Abbiamo scelto di strutturare gli endpoint in modo da supportare qualsiasi tipo di sport, senza limitarci alla sola NBA di nostro interesse iniziale. Questo comporta un minimo overhead in termini di complessità, ma offre grande flessibilità.

In questo modo, la nostra web app sarà in grado, in sviluppi futuri, di supportare e visualizzare le quote di qualsiasi sport presente su un sito di scommesse, sempre però limitate ai due mercati principali head-to-head e spread, che sono comuni alla maggior parte degli sport.

2.2.1 Endpoint

Di seguito vengono presentati gli endpoint disponibili:

- `/sports/getSportGroups`
  Restituisce l'elenco di tutti i gruppi di sport disponibili. Questo endpoint supporta un parametro opzionale: `all`, che se impostato a `true` restituisce sia i gruppi sportivi in stagione (con partite attualmente in corso) che quelli fuori stagione. La risposta di successo, con codice 200, contiene un elenco di tutti i gruppi sportivi disponibili.

- `/sports/getSports`
  Restituisce l'elenco di tutti gli sport disponibili. Questo endpoint supporta due parametri opzionali: `groupName`, che consente di filtrare gli sport per nome del gruppo, e `all`, che se impostato a `true` restituisce sia gli sport in stagione che quelli fuori stagione. La risposta di successo, con codice 200, contiene un elenco di tutti le chiavi identificative degli sport disponibili.

- `/sports/getSportEvents`
  Restituisce l'elenco di tutti gli eventi disponibili per uno sport specifico. È necessario specificare il parametro obbligatorio `sportKey`, che indica la chiave dello sport degli eventi. Sono supportati due parametri opzionali: `commenceTimeFrom` e `commenceTimeTo`, che permettono di filtrare gli eventi in base al loro orario di inizio e di fine. La risposta di successo, con codice 200, contiene un elenco di tutti gli eventi disponibili.

- `/odds/head2head`
  Restituisce le migliori quote head-to-head per un evento specifico. È necessario specificare i parametri obbligatori `eventId` e `sportKey`, che indicano rispettivamente l'ID dell'evento e la chiave dello sport dell'evento. È disponibile un parametro opzionale `regions`, che consente di specificare la regione per ottenere le quote. La risposta di successo, con codice 200, contiene due elenchi delle quote head-to-head, ciascuno ordinato dalla quota più alta, uno relativo alla squadra di casa vincente e uno per quella in trasferta.

- `/odds/spreads`
  Restituisce le migliori quote head-to-head con handicap per un evento specifico. È necessario specificare i parametri obbligatori `eventId` e `sportKey`, che indicano rispettivamente l'ID dell'evento e la chiave dello sport dell'evento. È disponibile un parametro opzionale `regions`, che consente di specificare la regione per ottenere le quote. La risposta di successo, con codice 200, contiene due elenchi delle quote head-to-head con handicap, ciascuno ordinato dalla quota più alta, uno relativo alla squadra di casa vincente e uno per quella in trasferta.

Per ciascuno degli endpoint presentati sopra, sono previste risposte con codice 201 nel caso in cui non ci siano quote, eventi o sport da restituire, con codice di errore 400 per richieste errate e 500 per errori interni del server.

Da notare come, oltre al supporto per tutti gli sport grazie al parametro chiave `sportKey` (che avremmo potuto omettere inserendo sempre la chiave relativa all'NBA), vi è anche una logica per la restituzione delle quote per ogni match, ordinate dalla più vantaggiosa per l'utente, prendendo in considerazione tutti i siti di scommesse disponibili. In questo modo, tramite la web app, l'utente può avere una visione completa delle quote disponibili per ogni match e scegliere quella che ritiene più conveniente basandosi sui siti di scommesse sui quali ha un conto aperto.

Inoltre, abbiamo reso disponibili per l'utente i link ai vari siti di scommesse, in modo da facilitarne l'accesso e il controllo della quota mostrata. Per ottenere questa informazione, non disponibile direttamente da `The Odds API`, abbiamo effettuato dello scraping direttamente sul web, basandoci sul nome del sito fornito appunto dal servizio esterno.

2.2.2 Dettagli implementativi

A livello implementativo, è stato usato Spring Boot per costruire il middleware, sfruttando la potenza delle sue annotazioni per ottenere un codice altamente strutturato, pulito e flessibile.

Inoltre, è importante sottolineare l'implementazione dell'autenticazione verso `The Odds API` tramite `API Key`. Abbiamo sfruttato il concetto di *Interceptor* di Spring, un componente che si interpone prima di ogni *HttpRequest* e tramite il quale è possibile modificare i parametri e l'header della richiesta. Questo ci ha permesso di omettere sia l'url base della `Odds API` sia la `API Key` in ogni richiesta, poiché queste informazioni sono state salvate nel *context* dell'applicazione e aggiunte automaticamente ad ogni richiesta in uscita.
Quindi, in caso di aggiornamenti dell'URL o di rinnovo dell'`API Key`, sarà sufficiente modificare solo il file di consigurazioni con le suddette informazioni, senza dover cercare e modificare ogni singola chiamata, portando grandi vantaggi in termini di manutenibilità.

2.2.3 Quote storiche

Il piano gratuito di `The Odds API` non ci ha permesso purtroppo di ottenere dati su eventi e quote storiche (già passate). Questo ci ha posto dei limiti nelle ultime fasi di sviluppo poiché, essendo terminata la stagione NBA, play-off compresi, non abbiamo potuto utilizzare i dati per mostrarli all'interno dell'applicazione.
Per far fronte a questo problema, abbiamo deciso di implementare una logica che ritorna delle quote fittizie, ma verosimili, per partite passate e per le quali i bookmakers non mantengono più le quote online.

Ovviamente, ciò non toglie il fatto che per tutti gli sport e competizioni per i quali sono presenti match in corso (o più in generale per le quali i bookmakers pubblicano le quote), la logica applicativa ritorna le quote reali.

2.3 NBA Api

Il middleware `nba_api` è, come anticipato, un REST API Server che si pone l'obiettivo di interfacciarsi con la libreria offerta da swar/nba_api per ottenere informazioni statistiche relative a partite, squadre e giocatori della NBA.
Anche in questo caso, il middleware ha lo scopo di facilitare l'interazione con `swar/nba_api`, diminuendone di gran lunga la complessità ed esponendo degli endpoint mirati e di facile comprensione.

2.3.1 Endpoint

Di seguito vengono presentati gli endpoint disponibili:

- `/teams` Restituisce l'elenco di tutte le squadre relative ai ticker passati e alla stagione. Questo endpoint supporta due parametri obbligatori: `team_tickers`, che è una lista dei ticker delle squadre di cui ottenere le informazioni, e `season`, che specifica la stagione NBA di riferimento. La risposta di successo, con codice 200, contiene un elenco di tutte le squadre con le relative informazioni base e giocatori.

- `/teams/team_ticker` Restituisce le informazioni della squadra e dei giocatori per un ticker di squadra specifico e una stagione. È necessario specificare il parametro obbligatorio `team_ticker`, che indica il ticker della squadra, e `season`, che specifica la stagione NBA di riferimento. La risposta di successo, con codice 200, contiene le informazioni base della squadra e un elenco di giocatori.

- `/teams/team_id/stats` Restituisce le statistiche della squadra per ID, stagione e data limite fino al quale cosniderarle, filtrando opzionalmente per numero di partite da considerare e luogo (in casa o fuori casa). È necessario specificare i parametri obbligatori `team_id`, che indica l'ID della squadra, `season`, che specifica la stagione NBA di riferimento, e `date_to`, che specifica la data fino alla quale considerare le statistiche. Sono supportati due parametri opzionali: `last_x`, che specifica il numero di ultime partite da considerare, e `home_away_filter`, che specifica il luogo delle partite (HOME o AWAY). La risposta di successo, con codice 200, contiene un elenco di statistiche della squadra.

- `/matches` Restituisce l'elenco di tutte le partite tra due date. È necessario specificare il parametro obbligatorio `date_from`, che indica la data di inizio delle partite da considerare. È disponibile un parametro opzionale `date_to`, che specifica la data di fine delle partite da considerare (se non inserito vengono prese le sole partite del giorno `date_from`). La risposta di successo, con codice 200, contiene un elenco di tutte le partite.

- `/match/match_id/stats` Restituisce le statistiche della partita per un ID partita specifico. È necessario specificare il parametro obbligatorio `match_id`, che indica l'ID della partita, e `match_date`, che specifica la data della partita. La risposta di successo, con codice 200, contiene le statistiche della partita.

- `/players/player_id/stats` Restituisce le statistiche del giocatore per ID, stagione, data, luogo e numero di partite. È necessario specificare i parametri obbligatori `player_id`, che indica l'ID del giocatore, `season`, che specifica la stagione NBA di riferimento, e `date_to`, che specifica la data fino alla quale considerare le statistiche. Sono supportati due parametri opzionali: `last_x`, che specifica il numero di ultime partite da considerare, e `home_away_filter`, che specifica il luogo delle partite (HOME o AWAY). La risposta di successo, con codice 200, contiene le statistiche del giocatore.

- `/standings` Restituisce la classifica delle conference est e ovest. È necessario specificare il parametro obbligatorio `date`, che indica la data da considerare per valutare la classifica. La risposta di successo, con codice 200, contiene le due classifiche delle conference est e ovest.
- `/feature_vector` Restituisce il feature vector di una determinata partita. È necessario specificare i parametri obbligatori `team_ticker`, che indica il ticker della squadra, `opp_team_ticker`, che specifica il ticker della squadra avversaria, `season`, che specifica la stagione NBA, e `date`, che specifica la data della partita. La risposta di successo, con codice 200, contiene il vettore delle feature.
- `/score` Restituisce il punteggio relativo al feature vector passato. La richiesta deve contenere un `feature_vector` nel corpo della richiesta. La risposta di successo, con codice 200, contiene un punteggio "più o meno" che rappresenta la differenza di punti della squadra di casa vincente (viceversa se il numero restituito ha segno meno).

2.3.2 Dettagli implementativi

Anche questo middleware è stato realizzato tramite la generazione assistita di Swagger Editor.
Tuttavia, per poter utilizzare le librerie Python di `swar/nba_api`, abbiamo dovuto utilizzare un ambiente Python con `Flask`.

Oltre alla facilitazione già fornita da Swagger, tramite Python è stato possibile mappare gli endpoint ai metodi direttamente dalla specifica YAML, aggiungendo qualche campo in più. Questo ha aumentato di gran lunga la manutenibilità e l'estensibilità del middleware. L'aggiunta di nuovi endpoint nel corso dello sviluppo ha comportato solo la definizione dei path nel codice Swagger e l'implementazione delle funzioni con la logica applicativa necessaria.

2.3.3 Problema di `swar/nba_api`

La libreria Python che abbiamo utilizzato per ottenere le statistiche effettua le richieste al sito `nba.com` tramite l'endpoint `stats.nba.com`. L'API esposta è soggetta a un rate limiting rigoroso, per questo può succedere che alcune chiamate falliscano a causa di timeout.

Un'altro problema riscontrato consiste nel fatto che l'API blocca tutte le richieste provenienti da cloud pubblici per ragioni di sicurezza: per questa proof of concept abbiamo deciso di implementare uno switch che indirizza le richieste tramite un proxy *solo quando in esecuzione in ambiente di produzione*, ovvero su Azure Cloud; il proxy in questione è in esecuzione on-premise.

In futuro, per migliorare l'affidabilità del middleware, si potrà implementare l'API NBA ufficiale.

2.3.4 Dati di allenamento del modello

Lo scopo di questo middleware non è solo quello di fornire degli endpoint per l'applicazione web. Infatti, grazie all'implementazione di metodi comuni e all'utilizzo delle librerie di `swar/nba_api`, sono state aggiunte parti di codice volte a popolare il database di allenamento del modello di machine learning per la previsione dell'esito delle partite.

Oltre al fatto che le due task condividono gran parte delle funzioni di utilità, questa scelta è stata fatta anche per un miglioramento delle prestazioni, particolarmente a lungo termine, tramite l'uso della **cache** per i metodi della libreria.

Come sviluppo futuro, non realizzato per mancanza di tempo, si potrebbe pensare di dividere questo middleware in tre microservizi: un REST API Server che utilizza le librerie di `swar/nba_api` e che fornisce endpoint per gli altri due microservizi, il primo che funge da middleware per l'applicazione e il secondo che si occupa di ottenere i dati e popolare il database di allenamento del modello di machine learning.

Nota: Per ciascuno dei middleware appena presentati, all'interno dei packages di progetto `bet_api` e `nba_api`, sono presenti le specifica Open API 3.0 completa, utilizzate per generare gli scheletri iniziali dei due REST API server.
Inoltre, sono resi disponibili ai seguenti link GitHub: bet_oas.yaml e nba_oas.yaml

3 Modello di predizione

3.1 Introduzione

Uno degli obiettivi principali di questo progetto è stato quello di sviluppare un modello di machine learning capace di predire l'esito delle partite NBA. Questo rappresenta una sfida ambiziosa che richiede un'analisi approfondita delle statistiche delle squadre e l'applicazione di avanzate tecniche di ML. Il dataset utilizzato comprende un array di statistiche per ciascuna squadra, sia per la squadra di casa che per quella in trasferta.

Sono stati testati sette differenti modelli sul set di training: Multi-Layer Perceptron (MLP), Random Forest, Support Vector Machine (SVM), XGBoost, Regressione Lineare, Modello Bayesiano e K-Nearest Neighbors (KNN).

Per sviluppare un modello competitivo rispetto alle soluzioni attualmente disponibili sul mercato, è stato inizialmente affrontato il problema come una questione di classificazione, utilizzando un MLP. Dato che i risultati ottenuti erano promettenti, sono state poi esplorate ulteriori possibilità per migliorare le prestazioni del modello. Di conseguenza, è stato convertito il problema di classificazione in un problema di regressione, con l'obiettivo di prevedere non solo l'esito (vittoria o sconfitta) ma anche il margine di punti con cui una squadra vince o perde.

L'approccio di regressione offre un livello di dettaglio superiore, consentendo non solo di determinare il vincitore della partita, ma anche di fornire una stima più precisa delle prestazioni delle squadre. Questa dualità di approccio, classificazione e regressione, ci permette di ottenere un modello robusto e versatile, in grado di adattarsi a diverse esigenze di predizione nel contesto delle partite NBA.

In sintesi, il progetto si articola in due fasi principali:

1. Creazione e valutazione di un modello di classificazione tramite MLP.
2. Espansione del modello in un contesto di regressione per migliorare la precisione delle predizioni.

I risultati ottenuti dalle varie sperimentazioni con i diversi modelli saranno discussi nelle sezioni successive, evidenziando i vantaggi e le limitazioni di ciascun approccio.

3.2 Il Dataset

I dati utilizzati per addestrare e testare i modelli sono stati raccolti da una vasta gamma di statistiche presenti nel Database di NBA.com. Per la scelta e la selezione di queste statistiche sono state prese in considerazione numerose risorse online tra cui ad esempio il paper "Which NBA Statistics Actually Translate to Wins" di Chinmay Vayda. Queste varie ricerche ci hanno così permesso di trovare quelle che a nostro avviso solo le statistiche più rilevanti per la predizione delle partite NBA.

```
['home_team','away_team','game_id','season','date','pts_H','pts_A','winner','playoff',
'fg_pct_A','fg3_pct_A','ft_pct_A','reb_A','tot_ast_A','tot_tov_A','tot_stl_A',
'tot_blk_A','ts_pct_A','w_pct_A','w_pct_last_5_games_A','fg_pct_H','fg3_pct_H',
'ft_pct_H','reb_H','tot_ast_H','tot_tov_H','tot_stl_H','tot_blk_H',
'ts_pct_H','w_pct_H','w_pct_last_5_games_H','off_rating_A','def_rating_A',
'off_rating_H','def_rating_H','lineup_efficiency_A','lineup_efficiency_H',
'referee_name','referee_id']
```

Questo è il Dataset da noi creato ed ottenuto tramite la `nba_api`. Risulta essere un dataset molto ampio e completo, che è stato poi ridotto e lavorato in fase di preprocessing e fine-tuning dei vari modelli.

3.3 Il Modello di Classificazione

3.3.1 MLP

Introduzione al MLP Il Multi-Layer Perceptron (MLP) è una classe di reti neurali artificiali ampiamente utilizzata nei problemi di classificazione e regressione. Un MLP è composto da uno strato di input, uno o più strati nascosti e uno strato di output. Ogni nodo (o neurone) in uno strato è connesso a ciascun nodo nel successivo strato, rendendolo un tipo di rete completamente connessa. I MLP utilizzano la retropropagazione per allenare il modello, aggiornando i pesi dei neuroni in base all'errore commesso nelle predizioni.

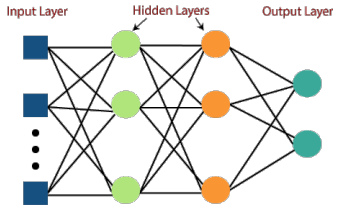


Figura 3: Multi-Layer Perceptron

Sviluppo del Modello L'idea iniziale è stata quella di creare un MLP per risolvere il problema di classificazione. Il primo passo è stato quello di effettuare il preprocessing dei dati, eliminando tutte le informazioni non necessarie per l'allenamento della rete o eliminando informazioni che rendevano il modello meno accurato(e dunque meno performante). In particolare, sono stati rimossi i seguenti campi:
`['pts_H', 'pts_A', 'referee_id', 'winner', 'home_team', 'away_team', 'referee_name', 'season', 'date']`

Inoltre, prima di poter utilizzare il vettore finale, composto dunque dalle statistiche delle due squadre, per allenare il nostro modello, è stato necessario preprocessarlo. È stato quindi usato il `MinMaxScaler`, ovvero un algoritmo che garantisce che i valori fossero compresi tra -1 e 1, il che ha portato notevoli miglioramenti sia in termini di accuracy del modello sia in termini di performance della rete.

Struttura del Modello Il modello di machine learning è stato quindi costruito utilizzando tre strati `Dense`, ovvero strati fortemente connessi in grado di apprendere dai dati in input per ottenere un preciso output. La rete neurale ottenuta è stata quindi addestrata, come consuetudine, utilizzando l'80% del Dataset suddiviso il Train Set e Validation Set. Per migliorare ulteriormente il modello è stato eseguito un processo di fine-tuning manuale per massimizzare l'accuratezza.

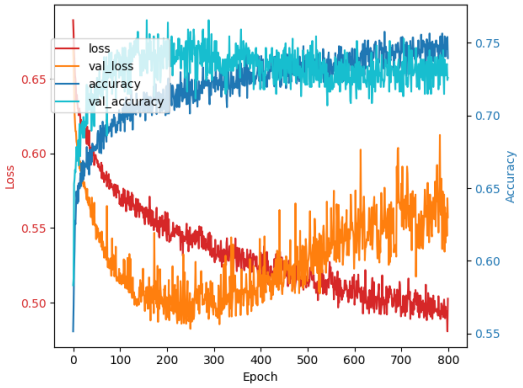


Figura 4: Pre Fine-Tuning e Overfitting

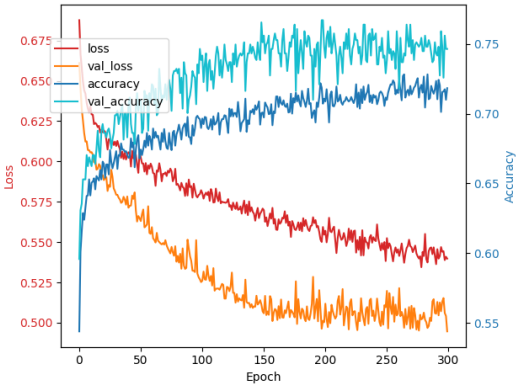


Figura 5: Post Fine-Tuning

Inizialmente, è stato utilizzato un *batch size* di 16 e sono state impostate 800 epoche per monitorare la funzione di *loss* e identificare il punto di *overfitting* del modello. Attraverso questo processo, è stato determinato che i parametri ideali per l'allenamento erano 300 epoche e un *batch size* di 32.

Risultati Prima di poter analizzare questo risultato occorre precisare che l'accuracy fornita dai modelli citati online risulta essere tra il 62% ed il 66%. In relazione a ciò è presente un interessante articolo, redatto dalla Bryant University, nel quale presentano il loro modello che possiede una accuracy del 65%.

In quest'ottica è possibile dire che il nostro modello finale ha ottenuto un'accuratezza estremamente competitiva, ottenendo una accuracy del 69,92%, dimostrando così la validità dell'approccio MLP nel contesto della classificazione delle partite NBA. Questi risultati indicano che il nostro modello è in grado di fornire predizioni accurate (7 volte su 10) sulla vittoria o sconfitta delle squadre, basandosi sulle statistiche pre-partita.

In sintesi, l'implementazione del MLP come modello di classificazione ha fornito una solida base per ulteriori miglioramenti e ottimizzazioni, consentendo di esplorare ulteriori tecniche e modelli per migliorare le capacità predittive.

```
Results MLP:
    Loss: 0.5629295706748962
    Accuracy: 0.6992385983467102
```

3.4 I Modelli di Regressione

Come anticipato, il problema di classificazione è stato convertito in un problema di regressione, in modo da testare nuovi approcci e verificare se il modello ottenuto potesse essere ulteriormente migliorato. Tutti i modelli analizzati in questa sezione sono stati prima ottimizzati utilizzando una `GridSearch` per ottenere la combinazione ideale di iperparametri per ciascun modello.
Di seguito, verranno quindi analizzati i vari modelli e i risultati ottenuti, in ordine crescente di accuracy fornita.

3.4.1 Analisi delle metriche

Nel valutare le prestazioni del modello di regressione per la predizione dei risultati delle partite NBA, sono stati utilizzati diversi indicatori chiave di errore e precisione.

- **Mean Absolute Error (MAE):** Il MAE misura la deviazione media tra le predizioni del modello e i valori reali. Nel nostro caso, il MAE indica dunque l'errore medio tra la differenza di punti predetti dal modello e i risultati reali delle partite.
- **Mean Squared Error (MSE):** Il MSE misura la media dei quadrati delle differenze tra predizioni e valori reali. Il MSE evidenzia dunque una variabilità nelle differenze tra predizioni e valori reali rispetto al MAE.
- **Root Mean Squared Error (RMSE):** Il RMSE è la radice quadrata del MSE ed è espresso nelle stesse unità della variabile target. Nel nostro caso, il RMSE indica che in media le predizioni del modello hanno un errore di una certa quantità di punti rispetto ai risultati reali.
- **Sign Accuracy:** La Sign Accuracy rappresenta la precisione del modello nel predire correttamente la direzione (positiva o negativa) delle differenze tra predizioni e valori reali. Questa metrica misura se il segno tra la differenza dei punti predetti e quella reale è concorde, in modo da prevedere quindi il vincitore di una partita indipendentemente dalla differenza dei punti.

Queste metriche forniscono una valutazione completa delle performance del modello di regressione, evidenziando sia l'errore medio delle predizioni che la sua capacità di predire correttamente l'esito delle partite NBA. Il confronto tra MAE, MSE e RMSE fornisce una panoramica dell'accuratezza delle predizioni a diversi livelli di dettaglio, mentre la Sign Accuracy fornisce una misura della precisione nella classificazione della direzione delle differenze.

3.4.2 KNN

Il K-Nearest Neighbors (KNN) è un algoritmo di machine learning utilizzato per problemi di classificazione e regressione. Nel contesto della regressione, KNN prevede il valore di una variabile target basandosi sui valori medi dei k vicini più prossimi nel dataset di addestramento. La vicinanza tra i punti dati è solitamente calcolata tramite una metrica di distanza, come la distanza euclidea. KNN è semplice da implementare e interpretare, ma può essere computazionalmente costoso per dataset di grandi dimensioni.

Il modello KNN è stato valutato utilizzando diverse metriche di errore, ottenendo i seguenti risultati:

```
Mean Absolute Error (MAE): 11.620278833967047
Mean Squared Error (MSE): 215.66732572877058
Root Mean Squared Error (RMSE): 14.6856162869922
Sign Accuracy: 0.6172370088719898
```

Il modello ha raggiunto un'accuratezza del 61,72%.

Sono stati estratti 10 match casuali dal test set per valutare le prestazioni del modello, ottenendo i seguenti risultati:

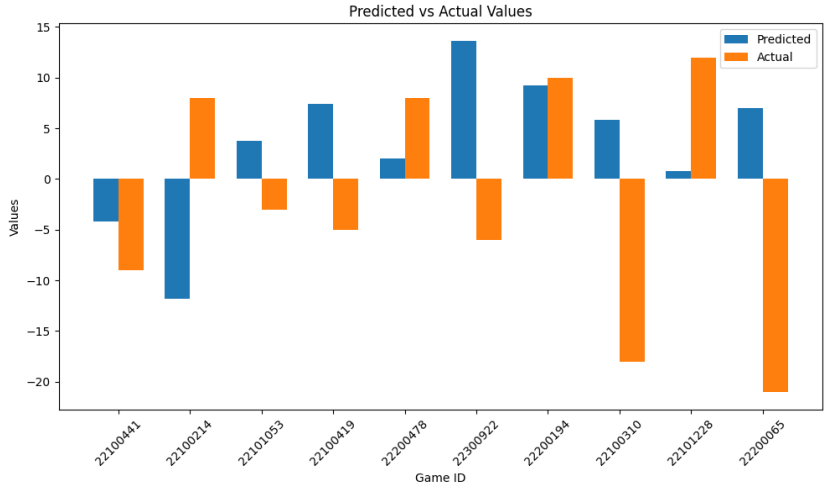


Figura 6: Comparazione differenza punti predetti rispetto a reali in KNN

3.4.3 SVM

Il modello SVM ha mostrato un'accuratezza migliore rispetto ad altri approcci. SVM (Support Vector Machine) è un algoritmo di apprendimento supervisionato utilizzato per la classificazione e la regressione. Esso cerca di trovare il miglior iperpiano o separatore tra i punti dei dati delle diverse classi. Nella figura sottostante è rappresentato un esempio di SVM con due classi.

Questo modello ha ottenuto i seguenti risultati nei test:

```
Mean Absolute Error (MAE): 10.360566704859455
Mean Squared Error (MSE): 173.73226432418772
Root Mean Squared Error (RMSE): 13.18075355676555
Sign Accuracy: 0.6806083650190115
```

Il modello ha raggiunto un'accuratezza del 68.06%. Inoltre, sono stati estratti 10 match casuali dal set di test per valutare ulteriormente le prestazioni del modello:

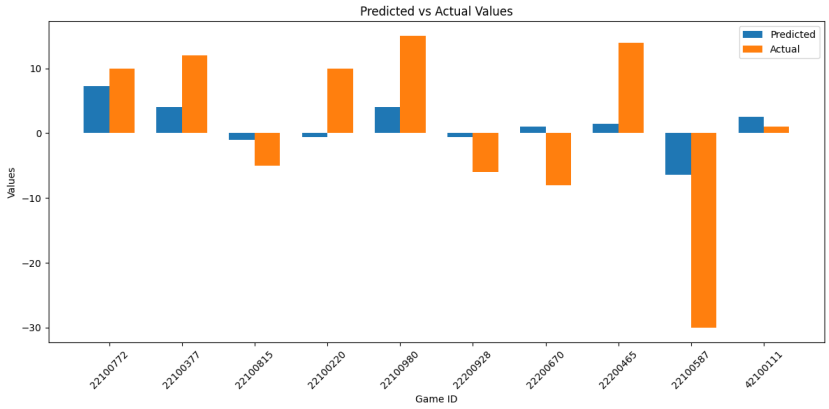


Figura 7: Comparazione differenza punti predetti rispetto a reali in SVM

Questi risultati dimostrano che il modello SVM ha una buona capacità predittiva per l'esito delle partite NBA, con un'ottima precisione e una buona gestione della variazione nei dati di test.

3.4.4 Random Forest

Il Random Forest è un metodo di apprendimento *ensemble* utilizzato per la classificazione e la regressione. Esso costruisce diversi alberi decisionali e li combina per ottenere previsioni più accurate. In un random forest, solo un sottoinsieme delle caratteristiche è considerato dall'algoritmo per dividere un nodo. Il modello classificherà anche l'importanza di ciascuna caratteristica nel prendere la decisione finale. Nella figura sottostante c'è un esempio di random forest con due alberi.

Questo modello ha ottenuto risultati soddisfacenti nei test, come mostrato di seguito:

```
Mean Absolute Error (MAE): 9.944223032615545
Mean Squared Error (MSE): 159.938928930258
Root Mean Squared Error (RMSE): 12.646696364278617
Sign Accuracy: 0.6958174904942965
```

Il modello ha raggiunto un'accuratezza del 69.58%. Inoltre, sono stati estratti 10 match casuali dal set di test per valutare ulteriormente le prestazioni del modello:

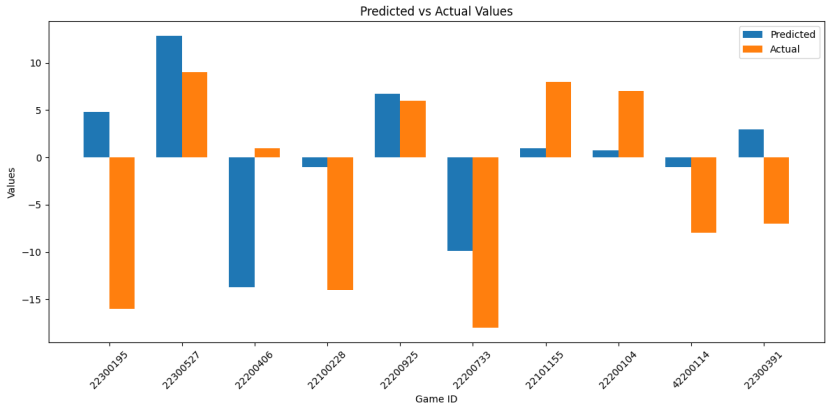


Figura 8: Comparazione differenza punti predetti rispetto a reali in Random Forest

Questi risultati indicano che anche il modello Random Forest si è dimostrato essere efficace nel predire l'esito delle partite NBA, con un'elevata accuratezza e una buona capacità di generalizzazione su nuovi dati di test.

3.4.5 XGBoost

XGBoost è un algoritmo di boosting estremamente popolare e efficace utilizzato per la classificazione e la regressione. Utilizza un insieme di alberi decisionali deboli, chiamati "weak learners", e li combina per migliorare progressivamente le prestazioni del modello.

Questo modello ha ottenuto i seguenti risultati nei test:

```
Mean Absolute Error (MAE): 9.6111730557072
Mean Squared Error (MSE): 149.89162676035912
Root Mean Squared Error (RMSE): 12.243023595515902
Sign Accuracy: 0.7046894803548795
```

Il modello ha raggiunto un'accuratezza del 70.47%. Inoltre, sono stati estratti 10 match casuali dal set di test per valutare ulteriormente le prestazioni del modello:

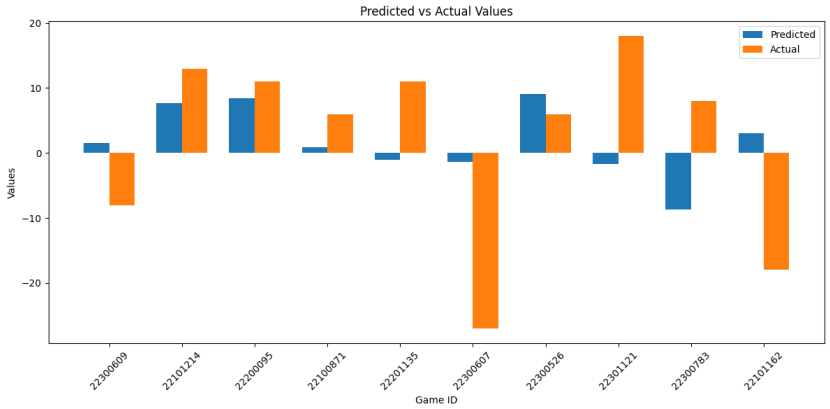


Figura 9: Comparazione differenza punti predetti rispetto a reali in XGBoost

Questi risultati indicano che il modello XGBoost risulta essere altamente efficace nel predire l'esito delle partite NBA. L'utilizzo di boosting ha permesso al modello di migliorare costantemente la precisione delle predizioni, rendendolo infatti una scelta potente per problemi complessi, come quello delle previsioni sportive.

3.4.6 Bayesian

Il modello Bayesian utilizza l'inferenza Bayesiana per stimare parametri incogniti. È particolarmente utile quando si hanno dati limitati e si desidera incorporare conoscenze pregresse o informative nel modello.

Questo modello ha ottenuto i seguenti risultati nei test:

```
Mean Absolute Error (MAE): 9.684169695449112
Mean Squared Error (MSE): 151.8468062429373
Root Mean Squared Error (RMSE): 12.322613612498662
Sign Accuracy: 0.7072243346007605
```

Il modello ha raggiunto un'accuratezza del 70.72%. Inoltre, sono stati estratti 10 match casuali dal set di test per valutare ulteriormente le prestazioni del modello:

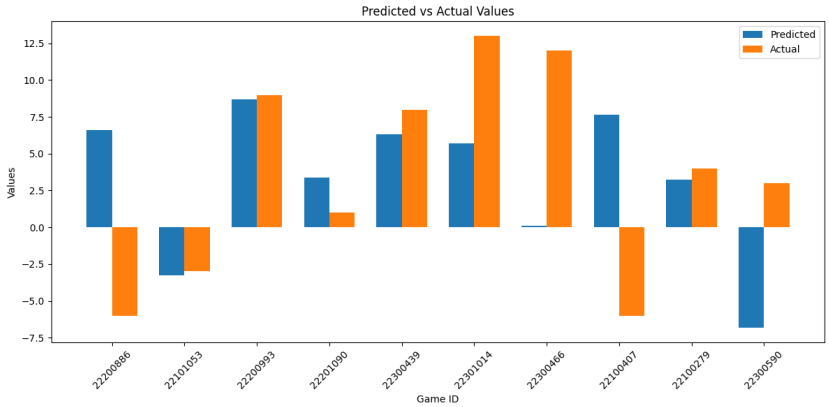


Figura 10: Comparazione differenza punti predetti rispetto a reali in Bayesian

Questi risultati validano la qualità del modello Bayesian, che ha mostrato una robusta gestione dell'incertezza nei dati di test. L'approccio basato sull'inferenza Bayesiana ha permesso al modello di integrare informazioni a priori con i dati osservati, migliorando così le performance complessive delle previsioni.

3.4.7 Linear Regression

La regressione lineare è un modello di machine learning che cerca di trovare la relazione lineare migliore tra una variabile dipendente (target) e una o più variabili indipendenti (features). È un metodo semplice ma potente per la predizione numerica.

Questo modello ha ottenuto i seguenti risultati nei test:

```
Mean Absolute Error (MAE): 9.673524553632301
Mean Squared Error (MSE): 151.58868540348476
Root Mean Squared Error (RMSE): 12.312135696274824
Sign Accuracy: 0.7135614702154626
```

Il modello ha raggiunto un'accuratezza del 71.36%. Inoltre, sono stati estratti 10 match casuali dal set di test per valutare ulteriormente le prestazioni del modello:

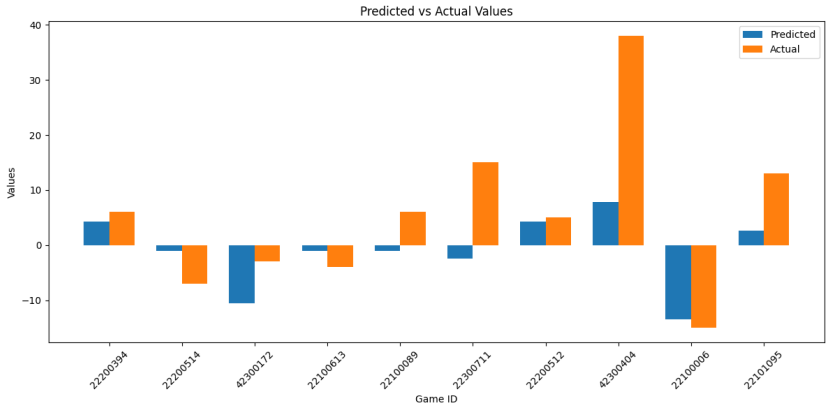


Figura 11: Comparazione differenza punti predetti rispetto a reali in Linear Regression

Questi risultati indicano che il modello di Regressione Lineare ha fornito la migliore capacità predittiva, con un'efficace modellazione della relazione lineare tra le variabili di input e il target. La semplicità e la trasparenza della regressione lineare lo rendono un buon punto di partenza per esplorare i modelli di previsione numerica nelle analisi sportive.

3.5 I Risultati

È dunque possibile ora analizzare i dati in maniera unitaria e completa. Per effettuare ciò occorre suddifidere l'analisi dei risultati in due distinte sezioni, una per analizzare i risultati inerenti al problema di classificazione: Vittoria o Sconfitta, ed un'altra per analizzare i risultati del problema di regressione.

Inerentemente al problema di classificazione, per gli algoritmi di regressione è stata creata una nuova metrica fittizia denominata sign accuracy che ci ha permesso di convertire il problema di regressione in classificazione. Comparando così le varie sign accuracy ottenute con i vari algoritmi, otteniamo il seguente risultato.

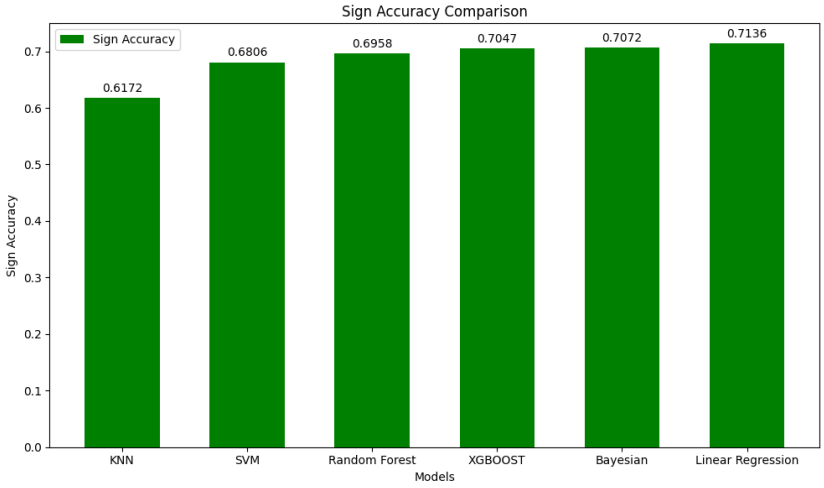


Figura 12: Sign accuracy a confronto

È possibile notare che l'algoritmo che ci fornisce una sign accuracy migliore è il Linear Regression che totalizza una precisione del 71,36%. Questo è poi seguito dal Bayesian Regressor con 70,72% e dal XGBoost Regressor con 70,46%. Questi algoritmi sono quindi riusciti ad ottenere, a parità di input, un risultato migliore anche del modello MLP definito inizialmente. In ultima posizione, vediamo come il K-Nearest Neighbor abbia le prestazioni peggiori.

Concentrandoci ora sull'analisi del problema di regressione e dunque sull'analisi della differenza tra i punti predetti ed i punti reali otteniamo risultati leggermente diversi. A tal proposito è stata messa a confronto la metrica `Mean Absolute Error`, che fornisce informazioni riguardanti l'errore medio dei vari modelli creati. Confrontando quindi questa metrica sono stati ottenuti i seguenti risultati.

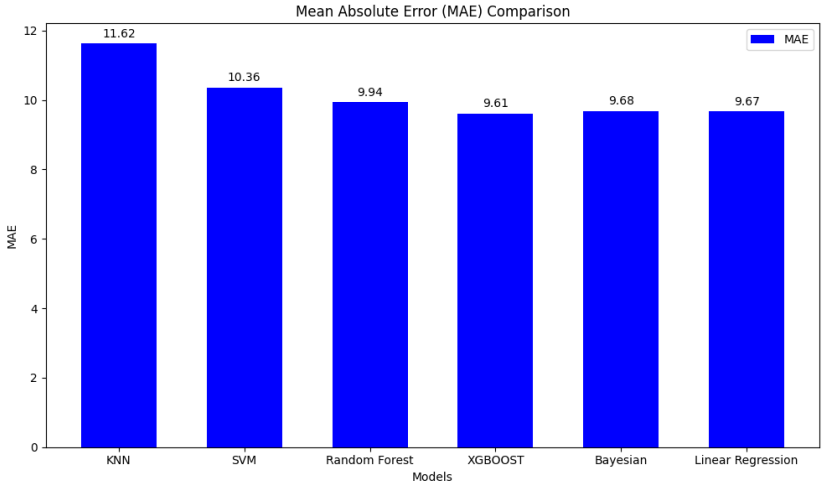


Figura 13: Caption

L'algoritmo più performante in termini di regressione risulta essere XGBoost con 9,61 punti di scarto medio, seguito poi dalla Linear Regression con 9,67 punti e dal Bayesian con 9,68. Il K-Nearest Neighbor, al contrario, continua a confermarsi il meno performante con 11,62 punti di scarto medio.

3.6 Importanza Metriche

I modelli di regressione sono stati poi confrontati per comprendere quali fossero le features che ciascuno di essi ha usato ed analizzato maggiormente per prevedere la differenza di punteggio delle partite. Nela figura sottostante sono stati messi a confronto i 3 modelli migliori (ovvero bayesian, xgboost e linear regression) ed il modello peggiore, ovvero KNN.

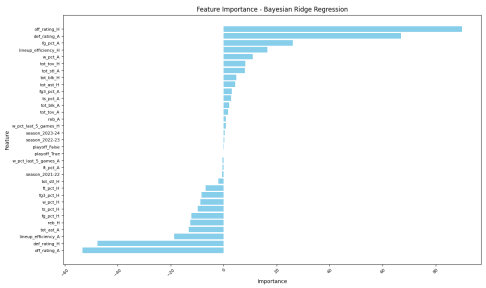


Figura 14: Bayesian

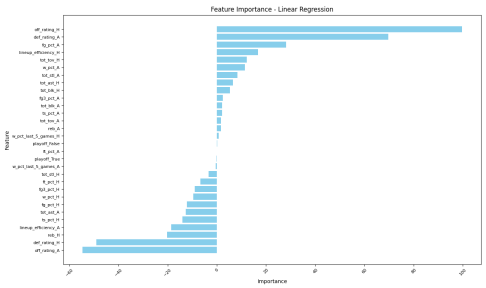


Figura 15: Linear Regression

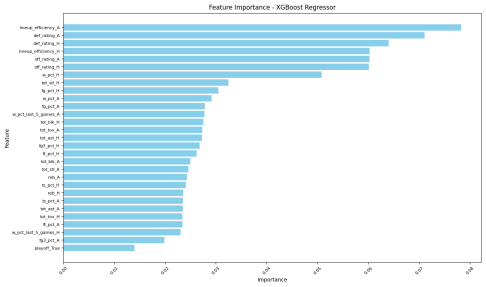


Figura 16: XGBoost

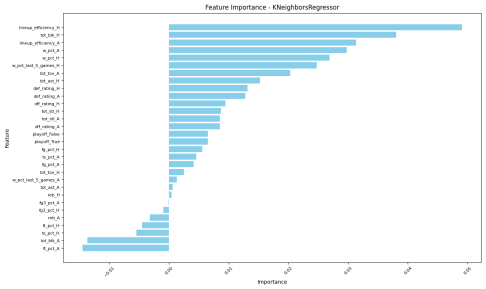


Figura 17: K-Nearest Neighbors

Si può vedere come i 3 modelli migliori (ovvero bayesian, xgboost e linear regression) valutino maggiormente, e dunque prediligano, le seguenti metriche:

- lineup_efficiency_A
- lineup_efficiency_H
- off_rating_A
- off_rating_H
- def_rating_H
- def_rating_A

Al contrario, il modello di regressione K-Nearest Neighbors mantiene solo le lineup_efficiency ed altre features, a scapito di: off_rating_A off_rating_H def_rating_A def_rating_H

4 Webapp

4.1 Introduzione

Rendere disponibili agli utenti finali i dati ottenuti dai servizi sopra presentati è responsabilità di una apposita applicazione web.

Questa dovrà interfacciarsi con i vari middleware attraverso le API disponibili, per ottenere i dati da presentare all'utente in maniera il più possibile pratica e ordinata.

La web app sarà accessibile da browser e dovrà supportare anche dispositivi mobili, quindi reagire in maniera *responsive* a schermi di dimensioni molto diverse.

4.2 Tecnologie utilizzate

Essendo già a conoscenza dell'ambiente finale di deployment, la scelta delle tecnologie è stata presa top-down, per assicurare completa compatibilità e supporto.

La piattaforma Azure mette a disposizione diverse opzioni per l'hosting di una web app, che può essere ad esempio ospitata su una tradizionale macchina virtuale o come container app.

Le esigenze del nostro servizio dal punto di vista dell'interfaccia utente sono modeste, in quanto si tratta di una semplice vetrina per le informazioni disponibili, senza bisogno di autenticazione o salvataggio di dati in uno storage dedicato.

Delinandosi come applicazione prettamente front-end, la scelta è ricaduta su una Static Web App. Questo servizio Azure è pensato per ospitare web app full-stack che non hanno un tradizionale backend, e offre un supporto integrato ad API sotto forma di Azure Functions o Container App.

Le SWA sono costruite usando framework web come Angular, React o Vue, tecnologie per cui non è richiesto rendering lato server: le applicazioni sono costituite interamente da asset statici e file html/css/javascript. Questi sono serviti non da un web server tradizionale, bensì gestiti da Azure e distribuiti globalmente.

Il deployment delle nuove versioni avviene in maniera automatica a partire da cambiamenti in una repository di codice, come GitHub mediante *Actions* o Azure stesso grazie alla *DevOps Pipeline*. Il workflow è auto-generato da Azure e necessita pochi parametri di configurazione oltre alla repository (ed eventualmente branch) di riferimento. Questo modus operandi è, per design, molto vicino al flusso di lavoro tipico dello sviluppatore e permette uno sviluppo organico seguendo la pratica del *Continuous Integration / Continuous Delivery*.

La scelta del framework javascript è ricaduta su Angular, sviluppato da Google e arrivato a luglio 2024 alla versione 18. È un framework open source maturo e facile da usare, adatto ad applicazioni web multipiattaforma. È direttamente supportato da Azure Static Web Apps e rispetta la caratteristica richiesta di rendering lato browser.

4.3 Angular

Le app Angular sono scritte in typescript e devono seguire una precisa struttura. Sono *single-page-applications* organizzate per componenti con singola responsabilità, componibili all'interno di file `.html` estesi per supportare tag aggiuntivi.

I componenti sono definibili direttamente in un unico file typescript ed esportati come classi. Sono identificati da un selettore, reso poi disponibile sotto forma di tag, e devono presentare un template HTML per istruire il rendering, che può anche essere inserito sotto forma di link a un file a sé.

È costume velocizzare lo sviluppo delle applicazioni web affidandosi a librerie di terze parti aggiuntive che mettano a disposizione componenti pronti all'uso. Nel caso di Courtsight è stato scelta **PrimeNG**. Questa libreria molto completa è sviluppata da PrimeTek, che l'ha declinata per tutti i

maggiori framework. Alla base di queste diverse librerie c'è il loro sistema *PrimeFlex*, che permette una facile gestione della distribuzione dei componenti a schermo semplificando le direttive CSS "flex".

4.4 Contenuto webapp

Homepage La pagina principale del sito presenta all'utente i vari sport supportati. L'integrazione con la `bet_api` ci può permettere in un futuro di mostrare le quote anche per sport diversi dal basket americano, che per ora rappresenta il principale focus.

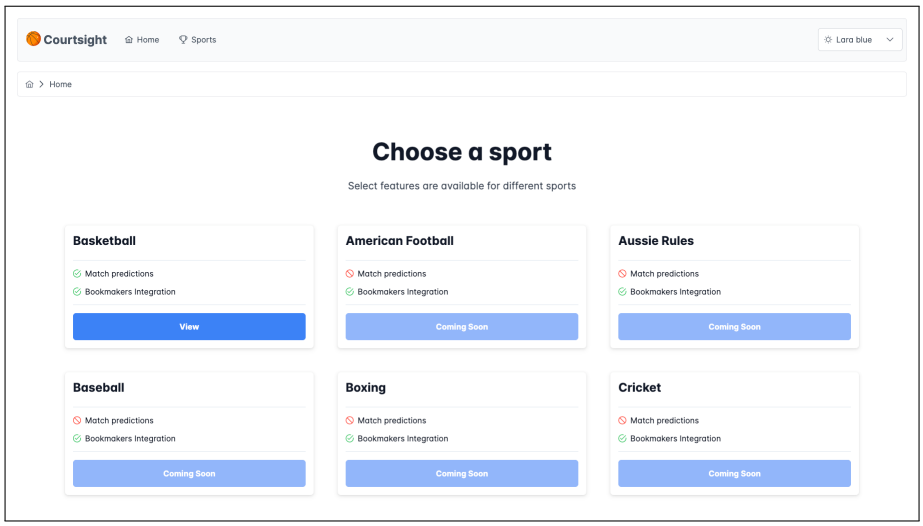


Figura 18: Homepage

Home - Basket La schermata home della sezione basket offre l'elenco delle partite della settimana attuale e la classifica aggiornata al periodo esposto, divisa tra le due conference NBA: East e West Coast.

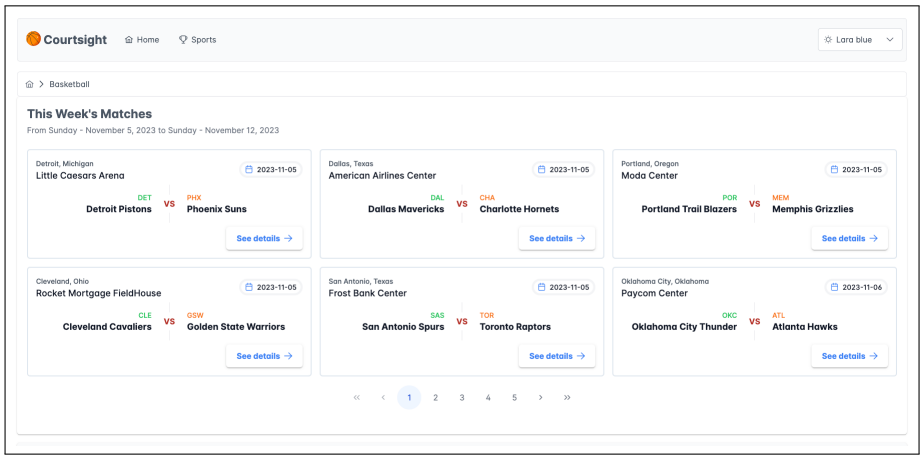
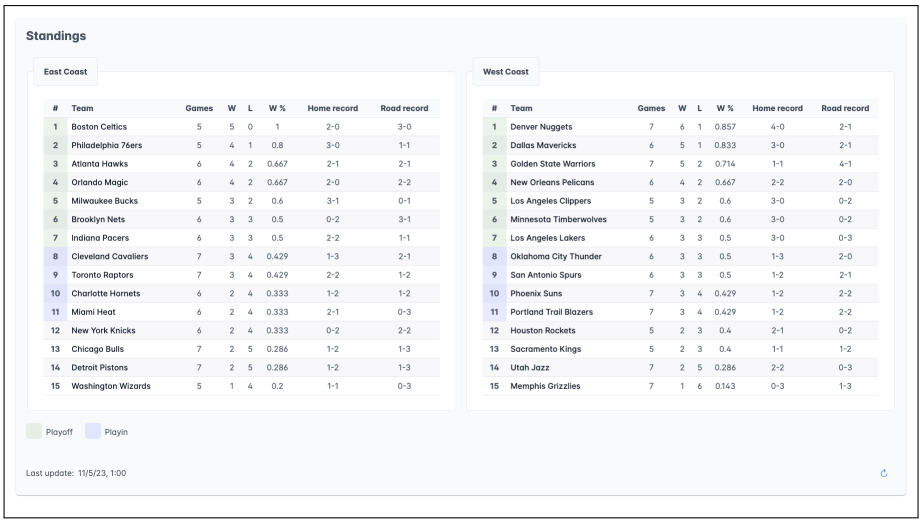


Figura 19: Lista dei prossimi match



#	Team	Games	W	L	W %	Home record	Road record
1	Boston Celtics	5	5	0	1	2-0	3-0
2	Philadelphia 76ers	5	4	1	0.8	3-0	1-1
3	Atlanta Hawks	6	4	2	0.667	2-1	2-1
4	Orlando Magic	6	4	2	0.667	2-0	2-2
5	Milwaukee Bucks	5	3	2	0.6	3-1	0-1
6	Brooklyn Nets	6	3	3	0.5	0-2	3-1
7	Indiana Pacers	6	3	3	0.5	2-2	1-1
8	Cleveland Cavaliers	7	3	4	0.429	1-3	2-1
9	Toronto Raptors	7	3	4	0.429	2-2	1-2
10	Charlotte Hornets	6	2	4	0.333	1-2	1-2
11	Miami Heat	6	2	4	0.333	2-1	0-3
12	New York Knicks	6	2	4	0.333	0-2	2-2
13	Chicago Bulls	7	2	5	0.286	1-2	1-3
14	Detroit Pistons	7	2	5	0.286	1-2	1-3
15	Washington Wizards	5	1	4	0.2	1-1	0-3

#	Team	Games	W	L	W %	Home record	Road record
1	Denver Nuggets	7	6	1	0.857	4-0	2-1
2	Dallas Mavericks	6	5	1	0.833	3-0	2-1
3	Golden State Warriors	7	5	2	0.714	1-1	4-1
4	New Orleans Pelicans	6	4	2	0.667	2-2	2-0
5	Los Angeles Clippers	5	3	2	0.6	3-0	0-2
6	Minnesota Timberwolves	5	3	2	0.6	3-0	0-2
7	Los Angeles Lakers	6	3	3	0.5	3-0	0-3
8	Oklahoma City Thunder	6	3	3	0.5	1-3	2-0
9	San Antonio Spurs	6	3	3	0.5	1-2	2-1
10	Phoenix Suns	7	3	4	0.429	1-2	2-2
11	Portland Trail Blazers	7	3	4	0.429	1-2	2-2
12	Houston Rockets	5	2	3	0.4	2-1	0-2
13	Sacramento Kings	5	2	3	0.4	1-1	1-2
14	Utah Jazz	7	2	5	0.286	2-2	0-3
15	Memphis Grizzlies	7	1	6	0.143	0-3	1-3

Figura 20: Classifica del campionato

Basket - Squadra Selezionando il nome di una squadra è possibile vedere i dettagli sulla sua data di fondazione e arena di casa, oltre all'elenco dei giocatori e le prossime partite che disputerà.

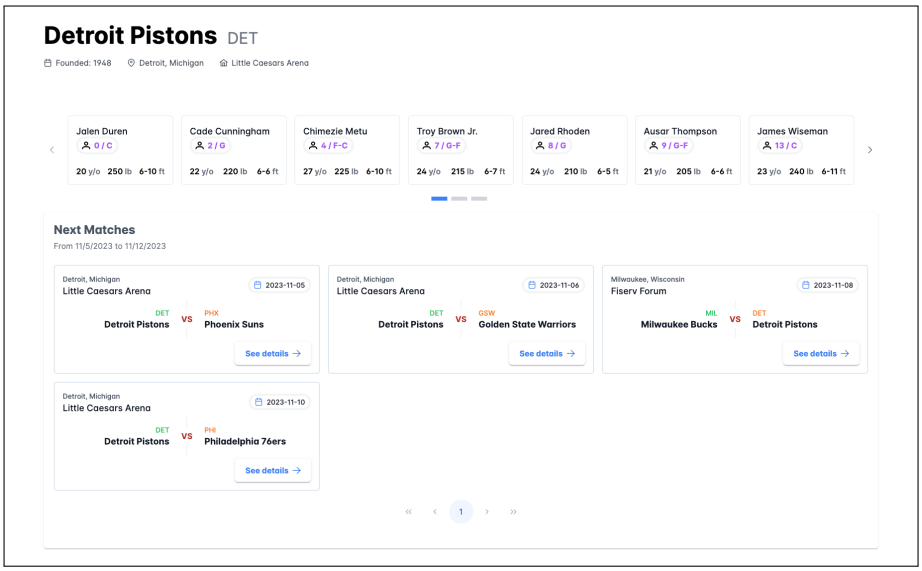


Figura 21: Dettagli squadra

Basket - Partita La schermata dedicata ai match è il vero luogo di incontro di tutti i microservizi sviluppati. Oltre a presentare il punteggio e le statistiche del match, è disponibile l'elenco di tutti i bookmaker e le quote della partita. Inoltre, premendo un bottone è possibile interrogare il modello di ML per predire il risultato del match.

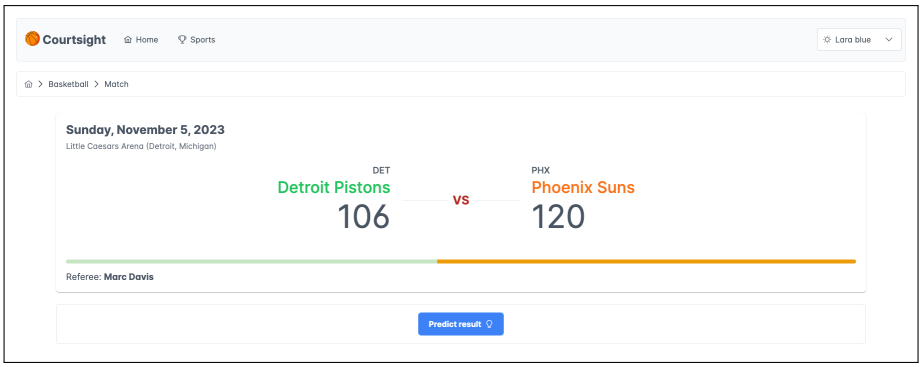


Figura 22: Informazioni match

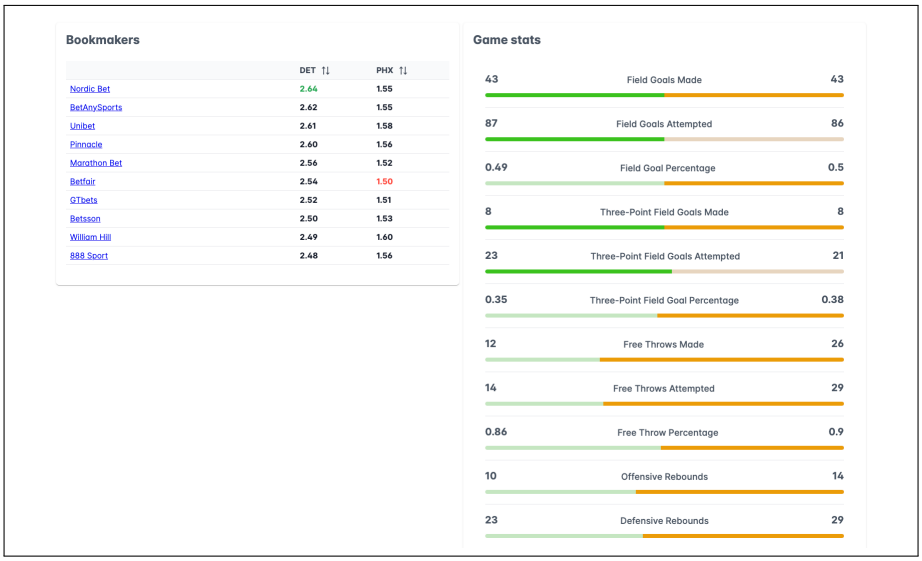


Figura 23: Statistiche e quote match

Mediante il sistema di routing di Angular, le pagine sono definite ognuna da un diverso componente, inserito dinamicamente nel root della applicazione a seconda dell'url richiesto.

4.5 Struttura dell'applicazione

In questa sezione verrà delineato l'intero elenco di componenti sviluppati. Nelle applicazioni Angular, quando il numero di componenti inizia a crescere, è bene suddividerli in cartelle organizzate per feature e l'elenco che segue presenta quindi la medesima strategia.

4.5.1 app

Alla base dell'applicazione web c'è un file `index.html` il cui body verrà dinamicamente aggiornato a runtime inserendo i vari componenti quando richiesto, e sarà quindi sempre presentato all'utente. `app-root` è l'unico componente indicato ed è il vero scheletro esterno dell'applicazione, in quanto verrà sempre disegnato. Ha al suo interno la barra del menu superiore, che ovviamente deve essere disponibile in ogni momento, e un contenitore fornito da Angular (`router-outlet`) per ospitare i componenti serviti dal sistema di routing. Il componente `menubar` oltre al titolo ospita anche i link ai vari sport e sulla sinistra un selettore del tema grafico da adottare, modificabile in tempo reale.

4.5.2 homepage

L'homepage presenta semplicemente all'utente l'elenco degli sport disponibili. Questi sono ottenuti dinamicamente e mostrati a schermo in base alle feature disponibili. L'elenco degli sport e il singolo item sono quindi due componenti diversi, `sport-showcase` e `sport-details`, mostrati all'interno del macro componente `home`.

4.5.3 basket/home

La pagina principale dedicata all'NBA quando caricata effettua il *fetch* delle partite della settimana che viene. Queste sono presentate come `match` all'interno di un elenco `match-list`, paginato a gruppi di sei.

Subito sotto, la classifica aggiornata alla giornata corrente è resa disponibile dal componente `standings`. Questo, data la caratteristica del campionato di avere in realtà due classifiche in base alla costa (East Conference e West Conference), presenta due diverse tabelle `standings-list`. Oltre alla semplice posizione di ogni squadra sono anche forniti tutti i dati più rilevanti, come ad esempio partite vinte o record fuori casa. Inoltre, le prime 8 righe di ogni tabella hanno una colorazione azzurra per segnalare che quelle squadre parteciperanno ai Playoff, mentre le successive 4 colorate di verde sono quelle che dovranno prima qualificarsi giocando i Play-in.

4.5.4 basket/team

Ogni team ha una pagina dedicata, accessibile premendo sul suo nome ovunque compaia all'interno dell'app. Il componente `team-details` ospita, assieme a dettagli sulla squadra come anno di fondazione e arena di casa, l'elenco delle prossime partite da disputare (riutilizzando il componente `match-list`) e la lista di tutti i giocatori. Quest'ultimo componente (`players-showcase`) è un "carosello" che mostra di ogni giocatore ruolo, età, numero di maglia, peso e altezza.

4.5.5 basket/match

La schermata dedicata ai match ha in primo piano il risultato (corrente o passato) e dati quali data, palazzetto e arbitro. Due componenti sono poi disposti subito sotto. Il primo è `bookmakers-list`, che dati gli *odds* di una partita mostra le quote per tutti i siti disponibili. Il secondo è invece `stats-table`, una lista di `stats-row` per elencare tutte le statistiche della partita (assist, rebound, falli, etc.). Al centro della pagina un bottone permette, se premuto, di dare il via alle operazioni necessarie per predire il risultato della partita, ovvero recuperare il feature vector e inviarlo all'endpoint per ottenere infine lo score. Se questo risultato è disponibile, viene allora mostrato il nome del vincitore previsto del match e lo scarto di punti previsto.

4.5.6 *time travel*

Per permettere di visualizzare partite vere anche a campionato terminato, un semplice selettore di data è reso disponibile in fondo alle pagine dove sarebbe invece richiesta la data odierna. È possibile passare al componente una funzione di callback per essere notificati quando il cambiamento di data avviene. In questo modo è possibile gestire il reaload di tutti i componenti dopo aver effettutato un fetch dei nuovi dati.

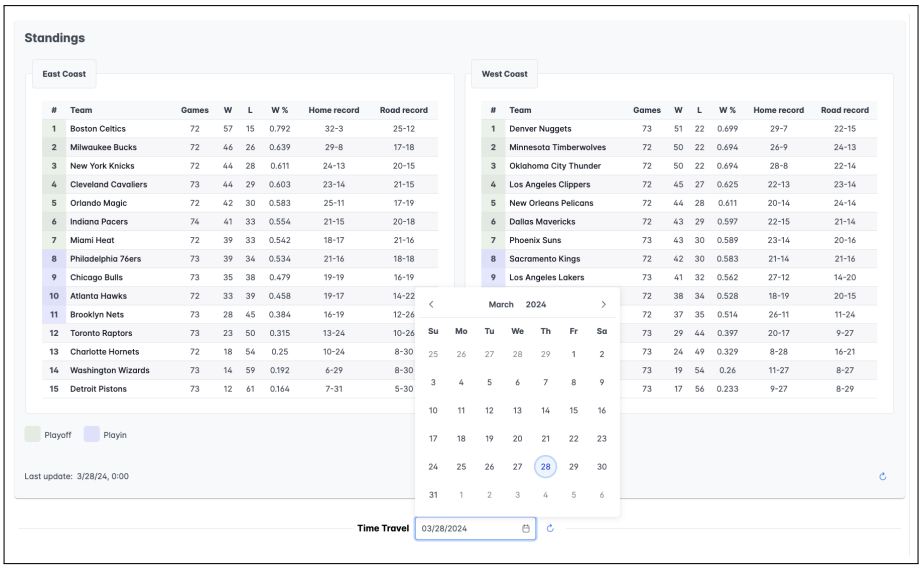


Figura 24: Selezione data per "time travel"

4.6 Servizi

Le applicazioni Angular sfruttano la Dependency Injection per rendere disponibili feature e dati ai vari componenti solo se esplicitamente richiesti. Una serie di *services* sono definiti per interfacciarsi alle API e nascondere i dettagli in merito a chiamate e ottenimento dei dati. I servizi sono divisi in file in base all'ambito di utilizzo anche se condividono in molti casi l'API a cui fanno riferimento. In particolare sono:

- SportsService: ottiene dalla `bet_api` l'elenco degli sport supportati.
- OddsService: ottiene dalla `bet_api` le quote per un dato match.
- StandingsService: ottiene dalla `nba_api` le classifiche aggiornate alla data richiesta.
- MatchesService: ottiene dalla `nba_api` l'elenco delle partite in un intervallo di date, le statistiche relative a una data partita e il feature vector per un determinato incontro.
- TeamsService: ottiene dalla `nba_api` informazioni generali relative a un team dato il suo "ticker" o le statistiche complete in base all'id.
- PlayersService: ottiene dalla `nba_api` le statistiche di un dato giocatore.
- PredictionsService: ottiene dal modello di Machine Learning la previsione dello score dato un certo feature vector.

Tre ulteriori servizi sono definiti per mettere a disposizione dell'intera app impostazioni e feature condivise, ovvero:

- TimeTravelService, per permettere di cambiare la data "odierna" di riferimento in tutta l'applicazione.
- BreadcrumbService, per la navigazione mediante breadcrumb.
- ThemeService, per il cambiamento dinamico del tema grafico.

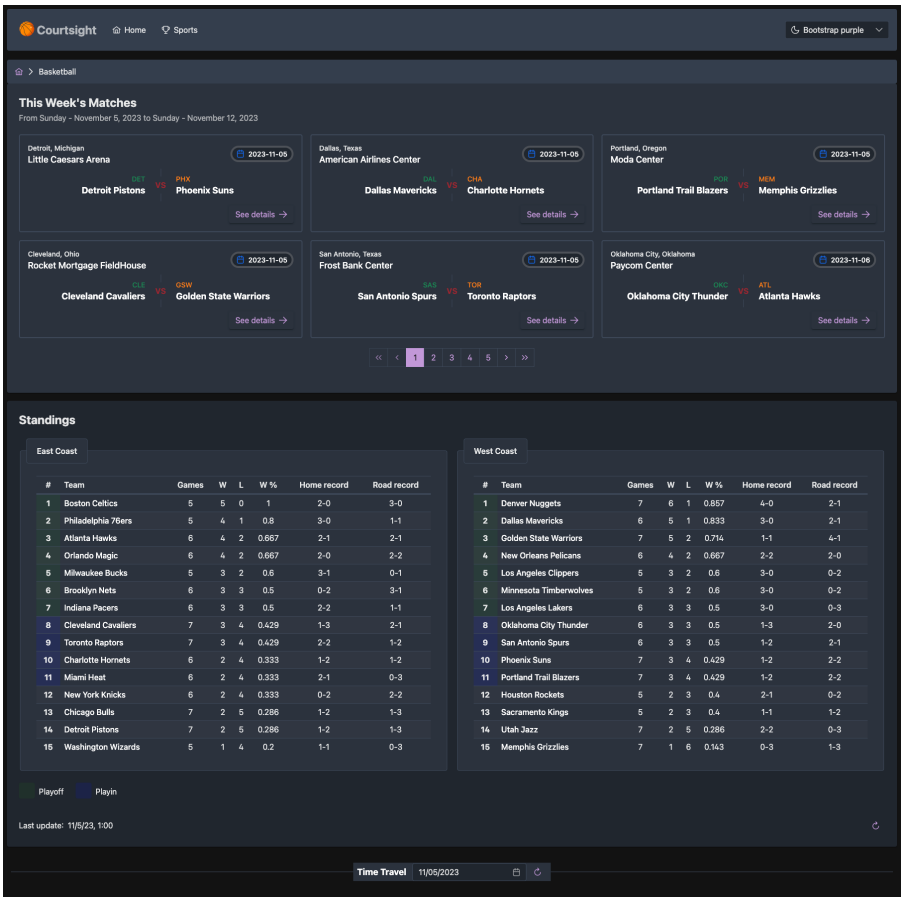


Figura 25: Dark mode selezionato dinamicamente

5 Deployment

In questa sezione verranno trattate le scelte in materia di deployment dell'architettura: discuteremo le scelte effettuate per i servizi Azure utilizzati, la configurazione di tali servizi e l'automazione del processo di deployment, così come le soluzioni e strategie adottate per garantire scalabilità e affidabilità dell'applicazione.

5.1 Componenti

Per riferimento si riportano di seguito i componenti principali che costituiscono l'applicazione.

- Middleware NBA Api
- Middleware Bet Api
- Frontend Applicazione Web
- Modello Machine Learning

Come detto in fasi precedenti, abbiamo preso la decisione di creare dei middleware per l'interazione con le API delle quote e delle statistiche NBA al fine di creare un'interfaccia che permette alla applicazione di astrarre dal funzionamento dei servizi sottostanti che forniscono i dati. Inoltre, questo approccio ci permetterà in futuro di modificare i servizi sottostanti da cui recuperiamo i dati per migliorare le prestazioni dell'applicazione.

L'applicazione deve essere in grado di adattarsi ad aumenti inaspettati del carico di richieste, per cui è stato necessario utilizzare servizi che supportassero funzionalità di scalabilità automatiche sia all'aumento del carico, sia alla sua diminuzione per mantenere il più possibile contenuti i costi del servizio.

Per la distribuzione dei due middleware abbiamo deciso di utilizzare container OCI in quanto sono lo strumento più adatto a effettuare il deployment di una architettura a microservizi.

5.2 Servizi Azure

Microsoft Azure offre una vasta gamma di servizi per qualsiasi tipo di deployment e la scelta del giusto servizio per un certo componente è cruciale per bilanciare costi di operazione e massimizzare l'efficienza e automazione del processo di deployment, minimizzando le interruzioni del servizio.

5.2.1 Container

Microsoft Azure offre varie opzioni per il deployment di componenti basati su container OCI, da cluster Kubernetes parzialmente gestiti ad ambienti completamente gestiti:

- Azure Container Instances (ACI)
- Azure Container Apps (ACA)
- Azure Kubernetes Services (AKS)
- Azure App Service (AAS)

Questi servizi hanno diversi livelli di complessità delle opzioni di configurazione. In particolare, AKS è il servizio che offre la maggiore granularità nelle opzioni di gestione dei cluster di container: si tratta di una istanza di Kubernetes gestita da Azure che fornisce una piattaforma di orchestrazione completa adatta a deployment molto complessi.

Durante la valutazione delle opzioni abbiamo escluso il servizio AAS in quanto si tratta di un servizio non nativamente improntato a deployment di applicazioni sotto forma di container: non offre infatti le opzioni di gestione della scalabilità offerte da altri servizi concepiti per container.

ACI offre una esperienza di deployment molto semplificata, al costo di non dare la possibilità di creare regole di scaling e replicazione personalizzate, facendo gestire ogni aspetto del deployment da Azure. ACA si pone a metà strada tra la semplicità di ACI e la complessità di AKS, in quanto nasconde dettagli di gestione del cluster Kubernetes, ma permette comunque di specificare regole e limiti di scaling con abbastanza granularità.

Abbiamo infine deciso di utilizzare Azure Container Apps per il deployment dei due microservizi.

Abbiamo distribuito i componenti middleware in un unico Azure Container App Environment, il componente che orchestra le ACA mantenendo aggiornamenti OS, procedure di recupero post-failure e bilanciamento delle risorse tra componenti. Questo servizio gestisce inoltre la creazione e configurazione del virtual network che si pone a protezione delle ACA istanziate.

5.2.2 Web Application

Per effettuare il deployment dell'applicazione web abbiamo ristretto l'offerta di Azure a due servizi:

- Azure Static Web App (SWA)
- Azure App Service (AAS)

Come in precedenza, il servizio AAS risulta non adatto agli scopi dell'applicazione in quanto essa necessita solo di un frontend che comunica con i due microservizi. Abbiamo quindi deciso di utilizzare il servizio SWA, che offre la possibilità di distribuire una webapp statica con gestione automatica di un CDN globale per garantire una distribuzione del contenuto veloce e affidabile.

Questo servizio offre anche la possibilità di associare un backend sotto forma di ACA o Azure Functions: in futuro sarà possibile aggiungere un nuovo microservizio che gestisca il backend dell'applicazione, implementando funzionalità come la creazione account utente.

5.2.3 Modello Machine Learning

Per rendere il modello accessibile al componente che ne deve mostrare i risultati abbiamo valutato diverse opzioni per creare appositi endpoint per richiedere una inferenza al modello.

La prima opzione che abbiamo considerato è stata quella di creare un container apposito e utilizzare una ACA per rendere gli endpoint accessibili al frontend dell'applicazione. Per quanto questa sarebbe potuta essere una soluzione appropriata, in quanto avrebbe reso più uniforme l'architettura che fa già largamente uso di ACA, Azure offre una soluzione migliore apposita per modelli di machine learning.

Azure Machine Learning Workspace mette a disposizione un ambiente completamente fornito per ogni step della creazione e gestione di un modello di machine learning.

Le funzionalità principali che sono risultate utili sono le seguenti:

- ambienti di sviluppo gestiti e configurabili
- categorizzazione e versioning dei modelli creati
- authoring di modelli tramite jupyter notebooks e strumenti di alto livello a componenti
- riusabilità dei componenti
- gestione automatica della cache di inferenze passate
- valutazione delle performance del modello tramite strumenti di analisi

Questo workspace permette inoltre di effettuare deployment dei modelli creati e di generare appositi endpoint HTTP per richiedere l'inferenza su una istanza del feature vector in modo sicuro, autenticando le richieste tramite chiavi API.

Durante il deployment del modello abbiamo riscontrato problematiche con la gestione delle politiche CORS: il sistema di deployment integrato non supporta la modifica delle politiche CORS per abilitare l'accesso da parte di una applicaizione web. Per risolvere questo problema abbiamo integrato questo endpoint all'interno di un ulteriore endpoint presente nel Middleware `nba_api`.

5.2.4 Servizi di supporto

I servizi appena descritti includono i componenti principali dell'applicazione, abbiamo però utilizzato diversi altri servizi Azure di supporto.

Gestione segreti Una necessità immediatamente evidente durante lo sviluppo è stata quella di avere a disposizione un sistema centralizzato dove archiviare in modo sicuro vari segreti come chiavi API e certificati. Si presta a questo scopo il servizio Azure Key Vault che offre la possibilità di immagazzinare segreti gestendone il versionamento.

Un'altra funzionalità che Key Vault offre è la possibilità di impostare la rotazione automatica di certificati crittografici, sfruttata dal sistema per rinnovare automaticamente i certificati legati al dominio personalizzato usato per raggiungere l'applicazione.

Gestione Immagini OCI Il sistema sfrutta diverse immagini OCI e deve essere in grado di gestirne il versionamento mantenendole private.

A questo scopo abbiamo sfruttato due istanze di Azure Container Registry, un servizio che offre la possibilità di immagazzinare definizioni di immagini con gestione delle versioni tramite tag; è inoltre meglio integrato con gli altri servizi di Azure di altri container registry.

Le credenziali per l'accesso all'istanza ACR principale, ovvero quella che contiene le immagini dei middleware, sono gestite tramite Azure Key Vault. La seconda istanza, dedicata alle immagini degli ambienti di sviluppo del Machine Learning Workspace, è completamente gestita dal Workspace stesso.

Gestione DNS Allo scopo di associare domini personalizzati ai servizi, in particolare alla applicazione web, abbiamo creato una Azure DNS Zone, risorsa che assume il compito di gestire la propagazione dei record DNS del dominio utilizzato.

Abbiamo fatto la scelta di trasferire il controllo dei DNS ad Azure per automatizzare il processo di provisioning dei certificati: questo processo necessita della creazione di record DNS temporanei per autenticare la creazione del certificato e dato che Azure è in controllo dei record può generare e verificare i certificati senza intervento.

Gestione Log Per la gestione dei log generati dai vari componenti dell'infrastruttura abbiamo fatto uso del servizio Azure Log Analitycs Workspace che offre una raccolta dei log centralizzata e mette a disposizione un linguaggio di query e altri strumenti per effettuare analisi sulle informazioni raccolte dai servizi.

Document Database Come supporto aggiuntivo alla cache delle richieste effettuate ai middleware, oltre ad alcuni accorgimenti effettuati nello sviluppo dei due componenti, abbiamo sfruttato un database a documenti per salvare i risultati di alcune richieste molto frequenti e che non necessitano di aggiornamenti frequenti.

5.3 Provisioning Infrastruttura

Data la vasta gamma di risorse Azure da gestire abbiamo deciso di utilizzare un sistema di provisioning automatizzato, a questo scopo abbiamo scelto lo strumento di Infrastructure as Code Terraform.

Tramite HCL, il linguaggio di configurazione di Terraform, è possibile definire l'infrastruttura da creare tramite appositi blocchi risorsa: ognuno di questi blocchi rappresenta una risorsa da generare e specifica tutti i parametri da configurare, come ad esempio immagine, regole di scalabilità e segreti per le ACA.

Tramite il comando `terraform apply` è possibile generare un piano di esecuzione ed eseguirlo: il risultato è la creazione delle risorse sull'abbonamento Azure specificato. Utilizzando il comando `terraform destroy` si da il via alla rimozione delle risorse specificate.

La rimozione di alcune risorse talvolta può incorrere in tempi molto lunghi, soprattutto se è necessario creare e distruggere spesso l'architettura. Per questo motivo abbiamo specificato alcuni componenti Azure come datasource; questo tipo di risorse rappresenta un componente sulla piattaforma cloud del quale è solo necessario prelevare gli attributi: alla distruzione non verranno intaccate queste risorse. Esempi di queste risorse sono gli Azure Container Registry, l'Azure Key Vault e l'Azure Container App Environment che necessitano di essere sempre disponibili nell'infrastruttura.

Abbiamo incontrato varie problematiche con l'utilizzo di Terraform, giustificabile dal fatto che le risorse Azure subiscono modifiche alla tipologia di parametri di configurazione con frequenza abbastanza alta, soprattutto per quanto riguarda i parametri legati a funzionalità in preview.

Un esempio di queste problematiche è relativa alla generazione di certificati per domini personalizzati per le Container App.

Un bug nella gestione dello stato Terraform del ciclo di vita dello stato di provisioning dei certificati impedisce la creazione di un certificato gestito da Azure e il suo provisioning senza intervento manuale dal portale Azure. A questo scopo abbiamo creato appositi script che sfruttano in modo programmatico (e non dichiarativo) Az CLI per sopperire alle mancanze dei moduli azure.

Di seguito si riporta come esempio lo script utilizzato per creare e effettuare il provisioning di un certificato TLS gestito da Azure. Lo script viene usato in tandem con una `null_resource` per includere l'operazione nello stato di Terraform.

```
DOES_CUSTOM_DOMAIN_EXIST=$(
  az containerapp hostname list \
    -n $CONTAINER_APP_NAME \
    -g $RESOURCE_GROUP \
    --query "[?name=='$CUSTOM_DOMAIN'].name" \
    --output tsv
)
if [ -z "${DOES_CUSTOM_DOMAIN_EXIST}" ]; then
  echo "adding custom hostname to container app first since it does not exist yet"
  az containerapp hostname add \
    -n $CONTAINER_APP_NAME \
    -g $RESOURCE_GROUP \
    --hostname $CUSTOM_DOMAIN \
    --output none
fi

MANAGED_CERTIFICATE_ID=$(
  az containerapp env certificate list \
    -g $RESOURCE_GROUP \
    -n $CONTAINER_APP_ENV_NAME \
    --managed-certificates-only \
    --query "[?properties.subjectName=='$CUSTOM_DOMAIN'].id" \
    --output tsv
)
if [ -z "${MANAGED_CERTIFICATE_ID}" ]; then
```

```
  MANAGED_CERTIFICATE_ID=$(
    az containerapp env certificate create \
      -g $RESOURCE_GROUP \
      -n $CONTAINER_APP_ENV_NAME \
      --hostname $CUSTOM_DOMAIN \
      --validation-method CNAME \
      --query "id" \
      --output tsv
  )
  echo "created cert for '$CUSTOM_DOMAIN'. waiting for it to provision now..."

  tries=0
  until [ "$tries" -ge 20 ]; do
    STATE=$(
      az containerapp env certificate list \
        -g $RESOURCE_GROUP \
        -n $CONTAINER_APP_ENV_NAME \
        --managed-certificates-only \
        --query "[?properties.subjectName=='$CUSTOM_DOMAIN'].properties.provisioningState" \
        --output tsv
    )
    echo "checking cert status... $STATE"
    if [[ $STATE == "Succeeded" ]] ; then
      echo "certificate is ready, provisioning state Succeeded"
      break
    else
      tries=$((tries + 1))
    fi

    sleep 15
  done
  if [ "$tries" -ge 20 ]; then
    die "waited for 5 minutes, checked the certificate status 20 times and its not done.
    check azure portal..."
  fi
else
  echo "found existing cert in the env. proceeding to use that"
fi

# check if the cert has already been bound
# if not, bind it then
IS_CERT_ALREADY_BOUND=$(
  az containerapp hostname list \
    -n $CONTAINER_APP_NAME \
    -g $RESOURCE_GROUP \
    --query "[?name=='$CUSTOM_DOMAIN'].bindingType" \
    --output tsv
)
if [ $IS_CERT_ALREADY_BOUND = "SniEnabled" ]; then
  echo "cert is already bound, exiting..."
else
  # try bind the cert to the container app
  echo "cert successfully provisioned. binding the cert id to the hostname"
  az containerapp hostname bind \
    -g $RESOURCE_GROUP \
    -n $CONTAINER_APP_NAME \
    --hostname $CUSTOM_DOMAIN \
    --environment $CONTAINER_APP_ENV_NAME \
    --certificate $MANAGED_CERTIFICATE_ID \
    --validation-method CNAME \
    --output none
  echo "finished binding. the domain is now secured and ready to use"
```

5.4 Continuous Integration

Allo scopo di velocizzare il deployment delle nuove versioni dei componenti rilasciate dagli sviluppatori, abbiamo creato workflow automatizzati per la generazione delle nuove immagini e il rilascio sulle istanze di Container App e Static Web App.

5.4.1 Workflow bet_api

Dato che `bet_api` è stata progettata usando Java abbiamo scelto Gradle come build system, implementando delle task personalizzate mirate a generare un Dockerfile, creare l'immagine e pubblicarla su Azure Container Registry.

Il workflow, rilasciato come Github Action, necessita di accedere ai dati delle credenziali per il registro: queste informazioni sono salvate come Github Secrets. Questa soluzione, utilizzata anche nel workflow per `nba_api`, si è resa necessaria a causa dell'impossibilità di configurare identità tramite Azure.

A causa di queste limitazioni sulla gestione dell'autenticazione non siamo stati in grado di aggiungere al workflow uno step per creare una nuova revisione basata sulla nuova immagine, operazione che quindi viene effettuata manualmente.

Di seguito si riporta un estratto delle task che generano l'immagine, insieme al workflow Github Actions.

```yaml
name: Gradle pushImage task
on:
  push:
    paths:
      - 'bet_api/**'
    branches: [ "master" ]

jobs:
  pushImage:
    env:
      ACR_USERNAME: ${{ secrets.ACR_USERNAME }}
      ACR_PASSWORD: ${{ secrets.ACR_PASSWORD }}
      ACR_SERVER: ${{ secrets.ACR_SERVER }}

    runs-on: ubuntu-latest
    permissions:
      contents: read

    steps:
    - uses: actions/checkout@v4

    - name: Set up JDK 17
      uses: actions/setup-java@v4
      with:
        java-version: '17'
        distribution: 'corretto'

    - name: Setup Gradle
      uses: gradle/actions/setup-gradle@v3.4.2

    - name: Push image with gradle wrapper
      run: ./gradlew pushImage
      working-directory: bet_api
```

Le task Gradle riportate sfruttano il plugin `bmuschko/gradle-docker-plugin` personalizzando i valori necessari a creare le immagini e aggiungere appropriati tag per le versioni.

```
tasks.register<DockerBuildImage>("buildImage") {
    dependsOn("createDockerfile")
    group = "fantanbadocker"
    description = "Build the image from the Dockerfile created"
    val dockerRepository = properties["dockerRepository"]
        ?: throw GradleException("dockerRepository property not set")
    dockerFile.set(file(layout.projectDirectory.toString() + "/build/docker/Dockerfile"))
    inputDir.set(file(layout.projectDirectory))
    images.add("${dockerRepository}/" + project.name + ":latest")
    images.add("${dockerRepository}/" + project.name + ":${project.version}")
}

tasks.register<DockerPushImage>("pushImage") {
    dependsOn("buildImage")
    group = "fantanbadocker"
    description = "Push the docker image to the repository"
    docker {
        registryCredentials {
            url.set(registryUrl)
            username.set(registryUsername)
            password.set(registryPassword)
        }
    }
    val dockerRepository = properties["dockerRepository"]
        ?: throw GradleException("dockerRepository property not set")
    images.add("${dockerRepository}/" + project.name + ":latest")
    images.add("${dockerRepository}/" + project.name + ":${project.version}")
}
```

5.4.2 Workflow `nba_api`

Per il componente **`nba_api`** abbiamo deciso di utilizzare uno script custom per aggiornare il Dockerfile partendo da un template definito e usando file testuali per i valori configurabili. Anche in questo caso abbiamo creato un workflow Github Actions per eseguire queste operazioni automaticamente ad ogni aggiornamento del codice dell'applicazione su una particolare branch della repository.

```
name: Create and push nba_api docker image
on:
  push:
    branches: [ "master" ]
    paths:
      - 'nba_api/**'

jobs:
  buildImage:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login to ACR
        uses: docker/login-action@v3
        with:
          registry: ${{ secrets.ACR_SERVER }}
          username: ${{ secrets.ACR_USERNAME }}
          password: ${{ secrets.ACR_PASSWORD }}

      - name: Read version
```

```
        id: version
        uses: juliangruber/read-file-action@v1
        with:
          path: nba_api/deployment/version

      - name: Read version
        id: repository
        uses: juliangruber/read-file-action@v1
        with:
          path: nba_api/deployment/repository

      - name: Update dockerfile
        run: ./deployment/update-dockerfile.sh
        working-directory: nba_api

      - name: Build and push
        uses: docker/build-push-action@v6
        with:
          push: true
          context: ./nba_api
          file: ./nba_api/deployment/Dockerfile
          tags: |
            "${{ secrets.ACR_SERVER }}/${{ steps.repository.outputs.content }}/nba_api:\
                ${{ steps.version.outputs.content }}"
            ${{ secrets.ACR_SERVER }}/${{ steps.repository.outputs.content }}/nba_api:latest
```

5.4.3 Workflow Static Web App

Per l'applicazione web il workflow è stato configurato direttamente da Azure, che permette di collegare la repository che contiene il codice del progetto. Ad ogni aggiornamento del codice il workflow procede a creare una nuova build del progetto ed effettua automaticamente il deployment sulla piattaforma cloud. In questo caso la gestione dei permessi limitata non ci ha impedito di automatizzare il deployment perché il servizio Static Web App permette di usare una autenticazione tramite token invece che tramite user assigned identity.

5.5 Scalabilità

Per garantire un certo livello di performance dell'applicazione anche sotto carichi elevati abbiamo stabilito regole di scalabilità precise. Per quanto riguarda il deployment del modello di machine learning queste impostazioni non sono configurabili, Azure gestisce autonomamente la creazione di container Docker ridondanti all'aumentare delle richieste.

Le Container App permettono di gestire la scalabilità replicando il container in esecuzione quando necessario, abbiamo configurato:

- numero minimo di repliche: 1
- numero massimo di repliche 15

per entrambe le Container App.

La seconda configurazione da effettuare consiste nei criteri che attivano la creazione di nuove repliche ed è possibile definirne una vasta gamma:

- numero di messaggi su una Azure Queue
- numero di richieste HTTP
- numero di connessioni TCP

- vasta gamma di regole custom su parametri di varie risorse Azure e non

Come regole di scalabilità abbiamo impostato che al raggiungimento di 50 richieste HTTP simultanee a una singola replica è necessario crearne una nuova. La creazione e distruzione delle repliche è completamente gestita da Azure.

Abbiamo deciso di impedire lo scale a zero dell'applicazione principalmente per garantire un certo livello di reattività a richieste "a freddo". Abbiamo preso questo decisione anche perché i container, a riposo, consumano una quantità di risorse trascurabile per i ratei di costo delle Container App.

5.6 Load Alerts

Per monitorare la qualità del servizio abbiamo impostato, tramite il servizio Azure Monitor, una serie di Service Level Objectives che, se non rispettati, inviano un warning al gruppo di responsabili del deployment allo scopo di investigare la fonte del problema.
Di seguito si riportano i Service Level Indicators utilizzati e relativi SLOs:

- Bet Api
  - Memory AVG > 210 MB
  - Network In AVG > 150 MB
  - vCore Usage TOTAL > 1.3c
  - Replica Count > 7
- NBA Api
  - Memory AVG > 350 MB
  - Network In AVG > 220 MB
  - vCore Usage TOTAL > 1.5c
  - Replica Count > 7

Questi valori sono stati raccolti in modo empirico durante uno dei load test effettuati per valutare la performance del sistema. Gli indicatori sono intesi come livello di warning non catastrofico allo scopo di individuare inefficienze in anticipo e intervenire.

5.7 Load Testing

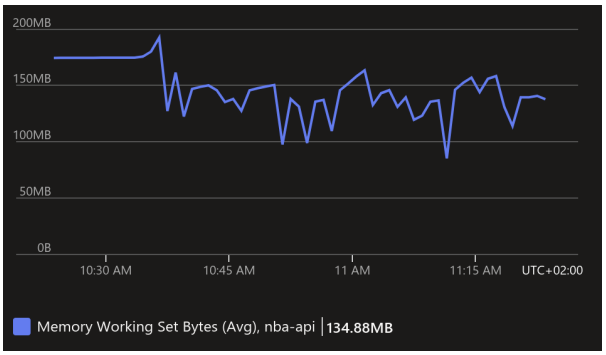
Abbiamo effettuato un load test su entrambe le API con diversi parametri e per generare test replicabili e configurabili abbiamo utilizzato il servizio Azure Load Test, che permette di definire una serie di endpoint ai quali fare richieste passando come parametri dati forniti tramite appositi file csv; il servizio fornisce statistiche sull'utilizzo di risorse dei container durante il test.

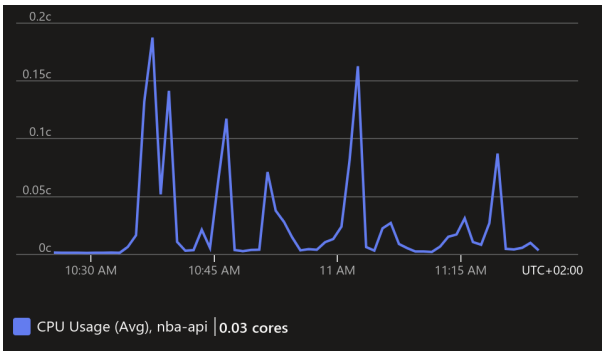
I test sono stati configurati compilando gli endpoint da testare e fornendo liste di parametri sui quali iterare per effettuare le richieste, il che permette di verificare che i sistemi di cache siano efficaci, in quanto viene riutilizzata molte volte la stessa richiesta.

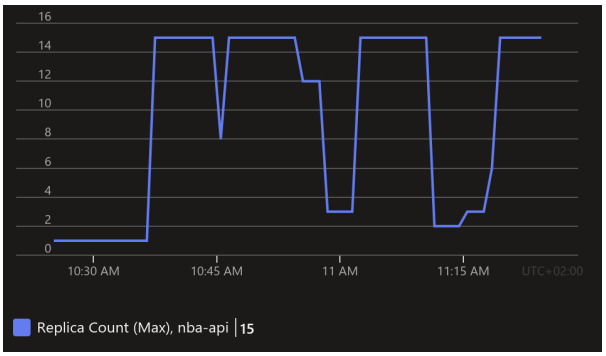
Il test per `nba_api` è stato generato con 2 engine che generano ognuno fino a 50 utenti concorrenti per una durata di 1 ora, quello per `bet_api` invece con 5 engine e fino a 100 utenti concorrenti per una durata di 2 ore e 30 minuti.

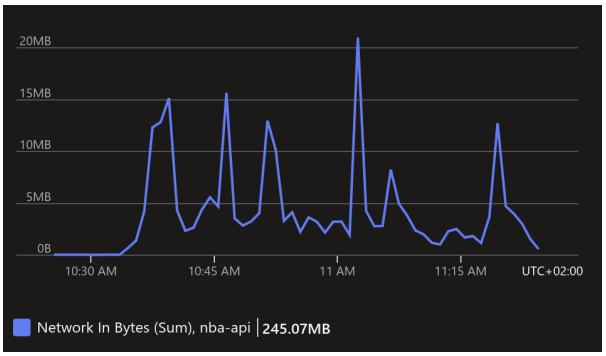
La discrepanza tra il numero di utenti tra i due test è dovuta al fatto che, a causa delle limitazioni di `swar/nba_api`, non si ottengono risultati accurati sovraccaricando eccessivamente l'API: si denota che le limitazioni non sono dovute alla implementazione ma al rate limiting dell'API.

Di seguito si riportano alcuni grafici che mostrano l'andamento di utilizzo delle risorse durante il test di `nba_api`.



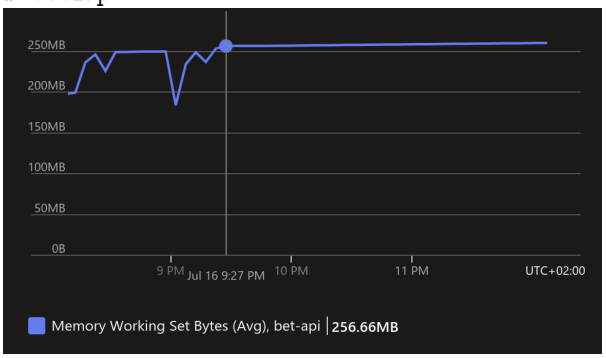


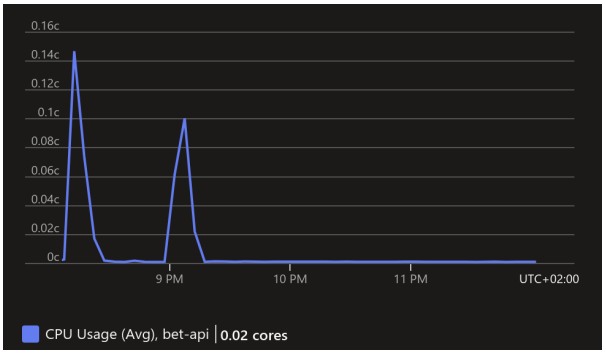


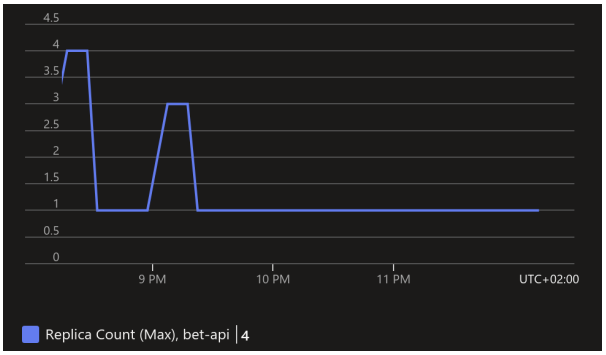


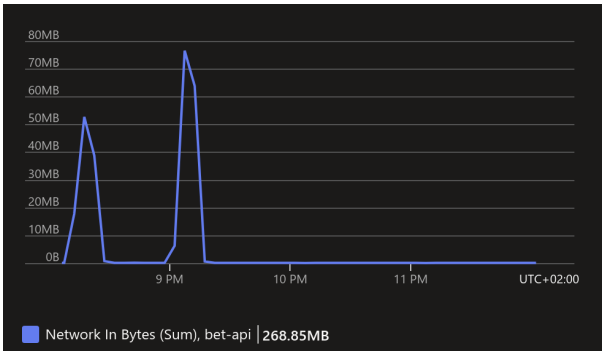
Come si può notare dai grafici, `nba_api` subisce una considerevole riduzione (circa 30%) della media di memoria di sistema utilizzata da una replica, mentre l'utilizzo di CPU medio incrementa fino a 0.2 vCPU in media per ogni replica. Questi grafici evidenziano il problema definito in precedenza con il rate limiting di `swar/nba_api`: quando l'API risponde con timeout la replica in questione viene terminata dopo qualche minuto perché il sistema non la ritiene più necessaria.

Di seguito si riportano alcuni grafici che mostrano l'andamento di utilizzo delle risorse durante il test di `bet_api`.









I risultati di `bet_api` differiscono significativamente da quelli di `nba_api`: si rivela molto più efficiente in quanto non necessita di un numero di repliche alto (massimo raggiunto pari a 4) per gestire un traffico maggiore di quello imposto a `nba_api`.

5.8 Stima costi

Services (Euro/Month)	
Azure cosmos DB	$\sim$ 6
Load Test	$\sim$ 10 (50h test)
Azure ML workspace	$\sim$ 90
Container apps x2	$\sim$ 600 (3x load test)
Registry x2	$\sim$ 10
DNS Zone	$\sim$ 0.80 (1 mld richieste)
Azure key vault	$\sim$ 5
Log analytics x2	$\sim$ 28
Various Alerts	0.00
Static web app	$\sim$ 80 (3x load test)
Total	**830**

La sezione di cost analysis è stata inaccessibile di recente quindi le valutazioni sui costi sono stimate basandosi sui prezzi riportati da Azure al momento della scrittura e sono stati aggiustati tenendo conto di un carico simile a circa 3 volte i load test effettuati.

Il servizio Static Web App risulta essere la soluzione meno costosa tra quelle presenti per hostare applicazioni web, dovuto al fatto che si tratta di una applicazione senza backend. Il costo si alzerebbe considerevolmente aggiungendo un Container App per il backend ($\sim$ €100).

Il servizio più costoso sono le Container App, in quanto sono le risorse che usano più vCPU in assoluto. In futuro si potrà sostituire il cluster ACA con un Azure Kubernetes Service che consentirebbe di abbassare il costo diretto del cluster, aggiungendo però overhead di gestione dello stesso.
Infine, le Azure Container Apps si sono rese utili per distribuire il più velocemente possibile una versione iniziale dell'applicazione.