Relazione Assignment #03

E-Bike Application - Event Driven Architecture

Software Architecture and Platforms

Ravaioli Michele - michele.ravaioli3@studio.unibo.it

2 gennaio 2025

Indice

1. Requisiti

Il progetto prevede due principali richieste di sviluppo:

1. **Servizio di noleggio di bici elettriche:**
   Basato sull'architettura **event-driven**, il servizio è progettato per gestire gli eventi relativi al ciclo di vita del noleggio, applicando dove necessario il pattern **Event Sourcing**. Si deve inoltre definire il deployment di questa applicazione per un sistema distribuito, basandosi su Kubernetes.
2. **Modello di bici a guida autonoma - "A-Bike":**
   È richiesto lo sviluppo di un modello avanzato di bicicletta elettrica a guida autonoma, denominato *A-Bike*, con due funzionalità principali:
   - **Raggiungere l'utente:** l'A-Bike è in grado di dirigersi autonomamente verso l'utente che richiede il servizio di noleggio.
   - **Ricarica automatica:** quando il livello di batteria è basso, l'A-Bike è in grado di dirigersi autonomamente alla stazione di ricarica più vicina.

2. Architettura Event-Driven

Questo progetto si basa interamente sul sistema realizzato nell'assignment #2. Non sono state rimosse funzionalità esistenti: tutte le REST API originariamente presenti sono ancora esposte. Per trasformare l'architettura in un sistema event-driven, sono stati aggiunti nuovi componenti ai singoli servizi senza alterare la struttura della clean architecture su cui sono stati progettati.

I servizi, invece di comunicare tra loro tramite chiamate alle REST API, utilizzano ora una comunicazione basata su eventi. Ogni sistema aggiorna la propria conoscenza ascoltando gli eventi pubblicati nei topic di interesse, eliminando così la necessità di effettuare query dirette ai servizi dipendenti. Gli eventi emessi sono rappresentati nella Figura 1.

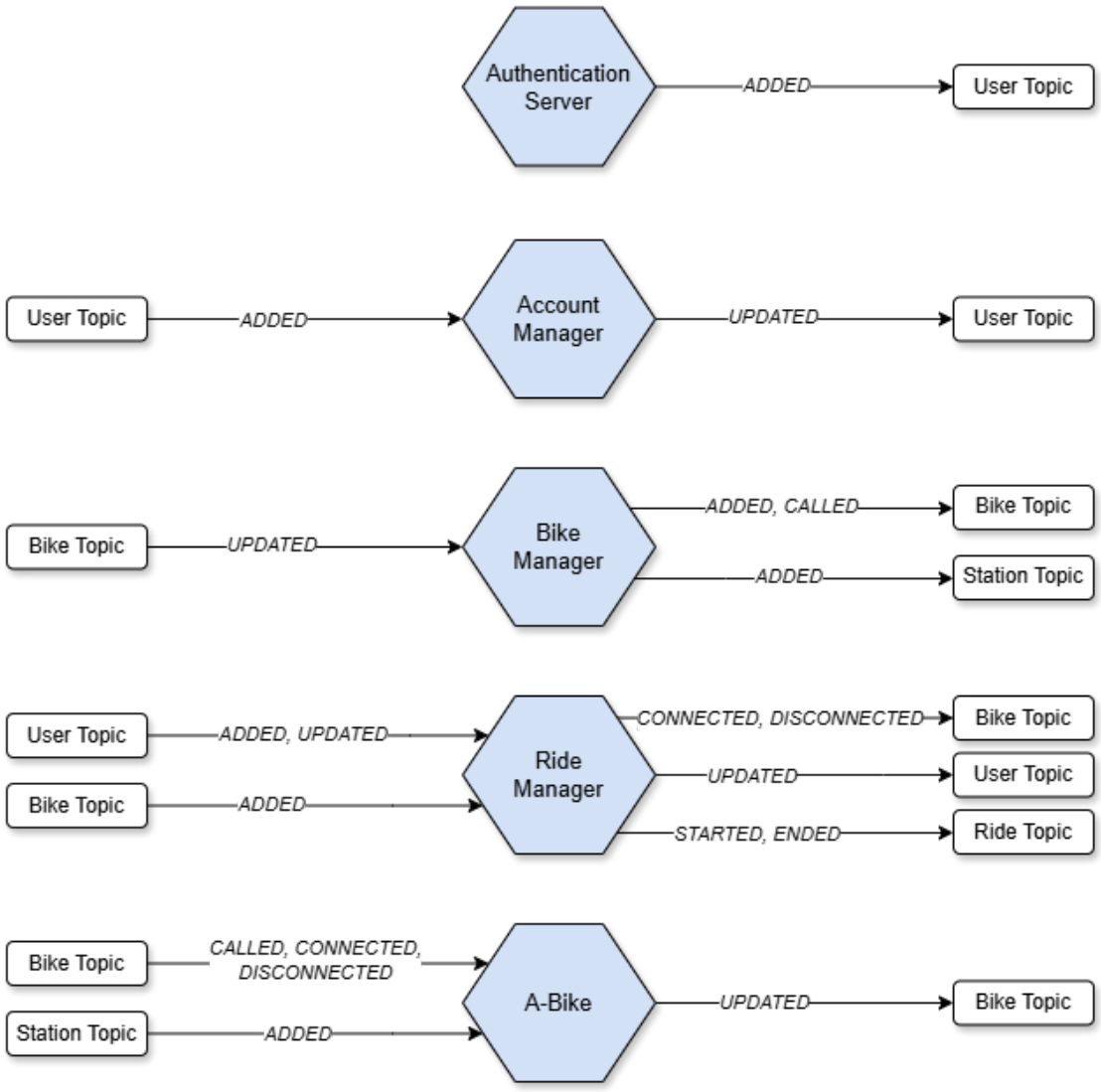

Fig. 1: Schema degli eventi

Per preservare la clean architecture, la gestione degli eventi è stata implementata attraverso una port, denominata *EventController*, che consente sia l'invio sia la gestione degli eventi. Questa soluzione è progettata in modo uniforme per tutti i servizi.

Come broker per gli eventi è stato utilizzato **Apache Kafka**, aggiunto come container visibile esclusivamente ai servizi. Per semplificare l'utilizzo di Kafka, sono stati implementati adapter specifici chiamati *KafkaEventController*, che, supportati dal framework **Micronaut**, facilitano l'invio e la lettura degli eventi. Questa struttura è illustrata nella Figura 2.

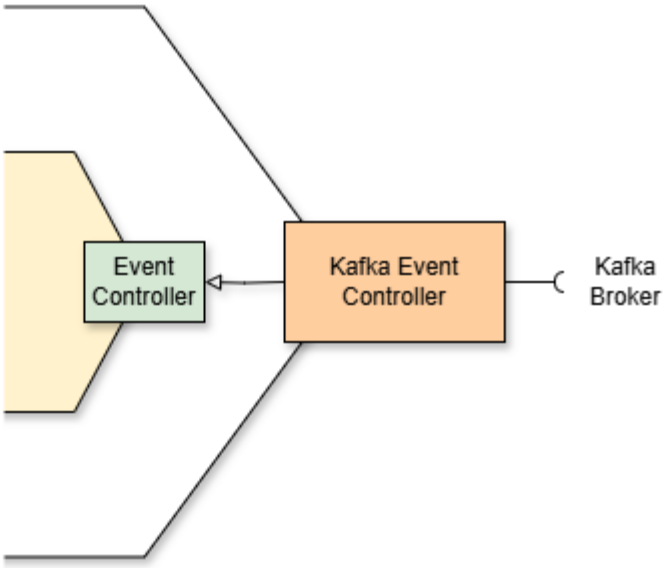

Fig. 2: Schema dell'architettura dell' Event Controller

Il pattern Event Sourcing è stato applicato al servizio RideManager, sostituendo il precedente database con un'implementazione basata su eventi, denominata *FileRideDatabase*. In questa configurazione, le ride non sono più memorizzate come entità statiche, ma come una sequenza di eventi, ovvero connessione e disconnessione.

Per ricostruire lo stato attuale del sistema, è necessario leggere il file di storage contenente tutti gli eventi e applicarli in ordine cronologico, dal primo all'ultimo evento registrato. Questo approccio consente di ricostruire interamente lo stato del sistema in modo coerente ed affidabile, oltre a fornire uno storico di tutte le ride effettuate dagli utenti.

Per garantire la disponibilità dell'applicazione su un'infrastruttura distribuita, il sistema è stato configurato per essere eseguito su **Kubernetes**. Questo ha richiesto la creazione delle immagini Docker per ciascun servizio, seguita dalla definizione delle configurazioni di deployment specifiche per Kubernetes, che sono archiviate nella directory *KubernetesConfig/*.

Ogni servizio è stato containerizzato utilizzando **Docker**, generando immagini specifiche che includono tutte le dipendenze necessarie per il funzionamento del servizio.

Per orchestrare i servizi, sono stati definiti i seguenti oggetti Kubernetes:

1. **Deployment:**
   Ogni servizio ha un oggetto di tipo Deployment che specifica:
   - L'immagine Docker da utilizzare.
   - Il numero di repliche per garantire la scalabilità e la tolleranza ai guasti.
2. **Service:**
   Per esporre i servizi di ciascun Deployment su un unico indirizzo gestito da Kubernetes, sono stati creati oggetti di tipo Service. Questi gestiscono:
   - Il bilanciamento del carico tra le repliche del servizio.
   - L'accesso interno o esterno ai servizi, secondo la configurazione delle porte.

Le configurazioni, comprensive dei file YAML definiti, permettono di avviare l'intera applicazione su un cluster Kubernetes con un semplice comando di applicazione delle configurazioni.

3. A-Bike

L'integrazione della bici a guida autonoma, denominata **A-Bike**, è stata agevolata dalla struttura preesistente del sistema sviluppato per l'assignment #2. In quel contesto, infatti, tutte le e-bike erano già trattate come entità attive del sistema, e pertanto il dominio iniziale del problema è rimasto pressoché invariato.

L'unica modifica significativa è stata l'introduzione delle stazioni di ricarica, non presenti nella versione precedente. Queste sono state aggiunte come nuove entità di dominio nel servizio *BikeManager*, insieme a REST API per visualizzare le stazioni esistenti e per aggiungerne di nuove.

La A-Bike è stata progettata come un agente autonomo che opera in un ambiente dinamico, seguendo un ciclo continuo di tre fasi:

1. **Sense:** percezione dell'ambiente circostante, in questo caso la topologia delle strade.
2. **Decide:** elaborazione delle informazioni raccolte per pianificare le azioni, tenendo conto dello stato interno della bici.
3. **Act:** esecuzione delle azioni pianificate, come il movimento verso l'utente o la stazione di ricarica.

Il comportamento della A-Bike è descritto da una macchina a stati finiti (vedi Figura 3), che include i seguenti stati:

- **IDLE:** la bici è in attesa di ordini.
- **CALLED:** la bici si dirige verso l'utente che l'ha richiesta.
- **CONNECTED:** la bici è utilizzata da un utente.
- **NEED_RECHARGE:** la bici si dirige verso la stazione di ricarica più vicina.

La transizione tra questi stati è regolata dalle condizioni operative, come la richiesta di un utente o il livello della batteria.

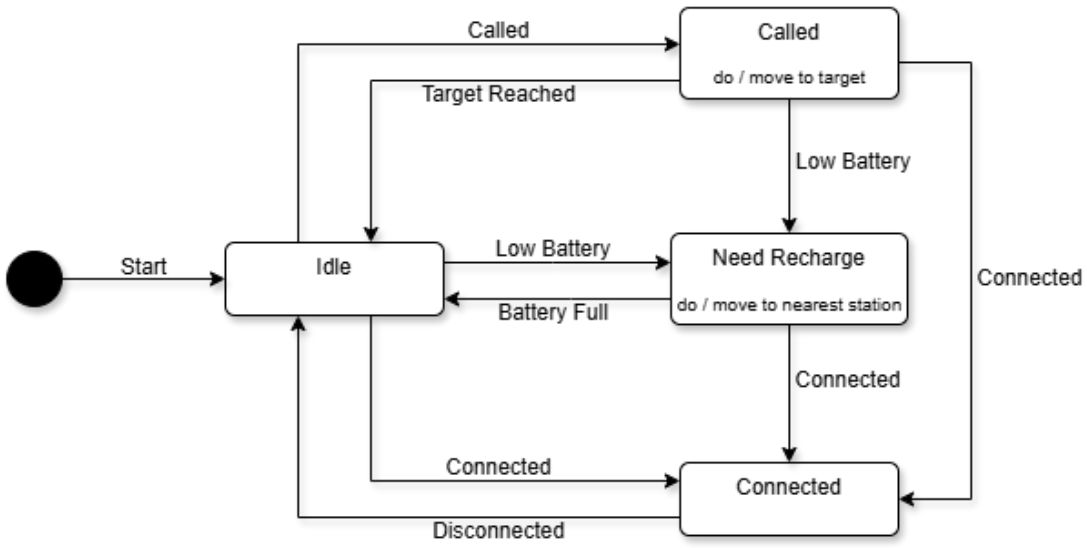

Fig.3: Diagramma degli stati della A-Bike

La soluzione è stata sviluppata seguendo un'architettura a strati:

- Modulo Core**:** contiene le classi generiche per i sistemi ad agenti, come *Environment* e *AbstractAgent*.
- Modulo A-Bike**:** implementa le specifiche del problema, includendo le classi *EmptyEnvironment* e *ABikeAgent*.

La A-Bike interagisce con gli altri servizi esclusivamente tramite eventi pubblicati nel topic delle bici. In particolare:

- Riceve notifiche relative all'aggiunta di nuove stazioni di ricarica o richieste di noleggio da parte degli utenti.
- Pubblica periodicamente eventi di tipo *UPDATED*, che riportano lo stato della bici.

Per rendere disponibile questa funzionalità agli utenti, è stato aggiornato il client esistente dell'assignment #2, aggiungendo l'opzione per richiedere una A-Bike (vedi Figura 4).

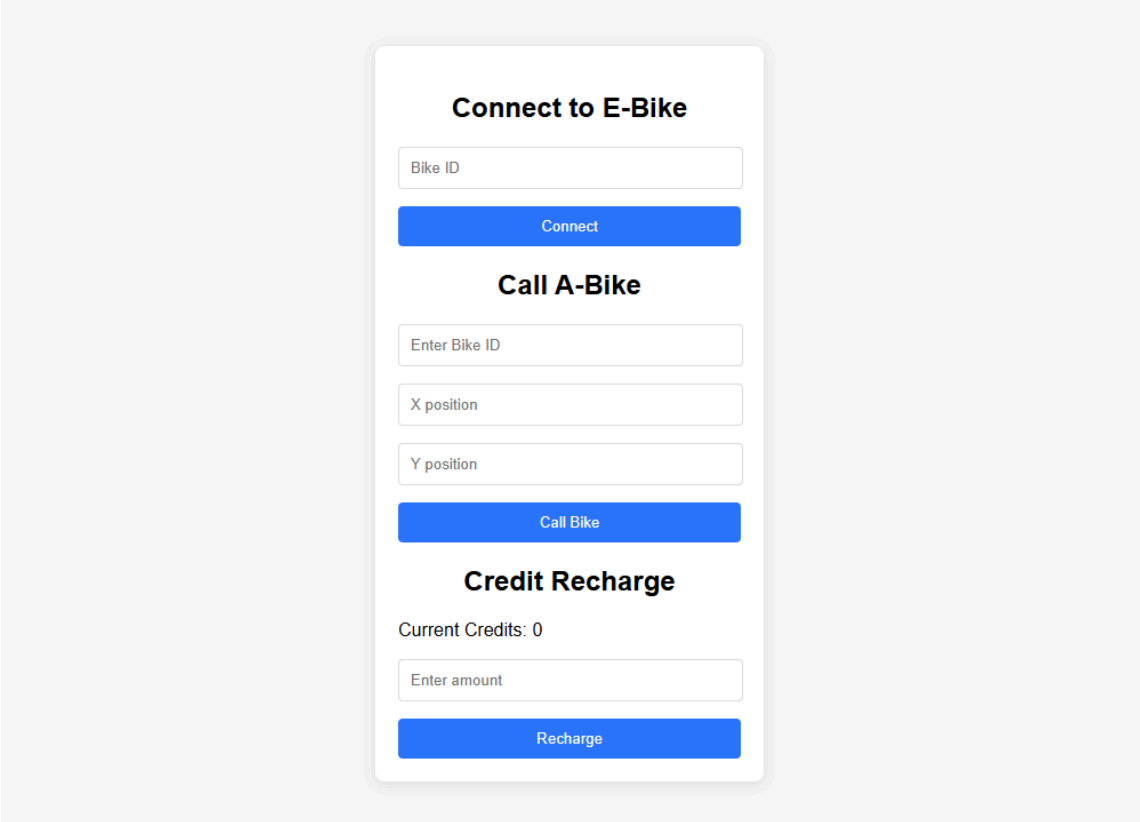

Fig.4: Schermata della gestione dell'account dell'utente

E’ stato inoltre aggiunto un piccolo client per la sola lettura **AppViewer** che permette di visualizzare la posizione delle bici in tempo reale, cosi’ da poter testare il funzionamento delle A-Bike.