**FestivalSync**

**Specifiche tecnico/funzionali**

|  |  |  |
| --- | --- | --- |
| Versione Documento |  | Data Documento |
| 1.0 |  | 05/12/2024 |
|  |  |  |

SOMMARIO

[1 STORIA DEL DOCUMENTO 3](#_Toc184250188)

[2 INTRODUZIONE 4](#_Toc184250189)

[2.1 Definizioni, abbreviazioni, acronimi 4](#_Toc184250190)

[2.1.1 Definizioni 4](#_Toc184250191)

[2.1.2 Acronimi 4](#_Toc184250192)

[2.2 Documenti di riferimento 4](#_Toc184250193)

[3 INFORMAZIONI GENERALI 5](#_Toc184250194)

[3.1 Versionamento Git 5](#_Toc184250195)

[4 Impostazione progetto 5](#_Toc184250196)

[4.1 Tecnologie utilizzate 5](#_Toc184250197)

[4.2 Microservizi 5](#_Toc184250198)

[4.2.1 Artist-Service 5](#_Toc184250199)

[4.2.2 Event-service 8](#_Toc184250200)

[4.2.3 Ticket-service 10](#_Toc184250201)

[5 Configurazioni tecniche 12](#_Toc184250202)

[5.1 Docker 12](#_Toc184250203)

[5.2 Kubernetes 12](#_Toc184250204)

[5.3 Prometheus 12](#_Toc184250205)

[5.4 Database 13](#_Toc184250206)

[5.5 Kafka 13](#_Toc184250207)

[6 Scelte Progettuali 13](#_Toc184250208)

[6.1 Scelte Tecnologiche 13](#_Toc184250209)

[6.2 Monitoraggio e Predittore ARIMA 14](#_Toc184250210)

[7 PUNTI DI ATTENZIONE 14](#_Toc184250211)

[8 ALLEGATI 15](#_Toc184250212)

# STORIA DEL DOCUMENTO

Le informazioni contenute in questo documento aggiornano e sostituiscono quelle contenute nelle versioni precedenti. Ogni modifica del documento sarà controllata nella apposita procedura.

|  |  |  |  |
| --- | --- | --- | --- |
| **Versione** | **Data** | **Descrizione della Revisione** | **Modificato da** |
| 1.0 | 05/12/2024 | Prima versione | Carlo Blatti |

# INTRODUZIONE

## Definizioni, abbreviazioni, acronimi

* **FestivalSync**: Sistema per la gestione di eventi, artisti e ticket di festival musicali.
* **API Gateway**: Punto di accesso centralizzato per le richieste ai microservizi.
* **Prometheus**: Strumento di monitoraggio utilizzato per raccogliere metriche dai microservizi.
* **ARIMA**: Modello per l’analisi e la previsione di dati temporali.

### Definizioni

### Acronimi

## Documenti di riferimento

# INFORMAZIONI GENERALI

Il progetto è realizzato utilizzando il framework Spring e il modulo Spring boot, con alcune librerie di supporto.

## Versionamento Git

Il progetto è disponibile su Git all’indirizzo https://github.com/CARLOBLATTI /festivalsync

# Impostazione progetto

Il progetto è stato sviluppato in linguaggio Java utilizzando il framework **Spring Boot** per i microservizi Il monitoraggio è stato implementato utilizzando Prometheus e Grafana mentre per il predittore è stato utilizzato il linguaggio Python.

## Tecnologie utilizzate

Per lo sviluppo del progetto si è fatto uso delle seguenti tecnologie:

* **Java**: v.17
* **Spring Boot**: v.3.1.0
* **Docker**: Per il containerization dei microservizi.
* **Kubernetes**: Per l'orchestrazione e la gestione dei container.
* **Prometheus**: Per il monitoraggio delle performance.
* **MariaDB/MySQL**: Per la gestione delle basi dati.
* **Python:** v. 3.12.5
* **Numpy**: v. 1.26.3
* **Matplotlib**: v. 3.9.3
* **Pndarima**: v. 2.0.4

## Microservizi

Sono stati implementati 3 microservizi che operano in maniera asincrona e che cooperano per la gestione di festival.

### Artist-Service

Il microservizio che si occupa della gestione degli Artisti, un artista può registrare la sua partecipazione ad un evento o può cancellarla, inoltre può verificare tutti gli eventi a cui si è registrato.

#### API

Il servizio mette a disposizione diverse API basate su http/REST che, una volta avviato l’applicativo in locale è possibile visionare dallo swagger all’indirizzo <http://localhost:8081/swagger-ui/index.html#/>.

##### POST [**/artist-service/add-artist**](http://localhost:8081/swagger-ui/index.html#/Artist%20Controller/addArtist)

Aggiungi un nuovo artista e invia un messaggio Kafka

* **Parameters**

No parameters

* **Request body**: application/json
* AddArtistRequest**:**

|  |  |
| --- | --- |
| **description:** | Richiesta per aggiungere un nuovo artista |
| **name\*** | **string *example: John Doe***  **Nome dell'artista** |
| **genre** | **string *example: Rock***  **Genere musicale dell'artista** |
| **country** | **string *example: Italy***  **Paese di origine dell'artista** |
| **location** | **string *example: Rome***  **Città di provenienza dell'artista** |
| **desiredEventCity\*** | **string *example: Rome***  **Città desiderata per l'evento** |
| **desiredEventDate\*** | **string($date) *example: 2024-05-15***  **Data desiderata per l'evento** |

AddArtistResponse**:**

|  |  |
| --- | --- |
| **description:** | Risposta per l'inserimento di un artista |
| **id** | **integer($int64) *example: 1***  **ID dell'artista** |
| **message** | **string *example: Artista aggiunto con successo***  **Messaggio di conferma** |

##### GET [**/artist-service/get-artist-events/{name}**](http://localhost:8081/swagger-ui/index.html#/Artist%20Controller/getArtistEvents)

Ottieni tutti gli eventi associati a un artista

name: nome dell’artista

Esempio risposta:

**GetArtistEventsResponse**

|  |  |
| --- | --- |
| **description:** | **Risposta per un artista e gli eventi a cui partecipa** |
| **events** | **[**  **Lista degli eventi a cui partecipa l'artista**  **EventModel{**   |  |  | | --- | --- | | **description:** | **Modello che rappresenta un evento** | | **id** | **integer($int64) *example: 1***  **ID univoco dell'evento** | | **name\*** | **string *example: Music Festival***  **Nome dell'evento** | | **date\*** | **string($date) *example: 2024-05-15***  **Data dell'evento** | | **location** | **string *example: Stadio Olimpico***  **Luogo dell'evento** | | **country** | **string *example: Italy***  **Paese in cui si svolge l'evento** | | **state** | **string *example: scheduled***  **Stato dell'evento** | | **artistsNumber** | **integer($int32) *example: 2***  **Numero di artisti che partecipano all'evento** | | **creationDate\*** | **string($date) *example: 2024-01-01***  **Data di creazione dell'evento** | |

##### GET [**/artist-service/all-artist**](http://localhost:8081/swagger-ui/index.html#/Artist%20Controller/getArtistById)

Recupera un tutti gli artisti registrati

Risposta:

**ArtistModel**

|  |  |
| --- | --- |
| **description:** | Modello che rappresenta un artista |
| **id** | **integer($int64) *example: 1***  **ID univoco dell'artista** |
| **name\*** | **string *example: John Doe***  **Nome dell'artista** |
| **events** | **EventModel{**   |  |  | | --- | --- | | **description:** | Modello che rappresenta un evento | | **id** | **integer($int64) *example: 1***  **ID univoco dell'evento** | | **name\*** | **string *example: Music Festival***  **Nome dell'evento** | | **date\*** | **string($date) *example: 2024-05-15***  **Data dell'evento** | | **location** | **string *example: Stadio Olimpico***  **Luogo dell'evento** | | **country** | **string *example: Italy***  **Paese in cui si svolge l'evento** | | **state** | **string *example: scheduled***  **Stato dell'evento** | | **artistsNumber** | **integer($int32) *example: 2***  **Numero di artisti che partecipano all'evento** | | **creationDate\*** | **string($date) *example: 2024-01-01***  **Data di creazione dell'evento** |   **}** |
| **country** | **string *example: Italy***  **Paese di origine dell'artista** |
| **location** | **string *example: Rome***  **Città di provenienza dell'artista** |
| **state** | **string *example: active***  **Stato attuale dell'artista** |
| **creationDate\*** | **string($date) *example: 2024-01-01***  **Data di registrazione dell'artista** |
| **genre** | **string *example: Rock***  **Genere musicale dell'artista** |

##### DELETE [**/artist-service/delete-artist/{id}**](http://localhost:8081/swagger-ui/index.html#/Artist%20Controller/deleteArtist)

Elimina la partecipazione di un artista

Esempio Risposta: “Artista eliminato correttamente”

### Event-service

Il microservizio permette di ottenere le informazioni relative ad un singolo even to, le informazioni di tutti gli eventi disponibili o di eliminare un evento.

#### API

Mette a disposizione API HTTP/REST in GET e DELETE disponibili tramite swagger alll’indirizzo <http://localhost:8082/swagger-ui/index.html#/>

##### GET [**/event-service/{id}**](http://localhost:8082/swagger-ui/index.html#/Event%20Controller/getEventById)

Recupera un evento tramite ID.

Risposta: EventModel

|  |  |
| --- | --- |
| **description:** | Modello che rappresenta un evento |
| **id** | **integer($int64) *example: 1***  **ID univoco dell'evento** |
| **name\*** | **string *example: Music Festival***  **Nome dell'evento** |
| **date\*** | **string($date) *example: 2024-05-15***  **Data dell'evento** |
| **location** | **string *example: Stadio Olimpico***  **Luogo dell'evento** |
| **country** | **string *example: Italy***  **Paese in cui si svolge l'evento** |
| **artists** | **[**  **Lista degli artisti che partecipano all'evento**  **ArtistModel{**   |  |  | | --- | --- | | **description:** | Modello che rappresenta un artista | | **id** | **integer($int64) *example: 1***  **ID univoco dell'artista** | | **name\*** | **string *example: John Doe***  **Nome dell'artista** | | **genre** | **string *example: Rock***  **Genere musicale dell'artista** | | **country** | **string *example: Italy***  **Paese di origine dell'artista** | | **location** | **string *example: Rome***  **Città di provenienza dell'artista** | | **state** | **string *example: active***  **Stato attuale dell'artista** | | **artistsNumber** | **integer($int32) *example: 2***  **Numero di artisti che partecipano all'evento** | | **creationDate\*** | **string($date) *example: 2024-01-01***  **Data di registrazione dell'artista** |   **}]** |
| **state** | **string *example: scheduled***  **Stato dell'evento** |
| **artistsNumber** | **integer($int32) *example: 2***  **Numero di artisti che partecipano all'evento** |
| **creationDate\*** | **string($date) *example: 2024-01-01***  **Data di creazione dell'evento** |
| **insertTimestamp** | **string($date-time)**  **Timestamp di inserimento dell'evento** |
| **updateTimestamp** | **string($date-time)**  **Timestamp di aggiornamento dell'evento** |

**}**

##### GET [**/event-service/all**](http://localhost:8082/swagger-ui/index.html#/Event%20Controller/getArtistById)

Recupera tutti gli eventi creati basati sul modello descritto nell’API precedente

Esempio:

**[**

**{**

**"id": 1,**

**"name": "Music Festival",**

**"date": "2024-05-15",**

**"location": "Stadio Olimpico",**

**"country": "Italy",**

**"artists": [**

**{**

**"id": 1,**

**"name": "John Doe",**

**"genre": "Rock",**

**"country": "Italy",**

**"location": "Rome",**

**"state": "active",**

**"artistsNumber": 2,**

**"creationDate": "2024-01-01"**

**}**

**],**

**"state": "scheduled",**

**"artistsNumber": 2,**

**"creationDate": "2024-01-01",**

**"insertTimestamp": "2024-12-04T22:31:28.843Z",**

**"updateTimestamp": "2024-12-04T22:31:28.843Z"**

**}**

**]**

##### DELETE [**/event-service/delete-event/{id}**](http://localhost:8082/swagger-ui/index.html#/Event%20Controller/deleteArtist)

Elimina un evento.  
Esempio di risposta: “Evento cancellato correttamente”

### Ticket-service

Il microservizio permette di ottenere le informazioni relative al ticket messo in vendita a seguito della creazione di un evento, inoltre permette di acquistare dei ticket e richiederne il rimborso.

#### API

Mette a disposizione API HTTP/REST in GET POST E PUT, disponibili tramite swagger alll’indirizzo <http://localhost:8083/swagger-ui/index.html#/>

##### GET [**/ticket-service/tickets/{eventId}**](http://localhost:8083/swagger-ui/index.html#/ticket-controller/getTicketsByEventId)

Ottieni i dettagli di un ticket in base all'eventId.

Risposta:

**Tickets{**

|  |  |
| --- | --- |
| **id** | **integer($int64)** |
| **eventId** | **integer($int64)** |
| **price** | **number** |
| **availability** | **integer($int32)** |
| **state** | **string** |
| **creationDate** | **string($date)** |
| **insertTimestamp** | **string($date-time)** |
| **updateTimestamp** | **string($date-time)** |

**}**

##### POST [**/ticket-service/purchase**](http://localhost:8083/swagger-ui/index.html#/ticket-controller/purchaseTicket)

Compra uno o più ticket.

Request:  
PurchaseTicketRequest{

description:

Richiesta per acquistare uno o più ticket

ticketId\* integer($int64)

example: 1

ID del ticket da acquistare

buyerName\* string

example: John Doe

Nome dell'acquirente

buyerEmail\* string

example: johndoe@example.com

Email dell'acquirente

buyerPhone\* string

example: +1234567890

Telefono dell'acquirente

quantity\* integer($int32)

minimum: 1

example: 2

Quantità di ticket da acquistare

}

Esempio di Risposta: “Ticket acquistato con successo!”

##### PUT [**/ticket-service/refund**](http://localhost:8083/swagger-ui/index.html#/ticket-controller/refundTicket)

Richiedi il rimborso di un ticket.

Request:  
RefundTicketRequest**{**

|  |  |
| --- | --- |
| **description:** | Richiesta per rimborsare un ticket |
| **soldTicketId\*** | **integer($int64) *example: 1***  **ID del ticket venduto da rimborsare** |

**}**

Esempio risposta: “Rimborso effettuato con successo!”

# Configurazioni tecniche

## Docker

Ogni microservizio è containerizzato utilizzando Docker. Esempio di **Dockerfile** per artist-service:

# Usa l'immagine base di OpenJDK  
FROM openjdk:17-jdk-slim  
  
# Imposta la directory di lavoro nel container  
WORKDIR /app  
  
# Copia il file JAR generato da Maven nella directory di lavoro del container  
COPY target/artist-service.jar app.jar  
  
# Definisce il comando da eseguire quando il container viene avviato  
ENTRYPOINT ["java", "-jar", "/app/app.jar"]  
  
# Espone la porta 8081  
EXPOSE 8081

Il file .docker-compose.yml viene utilizzato per orchestrare i container durante lo sviluppo.

## Kubernetes

I microservizi sono distribuiti anche utilizzando Kubernetes.

## Prometheus

Prometheus è utilizzato per raccogliere metriche da ogni microservizio. Ogni servizio espone le proprie metriche tramite l'endpoint /actuator/prometheus. Le metriche sono raccolte in tempo reale e sono visualizzabili tramite **Grafana**.

* **Micrometer** è la libreria utilizzata per integrare Spring Boot con Prometheus

## Database

Il database è di tipo MySql utilizzando **MariaDB** per la gestione dei dati relativi agli artisti, eventi e biglietti.

Il database è containerizzato tramite Docker e distribuito su Kubernetes utilizzando un **StatefulSet** per garantire la persistenza.

## Kafka

Per coordinare la gestione di eventi e artisti nei vari microservizi, sono stati definiti i seguenti **4 topic Kafka**:

1. **artist-added**
2. **event-created**
3. **event-deleted**
4. **artist-deleted**

Immaginiamo il seguente scenario: viene creato un nuovo artista e viene aggiunto ad un evento.

1. **Aggiunta dell'Artista**:
   * L'amministratore o un artista aggiunge una nuova registrazione (artist-service).
   * artist-service pubblica un messaggio sul topic artist-added.
   * event-service consuma questo messaggio per creare un nuovo evento o per registrare la partecipazione ad un evento esistente.
2. **Creazione dell'Evento**:
   * A seguito della creazione di un evento
   * event-service pubblica un messaggio sul topic event-created.
   * ticket-service consuma questo evento per generare i biglietti necessari.
3. **Cancellazione di un Artista**:
   * In caso di cancellazione di un artista (artist-service), viene inviato un messaggio sul topic artist-deleted.
   * event-service consuma il messaggio e aggiorna o cancella gli eventi collegati all’artista cancellato.
4. **Cancellazione di un Evento**:
   * In caso di cancellazione di un evento (event-service), viene inviato un messaggio sul topic event-deleted.
   * artist-service consuma il messaggio e elimina tutte le partecipazioni degli artisti all’evento
   * ticket-service consuma l’evento per rimborsare tutti i ticket acquistati ed effettuare una cancellazione logica del prodotto ticket, segnandolo come DELETED.

# Scelte Progettuali

## Scelte Tecnologiche

La scelta di utilizzare **Spring Boot** è stata guidata dalla necessità di avere una piattaforma solida e scalabile per lo sviluppo di applicazioni enterprise. **Kafka** è stato scelto per la sua capacità di gestire messaggistica in tempo reale, fondamentale per la sincronizzazione delle informazioni tra microservizi.

## Monitoraggio e Predittore ARIMA

Prometheus viene utilizzato per il monitoraggio delle performance dei microservizi, permettendo la raccolta di metriche in tempo reale. Inoltre, è stato implementato un predittore **ARIMA** per l’analisi dei dati relativi alla latenza delle chiamate HTTP/REST e dell’utilizzo della CPU.

# PUNTI DI ATTENZIONE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | Descrizione | Owner | Decisioni | Stato |
|  |  |  |  |  |
|  |  |  |  |  |

# ALLEGATI

|  |  |  |
| --- | --- | --- |
| **ID** | Allegato | Descrizione |
|  | [FestivalSync\_Forecast](file:///C:\Users\Carlo\workspace\spring\festivalsync\documentation\FestivalSync_Forecast.docx) | Dettaglio sul monitoraggio e sul predittore |
|  |  |  |
|  |  |  |