

# Progetto: Gestione degli Eventi

## 1. Descrizione del Progetto:

L'applicazione "Gestione degli Eventi" permette agli utenti di creare, visualizzare, modificare e cancellare eventi.

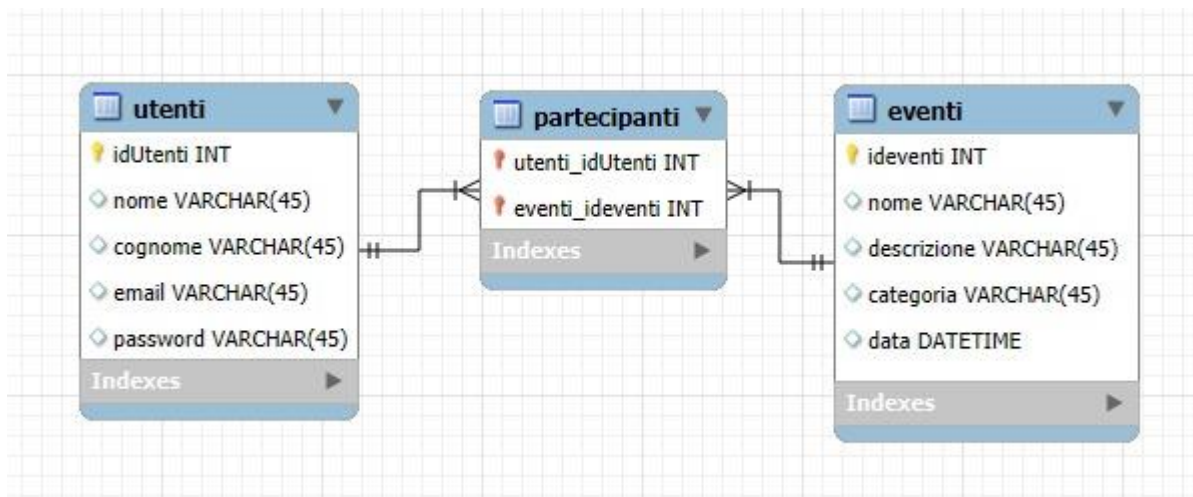
Gli utenti possono anche registrarsi per partecipare agli eventi.

Il progetto è sviluppato usando **Spring Boot**, con **Spring MVC** per la gestione delle richieste HTTP, **Spring Data JPA** per l'accesso ai dati del database relazionale nell'applicazione Java, **Thymeleaf** per la parte di visualizzazione (Front-End), e **MySQL** come database per memorizzare i dati relativi agli eventi e agli utenti.

## 2. Architettura del Sistema:

- **Back-End (Spring Boot):** gestisce la logica di business e l'interazione con il database. Include i servizi per la creazione, modifica, eliminazione degli eventi e per la gestione degli utenti;
- **Front-End (Thymeleaf):** fornisce una semplice interfaccia utente per visualizzare gli eventi, i dettagli e per registrarsi o loggarsi;
- **Database (MySQL):** memorizza i dati degli eventi, degli utenti e le informazioni sulla registrazione agli eventi.

## 3. Progettazione del Database:



Realizzazione della **modellazione concettuale** (entità-relazione) per ottenere una visione semplificata della realtà d'interesse.

In questo caso, abbiamo ipotizzato due entità, ovvero **eventi** e **utenti**, con vari attributi, che tra loro hanno una relazione **multi-a-molti** (N:M), con un'associazione che abbiamo deciso di rappresentare con una terza entità chiamata **partecipanti**, che contiene le chiavi esterne di entrambe le entità.









Table Name:

Charset/Collation:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
 id_utenti	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
 nome	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 cognome	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 email	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 password	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Realizzazione della **modellazione fisica** della tabella **utenti**, all'interno di un database chiamato **gestione\_eventi**, che memorizza informazioni sugli utenti con i seguenti campi:

- **id\_utenti**: chiave primaria dell'utente, con i vincoli **NOT NULL** per indicare che il valore della colonna non potrà mai essere mancante, e **AUTO\_INCREMENT** per generare

automaticamente il valore dell'identificatore univoco durante il popolamento del database;

- **nome:** nome dell'utente;
- **cognome:** cognome dell'utente;
- **email:** indirizzo email dell'utente con un vincolo **UNIQUE** per indicare che il valore della colonna non potrà mai essere duplicato;
- **password:** password per l'accesso dell'utente.









Table Name:

Charset/Collation:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
 id_evento	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
 nome	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 categoria	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 descrizione	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
 data	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Realizzazione della **modellazione fisica** della tabella **eventi**, all'interno di un database chiamato **gestione\_eventi**, che memorizza informazioni sugli eventi, con i seguenti campi:

- **id\_evento:** chiave primaria dell'evento, con i vincoli **NOT NULL** per indicare che i valori della colonna non potranno mai essere mancanti, e **AUTO\_INCREMENT** per generare automaticamente i valori dell'identificatore univoco durante il popolamento del database;
- **nome:** nominativo dell'evento;
- **categoria:** tipologia di evento (es: Musica, Tecnologia, Sport, etc...);
- **descrizione:** una breve descrizione dell'evento;
- **data:** la data dell'evento.






Table Name:

Charset/Collation:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
 utenti_idUtenti	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>
 eventi_ideventi	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Realizzazione della **modellazione fisica** della tabella **partecipanti**, all'interno di un database chiamato **gestione\_eventi**, che gestisce la relazione tra utenti ed eventi, con i seguenti campi:

- **utenti\_idUtenti**: identificatore univoco dell'utente che partecipa (chiave esterna);
- **eventi\_ideventi**: identificatore univoco dell'evento a cui partecipa l'utente (chiave esterna).

Inoltre, la combinazione di questi due campi, è la chiave primaria della tabella.

#### 4. Progettazione Spring:

Dopo aver generato il progetto di base con le dovute dipendenze con **Spring Initializr** e aver compilato il file di configurazione **application.properties** con le dovute componenti, come le informazioni per la connessione al database e la porta in ascolto al server, sono pronto a modellare il mio progetto con le specifiche richieste:

**Utente.java:**

src > main > java > com > example > gestioneEventi > gestioneEventi > model > J Utente.java > Utente > cognome

```
1 package com.example.gestioneEventi.gestioneEventi.model;
2
3 import java.util.List;
4
5 import jakarta.persistence.Column;
6 import jakarta.persistence.Entity;
7 import jakarta.persistence.GeneratedValue;
8 import jakarta.persistence.GenerationType;
9 import jakarta.persistence.Id;
10 import jakarta.persistence.ManyToMany;
11 import jakarta.persistence.Table;
12
13 @Entity
14 @Table(name = "utenti")
15 public class Utente {
16
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long id;
20
21     @Column(name = "nome")
22     private String nome;
23     @Column(name = "cognome")
24     private String cognome;
25     @Column(name = "email")
26     private String email;
27     @Column(name = "password")
28     private String password;
29
30     @ManyToMany(mappedBy = "utenti")
31     private List<Evento> eventi;
32
33     public Utente() {
34
35     }
36
37     public Utente(Long id, String nome, String cognome, String email, String password) {
38         this.id = id;
39         this.nome = nome;
40         this.cognome = cognome;
41         this.email = email;
42         this.password = password;
43     }
44 }
```

```

45     public Long getId() {
46         return id;
47     }
48
49     public String getNome() {
50         return nome;
51     }
52
53     public void setNome(String nome) {
54         this.nome = nome;
55     }
56
57     public String getCognome() {
58         return cognome;
59     }
60
61     public void setCognome(String cognome) {
62         this.cognome = cognome;
63     }
64
65     public String getEmail() {
66         return email;
67     }
68
69     public void setEmail(String email) {
70         this.email = email;
71     }
72
73     public String getPassword() {
74         return password;
75     }
76
77     public void setPassword(String password) {
78         this.password = password;
79     }
80
81     @Override
82     public String toString() {
83         return "Utente [id=" + id + ", nome=" + nome + ", cognome=" + cognome + ", email=" + email + ", password="
84             + password + "]\n";
85     }
86
87 }

```

La seguente classe Java, creato all'interno di un pacchetto chiamato **model**, rappresenta l'entità utente utilizzando **JPA** (Java Persistence API) per la gestione dei dati in un database.

Infatti, ritroviamo delle annotazioni che vanno ad indicare la classe come un'entità JPA, cioè una rappresentazione di una tabella nel database.

Una tra queste è la **@GeneratedValue(strategy = GenerationType.IDENTITY)** associata alla variabile `id`, che va ad indicare che quest'ultima viene generata automaticamente dal database.

Ovviamente, essendo la relazione tra la tabella evento e utente una molti-a-molti, utilizzeremo un'annotazione **@ManyToMany(mappedBy = "utenti")**, dove per l'appunto un utente può partecipare a più eventi e un evento può avere più utenti.

## Evento.java:

```
src > main > java > com > example > gestioneEventi > gestioneEventi > model > J Evento.java > Evento
1  package com.example.gestioneEventi.gestioneEventi.model;
2
3  import java.time.LocalDate;
4  import java.util.List;
5
6  import jakarta.persistence.Column;
7  import jakarta.persistence.Entity;
8  import jakarta.persistence.GeneratedValue;
9  import jakarta.persistence.GenerationType;
10 import jakarta.persistence.Id;
11 import jakarta.persistence.JoinColumn;
12 import jakarta.persistence.JoinTable;
13 import jakarta.persistence.ManyToMany;
14 import jakarta.persistence.Table;
15
16 @Entity
17 @Table(name = "eventi")
18 public class Evento {
19
20     @Id
21     @GeneratedValue(strategy = GenerationType.IDENTITY)
22     private Long id;
23
24     @Column(name = "nome")
25     private String nome;
26     @Column(name = "descrizione")
27     private String descrizione;
28     @Column(name = "categoria")
29     private String categoria;
30     @Column(name = "data")
31     private LocalDate data;
32
33     @ManyToMany
34     @JoinTable(name = "partecipanti", joinColumns =
35     @JoinColumn(name = "id_evento", referencedColumnName = "id"), inverseJoinColumns =
36     @JoinColumn(name = "id_utente", referencedColumnName = "id"))
37     private List<Utente> utenti;
38
39     public Evento() {
40
41     }
42 }
```

```

43     public Evento(Long id, String nome, String descrizione, String categoria, LocalDate data) {
44         this.id = id;
45         this.nome = nome;
46         this.descrizione = descrizione;
47         this.categoria = categoria;
48         this.data = data;
49     }
50
51     public Long getId() {
52         return id;
53     }
54
55     public String getNome() {
56         return nome;
57     }
58
59     public void setNome(String nome) {
60         this.nome = nome;
61     }
62
63     public String getDescrizione() {
64         return descrizione;
65     }
66
67     public void setDescrizione(String descrizione) {
68         this.descrizione = descrizione;
69     }
70
71     public String getCategoria() {
72         return categoria;
73     }
74
75     public void setCategoria(String categoria) {
76         this.categoria = categoria;
77     }
78
79     public LocalDate getData() {
80         return data;
81     }
82
83     public void setData(LocalDate data) {
84         this.data = data;
85     }
86

```

```

87     @Override
88     public String toString() {
89         return "Evento [id=" + id + ", nome=" + nome + ", descrizione=" + descrizione + ", categoria=" + categoria
90             + ", data=" + data + "]";
91     }
92
93
94

```

La seguente classe Java, creato all'interno di un pacchetto chiamato **model**, rappresenta l'entità evento utilizzando **JPA** (Java Persistence API) per la gestione dei dati in un database.

Infatti, ritroviamo delle annotazioni che vanno ad indicare la classe come un'entità JPA, cioè una rappresentazione di una tabella nel database.



Una tra queste è la **@GeneratedValue(strategy = GenerationType.IDENTITY)** associata alla variabile `id`, che va ad indicare che quest'ultima viene generata automaticamente dal database.

Ovviamente, essendo la relazione tra la tabella evento e utente una molti-a-molti, utilizzeremo un'annotazione **@ManyToMany**, dove per l'appunto un utente può partecipare a più eventi e un evento può avere più utenti.

Con la **@JoinTable** ci riferiamo alla tabella partecipanti, ovvero la join, che contiene entrambe le chiavi primarie di entrambe le entità correlate.

Con **@JoinColumn** ci riferiamo alla colonna che rappresenta la chiave esterna della tabella di join riferita all'entità principale (in questo caso la classe **Evento**), mentre con **inverseJoinColumn** ci riferiamo alla colonna che rappresenta la chiave esterna dell'entità inversa (in questo la classe **Utente**) nella tabella partecipanti.