Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA | onfoods | WPS

Piano Nazionale di Ripresa e Resilienza
Missione 4 - Componente 2 – Linea di Investimento 1.3
PE_00000003 - ONFOODS - SPOKE 4

Titolo del progetto
3SMICROBIOTECH4FOOD
SOLUZIONI SMART A SUPPORTO DELLE APPLICAZIONI BIOTECNOLOGICHE MICROBIAL-BASED PER UN RIUTILIZZO PIU' EFFICIENTE DEGLI SCARTI AGROALIMENTARI NELL'INDUSTRIA ALIMENTARE

Codice CUP
D53C24001000001

Soggetto attuatore (RTI)
POLITECNICO DI BARI
UNIVERSITA' DEGLI STUDI DI FOGGIA
WPS SRL
BONASSISALAB SRL

Titolo documento
D3.1 WP3 results deliverable

Autore	**Gabriele Lococciolo**
Controllato	**Alessandro Buscicchio**
Approvato	**Alessandro Buscicchio**
Versione	**1.4**
Data	**31 Agosto 2025**

Sommario

1. Introduzione 4
2. Software utilizzati per lo sviluppo della piattaforma 4
3. Struttura e entità back-end 5
3.1 Entità primo macchinario 6
3.2 Entità per il secondo macchinario 8
3.3 Entità per il terzo macchinario 9
4. Front-end 11
4.1 Upload 16
5. Progettazione database 19
5.1 Organization strains 21
5.2 Bacteria Strains 22
5.3 Organization standard 22
5.4 ExperimentalConditions,NewExperimentalConditions, LastExperimentalConditions 23
5.5 ExperimentalData,ExperimentalDataNew, LastExperimentalData 26
5.6 Matrix 28
5.7 User 29

Figura 1:ExperimentalData 6
Figura 2:ExperimentalConditions 7
Figura 3:ExperimentalDataNew 8
Figura 4: Entità NewExperimentalConditions 9
Figura 5:LastExperimentalData 10
Figura 6:Interfaccia di login 12
Figura 7:Schermata di registrazione 13
Figura 8: Dashboard interfaccia 14
Figura 9:Storico esperimenti 14
Figura 10:Recupero degli esperimenti 15
Figura 11:Pagina dettagli esperimento 15
Figura 12:Esperimento creato 16
Figura 13:Pagina di upload 16
Figura 14:Componente Uplaod 17
Figura 15:Funzione uploadFile 18
Figura 16:Navigazione verso la pagina Dettaglio 18
Figura 17:Schema grafico con le tecnologie e il loro ruolo 20
Figura 18: Classe Organization standard 22
Figura 19:tabella experimental_conditions 24
Figura 20: tabella new_experimental_conditions 25
Figura 21:tabella last_experimental_conditions 25
Figura 22:tabella experimental_data 26
Figura 23:tabella experimental_data_new 27
Figura 24:tabella experimental_data_last 28
Figura 25:Entità Matrix 28
Figura 26:Entità User 29

1. Introduzione

L'obiettivo del progetto è archiviare in maniera sistematica e strutturata tutti i dati sperimentali all'interno di un database centralizzato, che fungerà da base per successive analisi avanzate. Tali dati saranno infatti utilizzati per sviluppare algoritmi di Machine Learning in grado di suggerire le combinazioni di ceppi batterici più adatte a ottenere una curva di crescita ottimale, supportando così il processo decisionale degli analisti e migliorando l'efficienza delle sperimentazioni.
Attualmente, prima di ogni analisi, gli analisti registrano l'organizzazione dei ceppi su piastra all'interno di un documento Word, così da documentare la disposizione e le condizioni iniziali dell'esperimento. Una volta definita la composizione di ciascuna piastra, vengono configurati i parametri iniziali nel software di gestione del macchinario, che opera su matrici specifiche in funzione dell'analisi da condurre. Al termine del processo, lo strumento genera un file Excel contenente i dati grezzi di output, che rappresentano la base per le successive elaborazioni.
La necessità di integrare e normalizzare queste informazioni nasce dall'attuale frammentazione dei dati, che rende difficile non solo la consultazione e il confronto tra esperimenti differenti, ma anche l'automatizzazione delle analisi. Il progetto si propone dunque di sviluppare una piattaforma web che consenta la raccolta, l'organizzazione e la consultazione dei dati in maniera più semplice ed efficiente, riducendo errori manuali e aumentando la tracciabilità dei processi.
Parallelamente, la progettazione e realizzazione di un database relazionale costituisce un passo fondamentale per garantire coerenza, sicurezza e scalabilità del sistema. Una base dati ben strutturata non solo supporterà le esigenze attuali di archiviazione, ma consentirà anche in futuro l'applicazione di metodologie di analisi predittiva e l'addestramento di modelli di apprendimento automatico, con l'obiettivo di ottimizzare ulteriormente la gestione e la resa degli esperimenti.
In questo documento sarà quindi descritto in breve:

- lo sviluppo della piattaforma web e delle sue funzionalità principali;
- la progettazione e la struttura del database, con particolare attenzione ai criteri di normalizzazione e alle relazioni tra le entità coinvolte.

2. Software utilizzati per lo sviluppo della piattaforma

Per lo sviluppo del backend, viene adottato l'ambiente di sviluppo integrato (IDE) IntelliJ. Questo strumento si distingue per le sue funzionalità avanzate, che includono strumenti di refactoring, testing e debugging, rendendolo una scelta ottimale per la gestione di applicazioni complesse e per progetti a lungo termine.
Una delle sue caratteristiche principali è la funzionalità di completamento automatico, che opera in modo contestuale. Questa non si limita a completare le parole chiave, ma suggerisce in maniera intelligente metodi, variabili e classi, facilitando la scrittura del codice. La funzione di refactoring è ugualmente cruciale: consente la riorganizzazione sicura del codice, preservando la logica di base e contribuendo a mantenere una codebase pulita e facilmente manutenibile.
IntelliJ si contraddistingue anche per la sua interfaccia intuitiva e user-friendly. Il layout pulito e gli efficienti strumenti di navigazione permettono agli sviluppatori di muoversi

rapidamente tra file, metodi e classi. Ciò risulta particolarmente vantaggioso in progetti di grandi dimensioni, in quanto facilita la comprensione delle interconnessioni tra le diverse componenti del codice.
La gestione dei progetti rappresenta un altro punto di forza. L'IDE supporta nativamente sistemi di build come Maven, Gradle e Ant, integrando in modo fluido la gestione delle dipendenze e la configurazione del progetto. L'integrazione con sistemi di controllo versione come Git è completa, permettendo la gestione del versioning e del branching direttamente all'interno dell'ambiente di sviluppo.
Infine, il supporto per il debugging è notevolmente potente. IntelliJ offre un debugger interattivo che permette l'esecuzione del codice passo dopo passo, il monitoraggio di variabili ed espressioni e l'ispezione dello stato completo dell'applicazione, semplificando in modo significativo il processo di identificazione e risoluzione dei bug.
Per lo sviluppo del frontend, si utilizza VS Code, un editor di codice leggero e performante. A differenza degli IDE più completi, si avvia rapidamente e richiede risorse di sistema ridotte. VS Code supporta nativamente una vasta gamma di linguaggi di programmazione, tra cui JavaScript, TypeScript, HTML/CSS, Python, Go, C++, Java e Rust.
La sua principale flessibilità deriva dal vasto ecosistema di estensioni, che consentono di aggiungere il supporto per linguaggi e framework specifici. Grazie a questa caratteristica, VS Code è diventato uno strumento estremamente popolare e versatile, in particolare nel settore dello sviluppo web.

3. Struttura e entità back-end

Il progetto è stato creato utilizzando Spring Initializr, un generatore online che permette di creare rapidamente nuovi progetti con dipendenze già configurate
Le dipendenze utilizzate sono:

- **spring-boot-starter-web**: SpringBoot, in questo modo, configura automaticamente Spring MVC;
- **spring-boot-starter-data-jpa**: Aggiungendo questa dipendenza ottengo Spring Data JPA per abbreviare la scrittura di accesso ai dati. Hibernate che è un'implementazione predefinita di JPA per la persistenza dei dati. Spring ORM che è un modulo di integrazione tra Spring e JPA/Hibernate. HikariCP che è un pool di connessioni predefinito per migliorare le performance;
- **mysql-connector-j**: è un driver JDBC che permette alle applicazioni Java di connettersi ad un database MySQL e di eseguire query SQL;
- **spring-boot-starter-security**: Aggiungendo questa dipendenza si può configurare un sistema di sicurezza con l'autenticazione da parte degli utenti, autorizzazioni, gestioni delle sessioni e molto altro;
- **spring-boot-starter-mail**: Questa dipendenza fornisce il supporto automatico per Spring Email integrando il client JavaMail che permette di inviare email, aggiungere allegati e usare il server SMTP che deve essere

configurato in un file di configurazione come application.properties o application.yml

- **poi-ooxml**: è un modulo della libreria Apache POI che permette di lavorare con i formati di file Microsoft Office Open XML, infatti in questo caso è stata utilizzata per leggere e scrivere file Excel;
- **jackson-databind**: un modulo della libreria Jackson utilizzato per la serializzazione e deserializzazione dei dati tra oggetti Java e formati JSON.

Sono state definite le entità destinate alla memorizzazione dei dati provenienti dai macchinari.
Tali dati vengono estratti dai file in formato Excel generati dai macchinari stessi.

3.1 Entità primo macchinario

```java
@Entity  11 usages
@Table(name = "experimental_data")
@Data
@NoArgsConstructor
public class ExperimentalData {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @ManyToOne
    @JoinColumn(name = "filename_ID", nullable = false)  // Foreign Key
    private ExperimentalConditions experimentalConditions;

    //la colonna filename_ID in experimental_data è una chiave esterna che fa riferimento alla
    // ID di experimental_conditions

    private Integer wavelenghts;
    private LocalTime intervalTime;
    private Float wavelengthTemp;

    // Colonne A1-H12
    private String a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12;
    private String b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12;
    private String c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12;
    private String d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12;
    private String e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12;
    private String f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12;
    private String g1, g2, g3, g4, g5, g6, g7, g8, g9, g10, g11, g12;
    private String h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12;
```

Figura 1:ExperimentalData

La classe ExperimentalData è un model in Spring Boot che rappresenta la tabella experimental_data all'interno del database relazionale. Per mappare correttamente la classe alla tabella, vengono utilizzate diverse annotazioni
La relazione tra ExperimentalData e ExperimentalConditions è definita tramite:

- @ManyToOne: Stabilisce una relazione molti-a-uno, in cui molti record di ExperimentalData possono essere associati a un singolo record di ExperimentalConditions

- @JoinColumn(name = "filename_ID", nullable = false): Specifica che la colonna di join nella tabella experimental_data si chiama filename_ID. L'attributo nullable = false impone che questo campo non possa essere nullo, garantendo l'integrità referenziale

In sostanza, filename_ID è una chiave esterna che collega la tabella experimental_data alla tabella experimental_conditions. Questo collegamento avviene tramite la colonna ID di experimental_conditions, che viene associata alla colonna filename_ID di experimental_data.

```
public class ExperimentalConditions {
    @Id 2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(name = "filename",unique = true) 2 usages
    private String filename;
    @Column(name = "software_version") 2 usages
    private String softwareVersion;
    @Column(name = "plate_number") 2 usages
    private String plateNumber;
    @Column(name = "date") 2 usages
    private LocalDate date;
    @Column(name = "time") 2 usages
    private LocalTime time;
    @Column(name = "reader_type") 2 usages
    private String readerType;
    @Column(name = "reader_serial_no") 2 usages
    private String readerSerialNo;
    @Column(name = "plate_type") 2 usages
    private String plateType;
    @Column(name = "set_temperature") 2 usages
    private String setTemperature;
    @Column(name = "start_kinetic") 2 usages
    private String startKinetic;
    @Column(name = "shake_linear") 2 usages
    private String shakeLinear;
    @Column(name = "shake_frequency") 2 usages
    private String shakeFrequency;
    @Column(name = "`read`") 2 usages
    private String read;
    @Column(name = "read_wavelengths") 2 usages
    private String readWavelengths;
```

Figura 2:ExperimentalConditions

La figura 2 rappresenta una classe di un'entità JPA chiamata ExperimentalConditions, che è mappata su una tabella del database chiamata experimental_conditions.
Oltre alle varie variabili che mappano direttamente le colonne della tabella, la classe definisce una relazione uno-a-molti(One-to-Many) tra due entità ExperimentalConditions e ExperiementalData. La relazione implica che un oggetto ExperimentalConditions può avere molti oggetti ExperimentalData.

3.2 Entità per il secondo macchinario

```java
@Table(name = "experimental_data_new")
@Data
public class ExperimentalDataNew {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String times;
    private Integer length;



    @ManyToOne
    @JoinColumn(name = "filename_ID", nullable = false)  // Foreign Key
    private NewExperimentalConditions newExperimentalConditions;

    // Riga A
    @Column(length = 10)
    private String a01, a02, a03, a04, a05, a06, a07, a08, a09, a10, a11, a12;
    // Riga B
    @Column(length = 10)
    private String b01, b02, b03, b04, b05, b06, b07, b08, b09, b10, b11, b12;
    // Riga C
    @Column(length = 10)
    private String c01, c02, c03, c04, c05, c06, c07, c08, c09, c10, c11, c12;
    // Riga D
    @Column(length = 10)
    private String d01, d02, d03, d04, d05, d06, d07, d08, d09, d10, d11, d12;
    // Riga E
    @Column(length = 10)
    private String e01, e02, e03, e04, e05, e06, e07, e08, e09, e10, e11, e12;
    // Riga F
    @Column(length = 10)
    private String f01, f02, f03, f04, f05, f06, f07, f08, f09, f10, f11, f12;
    // Riga G
    @Column(length = 10)
    private String g01, g02, g03, g04, g05, g06, g07, g08, g09, g10, g11, g12;
    // Riga H
    @Column(length = 10)
    private String h01, h02, h03, h04, h05, h06, h07, h08, h09, h10, h11, h12;
```

Figura 3:ExperimentalDataNew

La classe ExperimentalDataNew è un model in Spring Boot che rappresenta la tabella experimental_data_new all'interno del database relazionale
Questa classe è simile a ExperimentalData, cambiano solamente i nomi dei pozzetti.
Anche qui abbiamo una relazione ma con la tabella NewExperimentalConditions.
In sostanza, filename_ID è una chiave esterna che collega la tabella experimental_data_new alla tabella new_experimental_conditions.

```java
@Entity  21 usages  tagarelli *
@Table(name = "new_experimental_conditions")
@Data
public class NewExperimentalConditions {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String user;
    private String path;
    private Integer testId;
    private String testName;
    private LocalDate date;
    private LocalTime time;
    private String measurementType;
    private String microplateName;
    private String noOfCycles;
    private String cycleTime;
    private String noOfFlashesPerWellAndCycle;
    private String wavelengthRange;
    private String wavelengtStepWidth;
    private String shake;
    private String movement;
    private Integer frequency;
    private Integer times;
    private String settingTime;
    private String readingDirection;
    private Integer targetTemperature;
    @JsonIgnore
    @OneToMany(mappedBy = "newExperimentalConditions", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<ExperimentalDataNew> experimentalDataNew;
```

Figura 4: Entità NewExperimentalConditions

Questa classe rappresenta un'entità JPA mappata sulla tabella new_experimental_conditions del database. Viene utilizzata per memorizzare le condizioni sperimentali associate ai dati provenienti dal secondo macchinario.
Come identificatore è utilizzato un campo id, generato automaticamente. Contiene campi come user, path, testId, testName e molti altri che descrivono le specifiche dei test. Contiene una relazione uno-a-molti con la classe ExperimentalDataNew

3.3 Entità per il terzo macchinario

```java
@Entity  18 usages  tagarelli
@Access(AccessType.FIELD)
@Table(name = "experimental_data_last")
@Data
public class LastExperimentalData {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String times;
    private Integer length;


    // Riga A
    @Column(length = 10)
    private String a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12;
    // Riga B
    @Column(length = 10)
    private String b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12;
    // Riga C
    @Column(length = 10)
    private String c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12;
    // Riga D
    @Column(length = 10)
    private String d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12;
    // Riga E
    @Column(length = 10)
    private String e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12;
    // Riga F
    @Column(length = 10)
    private String f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12;
    // Riga G
    @Column(length = 10)
    private String g1, g2, g3, g4, g5, g6, g7, g8, g9, g10, g11, g12;
    // Riga H
    @Column(length = 10)
    private String h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12;

    @ManyToOne
    @JoinColumn(name = "filename_ID", nullable = false)  // Foreign Key
    private LastExperimentalConditions lastExperimentalConditions;
```

Figura 5:LastExperimentalData

La classe LastExperimentalData è un model in Spring Boot che rappresenta la tabella experimental_data_last all'interno del database relazionale
Questa classe è simile a ExperimentalDataNew, cambiano solamente i nomi dei pozzetti.
Anche qui abbiamo una relazione ma con l'entità LastExperimentalConditions.
In sostanza, filename_ID è una chiave esterna che collega la tabella experimental_data_last alla tabella last_experimental_conditions.

```java
@Entity 15 usages  tagarelli
@Table(name = "last_experimental_conditions")
@Data
public class LastExperimentalConditions {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String user;
    private String path;
    private Integer testId;
    private String testName;
    private LocalDate date;
    private LocalTime time;
    private String measurementType;
    private String microplateName;
    private String noOfCycles;
    private String cycleTime;
    private Integer noOfFlashesPerWell;
    private String wavelengthRange;
    private Integer wavelengtStepWidth;
    private Integer shakingWidth;
    private String shakingMode;
    private String additionalShakingTime;
    private String settingTime;
    private String readingDirection;
    private Integer targetTemperature;
    @JsonIgnore
    @OneToMany(mappedBy = "lastExperimentalConditions", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<LastExperimentalData> lastExperimentalData;
```

Questa classe rappresenta un'entità JPA mappata sulla tabella last_experimental_conditions del database. Viene utilizzata per memorizzare le condizioni sperimentali associate ai dati provenienti dal secondo macchinario.
Come identificatore è utilizzato un campo id, generato automaticamente. Contiene campi come user, path, testId, testName e molti altri che descrivono le specifiche dei test. Contiene una relazione uno-a-molti con la classe LastExperimentalData
Ogni Service di ciascuna entità contiene dei metodi per memorizzare i dati che vengono recuperati dai file Excel.
Poiché la struttura dei file Excel generati varia a seconda del macchinario, è stato implementato un metodo specifico per ciascun macchinario, finalizzato all'estrazione e alla normalizzazione dei dati.
Per l'elaborazione dei file Excel è stata scelta la libreria Apache POI che offre una serie di metodi avanzati per la lettura e la manipolazione dei fogli di calcolo nei formati .xls e .xlsx

4. Front-end

L'interfaccia della piattaforma si presenta con la schermata di login per consentire agli utenti di autenticarsi al sistema tramite l'inserimento delle proprie credenziali(nome utente e password). L'interazione con il backend è mediata da un servizio chiamato AuthService.
I campi Username e password del form utilizzano la direttiva [(ngModel)] per il two-way binding, legando i valori del form alle proprietà del componente.
Il campo password è gestito con il componente p-password che offre funzionalità avanzate come la possibilità di mostrare o nascondere il contenuto tramite un'icona.

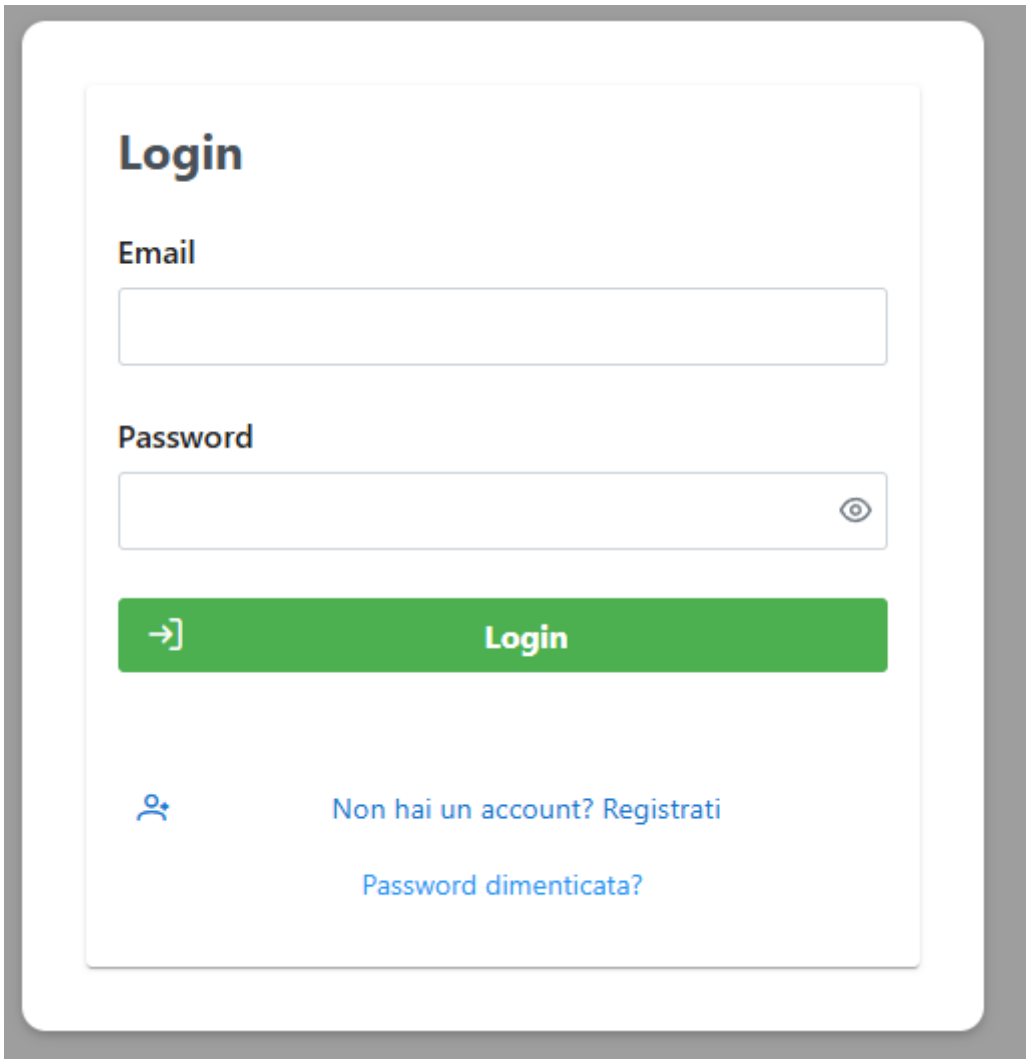

Figura 6:Interfaccia di login

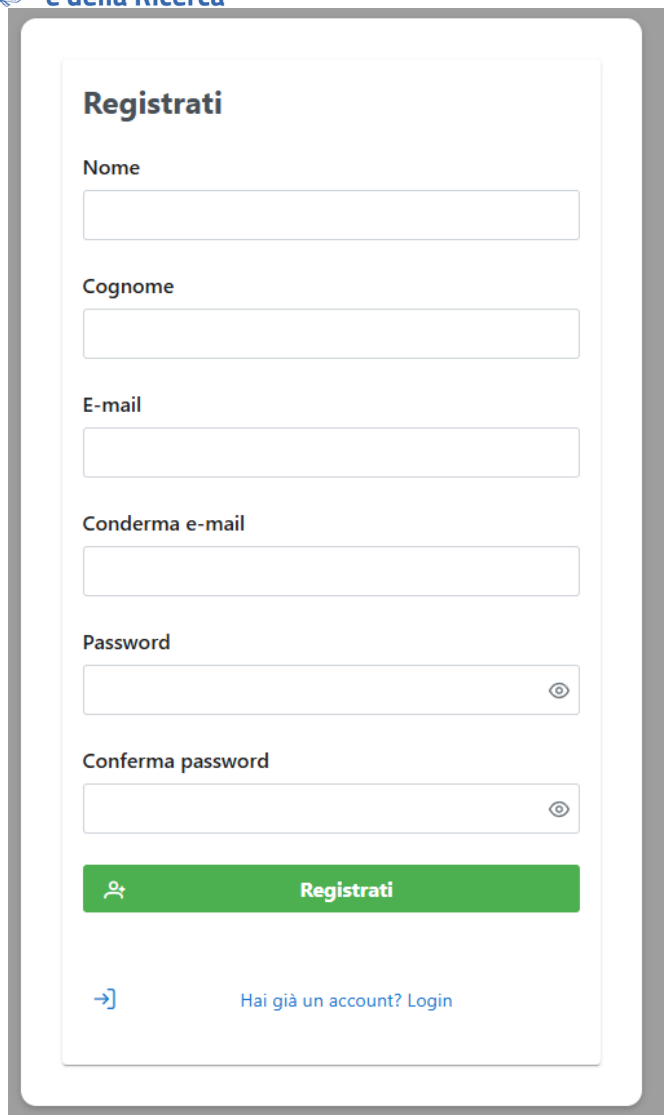

Figura 7:Schermata di registrazione

I campi previsti per il modulo registrazione sono: nome, cognome, e-mail, Conferma e-mail. Password e Conferma Password.
Lo sviluppo di questa interfaccia è stato realizzato utilizzando Angular, un framework open-source per lo sviluppo di applicazioni web.
Angular è particolarmente adatto alla creazione di Single Page Applications (SPA) dinamiche e interattive. Per migliorare l'aspetto e la funzionalità dell'interfaccia, è stata integrata la libreria PrimeNG, che offre un'ampia gamma di componenti pronti all'uso, come moduli, tabelle, griglie, layout e altro ancora.
Un componente in Angular è una classe TypeScript che gestisce una porzione dell'interfaccia utente e la logica associata. Ogni componente è composto da tre elementi principali:

- Classe (TypeScript): Gestisce i dati e la logica del componente, definendo metodi e proprietà.
- Template (HTML): Definisce la struttura visibile dell'interfaccia.
- Style (CSS o SCSS): Contiene gli stili specifici del componente.

Ogni componente ha un selector, un identificatore univoco che consente di richiamarlo all'interno di altri componenti

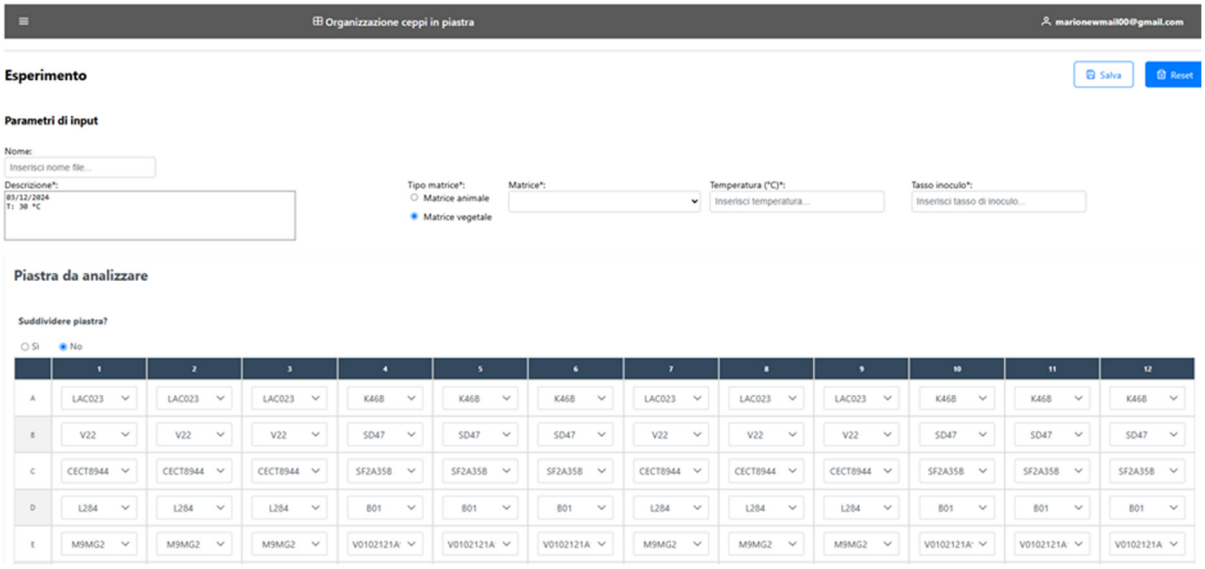

Figura 8: Dashboard interfaccia

L'interfaccia presenta una sezione superiore dedicata all'inserimento dei parametri input, tra cui il nome, la descrizione, la temperatura e il tasso di inoculo.
In base al tipo di matrice selezionato, l'utente può scegliere tra due opzioni tramite radio button: "Matrice vegetale" e "Matrice animale". Nella selezione associata all'etichetta "Matrice" vengono caricati i nomi corrispondenti ai tipi di matrice disponibili. Questi dati vengono recuperati dalla tabella "Matrix" del database.

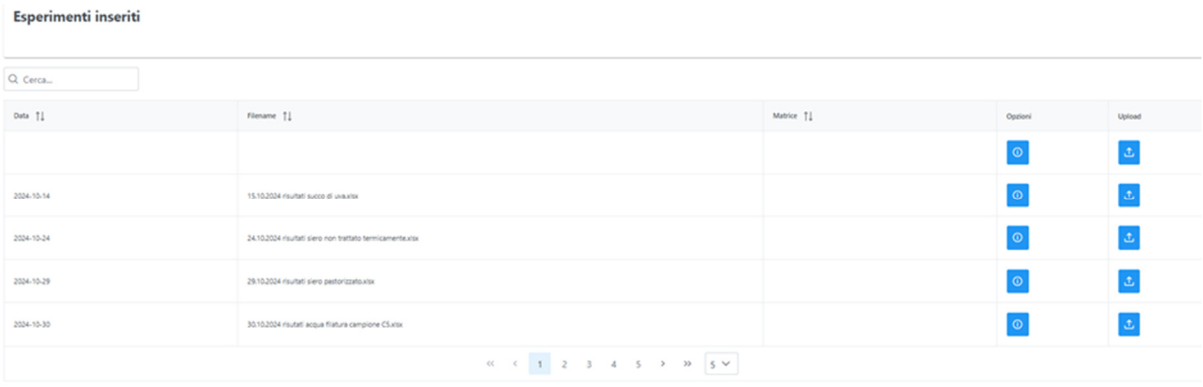

Figura 9:Storico esperimenti

In questa interfaccia sono visualizzati gli esperimenti precedentemente inseriti all'interno del sistema.
La tabella è gestita dal componente p-table fornito da PrimeNG. La tabella è configurata in questo modo:

- Paginazione abilitata con possibilità di scegliere tra 5,10,20 righe per pagina

- Ricerca attivata sui campi data, filename e matrice
- Data
- Filename
- Matrice
- Opzioni: Icona che permette di navigare verso la pagina dettagli
- Upload: Icona che permette di caricare i file Excel associati all'esperimento

Ogni riga della tabella rappresenta un singolo esperimento ottenuto tramite la variabile plates.

- plates.data: mostra i dati nella colonna data
- plates.filename: mostra i dati nella colonna filename
- plates.matrice: mostra i dati nella colonna matrice
- Colonna con l'icona pi pi-info-circle che richiama il metodo navigateToConditionDetail(plates.id) che reindirizza l'utente alla pagina dettagli dell'esperimento selezionato
- Colonna con l'icona pi pi-upload che attiva la funzione navigateToUpload(plates.id) per caricare il file excel legato al specifico esperimento selezionato

```
<tr>
  <td>{{ plates.data }}</td>
  <td>{{ plates.filename }}</td>
  <td>{{ plates.matrice }}</td>
  <td>
    <button pButton icon="pi pi-info-circle" (click)="navigateToConditionDetail(plates.id)"></button>
  </td>
  <td>
    <button pButton icon="pi pi-upload" (click)="navigateToUpload(plates.id)"></button>
  </td>

</tr>
```

Figura 10:Recupero degli esperimenti

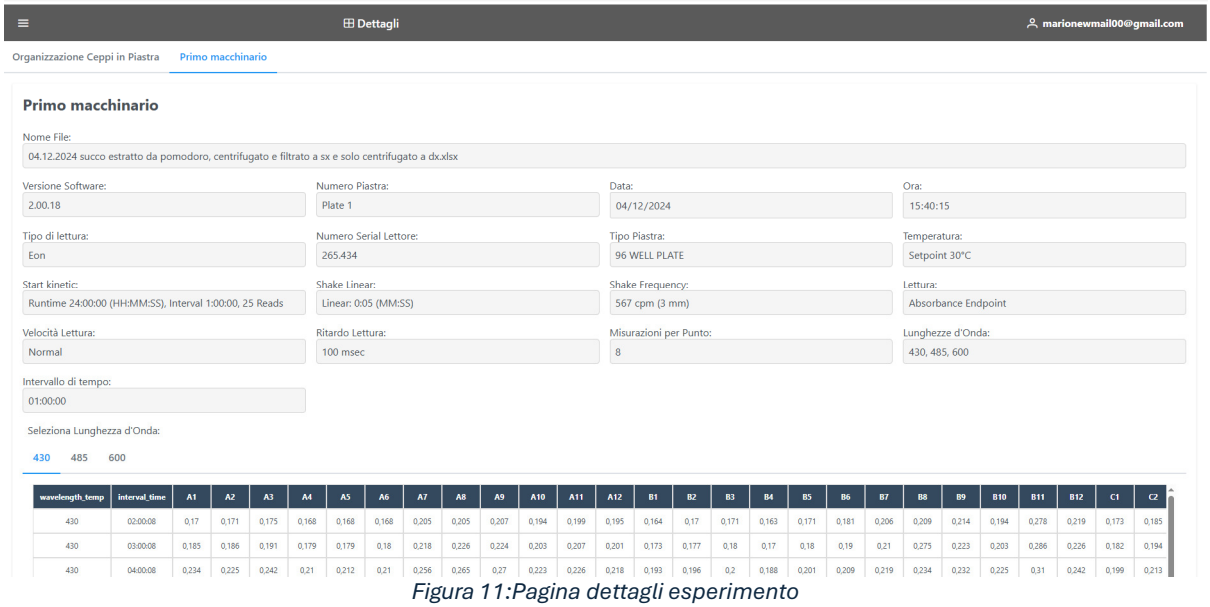

Figura 11:Pagina dettagli esperimento

Cliccando sull'icona opzioni dello specifico esperimento, l'utente verrà indirizzato alla pagina dettagli dell'esperimento.
Il componente Dettagli è composto da 2 schede:

- La prima scheda ha il titolo di Organizzazione Ceppi in Piastra
- La seconda scheda ha il titolo di "Primo macchinario" o "Secondo macchinario" o "Terzo macchinario" in base al file Excel caricato

4.1 Upload

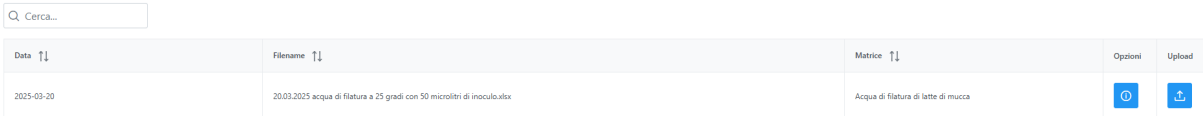

Figura 12:Esperimento creato

Quando l'utente clicca sul pulsante upload per il caricare il file Excel relativo ad un esperimento specifico verrà indirizzato a questa pagina:

Figura 13:Pagina di upload

```
<app-navbar [activeComponentName]="activeComponentName"></app-navbar>

<p-card header="Upload file Excel">
</p-card>

<p-fileUpload
  name="file"
  (onSelect)="onFileSelected($event)"
  [auto]="false"
  [customUpload]="true"
  [showUploadButton]="false"
  [showCancelButton]="false"
  accept=".xlsx,.xls"
  chooseLabel="Scegli file"
>
  <ng-template pTemplate="empty">
    <div class="flex align-items-center justify-content-center flex-column">
      <i class="pi pi-cloud-upload border-2 border-circle p-5 text-8xl text-400 border-400"></i>
      <p class="p-mt-3">Trascina qui il tuo file Excel o seleziona il file da "Scegli file"</p>
    </div>
  </ng-template>
</p-fileUpload>


<p-button
  label="Carica"
  icon="pi pi-upload"
  (click)="uploadFile()"
  [disabled]="!selectedFile">
</p-button>
<p-toast></p-toast> <!-- Per le notifiche -->
```

Figura 14:Componente Uplaod

Il componente Upload è progettato per permettere agli utenti di caricare un file Excel e inviarlo al back-end tramite una chiamata POST. Durante il processo:

- Viene letto un eventuale parametro id dall'URL
- Il file viene associato a tale id (se presente) e inviato al server
- Viene mostrato un messaggio di successo o errore tramite pop-up(toast).
- Al termine, se il caricamento ha successo e un recordId è presente, l'utente viene reindirizzato automaticamente alla pagina di dettaglio del record.

```
async uploadFile() {
  if (this.selectedFile) {
    const formData = new FormData();
    formData.append('file', this.selectedFile);
    if(this.recordId != null){
      formData.append('id', this.recordId.toString());
      console.log("ID recuperato: " + this.recordId);
    }
    const token = localStorage.getItem('authToken');

    try {
      const response = await fetch(`${environment.apiUrl}/api/excel/upload`, {
        method: 'POST',
        headers: {
          ...(token && { 'Authorization': `Bearer ${token}` })
        },
        body: formData
      });

      if (response.ok) {
        const result = await response.text();
        console.log('File caricato con successo!', result);
        this.messageService.add({ severity: 'success', summary: 'Successo', detail: 'File caricato con successo!' });

        this.selectedFile = null;

        if(this.recordId != null){
          const [message, fileNameID] = result.split('/');
          console.log("Message:" + message);
          console.log("FileNameID: " + fileNameID);

          console.log('ID recuperato dalla route:', fileNameID);
         // if (fileNameID) {
            this.router.navigate(['/dettaglio/', this.recordId]);
          //}
        }
```

Figura 15:Funzione uploadFile

La funzione uploadFile() gestisce il processo completo di invio del file al backend. Verifica se il file è stato selezionato e crea un oggetto formData al quale vengono aggiunti il file selezionato(selectedFile), l'eventuale recordId e il token di autenticazione salvato in localStorage

Se la risposta è positiva(response.ok) viene mostrato un messaggio di successo all'utente tramite PrimeNG. Se recordId è presente, l'utente viene reindirizzato alla pagina /dettaglio/{recordId}

```
this.router.navigate(['/dettaglio/', this.recordId]);
```

Figura 16:Navigazione verso la pagina Dettaglio

5. Progettazione database

La progettazione del database ha previsto l'impiego di un insieme di tecnologie, selezionate per garantire efficienza, scalabilità e coerenza tra le fasi di progettazione, implementazione e gestione del database. In particolare, le attività si sono concentrate sull'uso di SQL per la definizione e manipolazione dei dati, MySQL Workbench per la progettazione visuale e l'amministrazione del database, e della piattaforma Java con il Spring Framework per l'implementazione logica dell'applicazione e l'interazione con il sistema di persistenza.

SQL (Structured Query Language) costituisce lo standard di riferimento per la gestione dei database relazionali. Esso consente di definire la struttura logica delle tabelle, le relazioni e i vincoli, oltre a fornire strumenti per l'interrogazione, l'inserimento, l'aggiornamento e la cancellazione dei dati. Nel contesto di questo progetto, SQL rappresenta il linguaggio fondamentale per l'interfacciamento con il database MySQL.

Per la progettazione e gestione del database è stato impiegato MySQL Workbench, strumento ufficiale fornito da Oracle. MySQL Workbench ha permesso di modellare graficamente lo schema del database attraverso diagrammi EER (Enhanced Entity–Relationship), di eseguire operazioni di forward engineering per generare automaticamente la struttura fisica a partire dal modello concettuale e di amministrare agevolmente il DBMS durante lo sviluppo.

Sul versante applicativo, è stato scelto l'ecosistema Java, che offre portabilità, robustezza e un'ampia disponibilità di librerie e framework. In particolare, l'applicazione è stata sviluppata utilizzando il Spring Framework, piattaforma open source che semplifica la creazione di applicazioni modulari e scalabili. All'interno di Spring, è stata utilizzata la Java Persistence API (JPA) per la gestione della persistenza dei dati, astratta da un'implementazione ORM come Hibernate. Questa combinazione ha consentito di interagire con il database attraverso oggetti Java, evitando la scrittura diretta di query SQL nella maggior parte dei casi e riducendo così il codice boilerplate.

L'integrazione di queste tecnologie ha permesso di coprire in maniera efficace tutte le fasi del progetto: dalla modellazione concettuale e logica dei dati, alla realizzazione fisica del database, fino alla gestione e persistenza delle informazioni a livello applicativo, garantendo coerenza e affidabilità lungo l'intero ciclo di sviluppo.

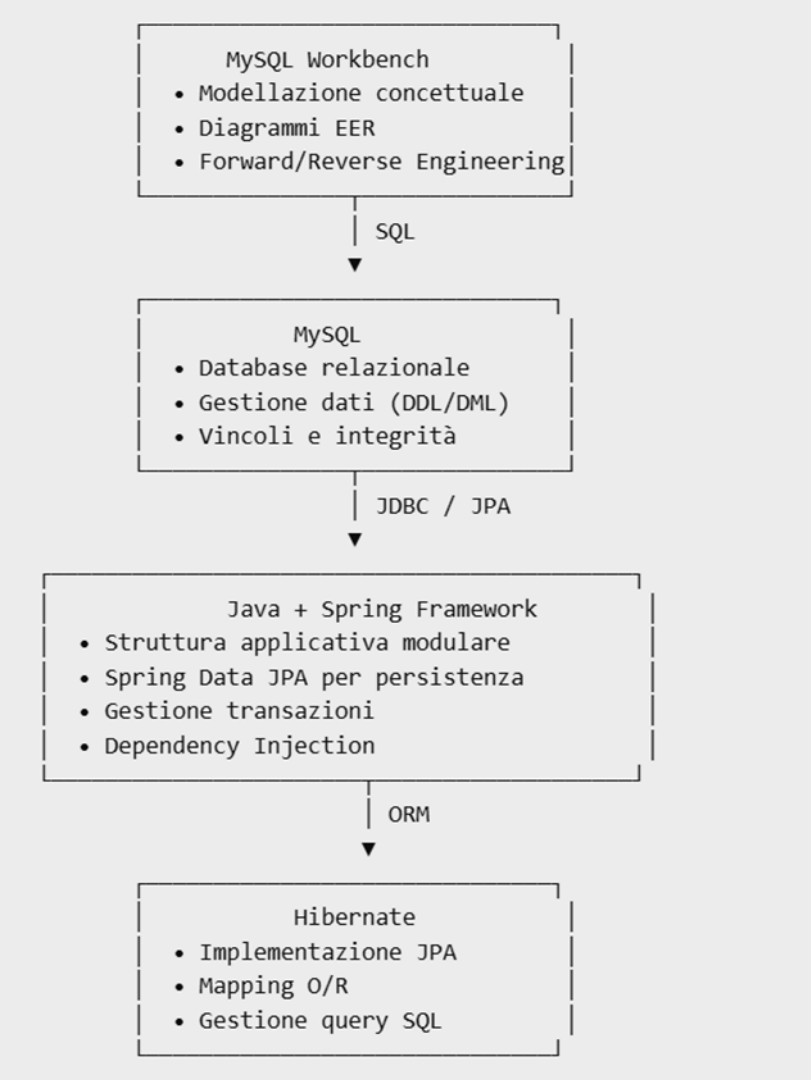

Figura 17:Schema grafico con le tecnologie e il loro ruolo

Il progetto 3SMICROBIOTECH4FOOD si inserisce nell'ambito della valorizzazione e gestione sostenibile degli scarti alimentari. In particolare, esso prevede l'acquisizione, l'analisi e l'archiviazione di dati relativi a esperimenti condotti su matrici biologiche, con l'obiettivo di studiare e caratterizzare specifici ceppi microbici di interesse.
Per supportare le funzionalità applicative, si è resa necessaria la progettazione e realizzazione di un database relazionale in grado di memorizzare in maniera strutturata e coerente le informazioni prodotte durante le attività sperimentali.
La decisione di utilizzare MySQL nel contesto del progetto è stata dettata da una serie di fattori strategici e tecnici:

- **Diffusione e supporto**: MySQL è ampiamente documentato, con una grande comunità di sviluppatori e un ricco ecosistema di strumenti e librerie compatibili.
- **Compatibilità con gli strumenti di sviluppo**: MySQL è facilmente integrabile con MySQL Workbench per la modellazione grafica, e con tecnologie come Java + Spring Data JPA + Hibernate per la gestione della persistenza a livello applicativo.
- **Prestazioni**: È ottimizzato per eseguire query ad alte prestazioni, supportando l'uso di indici e strategie di ottimizzazione automatica.

- **Affidabilità**: Il supporto per transazioni ACID garantisce la coerenza e la sicurezza dei dati
- **Licenza e costi**: MySQL è disponibile in versione open source, rendendolo particolarmente adatto a progetti di ricerca e sviluppo
- **Portabilità**: È multipiattaforma e può essere installato su sistemi operativi diversi (Windows, Linux, macOS), facilitando la distribuzione e la replicabilità dell'ambiente di sviluppo.

Il database "onfoods" contiene le seguenti tabelle:

- experimental_conditions
- experimental_data
- new_experimental_conditions
- experimental_data_new
- last_experimental_conditions
- experimental_data_last
- organization_standard
- bacteria_strains
- organization_strains_on_plate

5.1 Organization strains

La tabella organization_strains_on_plate costituisce un elemento centrale del database, finalizzato alla memorizzazione e alla gestione delle informazioni relative ai ceppi microbici disposti su piastre di coltura, in relazione agli esperimenti condotti nell'ambito del progetto 3SMICROBIOTECH4FOOD. La tabella è organizzata in più gruppi logici:

1. **Identificazione dell'esperimento**: id, date, filename_ID e filename che sono dei riferimenti a un file output prodotto dal macchinario associato all'esperimento e la descrizione
2. **Dati dei pozzi della piastra**: campi da a1 a h12: ciascun campo corrisponde a un pozzo specifico della piastra di coltura. Il valore registrato in ogni campo rappresenta il ceppo microbico
3. **Parametri sperimentali**:
   - matrice_sterile, matrice_non_sterile: tipologia di matrice utilizzata, distinguendo tra condizione sterile e non sterile
   - temperatura
   - tasso_inoculo
   - associazione_sterile e associazione_non_sterile
   - matrice: indica il nome della matrice
   - tipo_matrice
4. **Informazioni operative**
   - Username: Nome dell'operatore che ha registrato l'esperimento
   - Macchinario: Identifica lo strumento utilizzato per elaborare i dati

5.2 Bacteria Strains

```java
@Entity  10 usages  tagarelli *
@Table(name = "bacteria_strains")
@Data
@NoArgsConstructor
public class BacteriaStrains {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(name = "name")
    private String name;
```

La classe BacteriaStrains contiene l'id che rappresenta la chiave primaria della tabella e il nome che rappresenta il nome del ceppo batterico ed è mappato alla colonna name della tabella. Nella tabella bacteria_strains vengono memorizzati i nomi dei ceppi batterici, i quali sono resi disponibili nell'interfaccia utente per la compilazione della matrice. L'interfaccia, inoltre, mette a disposizione un campo di input testuale che consente all'utente di inserire manualmente il nome di un nuovo ceppo. Tale valore, una volta confermato, viene salvato all'interno della tabella

5.3 Organization standard

```java
public class OrganizationStandard {
    @Id 2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(name = "date") 2 usages
    private LocalDate date;
    @Column(name = "description") 2 usages
    private String description;
    private String matrice_sterile; 2 usages
    private String matrice_non_sterile; 2 usages
    @Column(name = "temperatura", precision = 4, scale = 1) 2 usages
    @JsonFormat(shape = JsonFormat.Shape.STRING)
    @NumberFormat(pattern = "#,##0.0") // Usa la virgola come separatore decimale
    private BigDecimal temperatura;
    @Column(name = "tasso_inoculo") 2 usages
    private Float tassoInoculo;
    @Column(name = "associazione_sterile") 2 usages
    private String  associazioneSterile;
    @Column(name = "associazione_non_sterile") 2 usages
    private String  associazioneNonSterile;
    @Column(name = "matrice") 2 usages
    private String matrice;
    @Column(name = "tipo_matrice") 2 usages
    private String tipoMatrice;
    @Column(name = "disposizione_tipo") 2 usages
    private String disposizioneTipo;
    @Column(length = 10) 2 usages
    private String a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12;
    @Column(length = 10) 2 usages
    private String b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12;
    @Column(length = 10) 2 usages
    private String c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12;
    @Column(length = 10) 2 usages
    private String d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12;
    @Column(length = 10) 2 usages
    private String e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12;
    @Column(length = 10) 2 usages
    private String f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12;
    @Column(length = 10) 2 usages
    private String g1, g2, g3, g4, g5, g6, g7, g8, g9, g10, g11, g12;
    @Column(length = 10) 2 usages
    private String h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12;
```

Figura 18: Classe Organization standard

Considerata la necessità di gestire differenti tipologie di disposizioni predefinite dei ceppi, è stata prevista la colonna disposizione_tipo. Tale campo consente di evitare che

l'utente debba inserire manualmente, di volta in volta, i nomi dei ceppi all'interno della matrice. Al contrario, l'utente può selezionare una disposizione già definita, la quale viene recuperata attraverso una richiesta GET che restituisce i dati direttamente da questa tabella.
In questo modo si ottiene una maggiore efficienza e standardizzazione nella gestione delle disposizioni, riducendo il rischio di errori manuali e garantendo coerenza tra i dati inseriti.

5.4 ExperimentalConditions,NewExperimentalConditions, LastExperimentalConditions

Ogni nuovo esperimento condotto sarà registrato tramite l'interfaccia del progetto e salvato nel database.
A ciascun esperimento verrà associato un file Excel, rappresentante l'output generato dal macchinario utilizzato.
I macchinari sono tre, i dati relativi al primo macchinario sono memorizzati nelle tabelle experimental_conditions ed experimental_data mentre i dati relativi al secondo macchinario sono memorizzati nelle tabelle new_experimental_conditions ed experimental_data_new mentre quelli relativi al terzo macchinario in last_experimental_conditions e experimental_data_last.
L'entity ExperimentalConditions rappresenta l'astrazione delle condizioni sperimentali associate agli esperimenti generati dal primo macchinario. Essa è mappata alla tabella experimental_conditions presente nel database MySQL.
Questa classe svolge una duplice funzione:

- Lato applicativo: fornisce un modello a oggetti tipizzato, utilizzato per manipolare i dati relativi alle condizioni sperimentali in modo coerente e sicuro all'interno del codice Java.
- Lato persistente: definisce la corrispondenza(mapping) tra le proprietà dell'oggetto Java e le colonne della tabella experimental_conditions, consentendo a JPA di gestire in modo trasparente le operazioni di inserimento, lettura, aggiornamento e cancellazione (CRUD) sul database.

In questa entità è presente una relazione uno-a-molti con l'entità ExperimentalData.
L'entity NewExperimentalConditions rappresenta l'astrazione delle condizioni sperimentali associati agli esperimenti generati dal secondo macchinario. Essa è mappata alla tabella new_experimental_conditions presente nel database MySQL.
In questa entità è presente una relazione uno-a-molti con l'entità ExperimentalDataNew
L'entity LastExperimentalConditions rappresenta l'astrazione delle condizioni sperimentali associati agli esperimenti generati dal terzo macchinario.
Essa è mappata alla tabella last_experimental_conditions presente nel database MySQL
In questa entità è presente una relazione uno-a-molti con l'entità LastExperimentalData
Sono state create più tabelle siccome le condizioni sperimentali sono diverse per ciascun macchinario

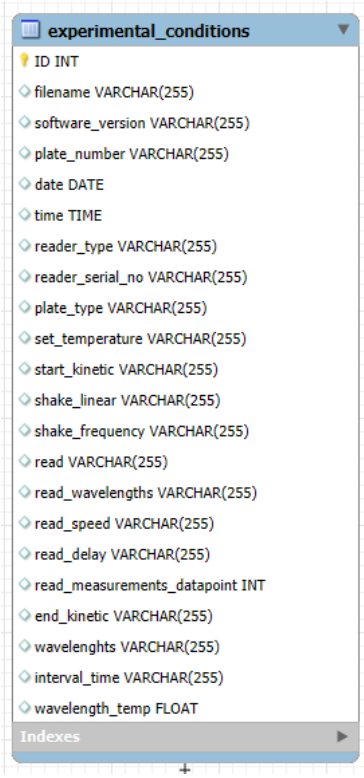

Figura 19:tabella experimental_conditions

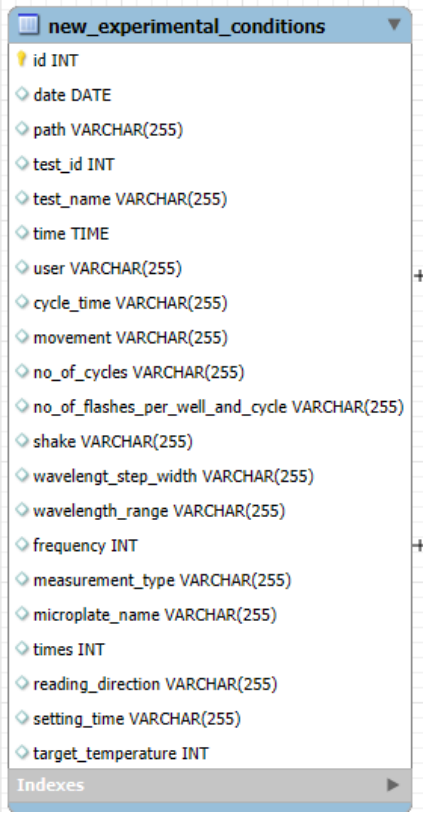

Figura 20: tabella new_experimental_conditions

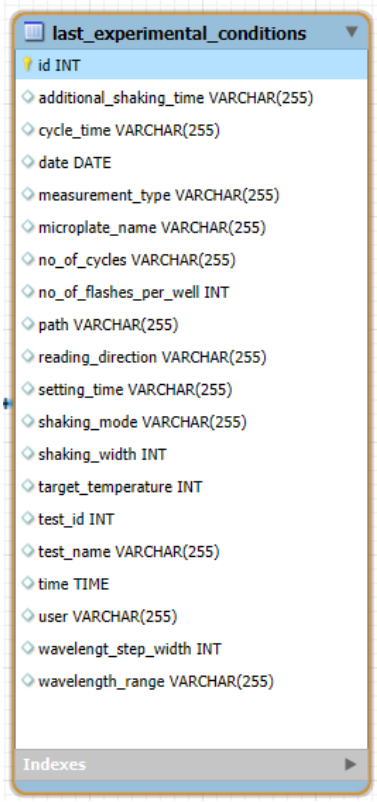

Figura 21:tabella last_experimental_conditions

5.5 ExperimentalData,ExperimentalDataNew, LastExperimentalData

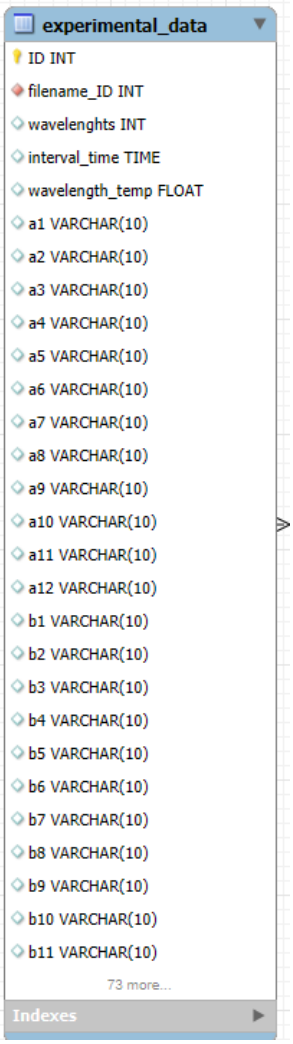

Figura 22:tabella experimental_data

Figura 23:tabella experimental_data_new

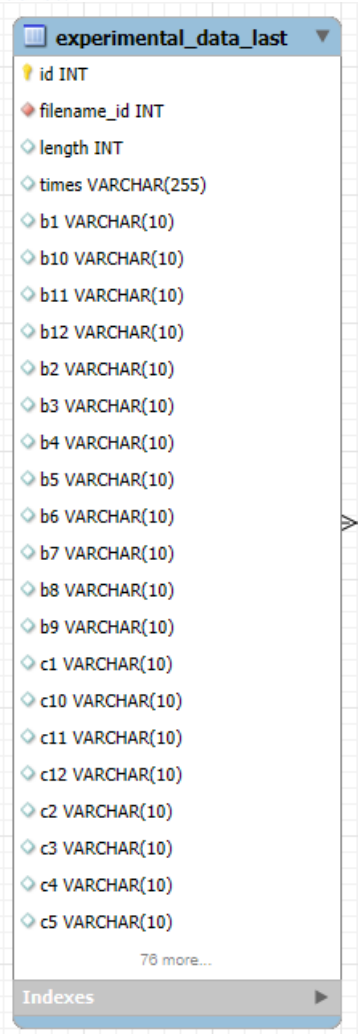

Figura 24:tabella experimental_data_last

In queste tre tabelle vengono memorizzati i dati generati dai macchinari

5.6 Matrix

```
@Entity  3 usages  tagarelli
@Table(name = "matrix")
@Data
@NoArgsConstructor
public class Matrix {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String type;
    private String nome;
```

Figura 25:Entità Matrix

L'entità Matrix rappresenta un modello di dati persistente, mappato su una tabella relazionale denominata matrix nel database. Essa è definita come classe Java annotata

con @Entity e @Table, in conformità alle specifiche JPA (Jakarta Persistence API), permettendo così l'integrazione diretta tra il modello a oggetti e la struttura relazionale sottostante.
La classe dispone di tre attributi principali:

- Id (Integer): Identificativo univoco della tabella, utilizzato come chiave primaria. È annotato con @Id e @ @GeneratedValue(strategy = GenerationType.IDENTITY), il che implica che il valore venga generato automaticamente dal database tramite una strategia di incremento automatico.
- type (String): campo testuale che definisce la tipologia della matrice, utile per categorizzare le matrici.
- nome (String): campo testuale che memorizza il nome associato alla matrice

5.7 User

```
@Entity  45 usages  tagarelli
@Table(name = "user")
@Data
@NoArgsConstructor
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String username;
    private String email;
    private String password;
    private String nome;
    private String cognome;
    @Column(name = "email_verified")
    private Boolean emailVerified = false;
    private int status =0;
    private String userType;
```

Figura 26:Entità User

L'entità User rappresenta il modello dati corrispondente alla tabella relazionale user nel database. È definita come classe Java annotata con @Entity e @Table, in conformità alle specifiche JPA (Jakarta Persistence API), consentendo la gestione della persistenza dei dati in modo diretto e integrato.
Gli attributi che compongono questa entità sono:

- **Id (Integer):** Identificativo dell'utente utilizzato come chiave primaria
- **username(String):** nome utente univoco per l'autenticazione all'interno del sistema
- **email(String):** indirizzo di posta elettronica associato all'utente
- **password(String):** credenziali di accesso dell'utente cifrate
- **nome(String)**
- **cognome(String)**

- **emailVerified(Boolean):** indica se l'indirizzo e-mail dell'utente è stato verificato. È mappato sul campo email_verified della tabella. Il valore predefinito è false
- **status(int)**:Stato dell'account, inizialmente impostato a zero. Quando l'account di un utente viene attivato dall'amministratore, il campo assume come valore numerico 1.
- **userType(String)**: Specifica la tipologia di utente, utile per la gestione dei ruoli e dei privilegi di accesso