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
D5.1 WP5 results deliverable

Autore	**Team WPS**
Controllato	**Alessandro Buscicchio**
Approvato	**Alessandro Buscicchio**
Versione	**1.4**
Data	**30 settembre 2025**

Sommario

1. Introduzione

Il presente progetto si inserisce nell'ambito della digitalizzazione e valorizzazione dei dati sperimentali in ambito microbiologico, con l'obiettivo di creare una base dati centralizzata, coerente e strutturata capace di supportare analisi avanzate e modelli predittivi di tipo Machine Learning (ML).

In particolare, il lavoro mira a integrare in un unico sistema informativo i risultati provenienti da differenti esperimenti di crescita microbica, archiviandoli in modo standardizzato all'interno di un database relazionale. Tale infrastruttura consente non solo la conservazione ordinata dei dati, ma anche la loro interrogazione e rielaborazione automatica per fini analitici e predittivi.

L'iniziativa nasce dall'esigenza di superare le limitazioni derivanti da archivi eterogenei o distribuiti, spesso difficili da gestire e da analizzare in modo sistematico. Attraverso la realizzazione di un processo di trasformazione del database MySQL in un database SQLite, il progetto rende possibile l'impiego diretto dei dati sperimentali in ambienti di calcolo dedicati al Machine Learning. SQLite, infatti, rappresenta una soluzione leggera, portabile e facilmente integrabile con i principali framework di analisi dati in Python, come Pandas, Scikit-learn

Per ottimizzare le operazioni estrazione e interrogazione del database, è stato integrato Ollama, un modello linguistico locale impiegato per la generazione automatica di query SQL. Grazie a questo strumento, è possibile formulare prompt in linguaggio naturale che vengono poi tradotti in query ottimizzate e conformi alla struttura del database

L'obiettivo finale del progetto è duplice. Da un lato, si intende garantire una gestione unificata e tracciabile dei dati biologici, assicurando integrità, coerenza e accessibilità. Dall'altro, si punta a sviluppare un modello predittivo basato su algoritmi di Machine Learning in grado di analizzare le curve di crescita batterica, stimare parametri significativi — come il coefficiente di determinazione (R^2), il tasso massimo di crescita (μ_{max}) e la fase di latenza (λ) — e suggerire le combinazioni di ceppi più promettenti per ottenere risultati ottimali. L'intero flusso di lavoro, descritto nel presente deliverable comprende:

- la raccolta e normalizzazione dei dati sperimentali provenienti dal database MySQL originale;
- la conversione automatica in formato SQLite, utile per la manipolazione efficiente dei dati e per l'addestramento dei modelli;
- la fase di analisi e predizione dei parametri di crescita microbica tramite algoritmi di Machine Learning;
- la validazione dei risultati e la loro potenziale applicazione nel supporto decisionale agli analisti di laboratorio.

2. Costruzione del database SQLite

Fasi di sviluppo:

1. **Connessione al database MySQL**

La funzione connect_db() stabilisce una connessione al database MySQL remoto utilizzando le credenziali fornite (host, utente, password e nome del database). Questo rappresenta il punto di accesso ai dati grezzi, che verranno successivamente recuperati ed elaborati.

2. **Caricamento dei dati**

La funzione load_strain_data_batch(cursor, experiment_ids) ha il compito di recuperare le informazioni relative ai ceppi microbici utilizzati in una serie di esperimenti, identificati dai rispettivi filename_ID

3. **Elaborazione dei singoli esperimenti**

La funzione process_single_experiment(cursor, exp_info, waves, strain_data) si occupa di processare un singolo esperimento, includendo:

- Recupero dei dati grezzi
- Pulizia dei dati
- Applicazione di un modello matematico (Baranyi-Roberts) per stimare i parametri di crescita microbiotica

Inoltre vengono annotate informazioni contestuali come temperatura, matrice e tasso di inoculo

4. **Esportazione dei risultati in formato SQL**

La funzione export_results(df, sql_file, table_schema, table_name='report') prende in ingresso i dati elaborati (organizzati in un dataframe Pandas) e li trascrive in un file .sql.
In questo passaggio vengono scritte le istruzioni CREATE TABLE (solo se la tabella non esiste già) e generate le istruzioni INSERT INTO per ciascun record. È prevista la gestione di valori mancanti, numerici e stringhe, garantendo la compatibilità sintattica con SQLite.

5. **Conversione in database**

La funzione sql_to_db(sql_file_path, db_file_path) completa il processo, importando lo script SQL appena generato all'interno di un database SQLite. L'uso di SQLite è motivato dalla sua leggerezza e dalla semplicità di integrazione in ambienti di machine learning, dove tipicamente è preferibile lavorare con database locali e autonomi, piuttosto che con server complessi come MySQL.

3. Architettura chatbot

Nel progetto è stato implementato un meccanismo di comunicazione tra due ambienti tecnologici distinti: Spring Boot, utilizzato per la gestione del backend applicativo principale in Java, e Flask, utilizzato per l'interfacciamento con un modello linguistico di grandi dimensioni (LLM), orchestrato tramite Ollama.
L'infrastruttura si articola in tre componenti principali:

- Frontend Angular – Interfaccia utente responsiva per l'interazione e l'invio di richieste;
- Backend Spring Boot (Java) – Strato applicativo intermedio che funge da bridge tra frontend e motore AI;

- Microservizio Flask (Python) – Modulo che comunica con il modello LLM tramite Ollama. In questo modulo viene implementato il modello ML per effettuare la predizione di alcuni parametri

3.1 Front-end chatbot Angular

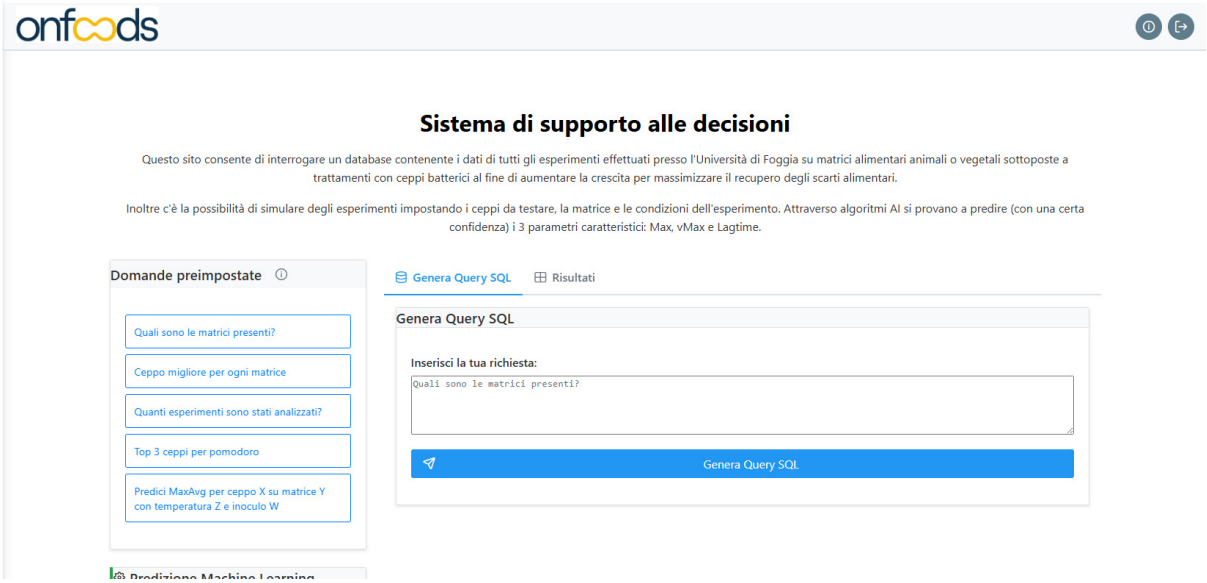


Figura 1:Interfaccia Chatbot

Il componente ChatbotComponent dell'applicazione Angular rappresenta l'interfaccia di interazione tra l'utente e il sistema di generazione ed esecuzione di query SQL basato su intelligenza artificiale.
Una delle sue funzioni principali è permettere all'utente di formulare richieste in linguaggio naturale, selezionare domande preimpostate o consultare la struttura delle tabelle presenti nel database, per poi ottenere e, se necessario, eseguire la query SQL generata dal sistema AI.
All'avvio del componente, tramite il metodo ngOnInit, vengono effettuate due operazioni preliminari fondamentali:

- **Recupero delle domande predefinite**: il sistema interroga il backend Java/Spring Boot per ottenere un elenco di possibili quesiti preimpostati che l'utente può selezionare senza dover digitare manualmente un prompt.
- **Recupero dell'elenco delle tabelle disponibili**: viene effettuata una chiamata alle API per ottenere i nomi delle tabelle presenti nel database, al fine di consentire all'utente di visualizzare eventuali schemi strutturali utili alla formulazione delle query.

L'utente può interagire con il sistema in due modalità principali:

- **Selezione di una domanda preimpostata**: in questo caso il componente imposta automaticamente il prompt e richiede al backend la query SQL già associata alla domanda selezionata.
- **Inserimento di un prompt personalizzato**: l'utente può scrivere una richiesta in linguaggio naturale, che il sistema trasmette al backend per ottenere la corrispondente query SQL generata dall'AI.

```
ngOnInit() {
  // Carica le domande preimpostate
  this.http.get<any>(`${environment.apiUrl}/api/ai/preset-questions`).subscribe(res => {
    this.presetQuestions = res.questions;
  });

  // Carica i nomi delle tabelle
  this.http.get<any>(`${environment.apiUrl}/api/ai/tables`).subscribe(res => {
    if (res.success) {
      this.tables = res.tables;
    }
  });
}
```

Figura 2:Metodo ngOnInit()

Il componente offre inoltre la possibilità di visualizzare lo schema di una tabella specifica. Se l'utente seleziona una tabella, viene inviata una richiesta al backend per ottenere la lista delle colonne e i relativi tipi di dato, così da permettere una consultazione diretta della struttura del dato prima della creazione della query.

```
loadTableSchema(table: string) {
  if (this.selectedTable === table) {
    this.selectedTable = '';
    this.currentSchema = null;
    return;
  }

  this.selectedTable = table;

  this.http.get(`${environment.apiUrl}/api/ai/table-schema/${table}`, { responseType: 'text' })
    .subscribe({
      next: (res: string) => {
        try {
          this.currentSchema = JSON.parse(res); // converto da stringa JSON a array di oggetti
        } catch (err) {
          console.error('Errore parsing schema della tabella', err);
          this.currentSchema = null;
        }
      },
      error: (err) => {
        console.error('Errore caricamento schema tabella', err);
        this.currentSchema = null;
      }
    });
}
```

Figura 3:Chiamata all'API per visualizzare lo schema della tabella specifica

Una volta ottenuta la query SQL, l'utente può decidere di eseguirla direttamente. In questo caso il componente invia la query al backend che ne cura l'esecuzione sul database, restituendo i risultati. Tali risultati vengono visualizzati sotto forma di tabella dinamica

```
generateQuery() {
  if (!this.prompt) return;
  this.loading = true;
  this.http.post<any>(`${environment.apiUrl}/api/ai/generate-sql`, { prompt: this.prompt })
    .subscribe(res => {
      this.generatedQuery = res.query;
      this.resultMessage = '';
      this.results = [];
      this.columns = [];
    }, null, () => this.loading = false);
}
```

Figura 4:Chiamata all'API per generare query

```
predict(form?: NgForm, onlySQL: boolean = false) {
  if (form && form.invalid) return;

  this.executing = true;

  const formData = new FormData();

  if (onlySQL) {
    // Solo SQL: invio la query generata
    formData.append('query', this.generatedQuery);
  } else {
    // Predizione ML: invio query ML_PREDICTION + parametri
    formData.append('query', 'ML_PREDICTION');
    this.campoVuoto.forEach(c => formData.append('ceppi[]', c));
    formData.append('matrice', this.selectedMatrice || '');
    formData.append('temperatura', form?.value.temperatura?.toString() || '');
    formData.append('tasso_inoculo', form?.value.inoculo?.toString() || '');
  }

  this.http.post<any>(`${environment.apiUrl}/api/ai/execute-query`, formData)
    .subscribe({
      next: (res) => {
        if (res.success) {
          this.results = res.results;
          this.columns = this.results.length > 0 ? Object.keys(this.results[0]) : [];
          this.resultSuccess = true;
          this.resultMessage = res.message || 'Operazione completata';
          this.activeTabIndex = 1;
          this.isMLPrediction = !onlySQL; // true se predizione ML, false se solo SQL

          // Calcolo dei migliori ceppi solo se ML
          if (!onlySQL && this.results?.length > 0) {
            this.bestMax = [...this.results].sort((a, b) => b.Max - a.Max)[0];
            this.bestVmax = [...this.results].sort((a, b) => b.vMax - a.vMax)[0];
            this.bestLag = [...this.results].sort((a, b) => a.Lagtime - b.Lagtime)[0];
          }
        } else {
          this.resultSuccess = false;
          this.resultMessage = res.message || 'Errore nell\'esecuzione';
          this.isMLPrediction = false;
        }
      },
```

Figura 5:Chiamata all'API per eseguire la query

La funzione predict(form?: NgForm, onlySQL: boolean = false) ha il compito di gestire l'invio di richieste al backend per due diverse finalità:

- eseguire semplicemente una query SQL pre-generata;
- avviare un processo di predizione basato sui modelli di Machine Learning

Il comportamento della funzione è determinato dal parametro onlySQL: se impostato a true, la funzione invia al server solo la query SQL già costruita; se invece è false, viene

eseguita la procedura di predizione automatica tramite l'apposita query ML_PREDICTION con i relativi parametri. Il flusso è il seguente:

- **Validazione del form**: Se è stato passato un form come argomento e questo risulta invalido, la funzione si interrompe immediatamente, evitando l'invio di richieste incomplete o errate.
- **Preparazione della richiesta**: Viene creato un oggetto FormData che conterrà i parametri da inviare al backend. Nel caso onlySQL = true, viene aggiunto un solo campo query, contenente la query SQL già generata. Mentre nel cado di predizione(onlySQL = false), al formData vengono invece aggiunti: la stringa fissa 'ML_PREDICTION' come tipo di query, l'elenco dei ceppi selezionati, la matrice scelta e i valori di temperatura e tasso di inoculo ottenuti dal form
- **Invio al backend**: La richiesta viene inviata al server tramite una chiamata HTTP POST verso l'endpoint api/ai/execute-query, utilizzando l'URL base definito in environment.apiUrl
- **Gestione della risposta**: Se il backend segnala success=true, i risultati vengono salvati in this.results. Contestualmente vengono aggiornate le colonne della tabella, i messaggi e gli stati di conferma e l'indicatore isMLPrediction, che distingue se la richiesta è stata una predizione ML o una semplice query SQL. Inoltre, se si tratta di una predizione ML, la funzione calcola e salva i migliori ceppi in base a tre metriche: bestMax (massimo valore di crescita), bestVmax (massima velocità di crescita) e bestLag (tempo di latenza più breve). Se la risposta contiene success = false, vengono aggiornati i flag interni in modo da segnalare l'errore all'utente, riportando anche il messaggio di errore inviato dal backend mentre in caso di errore di rete o problemi di connessione con il server, viene mostrato un messaggio generico di errore

All'interno dell'interfaccia utente c'è anche un'altra sezione dedicata alla funzionalità di predizione tramite modelli di Machine Learning. La struttura principale è incapsulata in un componente <p-card>, che fornisce un contenitore grafico con un'intestazione e un corpo.
Nell'intestazione viene visualizzato il titolo "Predizione Machine Learning", corredato da un'icona e da un pulsante informativo che richiama una finestra di dialogo tramite la funzione showInfoDialog2().
Il cuore del codice è un form Angular reattivo(mlform), associato alla funzione predict(mlform) che viene richiamata al momento dell'invio. All'interno del form l'utente può specificare i parametri necessari alla predizione:

- Ceppi: selezionabili tramite un componente <p-multiSelect>, con supporto a filtri e possibilità di selezione multipla;
- Matrice: selezionabile da menu a tendina(<p-dropdown>);
- Temperatura e tasso di inoculo: inseriti come valori numerici attraverso campi <input> con validazione obbligatoria.

Sotto al modulo di input viene fornita una legenda esplicativa che chiarisce il significato dei tre parametri predetti dal modello: Max, LagTime, vMax
A completamento del componente, è presente un link che permette di mostrare o nascondere le combinazioni ceppo/matrice disponibili, richiamando la funzione loadCombinations().
Durante il caricamento viene mostrato un indicatore di avanzamento (<p-progressSpinner>), mentre i dati vengono successivamente visualizzati in una tabella

dinamica (<p-table>), dotata di impaginazione e adattamento responsivo. La tabella elenca per ogni combinazione:

- Ceppo
- Matrice
- Numero di esperimenti associati

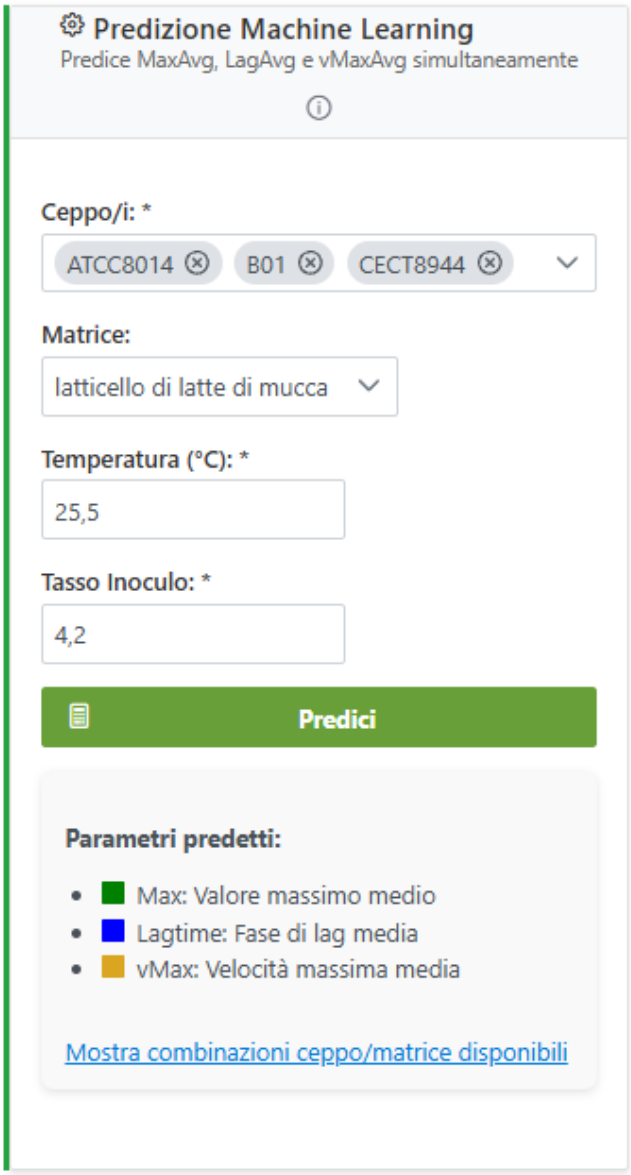


Figura 6:Inserimento dati per la predizione

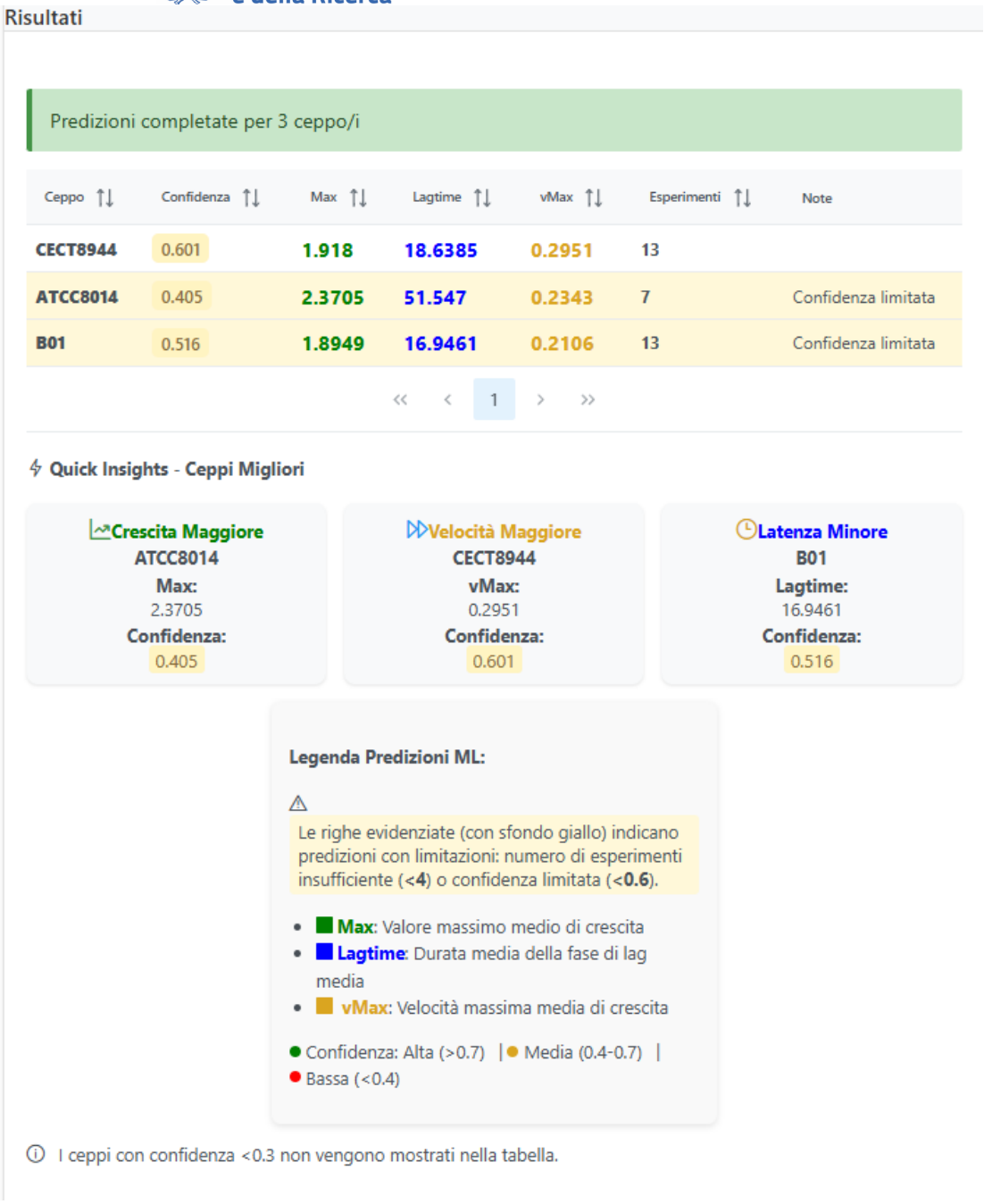


Figura 7:Risultati predizione

I risultati della predizione di Machine learning sono visualizzati in un contenitore principale che è un <p-card> di PrimeNG. All'interno del componente i risultati vengono visualizzati tramite un <p-table> di PrimeNG che supporta paginazione, ordinamento multiplo e layout responsivo. Le colonne della tabella riportano:

- Ceppo analizzato
- La confidenza della predizione
- I valori predetti: Max, Lagtime e vMax
- Eventuali note

Ogni riga può essere evidenziata con uno stile particolare per indicare predizioni con dati limitati o confidenza ridotta. La confidenza viene visualizzata tramite badge colorati, che

cambiano colore in funzione del valore (alta, media o bassa), fornendo un immediato riscontro visivo all'utente. Sotto la tabella, viene mostrata una sezione denominata "Quick Insights - Ceppi Migliori", che sintetizza i ceppi con prestazioni ottimali secondo tre metriche. Per ciascuna metrica viene evidenziato il ceppo corrispondente, il valore della predizione e la confidenza, con un layout grafico a card e icone illustrative. Questo permette all'utente di identificare rapidamente i ceppi più performanti.
La sezione include anche una legenda dettagliata, che spiega:

- il significato delle colonne Max, Lagtime e vMax
- il significato dei colori dei badge di confidenza (alta, media, bassa),
- la modalità di evidenziazione delle righe con predizioni limitate (numero di esperimenti insufficiente o confidenza ridotta).

Viene inoltre specificato che i ceppi con confidenza inferiore a 0.3 non vengono mostrati nella tabella, per evitare di visualizzare predizioni altamente incerte.

3.2 Backend in Flask

Nel contesto del progetto, l'obiettivo principale consisteva nello sviluppare un modello in grado di effettuare predizioni quantitative sui parametri biologici di interesse (MaxAvg, LagAvg, vMaxAvg) a partire da variabili sperimentali note (temperatura, tasso di inoculo, ceppo e matrice).
Inizialmente si pensava di utilizzare algoritmi non supervisionati ma per alcune motivazioni metodologiche è stato scelto un approccio di apprendimento supervisionato. In primo luogo, era disponibile un database di dati etichettati, contenente osservazioni storiche in cui a ogni combinazione di variabili sperimentali corrispondevano i valori noti dei parametri biologici. Ciò ha reso possibile l'addestramento di modelli capaci di apprendere una funzione che mappa gli input sugli output osservati.
In secondo luogo, lo scopo del progetto era chiaramente predittivo, ovvero la stima numerica diretta dei parametri biologici, e non l'esplorazione dei dati o l'individuazione di pattern nascosti, tipica dell'apprendimento non supervisionato.
Un ulteriore vantaggio dell'approccio supervisionato è la maggiore interpretabilità: mentre le tecniche non supervisionate (come il clustering o la riduzione dimensionale) avrebbero potuto evidenziare gruppi di ceppi o matrici con comportamenti simili, non sarebbero state in grado di fornire stime puntuali dei parametri biologici di interesse.
Infine, la possibilità di ottenere predizioni quantitative per nuove condizioni sperimentali rispondeva in modo più diretto alle esigenze applicative del progetto, rendendo l'apprendimento supervisionato la scelta più naturale ed efficace
Tra i diversi algoritmi di apprendimento supervisionato, nel progetto è stato impiegato il modello RandomForest, una tecnica basata su una collezione di alberi decisionali. L'idea alla base di questo approccio è quella di combinare la predizione di molteplici modelli deboli (gli alberi) per ottenere un modello complessivo più robusto
Per l'implementazione del backend vengono importati moduli essenziali per:

- Manipolazione dei dati (pandas, numpy)
- Costruzione di modelli predittivi (scikit-learn)
- Gestione del database (sqlite3)
- Serializzazione di oggetti(pickle)

In particolare sklearn è utilizzato per:

- **RandomForestRegressor**: modello predittivo basato su alberi decisionali

- **StandardScaler**: standardizza i dati
- **train_test_split**: per dividere i dati in set di addestramento e test
- **r2_score e mean_absolute_error**: metriche per valutare le performance del modello

```
class DatabaseMLPredictor:  1 usage
    def __init__(self, db_path):
        self.db_path = db_path
        self.models = {}
        self.scalers = {}
        self.model_metadata = {}
```

Figura 8:classe DatabaseMLPredictor

Questa classe serve per:

- accedere ai dati
- controllare la disponibilità
- addestrare modelli per ogni combinazione ceppo-matrice
- salvare i dati

Le principali funzioni implementate sono train_model e predict

```
def train_model(self, ceppo, matrice):  1 usage
    """
    Addestra modelli migliorati per un ceppo e matrice specifici.
    """
    df = self.get_training_data(ceppo, matrice)
    n_experiments = self.count_experiments(ceppo, matrice)
    targets = ['MaxAvg', 'LagAvg', 'vMaxAvg']
    model_key = f"{ceppo}_{matrice}"

    if len(df) == 0:
        return None, "Nessun campione disponibile per addestrare o stimare."

    # Pulizia dati (opzionale)
    #df_clean = self.clean_data(df)
    df_clean = df
    if len(df_clean) == 0:
        return None, "Tutti i campioni sono stati rimossi come outlier."

    # Fallback per dati molto scarsi
    if len(df_clean) < 3:
        fallback_predictions = {target: df_clean[target].mean() for target in targets}

        class FallbackModel:
            def __init__(self, value, std=None):
                self.value = value
                self.std = std or max(0.1 * abs(value), 0.01)

            def predict(self, X):
                return np.array([self.value] * len(X))

            def get_uncertainty(self, X):
                return np.array([self.std] * len(X))
```

Figura 9:Funzione train_model

La funzione train_model (self, ceppo, matrice) ha l'obiettivo di addestrare modelli predittivi per stimare tre variabili target – MaxAvg, LagAvg e vMaxAvg – a partire da un dataset sperimentale relativo ad uno specifico ceppo e ad una specifica matrice. L'approccio adottato integra strategie di fallback statistico per dataset di dimensioni ridotte e di machine learning supervisionato (con Random Forest) per dataset più consistenti. Le fasi principali del processo sono:

1. **Recupero e preparazione dei dati**

La funzione inizia ottenendo i dati di addestramento tramite il metodo get_training_data(ceppo, matrice) e determina il numero di esperimenti disponibili attraverso count_experiments(ceppo, matrice).
Se il dataset risulta vuoto, l'addestramento viene immediatamente interrotto e la funzione restituisce un messaggio che segnala l'impossibilità di procedere.
Successivamente, può essere prevista un'eventuale fase di pulizia dei dati, volta a eliminare outlier o anomalie che potrebbero compromettere la qualità del modello.

2. **Gestione dei dati scarsi(Fallback)**

Nel caso in cui il numero di campioni disponibili sia inferiore a tre, non viene eseguito un addestramento tradizionale.
In questa situazione, la funzione adotta una strategia di fallback basata sulla media delle variabili target.
A tal fine, viene definita una classe interna denominata FallbackModel, la quale fornisce una predizione costante corrispondente al valore medio stimato, associata a un'incertezza proporzionale.
Per ciascuna delle tre variabili target viene istanziato un modello di tipo fallback, a cui è attribuito un livello di confidenza basso (non superiore a 0,4), per riflettere la scarsa affidabilità delle previsioni ottenute da un numero di dati così limitato.
I modelli, insieme alle relative informazioni di supporto, vengono salvati e resi disponibili per successive attività di predizione.

3. **Addestramento con machine learning**

Qualora il dataset disponga di almeno tre campioni, l'addestramento viene condotto utilizzando un Random Forest Regressor fornito dalla libreria scikit-learn.
Le variabili esplicative impiegate sono Temperatura e Tasso_di_inoculo, che vengono preprocessate tramite standardizzazione mediante StandardScaler.

4. **Scelta adattiva della strategia di validazione**
   - Con meno di 8 campioni si utilizza una validazione Leave-One-Out Cross Validation (LOOCV).
   - Con dataset più ampi viene applicata una validazione K-Fold (fino a 5 fold, in base alla disponibilità di dati).

5. **Configurazione adattiva del modello**

La complessità della Random Forest è definita in modo adattivo in base al numero di osservazioni:

- **< 10 campioni:** modello semplificato, con pochi alberi e profondità limitata.
- **10–25 campioni:** configurazione intermedia, con un numero maggiore di alberi e profondità superiore.
- **> 25 campioni:** configurazione completa, caratterizzata da un numero elevato di alberi e parametri più flessibili.

6. **Valutazione delle prestazioni**

Per ciascun target vengono calcolate le seguenti metriche:

- R^2 medio e deviazione standard, per stimare la capacità predittiva del modello;
- Errore assoluto medio (MAE), per valutare l'accuratezza delle previsioni.

Quando disponibile, viene inoltre considerato lo score Out-of-Bag (OOB), che fornisce una misura aggiuntiva di generalizzazione del modello, indipendente dalla cross-validation.

7. Stima della confidenza e salvataggio dei modelli

Al termine dell'addestramento, i modelli ottenuti, insieme agli scaler di preprocessing e alle relative metainformazioni, vengono memorizzati all'interno delle strutture dati della classe (self.models, self.scalers, self.model_metadata).

È inoltre previsto il salvataggio persistente dei modelli tramite il metodo save_model, così da garantirne la disponibilità per utilizzi futuri.

8. Output finale

La funzione restituisce i modelli addestrati e un messaggio riassuntivo contenente:

- Il livello medio di confidenza stimato;
- Le metriche di validazione (R^2 medio, deviazione standard, MAE);
- l tipo di strategia di validazione adottata.

```python
def predict(self, ceppo, matrice, temperatura, tasso_inoculo):  5 usages (3 dynamic)
    """Effettua predizioni con stima dell'incertezza migliorata."""
    model_key = f"{ceppo}_{matrice}"

    if model_key not in self.models:
        if not self.load_model(model_key):
            models, message = self.train_model(ceppo, matrice)
            if models is None:
                return None, message

    try:
        predictions = {}
        uncertainties = {}
        targets = ['MaxAvg', 'LagAvg', 'vMaxAvg']

        X_new = np.array([[temperatura, tasso_inoculo]])
        model_info = self.model_metadata[model_key]

        for target in targets:
            model = self.models[model_key][target]

            if hasattr(model, 'get_uncertainty'):
                # Modello fallback
                pred = model.predict(X_new)[0]
                unc = model.get_uncertainty(X_new)[0]
                predictions[target] = float(pred)
                uncertainties[target] = float(unc)
            else:
                # Modello Random Forest
                X_new_scaled = self.scalers[model_key].transform(X_new)
                pred = model.predict(X_new_scaled)[0]
                predictions[target] = float(pred)

                # Stima incertezza dalle predizioni degli alberi individuali
```

Figura 10:funzione predict parte 1

```
        # Stima incertezza dalle predizioni degli alberi individuali
        if hasattr(model, 'estimators_') and len(model.estimators_) > 1:
            tree_predictions = [tree.predict(X_new_scaled)[0] for tree in model.estimators_]
            tree_std = np.std(tree_predictions)

            # Combina incertezza degli alberi con confidenza del modello
            model_confidence = model_info[target].get('confidence', 0.5)
            uncertainty = tree_std + (abs(pred) * (1 - model_confidence) * 0.2)
            uncertainties[target] = float(uncertainty)
        else:
            # Fallback per incertezza
            model_confidence = model_info[target].get('confidence', 0.5)
            uncertainties[target] = float(abs(pred) * (1 - model_confidence) * 0.3)

    # Confidenza complessiva
    avg_confidence = np.mean([model_info[target].get('confidence', 0.5) for target in targets])

    # Numero di esperimenti dal metadata
    n_experiments = model_info[targets[0]].get('n_experiments', 0)

    # Genera note basate sui criteri richiesti
    note = None
    if n_experiments < 4:
        note = "Numero di esperimenti ridotto"
    elif avg_confidence < 0.6:
        note = "Confidenza limitata"

    return {
        'predictions': predictions,
        'uncertainties': uncertainties,
        'confidence': float(avg_confidence),
        'n_experiments': n_experiments,
        'model_info': model_info,
        'note': note
    }, "Predizioni completate con successo"
```

Figura 11:funzione predict parte 2

Il meccanismo di funzionamento della funzione *predict* può essere ricondotto a tre fasi distinte e complementari:

1. **Caricamento o addestramento del modello**

La funzione verifica se esiste già un modello addestrato per la combinazione ceppo_matrice. In caso contrario, prova a caricarlo da memoria; se non disponibile, procede ad addestrarne uno nuovo tramite la funzione train_model. Se nessun modello può essere recuperato o creato, la funzione si interrompe restituendo un messaggio di errore.

2. **Predizione per ciascun target**

Per i tre target considerati, il codice recupera il modello corrispondente e genera la previsione:

- Se si tratta di un modello di fallback, la previsione e l'incertezza associata vengono calcolate direttamente attraverso metodi dedicati.
- Se invece il modello è una Random Forest, il nuovo campione viene prima normalizzato tramite lo scaler salvato e quindi passato al modello. In questo caso, l'incertezza viene stimata in modo empirico a partire dalla variabilità delle predizioni dei singoli alberi e dai valori di confidenza registrati nei metadati.

3. **Aggregazione e restituzione dei risultati**

Una volta completate le predizioni, la funzione calcola una confidenza media complessiva e recupera dai metadati il numero di esperimenti originari. Inoltre, aggiunge una nota diagnostica se il numero di esperimenti è molto basso o se la confidenza stimata è limitata. Infine, vengono restituiti: le predizioni per ciascun target, le stime di incertezza, la confidenza aggregata, i metadati del modello

3.3 Backend in Java

Il backend in Java agisce come gateway applicativo tra l'interfaccia web e il motore AI. I suoi compiti principali sono:

- Ricevere richieste REST dal frontend
- Inoltrarle al microservizio Python tramite chiamate http e gestisce la comunicazione tramite il componente RestTemplate;
- Gestire la formattazione dei dati e centralizzare il flusso delle API;
- Offrire un livello aggiuntivo di sicurezza e controllo dei dati.

Questa separazione permette di isolare la logica di presentazione da quella dell'intelligenza artificiale, rendendo il sistema più modulare e mantenibile
Il servizio è definito come un componente Spring(@Service), rendendolo iniettabile tramite dependency injection e facilmente accessibile da altri layer dell'applicazione
Viene utilizzato un oggetto RestTemplate, fornito da Spring, per effettuare comunicazioni HTTP verso un servizio Flask esposto sulla rete locale, all'indirizzo http://192.168.1.241:5000. Tutte le chiamate ai metodi AI sono incapsulate in metodi pubblici che rappresentano operazioni ben definite del sistema.

```java
@Service  2 usages  tagarelli
public class AIService {

    @Autowired  12 usages
    private RestTemplate restTemplate;

    //private final String flaskBaseUrl = "http://192.168.1.241:5000";
    private final String flaskBaseUrl = "http://localhost:5000";  12 usages


    public String generateSQL(String prompt, Integer presetId) {  1 usage  tagarelli
        String url = flaskBaseUrl + "/generate_query";

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);

        MultiValueMap<String, String> body = new LinkedMultiValueMap<>();
        if (prompt != null) body.add("prompt", prompt);
        if (presetId != null) body.add("preset_id", String.valueOf(presetId));

        HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(body, headers);

        try {
            ResponseEntity<String> response = restTemplate.postForEntity(url, request, String.class);
            return response.getBody();
        } catch (Exception e) {
            e.printStackTrace();
            return "{\"success\": false, \"message\": \"Errore durante la richiesta\"}";
        }
    }
```

Figura 12:Metodo generate SQL

Il servizio utilizza un'istanza di RestTemplate per effettuare richieste http verso l'API Flask. RestTemplate è un client http fornito da Spring che permette di eseguire GET e POST, gestire facilmente intestazioni, parametri e corpo delle richieste e ricevere le risposte mappandole in vari formati.
Il metodo generateSQL ha la funzione di interagire con un servizio esterno, sviluppato in Flask, per generare dinamicamente una query SQL a partire da determinati parametri.

```java
public String executeMLPrediction(String[] ceppi, String matrice, String temperatura, String inoculo) {
    String url = flaskBaseUrl + "/execute_query";

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);

    MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
    map.add("query", "ML_PREDICTION");
    for (String c : ceppi) {
        map.add("ceppi[]", c);
    }
    map.add("matrice", matrice);
    map.add("temperatura", temperatura);
    map.add("tasso_inoculo", inoculo);

    HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map, headers);
    ResponseEntity<String> response = restTemplate.postForEntity(url, request, String.class);

    return response.getBody();
}
```

Figura 13:Metodo executeMLPrediction

Il metodo executeMLPrediction ha il compito di inviare una richiesta HTTP POST a un servizio esterno per eseguire una predizione basata su Machine Learning.
Per prima cosa, viene costruito l'indirizzo dell'endpoint, concatenando l'URL di base (flaskBaseUrl) con il path /execute_query. Il corpo della richiesta viene costruito attraverso una LinkedMultiValueMap, che rappresenta una mappa in cui ciascuna chiave può essere associata a più valori. Nel corpo vengono aggiunti:

- il parametro fisso query con valore ML_PREDICTION, che indica al servizio remoto quale tipo di operazione eseguire;
- tutti i valori contenuti nell'array ceppi, che rappresentano i ceppi microbiologici da utilizzare nella predizione;
- i parametri matrice, temperatura e tasso_inoculo, che rappresentano rispettivamente la matrice di coltura, la temperatura di incubazione e il tasso di inoculo da considerare nell'analisi.

Una volta predisposti i parametri, viene creato un oggetto HttpEntity, che incapsula sia le intestazioni che il corpo della richiesta, e viene inviato al server Flask tramite il metodo postForEntity di RestTemplate.

```java
public String executeSQL(String query) {  1 usage  tagarelli
    String url = flaskBaseUrl + "/execute_query";

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);

    MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
    map.add("query", query);

    HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map, headers);

    ResponseEntity<String> response = restTemplate.postForEntity(url, request, String.class);

    return response.getBody();
}
```

Figura 14:metodo executeSQL

All'interno di questo metodo viene impiegata una LinkedMultiValueMap per la costruzione del payload, nel quale viene inserita la query SQL passata come parametro al metodo.
Successivamente, il corpo della richiesta e le relative intestazioni HTTP vengono incapsulati in un oggetto HttpEntity, che viene trasmesso mediante il metodo postForEntity della classe RestTemplate.
La risposta ottenuta, di tipo ResponseEntity<String>, viene quindi elaborata estraendone esclusivamente il corpo tramite response.getBody(), il quale viene infine restituito al chiamante.
Sebbene questo metodo condivida con executeMLPrediction l'utilizzo di RestTemplate per l'invio di una richiesta HTTP POST verso il medesimo endpoint (/execute_query), i due presentano finalità e logiche operative differenti.
In particolare, executeSQL ha come scopo l'invio di un singolo parametro – la query SQL completa da eseguire – mentre executeMLPrediction costruisce in modo dinamico un insieme più articolato di parametri.
Quest'ultimo, infatti, include un valore fisso (query=ML_PREDICTION) e aggiunge informazioni specifiche quali il ceppo microbiologico, la matrice di coltura, la temperatura e il tasso di inoculo, trasformando tali dati in un formato compatibile con il servizio Flask che riceve e gestisce la richiesta.

```java
public List<String> getMLStrains() {  1 usage  tagarelli
    String url = flaskBaseUrl + "/ml_strains";
    Map<String, Object> response = restTemplate.getForObject(url, Map.class);

    if (response == null || !response.containsKey("strains")) {
        throw new RuntimeException("Flask non ha restituito la chiave 'strains'");
    }

    return (List<String>) ((List<?>) response.get("strains"));
}

public List<Map<String, Object>> getMatrices() {  1 usage  tagarelli
    String url = flaskBaseUrl + "/matrices";
    System.out.println("Chiamata a Flask: " + url);

    Map<String, Object> response = restTemplate.getForObject(url, Map.class);
    System.out.println("Risposta da Flask: " + response);

    if (response == null || !response.containsKey("matrices")) {
        throw new RuntimeException("Flask non ha restituito la chiave 'matrices'");
    }

    return (List<Map<String, Object>>) response.get("matrices");
}
```

Figura 15:Metodi per recuperare matrici e ceppi batterici

Il metodo getMLStrains ha il compito di ottenere, tramite una richiesta HTTP GET, l'elenco dei ceppi microbiologici disponibili per le predizioni di Machine Learning fornito dal servizio Flask. La logica di esecuzione si sviluppa in tre fasi principali:

- Costruzione dell'endpoint: viene generato l'URL di destinazione concatenando la variabile flaskBaseUrl con il percorso /ml_strains.
- Invio della richiesta: attraverso restTemplate.getForObject, il metodo invia una richiesta GET all'endpoint e riceve una risposta in formato Map, contenente le coppie chiave-valore del JSON restituito dal servizio.
- Validazione e restituzione: se la risposta è nulla o priva della chiave strains, viene lanciata una RuntimeException; in caso contrario, il valore associato alla chiave viene estratto, convertito in una lista di stringhe e restituito al chiamante.

Il metodo getMatrices adotta una logica analoga, ma è finalizzato al recupero delle matrici di coltura disponibili dal medesimo servizio Flask.
L'endpoint viene costruito concatenando flaskBaseUrl con /matrices, e la richiesta GET viene eseguita tramite restTemplate.getForObject, che restituisce una mappa con la risposta JSON.
Qualora la risposta sia nulla o non contenga la chiave matrices, viene sollevata una RuntimeException; in caso di esito positivo, il metodo restituisce il contenuto della chiave convertito in una lista di mappe (List<Map<String, Object>>), così da mantenere le eventuali proprietà associate a ciascuna matrice.

Oltre ai metodi precedentemente descritti, sono state effettuate chiamate ad ulteriori endpoint del servizio Flask, in particolare ai metodi getTablesFromFlask e getTableSchema.
Il primo ha la funzione di recuperare l'elenco delle tabelle disponibili presso il backend Flask, consentendo di identificare le risorse dati accessibili dal sistema.
Il secondo, invece, è destinato a ottenere lo schema di una specifica tabella, a partire dal nome di quest'ultima fornito come parametro, restituendo così la struttura dei campi e delle relative tipologie gestite dal servizio.

4. Architettura piattaforma DSS

Dopo aver descritto nel capitolo precedente l'architettura del chatbot, in questo capitolo si analizzerà la piattaforma DSS che ne consente l'integrazione. La struttura è la seguente

- Frontend sviluppato in Angular con l'ausilio di PrimeNg
- Backend sviluppato in Java utilizzando il framework Spring

4.1 Frontend

Dopo aver effettuato l'autenticazione per accedere alla piattaforma, comparirà una dashboard in cui saranno mostrati tutti gli esperimenti effetttuati

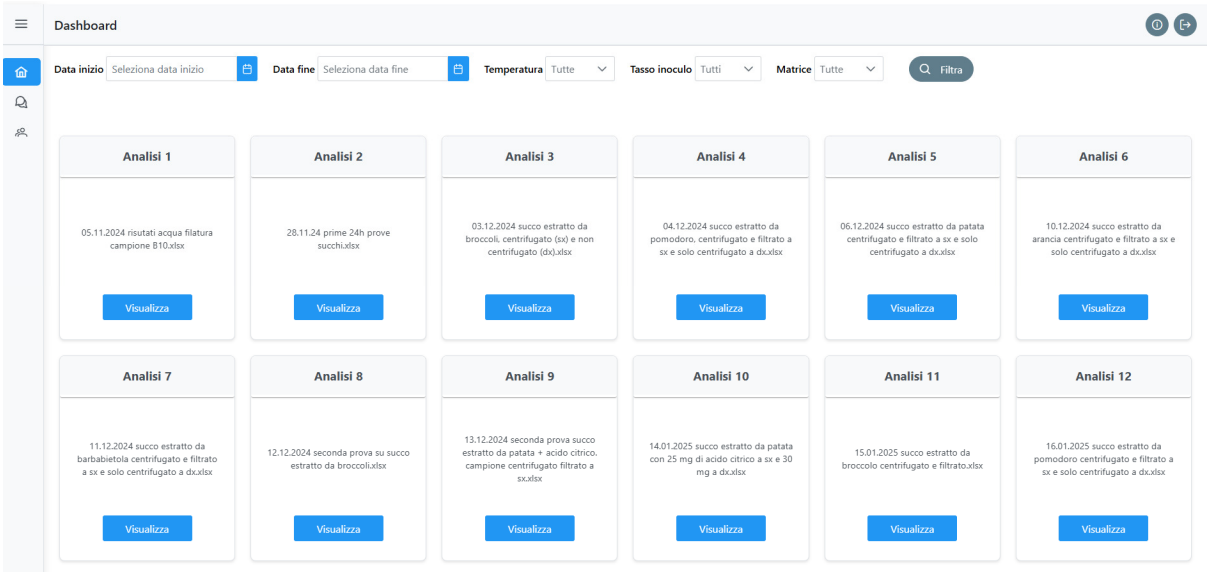


Figura 16:Dashboard

La dashboard dell'interfaccia utente è organizzata secondo una struttura chiara e modulare, pensata per garantire un'esperienza utente intuitiva ed efficiente. Sul lato sinistro è presente una sidebar di navigazione verticale, che consente l'accesso rapido alle principali sezioni dell'applicazione. Nella parte superiore della vista si trova una navbar orizzontale, la quale mostra il titolo della pagina corrente e include una serie di icone funzionali, tra cui:

- un'icona per la visualizzazione delle informazioni dell'utente autenticato;
- un'icona per l'esecuzione del logout.
- Al di sotto della navbar è collocata l'area principale del contenuto, la quale è suddivisa in due sezioni distinte:

1. Un modulo di input, che consente all'utente di filtrare i dati visualizzati in base a un intervallo temporale selezionabile (data di inizio e data di fine), poi ci sono tre moduli di select inerenti alla temperatura, tasso di inoculo e tipo di matrice e l'utente può effettuare la ricerca degli esperimenti filtrando in base anche a questi parametri;
2. Un contenitore dei risultati, che mostra in forma di elenco o schede i dati filtrati, ciascuno dei quali può essere visualizzato in dettaglio mediante un'apposita azione.

I risultati sono mostrati utilizzando il componente p-card di PrimeNg in una struttura iterativa. Ogni p-card include:

- Intestazione con titolo dinamico "Analisi N", dove N è l'indice incrementato.
- Contenuto con il nome del file (file.filename).
- Pulsante "Visualizza" che reindirizza alla pagina dei dettagli

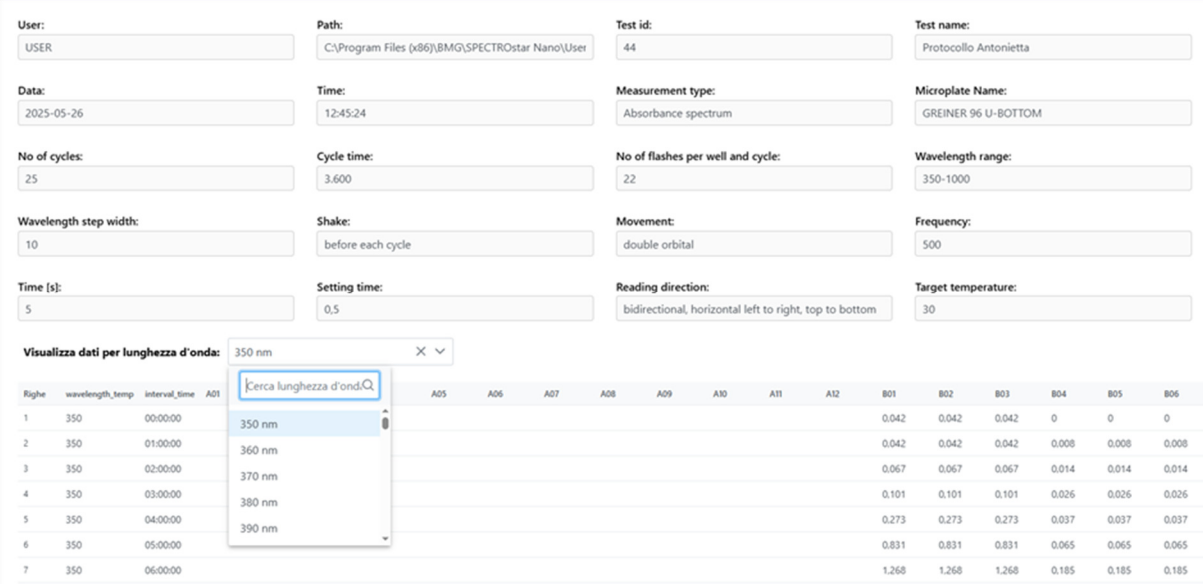


Figura 17:Dettagli esperimento

La pagina dettagli presenta tre sezioni: i parametri dell'esperimento e i dati restituiti dal macchinario con cui è stata effettuata l'analisi dell'esperimento e il grafico relativo alla concentrazione batterica dove è mostrato il calcolo

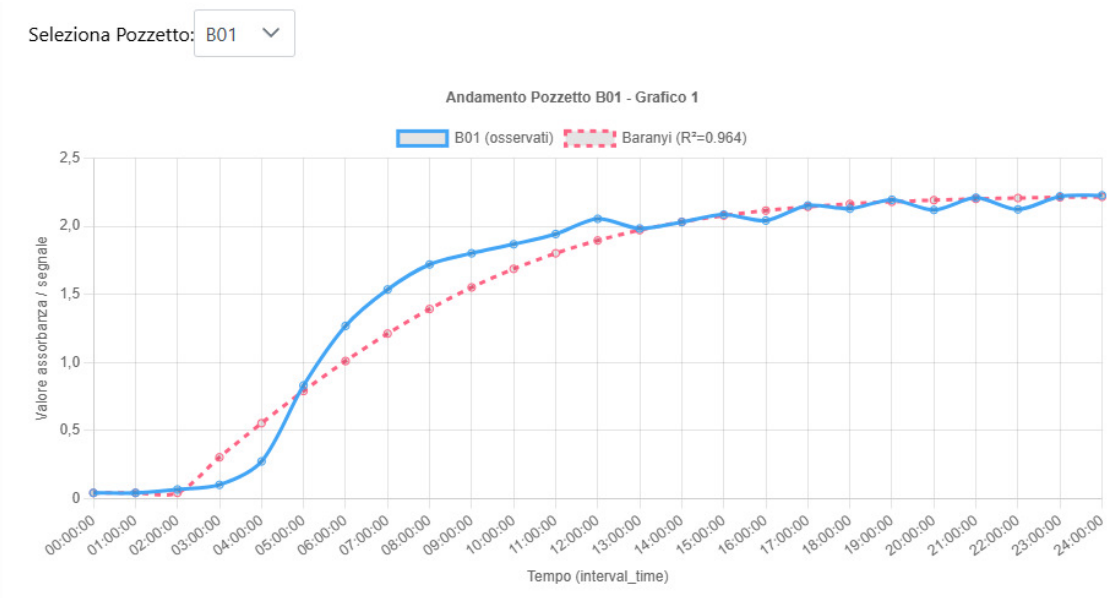


Figura 18:concentrazione batterica

Il grafico riportato in figura mostra la concentrazione batterica e il cacolo del coefficiente di determinazione R^2 per confrontare i dati osservati con i valori teorici stimati
Un valore di R^2 prossimo a 1 indica che il modello spiega gran parte della variabilità osservata, mentre valori vicini ad 1 denotano una scarsa capacità predittiva

4.2 Struttura backend in Java

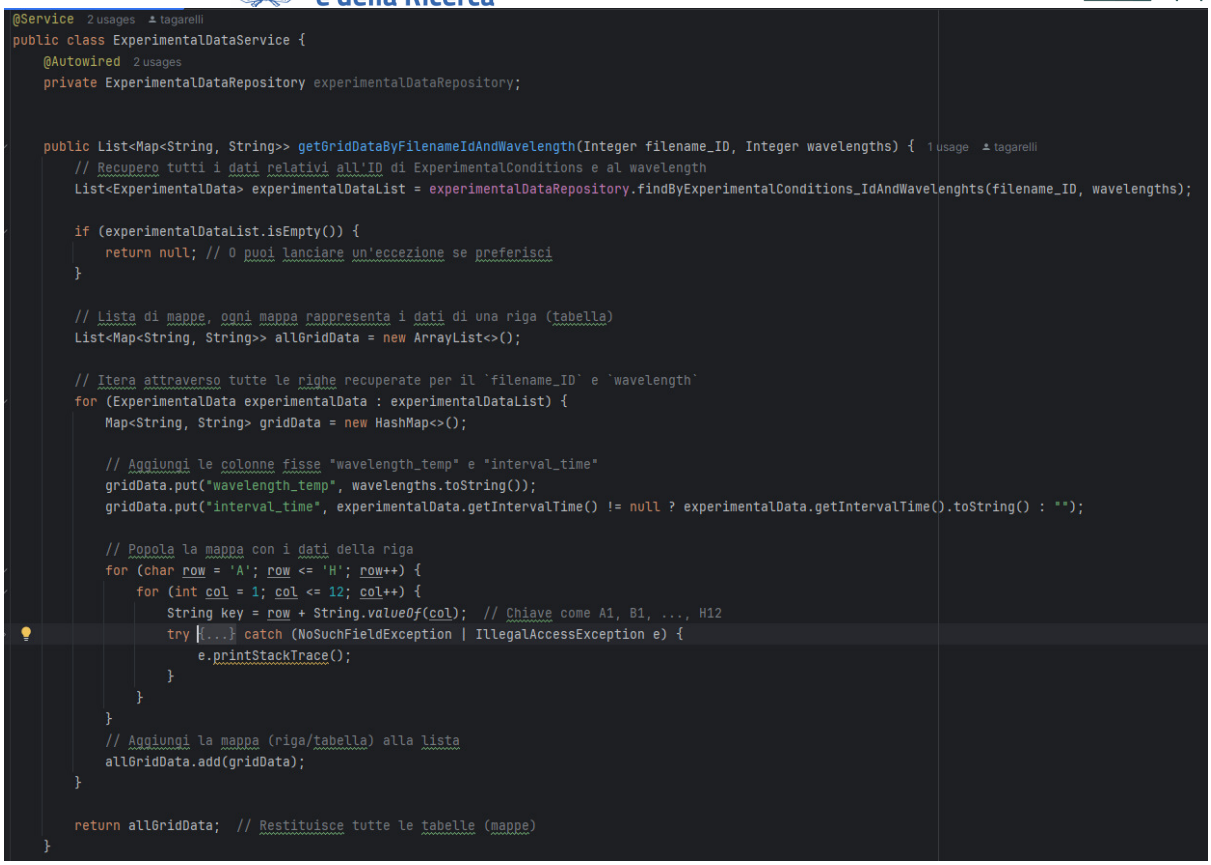
Il backend della piattaforma DSS è stato sviluppato in Java con il framework Spring che utilizza il pattern MVC (Model-View-Controller) separando le responsabilità di un'applicazione web in tre componenti principali:

- **Model**: rappresenta i dati e la logica di business, spesso mappati su entità JPA o oggetti di servizio.
- **View**: gestisce la presentazione dei dati dell'utente
- **Controller**: fa da intermediario tra Model e View, riceve le richieste dell'utente, richiama la logica di business e decide quale View restituire.

I model principali sono:

- OrganizationStrains
- ExperimentalData
- ExperimentalDataNew
- LastExperimentalData
- ExperimentalCoditions
- NewExperimentalConditions
- LastExperimentalConditions
- User

Il model OrganizationStrains rappresenta una piastra sperimentale con tutte le informazioni necessarie per descrivere campioni, pozzetti e condizioni sperimentali.
La classe ExperimentalData rappresenta i dati sperimentali estratti dal primo macchinario raccolti durante un esperimento. Ogni oggetto di questa entità corrisponde a una specifica condizione sperimentale indicato tramite una relazione alla tabella ExperimentalConditions.
All'interno dell'entità sono registrate informazioni generali come lunghezza d'onda, intervallo di tempo tra le misurazioni e la temperatura. Oltre a queste informazioni, viene memorizzata la disposizione dei dati nella piastra sperimentale, rappresentata dai pozzetti A1-H12. Ogni pozzetto contiene un valore relativo a quella posizione specifica, consentendo di conservare in maniera strutturata tutti i risultati di un esperimento condotto su una piastra multi-pozzetti.
La stessa struttura e logica di associazione tra condizioni sperimentali e dati raccolti è applicata ad altri due set di tabelle per il secondo macchinario (ExperimentalDataNew, NewExperimentalConditions) e ad ulteriori due tabelle per il terzo macchinario (LastExperimentalData, LastExperimentalConditions). In ciascun caso, le tabelle mantengono la stessa organizzazione concettuale, con una tabella principale che descrive le condizioni sperimentali e una tabella collegata che raccoglie i dati relativi ai pozzetti della piastra, garantendo così coerenza, tracciabilità e uniformità nella gestione dei dati provenienti da macchinari diversi.
Nel framework Spring, i Service rappresentano il livello dell'applicazione dedicato alla logica di business, ossia all'elaborazione dei dati e all'esecuzione delle operazioni principali che collegano il livello di accesso ai dati (Repository) con quello di presentazione (Controller).



```java
@Service  2 usages  tagarelli
public class ExperimentalDataService {
    @Autowired  2 usages
    private ExperimentalDataRepository experimentalDataRepository;


    public List<Map<String, String>> getGridDataByFilenameIdAndWavelength(Integer filename_ID, Integer wavelengths) {  1 usage  tagarelli
        // Recupero tutti i dati relativi all'ID di ExperimentalConditions e al wavelength
        List<ExperimentalData> experimentalDataList = experimentalDataRepository.findByExperimentalConditions_IdAndWavelenghts(filename_ID, wavelengths);

        if (experimentalDataList.isEmpty()) {
            return null; // O puoi lanciare un'eccezione se preferisci
        }

        // Lista di mappe, ogni mappa rappresenta i dati di una riga (tabella)
        List<Map<String, String>> allGridData = new ArrayList<>();

        // Itera attraverso tutte le righe recuperate per il `filename_ID` e `wavelength`
        for (ExperimentalData experimentalData : experimentalDataList) {
            Map<String, String> gridData = new HashMap<>();

            // Aggiungi le colonne fisse "wavelength_temp" e "interval_time"
            gridData.put("wavelength_temp", wavelengths.toString());
            gridData.put("interval_time", experimentalData.getIntervalTime() != null ? experimentalData.getIntervalTime().toString() : "");

            // Popola la mappa con i dati della riga
            for (char row = 'A'; row <= 'H'; row++) {
                for (int col = 1; col <= 12; col++) {
                    String key = row + String.valueOf(col);  // Chiave come A1, B1, ..., H12
                    try {...} catch (NoSuchFieldException | IllegalAccessException e) {
                        e.printStackTrace();
                    }
                }
            }
            // Aggiungi la mappa (riga/tabella) alla lista
            allGridData.add(gridData);
        }

        return allGridData;  // Restituisce tutte le tabelle (mappe)
    }
```

Figura 19:metodo recupero dati

Il metodo getGridDataByFilenameIdAndWavelength della classe ExperimentalDataService ha la funzione di recuperare e organizzare i dati sperimentali associati a un determinato esperimento, identificato dal suo filename ID e dalla lunghezza d'onda (wavelength) corrispondente. In particolare, il metodo interroga il repository dei dati sperimentali per ottenere tutti i record che corrispondono ai parametri forniti. Una volta recuperate le informazioni, i dati vengono elaborati e strutturati in una lista di mappe, in cui ogni mappa rappresenta una singola riga di dati, corrispondente a un set di misurazioni effettuate sulla piastra sperimentale.
All'interno di ciascuna mappa vengono inseriti i valori della temperatura, lunghezza d'onda e intervallo di tempo e i valori specifici di ogni pozzetto della piastra indicati con le coordinate da A1 a H12. I dati elaborati dal servizio vengono successivamente esposti attraverso il controller, che ne gestisce la disponibilità per l'interfaccia utente.
Una logica analoga è stata implementata anche per la gestione e il recupero dei dati provenienti dal secondo e terzo macchinario, mantenendo coerenza e uniformità nell'elaborazione delle informazioni