# Documentazione del Progetto LaboratorioA

**Team:** Digosciu Matteo, Faleiros Murilo, Franchi Davide.

**Università degli studi Insubria**

## 1. Descrizione Generale del Progetto

Il progetto ‘LaboratorioA’ è un’applicazione Java sviluppata per la gestione di un sistema di ristorazione, focalizzandosi su funzionalità di autenticazione, gestione utenti (amministratori, clienti, ospiti), gestione ristoranti e un sistema di feedback. L’applicazione è progettata per essere eseguita in un ambiente a riga di comando, utilizzando librerie come Lanterna e JLine per fornire un’interfaccia utente testuale interattiva e colorata. L’obiettivo principale è dimostrare l’implementazione di un’applicazione modulare e ben strutturata in Java, con una chiara separazione delle responsabilità tra i vari componenti (modelli, controller, servizi, vista).

### 1.1 Obiettivi del Progetto

Gli obiettivi principali di questo progetto includono:

* **Gestione Utenti:** Permettere la registrazione, l’autenticazione e la gestione di diversi tipi di utenti (amministratori, clienti, ospiti) con ruoli e permessi distinti.
* **Gestione Ristoranti:** Fornire funzionalità per la visualizzazione e la ricerca di ristoranti.
* **Sistema di Feedback:** Implementare un meccanismo per gli utenti per lasciare feedback sui ristoranti e per gli amministratori per gestire tali feedback.
* **Interfaccia Utente Interattiva:** Offrire un’esperienza utente intuitiva e reattiva tramite un’interfaccia a riga di comando avanzata.
* **Modularità e Manutenibilità:** Strutturare il codice in modo modulare per facilitare la manutenzione, l’estensione e la comprensione del progetto.
* **Utilizzo di Maven:** Dimostrare l’efficace gestione delle dipendenze e del ciclo di vita del progetto tramite Apache Maven.

### 1.2 Funzionalità Principali

Le funzionalità chiave implementate nel progetto sono:

* **Autenticazione e Registrazione:** Gli utenti possono registrarsi come nuovi clienti o accedere con credenziali esistenti. È presente anche una modalità ospite.
* **Gestione Ruoli:** Distinzione tra ruoli di Amministratore, Cliente e Ospite, con accesso differenziato alle funzionalità.
* **Visualizzazione Ristoranti:** Gli utenti possono visualizzare un elenco di ristoranti disponibili.
* **Ricerca Ristoranti:** Funzionalità di ricerca per filtrare i ristoranti in base a vari criteri.
* **Invio Feedback:** I clienti autenticati possono inviare feedback sui ristoranti.
* **Gestione Feedback (Amministratore):** Gli amministratori possono visualizzare e rispondere ai feedback.
* **Interfaccia a Riga di Comando:** Utilizzo di Lanterna per una UI testuale avanzata, inclusi menu, input e output colorati.
* **Persistenza Dati:** I dati degli utenti, dei ristoranti e dei feedback sono gestiti tramite file CSV, simulando una persistenza semplice.

### 1.3 Struttura dei Pacchetti e delle Cartelle

Il progetto segue una struttura standard Maven, con una chiara organizzazione dei pacchetti Java per promuovere la separazione delle responsabilità (MVC - Model-View-Controller e Service Layer). La directory principale del progetto è LaboratorioA/LaboratorioA.

Di seguito è riportata la struttura delle cartelle e dei pacchetti principali:



Questa struttura promuove una chiara separazione delle preoccupazioni, rendendo il codice più organizzato, leggibile e facile da mantenere. I modelli (model) rappresentano le entità dati, i controller (controller) gestiscono la logica di interazione e coordinano i servizi, i servizi (service) contengono la logica di business, e la vista (view) si occupa della presentazione all’utente. Le utilità (utils) forniscono funzioni di supporto generiche. La directory data è cruciale per la persistenza dei dati dell’applicazione.

## 2. Tecnologie e Strumenti Utilizzati

Il progetto è stato sviluppato utilizzando un set di tecnologie e strumenti standard nell’ecosistema Java, garantendo robustezza, manutenibilità e facilità di gestione del ciclo di vita del software.

### 2.1 Linguaggio: Java

Java è il linguaggio di programmazione principale utilizzato per lo sviluppo di questa applicazione. È un linguaggio orientato agli oggetti, robusto, sicuro e portabile, ampiamente utilizzato per applicazioni enterprise, mobili e desktop. In questo progetto, Java è impiegato per:

* **Definizione del Modello Dati:** Tutte le entità del dominio (es. Users, Restaurant, Feedback) sono definite come classi Java (POJO - Plain Old Java Objects) all’interno del pacchetto com.model.
* **Implementazione della Logica di Business:** La logica applicativa, come l’autenticazione (AuthService), la gestione dei feedback (FeedbackService)
* **Gestione delle Interazioni Utente:** I controller (com.controller) e le classi della vista (com.view) utilizzano Java per gestire l’input dell’utente, aggiornare lo stato dell’applicazione e presentare l’output.
* **Utilizzo di Librerie Esterne:** Il progetto sfrutta le capacità di Java per integrare librerie di terze parti (come Lanterna per la UI testuale e jBCrypt per l’hashing delle password) attraverso il sistema di gestione delle dipendenze Maven.

La versione di Java specificata nel pom.xml è Java 21, indicando l’utilizzo delle funzionalità più recenti del linguaggio.

### 2.2 Gestione Build e Dipendenze: Maven

Apache Maven è lo strumento centrale per la gestione del ciclo di vita del progetto, la compilazione e la gestione delle dipendenze. Il file pom.xml (Project Object Model) è il cuore della configurazione Maven del progetto.

#### 2.2.1 Dipendenze Principali

Le dipendenze dichiarate nel pom.xml sono fondamentali per le funzionalità dell’applicazione:

* **Lanterna**: Questa è una libreria cruciale per l’interfaccia utente. Lanterna è una libreria Java per creare interfacce utente testuali (TUI) in modo simile a ncurses o Turbo Vision. Permette di disegnare elementi grafici, gestire l’input da tastiera e visualizzare testo colorato nel terminale, fornendo un’esperienza utente ricca in un ambiente a riga di comando.
* **Jline**: JLine è una libreria per la gestione della console che fornisce funzionalità avanzate come il completamento automatico, la cronologia dei comandi e la modifica della riga di comando. È spesso usata in combinazione con Lanterna o altre librerie TUI per migliorare l’interazione dell’utente con l’applicazione a riga di comando.
* **jBCrypt**: jBCrypt è una libreria per l’hashing delle password che implementa l’algoritmo bcrypt. È utilizzata per archiviare le password in modo sicuro, trasformandole in un hash irreversibile prima di salvarle, proteggendo così le credenziali degli utenti anche in caso di violazione dei dati.

Queste dipendenze evidenziano la natura dell’applicazione come un’applicazione a riga di comando con funzionalità di sicurezza di base per la gestione delle password.

### 2.3 Controllo Versione: GitHub

Il progetto utilizza **GitHub** come sistema di controllo versione, consentendo una gestione strutturata e collaborativa del codice sorgente.  
Sono state definite due principali *branch*:

* **Main**: rappresenta la versione stabile e consolidata del progetto, destinata alla distribuzione o a contenere il codice pronto per l’utilizzo in produzione.
* **Test**: utilizzata per attività di sviluppo, sperimentazione e verifica di nuove funzionalità prima della loro integrazione nella branch principale.

Questa organizzazione facilita il flusso di lavoro, garantendo che le modifiche vengano testate e validate prima di essere incorporate nella versione stabile. ## 3. Processi e Metodologia di Sviluppo

Il progetto ‘LaboratorioA’, data la sua natura di applicazione Java con Maven, segue processi di sviluppo ben definiti per la compilazione, l’esecuzione e la gestione della qualità del codice. Sebbene non sia specificata una metodologia agile formale, la struttura del progetto suggerisce l’adozione di buone pratiche di ingegneria del software.

### 3.1 Compilazione ed Esecuzione del Progetto

La compilazione e l’esecuzione del progetto sono gestite principalmente tramite Apache Maven, che automatizza il processo di build e la risoluzione delle dipendenze.

#### 3.1.1 Esecuzione

Dopo la compilazione, il progetto può essere eseguito in diversi modi:

* **Tramite exec-maven-plugin**: Il pom.xml configura l’exec-maven-plugin per eseguire la classe com.Main. Questo è il modo più semplice per avviare l’applicazione direttamente da Maven:
* mvn exec:java
* Questo comando avvierà l’applicazione a riga di comando, che presenterà l’interfaccia utente testuale fornita da Lanterna.
* **Esecuzione del JAR compilato**: Dopo una build mvn clean install, un file JAR eseguibile (o un JAR contenente le classi compilate) viene generato nella directory target. Il nome del JAR sarà probabilmente restaurant-app-1.0-SNAPSHOT.jar. Per eseguirlo, è necessario assicurarsi che tutte le dipendenze siano disponibili nel classpath. Questo può essere fatto manualmente o utilizzando il classpath.txt generato:
* java -cp target/restaurant-app-1.0-SNAPSHOT.jar:$(cat classpath.txt) com.Main
* Oppure, se il JAR è un

JAR eseguibile con tutte le dipendenze incluse (tramite un plugin come maven-assembly-plugin o maven-shade-plugin, che non sembra essere configurato in questo pom.xml):

```bash

java -jar target/restaurant-app-1.0-SNAPSHOT.jar ```

Dato il `pom.xml` attuale, l'esecuzione tramite `exec-maven-plugin` o specificando il classpath manualmente è la via più diretta.

### 3.2 Buone Pratiche o Convenzioni di Codifica Presenti nel Progetto

Sebbene un’analisi approfondita richiederebbe la revisione di tutto il codice sorgente, alcune buone pratiche e convenzioni di codifica possono essere inferite dalla struttura del progetto e dai nomi dei file:

* **Separazione delle Responsabilità (MVC/Service Layer):** La chiara divisione in pacchetti controller, model, service, view e utils indica un’adesione al principio di separazione delle responsabilità. Questo rende il codice più modulare, testabile e facile da comprendere e mantenere. Ad esempio:
  + com.model: Contiene le classi che rappresentano i dati e la logica di dominio (es. Restaurant, Users, Feedback).
  + com.controller: Gestisce l’input dell’utente e coordina le interazioni tra il modello e la vista (es. AuthController, RestaurantController).
  + com.service: Incapsula la logica di business complessa e le operazioni sui dati (es. AuthService, CsvService, FeedbackService).
  + com.view: Si occupa della presentazione all’utente (es. UIMenu, ColoriAnsi).
  + com.utils: Contiene classi di utilità generiche (Helpers).
* **Convenzioni di Naming:** I nomi delle classi e dei pacchetti seguono le convenzioni standard di Java (CamelCase per le classi, lowercase per i pacchetti). Questo migliora la leggibilità del codice.
* **Gestione delle Eccezioni:** La presenza di blocchi try-catch-finally nel metodo main (com.Main) per la gestione delle eccezioni durante l’inizializzazione del terminale indica una consapevolezza della gestione degli errori, sebbene l’implementazione specifica debba essere valutata nel codice sorgente.
* **Persistenza Dati Semplice:** L’uso di file CSV nella directory data/ per la persistenza dei dati è una soluzione semplice e diretta per un progetto di laboratorio. Sebbene non sia una soluzione robusta per applicazioni di produzione, è adeguata per dimostrare i concetti di I/O e gestione dei dati.
* **Gestione Sicura delle Password:** L’integrazione di jbcrypt per l’hashing delle password è una buona pratica di sicurezza, prevenendo l’archiviazione delle password in chiaro.

## 4. Istruzioni per l’Uso

Questa sezione fornisce le istruzioni necessarie per configurare l’ambiente di sviluppo, compilare ed eseguire il progetto ‘LaboratorioA’ localmente.

### 4.1 Come Configurare l’Ambiente per Eseguire il Progetto

Per poter compilare ed eseguire il progetto, è necessario avere installati i seguenti strumenti:

1. **Java Development Kit (JDK) 21 o superiore:** Il progetto è configurato per utilizzare Java 21. È possibile scaricare il JDK dal sito ufficiale di Oracle o utilizzare distribuzioni open-source come OpenJDK. Assicurarsi che la variabile d’ambiente JAVA\_HOME sia impostata correttamente e che java e javac siano accessibili dal PATH del sistema.
   * **Verifica Installazione Java:** bash java -version javac -version
2. **Apache Maven 3.6.1 o superiore:** Maven è essenziale per la gestione del progetto e la compilazione. È possibile scaricarlo dal sito ufficiale di Apache Maven. Dopo l’installazione, aggiungere la directory bin di Maven al PATH del sistema.
   * **Verifica Installazione Maven:** bash mvn -version
3. **Un editor di testo o un IDE (Integrated Development Environment):** Sebbene non strettamente necessario per la compilazione da riga di comando, un IDE come IntelliJ IDEA, Eclipse o VS Code con il supporto Java e Maven è fortemente raccomandato per lo sviluppo, la navigazione del codice e il debug.
4. **Connessione a Internet:** Maven richiederà una connessione a Internet per scaricare le dipendenze dichiarate nel pom.xml dal repository Maven Central.

### 4.2 Comandi Maven Importanti

Una volta configurato l’ambiente e navigato nella directory radice del progetto (LaboratorioA/LaboratorioA), è possibile utilizzare i seguenti comandi Maven:

* **mvn clean**: Pulisce la directory target del progetto, rimuovendo tutti i file generati dalla build precedente. Questo è utile per assicurarsi che una nuova build parta da uno stato pulito. bash mvn clean
* **mvn compile**: Compila il codice sorgente Java del progetto. Questo comando non crea un pacchetto JAR, ma solo i file .class nella directory target/classes. bash mvn compile
* **mvn package**: Compila il codice sorgente, esegue eventuali test e impacchetta il codice compilato in un file JAR (o un altro formato di artefatto, a seconda della configurazione). Il file JAR verrà creato nella directory target. bash mvn package
* **mvn install**: Esegue le fasi di clean, compile, package e poi installa l’artefatto (il JAR) nel repository Maven locale (~/.m2/repository). Questo rende il JAR disponibile per altri progetti Maven sul sistema locale che potrebbero dipendere da esso. bash mvn install
* **mvn test**: Esegue tutti i test unitari e di integrazione configurati nel progetto. Se non ci sono test, questo comando completerà senza errori ma senza eseguire nulla di significativo. Come notato in precedenza, questo progetto non sembra avere test automatizzati configurati. bash mvn test
* **mvn exec:java**: Esegue l’applicazione Java direttamente da Maven, utilizzando la mainClass specificata nel pom.xml (com.Main). Questo è il modo più semplice per avviare l’applicazione per test rapidi o esecuzione diretta. bash mvn exec:java

### 4.3 Come Eseguire il Progetto Localmente

Per eseguire il progetto ‘LaboratorioA’ dopo aver configurato l’ambiente e scaricato le dipendenze, seguire questi passaggi:

1. **Navigare nella directory del progetto:** Aprire un terminale o un prompt dei comandi e navigare nella directory LaboratorioA/LaboratorioA (dove si trova il file pom.xml).

* cd /path/to/LaboratorioA/LaboratorioA

1. **Compilare il progetto:** Eseguire il comando Maven per compilare e installare il progetto. Questo scaricherà anche tutte le dipendenze necessarie.

* mvn clean install
* Attendere il completamento del processo. Verranno visualizzati messaggi relativi al download delle dipendenze e alla compilazione del codice. Al termine, dovrebbe apparire un messaggio BUILD SUCCESS.

1. **Eseguire l’applicazione:** Una volta che la build è stata completata con successo, è possibile avviare l’applicazione utilizzando l’exec-maven-plugin:

* mvn exec:java
* L’applicazione si avvierà nel terminale, presentando l’interfaccia utente testuale. Sarà possibile interagire con i menu e le funzionalità del sistema di gestione ristoranti.
* **Nota:** Se si riscontrano problemi di codifica dei caratteri durante l’esecuzione (es. caratteri speciali non visualizzati correttamente), assicurarsi che il terminale sia configurato per UTF-8. Su sistemi Linux/macOS, questo è solitamente il default. Su Windows, potrebbe essere necessario configurare il terminale (es. PowerShell o CMD) per UTF-8 (chcp 65001).

## 5. Conclusione

Il progetto ‘LaboratorioA’ rappresenta un esempio ben strutturato di applicazione Java a riga di comando, che dimostra l’uso di Maven per la gestione del progetto e l’integrazione di librerie esterne per un’interfaccia utente testuale interattiva e funzionalità di sicurezza di base. Sebbene sia un progetto di laboratorio, offre una solida base per comprendere i principi di sviluppo software in Java.

### 5.1 Vantaggi del Progetto

* **Struttura Modulare:** La chiara separazione in pacchetti model, controller, service, view e utils rende il codice facile da comprendere, mantenere ed estendere. Questo approccio promuove la riusabilità e riduce la complessità.
* **Gestione Efficiente delle Dipendenze con Maven:** L’uso di Maven semplifica enormemente la gestione delle librerie esterne, la compilazione e il packaging del progetto, automatizzando gran parte del ciclo di vita dello sviluppo.
* **Interfaccia Utente Interattiva:** L’integrazione di Lanterna e JLine fornisce un’esperienza utente ricca e interattiva in un ambiente a riga di comando, superando le limitazioni di un’interfaccia testuale standard.
* **Sicurezza delle Password:** L’utilizzo di jBCrypt per l’hashing delle password è una buona pratica di sicurezza fondamentale, proteggendo le credenziali degli utenti.
* **Persistenza Dati Semplice:** L’implementazione della persistenza tramite file CSV è un approccio diretto e comprensibile per un progetto didattico, che permette di salvare e caricare i dati senza la necessità di un database complesso.

### 5.2 Possibili Miglioramenti Futuri

Per trasformare ‘LaboratorioA’ da un progetto didattico a un’applicazione più robusta e pronta per la produzione, si potrebbero considerare i seguenti miglioramenti:

* **Introduzione di Test Automatizzati:** Implementare test unitari (es. con JUnit) per le classi di logica di business (service, model) e test di integrazione per i controller. Questo migliorerebbe significativamente la qualità del codice, la stabilità e la manutenibilità a lungo termine.
* **Persistenza Dati Robusta:** Sostituire la persistenza basata su file CSV con un database relazionale (es. H2, SQLite, PostgreSQL) o NoSQL.
* **Interfaccia Utente Grafica (GUI):** Per un’esperienza utente più moderna, si potrebbe sviluppare un’interfaccia utente grafica utilizzando framework come JavaFX o Swing, o addirittura un’applicazione web con Spring Boot e un frontend basato su JavaScript.
* **Refactoring e Pulizia del Codice:** Revisionare il codice per identificare e migliorare aree che potrebbero beneficiare di refactoring, come la riduzione della duplicazione del codice, l’applicazione di principi SOLID e l’ottimizzazione delle prestazioni.

Questi miglioramenti trasformerebbero il progetto ‘LaboratorioA’ in un’applicazione più completa e pronta per scenari d’uso reali, mantenendo al contempo la sua struttura modulare e i principi di buona progettazione software.