**Manuale Sviluppatore**

Questo manuale fornisce istruzioni dettagliate su come installare e lanciare il software sviluppato su un nuovo computer a partire dall'ottenimento del codice sorgente. Inoltre, fornisce una panoramica delle funzionalità implementate, includendo l'uso di design pattern specifici e i vantaggi derivati dalla loro adozione.

**Requisiti di Sistema**

* Java Development Kit (JDK) 11 o successivo
* Apache Maven 3.6.3 o successivo
* Xampp (che include PHP, MySQL, Apache)
* MySQL
* Un IDE compatibile con Maven (preferibilmente IntelliJ IDEA)
* Git

**1.Installazione**

1. Clonare il repository Git contenente il codice sorgente tramite url: [**https://github.com/leonedl/ProgettoSweng2024**](https://github.com/leonedl/ProgettoSweng2024)

**2.Configurare il Database**

Assicurarsi che XAMPP sia installato e in esecuzione. Creare un nuovo database utilizzando phpMyAdmin :

1. Aprire XAMPP Control Panel e avviare Apache e MySQL.
2. Aprire un browser web e andare su http://localhost/phpmyadmin.
3. Creare un nuovo database chiamato “swengdb”.

Aggiornare le configurazioni del database nel file src/main/resources/application.properties:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Nota: L'utente predefinito di MySQL in XAMPP è root e la password è vuota per impostazione predefinita. Se hai configurato una password per l'utente root, assicurati di aggiornare il campo.

**Compilare e Eseguire l'Applicazione**

1. Apri il progetto in IntelliJ IDEA.
2. Vai al menu Run > Edit Configurations.
3. Clicca su + e seleziona Maven
4. Configura la classe principale del progetto “ApplicationMain” e inserisci come comando del run “ spring-boot:run “
5. Clicca su Apply e poi su Run.

**4. Accesso all'Applicazione**

* 1. Una volta che l'applicazione è in esecuzione, apri un browser web e accedi a http://localhost:8080.

**PANORAMICA DELLE FUNZIONALITA’**

Il design pattern Model View Controller è stato utilizzato per gestire tutte le funzionalità dell’applicazione:

Nel contesto della gestione delle storie, è stato implementato come segue:

1. **Model (Modello):**
   * Le entità Story rappresentano i dati delle storie.
   * Il repository StoryRepository si occupa dell'accesso ai dati, utilizzando Spring Data JPA per interagire con il database.
2. **View (Vista):**
   * I file Thymeleaf (es. catalog.html) rappresentano la vista. Questi file contengono il markup HTML che visualizza le storie agli utenti.
3. **Controller:**
   * Il controller StoryController gestisce le richieste HTTP, coordina con i servizi per elaborare i dati e determina quale vista restituire.
4. **Service:**
   * Il servizio StoryService contiene la logica di business per la gestione delle storie. È un'interfaccia tra il controller e il repository, garantendo che il controller rimanga focalizzato sulla gestione delle richieste.

L'adozione del pattern Model-View-Controller (MVC) nella gestione delle storie ha portato a numerosi vantaggi specifici che hanno migliorato l'organizzazione, la manutenibilità e l'efficienza dello sviluppo del software.

**1. Separazione delle Responsabilità**

* **Modello:** La classe Story e il repository StoryRepository sono responsabili della gestione dei dati delle storie e dell'accesso al database. Questo significa che la logica di gestione dei dati è centralizzata e non è sparsa per il codice dell'applicazione.
* **Vista:** I file Thymeleaf gestiscono la presentazione delle storie agli utenti. Questo separa chiaramente il codice di presentazione dalla logica di business, rendendo più semplice la modifica della UI senza toccare la logica sottostante.
* **Controller:** Il StoryController gestisce le richieste HTTP e coordina le interazioni tra la vista e il modello. Questo mantiene il flusso dell'applicazione chiaro e ben definito.

**2. Facilità di Manutenzione**

La separazione delle responsabilità rende il codice più facile da mantenere. Le modifiche possono essere effettuate in modo isolato all'interno dei rispettivi componenti:

* **Modifiche ai dati:** Eventuali modifiche alla struttura dei dati delle storie (ad esempio, aggiungere un nuovo campo) possono essere fatte nel modello senza influenzare il controller o la vista.
* **Modifiche alla UI:** Cambiare l'aspetto o il layout delle pagine Thymeleaf può essere fatto senza alterare la logica di business o il flusso dell'applicazione.
* **Logica di business:** Aggiungere nuove funzionalità o modificare la logica di business nel servizio StoryService è possibile senza alterare il codice del controller o della vista.

**3. Testabilità**

L'adozione del pattern MVC facilita il testing:

* **Unit Testing:** I singoli componenti possono essere testati indipendentemente. Ad esempio, la logica di business nel StoryService può essere testata senza coinvolgere il controller o la vista.
* **Integration Testing:** L'interazione tra i componenti può essere testata in modo chiaro e modulare.

**4. Scalabilità**

La chiara separazione delle responsabilità permette di scalare l'applicazione con maggiore facilità:

* **Aggiunta di Nuove Funzionalità:** Nuove funzionalità possono essere aggiunte implementando nuovi controller, servizi e modelli senza impattare il codice esistente.
* **Distribuzione del Carico di Lavoro:** Il lavoro di sviluppo può essere distribuito tra diversi team o sviluppatori, con ciascuno concentrato su uno specifico componente (modello, vista o controller).

**5. Chiarezza del Codice**

Il codice è organizzato in modo che sia facile da comprendere:

* **Logica di Business Isolata:** La logica di business è isolata nel StoryService, rendendo il codice più chiaro e più semplice da capire.