Documentazione

1.Software Engineering Management

Project Plan

Indice

1. Introduzione

Il progetto consiste nello sviluppo di un sistema per la gestione di una biblioteca digitale. L'obiettivo principale è creare un'applicazione che permetta di gestire i libri e gli utenti della biblioteca con un sistema di autenticazione e ruoli (es. utenti standard e amministratori). Le funzionalità principali che implementeremo sono :

1. Gestione degli Utenti:
- Registrazione e autenticazione (login/logout).
- Ruoli di utenti: normali e amministratori.
- Profilo utente.

2. Gestione dei Libri:
- Aggiunta, modifica e cancellazione dei libri (solo per amministratori).

- Ricerca di libri (per titolo, autore, genere, etc.).

3. Sistema di Prestito:

- Richiesta di prestito di un libro.
- Restituzione di un libro.
- Storico dei prestiti.

2. Modello di processo

Il nostro team userà come ciclo di vita del software l'extreme programming con riunioni di pianificazione brevi riguardanti

l'immediata funzionalità da consegnare. Lo sviluppo prevederà il rilascio di versioni di prodotto funzionanti.

Il codice dell'applicazione può essere manipolato da qualsiasi sviluppatore e verrà scritto in base a regole condivise.

L'architettura dell'applicazione sarà più semplice possibile in modo che l'utente medio non abbia problemi nel capire il funzionamento.

3. Organizzazione progetto

Il team, composto da pochi membri, è organizzato secondo un approccio agile che non prevede gerarchie tra i membri.

Verranno quindi usati canali di comunicazione brevi ed un atteggiamento orientato alle persone piuttosto che formalistico. Inoltre settimanalmente si terrà un meeting tra i membri del gruppo per pianificare le attività che dovranno essere svolte entro la successiva riunione.

Per quanto riguarda la divisione dei ruoli all'interno del team si rimanda al paragrafo 7(Personale).

Infine in accordo con il ciclo di vita del software(XP) viene anche seguita la pratica del whole team ossia un gruppo che include persone con diverse competenze specialistiche.

4. Standard, linee guida, procedure

- Standard

Il progetto segue le convenzioni per la programmazione Java come come definite da Oracle (per info: Code Conventions for the Java Programming Language: Contents).

- Linee guida

Come descritto nel paragrafo 2 (Modello di processo) , le linee guida seguono le best practices di Xp (extreme programming). Inoltre questo argomento è ulteriormente approfondito nel paragrafo successivo.

- Procedure:

Dopo aver effettuato l'analisi dei requisiti, si procederà con la realizzazione dei diagrammi UML e successivamente con l'implementazione del codice. Saranno inoltre svolti vari test per verificare il corretto funzionamento del software.

La documentazione relativa al progetto verrà consegnata entro il 24 Dicembre 2024 ed i membri del gruppo decideranno di volta in volta, di comune accordo, come procedere con

l'aggiornamento della stessa.

5. Attività di gestione

Le attività di gestione sotto indicate serviranno per la maggior parte come linee guida che utilizzeremo per fornire uno sviluppo più lineare e per dare delle priorità più dettagliate:

- Incontri: principalmente ci saranno degli incontri all'inizio di ogni settimana dove si decideranno le attività che dovranno essere svolte per l'incontro successivo
- Git: utilizzo di git nel caso si riscontrino problemi e al fine di tenere tracciate le varie modifiche del progetto
- WhatsApp: per decisione orario incontri e per comunicazioni che hanno bisogno di una risposta immediata
-

6. Rischi

Rischio di non consegnare il sistema completo entro la scadenza prestabilita

Rischio di mancanza di informazioni critiche: a seguito della raccolta dei requisiti del progetto

alcune informazioni critiche potrebbero non essere state recepite con chiarezza e ciò potrebbe

richiedere ulteriori iterazioni dello sviluppo.

Rischi improvvisi: potrebbero sorgere problemi improvvisi con la costruzione del sistema oppure

inaspettate mancanze dei componenti del team per problemi inaspettati

7. Personale

Team members:

- Epis Davide addetto alla fase di modellazione
- Tironi Matteo responsabile della parte grafica (Java GUI)
- Morina Florian addetto alla parte di programmazione su eclipse

I ruoli sopra definiti indicano la principale attività svolta da ogni membro ma questo non implica l'esclusione dalle altre mansioni.

8. Metodi e tecniche

I metodi e le tecniche utilizzate durante le varie fasi del progetto, come ad esempio l'ingegneria dei requisiti e la fase di testing, saranno:

- Documentazione: la documentazione, fondamentale per la fase di manutenzione, sarà prodotta usando Javadoc. Tale scelta permette una comprensione più approfondita del codice, facendo sviluppare così una visione più ampia sulla funzione dell'intera applicazione.
- Scrittura dei test e determinazione delle milestones: -?-

- Ambiente di testing e strumentazione: verrà prestata particolare attenzione alla definizione dell'ambiente di prova e delle apparecchiature che saranno utilizzate, in quanto la veridicità dei test influirà sulle proprietà del software.
- Pianificazione di test e procedure di accettazione: l'ordine dei test e di integrazione verrà indicato chiaramente, così da poter ragionare in maniera attenta e corretta su ogni elemento, in particolare le varie procedure di test di accettazione verranno definite in modo da garantire una buona valutazione delle varie funzioni implementate.
- Ambiente di sviluppo: come IDE per lo sviluppo del codice verrà utilizzate Eclipse con, come linguaggio di programmazione, Java.
-

9. Garanzie di qualità

I fattori di qualità che verranno presi in considerazione durante tutto lo sviluppo del progetto saranno:

- Correttezza: il grado in cui verranno soddisfatte le specifiche del programma e i bisogni dell'utente
- Affidabilità: il fatto che il programma per svolgere la sua funzione utilizzi una precisione adeguata
- Efficienza: rispettare le risorse a nostra disposizione e utilizzandole al meglio per evitare sprechi
- Integrità: DA FARE SUCCESSIVAMENTE
- Usabilità: " "
- Manutenibilità: l'architettura del sistema verrà sviluppata cercando di ridurre il grado di dipendenza tra i componenti, cercando di ottenere più agevolazione sulle attività di correzione degli errori e l'inserimento di nuove funzionalità
-

10. Workpackages

Il progetto verrà suddiviso in attività, che saranno assegnate ai singoli membri del team, come indicato nel paragrafo 7 (Personale).

2.Software Life Cycle

Data la nostra poca esperienza in una situazione lavorativa come quella presupposta dalla consegna del progetto ma comunque la nostra voglia di offrire a coloro che saranno gli utenti finali un programma di qualità tutt'altro che scarsa, abbiamo deciso di ripiegare su un metodo agile ed in particolare il metodo di *eXtreme Programming* o XP, così che il nostro programma sia sempre in esecuzione ed in linea rispetto a quello che vogliamo realizzare.

Questo metodo agile si compone di varie *best practices* ed in particolare quelle che ci sono state maggiormente utili sono:

- Intera Squadra (*whole team*)
- La programmazione in coppia
- La proprietà collettiva del codice
- Uno standard di codifica comune
- Adottamento di un ritmo sostenibile

In Particolare uno dei principi chiave, su cui ci siamo focalizzati, di XP è il Cambiamento Incrementale ovvero che il nostro progetto cambia grazie a piccoli cambiamenti.

Infatti il codice cambia un po' alla volta così come il design ma anche l'esperienza della squadra, questi sono tutti aspetti che nello svilupparsi del progetto migliorano di continuo.

3.Configuration Management

Per condividere il progetto tra di noi, e aggiornarlo comunicandoci anche le varie aggiunte fatte, abbiamo utilizzato git hub in particolare abbiamo usato:

Il branch, o sezione, principale del repository è costituita da due cartelle:

- Src/Biblioteca, al cui interno sono contenuti il codice e l'interfaccia grafica;
- docs, al cui interno è presente la documentazione quindi il project plan, la documentazione generale e i diagrammi UML.

Git Issues:

Commit:

Git Branch:

Git Ignore

(Per maggiori specifiche sugli issue e sull'uso di GitHub si rimanda al repository: DE_FM_MT_Biblioteca)

4.People Management and Team Organization

L'organizzazione del nostro team con le varie occupazioni (già spiegata nel punto 7 del Project Plan) saranno come segue:

Epis Davide addetto alla fase di modellazione

Tironi Matteo responsabile della parte grafica (Java GUI)

Morina Florian addetto alla parte di programmazione su eclipse

Anche se i vari componenti hanno ruoli e compiti principali ciò non esclude che in generale ogni componente del team si potrà occupare di ogni elemento del lavoro come codice, documentazione e grafici UML.

5.Software Quality

Per la qualità del nostro progetto abbiamo deciso di affidarci alla classificazione dei fattori di qualità stabilità da McCall, con in particolare alcune delle sue categorie, e dall''ISO 9126.

Le categorie che sono state selezionate sono:

- Correttezza: definisce cioè che si vuole che il sistema soddisfi, i requisiti descritti e che permetta di raggiungere gli obiettivi preposti;
- Affidabilità: si vuole che il sistema sia in grado di eseguire la sua funzione in modo accurato e senza problemi, garantendo che sia in grado di tollerare i malfunzionamenti;

- Efficienza: è la quantità di risorse di calcolo e di codice richiesta da un programma per eseguire una funzione;
- Integrità: è la misura per cui il sistema deve mantenere i dati personali protetti, senza lasciare che persone non autorizzate possano accedervi e modificarne il contenuto;
- Usabilità: si necessita che il sistema sia di facile comprensione per tutti gli utenti e che sia quindi facilmente e intuitivamente utilizzabile;
- Manutenibilità: è la misura per cui si garantisce che il programma possa essere modificato dopo la sua consegna, nel caso in cui siano rivelati malfunzionamenti o bug. Si vuole anche che il programma sia facilmente leggibile dai programmatori, e ciò serve per rendere più semplice la manutenzione successiva del sistema;
- Portabilità: lo sforzo richiesto per trasferire il programma da un ambiente, che sia hardware o software, ad un altro;
- Interoperabilità: ossia lo sforzo richiesto per accoppiare un sistema ad un altro.

6.Requirement engineering

Caratteristiche dell'utente

Gli utenti saranno formati riguardo al funzionamento del sistema software.
Gli utenti finali useranno il nostro programma, senza la necessità di avere alcuna conoscenza specifica sul suo funzionamento.

Tecniche di elicitazione dei requisiti

Il requisito principale di informazione per il processo di elicitazione dei requisiti sono gli utenti dell'applicazione, a tale scopo sono stati coinvolti nostri amici e conoscenti (con un range di età ampio) per riuscire così ad identificare un campione che potesse rappresentare un insieme generale di utenti e grazie a ciò, ci è stato facile raccogliere dati su ciò che dei possibili utenti vorrebbero vedere aggiunto in una app come la nostra.

Un altro aspetto fondamentale sono i gestori grazie ai quali siamo riusciti ad identificare delle funzionalità necessarie per migliorare il sistema di log in e accesso alla biblioteca digitale.

In conclusione abbiamo utilizzato come tecniche di elicitazione: l'intervista con un gruppo di utenti e l'analisi delle attività per controllare l'identificazione dei membri e registrare i libri presi in prestito o venduti.

Ovviamente abbiamo tenuto in considerazione la "classificazione" MoSCoW cioè quelle funzioni che il sistema deve, dovrebbe, potrebbe e non dovrebbe avere.

Prototipazione

Per lo sviluppo del codice utilizzeremo la prototipazione, che ci permetterà di avere man mano una versione sempre più aggiornata e con più funzionalità.

Tramite questo inoltre avremo in qualsiasi fase dello sviluppo un prodotto funzionante, al quale potremo poi aggiungere migliorie.

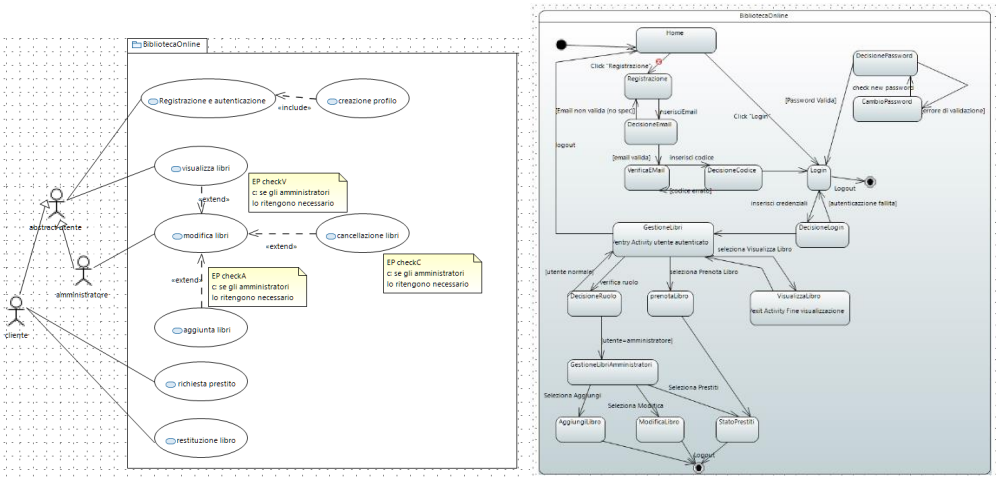
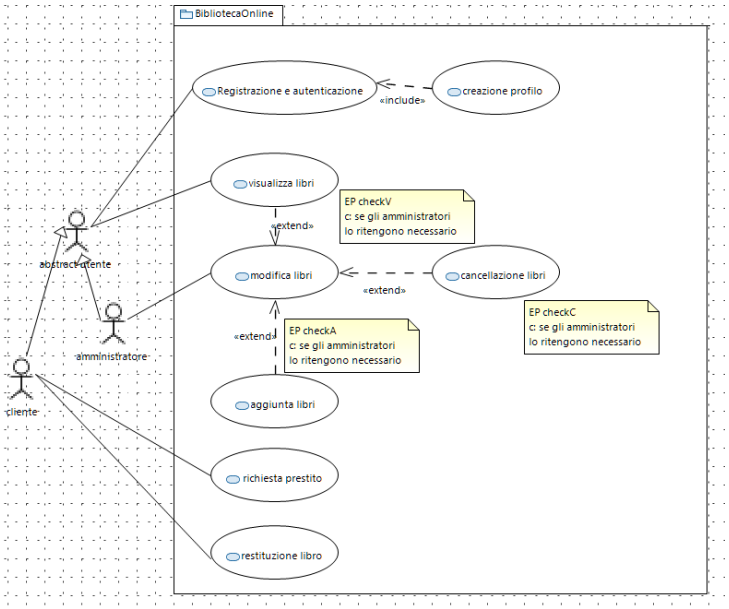
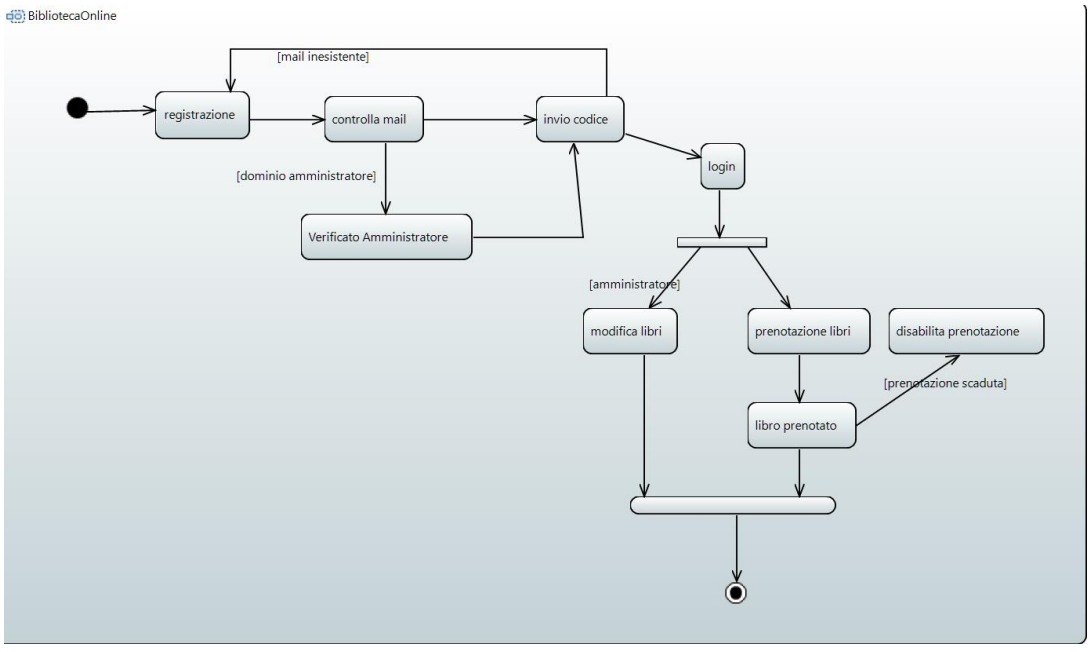
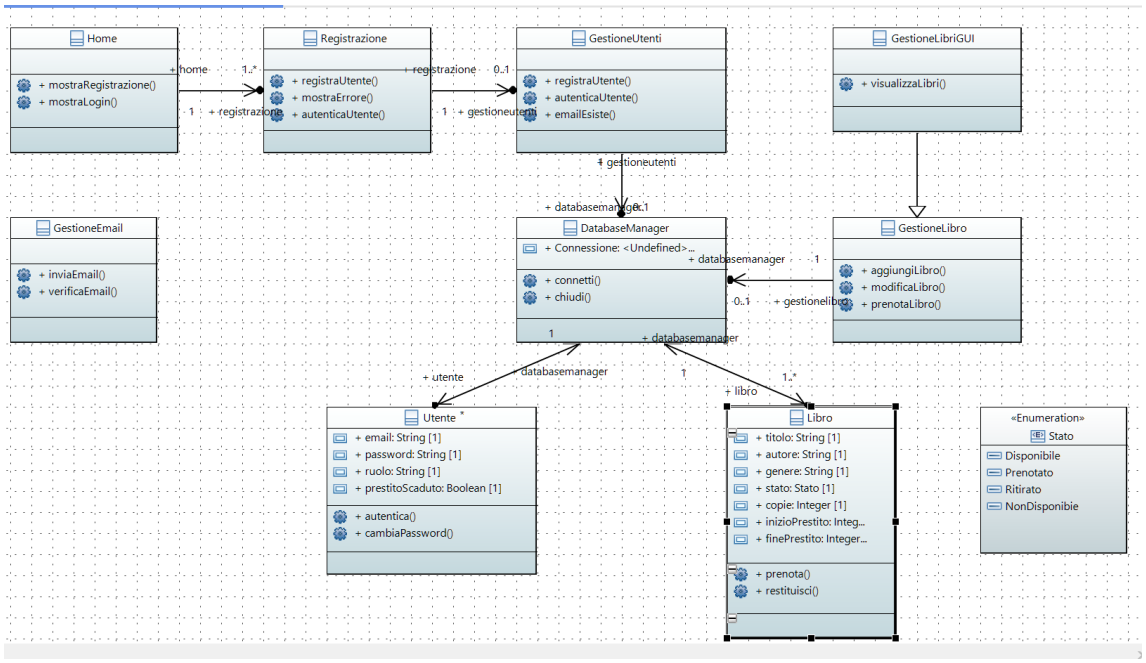


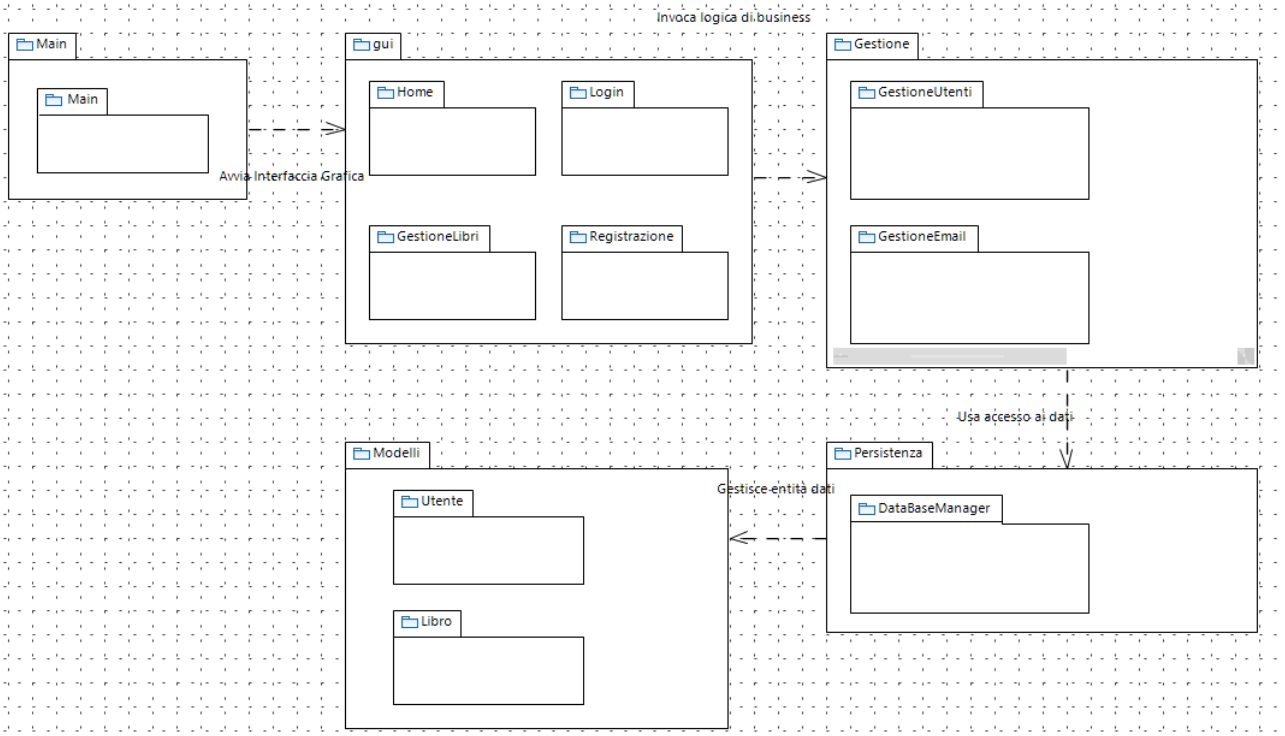
Tabella 9.3

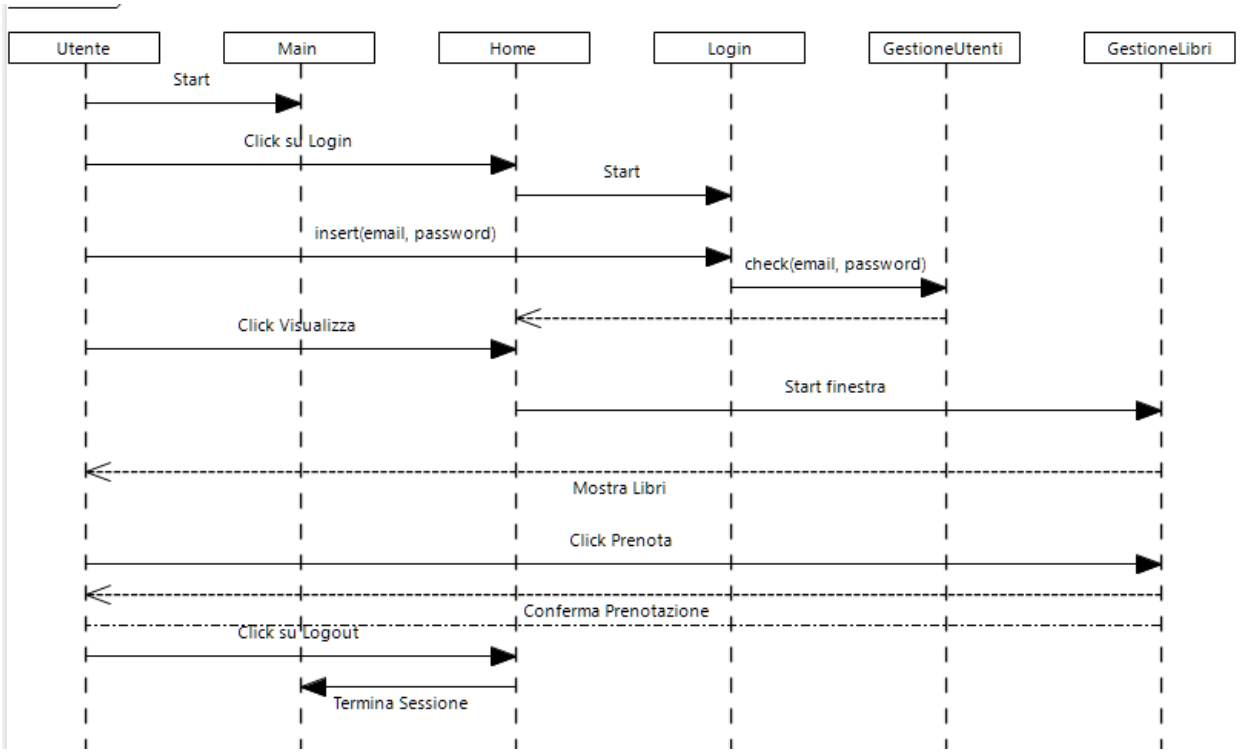
7.Modelling



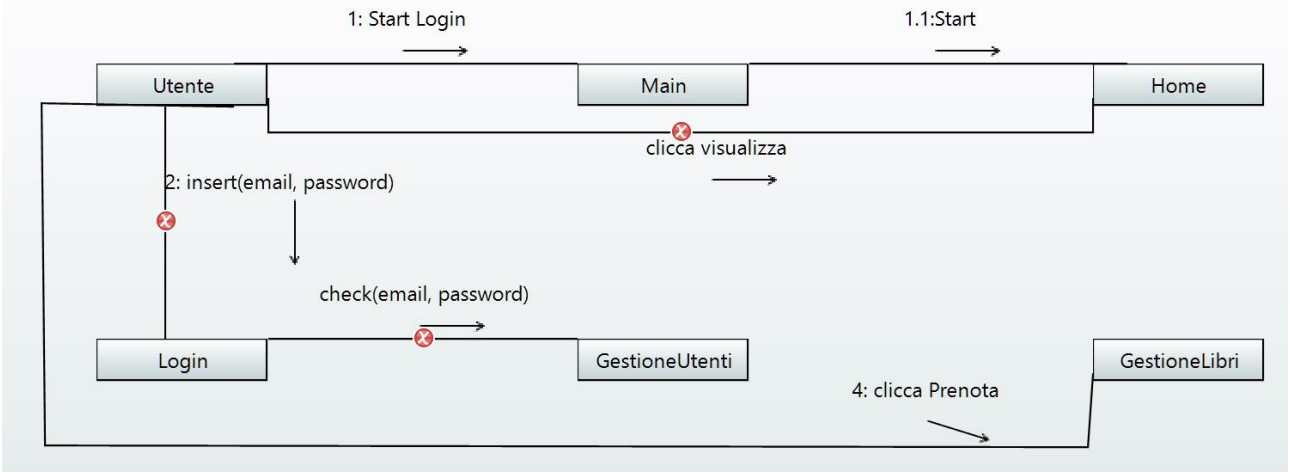


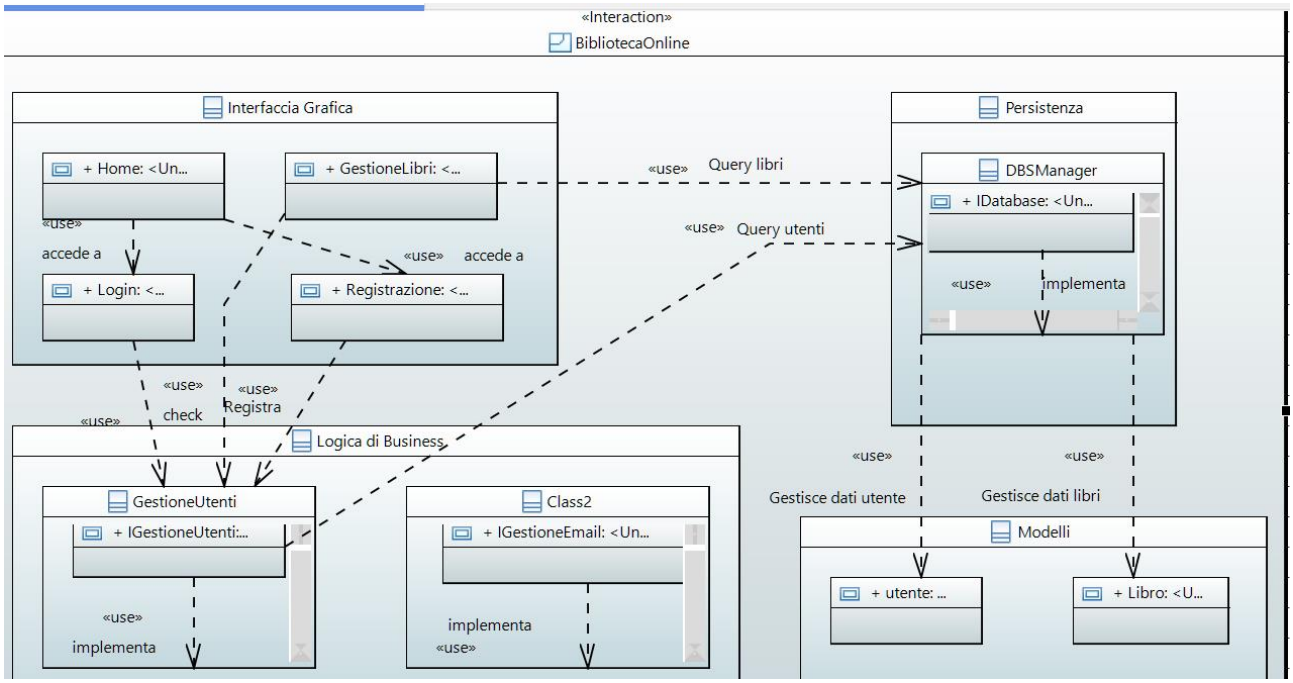

Home
+ mostraRegistrazione()
+ mostraLogin()
+home
1..*
1
+ registrazione
Registrazione
+ registraUtente()
+ mostraErrore()
+ autenticaUtente()
+ registrazione
0..1
1
+ gestioneutenti
GestioneUtenti
+ registraUtente()
+ autenticaUtente()
+ emailEsiste()
GestioneLibriGUI
+ visualizzaLibri()
+ gestioneutenti
+ databasemanager
0..1
GestioneEmail
+ inviaEmail()
+ verificaEmail()
DatabaseManager
+ Connessione: <Undefined>...
+ connetti()
+ chiudi()
+ databasemanager
1
0..1
+ gestionelibro
GestioneLibro
+ aggiungiLibro()
+ modificaLibro()
+ prenotaLibro()
1
+ databasemanager
+ utente
databasemanager
1
1..*
+ libro
Utente
*
+ email: String [1]
+ password: String [1]
+ ruolo: String [1]
+ prestitoScaduto: Boolean [1]
+ autentica()
+ cambiaPassword()
Libro
+ titolo: String [1]
+ autore: String [1]
+ genere: String [1]
+ stato: Stato [1]
+ copie: Integer [1]
+ inizioPrestito: Integ...
+ finePrestito: Integer...
+ prenota()
+ restituisci()
«Enumeration»
Stato
Disponibile
Prenotato
Ritirato
NonDisponibie

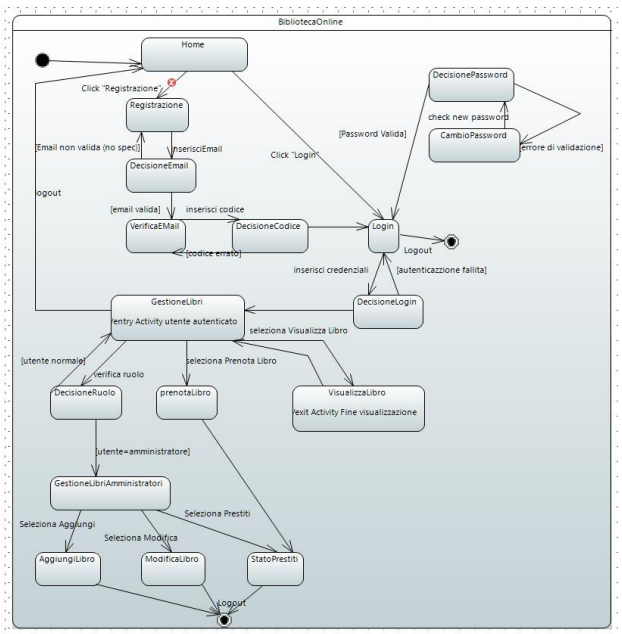

Invoca logica di business
Main
Main
Main
Avvia Interfaccia Grafica
gui
Home
Login
GestioneLibri
Registrazione
Gestione
GestioneUtenti
GestioneEmail
Usa accesso ai dati
Modelli
Utente
Libro
Gestisce entità dati
Persistenza
DataBaseManager



BibliotecaOnline







8.Software Architecture

Vista Logica (Logical View): Questa vista rappresenta la struttura statica del sistema, inclusi i componenti e le loro relazioni.

Per il nostro progetto sono:

Moduli: GUI, Logica di Business, Persistenza, Modelli.

Diagrammi utili: Class Diagram.

Vista dei Componenti e Connettori (Component and Connector View): Mostra i componenti in esecuzione e le interazioni tra di essi.

Componenti: DatabaseManager, GestioneUtenti, GestioneLibri, ecc.

Connettori: interfacce o chiamate a metodi che collegano i componenti.

Diagrammi utili: Component Diagram.

Una Vista con Connettori e Componenti e questa deve includere:

GUI (es. Home, Login).

Logica di Business (es. GestioneLibri, GestioneUtenti).

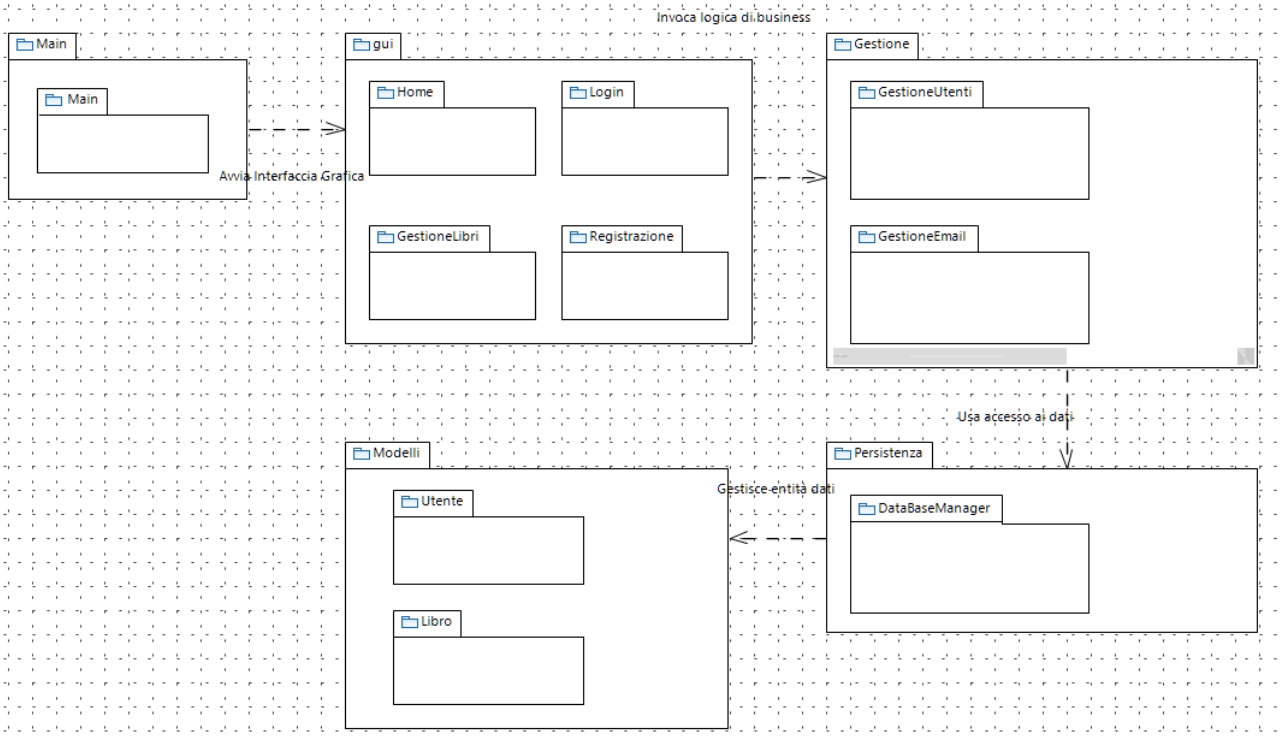
Persistenza (DatabaseManager).

Connettori:

Chiamate di procedura: Relazioni tra componenti.

Flusso di dati: Query dal gestore persistente al database.

Dati condivisi: Modelli (Utente, Libro).



Viste e tabella 11.4, maven

9.Software design

Per il design del nostro progetto ci siamo affidati in particolare ai cosiddetti design pattern in particolare:

Singleton (non esplicitamente presente, ma suggerito per DatabaseManager): assicura che una classe abbia una sola istanza e fornisce un punto globale di accesso.

Nel progetto: La classe DatabaseManager è candidata ideale per implementare il Singleton. Se non è ancora così, sarebbe utile applicare questo pattern per gestire l'accesso al database in modo centralizzato ed evitare la creazione di più connessioni.

```
public class DatabaseManager {

    private static DatabaseManager instance;

    private Connection conn;
```

```
    private DatabaseManager() {
        // Configurazione della connessione al database
    }
    public static synchronized DatabaseManager getInstance() {
        if (instance == null) {
            instance = new DatabaseManager();
        }
        return instance;
    }
    public Connection getConnection() {
        return conn;
    }
}
```

Observer (non presente ma applicabile per GestioneEmail): permette la comunicazione tra classi riducendo l'accoppiamento. Ideale per notificare eventi come invio di email o modifiche allo stato del sistema. in particolare per: notificare gli utenti (es. invio di email in GestioneEmail).

La classe GestioneEmail può essere un Publisher.

Gli osservatori possono rappresentare diverse modalità di notifica (es. email, SMS).

```
public interface Observer {
    void update(String message);
}
public class EmailNotificator implements Observer {
    public void update(String message) {
        System.out.println("Inviando email: " + message);
    }
}
public class GestioneEmail {
    private List<Observer> observers = new ArrayList<>();
    public void addObserver(Observer observer) {
        observers.add(observer);
    }
    public void notifyObservers(String message) {
```

```
        for (Observer observer : observers) {
            observer.update(message);
        }
    }
}
```

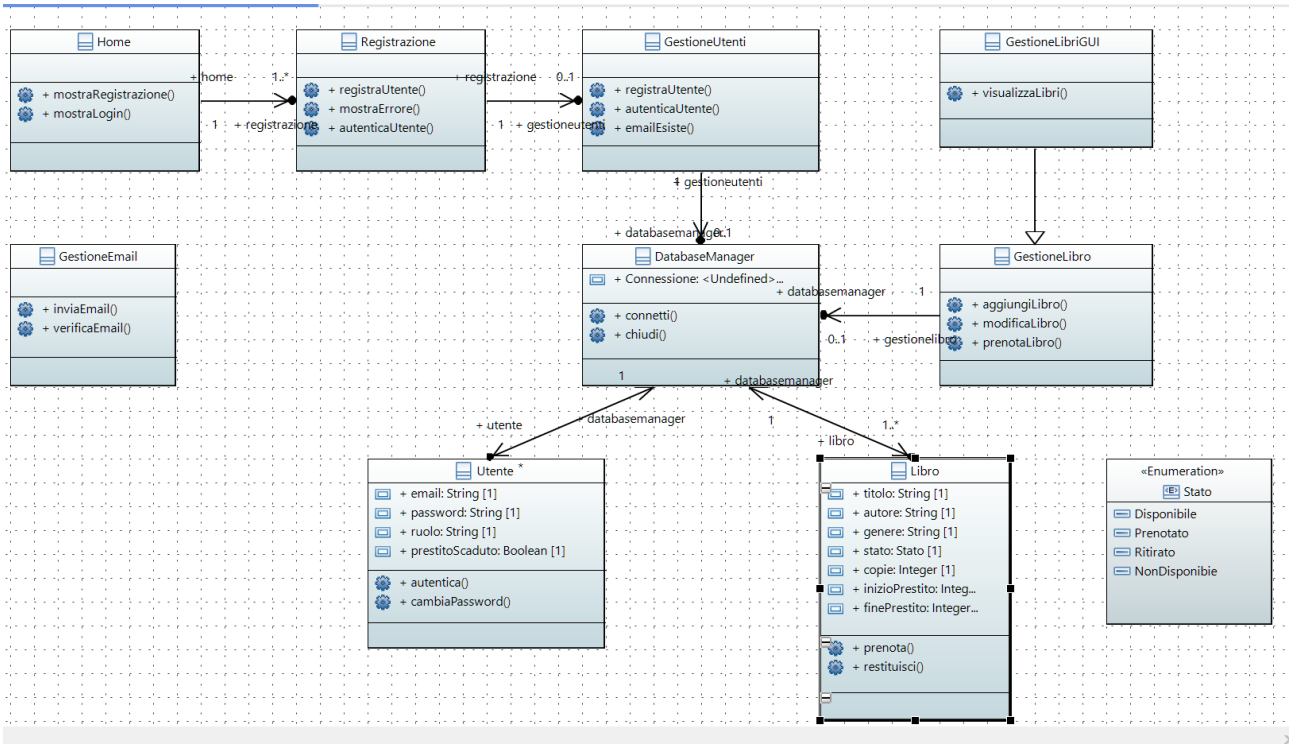
Strategy (non presente ma utile per GestioneLibri): permette di selezionare dinamicamente un algoritmo tra più opzioni. Questo è utile se si vuole implementare logiche diverse per operazioni sui libri (es. diverse regole di prenotazione). In particolare per Gestione di algoritmi di ordinamento o filtri sui libri.

```
public interface BookSortingStrategy {
    void sort(List<Book> books);
}
public class SortByTitle implements BookSortingStrategy {
    public void sort(List<Book> books) {
        books.sort(Comparator.comparing(Book::getTitle));
    }
}
public class SortByAuthor implements BookSortingStrategy {
    public void sort(List<Book> books) {
        books.sort(Comparator.comparing(Book::getAuthor));
    }
}
public class GestioneLibri {
    private BookSortingStrategy sortingStrategy;

    public void setSortingStrategy(BookSortingStrategy strategy) {
        this.sortingStrategy = strategy;
    }
    public void sortBooks(List<Book> books) {
        sortingStrategy.sort(books);
    }
}
```

Factory (non presente ma utile per creare oggetti Utente o Libro): delega la creazione di oggetti a una classe centralizzata, permettendo maggiore flessibilità. In particolare per creare oggetti Utente o Libro basati su parametri specifici.

```
public class UserFactory {
   public static Utente createUser(String role, String email, String password) {
      if (role.equals("admin")) {
         return new AdminUser(email, password);
      } else {
         return new StandardUser(email, password);
      }
   }
}
```



Design patterns, tools e McCabel - insieme

10.Software Testing

Test - alla fine

11.Software Maintenance

La manutenzione che meglio rappresenta quella adottata principalmente nel nostro progetto è la manutenzione preventiva, ovvero l'insieme di attività finalizzate ad aumentare la manutenibilità del sistema. Tra queste rientrano l'aggiornamento della documentazione, l'aggiunta di commenti e il miglioramento della struttura del sistema.

In particolare per la manutenzione del nostro progetto ci siamo affidati al *refactoring*, ovvero alla ristrutturazione del codice, che rappresenta una delle pratiche fondamentali della pratica dell'*Extreme Programming* (XP).
Nel nostro caso, l'attività di *refactoring* non è stata pianificata in modo esplicito; si può infatti affermare che questa attività di manutenzione si sia svolta in modo spontaneo durante la nostra progettazione quotidiana.

Per mantenere il controllo sul nostro progetto attraverso il *refactoring*, ci siamo focalizzati sulla ricerca di "cattivi odori" nel codice, tra cui:

- Metodi lunghi: l'individuazione di metodi eccessivamente lunghi;
- Campi temporanei: variabili di classe utilizzate solo in circostanze specifiche;
- *Refused Bequest* (lascito rifiutato): una sottoclasse che non supporta tutti i metodi o i dati ereditati;
- Classe pigra: una classe che non svolge un ruolo significativo;
- Codice duplicato: la ripetizione di porzioni di codice in più punti;
- Catene di messaggi: situazioni in cui una classe chiede un oggetto a un'altra classe, che a sua volta ne chiede un altro, creando una catena di richieste;
- Middle Man: una classe che delega la maggior parte dei propri compiti ad altre classi. Va notato che delegare non è intrinsecamente un problema, ma lo diventa quando una classe delega eccessivamente;
- Libreria incompleta: l'uso di una libreria che non fornisce tutte le funzionalità necessarie per l'attività in corso.