Museum visits manager

Gio Formichella, Davide Lombardi, Michael Bartoloni

Indice

1 Introduzione

Il nostro programma offre a musei funzionalità di organizzazione di visite guidate e pubblicizzazione mediante newsletter e catalogo opere. Ai futuri visitatori permette di prenotarsi alle visite e rimanere informati riguardo agli appuntamenti e alle notizie relative al museo.

- L'ufficio prenotazioni organizza le visite fissando una data, un orario, uno o più itinerari, una guida(che parla una certa lingua), il numero massimo di visitatori e il prezzo del biglietto di ingresso. Le visite possono essere modificate e cancellate e, al termine di queste operazioni, vengono notificati tutti i visitatori interessati via email.

- Il curatore museale può fissare e cancellare itinerari e aggiungere opere d'arte agli itinerari esistenti; ogni opera ha un nome, un autore ed uno stato. Le nuove aggiunte alla collezione vengono pubblicizzate mediante il servizio di newsletter. È anche possibile modificare lo stato dell'opera tra: "in esposizione", "in manutenzione" e "in prestito al museo/collezione x". Se un'opera passa dallo stato di esposizione ad uno degli altri due vengono notificati i visitatori la cui visita avrebbe incluso l'opera rimossa, se, invece, un'opera torna allo stato espositivo ad essere notificati sono gli iscritti alla newsletter.

- Gli interessati al servizio visite hanno la possibilità di visualizzare le visite organizzate, con le relative informazioni, e prenotare uno o più posti se il limite massimo di posti disponibili non è stato raggiunto. Al termine della prenotazione è data la possibilità di pagare e stampare il biglietto di ingresso. In caso di ripensamenti è fornita l'opzione di cancellazione della propria prenotazione.

  Il visitatore ha la possibilità di informarsi riguardo alle opere in esposizione mediante il catalogo opere e rimanere aggiornato riguardo alle notizie del museo iscrivendosi al servizio di newsletter.

2 Progettazione

2.1 Use case diagrams

L'applicativo ha 3 diversi attori: il curatore museale (Curator), l'ufficio prenotazioni (BookingOffice) e il visitatore (Visitor). Il curatore si occupa della gestione delle opere d'arte del museo e della loro disposizione all'interno di itinerari. L'ufficio prenotazioni si occupa dell'organizzazione delle visite guidate e i visitatori sono i fruitori dei servizi offerti. Lo schema dei casi d'uso, figura 1, è realizzato secondo lo standard UML mediante StarUML.

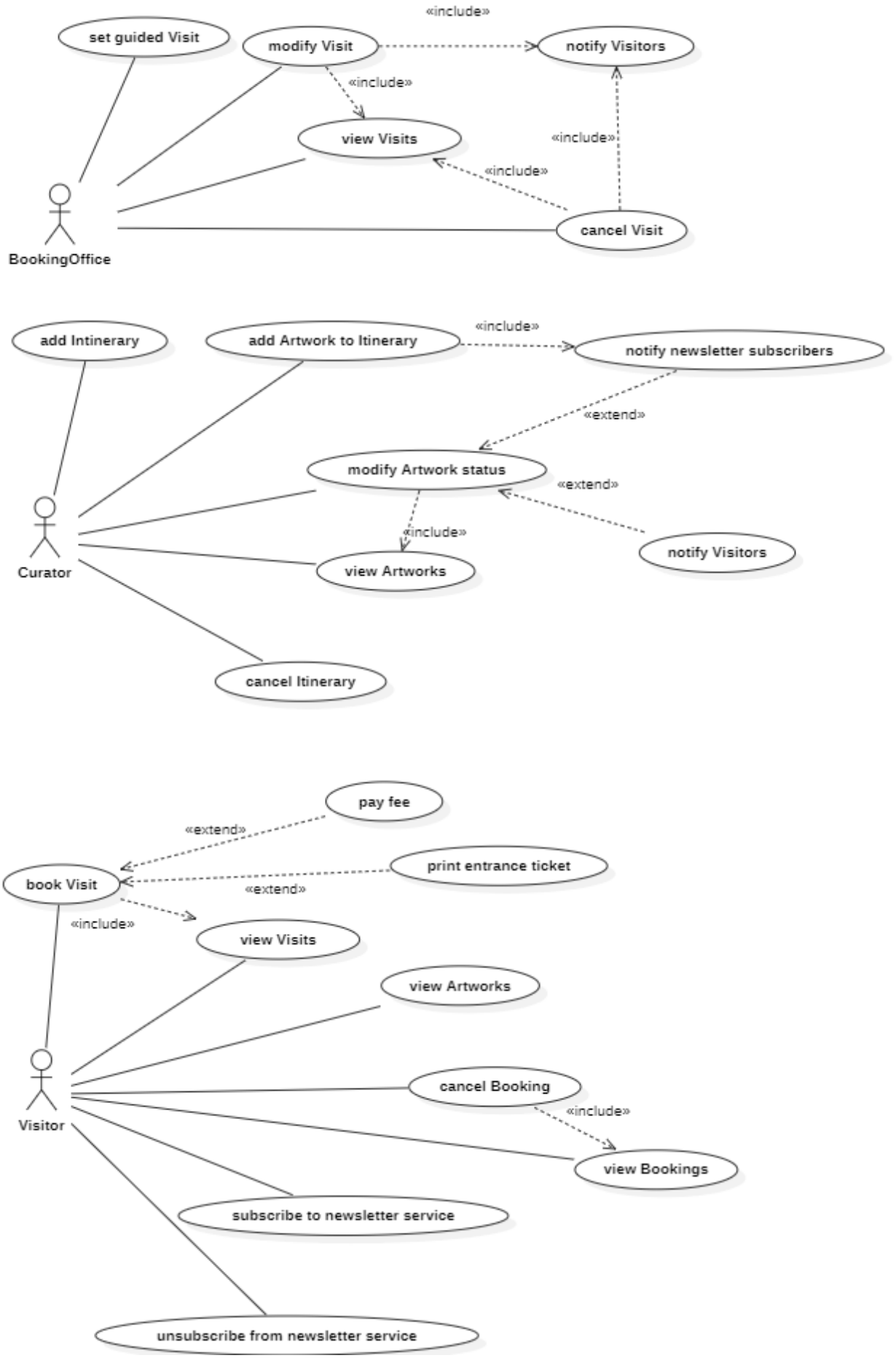

Figure 1: Use case diagrams

2.2 Use case templates

Abbiamo deciso di descrivere mediante template alcuni dei casi d'uso dell'applicazione: 2 per l'ufficio prenotazioni, 2 per i visitatori e 2 per il curatore.

UC-1	Set guided visit
Description	L'utente di tipo BookingOffice fissa una visita guidata stabilendo giorno, orario, itinerario, lingua, numero massimo di visitatori e prezzo del biglietto di ingresso.
Level	User goal
Main actors	BookingOffice
Basic course	1. Utente clicca su CREATE VISIT. 2. Il sistema mostra una tabella con i campi: date, time, itineraries, language, maxVisitors e ticketPrice da riempire (figura 2). 3. Una volta riempiti tutti i campi l'utente preme il pulsante CREATE. 4. Se tutti i valori inseriti sono corretti il sistema registra la visita.
Alternative course	4. Se i valori inseriti non hanno il formato richiesto o sono invalidi il sistema notifica l'utente (figura 2 in rosso).

UC-2	Cancel visit
Description	L'utente di tipo BookingOffice cancella la visita guidata scelta.
Level	User goal
Main actors	BookingOffice
Basic course	1. Utente clicca sul bottone CANCEL VISITS della button bar. 2. Il sistema mostra un elenco di tutte le visite registrate con le relative informazioni (figura 3). 3. L'utente seleziona la visita da cancellare e preme il pulsante DELETE. 4. Il sistema rimuove la visita e notifica via email tutti i visitatori interessati dalla cancellazione.
Issues	1. Come avviene il risarcimento del biglietto acquistato dal visitatore ?

UC-3	Book visit
Description	L'utente di tipo Visitor prenota un certo numero di posti alla visita, paga la prenotazione e stampa il biglietto di ingresso
Level	User goal
Main actors	Visitor
Basic course	1. Utente clicca sul bottone BOOK VISIT della button bar. 2. Il sistema mostra un elenco di tutte le visite registrate con le relative informazioni (figura 4). 3. L'utente seleziona la visita di interesse, inserisce il numero di posti che vuole prenotare e clicca sul pulsante CONFIRM. 4. Se il numero di posti richiesti non supera quello dei posti disponibili la prenotazione viene registrata dal sistema e viene mostrata all'utente una nuova pagina per il pagamento della visita. 5. L'utente riempie i campi richiesti per il pagamento e poi preme il pulsante PAY. 6. Se i parametri sono corretti il sistema registra l'avvenuto pagamento e mostra all'utente una pagina per la stampa del ticket di ingresso. 7. L'utente preme il pulsante PRINT TICKET 8. Il sistema stampa il biglietto di ingresso.
Alternative course	4. Se il numero di posti richiesti supera il numero di posti disponibili il sistema notifica l'utente con un messaggio di errore (figura 4 in rosso). 5. L'utente può scegliere di non pagare il biglietto online (verrà pagato di persona al museo) 6. Se i dati inseriti sono incorretti il sistema notifica l'utente con un messaggio di errore. 7. L'utente può scegliere di non stampare immediatamente il biglietto non premendo il pulsante.

UC-4	View artworks
Description	L'utente di tipo Visitor visualizza le opere del museo ed il loro stato
Level	User goal
Main actors	Visitor
Basic course	1. Utente clicca sul bottone VIEW CATALOG. 2. Il sistema mostra una tabella con le informazioni delle opere ed il loro stato (figura 5).

UC-5	Add artwork to itinerary
Description	L'utente di tipo Curator aggiunge un'opera ad un itinerario
Level	User goal
Main actors	Curator
Basic course	1. Utente clicca sul bottone ADD ARTWORK. 2. Il sistema mostra una tabella con i campi name, artist, itinerary, status e un campo opzionale aggiuntivo per lo stato da riempire (figura 6). 3. L'utente riempie i campi e preme il pulsante ADD. 4. Se i valori inseriti sono considerati ammissibili il sistema registra l'aggiunta di una nuova opera e notifica via email gli iscritti alla newsletter.
Alternative course	4. Se un valore inserito non è ammissibile il sistema notifica l'utente un messaggio di errore.

UC-6	Modify artwork status
Description	L'utente di tipo Curator modifica lo stato di una delle opere
Level	User goal
Main actors	Curator
Basic course	1. Utente clicca sul bottone MODIFY ARTWORKS STATUS della button bar. 2. Il sistema mostra una tabella con le opere ed il loro stato e sotto un data grid per inserire le informazioni relative al nuovo stato (figura 7). 3. L'utente seleziona l'opera il cui stato vuole modificare, riempie i campi del data grid e preme il pulsante CONFIRM. 4. Se le informazioni inserite sono diverse dalle originali il sistema registra le modifiche.
Alternative course	4. Se le informazioni aggiunte sono le stesse di quelle originali il sistema non compie aggiornamenti. Se, invece, sono invalide il sistema notifca l'utente con un messaggio di errore.

2.3 Mockups

Abbiamo creato dei mockups delle interfacce web dell'applicazione per i differenti utenti. La realizzazione è avvenuta mediante le funzionalità offerte da Balsamiq.

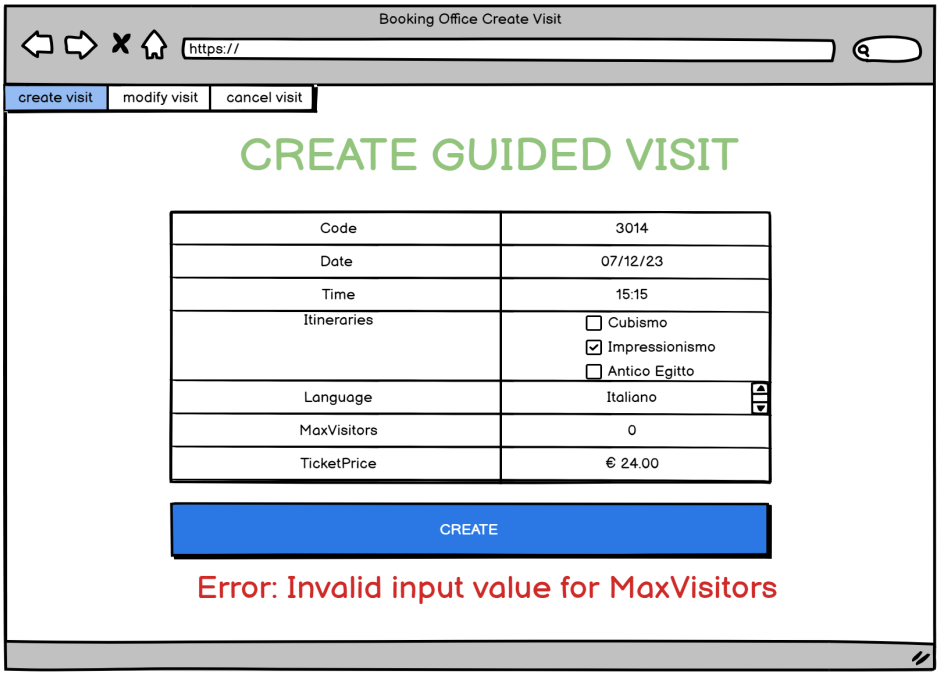

Figure 2: Interfaccia creazione visita

Figure 3: Interfaccia annullamento visita

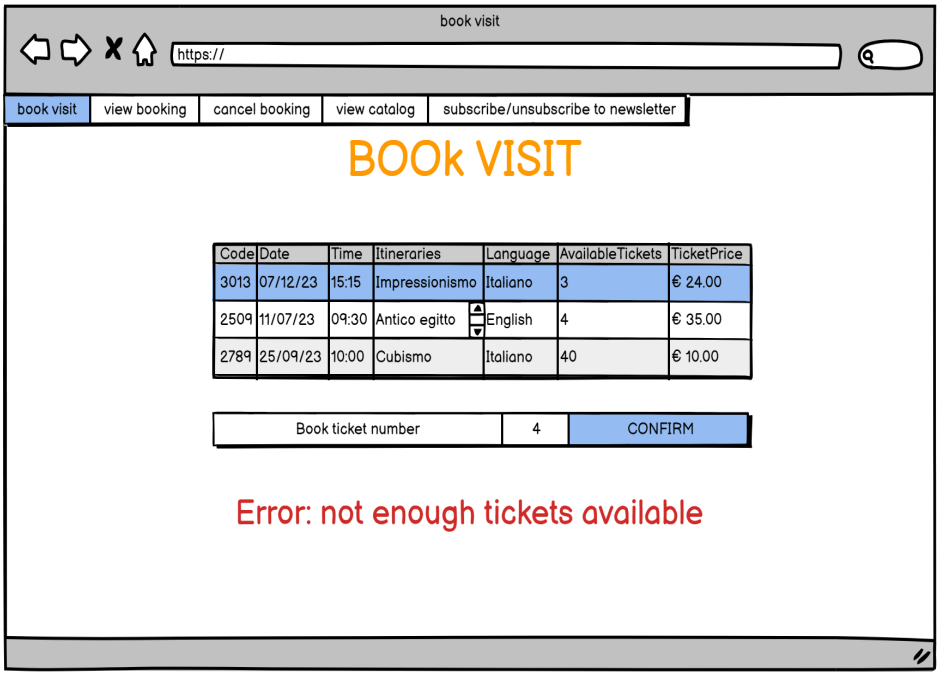

Figure 4: Interfaccia prenotazione visita

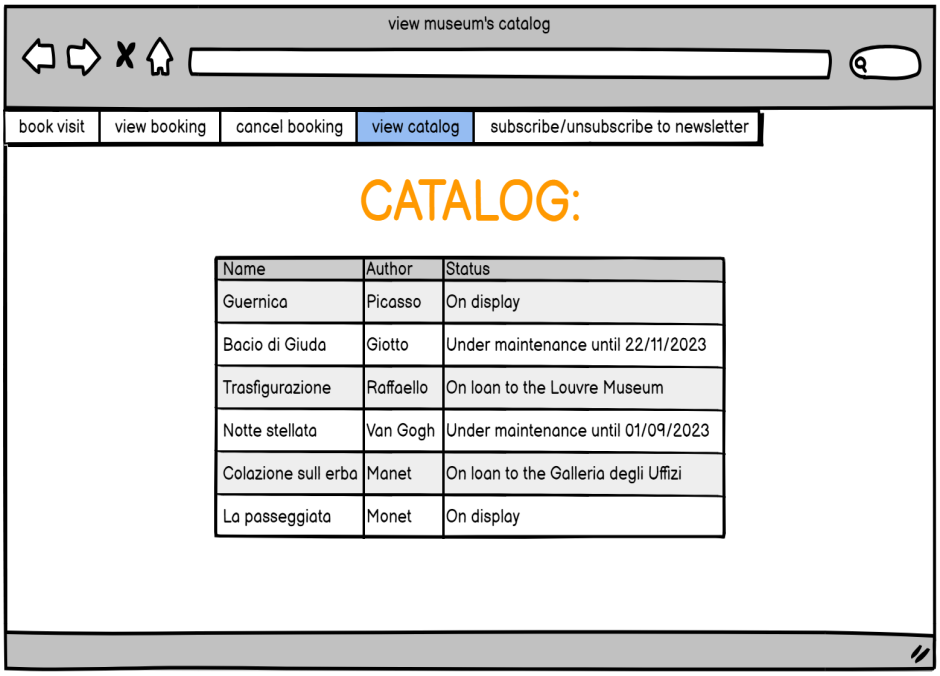

Figure 5: Interfaccia catalogo

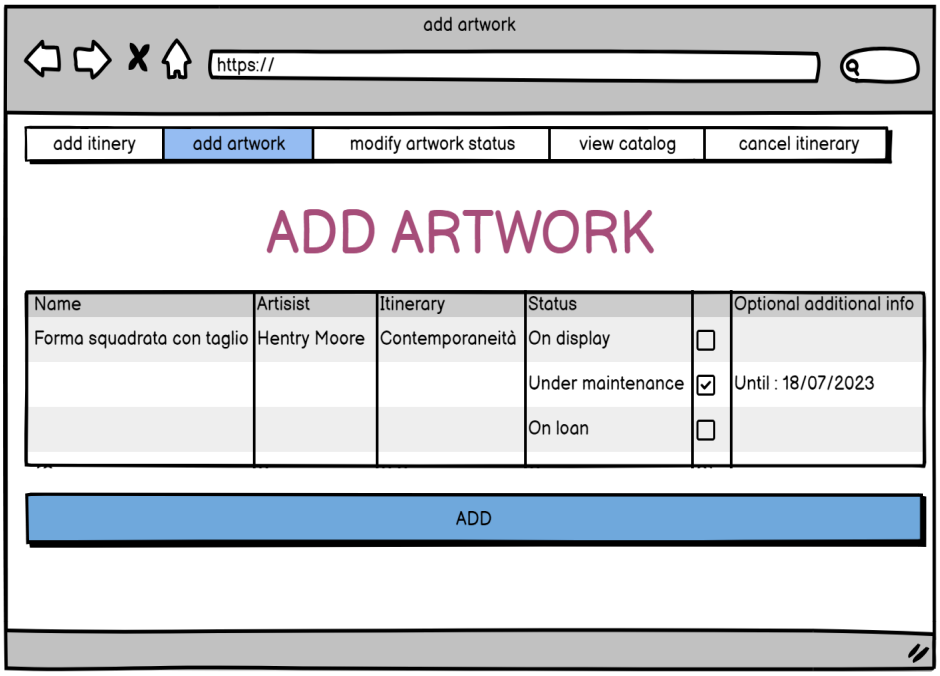

Figure 6: Interfaccia aggiunta opera

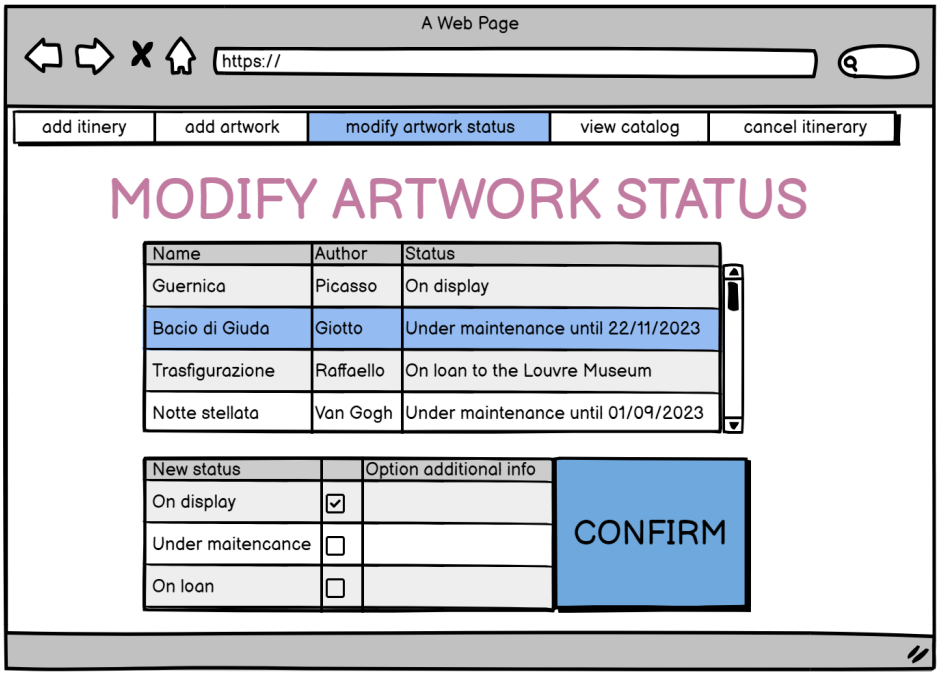

Figure 7: Interfaccia modifica stato dell'opera

2.4 Class diagram

Lo schema dei package e delle classi, figura 8, è realizzato con StarUML e segue le linee guida dello standard UML. Il programma è articolato in 3 package: business logic, domain model e object-relational mapping (orm) e la gestione dei dati persistenti è affidata al relational database managment system (rdbms).

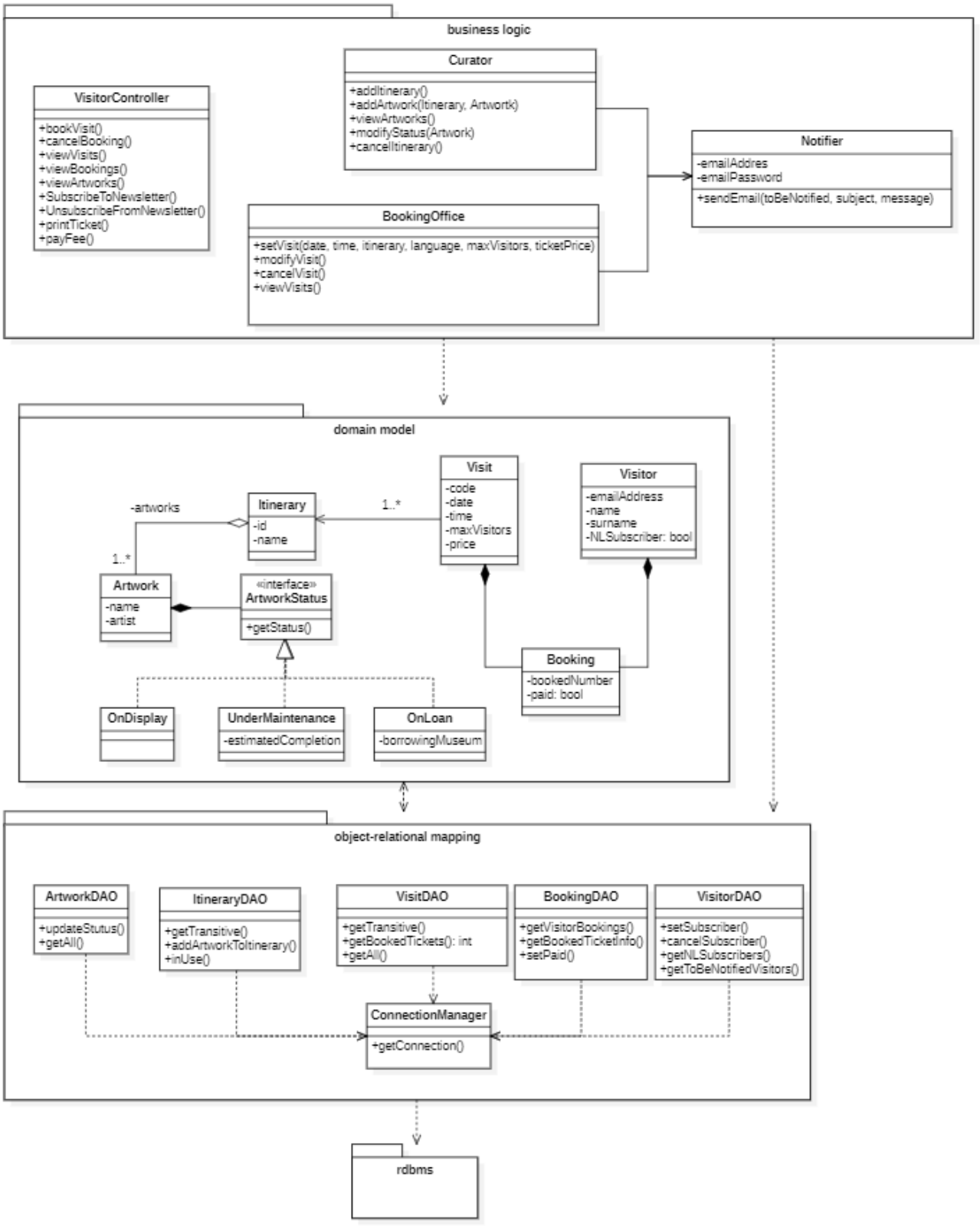

Figure 8: Class diagram

3 Implementazione

Abbiamo realizzato il programma in Java ed abbiamo organizzato il codice come da figura 14. Il repository è visibile al seguente link: `https://github.com/Gio-Formichella/ElaboratoSWE`

3.1 Domain model

Il domain model, figura 9, è il package contenente le entità museali rappresentate nel software. Si articola nelle seguenti classi:

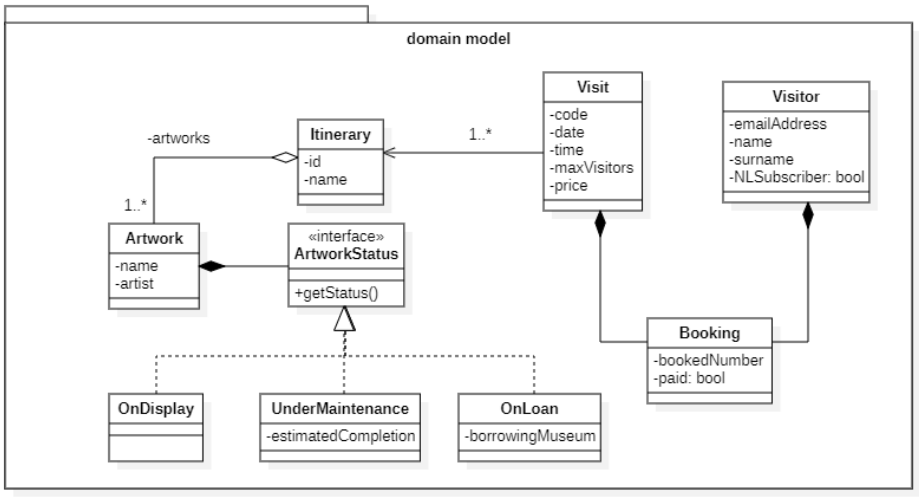

Figure 9: Domain model

3.1.1 Artwork

Rappresenta un opera d'arte del museo. Ha attributi per il nome e l'autore dell'opera ed ha un riferimento ad un oggetto di tipo ArtworkStatus che ne modella lo stato.

3.1.2 ArtworkStatus, OnDisplay, UnderMaintenance e OnLoan

ArtworkStatus è l'intefaccia che le classi OnDisplay, UnderMaintenance e OnLoan implementano per rappresentare rispettivamente lo stato "in mostra", "in manutenzione" ed "in prestito". I metodi presenti nell'interfaccia sono getStatus(), che ritorna lo stato dell'opera, e copy(), che ritorna una copia dell'oggetto.

Abbiamo scelto una soluzione di questo tipo invece che una semplice enum class per rappresentare lo stato perché le classi UnderMaintenance e OnLoan hanno delle informazioni aggiuntive, rispettivamente: il giorno stimato di completamento del restauro e il museo o collezione in cui è attualmente in esposizione l'opera, informazioni non rappresentabili con una enum class.

3.1.3 Itinerary

È la classe che rappresenta gli itinerari organizzati dal curatore museale. Ciascun itinerario ha un nome ed un codice identificativo. La classe ha inoltre un ArrayList contenente tutte le opere d'arte associate all'itinerario.

3.1.4 Visit

La classe Visit modella le visite organizzare dall'ufficio prenotazioni. Ciascuna visita ha un codice identificativo, la data e l'ora in cui avrà inizio, la lingua nella quale verrà tenuta la visita, il numero massimo di visitatori che possono partecipare, il prezzo del biglietto di ingresso ed una lista degli itinerari presenti nel tour.

3.1.5 Visitor

Mantiene le informazioni del visitatore. Ha attributi per il nome, cognome, indirizzo di posta elettronica del visitatore ed un attributo booleano che segnala l'iscrizione al servizio di newsletter.

3.1.6 Booking

È la classe della prenotazione, associa un Visitor ad una Visit, ha un codice univoco ed un attributo booleano per indicare se il biglietto di ingresso è già stato pagato o meno.

3.2 Business logic

Il package business logic contiene le classi controller, una per ogni attore, che implementano gli use case sfruttando le classi del domain model ed i DAO dell'orm. Inoltre è presente una classe Notifier che si occupa di spedire le email richieste dal Curator e dal BookingOffice.

Per maggiore semplicità abbiamo evitato di affidare la scelta dei codici degli oggetti al DBMS, con conseguente uso di Universally Unique Identifiers, e, invece, abbiamo optato per mantenerla come responsabilità dell'utente.

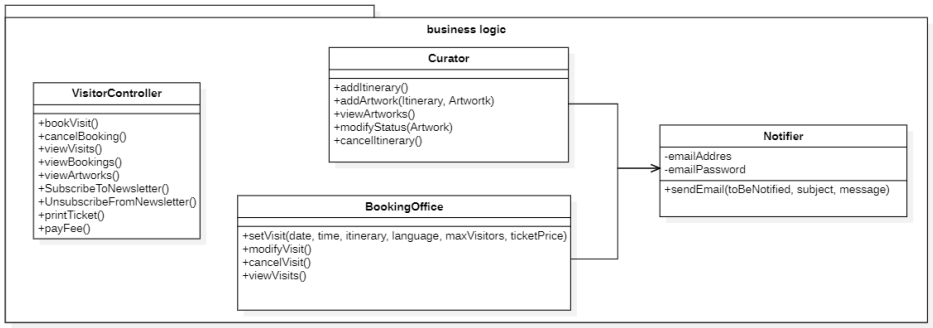

Figure 10: Business logic

3.2.1 Curator

La classe Curator ha operazioni proprie del curatore museale. Il metodo addItinerary() aggiunge un itinerario alla collezione mentre il metodo addArtwork() aggiungere un'opera d'arte ad un itinerario e pubblicizza l'aggiunta con una newsletter proveniente dall'indirizzo email museale. Il metodo viewArtworks() permette di visionare il catalogo opere e modifyStatus() modifica lo stato di una delle opere selezionate con successiva notifica dei visitatori interessati. Infine, cancelItinerary() rimuove l'itinerario stabilito se non è presente in alcuna visita, in caso contrario il programma segnala al curatore un messaggio di errore.

3.2.2 BookingOffice

La classe BookingOffice presenta metodi per le operazioni dell'ufficio prenotazioni del museo. Il metodo setVisit() crea una nuova visita guidata mentre il metodo cancelVisit() elimina una visita dal database. La creazione di una visita ha dei vincoli (ad esempio maxVisitors deve essere 0), se violati il metodo lancia un'eccezione. La funzione modifyVisit() consente la modifica di una visita senza causarne la cancellazione. Sia nel caso della cancellazione che della modifica vengono notificati tramite servizio di posta elettronica i visitatori aventi una prenotazione per la visita selezionata. Infine il metodo viewVisits() permette la visualizzazione di tutte le visite organizzate dal museo.

3.2.3 VisitorController

Ha le operazioni del visitatore. Il metodo bookVisit() consente di prenotare un certo numero di posti in una visita, se però il numero di posti è però insufficiente il metodo lancia un'eccezione. Il metodo payFee() consente di pagare il prezzo del biglietto di ingresso mentre il metodo cancelBooking() permette di cancellare una delle proprie prenotazioni. Il visitatore può inoltre stampare il biglietto di ingresso con printTicket(). I metodi per la visualizzazione delle visite e delle prenotazioni del visitatore sono rispettivamente viewVisits() e viewBookings().

3.2.4 Notifier

La classe Notifier è un Singleton (simula un bean) e ha come unico metodo: sendEmail. Il metodo riceve come input: l'indirizzo del mittente, la lista degli indirizzi dei riceventi, l'oggetto e il messaggio da inviare, e crea una sessione e invia le email con i dati forniti e con il logo del museo.

3.3 Orm

Il package object-relational mapping contiene le classi DAO che, interagendo con il RDBMS, offrono all'applicazione servizi inerenti alla persistenza dei dati.

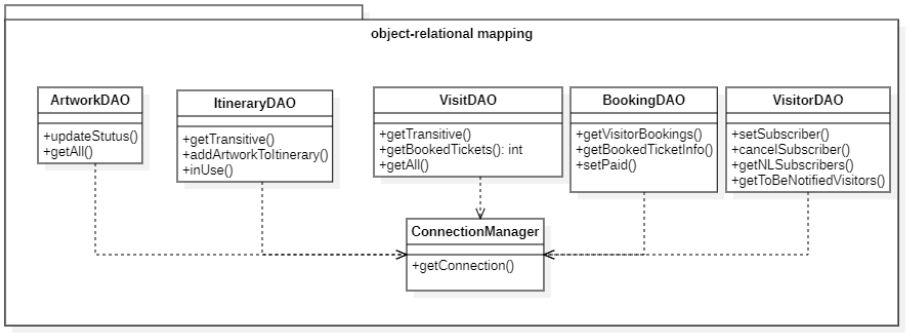

Figure 11: Object-relational mapping

3.3.1 ConnectionManager

La classe si occupa di fornire la connessione al database alle classi dao che la richiedono mediante il metodo statico getConnection(). Il metodo passa al chiamante un oggetto di tipo Connection garantendo l'unicità della connessione.

3.3.2 ArtworkDAO

La classe ArtoworkDAO si occupa di istanziare e memorizzare oggetti di tipo Artwork. A questo fine ha metodi di get, insert e delete per rispettivamente prelevare, inserire e rimuovere informazioni relative ad un'opera dal database. Il metodo getAll() permette di prelevare tutte le opere del museo e il metodo modifyStatus() aggiorna l'informazione dello status di un'opera.

3.3.3 ItineraryDAO

La classe, oltre ai metodi canonici di insert e delete, ha un metodo addArtworkToItinerary(Artwork, Itinerary) che crea un'associazione tra l'opera e l'itinerario. Il metodo getTransitive() preleva ed istanzia dal database l'oggeto Itinerary richiesto e nel farlo può dover istanziare 0 o più oggetti Artwork, per questo è transitivo.

3.3.4 VisitDAO

Oltre a insert e delete, è presente un metodo update per la modifiche dettate dal metodo modifyVisit() dalla classe BookingOffice. Il metodo getTransitive() istanzia un oggetto di tipo Visit e quindi dovrà istanziare anche uno o più oggetti di tipo Itinerary, il metodo ha quindi una transitività.

3.3.5 BookingDAO

Si occupa di creare e memorizzare oggetti di tipo Booking. La classe ha un metodo getVisitorBookings() che riceve in input un oggeto Visitor e restituisce la lista di Booking associata al visitatore. Il metodo getBookingVisit(), invece, restituisce le informazioni sulla prenotazione, visitatore e visita necessarie per la stampa del biglietto di ingresso. Il metodo setPaid(code) semplicemente imposta a true la variabile booleana del pagamento della tupla nel database con codice corrispondente.

3.3.6 VisitorDAO

La classe VisitorDAO ha un metodo getNLSubscribers() che ritorna la lista dei visitatori iscritti alla newsletter selezionando dal database le tuple con campo newsletter settato a true. Il metodo getToBeNotifiedVisitors() è stato sovraccaricato, può ricevere come input o un Artwork o un Visit e restituisce una lista di Visitor le cui prenotazioni riguardano l'opera o la visita selezionata. I metodi setSubscriber() e cancelSubscriber() rispettivamente scrivono true e false nel campo newsletter della tupla associata al Visitor di input.

3.4 Database

Lo schema Entity–relationship, riportato in figura 12, è stato realizzato con StarUML e descrive le entità del dominio museale e le relazioni tra queste. Dallo schema ER è stato tradotto lo schema logico descritto in figura 13. Per la gestione del database abbiamo impiegato PostreSQL, un database managment system gratuito e open-source anche conosciuto come Postgres.

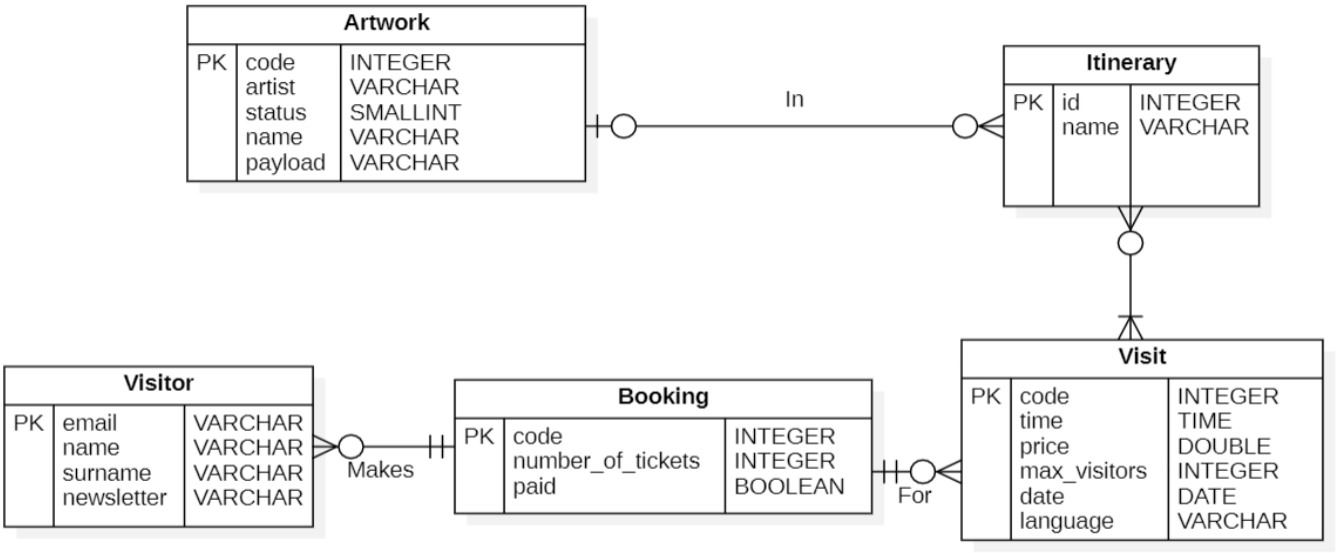

Figure 12: ER diagram

```
Artwork(PK(code), name, artist, status, payload)

Itinerary(PK(id), name)

Artwork_Itinerary(PK(artwork, itinerary)), FK(artwork) REF Artwork,
        FK(itinerary) REF Itinerary

Visitor(PK(email), name, surname, newsletter)

Visit(PK(code), date, time, language, max_visitors, price)

Visit_Itinerary(PK(visit, itinerary)), FK(visit) REF Visit,
        FK(itinerary) REF Itinerary

Booking(PK(code), visitor, visit, number_of_tickets, paid),
        FK(visitor) REF Visitor, FK(visit) REF Visit
```

Figure 13: Schema logico del database

3.5 lib & imgs

Nella cartella lib abbiamo inserito le librerie e le API utilizzate. Per l'invio di email abbiamo usato le API javax.mail e activation, per la connessione al database il driver postgresql-42.6.0 e, infine, per il testing ed il code converage abbiamo usato rispettivamente JUnit e JaCoCo. Nella directory imgs, invece, abbiamo inserito il logo del museo utilizzato nelle email.

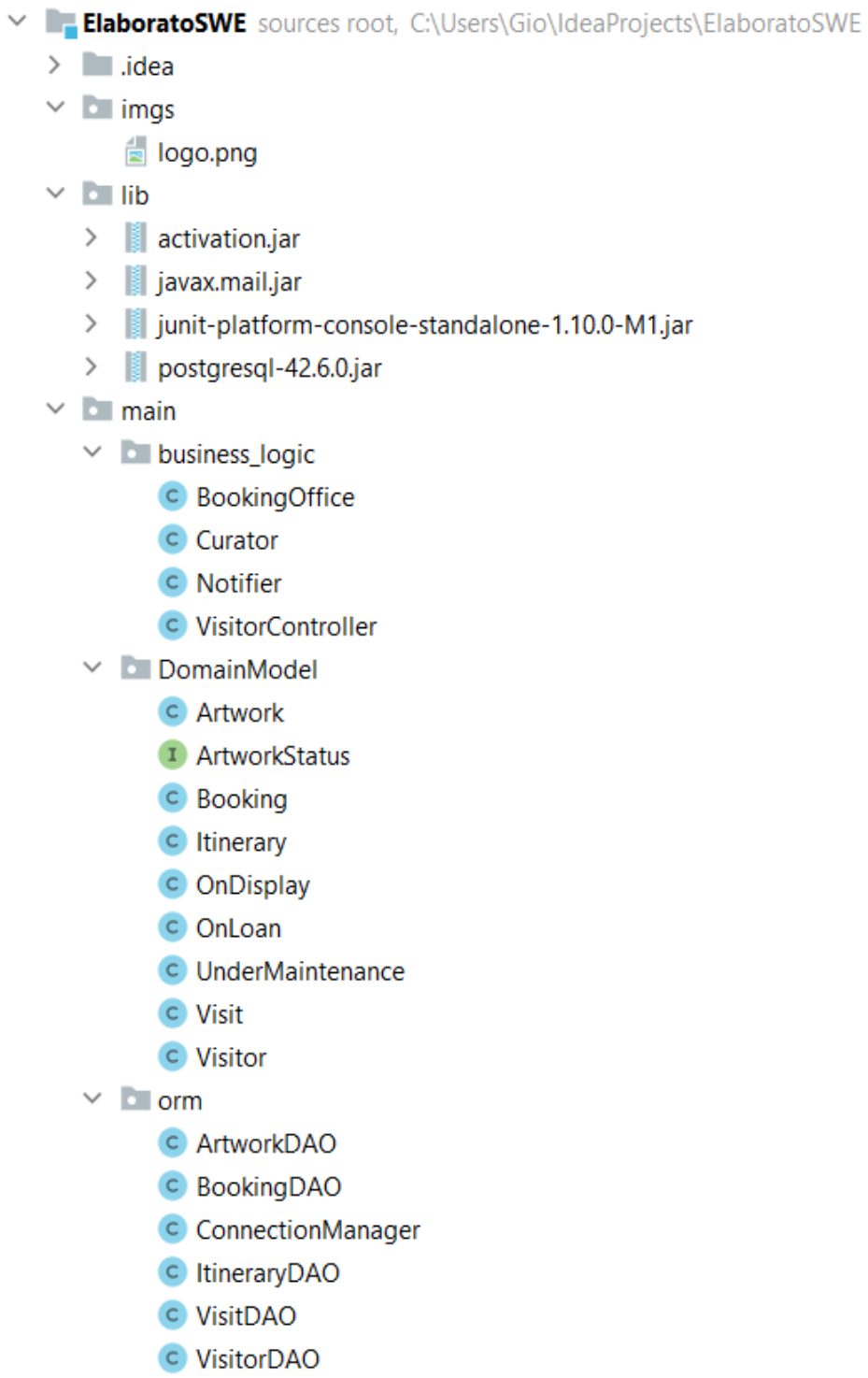

Figure 14: Organizzazione codice

4 Testing

Per il testing del codice abbiamo utilizzato il framework JUnit e abbiamo scritto test di unità per tutte le classi implementate ad eccezione di quelle del domain model trattandosi di classi POJO. I test sono organizzati come in figura 15.

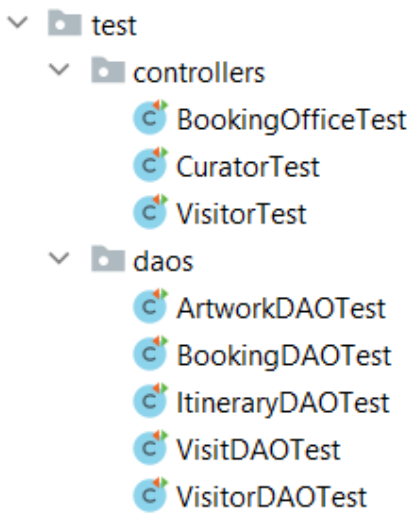

Figure 15: Organizzazione dei test

4.1 Testing controllers

4.1.1 CuratorTest

In questa classe vengono testati i metodi della classe Curator della business logic. I test relativi di addItinerary() e addArtwork() verificano che vengano aggiunti al database nuovi itinerari e nuove opere con stati diversi senza perdita di informazione. Il test di modifyStatus() inserisce un nuovo Artwork nel database e verifica la corretta avvenuta dei passaggi di stato tra OnDisplay, UnderMaintenance e OnLoan. Il test di viewArtowrks(), invece, verifica che venga ritornata dal metodo la lista di Artwork del museo. Infine, il test cancelItinerary() verifica che nel caso in cui l'itinerario non appartenga ad alcuna visita la tupla dell'Itinerary sia rimossa mentre nel caso in cui l'itinerario appartenga ad almeno una visita venga lanciata un'eccezione con il corretto messaggio di errore.

Non è stato verificato nei test il corretto invio delle email ai visitatori poiché non realizzabile con JUnit. La correttezza del metodo sendEmail di Notifier è stato verificata a livello empirico, vedi figura 16.

Figure 16: Email dal museo

4.1.2 BookingOfficeTest

In BookingOfficeTest sono presenti i test relativi ai metodi di BookingOffice. Il test di setVisit() verifica il corretto inserimento di una nuova visita nel database nel caso in cui i parametri forniti siano validi, nel caso in cui sono invalidi verifica che venga lanciata un'eccezione con corrispondente messaggio di errore. Il test associato a cancelVisit() verifica che la tupla dell'itinerario corrispondente sia rimossa con successo. Inoltre si testa che modifyVisit() si accerta che i parametri della visita siano modificati con successo e che venga lanciata un'eccezione se i parametri sono non ammissibili. Per il metodo viewVisits() ci si accerta solo che venga restituito un array non vuoto. Sono anche presenti test che verificano la modifica o l'inserimento di una visita sbagliando il dominio del prezzo o del numero di visitatori, accertandosi che vengano lanciate le eccezioni giuste.

Come per CuratorTest la verifica del corretto invio delle email è verificato solo a livello empirico.

4.1.3 VisitorControllerTest

In questa classe abbiamo verificato il corretto funzionamento delle operazioni relative alle prenotazioni. Abbiamo verificato che bookVisit() crei correttamente una prenotazione, payFee() setti a true la variabile del pagamento dell'ingresso nella tupla corrispondente del database e che cancelVisit() rimuova la prenotazione selezionata dall'utente. Sono testate anche le operazioni relative al servizio di newsletter, i test subscribeToNewsletter() e unsubscribeFromNewsletter() verificano che l'utente sia effettivamente iscritto e disiscritto dalla lista dei subscriber verificando il valore del campo newsletter della tupla della tabella Visitor. Infine è verificato che getBookingVisit restituisce le informazioni necessarie alla stampa del biglietto di ingresso.

4.2 Testing DAOs

I test relativi ai metodi dei DAO hanno una struttura similare. Sono composti da un set up con inserimento di tuple nel DB che saranno poi interessate dal metodo testato, fanno eccezione i test di insert che questa fase non la richiedono. A seguire è eseguito il metodo sotto test e poi vengono fatte delle asserzioni che verificano il corretto funzionamento e quindi la correttezza della query sql. Terminata la porzione del codice di testing, è presente un tear down dove vengono rimosse le tuple originariamente inserite. Un esempio è il test del metodo get del ArtworkDAO in figura 17.

La struttura dei test non è ottimale in quanto non sono self-contained: il codice di testing di un metodo include altri metodi potenzialmente non funzionanti come da aspettativa. Per risolvere il problema abbiamo

provato ad utilizzare come in-memory database H2 con conseguente impiego di Maven ma, dati i problemi riscontrati dalla conseguente ristrutturazione e riconfigurazione (file pom.xml) e dalla scarsità di tempo rimanente alla consegna, abbiamo mantenuto i test nel loro stato corrente.

Per questa ragione se uno qualsiasi dei metodi smette di funzionare allora molti test falliranno ma poiché tutti i test terminano con esito positivo, figura 18, segue che tutti i metodi sono corretti.

```java
@Test
void get() {
    ArtworkDAO dao = new ArtworkDAO();
    Artwork a = new Artwork( code: 1,  name: "AnArtwork",  author: "Gio", new OnDisplay());
    try {
        dao.insert(a);
        Artwork retrieved = dao.get(a.getCode());
        assertEquals(retrieved.getCode(), a.getCode());
        assertEquals(retrieved.getName(), a.getName());
        assertEquals(retrieved.getAuthor(), a.getAuthor());
        assertEquals(retrieved.getStatus(), a.getStatus());
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            dao.delete(a);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Figure 17: Test metodo get di ArtworkDAO in ArtworkDAOTest

4.3 Esiti dei test

Gli esiti dei test sono in figura 18. Il code coverage raggiunto è del 94%.

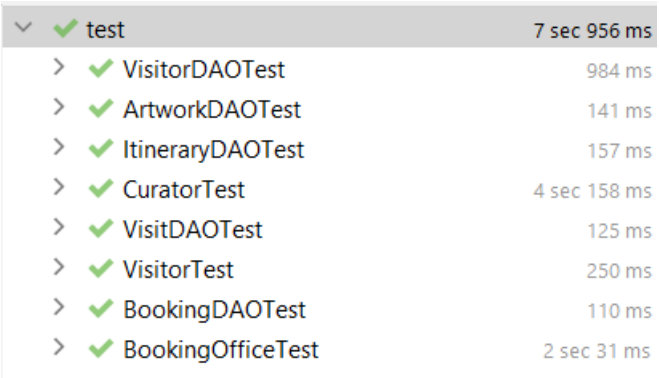

Figure 18: Risultati dei test