Cat Café Management System

Matteo Amato and Sara Giancarli

July 8, 2025

Contents

1 Software Requirements Specification

1.1 Introduction

1.1.1 Purpose of the Document

This document specifies the functional and non-functional requirements for the **Cat Café Management System** , a Java application that supports day-to-day operations of a themed cat café, including table reservations, cat adoptions and staff administration.

1.1.2 System Overview

The Cat Café Management System is packaged as an executable JAR built with Oracle JDK SE 24. It exposes four primary capabilities:

UC1 **Reserve Table** – user can book seating at specific dates and times.

UC2 **Adopt Cat** – user can submit adoption requests for resident cats.

UC3 **Review** – the user can leave a review of the service.

UC4 **Manage Request** – staff can approve or reject customer request.

UC5 **Manage Cats** – staff can create, edit and archive cat profiles.

UC6 **Manage Activity** – staff can create, edit and archive activity.

1.1.3 Hardware and Software Requirements

Hardware

- **Processor**: modern CPU capable of running Java byte-code.
- **Memory**: $\geq$ 4 GB RAM.
- **Storage**: at least 500 MB free disk space for application files and offline data.

Software

- **Oracle JDK SE 24** — required runtime.
- **MySQL Server 8.x** — persistent storage for cats, users and bookings.
- **Development IDE** (optional) — Eclipse 2025-06 or IntelliJ IDEA 2025.1.

1.1.4 Analysis of Similar Platforms

There are already several platforms in the bar/restaurant-booking and remote-adoption sectors, such as `theFork.it` and `Ilcercapadrone.it`. However, they have some limitations:

- `theFork.it`: it offers a wide range of venues, but the user interface can be overly complex for some users.
- `Ilcercapadrone.it`: it allows users to adopt a cat through a catalogue, but its interface may not be usable on all devices.

1.2 User Stories

Matteo

US-1
As a user, I want to reserve a table, so that I can better plan my day.

US-2
As a user, I want to receive a confirmation email for my reservation requests, so that I know they succeeded.

US-3
As a staff member, I want to manage customer requests, so that I can reject non-compliant ones.

Sara

US-1
As a staff member, I want to update the cat gallery, so that it reflects the current cats.

US-2
As a user, I want to read information about the cats, so that I can learn about their lifestyle and personality.

US-3
As a staff member, I want to write cat descriptions, so that people are more likely to adopt.

1.3 Functional Requirements

Matteo

FR-1
The system shall send a confirmation email to the customer who booked a table.

FR-2
The system shall require a successful login before a user can create a table reservation.

FR-3
The system shall enable staff members to view all reservations and inspect their details

Sara

FR-1
The system shall provide a list of adoptable cats currently present in the café.

FR-2
The system shall send an email confirming that an adoption request has been registered.

FR-3
The system shall prevent concurrent adoption requests for the same cat.

1.4 Use Case Diagram

The diagram represent the main functionality of Cat Cafe

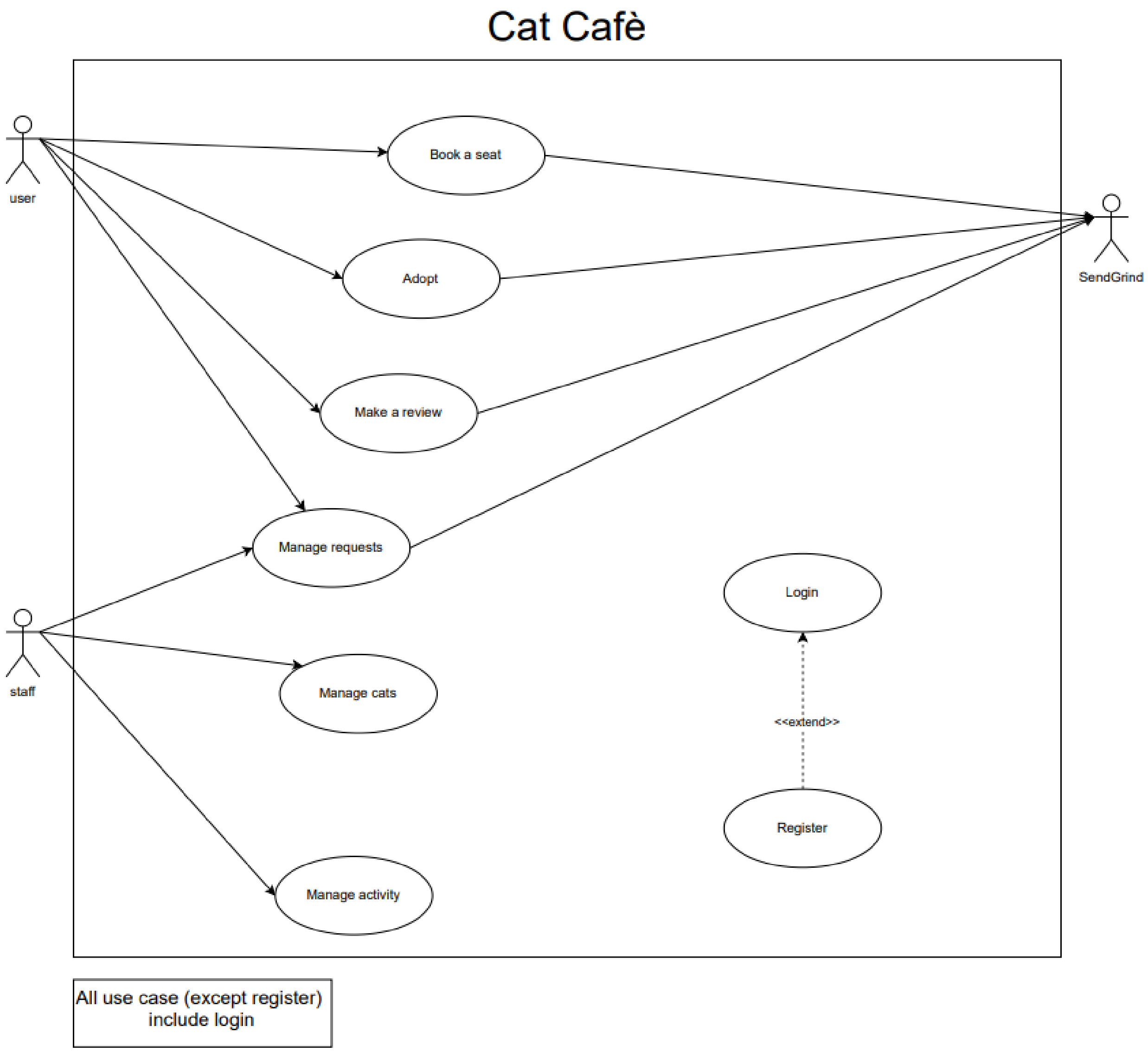


Figure 1: use case diagram

1.5 Internal Steps

1.5.1 Book A Seat

1. The user logs in.
2. The system shows the home page with navigation buttons.
3. The user selects *Prenota un tavolo.*
4. The system displays the reservation form.
5. The user fills in date, time and table size.
6. The system validates the entered data and checks seat availability for that date.
7. The system shows a summary of entered data.
8. The user confirms.
9. The system stores the booking in the database

Extension
6a. **Not enough seats available for that date**: the system warns the user that number of seats is not available for that date return to Step 5.
6b. **Data validation failed**: the system detect that one or more fields contain errors(e.g. date in incorrect format, empty field), display an error message and invites the user to correct the data, return to Step 5.
7a.**Cancel Booking** : The user cancels the booking, return to Step 2
8a.**Error saving to DB** : the system encounters a problem saving the booking, notifies the user ad return to Step 2

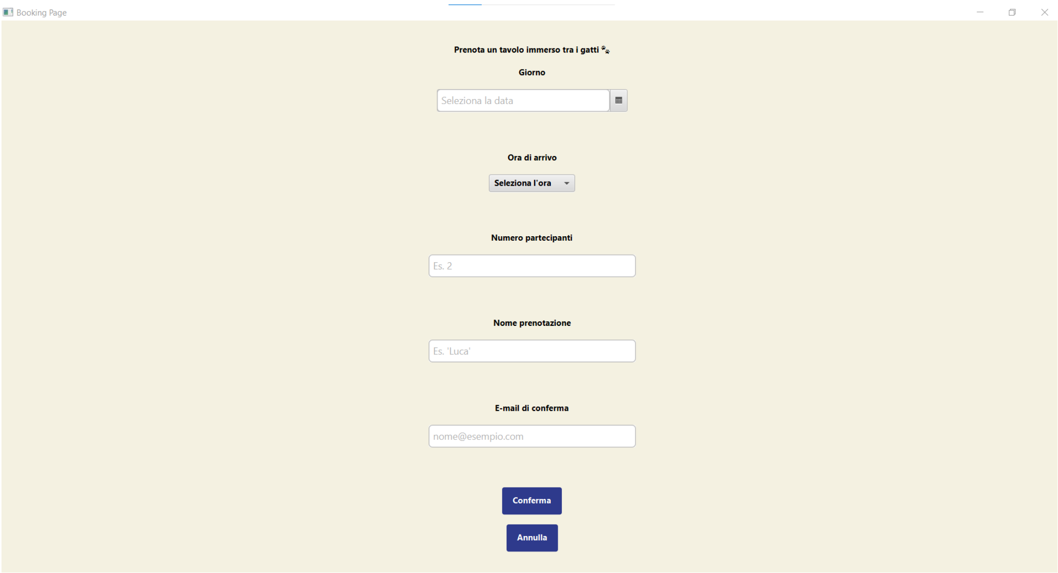
1.5.2 Internal steps Adopt Cat

1. The user logs in.
2. The system shows the home page.
3. The user selects *Adotta.*
4. The system shows the adoption form and cat list.
5. The user chooses a cat and enters personal data.
6. The system ensures no duplicate adoption request exists for that cat and user.
7. The system shows a summary of the request.
8. The user confirms.
9. The system saves the request and emails a confirmation.

Extension
6a. **Duplicate request**the system detect that an adoption with that cat and user has already been requested return to Step 4.
7a.**Cancel Adoption** : The user cancels the adoption, return to Step 2
8a.**Error saving to DB** : the system encounters a problem saving the booking, notifies the user ad return to Step 2

2 Storyboard

This section show the principal user views .



Booking Page

Prenota un tavolo immerso tra i gatti 🐾

Giorno

Seleziona la data

Ora di arrivo

Seleziona l'ora

Numero partecipanti

Es. 2

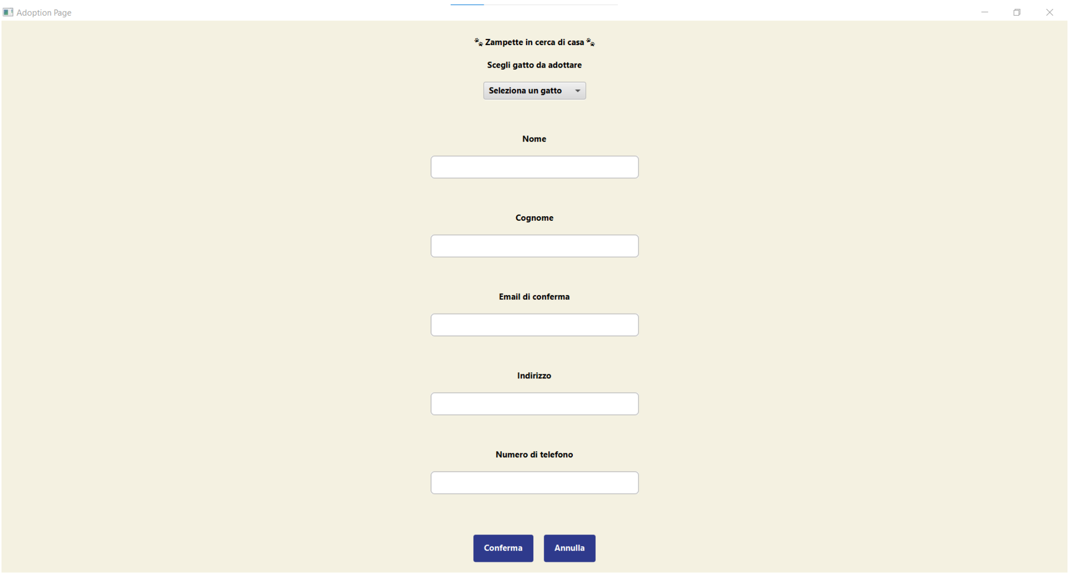
Nome prenotazione

Es. 'Luca'

E-mail di conferma

nome@esempio.com

Figure 2: Book seat view.



Adoption Page

🐾 Zampette in cerca di casa 🐾

Scegli gatto da adottare

Seleziona un gatto

Nome

Cognome

Email di conferma

Indirizzo

Numero di telefono

Figure 3: Adopt cat view

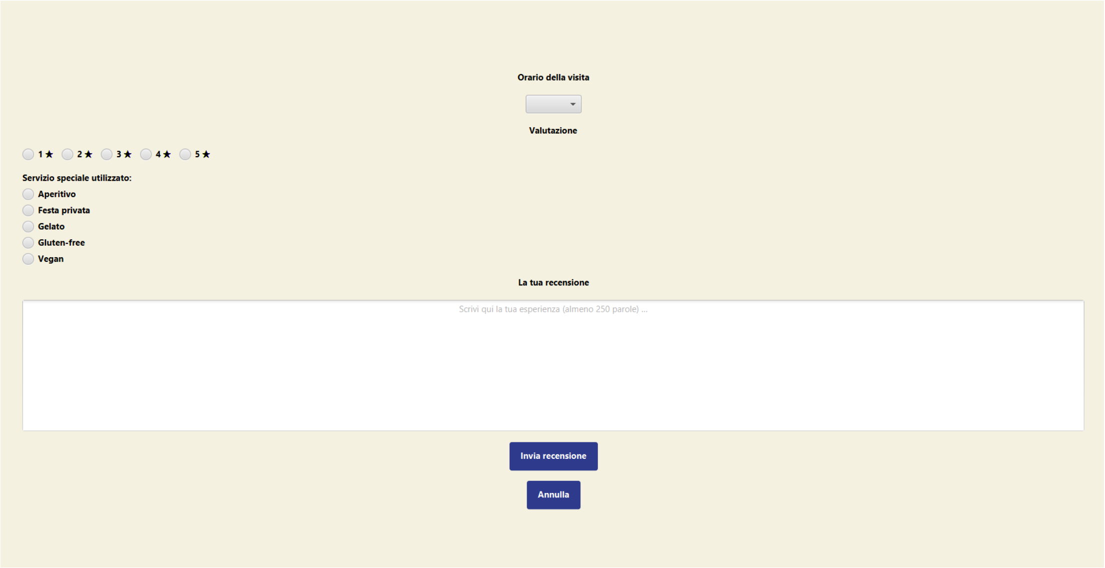
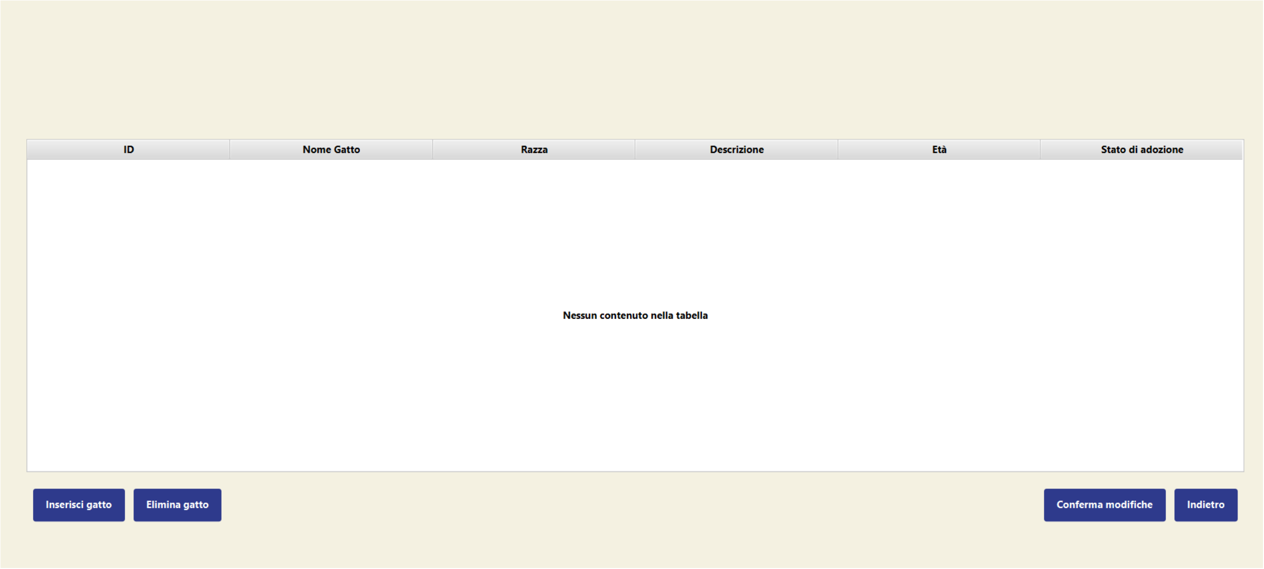


Figure 4: Review view



ID	Nome Gatto	Razza	Descrizione	Età	Stato di adozione

Nessun contenuto nella tabella

Figure 5: ManageCat view

3 Class Diagram

3.1 Vopc-BCE

The class diagram of the cat cafe Management System is organised into a set of well-defined packages:

Entity

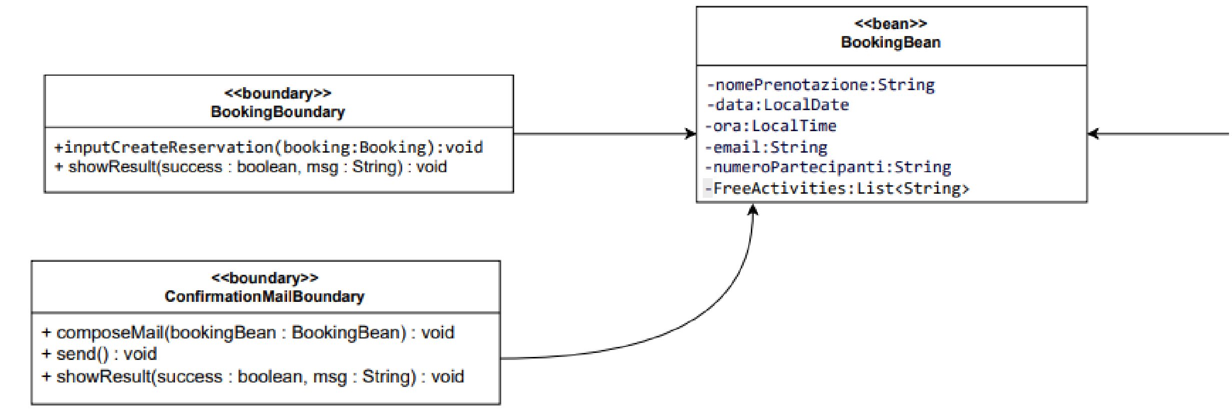
Contains the domain classes — e.g., `User`, and `Booking`. These objects encapsulate state and expose standard getter/setter methods.

Control

The *BookingController* drives business logic and mediates between entity and user interface. In the booking use case, the class `BookingService` coordinates the table–reservation workflow and `RequestAdoptionController` coordinates the adoption workflow.

Boundary

Boundary classes are the classes that mediate between the system and the environment



<<boundary>>
BookingBoundary
+inputCreateReservation(booking:Booking):void
+ showResult(success : boolean, msg : String) : void
<<boundary>>
ConfirmationMailBoundary
+ composeMail(bookingBean : BookingBean) : void
+ send() : void
+ showResult(success : boolean, msg : String) : void
<<bean>>
BookingBean
-nomePrenotazione:String
-data:LocalDate
-ora:LocalTime
-email:String
-numeroPartecipanti:String
-FreeActivities:List<String>

Figure 6: Book a Seat BCE Boundary and beans

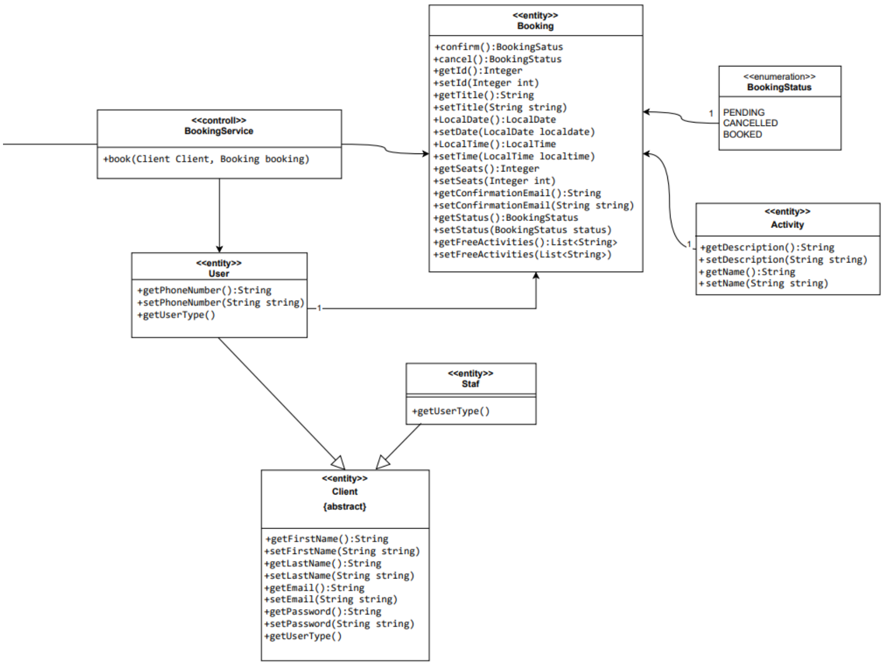


Figure 7: Book a Seat BCE Controller and entity

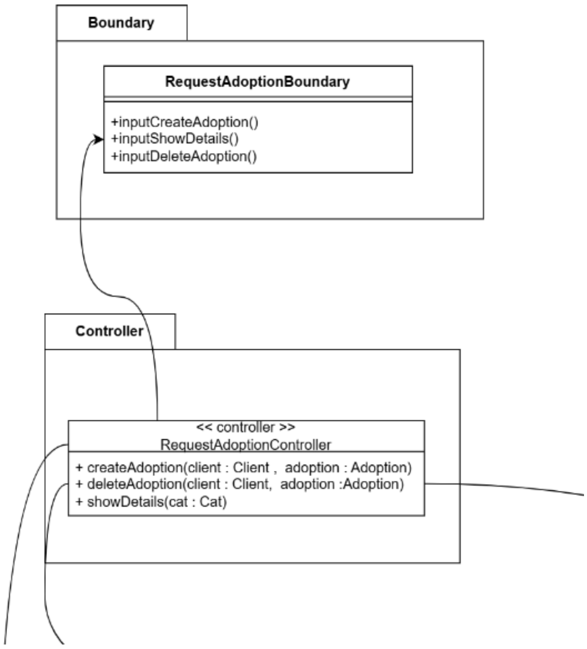


Figure 8: Adopt BCE Controller, Boundary and beans

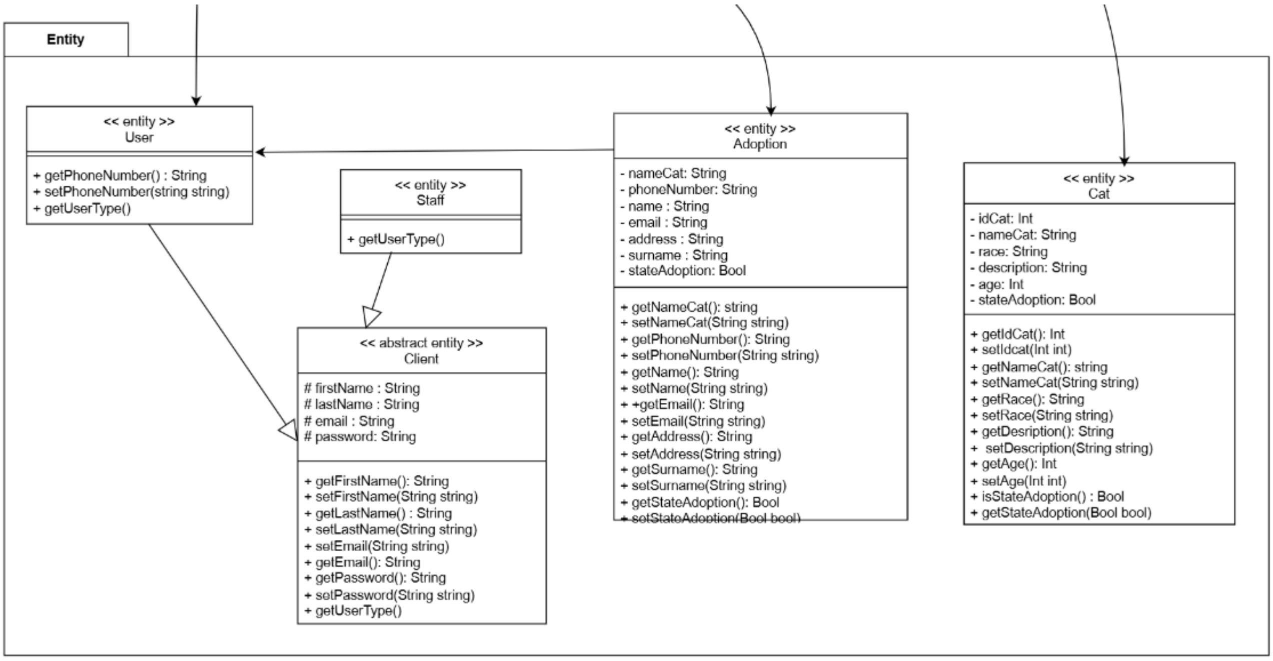


Figure 9: Book a Seat BCE entity

3.2 Vopc-MVC

Entity

Contains the domain classes — e.g., `User`, and `Booking`. These objects encapsulate state and expose standard getter/setter methods.

Controller — Application & UI

The *application controller* drives business logic and mediates between model and user interface. In the booking use case, the class `BookingService` coordinates the table–reservation workflow and `RequestAdoptionController` coordinates the adoption workflow. The *UI controller* layer deals with user events and refreshes the JavaFX views accordingly.

View

Hosts the JavaFX screens that constitute the graphical interface.

The classes respect the Model–View–Controller (MVC) pattern, with models located in the `Entity` package.

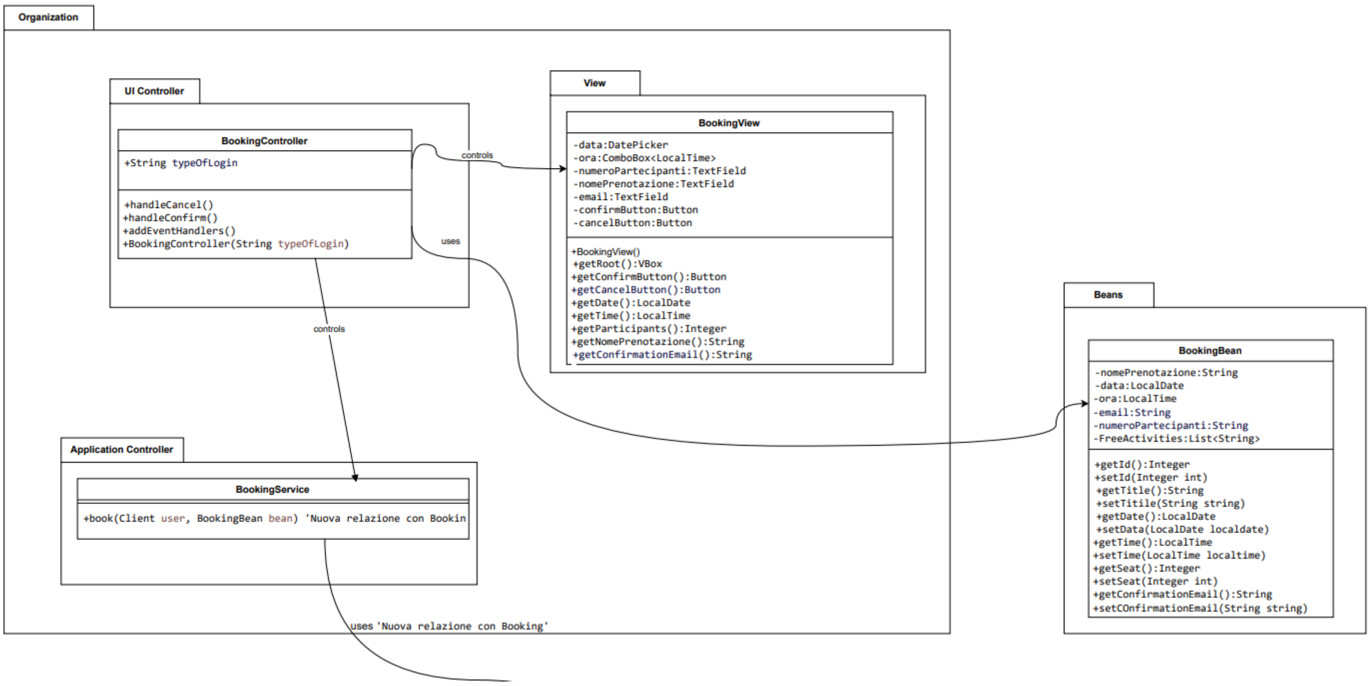


Figure 10: Booking-Controller View and beans

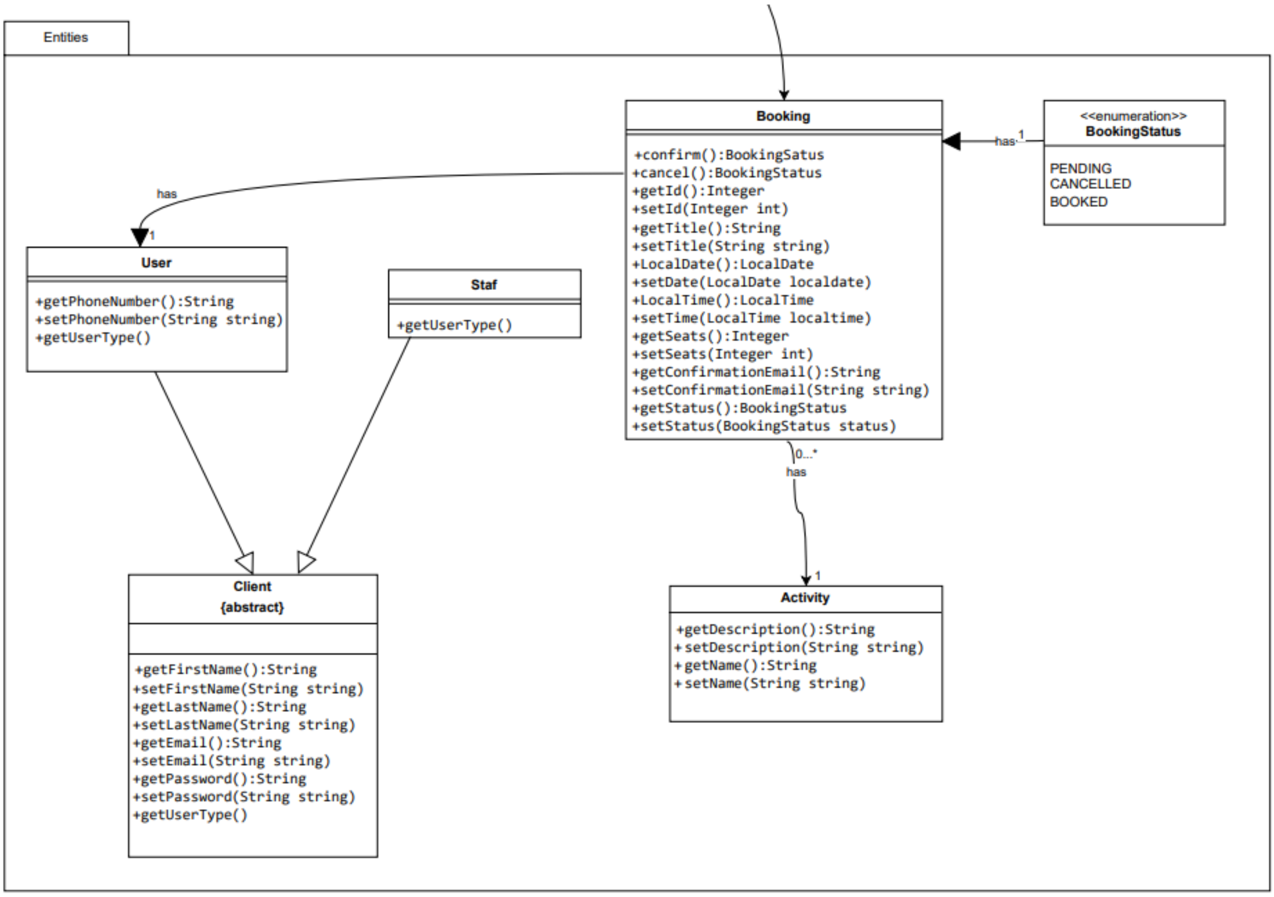


Figure 11: Booking-Entity

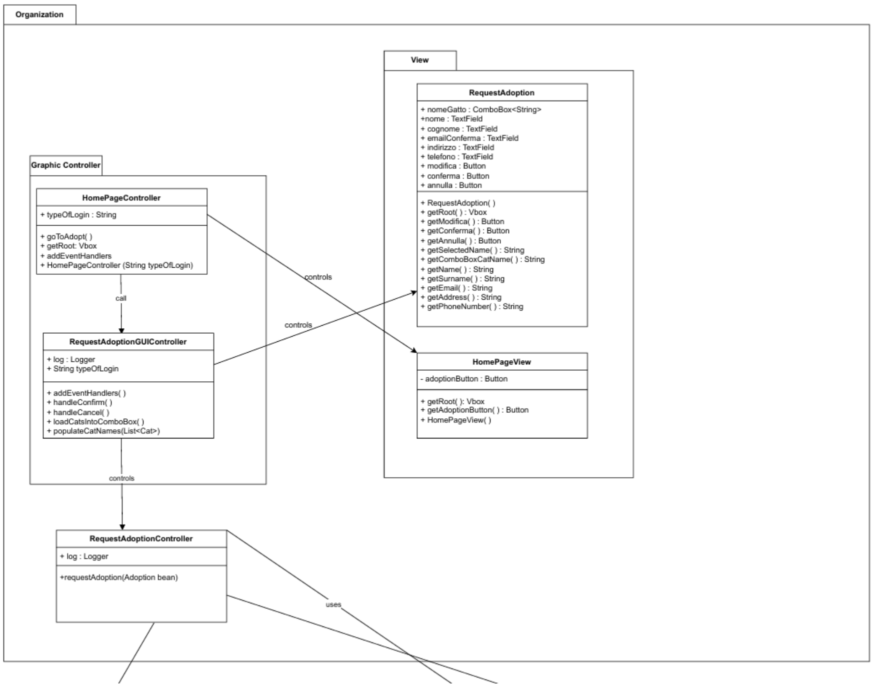


Figure 12: Adoption-Controller and View

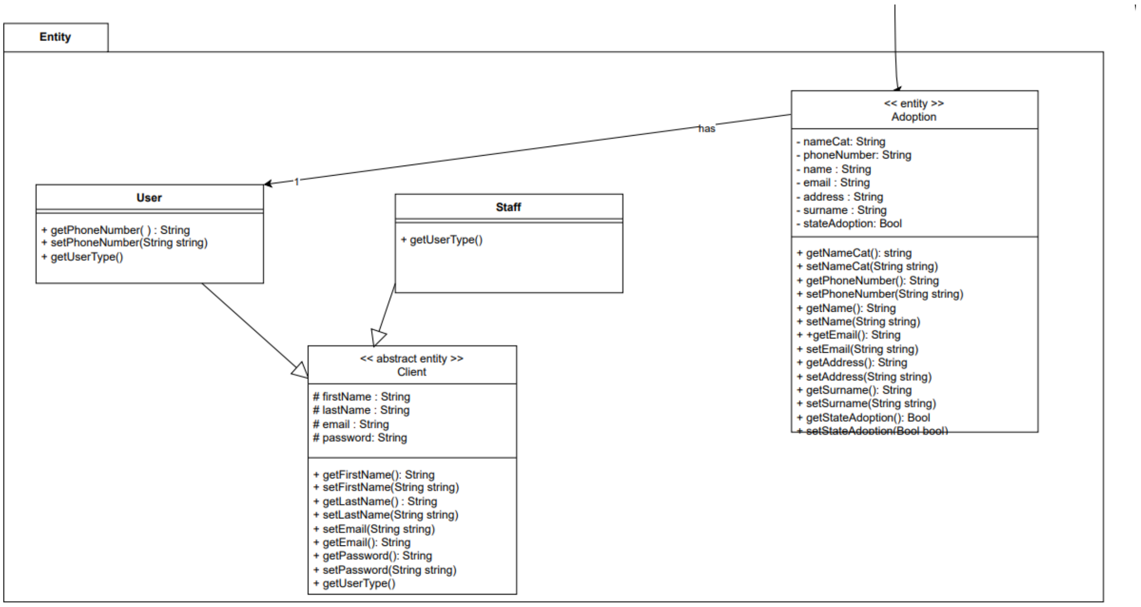


Figure 13: Adoption-Entity

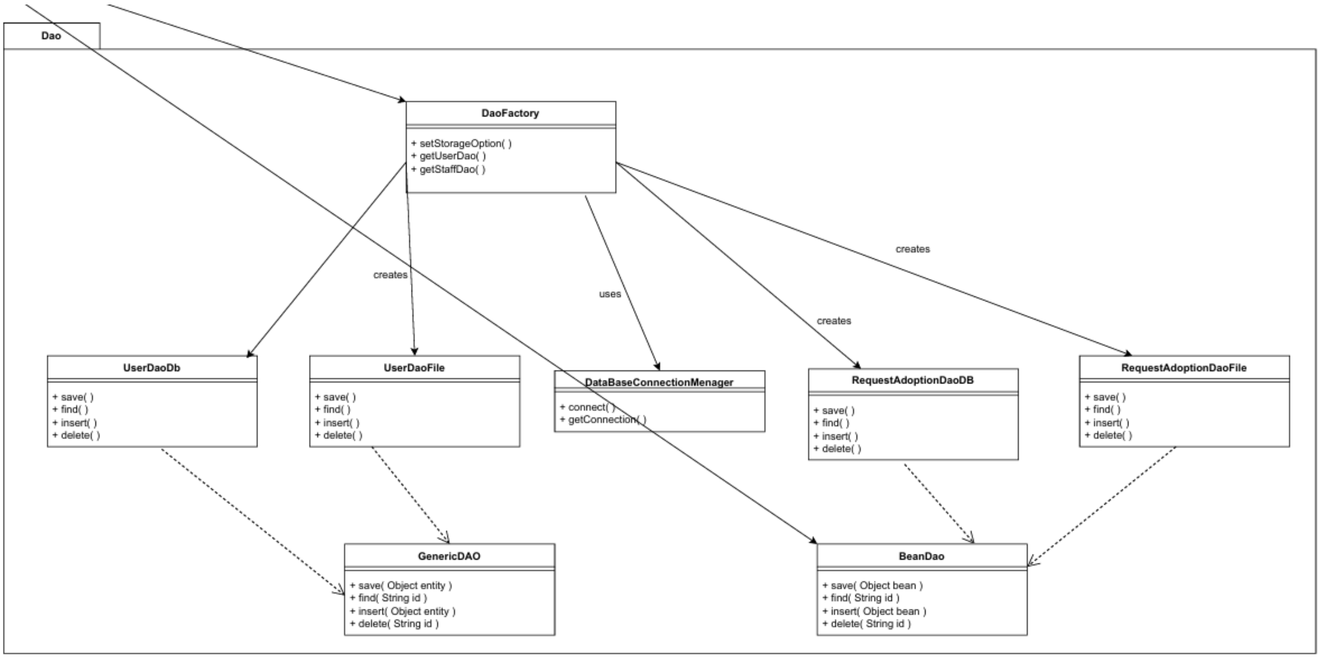


Figure 14: Adoption-Daos

3.3 Design level diagram

At the design level, the classes are detailed together with their specific relationships and responsibilities. As in the VOPC, the classes are organised according to the MVC pattern (the models are in the `Entity` package). The Stateless DAOs are not represented because they would have increased the size of the DAO; the links are the same as for the Database DAO. I cannot show the entire diagram—it is too large—therefore. Here I have split it by individual packages:

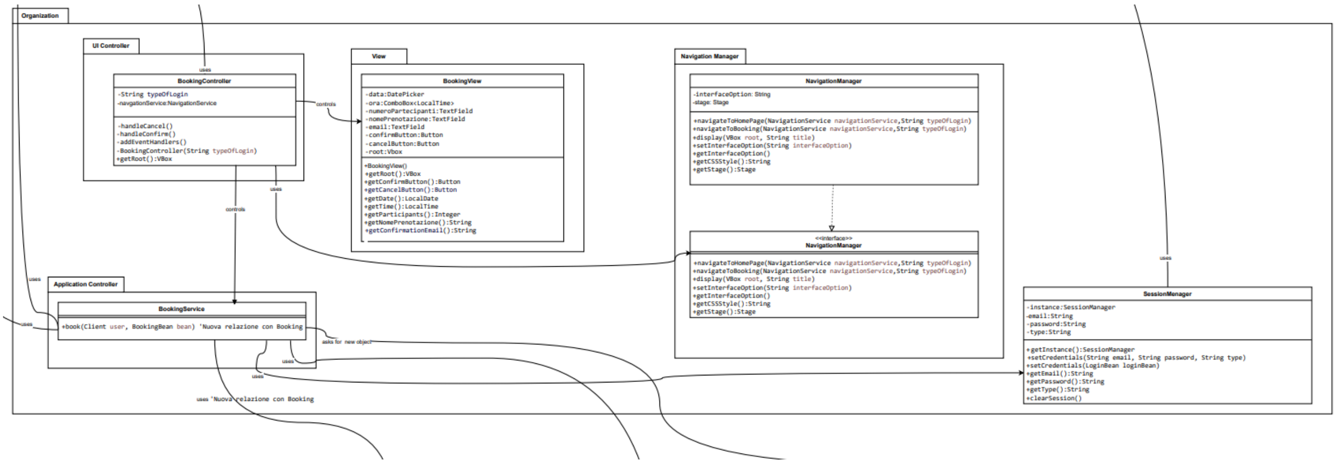

Organization
UI Controller
uses
BookingController
-String typeOfLogin
-navgationService:NavigationService
-handleCancel()
-handleConfirm()
-addEventHandlers()
-BookingController(String typeOfLogin)
+getRoot():VBox
controls
View
BookingView
-data:DatePicker
-ora:ComboBox<LocalTime>
-numeroPartecipanti:TextField
-nomePrenotazione:TextField
-email:TextField
-confirmButton:Button
-cancelButton:Button
-root:Vbox
+BookingView()
+getRoot():VBox
+getConfirmButton():Button
+getCancelButton():Button
+getDate():LocalDate
+getTime():LocalTime
+getParticipants():Integer
+getNomePrenotazione():String
+getConfirmationEmail():String
Navigation Manager
NavigationManager
-interfaceOption: String
-stage: Stage
+navigateToHomePage(NavigationService navigationService,String typeOfLogin)
+navigateToBooking(NavigationService navigationService,String typeOfLogin)
+display(VBox root, String title)
+setInterfaceOption(String interfaceOption)
+getInterfaceOption()
+getCSSStyle():String
+getStage():Stage
<<interface>>
NavigationManager
+navigateToHomePage(NavigationService navigationService,String typeOfLogin)
+navigateToBooking(NavigationService navigationService,String typeOfLogin)
+display(VBox root, String title)
+setInterfaceOption(String interfaceOption)
+getInterfaceOption()
+getCSSStyle():String
+getStage():Stage
uses
controls
uses
Application Controller
BookingService
+book(Client user, BookingBean bean) 'Nuova relazione con Booking
uses
asks for new object
uses
uses
uses 'Nuova relazione con Booking
uses
SessionMenager
-instance:SessionManager
-email:String
-password:String
-type:String
+getInstance():SessionManager
+setCredentials(String email, String password, String type)
+setCredentials(LoginBean loginBean)
+getEmail():String
+getPassword():String
+getType():String
+clearSession()

Figure 15: Booking-Controller and View

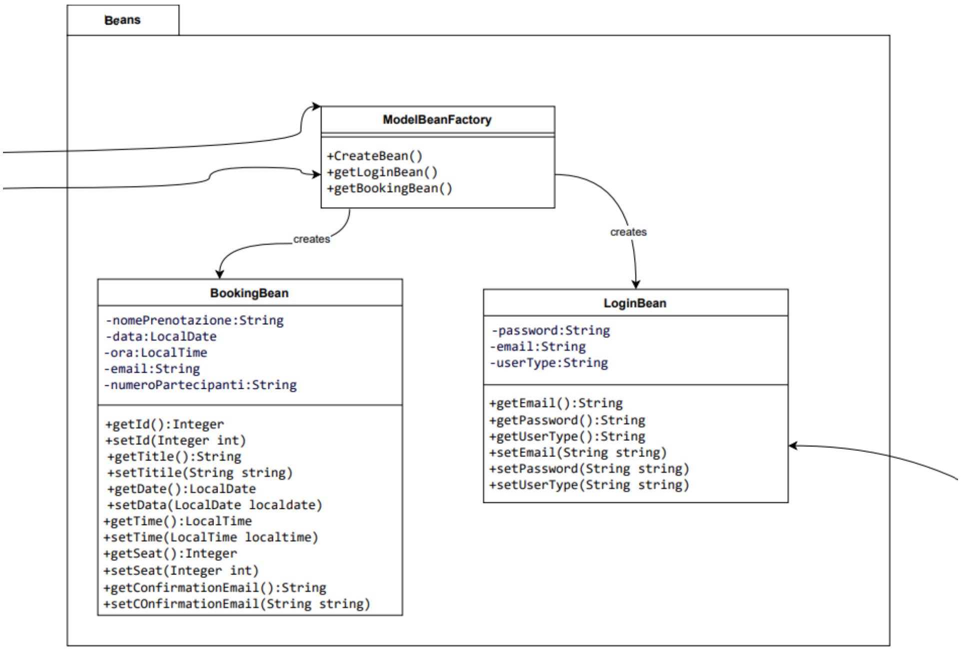


Figure 16: Booking-Beans

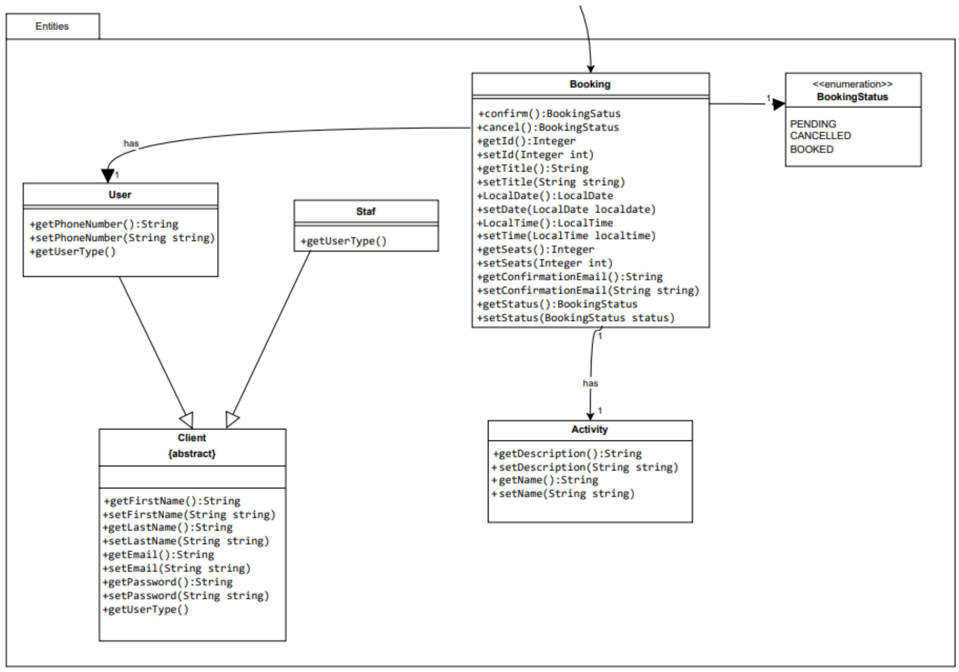


Figure 17: Booking-Entity

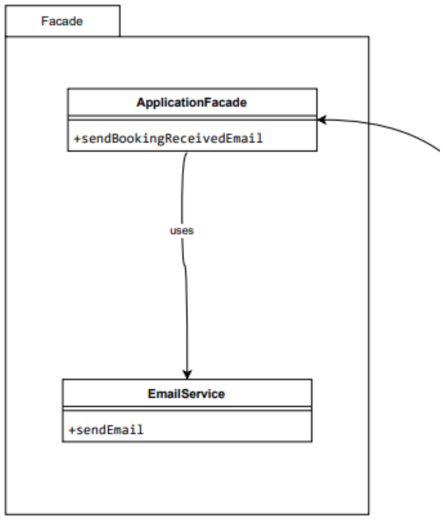


Figure 18: Booking-Pattern Facade

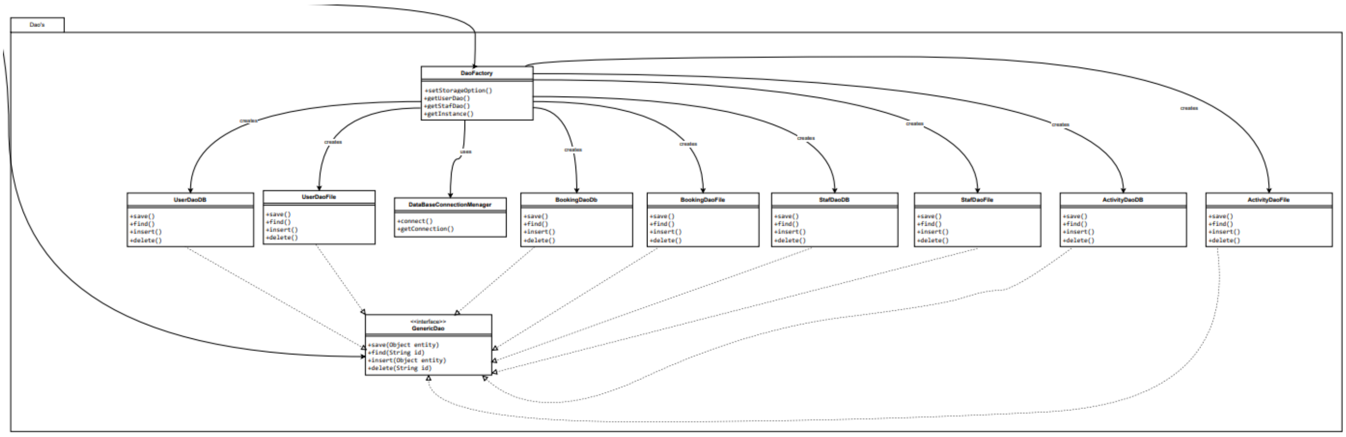


Figure 19: Booking-Daos- Pattern Factory and Singleton

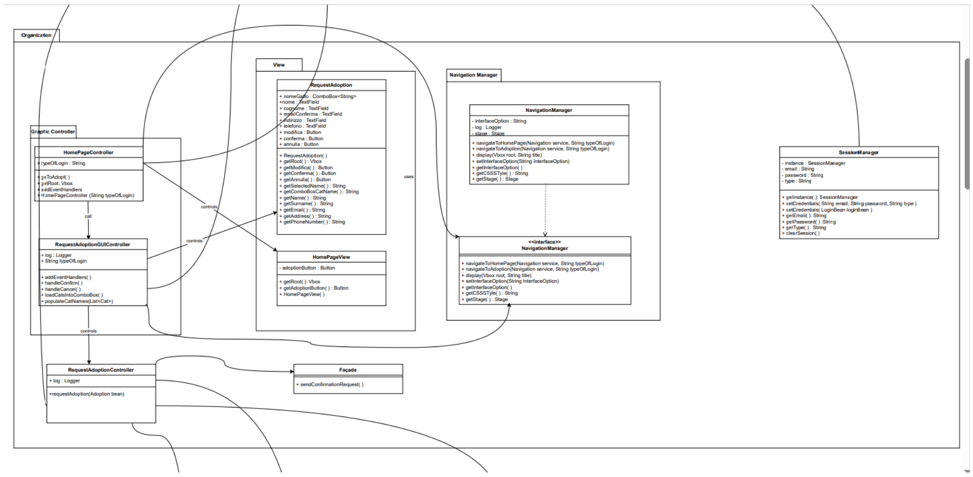

Organization
Graphic Controller
HomePageController
+ typeOfLogin : String
+ goToAdopt( )
+ getRoot: Vbox
+ addEventHandlers
+ HomePageController (String typeOfLogin)
call
RequestAdoptionGUIController
+ log : Logger
+ String typeOfLogin
+ addEventHandlers( )
+ handleConfirm( )
+ handleCancel( )
+ loadCatsIntoComboBox( )
+ populateCatNames(List<Cat>)
controls
View
RequestAdoption
+ nomeGatto : ComboBox<String>
+nome : TextField
+ cognome : TextField
+ emailConferma : TextField
+ indirizzo : TextField
+ telefono : TextField
+ modifica : Button
+ conferma : Button
+ annulla : Button
+ RequestAdoption( )
+ getRoot( ) : Vbox
+ getModifica( ) : Button
+ getConferma( ) : Button
+ getAnnulla( ) : Button
+ getSelectedName( ) : String
+ getComboBoxCatName( ) : String
+ getName( ) : String
+ getSurname( ) : String
+ getEmail( ) : String
+ getAddress( ) : String
+ getPhoneNumber( ) : String
HomePageView
- adoptionButton : Button
+ getRoot( ): Vbox
+ getAdoptionButton( ) : Button
+ HomePageView( )
uses
Navigation Manager
NavigationManager
- interfaceOption : String
- log : Logger
- stage : Stage
+ navigateToHomePage(Navigation service, String typeOfLogin)
+ navigateToAdoption(Navigation service, String typeOfLogin)
+ display(Vbox root, String title)
+ setInterfaceOption(String InterfaceOption)
+ getInterfaceOption( )
+ getCSSSTyle( ) : String
+ getStage( ) : Stage
<<interface>>
NavigationManager
+ navigateToHomePage(Navigation service, String typeOfLogin)
+ navigateToAdoption(Navigation service, String typeOfLogin)
+ display(Vbox root, String title)
+ setInterfaceOption(String InterfaceOption)
+ getInterfaceOption( )
+ getCSSSTyle( ) : String
+ getStage( ) : Stage
SessionManager
- instance : SessionManager
- email : String
- password : String
- type : String
+ getInstance( ): SessionManager
+ setCredentials(String email, String password, String type)
+ setCredentials(LoginBean loginBean)
+ getEmail( ): String
+ getPassword( ) :String
+ getType( ): String
+ clearSession( )
RequestAdoptionController
+ log : Logger
+requestAdoption(Adoption bean)
Façade
+ sendConfirmationRequest( )

Figure 20: Adoption-Controller and views

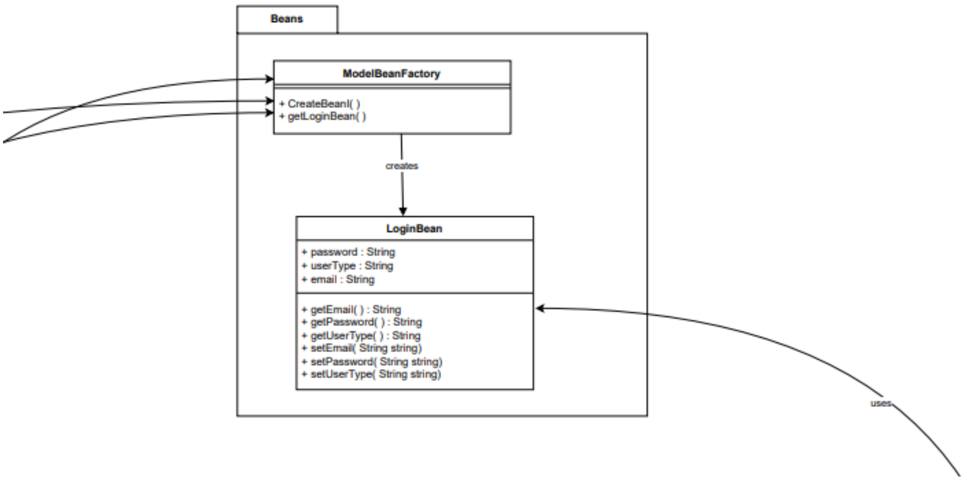


Figure 21: Adoption-Beans

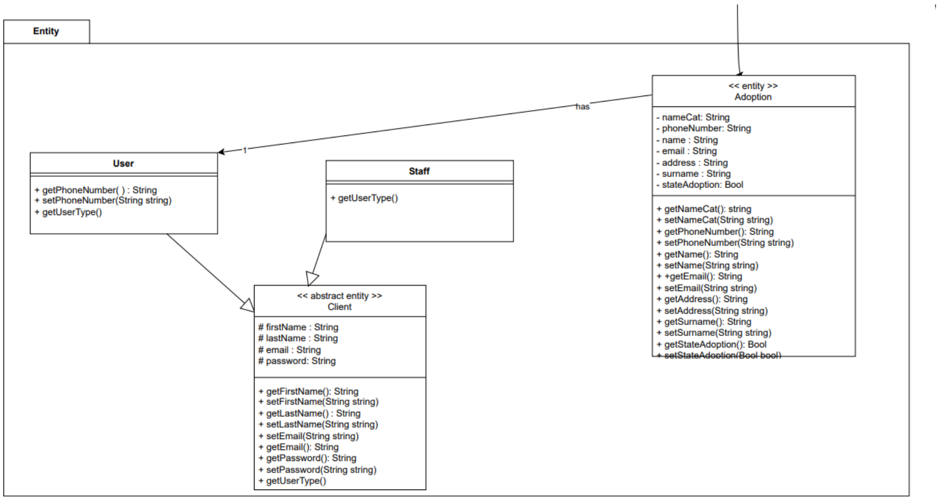


Figure 22: Adoption-Entity

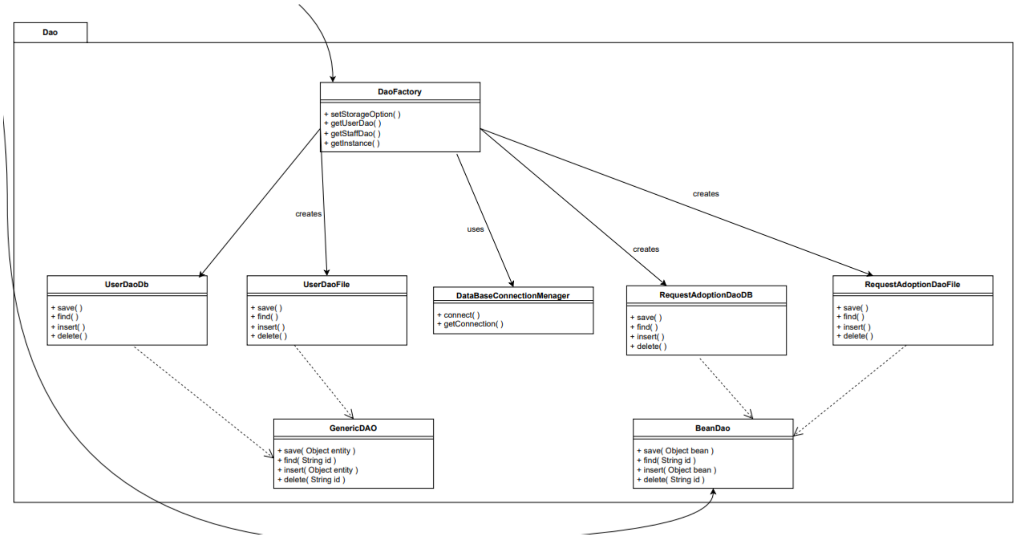


Figure 23: Adoption-Daos , factory pattern and singleton

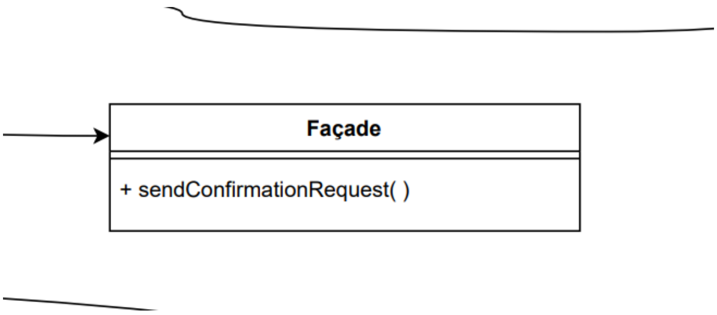


Figure 24: Adoption-Facade pattern

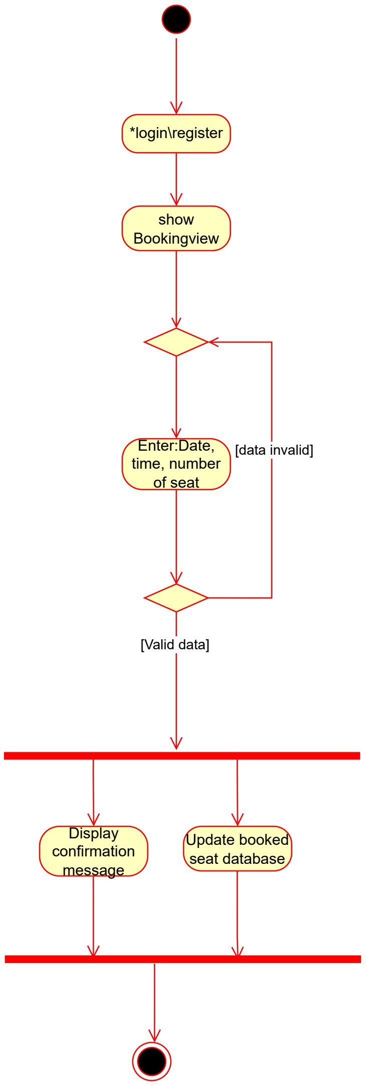
3.4 Design pattern

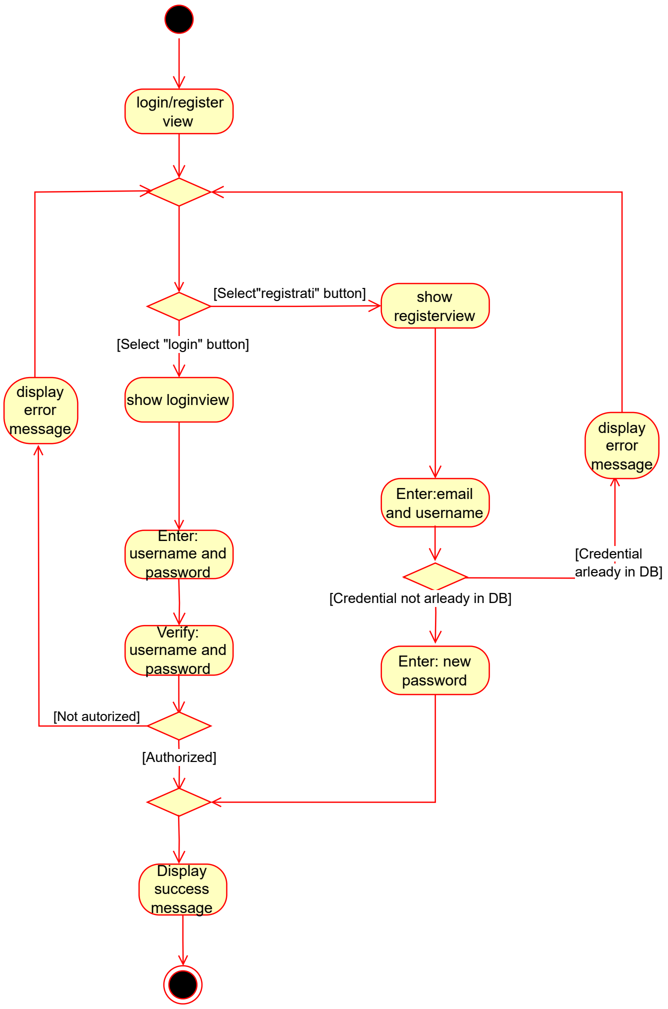
We adopted the Singleton pattern in the `SessionManager` class to ensure that the user instance interacting with the system is unique during its use. Secondly, I applied a Factory Method for creating the DAOs used by `BookingService`, so that the correct DAO can be chosen according to the system configuration (Database, File System, or Stateless). In this way I preserved a separation of responsibilities in line with the MVC pattern.

Finally, the `ApplicationFacade` class implements the Facade pattern, allowing every class to access services implemented by other classes, regardless of the class that provides those services. `ApplicationFacade` undertakes to supply this functionality without needing to know the inner workings of any other system class involved in the process. This enables the classes that use it to interact with multiple subsystems without knowing the details of their implementation.

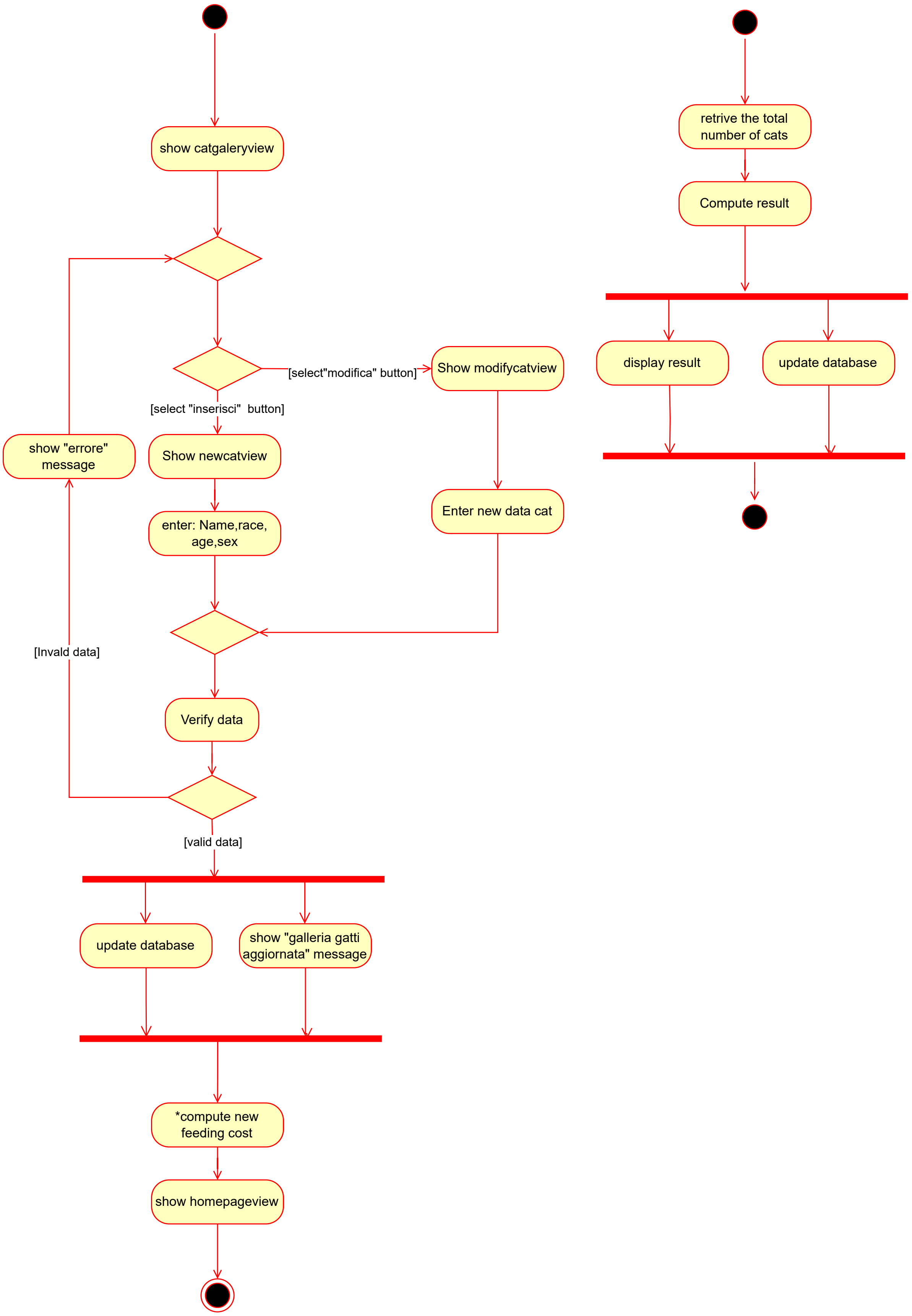
3.5 Activity diagram

USE CASE: BOOK SEAT

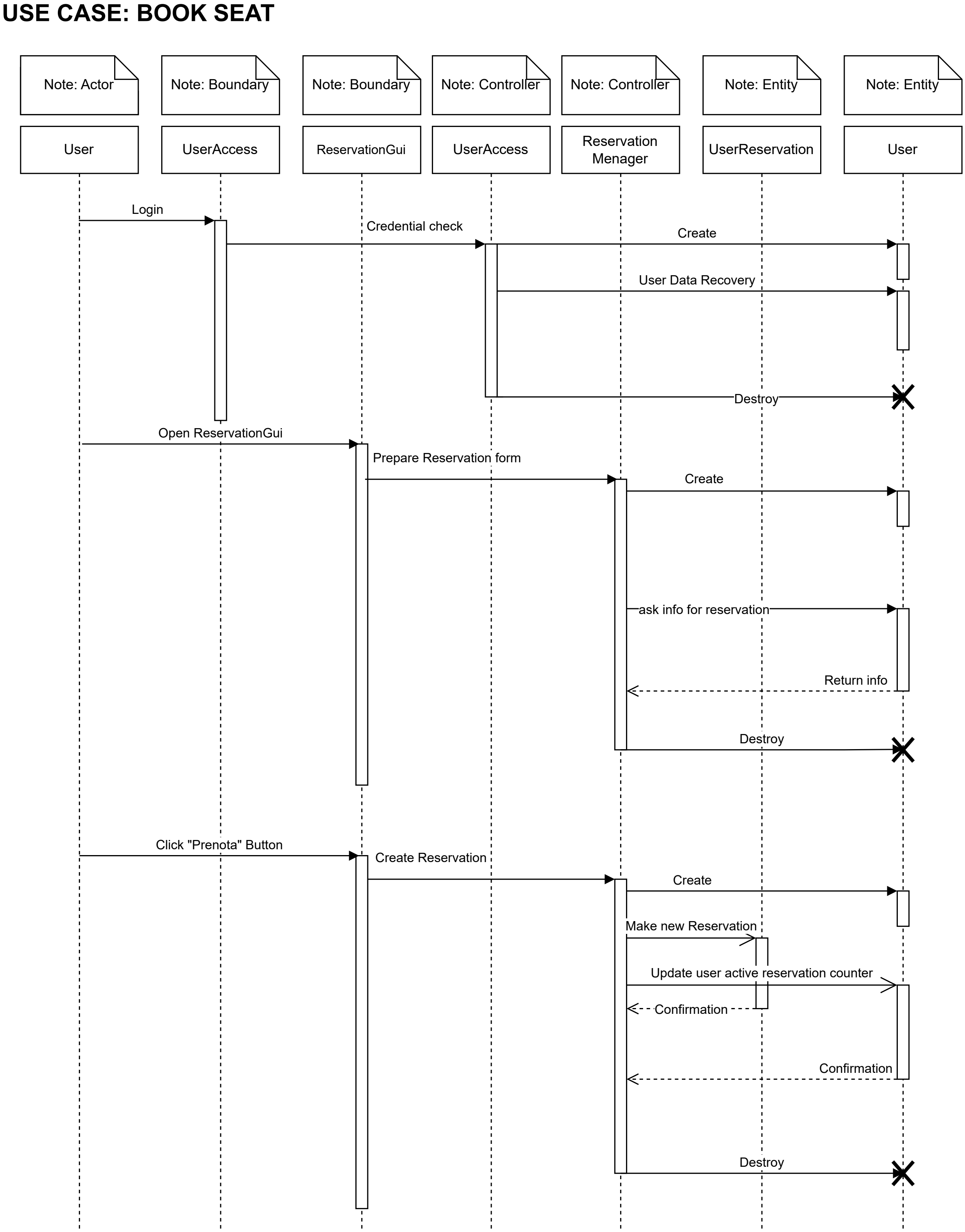

*login\register
show Bookingview
Enter:Date, time, number of seat
[data invalid]
[Valid data]
Display confirmation message
Update booked seat database



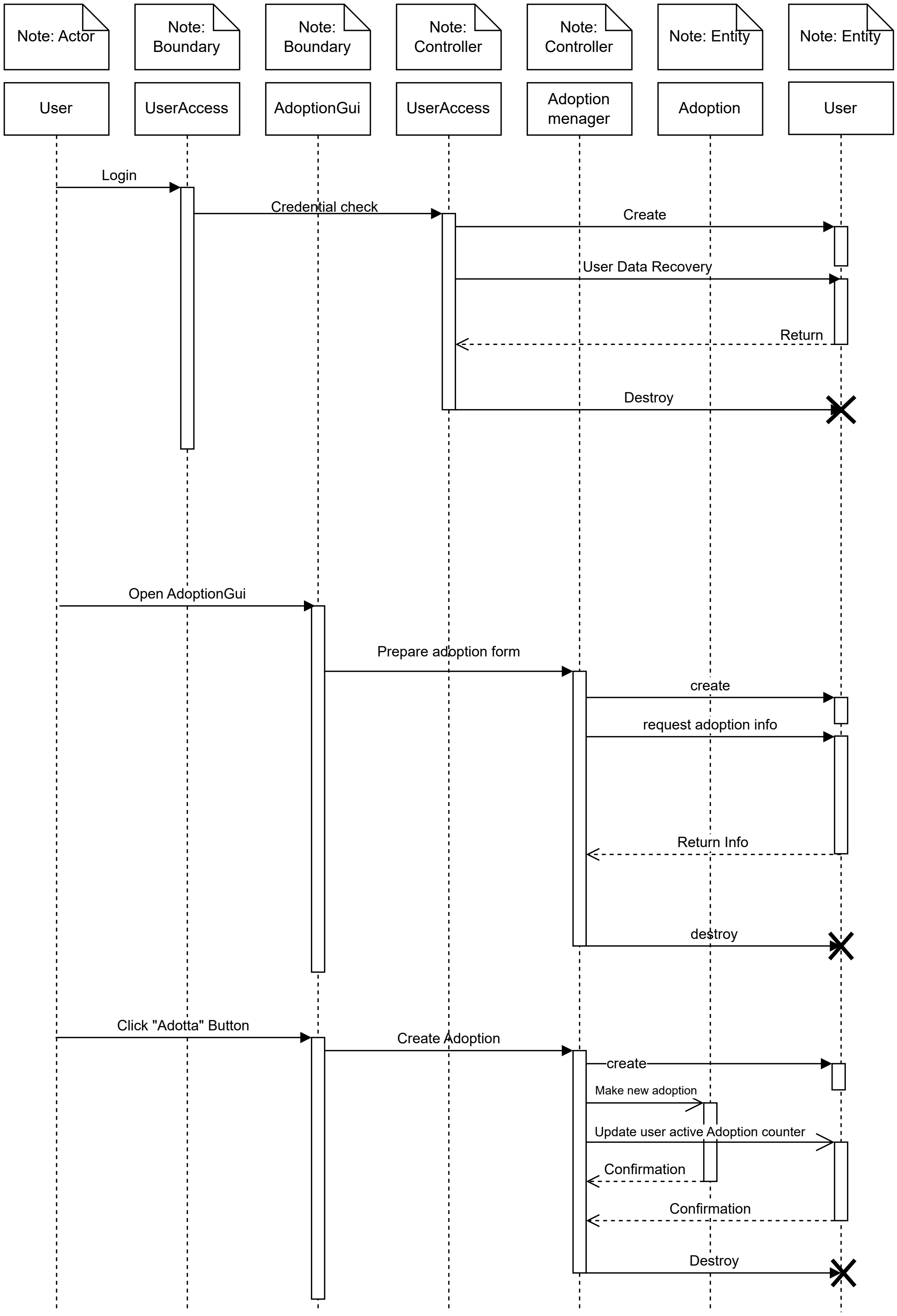
USE CASE: MANAGE CAT



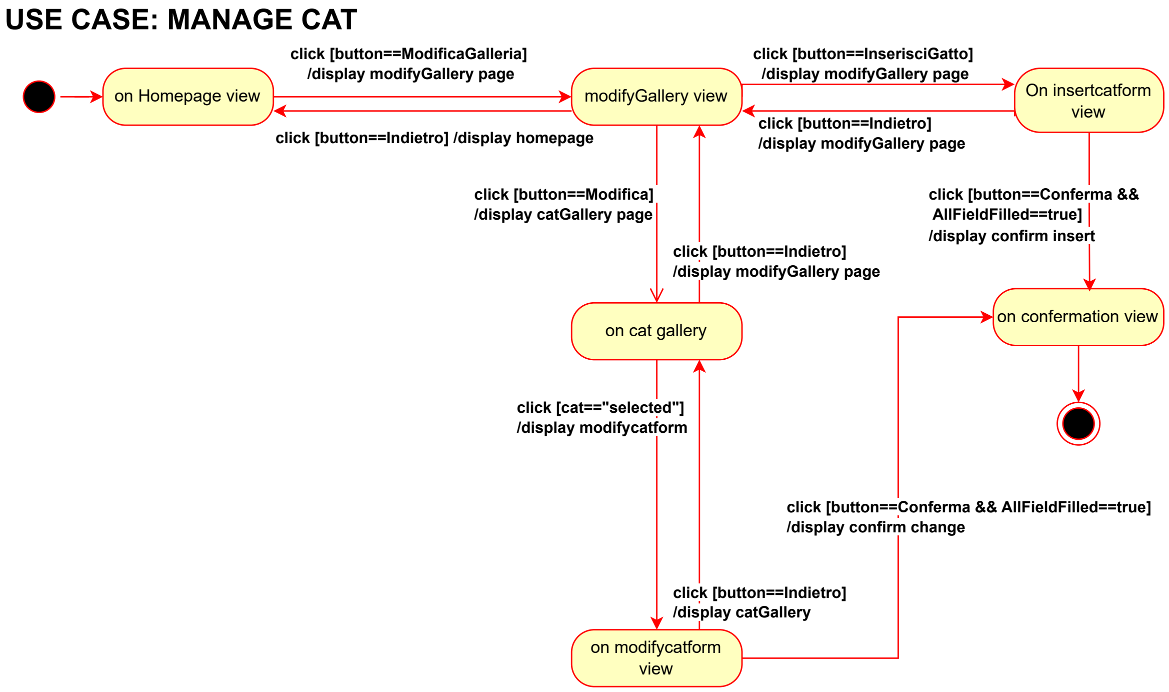
3.6 Sequence diagram



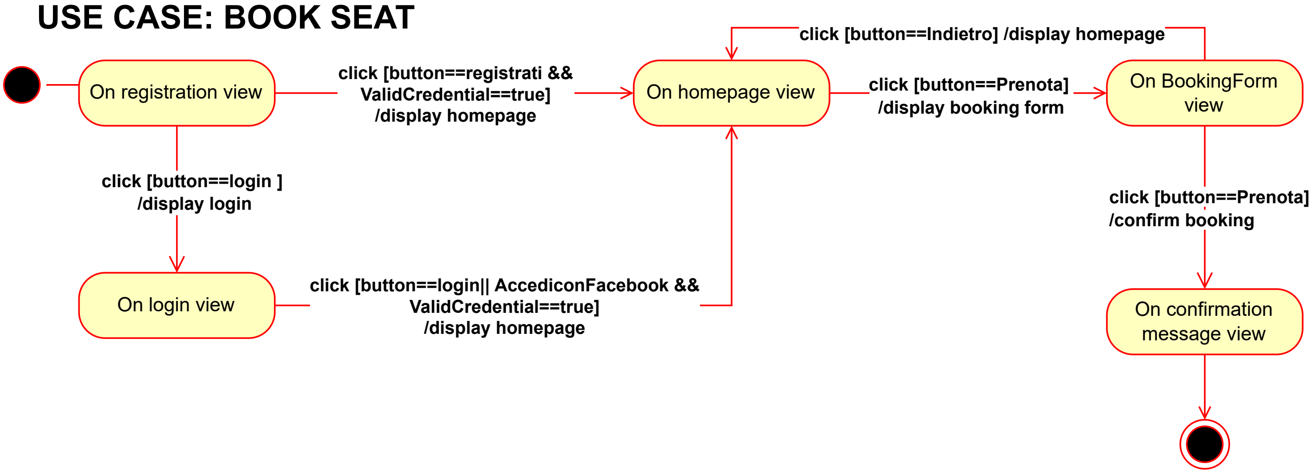
USE CASE: ADOPT



3.7 State diagram



USE CASE: BOOK SEAT


On registration view
click [button==registrati && ValidCredential==true] /display homepage
On homepage view
click [button==Indietro] /display homepage
click [button==Prenota] /display booking form
On BookingForm view
click [button==login] /display login
On login view
click [button==login|| AccediconFacebook && ValidCredential==true] /display homepage
click [button==Prenota] /confirm booking
On confirmation message view

4 Testing

In the application testing module, three main test classes have been implemented to verify the correct functioning of the fundamental system functionalities.

`BookingServiceTest`

- `testPrenotazione`: Checks that a valid booking is successfully created and correctly stored in the database.
- `testValidazione`: Verifies that the system rejects a booking with invalid input data.

`BookingServiceTest`

- `testRichiestaValida`: Checks that a valid Adoption is successfully created and correctly stored in the database.
- `testRichiestaNonValida`: Verifies that the system rejects an Adoption with invalid input data.

`ValidateLoginTest`

- `testValidCredentials`: Ensures that a user with correct credentials can log in to the system.
- `testInvalidCredentials`: Confirms that the system denies access to users with incorrect credentials and raises an appropriate exception.
- `testEmptyFields`: Checks that the system properly handles login attempts with empty fields.

`ClientRegistrationControllerTest`

- `testRegistrazioneValida`: Ensures that a user with correct credentials can register into the system.
- `testErroreValidazione`: Confirms that the system denies registration to users with incorrect credentials and raises an appropriate exception.
- `testUtenteEsistente`: Checks that the system properly handles registration attempts with empty fields.

`DaoFactoryTest`

- `givenStateless_whenGetUserDao`: Ensures that the correct DAO implementation for customers is returned, according to the selected storage mode.
- `givenStateless_whenGetStaffDao`: Verifies that the DAO for staff members is correctly initialised.
- `givenStateless_whenGetBookingDao`: Confirms that booking management is handled by the appropriate DAO class.
- `givenDatabase_whenGetDaos_thenUseConnectionManager`: Checks that the database connection is correctly established via the connection manager.

DaoFactoryTest

- testMappingCampi: Ensures that the correct Bean implementation for customers is returned
- testStatoIniziale: Verifies that the starter Adoption State is false .

5 persistence management

The system uses two different types of persistence modes, in addition to the memory for demo mode.

5.1 File System

The system relies on a set of JSON files that store information about the entities used by the application and, in the case of bookings—which are aggregates of multiple entities—their associations. The files are organised as follows:

- connectionSettings.json: Contains the server-connection settings, including the URL, user name, and password required to access the remote database.
- cats.json: A JSON file that serialises data related to cats.
- bookings.json: A JSON file that serialises data on the bookings made.
- request_adoptions.json: A JSON file that serialises information on completed adoptions.
- staf.json: A JSON file that serialises data concerning staff members.
- users.json: A JSON file that serialises data concerning users.
- activity.json: A JSON file that serialises data concerning activity.

5.2 DataBase

The other type of database management system used is the connection to a MySQL server to access the information in real time. The MySQL database stores, as in the case of the file system, the entities, using at least one table for each entity. In particular:

- The database tables correspond to the main entities of the system, such as the customer, rooms, reservations and services.
- Updates made to the data via the file system are synchronized with the MySQL database to ensure the consistency of the information

6 Link

The following links are available for the project:

- **Source Code (GitHub):** https://github.com/MatteoAmato18/ISPW-Cat-Cafe/tree/171f43bb5e42d7cb5b1d98424c83bcc74af3ff8d/src — the complete code of our programme.
- **Class Diagrams:** https://github.com/MatteoAmato18/ISPW-Cat-Cafe/tree/171f43bb5e42d7cb5b1d Deliverables — all class diagrams in full resolution for easier viewing.

- **SonarCloud Report:** The system was analysed with SonarCloud, which identified potential bugs and vulnerabilities. The results are available at `https://sonarcloud.io/summary/new_code?id=MatteoAmato18_ISPW-Cat-Cafe&branch=main`.
- **Presentation Video:** `https://github.com/MatteoAmato18/ISPW-Cat-Cafe/tree/91da062c6d852bf5 Video` — the presentation video of the Cat Café Management System.