# My Holiday

## Project for ISPW course - 2023/2024

## Index

Notes:
- All terms formatted as **term*** are defined in the dictionary of terms.

- All diagrams are available as standalone PDF files in the diagrams folder.

# 1 Software requirements specification

## 1.1 Introduction

### 1.1.1 Dictionary of terms

| | |
|---|---|
| **Holiday requirements** | Set of properties required for a holiday:<br>destination, available budget, departure date, return date, number of travelers, type of accommodation (unspecified or hotel), accommodation quality (from one to five stars), number of rooms, type of transport (unspecified, airplane, train, bus or ferry), transport quality (from one to five stars), place of departure. |
| **Offer** | Set of properties chosen by a travel agency in an attempt to satisfy some specific holiday requirements.<br>The properties are:<br>destination, price, departure date, return date, number of travelers, type of accommodation (only hotel is available for now), name of the accommodation, accommodation quality (from one to five stars), accommodation address, number of rooms, type of transport (airplane, train, bus or ferry), name of the transport company, transport quality (from one to five stars), place of departure. |

### 1.1.2 Overview of the defined system

My Holiday is a system whose purpose is to help users to find the ideal holiday.
A registered user can publish announcements, in an announcement the user describes his **holiday requirements***.

Travel agencies registered with the system can read the announcements posted by users and respond with an **offer***.

Once a user receives an **offer*** for one of his announcements he can:

- reject it: in this case the offer will be removed from the system.

- accept it: in this case the system will book the accommodation and transport offered by the travel agency and then it will request payment via credit card. If payment is successful then the system will enable the user to download the documents related to the holiday (i.e transport tickets).

- request changes on it: in this case the user specifies which changes are required on the received offer, those will be sent to the travel agency that made the **offer***.
  The travel agency can reject the requested changes (in which case the original **offer*** will remain valid) or make a counter offer trying to satisfy user requests (this new **offer*** will replace the original one).

### 1.1.3 Hardware and software requirements

The system has been developed as a desktop application using the Java programming language (JDK 21).

There are no particular hardware requirements needed to run the system, it can be executed on every machine that supports (and has installed) a Java Runtime Environment.

The persistency layer uses a MySQL database and for the correct working of the system a file called "myHolidayDb.cfg" is required, such file contains credentials for database access.

The system is designed to use external APIs to: research and book accommodations, research and book transports, make payments with credit card and verify the existence of addresses and locations.
For now the accommodations, transports and address checking APIs are simulated internally by the system while the payment API is not present due to the fact that the "accept offer" use case has not been implemented.

### 1.1.4 Related systems

**Booking.com**

Booking.com is one of the most popular website to book holidays online, this system enables the user to search for hotels and flights.
The main difference with My Holiday is that the user must research the accommodation and flight and then choose between the available options.
In our system instead the user posts an announcement and then the travel agencies responds with offers, so the travel agencies are the ones researching for announcements.
Another difference is that Booking.com offers only airplanes as a means of transport while My Holiday offers airplanes, trains, buses and ferries.
On the other hand Booking.com offers some additional services (for example car rental) that My Holiday is missing.

**Trivago**

Trivago is another popular website used to book holidays online.
In particular Trivago is a search engine that enables the user to search for hotels and compare their prices on different OTA (Online Travel Agencies).
As said with Booking.com, the main difference with My Holiday is that the user must research the accommodation and then select between the available options; in our system instead the user publish his **holiday requirements*** and receives **offers*** from travel agencies.
As a con Trivago does not enable the user to search for means of transport while My Holiday does.

## 1.2 User stories

1] As a user I want to publish an announcement that describes my **holiday requirements*** so that I can get **offers*** from travel agencies.

2] As a travel agency I want to respond to the announcements posted by users with **offers*** so that I can find clients.

3] As a user I want to request changes on the **offers*** received for my announcements so that I can get **offers*** that fits my needs.

## 1.3 Functional requirements

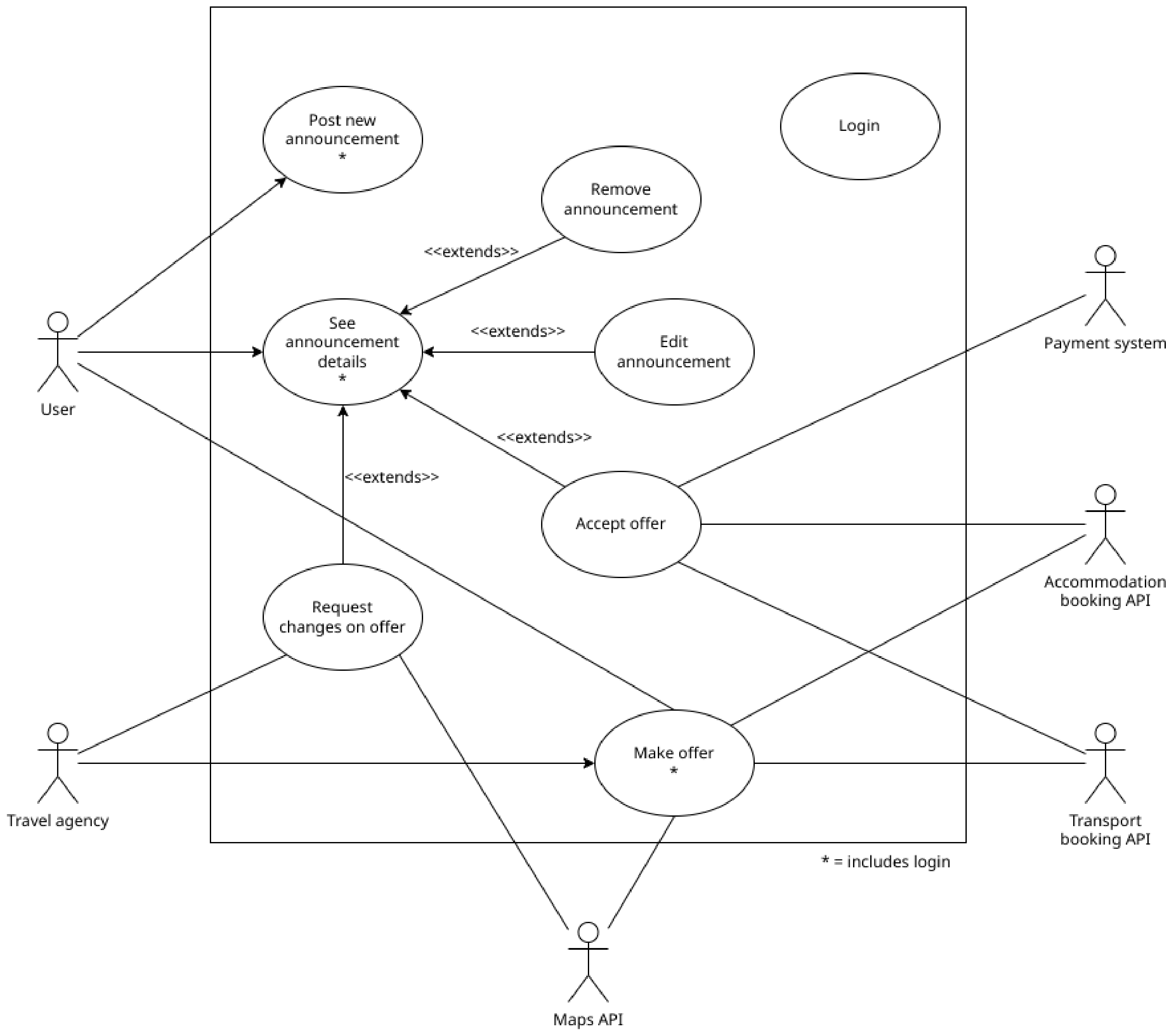
1] The system shall display all the announcements posted by users to travel agencies.

2] When the user accepts an **offer*** made by a travel agency, the system shall book the accommodation and the transport and then request payment by credit card from the user.

3] The system shall display for each announcement the number of views by travel agencies.

## 1.4 Use cases

### 1.4.1 Use case diagram



### 1.4.2 Internal steps

Assuming that the user has already selected an offer received for one of his announcements, those are the internal steps for the “request changes on offer” use case:

1] The system shows the details of the **offer***.
2] The system gives a form to describe changes in the **offer***.
3] The user fills in the fields of the form for which a change is required.
4] The user clicks the send request button.
5] The system checks that at least one field of the form has been filled in.
6] The system sends the requested changes to the travel agency that made the **offer***.
7] The system shows to the travel agency the changes requested by the user.
8] The travel agency evaluates the requested changes and clicks the make counter offer button.
9] The system gives a form to describe the new **offer***.
10] The travel agency fills in the form.
11] The travel agency clicks the send counteroffer button.
12] The system checks that all the fields of the form have been filled in.
13] The system updates the original **offer*** with the new one.
14] The system notifies the user of the counteroffer.

Extensions:

5a] All fields are blank: The system displays an error message explaining that no change has been specified.

8a] Travel agency clicks the reject changes button: the system removes the request of changes from the database and notifies the user.

12a] There is an empty field: The system displays an error message and highlights the empty field.

## 2 Storyboards

Storyboards are to big too be integrated into this document, you can find them in the diagram folder.

# 3 Design

## 3.1 Class diagram

The class diagram is too big to be integrated into this document, you can find it in the diagram folder.

## 3.2 Design patterns

One of the design patterns implemented in the system is Adapter, in particular this pattern has been used to adapt the HolidayOffer model class in order to implement the ChangesRequest model class.

The idea behind the implementation is that we can represent a request of changes (made by a user on one of the received offers) as a couple of holiday offers.
The first offer is the one made by the travel agency, while the second is the offer that the user would like to receive.
Comparing those two holiday offers we can see where they differ and so we can determine the changes requested by the user.

To implement the pattern I had to introduce the following classes and interfaces:

- ChangesRequest: an interface that declares all the operations that can be performed on a request of changes.

- HolidayOfferAdapter: class that aggregates two instances of HolidayOffer and uses them to implement the operations declared by the ChangesRequest interface.

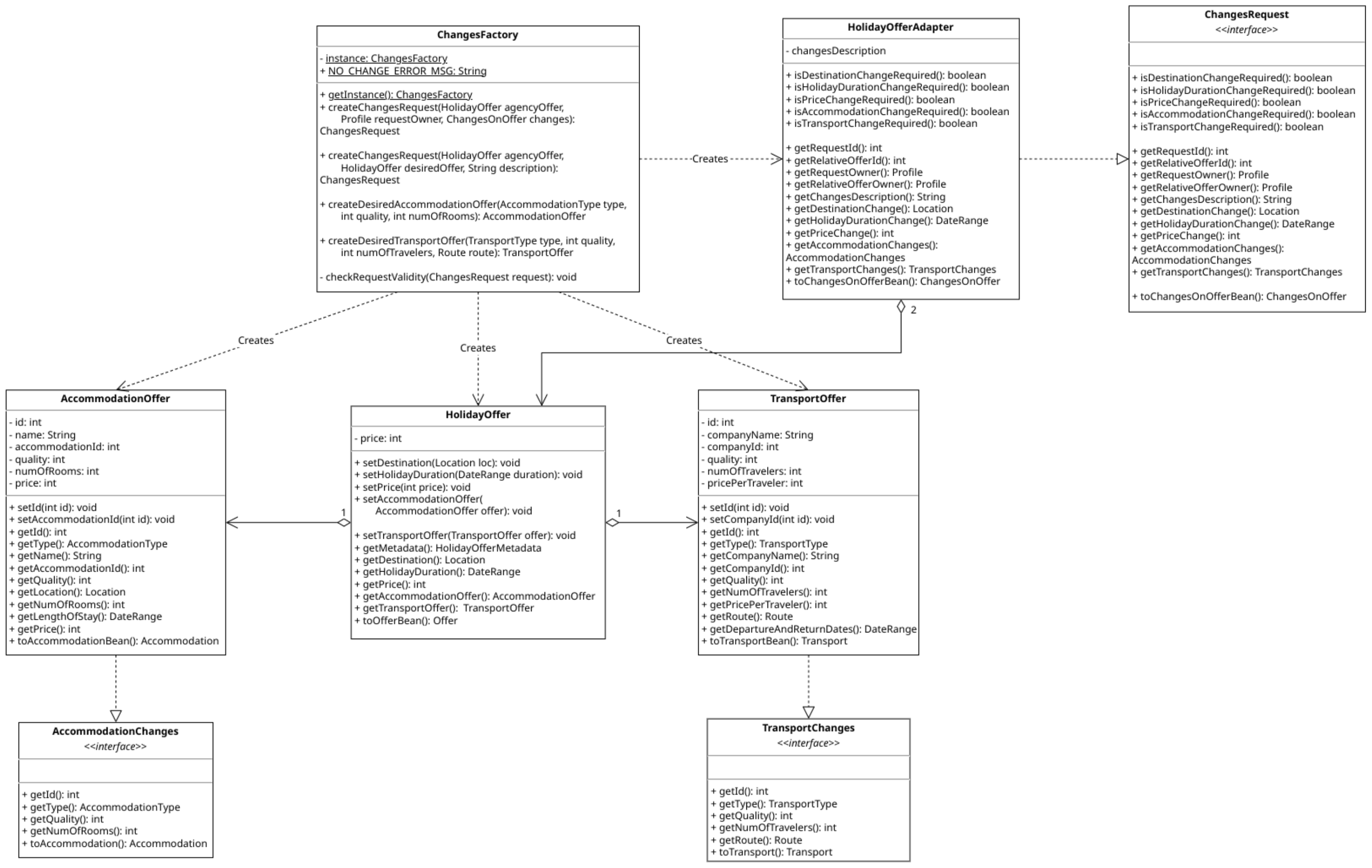
- ChangesRequestFactory: factory class used in the system to build HolidayOfferAdapter instances that are returned as ChangesRequest.

- AccommodationChanges: an interface implemented by the AccommodationOffer class, enables us to reuse the AccommodationOffer class to represent changes requested on the accommodation.

- TransportChanges: an interface implemented by the TransportOffer class, enables us to reuse the TransportOffer class to represent changes requested on the transport.

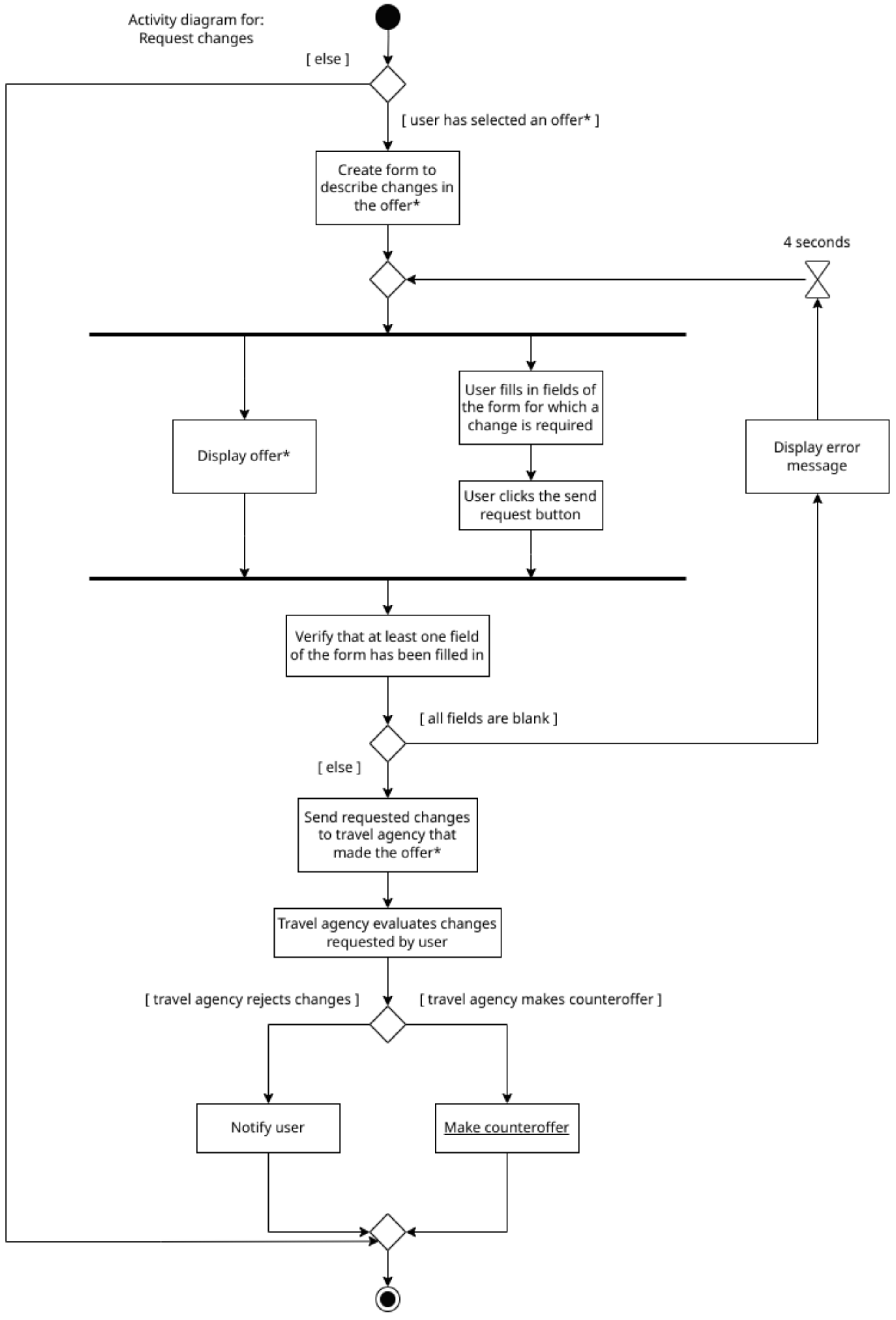
A pro of using the adapter pattern in this way is that we don't need to write other model classes to represent the concept of a request of changes.
A con is that the ChangesRequestFactory is coupled to the HolidayOffer, AccommodationOffer and TransportOffer classes; indeed creation of such classes is required to build the offer desired by the user.

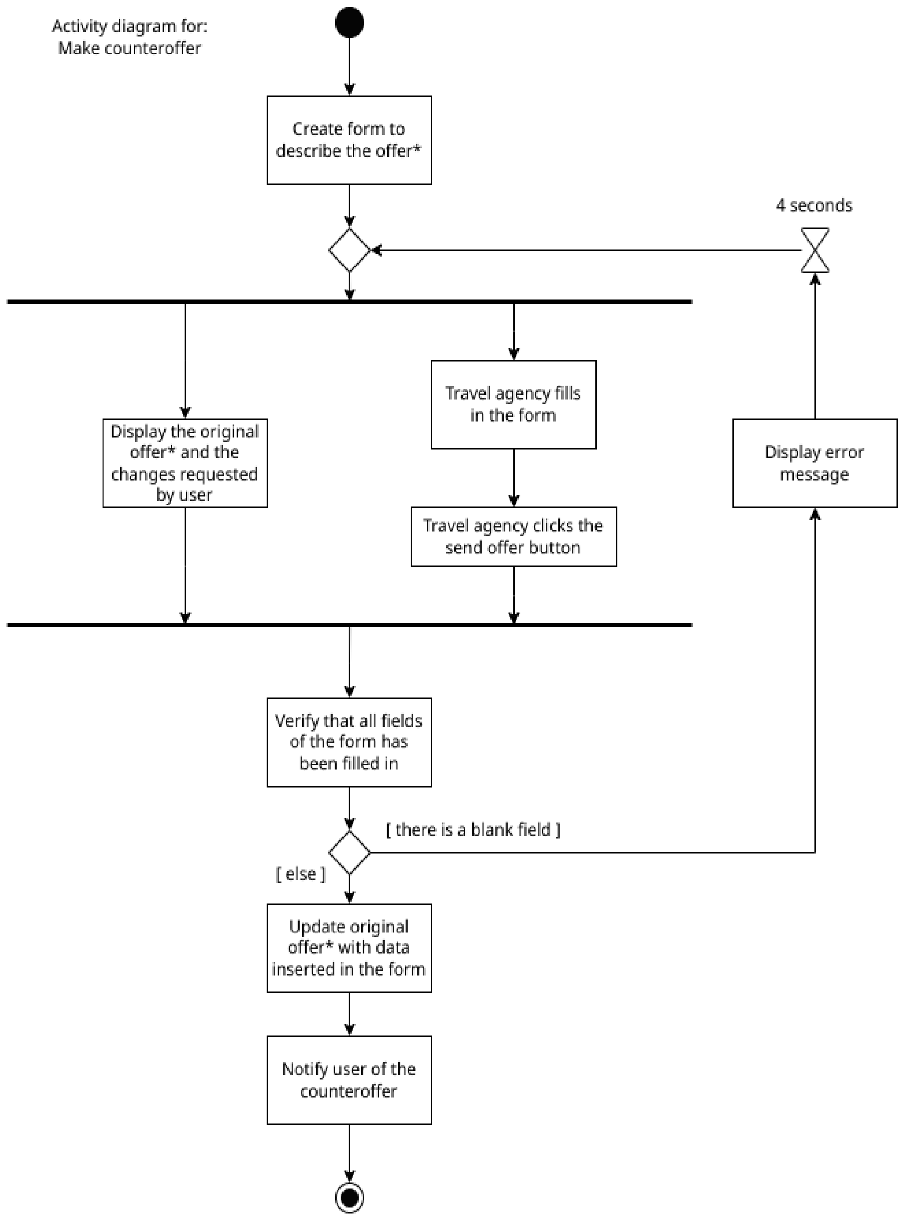
Here is a snippet extracted from the class diagram that represents the pattern implementation:


ChangesFactory
- instance: ChangesFactory
+ NO_CHANGE_ERROR_MSG: String
+ getInstance(): ChangesFactory
+ createChangesRequest(HolidayOffer agencyOffer, Profile requestOwner, ChangesOnOffer changes): ChangesRequest
+ createChangesRequest(HolidayOffer agencyOffer, HolidayOffer desiredOffer, String description): ChangesRequest
+ createDesiredAccommodationOffer(AccommodationType type, int quality, int numOfRooms): AccommodationOffer
+ createDesiredTransportOffer(TransportType type, int quality, int numOfTravelers, Route route): TransportOffer
- checkRequestValidity(ChangesRequest request): void
Creates
HolidayOfferAdapter
- changesDescription
+ isDestinationChangeRequired(): boolean
+ isHolidayDurationChangeRequired(): boolean
+ isPriceChangeRequired(): boolean
+ isAccommodationChangeRequired(): boolean
+ isTransportChangeRequired(): boolean
+ getRequestId(): int
+ getRelativeOfferId(): int
+ getRequestOwner(): Profile
+ getRelativeOfferOwner(): Profile
+ getChangesDescription(): String
+ getDestinationChange(): Location
+ getHolidayDurationChange(): DateRange
+ getPriceChange(): int
+ getAccommodationChanges(): AccommodationChanges
+ getTransportChanges(): TransportChanges
+ toChangesOnOfferBean(): ChangesOnOffer
ChangesRequest
<<interface>>
+ isDestinationChangeRequired(): boolean
+ isHolidayDurationChangeRequired(): boolean
+ isPriceChangeRequired(): boolean
+ isAccommodationChangeRequired(): boolean
+ isTransportChangeRequired(): boolean
+ getRequestId(): int
+ getRelativeOfferId(): int
+ getRequestOwner(): Profile
+ getRelativeOfferOwner(): Profile
+ getChangesDescription(): String
+ getDestinationChange(): Location
+ getHolidayDurationChange(): DateRange
+ getPriceChange(): int
+ getAccommodationChanges(): AccommodationChanges
+ getTransportChanges(): TransportChanges
+ toChangesOnOfferBean(): ChangesOnOffer
2
Creates
Creates
Creates
AccommodationOffer
- id: int
- name: String
- accommodationId: int
- quality: int
- numOfRooms: int
- price: int
+ setId(int id): void
+ setAccommodationId(int id): void
+ getId(): int
+ getType(): AccommodationType
+ getName(): String
+ getAccommodationId(): int
+ getQuality(): int
+ getLocation(): Location
+ getNumOfRooms(): int
+ getLengthOfStay(): DateRange
+ getPrice(): int
+ toAccommodationBean(): Accommodation
1
HolidayOffer
- price: int
+ setDestination(Location loc): void
+ setHolidayDuration(DateRange duration): void
+ setPrice(int price): void
+ setAccommodationOffer(AccommodationOffer offer): void
+ setTransportOffer(TransportOffer offer): void
+ getMetadata(): HolidayOfferMetadata
+ getDestination(): Location
+ getHolidayDuration(): DateRange
+ getPrice(): int
+ getAccommodationOffer(): AccommodationOffer
+ getTransportOffer(): TransportOffer
+ toOfferBean(): Offer
1
TransportOffer
- id: int
- companyName: String
- companyId: int
- quality: int
- numOfTravelers: int
- pricePerTraveler: int
+ setId(int id): void
+ setCompanyId(int id): void
+ getId(): int
+ getType(): TransportType
+ getCompanyName(): String
+ getCompanyId(): int
+ getQuality(): int
+ getNumOfTravelers(): int
+ getPricePerTraveler(): int
+ getRoute(): Route
+ getDepartureAndReturnDates(): DateRange
+ toTransportBean(): Transport
AccommodationChanges
<<interface>>
+ getId(): int
+ getType(): AccommodationType
+ getQuality(): int
+ getNumOfRooms(): int
+ toAccommodation(): Accommodation
TransportChanges
<<interface>>
+ getId(): int
+ getType(): TransportType
+ getQuality(): int
+ getNumOfTravelers(): int
+ getRoute(): Route
+ toTransport(): Transport

## 3.3 Activity diagram

This is the activity diagram of the “Request changes on offer” use case:

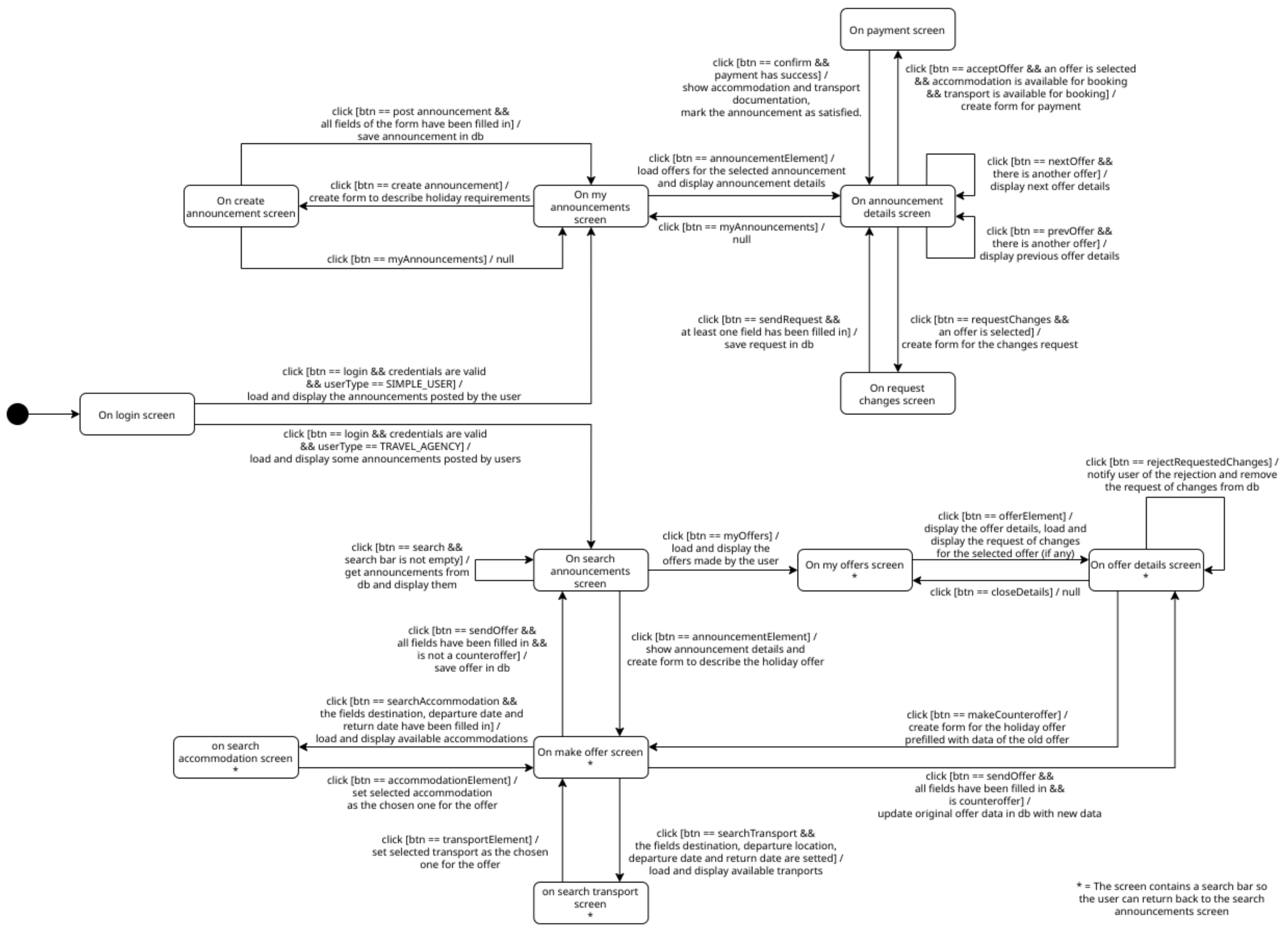

Activity diagram for:
Request changes
[ else ]
[ user has selected an offer* ]
Create form to describe changes in the offer*
4 seconds
Display offer*
User fills in fields of the form for which a change is required
User clicks the send request button
Display error message
Verify that at least one field of the form has been filled in
[ all fields are blank ]
[ else ]
Send requested changes to travel agency that made the offer*
Travel agency evaluates changes requested by user
[ travel agency rejects changes ]
[ travel agency makes counteroffer ]
Notify user
Make counteroffer


Activity diagram for:
Make counteroffer
Create form to describe the offer*
4 seconds
Display the original offer* and the changes requested by user
Travel agency fills in the form
Travel agency clicks the send offer button
Display error message
Verify that all fields of the form has been filled in
[ there is a blank field ]
[ else ]
Update original offer* with data inserted in the form
Notify user of the counteroffer

## 3.4 Sequence diagram

The sequence diagram is too big to be integrated into this document, you can find it in the diagram folder.

## 3.5 State diagram


On payment screen
click [btn == confirm && payment has success] / show accommodation and transport documentation, mark the announcement as satisfied.
click [btn == acceptOffer && an offer is selected && accommodation is available for booking && transport is available for booking] / create form for payment
click [btn == post announcement && all fields of the form have been filled in] / save announcement in db
click [btn == announcementElement] / load offers for the selected announcement and display announcement details
click [btn == nextOffer && there is another offer] / display next offer details
On create announcement screen
click [btn == create announcement] / create form to describe holiday requirements
On my announcements screen
On announcement details screen
click [btn == myAnnouncements] / null
click [btn == prevOffer && there is another offer] / display previous offer details
click [btn == myAnnouncements] / null
click [btn == sendRequest && at least one field has been filled in] / save request in db
click [btn == requestChanges && an offer is selected] / create form for the changes request
click [btn == login && credentials are valid && userType == SIMPLE_USER] / load and display the announcements posted by the user
On request changes screen
On login screen
click [btn == login && credentials are valid && userType == TRAVEL_AGENCY] / load and display some announcements posted by users
click [btn == rejectRequestedChanges] / notify user of the rejection and remove the request of changes from db
click [btn == offerElement] / display the offer details, load and display the request of changes for the selected offer (if any)
click [btn == search && search bar is not empty] / get announcements from db and display them
On search announcements screen
click [btn == myOffers] / load and display the offers made by the user
On my offers screen *
On offer details screen *
click [btn == closeDetails] / null
click [btn == sendOffer && all fields have been filled in && is not a counteroffer] / save offer in db
click [btn == announcementElement] / show announcement details and create form to describe the holiday offer
click [btn == searchAccommodation && the fields destination, departure date and return date have been filled in] / load and display available accommodations
click [btn == makeCounteroffer] / create form for the holiday offer prefilled with data of the old offer
on search accommodation screen *
On make offer screen *
click [btn == accommodationElement] / set selected accommodation as the chosen one for the offer
click [btn == sendOffer && all fields have been filled in && is counteroffer] / update original offer data in db with new data
click [btn == transportElement] / set selected transport as the chosen one for the offer
click [btn == searchTransport && the fields destination, departure location, departure date and return date are setted] / load and display available tranports
on search transport screen *
* = The screen contains a search bar so the user can return back to the search announcements screen

# 4 Testing

As requested three test classes have been implemented:

- TestChangesRequestFactory: this class is used to test the correct behavior of the ChangesRequestFactory and implements two test cases.
  In the first test case, we try to build a request of changes in which the agency offer and the desired offer are the same instances; this should result in an

exception being thrown.
In the second test case, we try to build a valid request of changes using two different holiday offers; in this case the creation should succeed.

- TestAnnouncementManager: this class is used to test the insertion of an announcement into the system.
  It creates a dummy announcement and uses the AnnouncementManager (controller class) to insert it into the system; to do so it uses a test account.
  After the insertion the AnnouncementManager is used again to obtain a list of all the announcements posted by the test user and the existence of the dummy announcement in the returned list gets verified.

- TestOfferManager: this class is used to test the insertion of an offer into the system.
  It creates a dummy offer addressed to the dummy announcement posted by the TestAnnouncementManager test case and uses the OfferManager (controller class) to insert it into the system; to do so it uses a test account.
  After the insertion the OfferManager is used again to obtain a list of all the offers made by the test travel agency and the existence of the dummy offer in the returned list gets verified.

# 5 Code

The source code of the project can be found at:
https://github.com/TeseiRoberto/ISPW-Project

## 5.1 Exceptions

Exceptions have been widely used across the system, in particular:

- IllegalArgumentExceptions are thrown to signal that the parameters given to a method are not valid.

- IllegalCallerExceptions are thrown by logic controllers to signal that the user who is requesting an operation does not have permission to execute that operation.

Along with the exceptions already defined by Java we have declared two custom exception types:

- DbException: this exception is used to signal that some error occurred in the persistency layer.

- ApiException: this exception has been used to signal that an API has generated some error while performing some operation.

## 5.2 Sonarcloud

To ensure the quality of the developed system, Sonarcloud has been used.
The results of code analyses can be found at the following link:
https://sonarcloud.io/project/overview?id=TeseiRoberto_ISPW-Project