AIR Reservations

Technical Document

Description

Is a booking application to reserve flights.
A logged user can search the flights, pay, receive the tickets, cancel the reservation, choose the seat of the flight, checkout the flight and print the boarding pass.
A staff user can manage the tickets of the user.
The application manages the notifications through email and pdf attachments

Users and groups

The application creates demo users for traveler and staff groups so the web app is ready to start.
This users are generated using the activiti framework described in the architecture paragraph.
Here the operations of the users groups:

TRAVELER
- searches the flights
- adds the flights to the shopping cart
- pay
- do checkout
- cancel the flight
- print the boarding pass

STAFF
- read the payed tickets
- cancel the current checkout tickets

Models

The model is pretty simple represented by 4 classes:

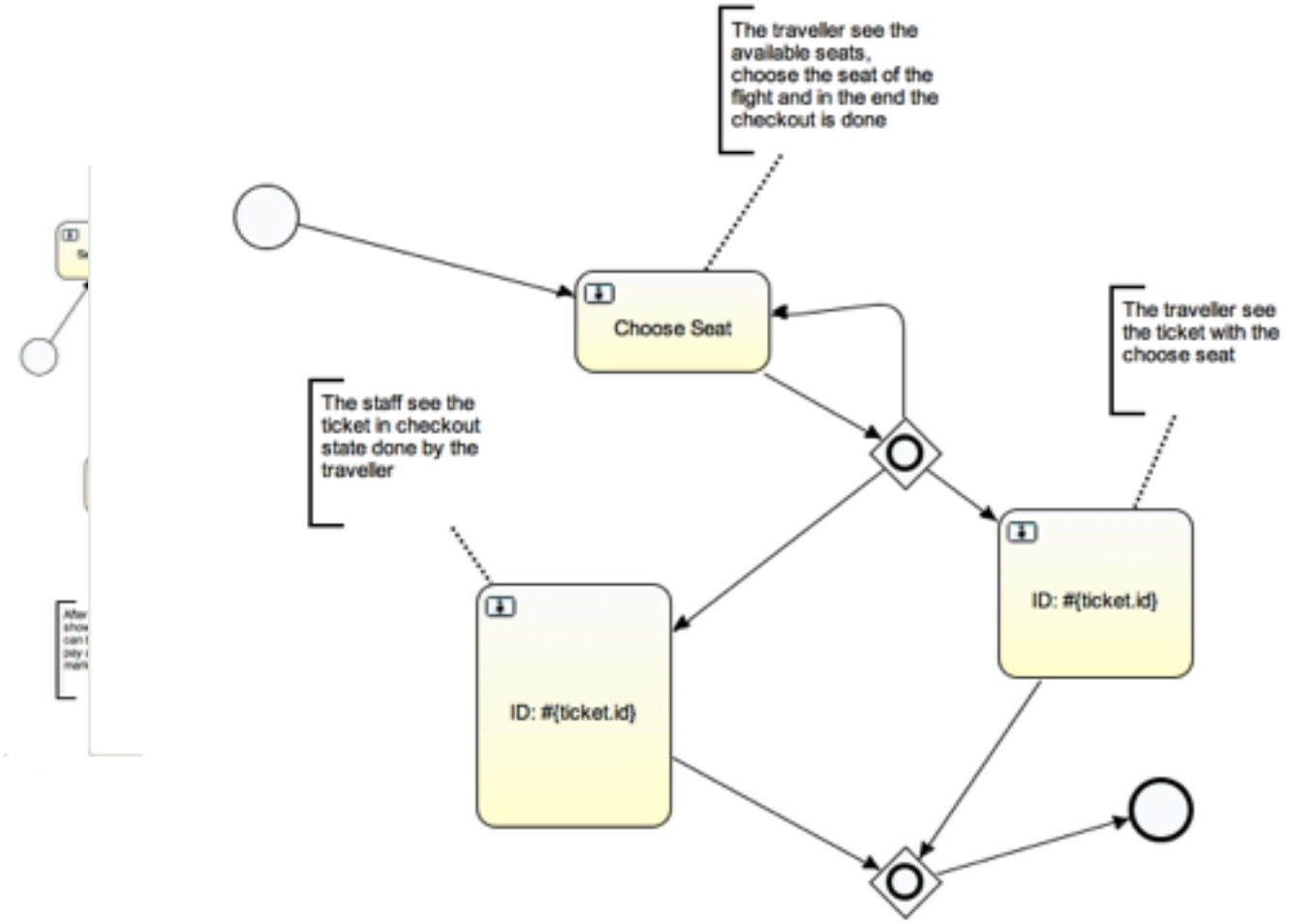
Address - The address of the flight. It specify city, state, and airport name
Flight - The flight. It can be moved on 5 different states:
    STARTED - The flight is ready to choice
    REQUESTED - The flight is choose by the user
    ALERTED - The flight is notified to the user if the checkout is not still done
    CHECKOUT - The checkout of the flight is done
    CANCELED - The flight is canceled by the user or by the scheduler
Payment - The payment. It contains all the choose flights by the user, the info of the payment of the flights. It contains the credit card info, the choose flight and the sum of the prizes to pay
Ticket - When the user pay the flights, the tickets are created. It generates an id and it represents a single flight

Use cases

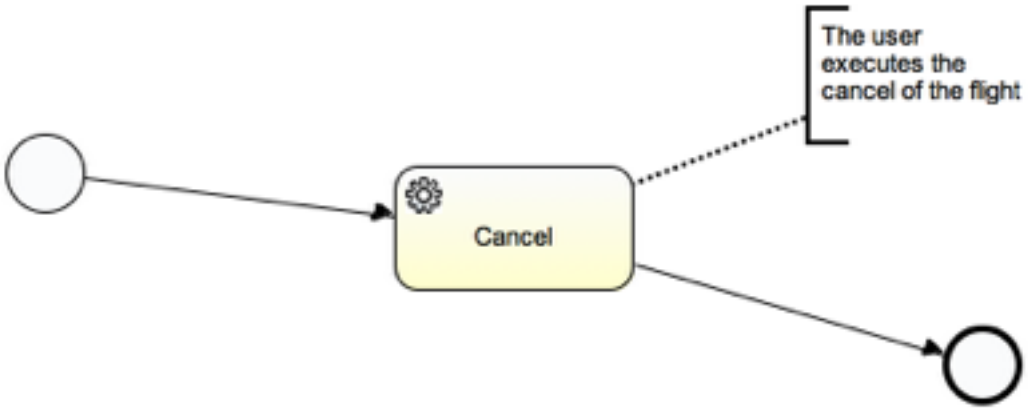
4 use cases to represent the entire flow of the application.

Reservations - It starts with the search of the flight by the traveler and it ends with the ticket receipt sent by email

Scheduler - It’s a cron that delete all expired tickets and send mail alerts to the travelers



Checkout - It starts with the choice of seat by the traveler and when the checkout is done, he receive a receipt where he can print the boarding pass and a task go to the staff, so it can manage the ticket informations



Cancel - The refund operation by the traveler

Architecture

Activiti is a workflow framework. With it I can design the use cases and automatically the follow can be executed.
The use cases seen before are described with Activiti the the BPMN language. The flows create the user tasks used by the traveler and staff users.
Samples of task are the choice of the flight, the search, the choice of the seat, the checkout or the cancel operation.
Each task need a java action that describes the operation to do.
Mail tasks are described in the flows and let to send email to the desired user.
The web app is ready because Activiti has an own web app that simply reads the tasks according the logged user.

Here the java packages of the application inside the src/main/java folder of the project:

it.vige.reservations: Main utilities for the project
it.vige.reservations.model: the models described in the models paragraph
it.vige.reservations.bpm: the bpmn actions assigned to the tasks. These actions model the models. See the javadocs built through maven for details. The resume of the project show how create them.
org.activiti.explorer: the customization of the activiti web app. Only need to filter functionalities not requested for the application

Here the test java packages of the application inside the src/test/java folder of the project:

it.vige.reservations.test: All unit test cases. See the javadocs for details

Here the resources of the application inside the src/main/resources folder of the project:

engine.properties: Used to customize the mail client parameters as the hostname and the port. Actually it points to the local mail server started after the start of the application
bpm: The folder containing the flows written through activiti and executed in the application
org/activiti/explorer/images: images for the three test user profiles of the application

Here the test resources of the application inside the src/test/resources folder of the project:

activiti.cfg-mem.xml: teh activiti config file used by the unit tests to start the database and the activiti services. The web app doesn't need this file because is inside the activiti web app

The project is managed by maven (see the third party libraries paragraph for detail) through the file:
pom.xml: in this file are described the third party dependencies

When the web app starts a database is automatically created. The login is done through database

Third party libraries

activiti: (http://www.activiti.org) the framework for the workflows. See the Architecture paragraph for details

vaadin: (http://www.vaadin.org) the framework to write web apps. Used inside the activity web app. Used in air reservation to customize the pages of the activity web app

hypersonic: (http://www.hsqldb.org) the default database of the application.

subethamail: (http://www.subethamail.org) used as mail server to send the mails to the users.

junit: (http://www.junit.org) library core to write the unit tests.

itextpdf: (http://www.itextpdf.org) used to create the pdf documents as the ticket receipt and the boarding pass

jetty: (http://www.eclipse.org/jetty) the servlet engine where is built the application

maven: (http://www.maven.org) all the project is built through maven. Maven let to compile, start and create documentations for the project