Air Ticket Reservation System

Architecture and Design

Scope: Crossover Project Assignment

Author: Dino Lupo

e-mail: dino.lupo@gmail.com

1. ARCHITECTURE AND DESIGN

1.1. Scope

This is the Design Document required by the project assignment. This document includes the following elements:

- High level requirement analysis
- High level presentation of the data model
- Architecture diagrams describing the composition and working of the system, explaining the component interaction and process, control and data flows.
- Explain the breakdown of the system into components with technical implementation details of each component, along with the design patterns involved (with reasons that justify your choices).
- Present the UI screens and how they are connected through user interactions with those screens
- Use both visual elements (diagrams) and text descriptions to maximize the amount of information conveyed, while keeping the document as compact as possible.
- Diagrams in any format (like UML) are not mandatory but nice to have, it is upto you how best to depict the logical design.
- Create the document as a single office file (doc, pdf, etc), or a set of linked files where only one needs to be opened (index.html, etc).

2. CROSSOVER REQUIREMENTS

2.1. Functional Requirements

Here is a summary of the functional requirements for the system:

1.1.1. Public users

REQ-1. Are able to login using their google+, twitter or LinkedIn accounts. Either one is enough.

REQ-2. Are able to reserve air tickets searching from multiple flights based on multiple criteria.

REQ-3. Can make payment using credit cards.

REQ-4. Receive tickets as PDF via email. This email should contain useful links like "My Bookings", "Checkout Now", "Cancel booking" that will not require the user to login.

REQ-5. Are able to perform online checkout/cancellation, from 48 hrs and upto 4 hrs from departure, choose seats, and print boarding passes.

REQ-6. Receive email alert at 48 hrs from departure, regarding their reservation, to encourage them for early online checkout. This email should contain a "Checkout Now" link that will not require the user to login.

1.1.2. Airline staff

REQ-7. Are able to login using their internal credentials, even from public computers.

REQ-8. Receive reservation chart as PDF at 1 hrs from departure.

REQ-9. Cancel any ticket, generating email alert and refund to the public user.

REQ-10. Cancel the whole flight, generating email alerts and refunds to the public users.

Assume any other functionality that may be necessary to achieve the above requirements based on logic and your experience. In case you think there are details missing, think of this as skill assessment, where you can assume details on your own.

2.2. Other Technical and Non-functional Requirements

REQ-11. Must create a Single-page Application (SPA) based web portal with responsive design.

REQ-12. System core must be ready to serve mobile apps in future w/o any modification.

REQ-13. System will be installed on cloud with services that you recommend enabled.

REQ-14. All the alerts sent should be guaranteed (not lost due to some technical errors).

REQ-15. Technologies are not recommended as it is a design decision you should make as per the position applied for

3. HIGH LEVEL REQUIREMENT ANALYSIS

3.1. Functional Requirements Analysis for Public Users

1.1.3. REQ-1 - Are able to login using their google+, twitter or LinkedIn accounts. Either one is enough.

The requirement states that that the user shall be authenticated to access the platform. This means that the public user functionalities of the system are not accessible if the user has not logged in into one of the platform listed before.

1.1.4. REQ-2 - Are able to reserve air tickets searching from multiple flights based on multiple criteria.

The user will be able to access a Search area on the page where there are multiple search fields like the following:

- Type: Round-Trip or One-Way
- From (mandatory)
- To (mandatory)
- Departure date (mandatory)
- Return date (mandatory only if it is Round-Trip)
- Cabin (mandatory, select from the following values: Economy, Premium Economy, Business, First)
- Number of seats for each of the following categories (mandatory, default 1 Adult seat):
    - Adults (18-64)
    - Seniors (65+)
    - Youth (12-17)

- Child (2-11)
- Seat Infant (under 2)
- Lap Infant (under 2)

User can select an option to show Flexible Dates and select the extra information:

+/- days: number of days before and after the selected date (Departure Date and Return Date), searched as a date interval.

The Search button will trigger the search and the screen will show a list of all the available flights that match the selected criteria.

The Flight list shown will be orderable (ascending and descending) by:

- Price
- Departure takeoff date/time
- Departure landing date/time
- Return takeoff date/time
- Return landing date/time

It should be possible to select/deselect some values after the search like:

- Airline Carrier
- Airports (if the city has multiple airports it can be useful to select only one or some)
- Price Range
- Restrict the Date Range of Departure dates / Return dates

1.1.5. REQ-3 - Can make payment using credit cards.

The user can pay for flights with the following methods:

- Credit Card / Debit Card
- PayPal

1.1.6. REQ-4 - Receive tickets as PDF via email. This email should contain useful links like "My Bookings", "Checkout Now", "Cancel booking" that will not require the user to login.

This requirement implies that some web resources should be accessed without authentication. The e-mail should contain links to those public resources. Those resources should not be indexed by search engines and should contain a unique id to the public resource, for example:

https://webticketing.crossover.com/open?id=0BxfVNwqExgYROWJmYWNjMzEtOTUxZi00M2ZjLWI5ZTctZTYxNzNkNjRiMGFl

1.1.7. REQ-5 - Are able to perform online checkout/cancellation, from 48 hrs and upto 4 hrs from departure, choose seats, and print boarding passes.

Users can do certain operations in a time interval from 48 hrs upto 4 hrs from departure:

- Choose seats
- Checkout
- Cancellation
- Print boarding passes

1.1.8. REQ-6 - Receive email alert at 48 hrs from departure, regarding their reservation, to encourage them for early online checkout. This email should contain a "Checkout Now" link that will not require the user to login.

The requirement need a timer that checks (e.g. every minute) for user reservations and send emails with links to the user to encourage for web checkout.

3.2. Functional Requirements Analysis for Airline Staff

1.1.9. REQ-7 - Are able to login using their internal credentials, even from public computers.

The back office portal must be exposed on Internet and should be secured using HTTPS and authentication.

1.1.10. REQ-8 - Receive reservation chart as PDF at 1 hrs from departure.

The requirement need a timer that checks (e.g. every minute) for flights and send emails with attached PDF of the flight details.

1.1.11. REQ-9 - Cancel any ticket, generating email alert and refund to the public user.

The airline staff can cancel tickets. This operation should automatically send email to the customer and start the refund operation with the banking provider.

1.1.12. REQ-10 - Cancel the whole flight, generating email alerts and refunds to the public users.

This is a special case of cancellation requirement where an entire flight is cancelled and emails and refunds are generated for all the customers of that flight.

3.3. Other Technical and Non-functional Requirements

1.1.13. REQ-11 - Must create a Single-page Application (SPA) based web portal with responsive design.

The web application user interface should be adaptable on different devices/resolutions without any drawbacks. The web application should give the impression of a UI of a standard client-server application. This means that navigating between links and buttons of the application should not trigger an entire refresh of the page. To be clear, if the application

template contains elements such as Header, Messages, Navigation Bar, Main Content and Footer, then, navigating through the function of the navigation bar, only the required contents should be updated asynchronously. Take for example the following template page:

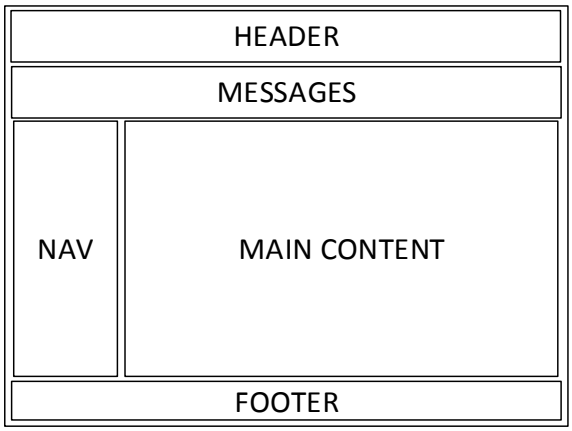


FIGURE 1 - SAMPLE APPLICATION TEMPLATE STRUCTURE

Selecting some functions into the left navigation bar (NAV), should update only the Main Content section. And if any message should appear in respond to some user action in the Main Content section, only the Messages section should appear.

1.1.14. REQ-12 - System core must be ready to serve mobile apps in future w/o any modification.

All the backend services must be implemented with REST based APIs.

1.1.15. REQ-13 - System will be installed on cloud with services that you recommend enabled.

The system must be ready to be deployed on common cloud providers such as AWS.

1.1.16. REQ-14 - All the alerts sent should be guaranteed (not lost due to some technical errors).

The system must have a guaranteed delivery notification system to ensure that all email alerts are delivered.

1.1.17. REQ-15 - Technologies are not recommended as it is a design decision you should make as per the position applied for

This project is inherently enterprise. For this reason, standard enterprise technologies have to be used to ensure maintainability and robustness over time. At the same time, it should use modern technologies to support mobile applications, nice UI/UX on the frontend and responsive design.

4. HIGH LEVEL PRESENTATION OF THE DATA MODEL

We are going to use an object oriented data model for the data layer(with JPA, Java EE 7 platform), so we show some of the relationship between Entities in the following diagram:

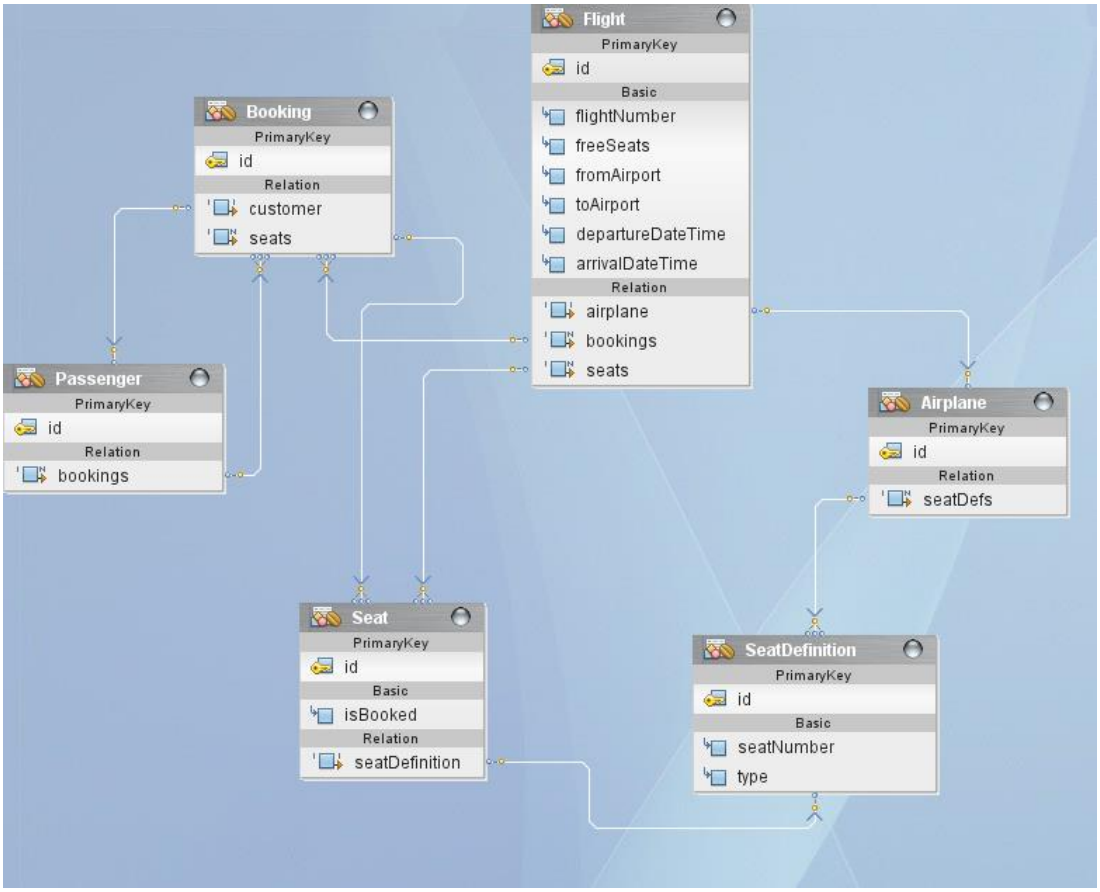


FIGURE 2 - DATA MODEL – FLIGHTS AND RESERVATIONS

This is the high level (and simplified version) of the data model for the system. Some basic attributes and the most important relations between entities are shown in the figure. We are assuming only the section related to the reservation and flights module. No entities and relations with PDFs, eMails and CreditCard modules are shown. Let's explain some entities in more detail:

4.1. Airplane

This entity represents the airplane that is connected to the Flight. It has some basic attributes like type, company, model, etc. and a relation with many SeatDefinition entitities.

4.2. SeatDefinition

This is a seat available in an airplane. Some of its basic attributes are seatNumber and type. This is like a template entity because, every time a new Flight is created into the system, for each SeatDefinition associated with the Airplane, it is created a Seat instance associated with the Flight. The Seat instance will act like a placeholder for a Booking instance.

4.3. Flight

In this simplified version of the Data Model, we assume that there are no Flight Legs. This means that a flight goes from one airport to another, while in the reality there can be multiple legs for each flight, usually 1 for a direct flight, 2 for a flight with a connection airport between origin and destination.

The Flight is connected to the following entities:

- one Airplane
- multiple Booking
- multiple Seats

Some basic properties are present on the entity, one worth mentioning is “freeSeats” that is a calculated property that stores the number of free seats on that flight, this can be useful when searching and showing flights without accessing connected entities that can be set to lazy instead of eager fetch type.

4.4. Seat

This entity represents an available seat for a Booking. It has some basic properties, one worth to mention is “isBooked” that is updated when a Booking is created and associated with a Seat.

4.5. Booking

Booking maintains all the information about the reservation. It is connected to one Passenger and to one or more Seats.

4.6. Passenger

The Passenger is connected to one or more Bookings.

5. SYSTEM MODULES AND DATA FLOW

In the following picture we show how the different modules collaborate to implement the requirements:

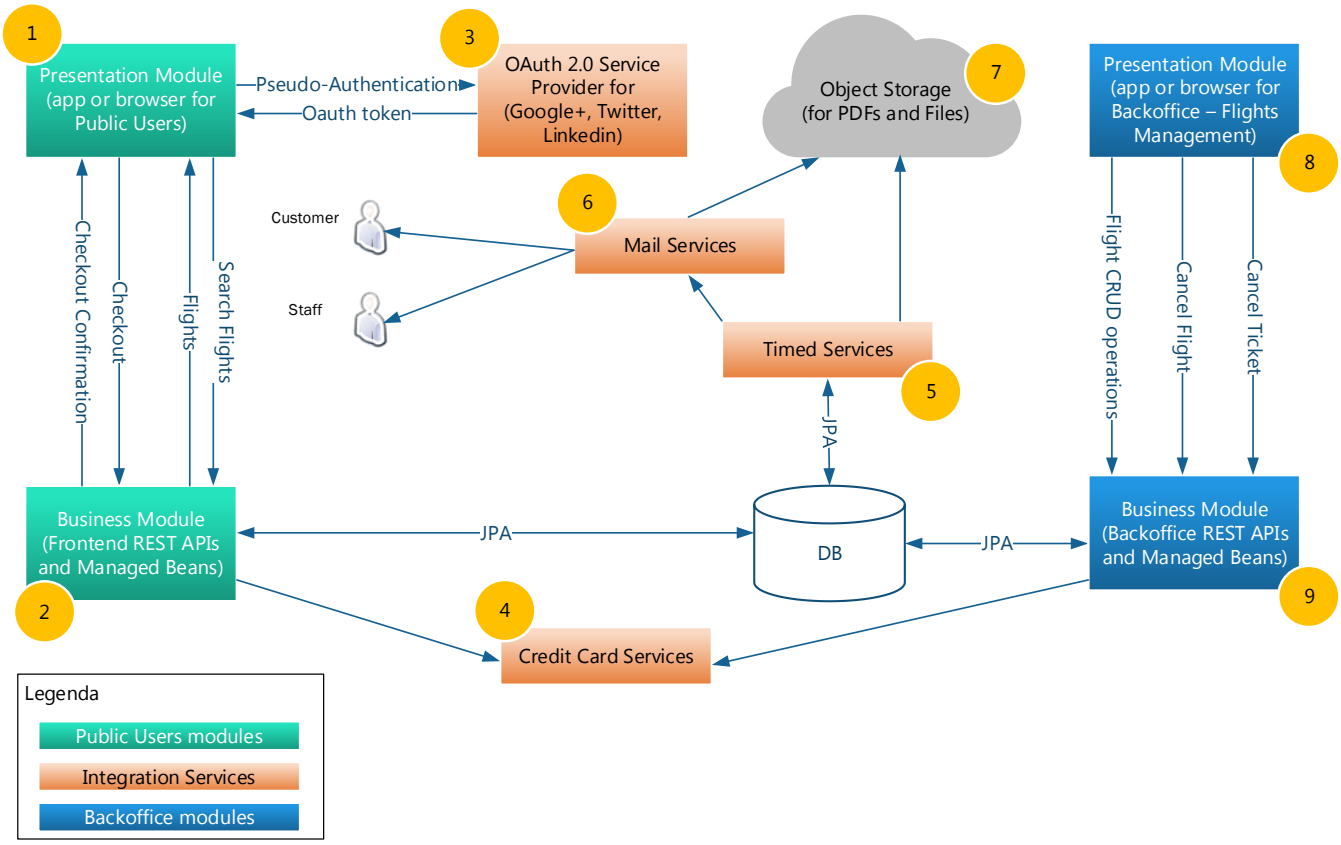


FIGURE 3 - DATA FLOW

1) Public Users frontend module is the presentation module that can be accessed in two ways:
    a. via an OAuth authorization token, the request is intercepted by a filter that protects resources and redirects to one of the OAuth Providers for login
    b. public resources with a unique id to access personal information like bookings or checkout

The Frontend module is built with JSF 2.2 for productivity and maintainability but can also be mixed or completely written with HTML5/Javascript/CSS3 technologies.

2) and 9) The Business modules are organized as follows:
    a. Protocol agnostic "Manager" classes that offer CRUD operations on Entities and other business related functions like Checkout/Cancellation, Search Flights, Cancel Booking or entire Flight, etc.
    b. Managed Bean for JSF pages that are a proxy versus the "Manager" classes.
    c. Resource classes to expose REST services that also act as a proxy versus the "Manager" classes.
    d. JPA Entities to store classes into the DB.
3) This module contains the filter and the necessary classes to implement OAuth security.
4) This module contains the classes for connection versus the Payment services.
5) Timed services contains all the TimerService (Java EE) beans needed to schedule operations like:
    a. sending alert emails using the Mail Services module
    b. enabling/disabling certain features like checkout before/after 48 hrs from the departure date and time, setting flags on Entities
    c. automatically generate flight reports or boarding passes in PDF and store them into the object storage cloud (like Amazon S3)
6) This module implements the mail system, using the JavaMail API that is needed to send emails and to ensure that the notifications are delivered even in the case of a technical problem.
7) This is an external object storage system, like Amazon S3, which stores all the necessary PDF files, like reservation chart and tickets. Files are generated offline by other modules.

8) This is the Presentation Module for Back Office functions. The security is implemented with:
    a. Application layer, using Declarative Security in deployment descriptors (web.xml).
    b. Transport layer, (HTTPS configuration).
    c. The Java EE container (JBoss EE or Wildfly 10.x) can be configured to use LDAP (compatible with AWS Directory Services) or Database (compatible with AWS RDS) for the login-module configuration.

6. TECHNICAL DETAILS OF THE PROPOSED SOLUTION

6.1. Project Structure

The solution is organized with the Boundary Control Entity pattern, where each sub-module is divided into three pieces (BCE).

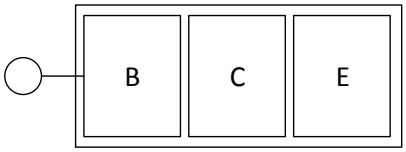


FIGURE 4 - BCE COMPONENT

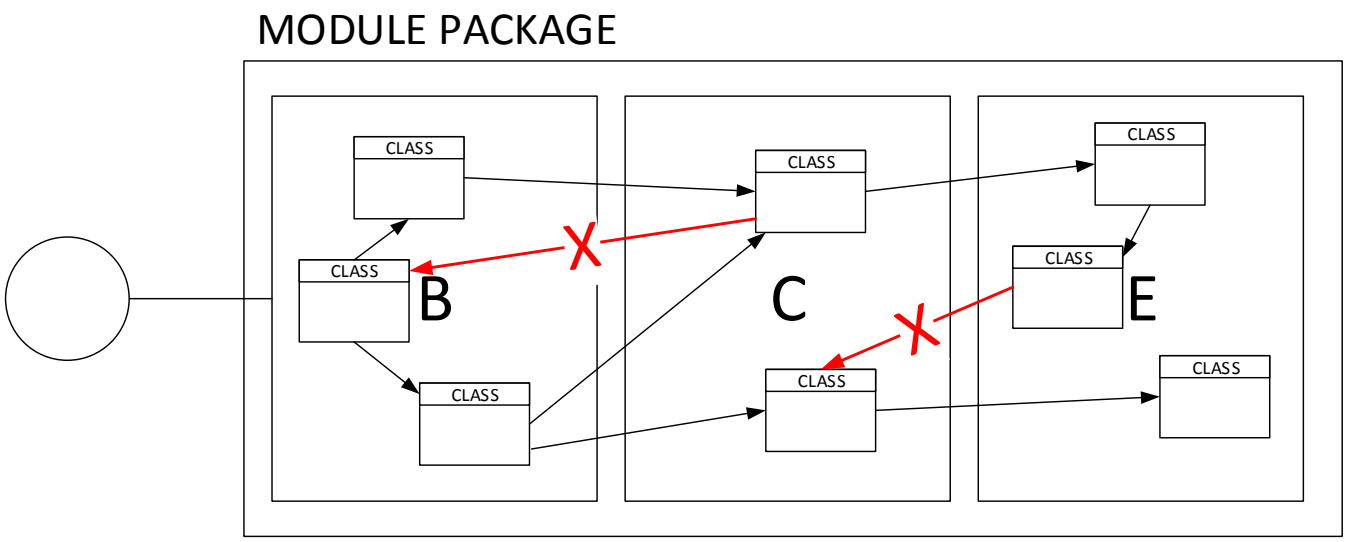
The component part type is divided into the following sub-packages:

Boundary contains all the classes that have all the interfaces for humans or other objects.

Control this optional package can contain classes with some logic like validation, complex tasks that cannot stay in the boundary package, etc.

Entity contains all the classes for persistence of entities

Each class can reference other classes in only in the same module type or versus the entity layer. It is not permitted to have references from the E classes to C or B classes and from the C classes to the B classes:



For package definition we can have the following convention:

```
<company name>.<application name>.<layer>.<component>.<type>
```

We want to separate Presentation from Business Logic and Integration classes, the `layer` part of the package path should be one of the following:

- presentation (layer)
- business (layer)
- integration (layer)

Our template project contains packages for both presentation and business layer, and the modules needed to manage Flights CRUD operations:

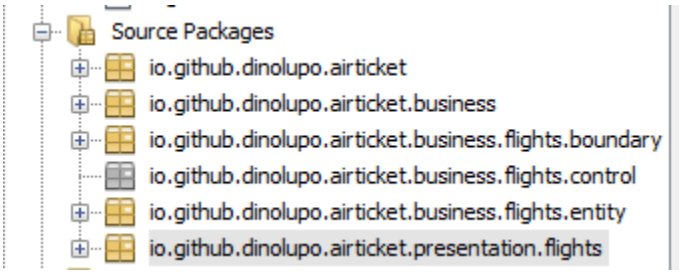


FIGURE 5 - PACKAGES

The application contains many components that have connections between each other, and every component is a Java Package that comprehends all the Boundary, Control and Entity packages. We can have references between classes of different modules, but we should avoid references between classes in the Boundary package of different modules:

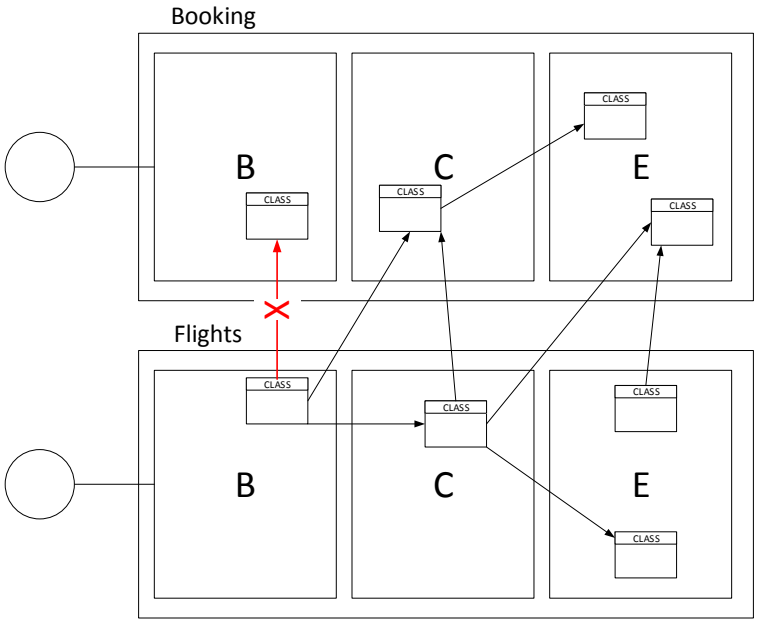


FIGURE 6 - REFERENCES BETWEEN MODULES

6.2. Separating The Concerns of A Boundary

Maintaining all the business behaviors in the Rest resource class can be complicated when you want to create unit tests. So we put all the behaviors in a separate class and inject an instance into the rest resource class. Given the example of Flight package, let's see how the classes are organized:

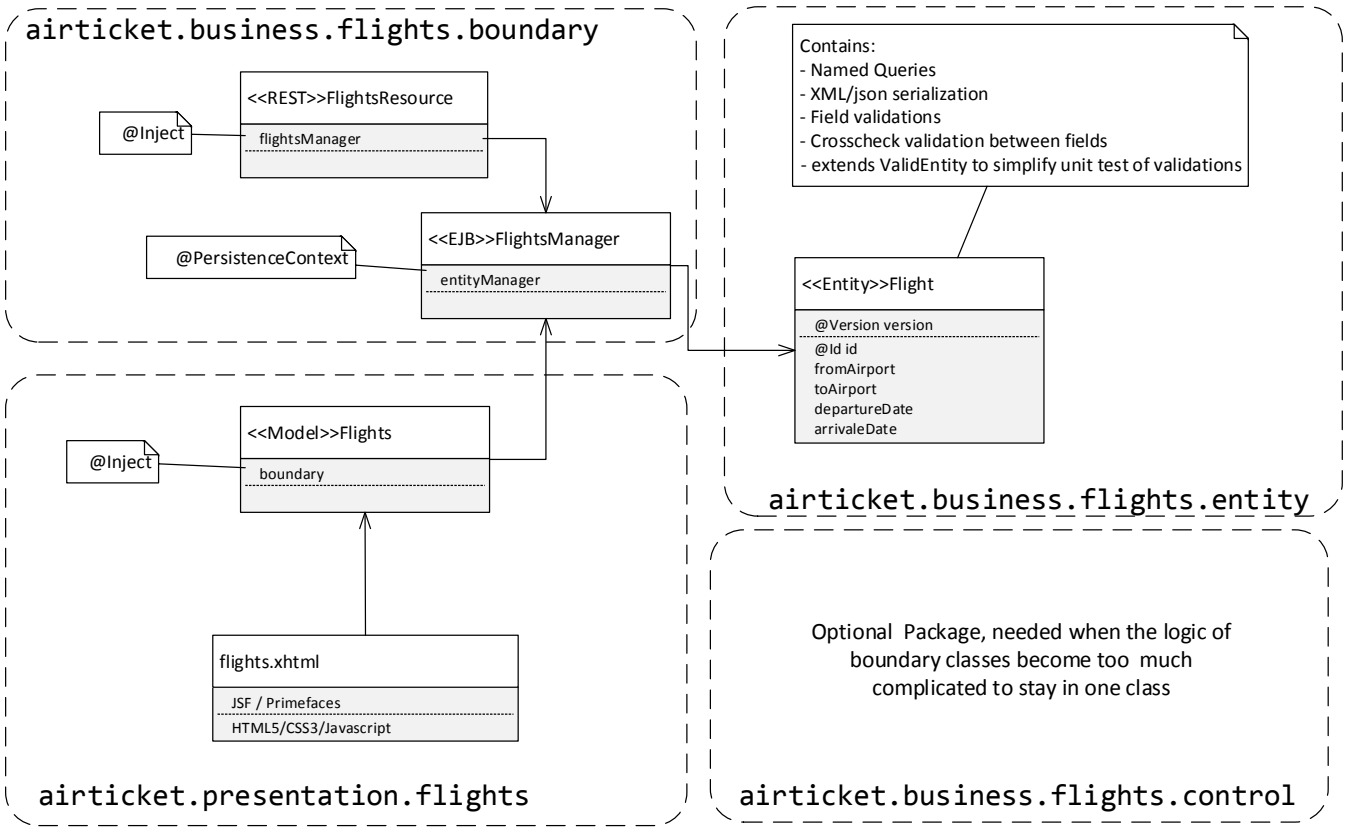


FIGURE 7 - CLASS DIAGRAM OF FLIGHTS BOUNDARY CLASSES

6.3. Timed Services - Configurable Timers

We can use Configurable Timers using the `TimerService` of the JEE Application Server. In the following class we create a sample `ScheduleExpression` that could take the configuration parameters from an external configuration file (e.g. properties). The class can be used as a template for sending emails as stated by the specifications:

```
@Startup
@ConcurrencyManagement(ConcurrencyManagementType.BEAN)
@Singleton
public class EmailAlertUsingTimer {

    @Inject
    EMailManager emailManager;

    @Resource
    TimerService timerService;
    Timer timer;
```

```
    @PostConstruct
    public void initialize(){
        ScheduleExpression scheduleExpression = new ScheduleExpression();
        scheduleExpression.second("*/3").minute("*").hour("*");
        this.timer = timerService.createCalendarTimer(scheduleExpression);
    }

    @Timeout
    public void sendAlertEmails(){
        System.out.printf("*************** Sending Alert Emails Job – timestamp: %s
**************\n", new Date());
        emailManager.sendEmails();
    }

}
```

6.4. i18n. Internationalization of both Validation Messages and JSF Strings

First of all, all localized messages are put in property files. Those files are defined in the WEB-INF/faces-config.xml file.
In the faces-config.xml we have basically two types of property files:

- **<message-bundle>**
- **<resource-bundle>**

In our example application, we defined the following:

```
WEB-INF/faces-config.xml
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
    <application>
        <resource-bundle>
            <base-name>resources</base-name>
            <var>i18n</var>
        </resource-bundle>
        <message-bundle>messages</message-bundle>
    </application>
</faces-config>
```

This means that we should have two files: one with name `resources.properties` and the other with name `messages.properties`. They are both in the src root, no package used here, so they are located in the folder `main/resources` of our project (this is a Maven Standard Directory)

1.1.18. `<resource-bundle>` tag explanation

The `<resource-bundle>` has to be used whenever you want to register a localized resource bundle which is available throughout the entire JSF application without the need to specify `<f:loadBundle>` in every single view.

resources.properties content

```
index.caption=Caption
index.description=Description
index.priority=Priority
```

The resource bundle has a `<var>` tag, that identifies the variable name that can be used in the JSF pages of our application.
For example let's suppose we want to internationalize the label of the description field, we put a `outputLabel` with the value referring to the property `index.description` in our file `resources.properties`. We also have almost the same value for the label `property` of `inputText`:

```
<p:outputLabel for="description" value="#{i18n['index.description']}: "/>
<p:inputText id="description" value="#{index.todo.description}"
label="#{i18n['index.description']}"/>
```

The label will be used also by the validation messages generated by JSF (see description of message-bundle)

1.1.19. `<message-bundle>` tag explanation

The `<message-bundle>` has to be used whenever you want to override JSF default warning/error messages which is been used by the JSF validation/conversion stuff. Keys of the default warning/error messages are defined in chapter 2.5.2.4 of the JSF specification.
In our case, in `messages.properties` file we should put all the localized validation messages, so I decided to put here the following:

messages.properties content

```
javax.faces.validator.BeanValidator.MESSAGE = {1}: {0}
validation.flight.crosscheckfailed=Departure date must be before than arrival date.
```

`javax.faces.validator.BeanValidator.MESSAGE` is necessary because when bean annotated validators fails, the default message does not contain field information see here

To be consistent, we should i18n also the cross check validation messages:

1) Use localized message key when using cross-check validator:

```
...
@CrossCheck(message = "validation.flight.crosscheckfailed")
public class ToDo implements ValidEntity {...}
```

2) Modify the message shown using FacesMessage reading a property from the message bundle:

```
...
@Model
public class Flights  {
...
    public void showValidationError(String content) {
        FacesContext context = FacesContext.getCurrentInstance();
        String msgBundle = context.getApplication().getMessageBundle();
        Locale locale = context.getViewRoot().getLocale();
        ResourceBundle messageBundle = ResourceBundle.getBundle(msgBundle,locale);
        FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_WARN,
                messageBundle.getString(content), content);
        context.addMessage(null, message);
    }
...
}
```

7. GOOGLE OAUTH 2.0 AUTHENTICATION

https://developers.google.com/api-client-library/java/google-oauth-java-client/oauth2

8. REQUIRED SERVICES FOR THE CLOUD DEPLOYMENT ON AMAZON WEB SERVICES

After configuring a suitable server capable to run Java 1.8 and Java EE Application Servers, we should enable the following services:

- AWS Directory Services (if LDAP is already present in the organization)
- AWS RDS for PostgreSQL (for the database)
- AWS S3 for Object Storage (to store files like PDFs)