Politecnico di Milano
M. Sc. Course in IT Engineering

Software Engineering 2 – Internal Project
Prof. Elisabetta di Nitto – a.y. 2013/2014

Acceptance Testing Document

VERSION 1

by
Edoardo Mondoni (816283)

Table of contents

Index of pictures

Index of tables

Index of test cases

Preamble

This is the Acceptance Testing Document for the Software Engineering 2 2013 internal project. It shows the results of the acceptance testing phase, which has been carried out on the analogous project by Christian Crisciullo, Francesco Maria Filipazzi and Giuseppe Taverna (located at `https://code.google.com/p/travel-dream-crisciullo-filipazzi-taverna/`). This task has been accomplished based on the following material:

- the Requirements Analysis and Specification Document (`RASD-crisciullo-filipazzi-taverna.pdf`);
- the Design Document (`DD-crisciullo-filipazzi-taverna.pdf`);
- the Install Guide (`Travel Dream Installation.pdf`);
- the User Guide (`UserGuide.pdf`);
- the executable package (`TravelDreamSystem.ear`).

Please note that the implementation-related deliverables have been extracted from the `TravelDream_Crisciullo_Filipazzi_Taverna_ERRATACORRIGE.zip` file, and are thus the latest versions available.

The remainder of the document is organized as follows:

- **Section 1** reports on the installation process and on the issues encountered during the application's deployment;
- **Section 2** details the testing conducted on the system, including a summary of the preparatory study of the accompanying documents.

1 Installation

The software has been deployed following the instructions in the Install Guide. The operating environment conformed to the prescriptions stated in the *Prerequisites* section of said document.

1.1 Preliminary measures and deployment

No constraint being expressed for what concerns the database settings, a new schema and a separate user were created for this testing job. The details are listed in Table 1.

Database settings	
Username	`'TDtesting'@'localhost'`
Password	`test`
Schema	`TravelDreamTesting`

TABLE 1: DATABASE SETTINGS FOR THE TESTING ENVIRONMENT

According to the directions in the guide, the EAR package was then imported into Eclipse IDE, marking the `TravelDreamSystemEJB` and `TravelDreamSystemWeb` JARs to be imported as separate projects rather than as Utility JARs. The MySQL Connector/J was then added to the web project's build path.

The Java Persistence API was then configured for the EJB project by enabling the relative facet, as is indicated in the Install Guide. Configuring the persistence.xml file did not cause any problems either, since the *Generate Tables from Entities…* command terminated gracefully and without errors as expected.

The creation of the first administrator was carried out according to the guide's instructions: a manual INSERT SQL statement was executed to add a staff account with username `testadmin`, password `password` and the `PERMISSION` flag set.

At this point, however, the system was found to be not ready for deployment: as is clear from Picture 1, nearly all import declarations failed and caused Eclipse's validators to return compilation error warnings. Further investigation on the code revealed the need to include the EJB project into the Web project's class path, since a lot of `import` declaration reference classes and packages located outside the Web project. Correcting the class path resulted, in fact, in all compilation error warnings disappearing.

1.2 Post-deployment issues

After deploying the software on Glassfish, other issues were encountered while trying to access the web application's homepage. Simply typing `http://localhost:8080/TravelDreamSystemWeb` into the browser's address bar resulted, in fact, in a HTTP 500 error. More precisely, Glassfish raised a `FacesFileNotFoundException` with the following message:

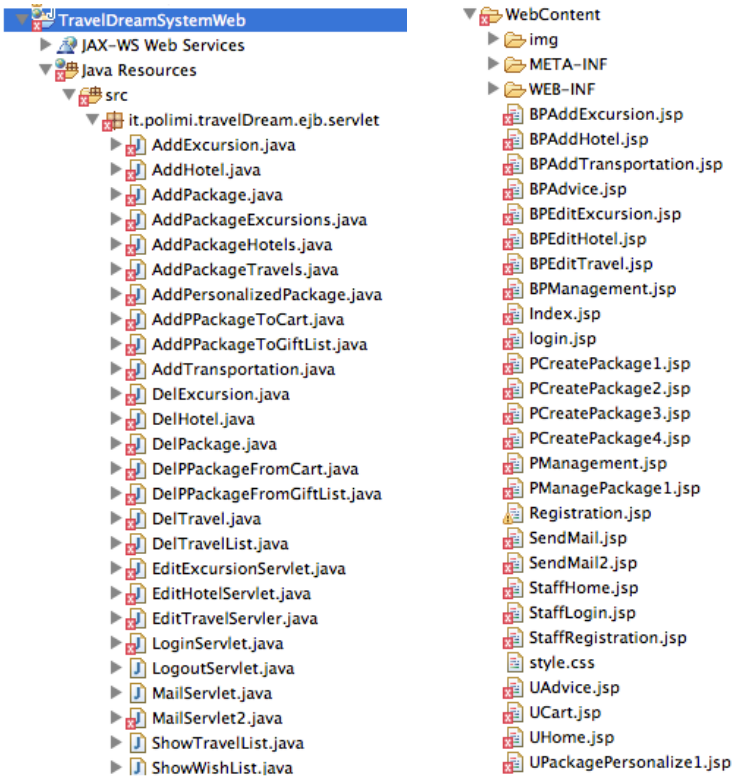
```
com.sun.faces.context.FacesFileNotFoundException: /index.xhtml Not Found in
ExternalContext as a Resource
```

Mindful of a similar complication emerging during the implementation phase, a quick inspection of the `/WEB-INF/web.xml` file (Code snippet 1) showed that the `welcome-file` tag was reporting an incorrect filename for the homepage. Editing said entry to `Index.jsp` (which might as well have been `login.jsp` and had to be guessed, since no indication was to be found in any of the provided documents) did in fact solve the issue.

```
<welcome-file-list>
    <welcome-file>index.xhtml</welcome-file>
</welcome-file-list>
```

CODE SNIPPET 1: INCORRECT SETTINGS IN THE WEB.XML FILE

Being the application's user interface finally accessible, the installation was deemed successful.



PICTURE 1: GETTING THE SOFTWARE TO WORK REQUIRED AN ADDITIONAL STEP

1.3 Conclusions

The install guide did not contain wrong or ambiguous instructions, but it fell short of informing the user about:

- the need to include the EJB project in the Web project's class path;
- the exact URL of the website's homepage.

The latter piece of information would have been redundant (though equally welcome) if the `web.xml` file had contained the right setting.

2 Test cases

Once the application was correctly installed, the system has been thoroughly tested to verify the achievement of its goals and the correct implementation of its functions. This phase of the testing process could not be carried out without a careful reading of the preliminary documents accompanying the software – i.e. the Requirements Analysis and Specification Document and the Design Document – and, needless to say, of the User Guide.

2.1 Testing context

In order to proceed with a systematic test of the system, particular attention was paid to the parts of the aforementioned documents reporting lists of its goals and of its functionalities. What follows is a summary of the development team's findings on that matter.

2.1.1 Goals of the application

The developers defined a set of nine goals for their software in Section 1.4 of the Requirements Analysis and Specification Document:

- **[G1]** Registered users can buy a package at any time.
- **[G2]** Registered users can join another user's list at any time.
- **[G3]** Registered users can compose a list or a gift list at any time.
- **[G4]** Registered users can invite other users to see a package.
- **[G5]** Registered users can invite other users to a gift list at any time.
- **[G6]** Guests can view their invitations at any time.
- **[G7]** Employees can add hotels, flights and excursions at any time.
- **[G8]** Employees can compose packages at any time.
- **[G9]** System administrators can add employee accounts at any time.

2.1.2 Actors of the application

Subsection 3.1.3 of the Requirements Analysis and Specification Document reports a clear characterization of the system's actors.

- **Guests** are all those visiting the web site for the first time or registered customers who are not logged into the system yet.
- **Customers** are logged people who registered to the system with the intention of using its functionalities.
- **Employees** have a job at *TravelDream* and are in charge of managing basic products and travel packages.
- **System administrators** are responsible for the management of the system and of the accounts it contains.

Some of the above descriptions were mildly altered to remove minor conflicts with the goals listed in Paragraph 2.1.1. For example, the original description for employees read: "[They are] workers of *TravelDream* that insert basic products and compose travel packages". This could have been considered inconsistent with goal [G7] and has thus been harmonized with said statement.

2.1.3 Functionalities of the application

For what concerns the application's functionalities, two lists can be found in Subsection 3.1.2 of the Requirements Analysis and Specification Document and in Section 2.1 of the Design Document. The first one classifies those functions by performing actor, while the second one groups them according to a logical criterion.

Categorizing the functionalities according to the actor(s) performing them seems a preferable solution. Also, the list in the Design Document does not fully overlap with the one in the Requirements Analysis and Specification Document: some functions, in fact, are missing (customers can "browse and search through offered packages", but this is nowhere to be found in the Design Document) or their performing actors are not clear (F04 states "Consult a user profile", but who can do so? And again, customers can "receive invitations to a friend's gift list" and "visualize that gift list", but doesn't that conflict with a guest's ability to receive invitations and see their objects?). As a result, an enumeration of the application's functionalities – grouped by actor – was rebuilt based on both lists.

Functionalities available to unregistered users only

- **[F1]** Register to the system as customers.

Functionalities available to registered users only

- **[F2]** Login to the system.
- **[F3]** Logout from the system.

Functionalities available to both registered and unregistered users

- **[F4]** Receive invitations to other people's travel packages.
- **[F5]** View a travel package for which an invitation was received.

Functionalities available to customers

- **[F6]** Browse and search through the packages offered by the company.
- **[F7]** Personalize a package offered by the company in one's own travel list.
- **[F8]** Add a package to one's own travel list.
- **[F9]** Remove a package from one's own travel list.
- **[F10]** Add a package to one's own gift list.
- **[F11]** Remove a package from one's own gift list.
- **[F12]** Add a package to one's own cart.
- **[F13]** Remove a package from one's own cart.
- **[F14]** Join a package in a friend's travel list for which an invitation was received.
- **[F15]** Send an invitation to one's own travel list to a friend.
- **[F16]** Send an invitation to one's own gift list to a friend.
- **[F17]** Buy a package from one's own cart.
- **[F18]** Buy a package from another customer's gift list for which an invitation was received.
- **[F19]** Buy a package in another customer's travel list that was previously joined.

Functionalities available to employees

- **[F20]** Create a basic product.
- **[F21]** Edit a basic product.
- **[F22]** Delete a basic product.
- **[F23]** Compose basic products into a pre-defined travel package.
- **[F24]** Edit a pre-defined travel package.
- **[F25]** Delete a pre-defined travel package.

Functionalities available to system administrators

- **[F26]** Add employee accounts.
- **[F27]** Edit employee accounts.
- **[F28]** Delete employee accounts.

The following table reports the source for each one of the functionalities listed above. It indicates whether the function has been borrowed from the list found in the Requirements Analysis and Specification Document or from the one in the Design Document (or if it can be encountered in both); it also provides further detail on where in those lists is the original function to be found.

Function	Found in RASD?	Where in RASD?	Found in DD?	Where in DD?
[F1]	Yes	Guest/1	Yes	F02
[F2]	Yes	Guest/2	Yes	F03
[F3]	Yes	Customer/13 Employee/3 Administrator/4	Yes	F03
[F4]	Yes	Guest/3	No	--
[F5]	Yes	Guest/3	Yes	F11
[F6]	Yes	Customer/1	No	--
[F7]	Yes	Customer/2	Yes	F08
[F8]	Yes	Customer/3	Yes	F08
[F9]	Yes	Customer/3	Yes	F08
[F10]	Yes	Customer/3	Yes	F12
[F11]	Yes	Customer/3	Yes	F12
[F12]	Yes	Customer/3	Yes	F16
[F13]	Yes	Customer/3	Yes	F16
[F14]	Yes	Customer/6	Yes	F10
[F15]	Yes	Customer/7	Yes	F09
[F16]	Yes	Customer/8	Yes	F13
[F17]	Yes	Customer/11	Yes	F16
[F18]	Yes	Customer/11	No	--
[F19]	Yes	Customer/12	Yes	F15
[F20]	Yes	Employee/1	Yes	F05
[F21]	Yes	Employee/1	Yes	F05

Function	Found in RASD?	Where in RASD?	Found in DD?	Where in DD?
[F22]	Yes	Employee/1	Yes	F05
[F23]	Yes	Employee/2	Yes	F06
[F24]	No	--	Yes	F06
[F25]	No	--	Yes	F06
[F26]	Yes	Administrator/1	Yes	F01
[F27]	Yes	Administrator/1	Yes	F01
[F28]	Yes	Administrator/1	Yes	F01

TABLE 2: DEVELOPMENT TEAM'S DOCUMENTS SOURCES FOR THE RESTRUCTURED LIST OF FUNCTIONALITIES

2.2 Testing report

The test cases in the following pages constitute the core of the testing phase. Each one of them was laid out with the intention of verifying the correct execution of one function among those enucleated in the previous subsection.

The test cases have been developed keeping in mind that the object of this phase is the application's acceptance testing: the main question to answer is therefore "Does the system deliver the service it is supposed to, according to the developers' specifications?". This implies that the tests focused **on the outcome of an operation, given an input**, thus treating the system as a black-box; the contents of the database and the application's code were considered only in presence of bugs or unexpected behaviour, so as to give a better explanation of the reasons of such anomalies.

Given that premise, the structure of the test cases highlights the input/output focus of the experiments conducted on the target software. Here is a brief overview of the sections each test case is divided into:

- *Functionalities* lists all the functions that have been put to trial in the test case.
- *Environment* reports the specific web page(s) where the test case took place.
- *Preliminary notes* is an optional section reporting relevant annotations on what happened during the testing procedures.
- *Expected outcome* describes what the system is expected to do when the flow of events related to the test case has ended. Different outcomes can be specified in relation to different classes of inputs, i.e. the result of the operation when providing correct input might diverge from what happens when fallacious information is entered. This section draws from various elements found in the Requirements Analysis and Specification Document, i.e. mainly the use cases in Subsection 3.1.3.
- *Outcome with correct input* presents the output of the system in case consistent input was provided. Typically, the application should behave exactly as expected.
- *Outcome with anomalous input*, on the other hand, records the results of the test case when erroneous data were entered. The system should usually reject the user's request in this case.
- *Actual outcome* replaces the previous two sections in case the tested procedure does not require any input on the user's part.
- *Conclusions* summarizes the findings of the previous sections declaring the test passed (green background) or failed (red background). The verdict depends on whether the system actually delivers the functionalities referenced in the appropriate section.

[T1] Customer registration	
Functionalities	**[F1]** Register to the system as customers.
Environment	The application's customer registration page (`/Registration.jsp`).
Expected outcome	A new user account is created, allowing login to the system with the specified credentials. The operation is expected to fail if inconsistent input is provided to the system (e.g. empty fields, invalid e-mail address) or if the user is already registered, as is stated in the *Exceptions* section of the "Registration to the system" use case (RASD page 13).
Outcome with correct input	Providing correct and consistent data to the system results in the user being registered and gaining full access to the site as expected.
Outcome with anomalous input	*Empty fields.* Leaving one or more fields empty triggers an error message and causes the procedure not to terminate correctly, as expected. *Inconsistent values.* Entering invalid e-mail addresses such as "aaaa" and "/" still results in the procedure completing successfully and the account being saved to the database with no warnings. *Re-registration.* Any attempt to register a user with the same username and/or the same e-mail address as another account in the database causes an error message to be shown and the account not to be saved.
Conclusions	The registration process does what it's supposed to: guests can access the appropriate form and autonomously sign up, thus obtaining access credentials to *TravelDream*'s website. Though the system allows them to cheat on their e-mail address, the function is correctly implemented.

TEST CASE 1: USER REGISTRATION

[T2] Customer login	
Functionalities	**[F2]** Login to the system.
Environment	The application's customer login page (`/login.jsp`).
Expected outcome	The system should allow the user into the customer-reserved area of the site and present them with their homepage until a logout request is submitted. Providing a non-existent username, failing to type the correct password or entering staff credentials should bar the user from entering said area, instead.
Outcome with correct input	Filling the login form with a correct username-password couple (i.e. with credentials specified in a successful registration process) correctly results in the redirection to the customer's personal page.
Outcome with anomalous input	*Empty fields*. The system does not grant access to the reserved area. *Non-existent username or wrong password.* The system does not grant access to the reserved area. *Staff credentials.* Entering staff credentials does not allow access to the customer-reserved area.
Conclusions	The login process is correctly structured: permission to enter the customer's own area is granted only upon entering a valid username-password credentials couple. No bug whatsoever having been found, the function – for what concerns customers – is correctly implemented.

TEST CASE 2: CUSTOMER LOGIN

[T3] Staff login

Functionalities	**[F2]** Login to the system.
Environment	The application's staff login page (`/StaffLogin.jsp`).
Preliminary notes	It should be noted that nowhere in the accompanying guides does the development team state that staff members do not perform their login in the same page as customers. In fact, the only way to access any staff-reserved page is to manually type the desired URL – which had to be guessed looking at the `WebContent` folder in Eclipse – in the browser's address bar.
Expected outcome	The system should allow the user into the staff-reserved area of the site and present them with their homepage until a logout request is submitted. Providing a non-existent username, failing to type the correct password or entering customer credentials should bar the user from entering said area, instead.
Outcome with correct input	Filling the login form with a correct username-password couple (i.e. with credentials obtained by an administrator) correctly results in the redirection to the staff member's personal page.
Outcome with anomalous input	*Empty fields*. The system does not grant access to the reserved area, although no error message is shown on screen. *Non-existent username or wrong password.* The system does not grant access to the reserved area, although no error message is shown on screen. *Customer credentials.* Entering staff credentials does not allow access to the customer-reserved area, although no error message is shown on screen.
Conclusions	The login process is correctly structured: permission to enter the staff member's own area is granted only upon entering a valid username-password credentials couple. The lack of detail about the separate login pages for customers and staffers caused some trouble; nevertheless, the function is correctly implemented.

TEST CASE 3: STAFF LOGIN

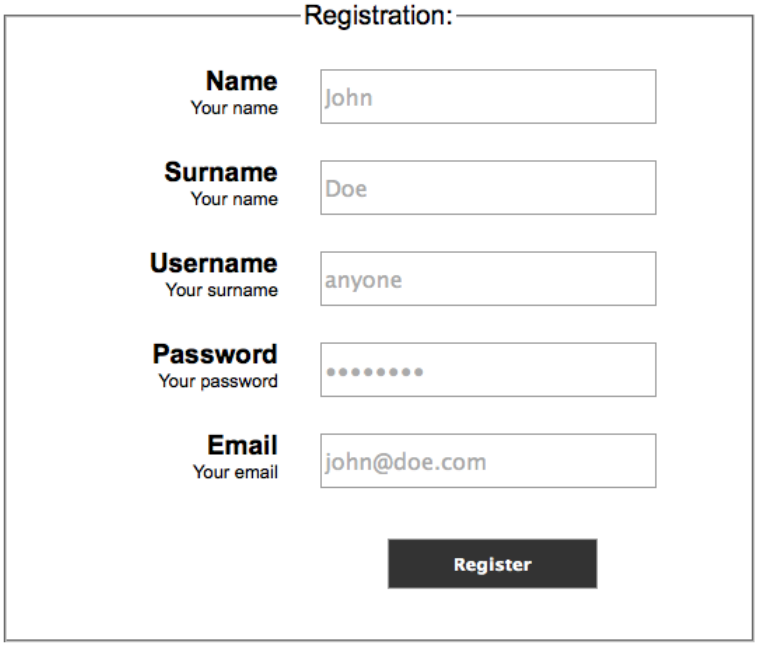
[T4] Logout from the system

Functionalities	**[F3]** Logout from the system.
Environment	Any page in the website requiring login.
Expected outcome	The system closes the user's session – whether it be a customer or a staff member – and requires them to undergo the login procedure again in order to enter their reserved area.
Actual outcome	The procedure does not expect any input. Clicking on the *Logout* button always results in the user's session being actually terminated.
Conclusions	The logout process is intuitive and seamless. Trying to manually re-enter the reserved pages after logging out triggered the request for a credentials pair again; the function is therefore correctly implemented.

TEST CASE 4: LOGOUT FROM THE SYSTEM

[T5] Creation of a new employee account	
Functionalities	**[F26]** Add employee accounts.
Environment	The application's staff registration page (`/StaffRegistration.jsp`).
Expected outcome	The system is expected to store a new account to the database, whose credentials are supposed to grant access to the staff-reserved area. The operation is expected to fail if inconsistent input is provided to the system (e.g. empty fields, invalid e-mail address) or if the employee is already registered, as is stated in the *Exceptions* section of the "Register a new employee" use case (RASD page 21).
Outcome with correct input	Filling in the form with correct and consistent information does **not** allow access to the staff area with the specified credentials. Further investigation as to why this happened revealed that the account had in fact been saved to the database; only, the fields were almost all scrambled up, as is testified by Picture 2 in the following page (which results in the tuple shown below to be inserted into the database). This, in turn, was caused by a simple, yet lethal, mistake in the `StaffRegistrationServlet#doPost` method, where arguments are passed to the `StaffMgrRemote#addEmployee` method in the wrong order. As a result, login is still possible, but the staffer would have to type their first name in the login form's password field – or the administrator should scramble the data they type in the registration form according to how they are saved.
Outcome with anomalous input	*Empty fields.* Leaving one or more fields empty triggers an error message and causes the procedure not to terminate correctly, as expected. *Inconsistent values.* Entering invalid e-mail addresses – i.e. surnames, given the scrambling schema described above - such as "aaaa" and "/" still results in the procedure completing successfully and the account being saved to the database with no warnings. *Re-registration.* Any attempt to register a user with the same username as another account in the database causes an error message to be shown and the account not to be saved. However, due to the aforementioned field scrambling, the other field checked for uniqueness is not the e-mail address (as it happened in the customer registration process), but what's written into it, i.e. the staffer's surname. This potentially bars the system administrator from adding any account for staff members having the same surname as another. As a side note, incurring in any of these behaviours silently redirects the administrator to the *Customer* registration form, instead of simply clearing the current one. This implies that filling in the fields appearing on screen after a mistake was done creates a customer account instead of a staff profile, unless the administrator clicks on the *back* button of their browser.
Conclusions	There actually *is* a way to get things working here, but it requires knowledge of the code (which a simple staffer in a real-world case should not have to learn about). As a matter of fact, trying to log into the application with the credentials provided during the registration phase does not grant access to the system. It is therefore safe to say that the function was not correctly implemented.

TEST CASE 5: CREATION OF A NEW EMPLOYEE ACCOUNT



PICTURE 2: STAFF REGISTRATION FORM AS IT WAS FILLED IN DURING THE TESTING PROCEDURE

```
mysql> select * from staff where username = 'anyone';
+---------+------+--------------+----------+------------+----------+----------+
| IDSTAFF | MAIL | NAME         | PASSWORD | PERMISSION | SURNAME  | USERNAME |
+---------+------+--------------+----------+------------+----------+----------+
|       6 | Doe  | john@doe.com | John     |          0 | password | anyone   |
+---------+------+--------------+----------+------------+----------+----------+
1 row in set (0,00 sec)
```

MYSQL SNIPPET 1: THE TUPLE CREATED BY SUBMITTING THE FORM IN PICTURE 2

[T6] Modification of an employee account

Functionalities	**[F27]** Edit employee accounts.
Environment	A page could not be found for this function.
Conclusions	Nowhere in the staff area is an account management section to be found. The account modification function has therefore not been implemented. It is obviously possible for the administrator to edit the account's details with an SQL statement, but direct operation on the database should not be considered as a valid solution when testing the *software*'s functionalities.

TEST CASE 6: MODIFICATION OF AN EMPLOYEE ACCOUNT

[T7] Deletion of employee accounts

Functionalities	**[F28]** Delete employee accounts.
Environment	A page could not be found for this function.
Conclusions	Nowhere in the staff area is an account management section to be found. The account modification function has therefore not been implemented. It is obviously possible for the administrator to delete the desired accounts with an SQL statement, but direct operation on the database should not be considered as a valid solution when testing the *software*'s functionalities.

TEST CASE 7: DELETION OF EMPLOYEE ACCOUNTS

[T8] Creation of basic products of all kinds

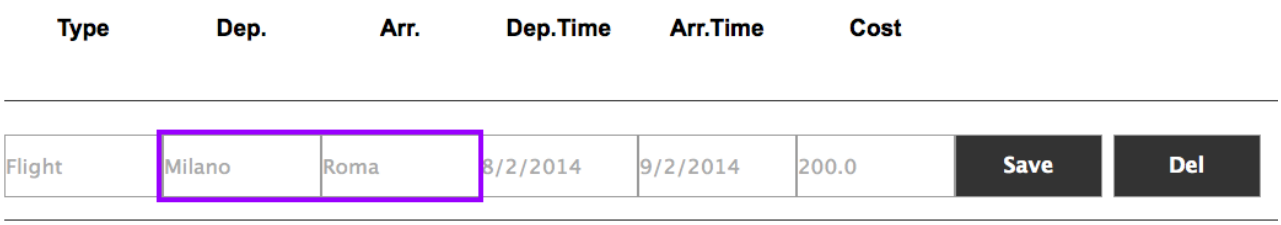
Functionalities	**[F20]** Create a basic product.
Environment	The application's flight creation page (`/BPAddTransportation.jsp`). The application's hotel creation page (`/BPAddHotel.jsp`). The application's excursion creation page (`/BPAddExcursion.jsp`).
Expected outcome	A new basic product is added to the application's database, whose details match those entered by the employee in the creation form. The operation is expected to fail if inconsistent information is provided (e.g. blank fields, invalid price…), even though the only reference use case (RASD page 20, which exemplifies the creation of an excursion but can be extended to all kinds of basic products) does not contemplate these possibilities in the *Exceptions* section.
Outcome with correct input	A flight with the typed features is correctly saved into the database and is available for inclusion in a travel package.
Outcome with anomalous input	*Empty fields.* Leaving one or more fields empty results in the form being cleared and an error message being shown (except for the excursion creation process, where no warning is issued to the employee). *Flight: arrival before departure.* If the entered arrival date precedes the departure date, the system correctly prevents the creation of the flight. *Past dates.* Providing past dates in any field requiring a date does not cause any exceptions to be raised. Though no requirement specified this as anomalous behaviour, it feels sensible to highlight it. *Decimal price.* The *Cost* field in any of the three forms only accepts integer values, in spite of the corresponding database column being declared as `float`, causing an ungraceful HTTP 500 error message to be shown on screen. The reason for this resides in the `AddTransportation#doPost` method, which performs validation of said field with the `Integer#parseInt` facility. Again, though no requirement declared this as anomalous behaviour, it is reasonable to assume that this constitutes a bug, since a price in euros might well contemplate decimal fractions. *Negative price.* Furthermore, negative values are allowed in the *Cost* field. Since it makes no sense to indicate a negative value for a price, this can be considered a bug even if it doesn't violate any specified constraint. *Excursion: non-digit characters or negative values in* Person *field.* Similarly, the *Person* field does not accept characters other than numbers, though it doesn't complain when a negative integer is provided, but the warning message it throws is an HTTP 500 error from Glassfish. *Excursion: non-digit characters in* Duration *field.* The system allows to save an excursion even though the *Duration* field contains a non-numeric string.
Conclusions	Entering correct and consistent data into the creation form actually performs the operation as expected. The numerous bugs that affect these procedures allow erroneous values to be saved to the database, thus making the forms "typo-unfriendly"; still, the test is passed because the function is actually performed by the application and correctly implemented.

TEST CASE 8: CREATION OF A FLIGHT

[T9] Modification of a flight

Functionalities	**[F21]** Edit a basic product.
Environment	The application's flight management page (`/BPEditTravel.jsp`).
Preliminary notes	The modification screen is affected by a string truncation defect, as shown in Picture 3 and MySQL snippet 2: every string is truncated at the first whitespace character, although the basic product's data show correctly by querying the database directly.
Expected outcome	The selected flight's details are changed according to what the employee typed in the modification form. The old version of the flight is not available in the system anymore.
Outcome with correct input	Leaving the price field intact results in an error message warning the employee about a `NumberFormatException`. This is due to the casting issues explained in test case [T8]: the cost is rendered as a float value, but not modifying it causes the `Integer#parseInt` method – called in `EditFlightServlet#doPost` – to raise the exception because of the decimal separator. Re-entering the field's value as an integer (e.g. from "200.0" to "200") manages to keep the exception at bay. Even in the absence of such an issue, however, entering correct data causes a blank screen to be shown upon clicking the *Save* button. Returning to the flight modification screen highlights that no change has been recorded on the database.
Outcome with anomalous input	*Empty fields.* Voiding one or more fields causes an error message to be shown. *Dates.* A warning message is issued if the date is badly formatted, but the blank screen is shown again if the arrival date precedes the departure date (which suggests this configuration would be accepted by the validation mechanisms).
Conclusions	Being it impossible to propagate the modifications to the database, the function – as far as flights are concerned – is to be considered badly implemented.

TEST CASE 9: MODIFICATION OF A FLIGHT



PICTURE 3: TRUNCATED STRINGS IN THE FLIGHT MODIFICATION SCREEN

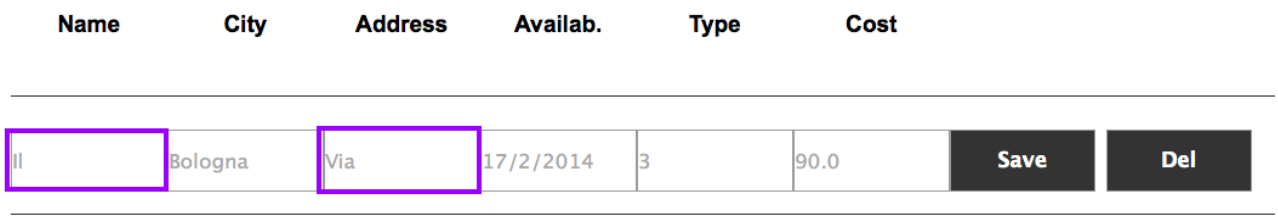
```
mysql> select DEPARTUREFROM, GOINGTO, INBOUND, OUTBOUND from transportation;
+-----------------+---------------+----------+----------+
| DEPARTUREFROM   | GOINGTO       | INBOUND  | OUTBOUND |
+-----------------+---------------+----------+----------+
| Milano Malpensa | Roma Ciampino | 9/2/2014 | 8/2/2014 |
+-----------------+---------------+----------+----------+
1 row in set (0,00 sec)
```

MYSQL SNIPPET 2: THE FLIGHT'S STRINGS ARE CORRECTLY SAVED IN THE DATABASE

[T10] Modification of a hotel

Functionalities	**[F21]** Edit a basic product.
Environment	The application's hotel management page (`/BPEditHotel.jsp`).
Preliminary notes	The modification screen is affected by a string truncation defect, as shown in Picture 4 and MySQL snippet 3: every string is truncated at the first whitespace character, although the basic product's data show correctly by querying the database directly.
Expected outcome	The selected hotel's details are changed according to what the employee typed in the modification form. The old version of the hotel is not available in the system anymore.
Outcome with correct input	The *Availability* field does not accept well-formed dates as inputs. This happens because of a coding mistake at line 57 in `EditHotelServler#doPost`, where validation of said field is erroneously delegated to the `Integer#parseInt` method (which was likely to be used on the *Price* value instead): checking a date with that method obviously results in a `NumberFormatException` to be thrown. This oversight makes it impossible to complete the operation.
Outcome with anomalous input	*Empty fields.* Voiding one or more fields causes an error message to be shown. Any other combination of valid and wrong inputs triggers the aforementioned `NumberFormatException`.
Conclusions	The procedure rejects well-formed values entered in the fields. As a result, there is no way whatsoever in which the hotel's details can be modified. The function – as far as hotels are concerned – is thus to be considered badly implemented.

TEST CASE 10: MODIFICATION OF A HOTEL



PICTURE 4: TRUNCATED STRINGS IN THE HOTEL MODIFICATION SCREEN

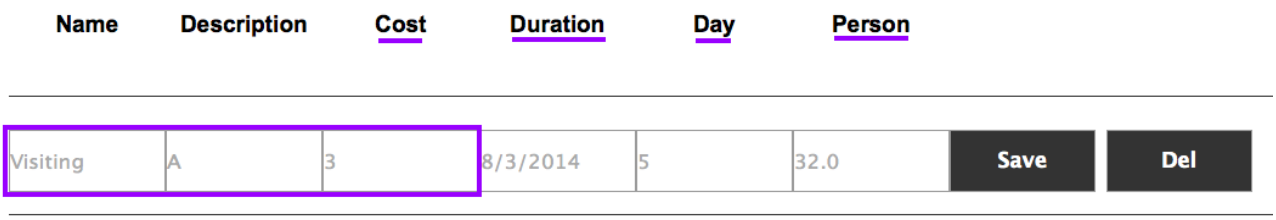
```
mysql> select NAME, CITY, ADDRESS from hotel;
+-------------+---------+--------------------+
| NAME        | CITY    | ADDRESS            |
+-------------+---------+--------------------+
| Il Guercino | Bologna | Via Luigi Serra, 7 |
+-------------+---------+--------------------+
1 row in set (0,00 sec)
```

MYSQL SNIPPET 3: THE HOTEL'S STRINGS ARE CORRECTLY SAVED IN THE DATABASE

[T11] Modification of an excursion

Functionalities	**[F21]** Edit a basic product.
Environment	The application's excursion management page (`/BPEditExcursion.jsp`).
Preliminary notes	The modification screen is affected by a string truncation defect, as shown in Picture 5 and MySQL snippet 4: every string is truncated at the first whitespace character, although the basic product's data show correctly by querying the database directly. Moreover, the columns are mislabelled (they do not match with the fields they contain).
Expected outcome	The selected excursion's details are changed according to what the employee typed in the modification form. The old version of the excursion is not available in the system anymore.
Outcome with correct input	Upon clicking the *Save* button, an HTTP 500 error message from Glassfish informs that a `NullPointerException` was thrown. Further investigation revealed that the exception is triggered at line 63 in `EditExcursionServlet#doPost`, but no clue was found as to why causes this. Reloading the excursion management reveals that no change is made to the values saved in the database.
Outcome with anomalous input	*Empty fields.* Voiding one or more fields causes an error message to be shown. Any other combination of valid and wrong inputs triggers the aforementioned `NullPointerException`.
Conclusions	The procedure throws an exception in nearly any case without ever getting to store the modified values into the database. As a consequence, the reference function is to be considered badly implemented for what concerns excursions.

TEST CASE 11: MODIFICATION OF AN EXCURSION



PICTURE 5: TRUNCATED STRINGS AND MISLABELLED COLUMNS IN THE EXCURSION MODIFICATION SCREEN

```
mysql> select NAME, DESCRIPTION, TOTALCOST, DURATION, AVAILABILITYDAY from
excursion where name = "Visiting London";
+-----------------+--------------+-----------+----------+-----------------+
| NAME            | DESCRIPTION  | TOTALCOST | DURATION | AVAILABILITYDAY |
+-----------------+--------------+-----------+----------+-----------------+
| Visiting London | A brief visit |        32 | 3 days   | 8/3/2014        |
+-----------------+--------------+-----------+----------+-----------------+
1 row in set (0,00 sec)
```

MYSQL SNIPPET 4: THE EXCURSION'S STRINGS ARE CORRECTLY SAVED IN THE DATABASE

[T12] Deletion of basic products of all kinds

Functionalities	**[F22]** Delete a basic product.
Environment	The application's flight management page (`/BPEditTravel.jsp`). The application's hotel management page (`/BPEditHotel.jsp`). The application's excursion management page (`/BPEditExcursion.jsp`).
Expected outcome	The basic product that's been selected for deletion disappears from the basic products list and cannot be included in travel packages anymore. The system's behaviour in case a basic product is part of one or more travel packages has not been specified in any of the accompanying documents.
Actual outcome	Clicking the *Del* button in one of the displayed rows correctly triggers the correspondent basic product's immediate deletion. A message informs the employee that the procedure terminated gracefully. The deletion of a basic product that is part of a package is denied by the system: an error message informs the employee that such an operation is forbidden.
Conclusions	An employee can certainly delete any one of the basic products in the application's database, provided that they are not part of a basic product. Considering that this limitation is in all likelihood a design constraint, the function can be declared correctly implemented.

TEST CASE 12: DELETION OF BASIC PRODUCTS OF ALL KINDS

[T13] Creation of travel packages

Functionalities	**[F23]** Compose basic products into a pre-defined travel package.
Environment	The application's package creation page (`/PCreatePackage1.jsp`).
Expected outcome	A new travel package is created in the application's database with the name and the components entered by the employee during its creation process. The package is available to customers for personalization, purchase and insertion in the travel list, in the gift list or in the cart.
Outcome with correct input	The travel package is correctly inserted into the database carrying all and only the components selected during the composition procedure; it is immediately available to any customer.
Outcome with anomalous input	*Empty name.* Leaving the *Name* field blank, thus creating a null-named package, is permitted by the system. *Package composition.* The system does not seem to perform any kind of checks on the number and the quality of the package's components, since even empty packages are seamlessly created. Though not conflicting with any written requirement, this allows the employee to create *de facto* useless packages, given that at least two flights are needed for a customer to begin the personalization. *Duplicate packages.* Packages with the same name can coexist.
Conclusions	The application allows the creation and the subsequent publication of travel packages. Though it is up to their creator to check for their correctness, the function is surely to be considered implemented.

TEST CASE 13: CREATION OF TRAVEL PACKAGES

[T14] Modification of travel packages

Functionalities	**[F24]** Edit a pre-defined travel package.
Environment	A page could not be found for this function.
Conclusions	The staff-reserved section does not host a page where to edit the contents of a travel package. It is therefore safe to say that this function was not implemented.

TEST CASE 14: MODIFICATION OF TRAVEL PACKAGES

[T15] Deletion of travel packages

Functionalities	**[F25]** Delete a pre-defined travel package.
Environment	The application's package deletion page (`/PManagePackage1.jsp`).
Preliminary notes	Packages saved with no name are listed with an odd "name=name" title. Furthermore, the string truncation issue encountered in test cases [T9], [T10] and [T11] affects the titles of the packages, too: any word beyond the first is not displayed.
Expected outcome	The package that's been selected for deletion disappears from the packages list and cannot be personalized by customers anymore. The system's behaviour in case the package is in one or more customers' travel list, gift list, or cart has not been specified in any of the accompanying documents.
Actual outcome	The package is correctly deleted and is not available to customers anymore, as expected. Trying to delete a package that's inserted in at least one customer's travel list, or gift list, or cart fires an ungraceful HTTP 500 error informing the employee of an `EJBException`. A quick analysis of the stack trace reveals that the exception is caused by a violation of a foreign key constraint while executing an SQL `DELETE` statement. The deletion is therefore forbidden by the DBMS's referential integrity checks.
Conclusions	The application allows the employees to correctly delete packages (even though the truncation of their name makes them hardly recognizable). Although the restriction on currently-listed packages was never mentioned, it can be regarded as a plausible limitation. The function is thus correctly implemented.

TEST CASE 15: DELETION OF TRAVEL PACKAGES

[T16] Browsing and searching through the company's offer

Functionalities	**[F6]** Browse and search through the packages offered by the company.
Environment	The application's package search page (`/USearch.jsp`).
Preliminary notes	Packages saved with no name are listed with an odd "name=name" title. Furthermore, the string truncation issue encountered in test cases [T9], [T10], [T11] and [T15] affects the titles of the packages, too: any word beyond the first is not displayed.
Expected outcome	The customer is able to see the company's entire travel package offer. No clue has been given, in the accompanying documents, as to what differences the action of *browsing* and that of *searching* (if any difference was ever intended).
Actual outcome	The customer is actually able to see a list of currently available packages, though the impossibility to read beyond the first word could cause some distinguishability issue. No filter has been implemented, so there is no way to actually search through the list.
Conclusions	In spite of the lack of a search field, and even considering the mutilated titles, a customer is actually able to view a list of packages offered by the system. The function can therefore be considered implemented.

TEST CASE 16: BROWSING AND SEARCHING THROUGH THE COMPANY'S OFFER

[T17] Personalization of a travel package

Functionalities	**[F7]** Personalize a package offered by the company in one's own travel list. **[F8]** Add a package to one's own travel list.
Environment	The set of pages concurring to the execution of this task (`/USearch.jsp`, `/UPackagePersonalize1.jsp` and following…).
Expected outcome	A new personalized package, corresponding to the choices made during the process, is added to the customer's travel list. It can then immediately be moved to the gift list or to the cart.
Actual outcome	The package is actually created and accessible in the travel list. The system imposes that the return flight take off after the arrival one lands (and there is no way to elude this constraint, since ineligible return flights are stripped off the list), while the hotel list only includes hotels available before the departure flight takes off. A similar schema is replicated for the (optional) excursions, where only those taking place between the arrival and departure date are shown. It should be noted that the various personalization pages all suffer from the string truncation issue described in several of the previous test cases, and that the excursion table's columns are again mislabelled (as well as the flight selection pages, where a *Type* column reports the flight's unique ID instead).
Conclusions	The process can be carried out by the customer and the system does what it should, but the flows of events found in the relative use cases (RASD pages 15 and 16) – which are the only official go-to references when it comes to assess whether a system meets the requirements or not – describe a completely different scenario: no date can be selected, no tourist guides can be chosen, the package's details cannot be viewed before actually personalizing it… Function [F7] is therefore not implemented the way it should have been.

TEST CASE 17: PERSONALIZATION OF A TRAVEL PACKAGE

[T18] Removing a package from one's own travel list

Functionalities	**[F9]** Remove a package from one's own travel list.
Environment	The application's travel list management page (`/UTravelList.jsp`).
Preliminary notes	This page suffers from the same string truncation issue that was found in many other test cases.
Expected outcome	The selected package is correctly deleted from the customer's travel list.
Actual outcome	The package disappears from the travel list after an acknowledgment message is shown on screen.
Conclusions	The function has been correctly implemented.

TEST CASE 18: REMOVING A PACKAGE FROM ONE'S OWN TRAVEL LIST

[T19] Adding a package to one's own gift list

Functionalities	**[F10]** Add a package to one's own gift list.
Environment	The application's travel list management page (`/UTravelList.jsp`).
Preliminary notes	This page suffers from the same string truncation issue that was found in many other test cases.
Expected outcome	The selected package is moved from the customer's travel list to their own gift list, and is thus accessible to everyone having a direct link to said gift list.
Actual outcome	Clicking on the *ToGiftList* button actually results in the package disappearing from the travel list and being inserted into the same customer's gift list (no acknowledgment message appears on screen).
Conclusions	The function has been correctly implemented.

TEST CASE 19: ADDING A PACKAGE TO ONE'S OWN GIFT LIST

[T20] Deletion of a package from one's own gift list

Functionalities	**[F11]** Remove a package from one's own gift list.
Environment	The application's gift list management page (`/UWishList.jsp`).
Preliminary notes	The travel packages listed in this page do not appear with their own name, and not even with its first word, but are all identified by an inexplicable "it.polimi.tr" prefix that clearly reminds of a Java fully qualified name (see Picture 6).
Expected outcome	The selected package is deleted from the customer's gift list and is not accessible to its owner nor to anyone having a direct link to said gift list anymore.
Actual outcome	The expectations are met, since the package is actually deleted from the gift list.
Conclusions	The function has been correctly implemented.

TEST CASE 20: DELETION OF A PACKAGE FROM ONE'S OWN GIFT LIST

PICTURE 6: WRONG PACKAGE NAME IN THE GIFT LIST AND IN THE CART

[T21] Adding a package to one's own cart

Functionalities	**[F12]** Add a package to one's own cart.
Environment	The application's travel list management page (`/UTravelList.jsp`).
Preliminary notes	This page suffers from the same string truncation issue that was found in many other test cases.
Expected outcome	The selected package is moved from the customer's travel list to their own cart, and is thus eligible for purchase.
Actual outcome	Clicking on the *ToCart* button actually results in the package disappearing from the travel list and being inserted into the same customer's cart (no acknowledgment message appears on screen).
Conclusions	The function has been correctly implemented.

TEST CASE 21: ADDING A PACKAGE TO ONE'S OWN CART

[T22] Deletion of a package from one's own cart

Functionalities	**[F13]** Remove a package from one's own cart.
Environment	The application's cart management page (`/UCart.jsp`).
Preliminary notes	The travel packages listed in this page do not appear with their own name, and not even with its first word, but are all identified by an inexplicable "it.polimi.tr" prefix that clearly reminds of a Java fully qualified name (see Picture 6).
Expected outcome	The selected package is deleted from the customer's cart and becomes inaccessible.
Actual outcome	The expectations are met, since the package is actually deleted from the cart.
Conclusions	The function has been correctly implemented.

TEST CASE 22: DELETION OF A PACKAGE FROM ONE'S OWN CART

[T23] Joining a friend's package

Functionalities	**[F14]** Join a package in a friend's travel list for which an invitation was received.
Environment	The application's travel list management page (`/UTravelList.jsp`), using the link found in the invitation e-mail.
Expected outcome	The selected package is successfully joined by the invitee.
Actual outcome	Reaching the travel list management page passing the inviter's user ID as a parameter correctly returns the list with the *Join* button next to the various packages' names. However, clicking on that button triggers an HTTP 404 error from the server ("The requested resource is not available"). Glancing at the code, it turns out that the function has actually not been implemented, since the button refers to a `JoinTravelList` servlet which can't be found in the web project.
Conclusions	No travel package being actually joinable, the function is not carried out by the system.

TEST CASE 23: JOINING A FRIEND'S PACKAGE

[T24] Sending an invitation to one's own travel list

Functionalities	**[F15]** Send an invitation to one's own travel list to a friend.
Environment	The application's travel list invite page (`/SendMail2.jsp`).
Expected outcome	An e-mail message containing a link to the sender's travel list is sent to the address entered in the form.
Actual outcome	An e-mail message from traveldreamsystem@gmail.com with the expected content is correctly and immediately delivered to the specified address.
Conclusions	The function has been correctly implemented.

TEST CASE 24: SENDING AN INVITATION TO ONE'S OWN TRAVEL LIST

[T25] Sending an invitation to one's own gift list

Functionalities	**[F16]** Send an invitation to one's own gift list to a friend.
Environment	The application's gift list invite page (`/SendMail.jsp`).
Expected outcome	An e-mail message containing a link to the sender's gift list is sent to the address entered in the form.
Actual outcome	An e-mail message from traveldreamsystem@gmail.com with the expected content is correctly and immediately delivered to the specified address.
Conclusions	The function has been correctly implemented.

TEST CASE 25: SENDING AN INVITATION TO ONE'S OWN GIFT LIST

[T26] Purchase of a package in one's own cart

Functionalities	**[F17]** Buy a package from one's own cart.
Environment	The application's cart management page (`/UCart.jsp`).
Expected outcome	No use case has been found regarding the purchase process, so no expectations can be laid out.
Actual outcome	Clicking on the *Buy* button triggers an HTTP 404 error from the server ("The requested resource is not available"). Glancing at the code, it turns out that the function has actually not been implemented, since the button does not even specify a servlet in the `action` attribute.
Conclusions	Although no expected outcome was specified, an HTTP 404 error message surely cannot be considered as a graceful termination for the purchase process, which doesn't even start. It is therefore safe to declare the function not implemented.

TEST CASE 26: PURCHASE OF A PACKAGE IN ONE'S OWN CART

[T27] Purchase of a package in one's own cart

Functionalities	**[F18]** Buy a package from another customer's gift list for which an invitation was received.
Environment	The application's gift list management page (`/UWishList.jsp`), using the link found in the invitation e-mail.
Expected outcome	The selected package is inserted in the buyer's cart, as stated in the reference use case (RASD page 19).
Actual outcome	Clicking on the *Buy* button triggers an HTTP 404 error from the server ("The requested resource is not available"). Glancing at the code, it turns out that the function has actually not been implemented, since the button does not even specify a servlet in the `action` attribute. The package cannot be found in the buyer's cart.
Conclusions	Since the actual outcome differs from the expected one, the function has not been implemented correctly.

TEST CASE 27: PURCHASE OF A PACKAGE IN ONE'S OWN CART

[T28] Purchase of a previously-joined package

Functionalities	**[F19]** Buy a package in another customer's travel list that was previously joined.
Environment	According to Assumption 5 in Section 2.4 of the Requirements Analysis and Specification Document, a package should appear in the travel lists of all customers who joined it; it can then be moved to the cart according to Assumption 7. The environment should therefore be the buyer's cart (`/UCart.jsp`).
Expected outcome	No use case has been found regarding the purchase of a joined package, so no expectations can be laid out.
Actual outcome	Test case [T23] highlighted the impossibility to join a package. It is therefore impossible to judge on the feasibility of a joined package's purchase.
Conclusions	The function cannot be tested since the entry condition for this procedure cannot be fulfilled.

TEST CASE 28: PURCHASE OF A PREVIOUSLY-JOINED PACKAGE

[T29] Receiving an invitation to a friend's travel list

Functionalities	**[F4]** Receive invitations to other people's travel packages.
Environment	The receiver's e-mail client.
Expected outcome	An e-mail carrying a valid link to the sender's travel list is received, independently of the recipient being registered to the system or not.
Actual outcome	An e-mail is received both in case the recipient is registered and in case they are not.
Conclusions	Anyone can receive invitations to a travel list.

TEST CASE 29: RECEIVING AN INVITATION TO A FRIEND'S TRAVEL LIST

[T30] Receiving an invitation to a friend's gift list

Functionalities	**[F4]** Receive invitations to other people's travel packages.
Environment	The receiver's e-mail client.
Expected outcome	An e-mail carrying a valid link to the sender's gift list is received, independently of the recipient being registered to the system or not.
Actual outcome	An e-mail is received both in case the recipient is registered and in case they are not.
Conclusions	Anyone can receive invitations to a gift list.

TEST CASE 30: RECEIVING AN INVITATION TO A FRIEND'S GIFT LIST

[T31] Viewing the object of an invitation

Functionalities	**[F5]** View a travel package for which an invitation was received.
Environment	The friend's travel list (`/UTravelList.jsp`) or gift list (`/UWishList.jsp`).
Expected outcome	The recipient can view the sender's travel list or gift list, depending on what the invitation was for. This must happen independently of the recipient being logged to the system or not.
Actual outcome	It is actually possible to visit the friend's travel list or gift list. Trying to access other screens results in the login page being shown.
Conclusions	Even though it is not possible to be actually invited to a single package, but rather to an entire travel list or gift list, the invitation mechanisms works in its entirety. This function has therefore been correctly implemented.

TEST CASE 31: VIEWING THE OBJECT OF AN INVITATION

2.3 Conclusions

Each one of the functions outlined in Paragraph 2.1.3 has been thoroughly tested, trying every possible combination of inputs and recording the relative output. Table 3 contains a summary of the 31 test cases conducted on the system.

Function	Function description	Test cases	Passed?
[F1]	Register to the system as customers.	[T1]	✓ Yes
[F2]	Login to the system.	[T2] [T3]	✓ Yes
[F3]	Logout from the system.	[T4]	✓ Yes
[F4]	Receive invitations to other people's travel packages.	[T29] [T30]	✓ Yes
[F5]	View a travel package for which an invitation was received.	[T31]	✓ Yes
[F6]	Browse and search through the packages offered by the company.	[T16]	✓ Yes
[F7]	Personalize a package offered by the company in one's own travel list.	[T17]	× No
[F8]	Add a package to one's own travel list.	[T17]	✓ Yes
[F9]	Remove a package from one's own travel list.	[T18]	✓ Yes
[F10]	Add a package to one's own gift list.	[T19]	✓ Yes
[F11]	Remove a package from one's own gift list.	[T20]	✓ Yes

Function	Function description	Test cases	Passed?
[F12]	Add a package to one's own cart.	[T21]	✓ Yes
[F13]	Remove a package from one's own cart.	[T22]	✓ Yes
[F14]	Join a package in a friend's travel list for which an invitation was received.	[T23]	× No
[F15]	Send an invitation to one's own travel list to a friend.	[T24]	✓ Yes
[F16]	Send an invitation to one's own gift list to a friend.	[T25]	✓ Yes
[F17]	Buy a package from one's own cart.	[T26]	× No
[F18]	Buy a package from another customer's gift list for which an invitation was received.	[T27]	× No
[F19]	Buy a package in another customer's travel list that was previously joined.	[T28]	× No
[F20]	Create a basic product.	[T8]	✓ Yes
[F21]	Edit a basic product.	[T9] [T10] [T11]	× No
[F22]	Delete a basic product.	[T12]	✓ Yes
[F23]	Compose basic products into a pre-defined travel package.	[T13]	✓ Yes
[F24]	Edit a pre-defined travel package.	[T14]	× No
[F25]	Delete a pre-defined travel package.	[T15]	✓ Yes
[F26]	Add employee accounts.	[T5]	× No
[F27]	Edit employee accounts.	[T6]	× No
[F28]	Remove employee accounts.	[T7]	× No

TABLE 3: SUMMARY OF THE TEST CASES

As a consequence, the recap in Table 4 can be outlined for what concerns the goals laid out by the development team:

Goal	Goal description	Achieved?
[G1]	Registered users can buy a package at any time.	× **No.** Purchase-related functions were not implemented.
[G2]	Registered users can join another user's list at any time.	× **No.** Joining a package (not a list, anyway) results in an error message.
[G3]	Registered users can compose a list or a gift list at any time.	✓ **Yes.**
[G4]	Registered users can invite other users to see a package.	✓ **Yes.**
[G5]	Registered users can invite other users to a gift list at any time.	✓ **Yes.**
[G6]	Guests can view their invitations at any time.	✓ **Yes.**
[G7]	Employees can add hotels, flights and excursions at any time.	✓ **Yes.**
[G8]	Employees can compose packages at any time.	✓ **Yes.**
[G9]	System administrators can add employee accounts at any time.	× **No.** They actually can, but the account's details don't match what the administrator types in the form.

TABLE 4: GOAL ACHIEVEMENT

2.4 Judgement criteria

Some of the test cases – in spite of their functionalities actually being implemented – have been judged failed because they did not conform to their relative use case prospect or because they violated one or more of the assumptions made by the development team. This is what happened for test case [T17], for instance: a personalization procedure is actually offered by the system, but its differences with how it was designed cannot be ignored.

In other cases, where the requirement or design violations were minor, such differences were winked at. Think, for instance, of test case [T31] and, more generally, of all invitation-related functionalities: multiple statements were made about invitations regarding a single package, but the system only allows a customer to invite a friend to their entire travel list or gift list. Because the essence of the functionality was maintained, though, the functionalities were held to be correctly implemented.

When no reference could have been found in the accompanying documents, common sense (e.g. test cases [T12] and [T15]) or adaptation of similar use cases (e.g. test case [T8], where only a use case for the creation of an excursion was available) have been chosen as judgemental criteria.

Miscellanea

Changelog

Version 1 (February 10th, 2014)

First release of the Acceptance Testing Document.

Effort

A total of *27 hours* were spent on the production of this document. In particular:

- *3 hours* were dedicated to a preliminary reading of the Requirements Analysis and Specification Document and of the Design Document;
- *2 hours* were spent on the software's installation (including reading the Installation Guide);
- *6 hours* were reserved to the rationalization of the list of functionalities and to other preparatory works (i.e. the contents of Subsection 2.1);
- *16 hours* was what it took to conduct the actual testing on the system, including recording its results in this document and writing the conclusions.