Politecnico di Milano
M. Sc. Course in IT Engineering

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

Design Document

VERSION 2

by
Edoardo Mondoni (816283)

Table of contents

Index of pictures

Preamble

This is the Design Document for the Software Engineering 2 2013 internal project, named *TravelDream*. It is the result of the software design phase, which immediately followed the requirements analysis and paves the way for the implementation of the application. IEEE Recommended Practice 1016-1998, which is the design phase correspondent of what IEEE 803-1998 was for the requirements stage, suggests a way too complicated structure, considering the academic nature of this project. For this reason, this deliverable has been built on the basis of an *ad hoc* architecture. The reference version for the Requirements Analysis and Specification Document, wherever it is cited, is version 4.

The design process, which has been conducted on the basis of a bottom-up approach, consisted of two different and mutually independent phases: the first one focused on the design of the system's database, while the second one dealt with the application's under-the-hood architecture at different levels of detail.

The rest of the document is organized as follows:

- **Section 1** illustrates the stages into which the persistent data design process has articulated, from the informal summary of relevant information from the Requirements Analysis and Specification Document to the elaboration of the final relational schema;
- **Section 2** contains a thorough exposition of the concepts and the reasons underlying the design decisions concerning the application's software architecture, user interface and internal component structure.

1 Persistent data

This section illustrates the results of the database definition process:

- **Subsection 1.1** collects the relevant information from various parts of the Requirements Analysis and Specification Document to build a textual description of the database;
- **Subsection 1.2** focuses on the construction of a suitable entity-relationship model of the database, drawing from said description and refining it on a common-sense basis;
- **Subsection 1.3** finally transposes the definitive E-R diagram into a relational model, which also undergoes a perfecting stage to polish the remaining inaccuracies.
- **Subsection 1.4** illustrates the changes made to the database schema during the implementation.

1.1 Natural language description

In order to proceed with the drafting of the entity-relationship conceptual diagram for *TravelDream Agent*'s database, a textual description of the system has been redacted. This step, though apparently redundant, aims at gathering all the details relevant to the modelling of the persistent data into a single piece of text, drawing from the provisions contained in the Requirements Analysis and Specification Document.

> TravelDream Agent *is a software supporting the sales process for a company named* TravelDream. *Its main purpose is to enable the company to sell its products – in the form of travel packages – to the customers subscribing to their website. Said subscription process is free of charge and only requires the potential customer to enter their name, surname, date of birth, home address and e-mail address other than choosing an original username and a password.*
>
> TravelDream*'s travel solutions – which are identified by a unique code and have a name and a description – consist of atomic components called basic products: at the moment, only three types of basic products are available, namely flights, hotels and excursions, but* TravelDream *plans to add more transportation means in the foreseeable future. Given their different natures, these basic products feature very distinguished set of attributes (besides being all assigned a unique ID and a facultative description): flights, in fact, carry an indication of the departure and arrival times and places, of their frequency (in terms of weekdays) and of their call sign (the one starting with the airline company's IATA code); hotels are instead accompanied by their name, their address, information regarding check-in and check-out times and weekdays and the price for the nightly stay, along with optional photos; finally, excursions must specify their name, departure and arrival times and places, the list of visits they consist of, the weekdays on which they take place and, again, an optional set of photos.*
>
> *Basic products cannot be, however, directly bought by the customers: they first have to be associated into packages by the company's employees, who are also entitled to modifying and*

removing them from the system; they are also responsible for the population and maintenance of the system's basic product database. Packages can be customized, though: any customer can edit the list of basic products a travel bundle is composed of, in order to buy a product perfectly complying with their desires; pre-defined bundles are obviously also available for purchase on a customer's part. Customized packages are not accessible to customers other than the one who created them, and can subsequently be modified or removed.

It is clear by now that TravelDream Agent *interfaces with several categories of users, each of which is assigned a different set of abilities. Employees and customers can perform the aforementioned actions. Administrators do not perform business-related tasks, instead, but are in charge with the management of the user pool: they can edit any user's related data – except for their username – as well as deleting their account; they are also in charge of the creation of new administrator and employee accounts – whereas the creation of customer accounts is only allowed through the registration process.*

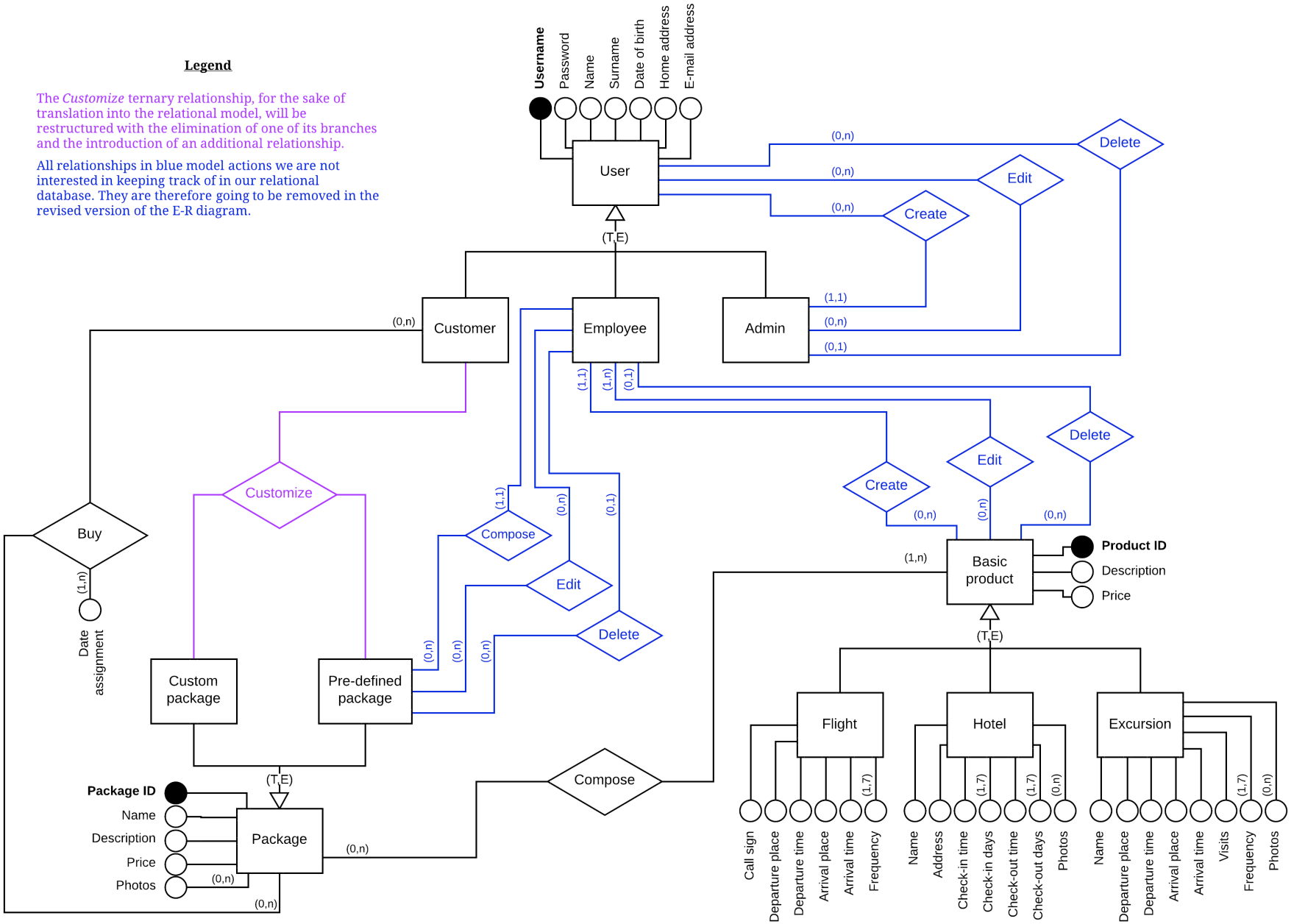
1.2 Entity-relationship modelling

The informal description in the previous subsection has been translated into the preliminary entity-relationship diagram on page 6. As is almost always the case with the conceptual design of databases, such diagrams need to be refined in order to respect formal constraints guaranteeing the model's portability to the relational logic (absence of illicit relationship loops, simplification of generalizations, transformation of multiple attributes…); the restructured diagram is shown on page 8.

1.2.1 Preliminary diagram

The composition of the following schema, strikingly similar to the Class Diagram contained in the Requirements Analysis and Specification Document, has been carried out in strict compliance with the contents of the description in Subsection 1.1. This has led to the inclusion of redundant and *n*-ary relationships, multiple attributes and entity generalizations, which cannot persist in the final version of the entity-relationship model. A brief explanation for the major design choices leading to this preliminary diagram follows.

- **Entities** have been derived from the aforementioned text, and their **attributes** reflect what has been therein specified.
- A pervasive use of **generalization constructs** has been opted for, in order to factor the attributes and the relationships common to many different entities. The (T,E) label indicates the *totality* and *exclusiveness* of the generalization relations, thus resulting in the children constituting a partition of the parent set of entities.
- The **blue relationships** perfectly correspond to the actions that employees and administrators can perform on the entities they're responsible for. In fact, there is no need for the database to store this kind of data, e.g. who created a basic product, who deleted a user or who composed a package: it is the business logic's duty to prevent users from performing unauthorized actions. Such relationships are therefore right in that they correctly model the system-to-be's behaviour, but are not to stay in the diagram because the information they carry is of no use to the application's functional purpose.
- The **purple *n*-ary relationship** indicates that the customization involves a customer, the pre-defined package it is based upon and the final custom package. For translation purposes, this relationship needs to be made binary; its expressive value will, however, be maintained through the introduction of other items (entities, binary relationships or both).



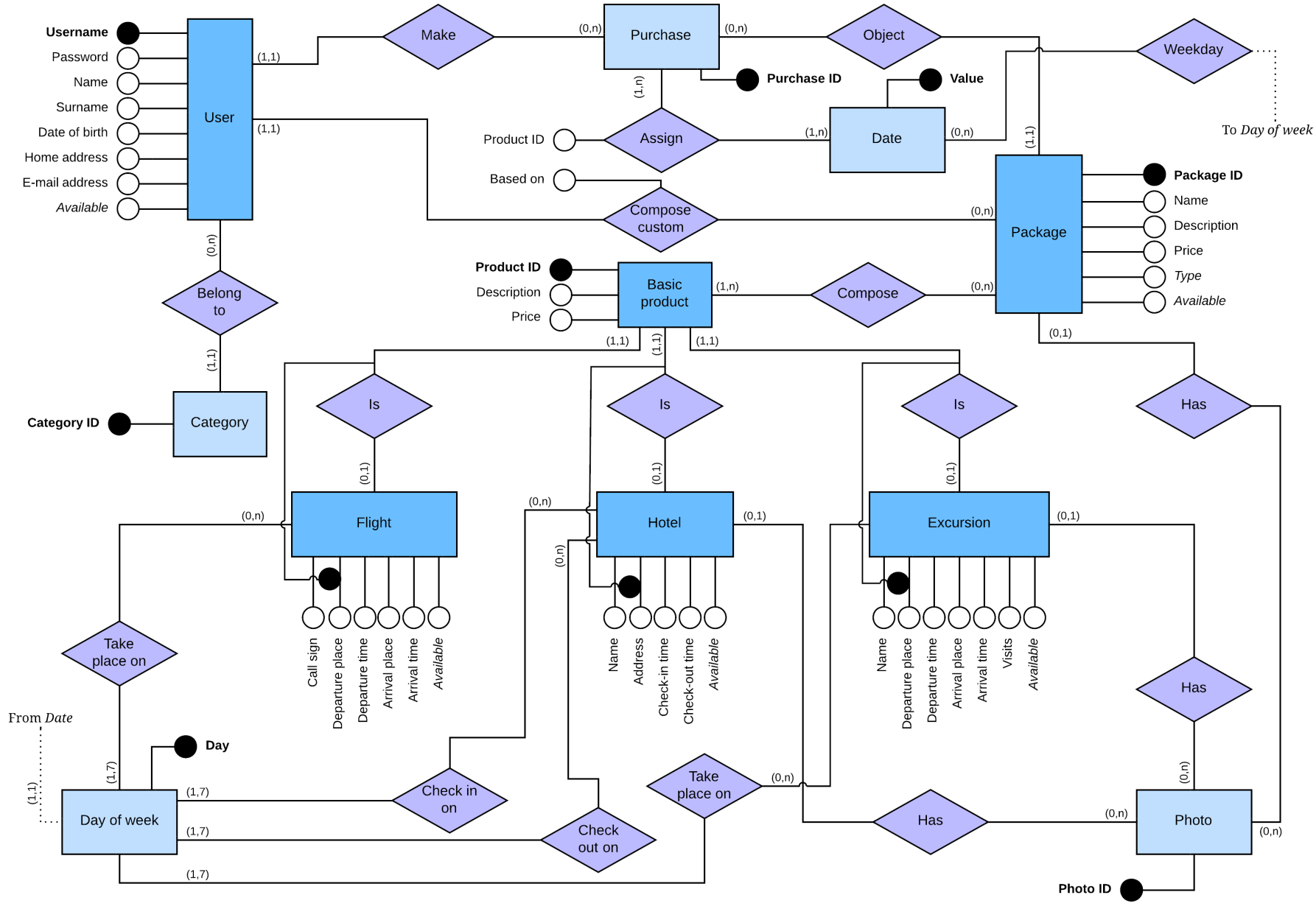
PICTURE 1: PRELIMINARY ENTITY-RELATIONSHIP DIAGRAM

1.2.2 Restructuration of the diagram

As anticipated in the opening of this subsection, the preliminary diagram needs to be reworked in order to be ready for the transposition into the relational model. What follows is a detailed account of what has been changed and why.

- ***Available attribute.*** An *Available* attribute has been added to all entities representing an object that can be removed from the system (i.e. a user, a basic product and a package). As a matter of fact, nothing is ever actually deleted from the system's database, in order for external keys not to be nulled or – even worse – for deletions to cascade to all tuples referencing the one that's being removed. What the user perceives as a deletion operation simply consists, instead, in setting said *Available* attribute to `false` so that queries will expunge it from the result set.
- ***Multiple and optional attributes.*** Attributes carrying multiplicity indicators on them have been transformed, as usual, in separate entities. The *Frequency* attribute in the *Flight* and *Excursion* entities, together with the *Check-in days* and *Check-out days* attributes in *Hotel*, has thus become a *Day of week* entity linked to the three basic product entities through various relationships. In the same way, the *Photos* attribute on *Package*, *Hotel* and *Excursion* has been externalized to an entity of the same name.
- ***Generalizations.*** The *User*, *Basic product* and *Package* generalizations had to be dealt with, and have received different treatments based on the distribution of the relationships and of the attributes they were linked to:
  - Reduction to the superclass has been opted for the *User* and *Package* entities, since their children had no specific attributes and were not intensely involved in relationships with other entities. Taking the other path, i.e. reduction to the subclass, would have resulted in a plethora of repeated attributes on the children entities, not to mention the replication of relationships.
  - Conversion to weak entities was the most viable option for the *Basic product* entity, instead, because of the variety of different attributes characterizing its children. Reduction to the superclass would in fact have induced the inclusion of all attributes into the *Basic product* table, thus causing null values to proliferate, while reducing to subclass would have led to a situation where basic products were split into three different tables, thus losing every reference to their common nature.

  The introduction of logical incongruences during this process was inevitable: for example, it looks like it is any user's prerogative to purchase and customize bundles, whereas only customers are granted such ability. These flaws, however, are concerned with the dynamic behaviour of the system, rather than with the way its data are structured: it is, in fact, the business logic's duty to ensure that no non-customer perform purchases. It can therefore be concluded that no constraint or requirement is being violated because of this imprecision.
- ***Blue relationships.*** Many of the relationships in the preliminary diagram, namely those in blue, have been eliminated because the behaviours they modelled are not required to be permanently stored in the database, as explained on page 5.
- ***Glassfish authentication compliance.*** Glassfish version 4.0 – which is the elected Application Server for the project as stated in Paragraph 2.1.2 of the Requirements Analysis and Specification Document – offers automatic handling of authentication procedures and takes care of the permissions to access data and methods according to the credentials the user logs in with. The *Category* entity was therefore introduced in order to comply with the database schema Glassfish requires for this automation to work, with the intent of making it a separate database table instead of declaring it as a simple attribute to *User*.



PICTURE 2: REVISED ENTITY-RELATIONSHIP DIAGRAM

(legend follows on page 8)

LEGEND FOR THE REVISED E-R DIAGRAM

Blue entities are the ones that resisted through the restructuration of the diagram.
Light blue entities have been introduced during the restructuration.
Associations are in purple for readability purposes.
Italic attributes attached to the inherited entities have been added during the restructuration.

1.3 Relational schema

The translation of the conceptual entity-relationship diagram into the relational model was carried out on the basis of the common general rules. This process produced a raw output, shown in the table below, which has subsequently been the object of a refinement process presented in the following pages.

Relational model – raw version	
USER	(USERNAME, PASSWORD, NAME, SURNAME, DATE OF BIRTH, HOME ADDRESS, E-MAIL, AVAILABLE)
CATEGORY	(CATEGORY ID)
USER CATEGORY	(USERNAME, CATEGORY)
BASIC PRODUCT	(PRODUCT ID, DESCRIPTION, PRICE)
FLIGHT	(PRODUCT ID, CALL SIGN, DEPARTURE PLACE, DEPARTURE TIME, ARRIVAL PLACE, ARRIVAL TIME, AVAILABLE)
HOTEL	(PRODUCT ID, NAME, ADDRESS, CHECK-IN TIME, CHECK-OUT TIME, AVAILABLE)
EXCURSION	(PRODUCT ID, NAME, DEPARTURE PLACE, DEPARTURE TIME, ARRIVAL PLACE, ARRIVAL TIME, VISITS, AVAILABLE)
PACKAGE	(PACKAGE ID, NAME, DESCRIPTION, PRICE, TYPE, AVAILABLE, BASED ON, AUTHOR)
COMPONENT	(PACKAGE ID, PRODUCT ID)
PURCHASE	(PURCHASE ID, BUYER, PACKAGE ID)
DATE ASSIGNMENT	(PURCHASE ID, PRODUCT ID, DATE)
PHOTO	(PHOTO ID)
DATE	(VALUE, WEEKDAY)
DAY OF WEEK	(WEEKDAY)
FLIGHT FREQUENCY	(PRODUCT ID, CALL SIGN, WEEKDAY)
EXCURSION FREQUENCY	(PRODUCT ID, EXCURSION NAME, WEEKDAY)
HOTEL CHECK-IN	(PRODUCT ID, HOTEL NAME, WEEKDAY)
HOTEL CHECK-OUT	(PRODUCT ID, HOTEL NAME, WEEKDAY)
EXCURSION PHOTOS	(PRODUCT ID, EXCURSION NAME, PHOTO ID)
HOTEL PHOTOS	(PRODUCT ID, HOTEL NAME, PHOTO ID)
PACKAGE PHOTOS	(PACKAGE ID, PHOTO ID)
LEGEND	
UNDERLINED columns are part of the primary key; BLUE columns are part of a foreign key.	

This preliminary schema, however, suffers from some inaccuracies:

- ***Glassfish authentication compliance.*** Glassfish Application Server requires the CATEGORY column in the USER CATEGORY table to be part of the primary key as well, since the same user can belong to different groups. Because this behaviour is not in line with *TravelDream Agent*'s requirements, the CATEGORY column will be made part of the primary key, but software constraints will be enforced that prevent the same user from being part of more than one group.
- ***Nullability of the AUTHOR and BASED ON columns.*** For every tuple representing a pre-defined package in the PACKAGE table, the AUTHOR and BASED ON columns (derived from the *Compose custom* relationship) would be null. This unwanted behaviour, stemming from the reduction to superclass operated on the *User* generalization, will be remedied by moving this information to a dedicated table, named CUSTOM PACKAGE. Ensuring that no pre-defined package appears in that table, and that all custom ones do, will then be the software's duty (possibly with the help of SQL assertions and triggers).

Relational model – final version	
USER	(USERNAME, PASSWORD, NAME, SURNAME, DATE OF BIRTH, HOME ADDRESS, E-MAIL, AVAILABLE)
CATEGORY	(CATEGORY ID)
USER CATEGORY	(USERNAME, CATEGORY)
BASIC PRODUCT	(PRODUCT ID, DESCRIPTION, PRICE)
FLIGHT	(PRODUCT ID, CALL SIGN, DEPARTURE PLACE, DEPARTURE TIME, ARRIVAL PLACE, ARRIVAL TIME, AVAILABLE)
HOTEL	(PRODUCT ID, NAME, ADDRESS, CHECK-IN TIME, CHECK-OUT TIME, AVAILABLE)
EXCURSION	(PRODUCT ID, NAME, DEPARTURE PLACE, DEPARTURE TIME, ARRIVAL PLACE, ARRIVAL TIME, VISITS, AVAILABLE)
PACKAGE	(PACKAGE ID, NAME, DESCRIPTION, PRICE, TYPE, AVAILABLE)
COMPONENT	(PACKAGE ID, PRODUCT ID)
CUSTOM PACKAGE	(PACKAGE ID, BASED ON, AUTHOR)
PURCHASE	(PURCHASE ID, BUYER, PACKAGE ID)
DATE ASSIGNMENT	(PURCHASE ID, PRODUCT ID, DATE)
PHOTO	(PHOTO ID)
DATE	(VALUE, WEEKDAY)
DAY OF WEEK	(WEEKDAY)
FLIGHT FREQUENCY	(PRODUCT ID, CALL SIGN, WEEKDAY)
EXCURSION FREQUENCY	(PRODUCT ID, EXCURSION NAME, WEEKDAY)
HOTEL CHECK-IN	(PRODUCT ID, HOTEL NAME, WEEKDAY)
HOTEL CHECK-OUT	(PRODUCT ID, HOTEL NAME, WEEKDAY)
EXCURSION PHOTOS	(PRODUCT ID, EXCURSION NAME, PHOTO ID)
HOTEL PHOTOS	(PRODUCT ID, HOTEL NAME, PHOTO ID)
PACKAGE PHOTOS	(PACKAGE ID, PHOTO ID)
LEGEND	
UNDERLINED columns are part of the primary key; BLUE columns are part of a foreign key.	

1.4 Post-implementation notes

During the implementation phase, some changes have been made to the relational model as exposed in the previous page:

- *AVAILABLE attribute on the USER table.* Due to Glassfish's built-in authentication mechanism, it was not possible to adopt the deletion policy that has been maintained on every other kind of entity in the system. Setting the available attribute to false would indeed exclude the user from the administrator's user table, but its credentials would still be accepted to log into the system. No other choice – besides modifying the `j_security_check` function – could therefore be made but reverting to a traditional row deletion for users.
- *CATEGORY table*. Contrarily to what was stated in the previous subsection, Glassfish doesn't need a separate table listing the various user groups: the CATEGORY table was consequently removed.
- *DATE and DAY OF WEEK tables.* The day-of-week/date check has been moved to the business logic and the Java Persistence APIs autonomously manage enum types, so the utility of these tables has proven null.
- *Date assignments*. As a consequence, three separate tables have been laid out for the date assignments. It was not possible to merge all date assignments in one table because hotels need two dates (one of the two columns would have been null for all excursions and hotels).
- *AVAILABLE attribute on basic products.* It had inexplicably been placed on the EXCURSION, FLIGHT, and HOTEL tables, now it has been coherently moved up to the parent BASIC PRODUCT table.
- *Basic product primary keys.* There was no need for the composite keys on the EXCURSION, FLIGHT, and HOTEL tables, which have been modified to feature a simple primary key.

Relational model – as it results from the implementation phase	
USER	(USERNAME, PASSWORD, NAME, SURNAME, DATE OF BIRTH, HOME ADDRESS, E-MAIL)
USER CATEGORY	(USERNAME, CATEGORY)
BASIC PRODUCT	(PRODUCT ID, DESCRIPTION, PRICE, AVAILABLE)
FLIGHT	(PRODUCT ID, CALL SIGN, DEPARTURE PLACE, DEPARTURE TIME, ARRIVAL PLACE, ARRIVAL TIME)
HOTEL	(PRODUCT ID, NAME, ADDRESS, CHECK-IN TIME, CHECK-OUT TIME)
EXCURSION	(PRODUCT ID, NAME, DEPARTURE PLACE, DEPARTURE TIME, ARRIVAL PLACE, ARRIVAL TIME, VISITS)
PACKAGE	(PACKAGE ID, NAME, DESCRIPTION, PRICE, TYPE, AVAILABLE)
PACKAGE COMPONENTS	(PACKAGE ID, PRODUCT ID)
CUSTOM PACKAGE	(PACKAGE ID, BASED ON, AUTHOR)
PURCHASE	(PURCHASE ID, BUYER, PACKAGE ID)
EXCURSION DATES	(PURCHASE ID, PRODUCT ID, DATE)
FLIGHT DATES	(PURCHASE ID, PRODUCT ID, DATE)
HOTEL DATES	(PURCHASE ID, PRODUCT ID, CHECK-IN DATE, CHECK-OUT DATE)
PHOTO	(PHOTO ID, FILENAME)
BASIC PRODUCT PHOTO	(PRODUCT ID, PHOTO ID)
PACKAGE PHOTO	(PACKAGE ID, PHOTO ID)
FLIGHT FREQUENCY	(PRODUCT ID, WEEKDAY)

EXCURSION FREQUENCY	(PRODUCT ID, WEEKDAY)
HOTEL CHECK-IN	(PRODUCT ID, WEEKDAY)
HOTEL CHECK-OUT	(PRODUCT ID, WEEKDAY)

LEGEND

UNDERLINED columns are part of the primary key; BLUE columns are part of a foreign key.

2 Application design

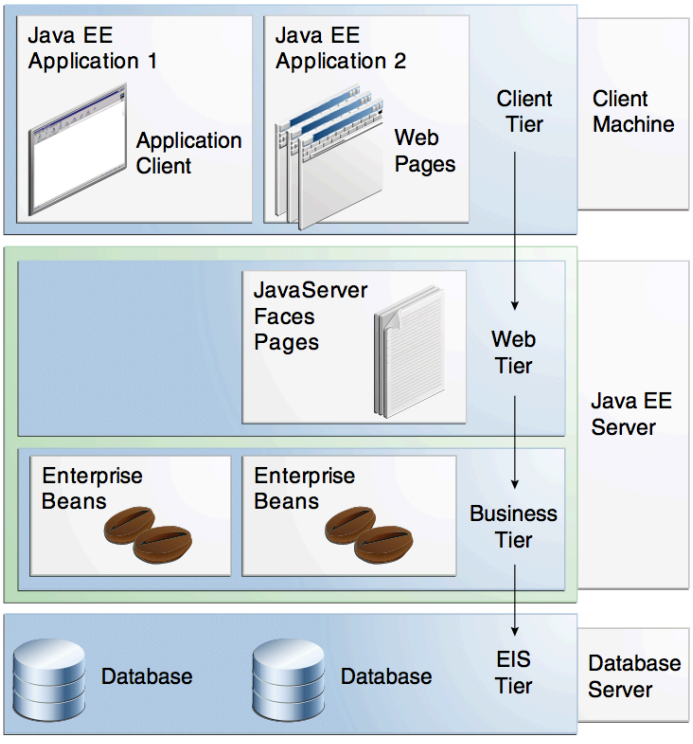
The purpose of this section is to illustrate how the software-to-be is going to be structured, from a variety of points of view:

- **Subsection 2.1** describes the software architecture induced by the Java EE 7 platform;
- **Subsection 2.2** focuses on the design of the user experience, i.e. the development of the navigation models for what will be *TravelDream Agent*'s website;
- **Subsection 2.3** contains a preliminary schema of the program's components.

2.1 Architecture overview

As stated in Subsection 2.4 of the Requirements Analysis and Specification Document, one of the constraints the project is subject to is the exploitation of the Java EE 7 platform. This not only imposes that the application be written in Java, but also heavily influences its internal structure.

Java SE is, in fact, less than suitable to the development of enterprise software. Because applications of this kind often imply – among others – the implementation of a client-server paradigm, parallelism, multiple simultaneous users and multiple external data sources, leaving the management of such aspects to software engineers and programmers would obligate the profusion of a lot of effort into the design of low-level non-functional aspects, to the detriment of the precision and definition of the application's functional kernel, i.e. the business logic.



PICTURE 3: JAVA EE 7 APPLICATION ARCHITECTURE

Java EE constitutes, instead, an extension of the Java SE APIs aimed at reducing the coding work for applications meeting enterprise criteria. It involves the use of a so-called Application Server, a piece of software in charge of handling the aforementioned non-functional details including database and network connections, user authentication and differentiation, remote method invocation and much more, and – above all – requires the software to conform to the platform's architectural prescriptions.

The Java EE architecture, graphically represented in Picture 3, stratifies an enterprise application into four tiers:

- The **Client Tier**, residing on the client machine, is responsible for the graphical user interface, which can either be web-based or consist of a separate application. In neither case does this layer meddle with business logic or data storage: only input and output facilities are provided at this level, delegating the execution of every other task to the lower tiers of the architecture.
- It is the **Web Tier**'s duty, instead, to collect requests from clients, to hand them the relative responses and to maintain sessions. If the JavaServer Faces specification has been adopted, Managed Beans are handled at this level through the use of Web Containers. This tier resides on the server machine and is still not in charge of anything related to the business logic.
- The **Business Tier**, as its name suggests, takes charge of all processing related to the business logic, hence constituting the functional core of a Java EE application. The adoption of the Enterprise JavaBeans model is mandatory in order to benefit from the platform's advantages: EJB Containers provide dynamic management of EJB components (instantiation, removal, activation/passivation…), ensuring the enforcement of resource injection mechanisms all the while. This layer also deals with persistent data, allowing the developer to abstract from SQL queries through the use of Entities.
- The **EIS Tier** handles the interactions with relational database servers and other data sources, if needed, and is obviously resident on the database server machine.

TravelDream Agent will therefore adopt the above architectural paradigm; for further details about supporting software (e.g., database management systems) see Subsection 2.1 of the Requirements Analysis and Specification Document.

2.2 User experience

Opting for the implementation of a web-based user interface is a lot more convenient than developing a separate application to be run on client machines. Flexibility, for instance, is greatly improved, because any change to the system – be it a simple maintenance update or one broadening the set of available functionalities – is immediately available to all users without requiring them to actually perform any upgrade procedure.

At this stage in the software development process, it is therefore important to provide a comprehensive plan of how users will interact with the system-to-be. While decisions on the graphical details can without a doubt be postponed to the development phase, the aim of this subsection is to clearly delineate the three essential aspects of a user interface:

- what **information** is displayed or required in each of the screens the user will deal with;
- which **operations** can be performed in each of those screens;
- what are the **consequences** of those operations, both in terms of graphical transitions and of internal state changes (e.g. writing to the database).

2.2.1 Notation

The navigation models shown in the following pages fulfil their task through the use of a "hybrid" UML notation: mostly borrowed from the Class Diagrams, it has been integrated with the **transition labels** found in Statecharts (i.e. *operation [condition for the transition to take place] / additional consequences*). **Classes** represent graphical elements: the fields area indicates the information provided or required in the interface, while the methods area obviously carries the operations that can be performed in that piece of interface; **abstract** classes have occasionally been employed in the following diagrams with

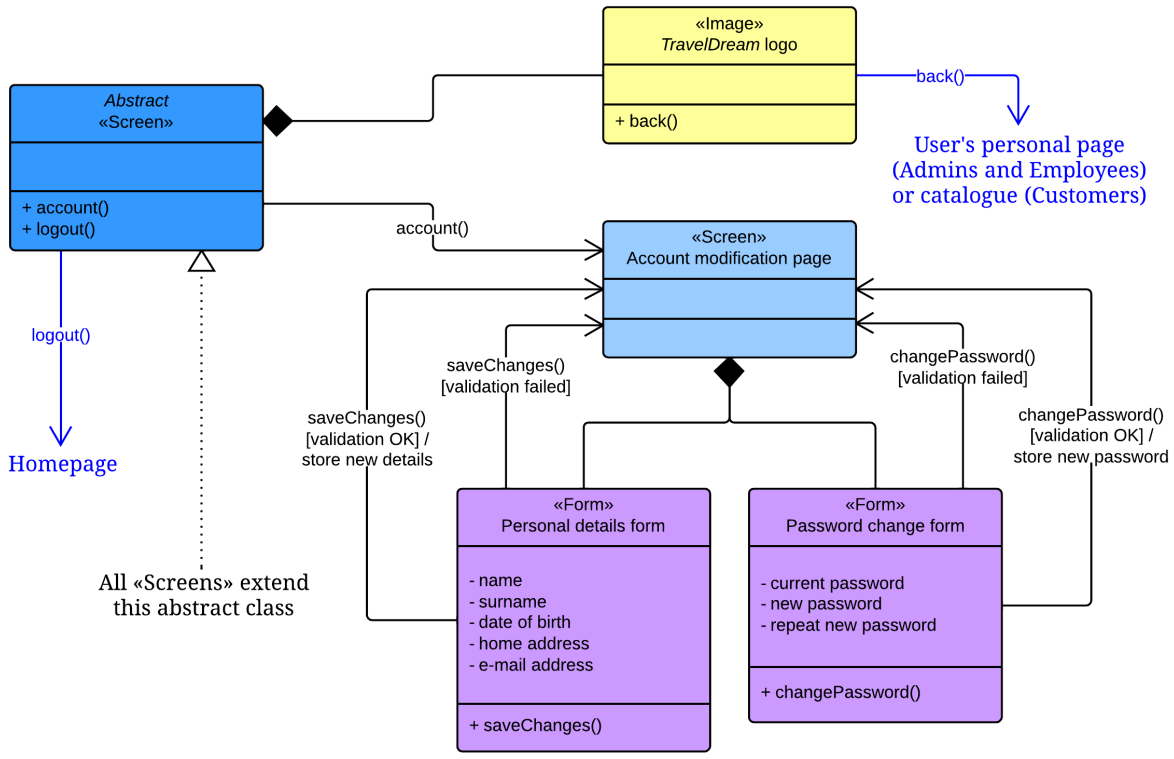
their usual meaning: they are actually never instantiated, but factor the features that their children have in common. All **associations** retain their original meaning, except for the directed association, which clearly models the transition between two graphical elements. Occasionally, a **Java-like syntax** has been adopted: arrays represent sets of elements, and the *.length* attribute indicates their cardinality.

Classes, though, are not all the same: different kinds of graphical elements have been distinguished thanks to five UML Stereotypes:

- **«Screen»** unsurprisingly models a web page. It is a sort of container for every other graphical element, and – as emerges from Picture 4 – features a couple of operations that every other «Screen» inherits. **«UnloggedScreen»**, only found in Picture 6, constitutes a slight variant of the «Screen» stereotype, in that it does not contain the *account()* and *logout()* operations.
- **«Form»** is always modelled as part of a «Screen»: as its name suggests, it is a section of a web page where the user is requested to enter some information, which can then be sent to the server by invoking an adequate operation.
- **«Pop-up»** is an alternative to a «Screen»: it allows new content to be shown without the need to leave the page and load a new one. «Pop-ups» can contain «Forms», just like «Screens».
- A **«ConfirmationDialog»** asks the user for a confirmation of their intention to proceed with the selected operation. It is particularly used in the case of deletions.
- Finally, **«Image»** simply models a picture. It is only used to show that all «Screens» include a clickable *TravelDream* logo bringing the user back to their homepage.

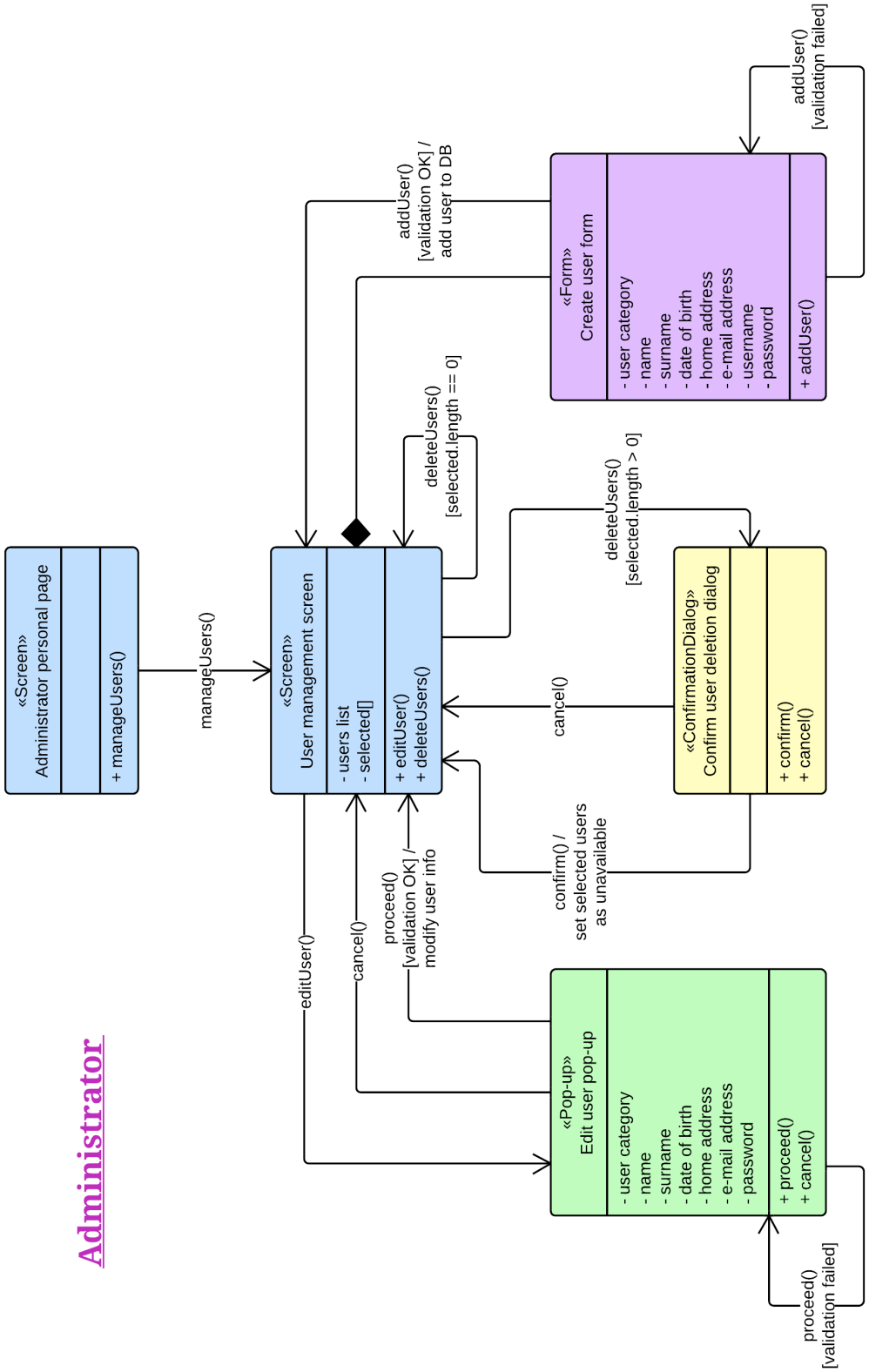
2.2.2 Navigation diagrams

The navigation model articulates over six different diagrams. The first one (Picture 4) illustrates the «Screen» stereotype. Blue transitions express inter-picture references.

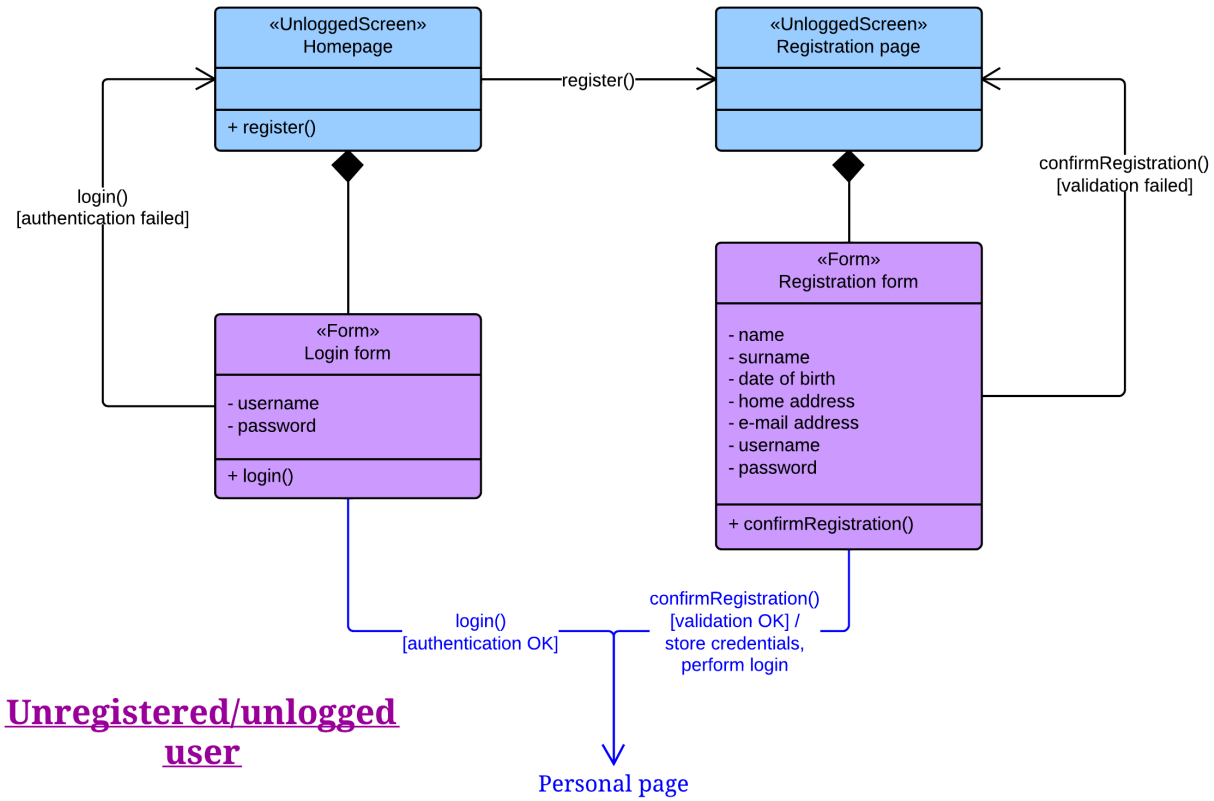


PICTURE 4: DESCRIPTION OF THE «SCREEN» STEREOTYPE

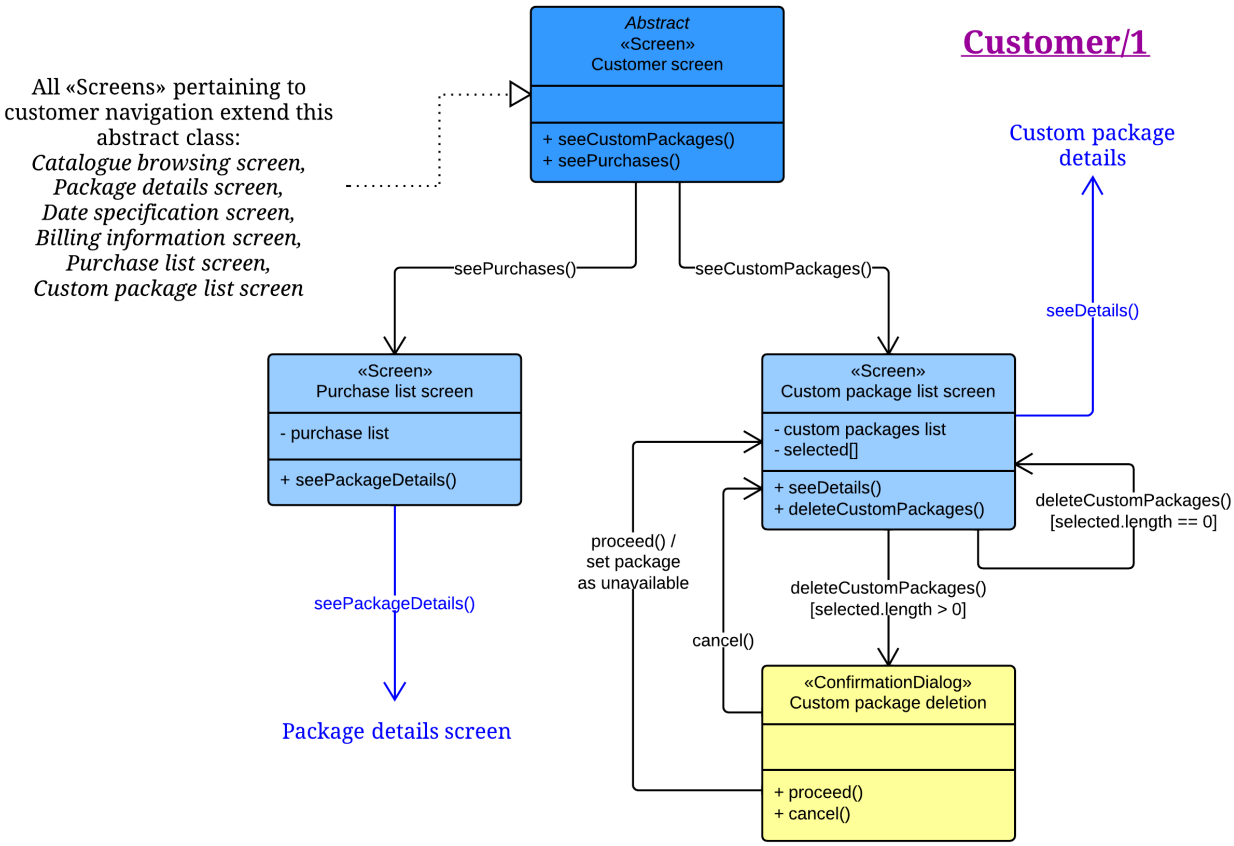
The following schemas all deal with user-specific navigation paths. The decision has been made to differentiate the diagrams on a user-category-based criterion, since it is logical to assume the sets of screens they access to be disjoint just like their respective functionalities. Picture 5 shows therefore the navigation paths for an Administrator, Picture 6 deals with unregistered or unlogged users, Picture 7 and Picture 8 illustrate the navigation possibilities Customers are endowed with and Picture 9 and Picture 10 do the same for Employees.



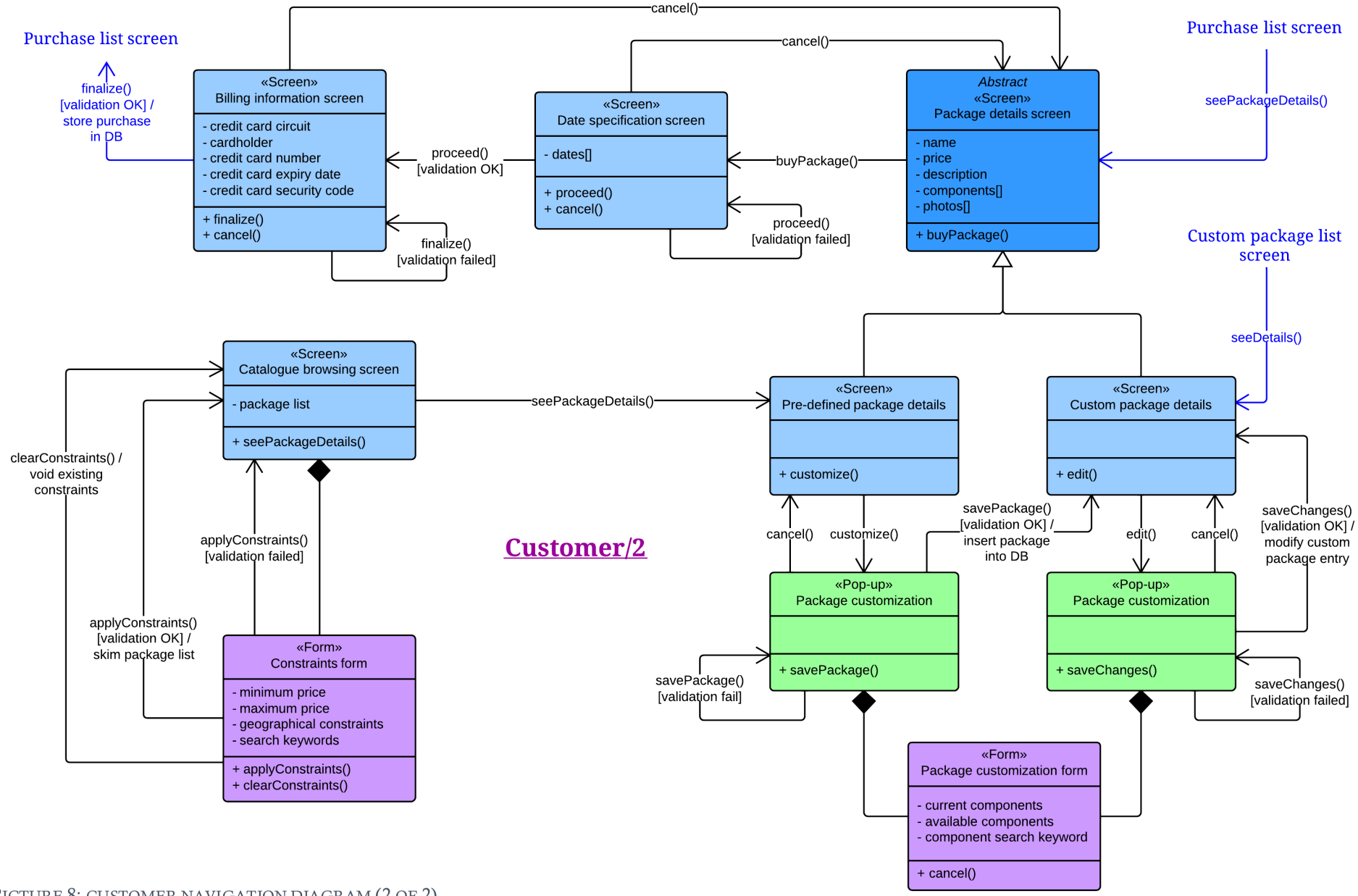
PICTURE 5: ADMINISTRATOR NAVIGATION DIAGRAM



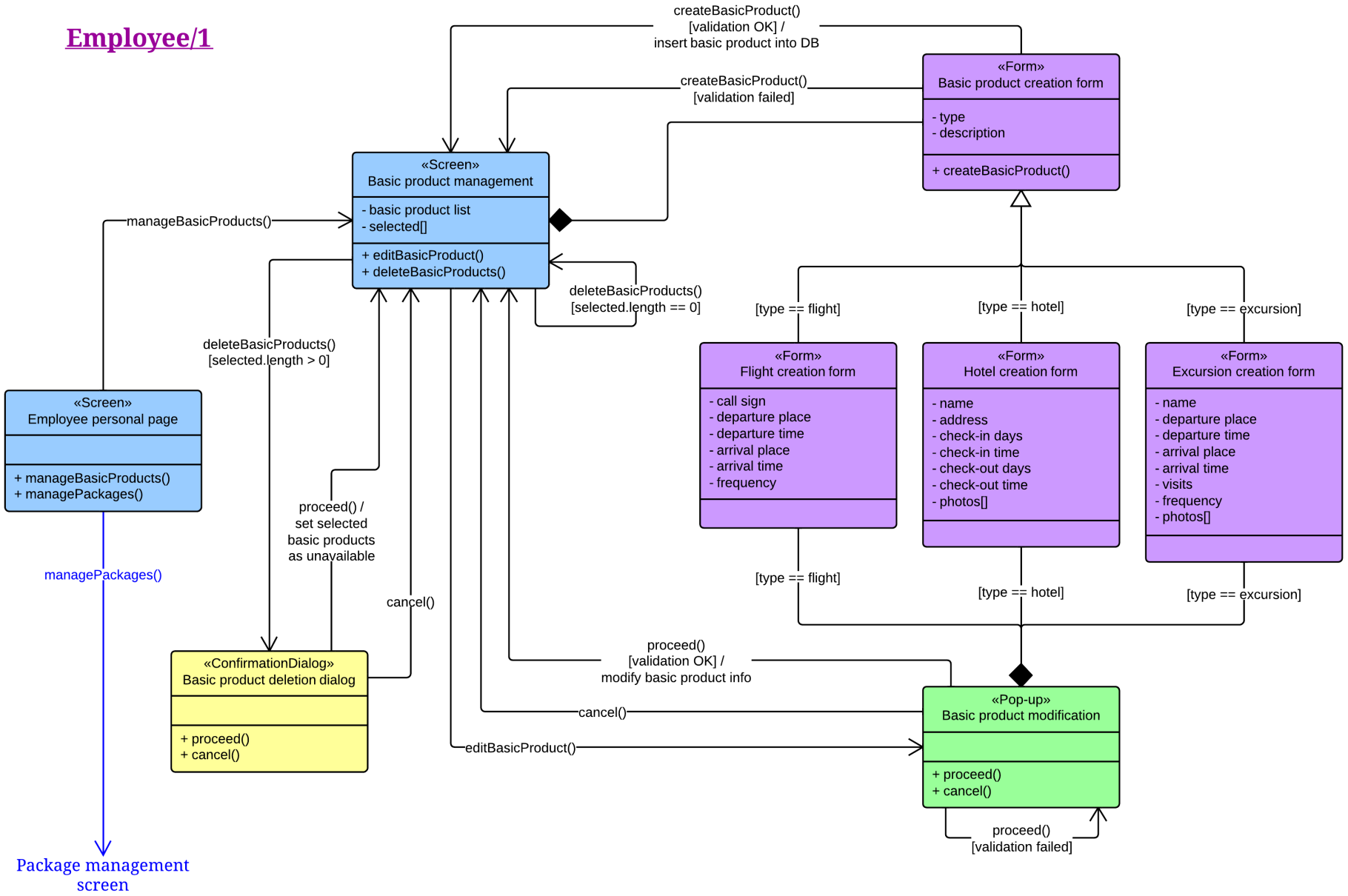
PICTURE 6: UNREGISTERED/UNLOGGED USER NAVIGATION DIAGRAM



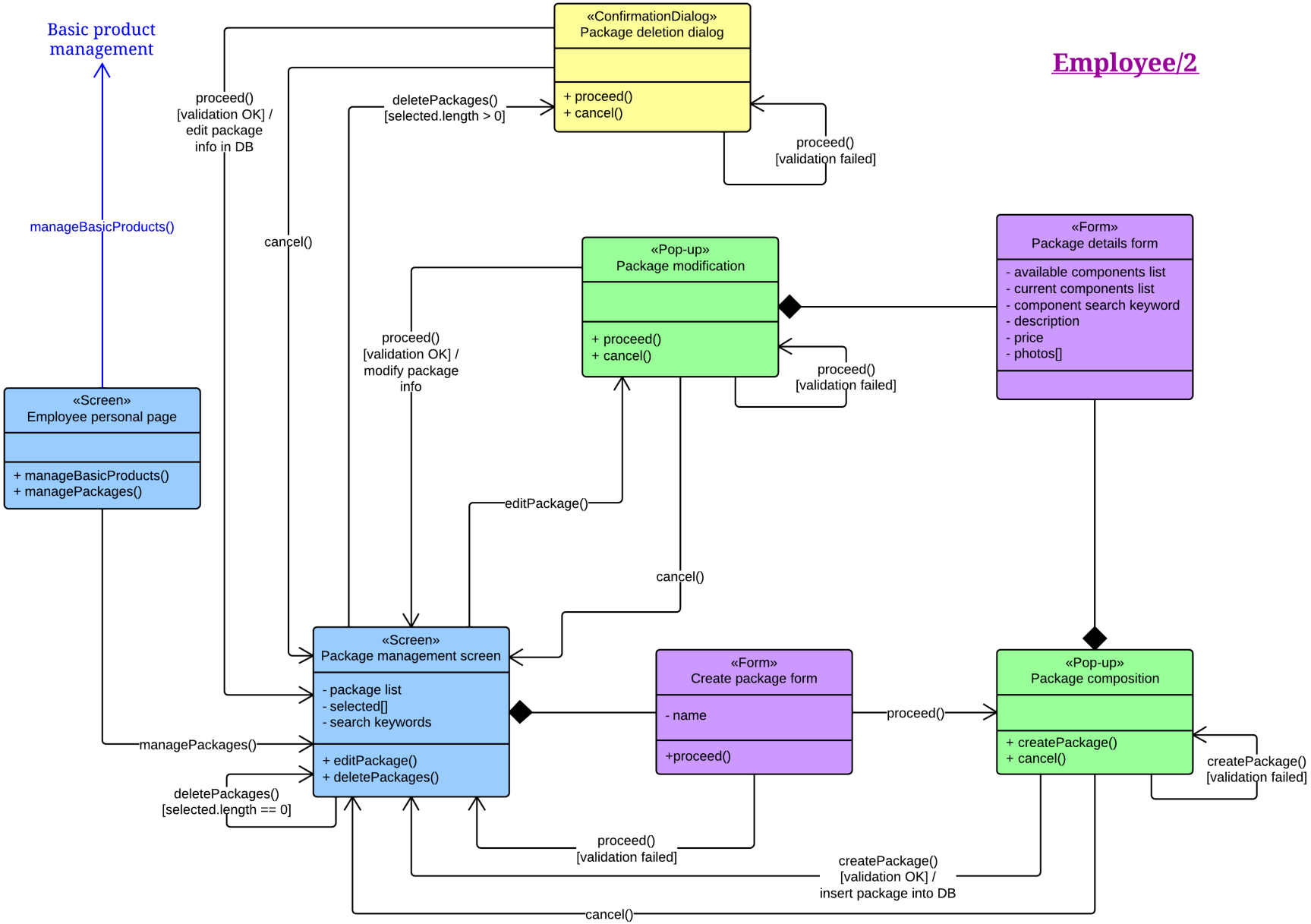
PICTURE 7: CUSTOMER NAVIGATION DIAGRAM (1 OF 2)



PICTURE 8: CUSTOMER NAVIGATION DIAGRAM (2 OF 2)



PICTURE 9: EMPLOYEE NAVIGATION DIAGRAM (1 OF 2)



PICTURE 10: EMPLOYEE NAVIGATION DIAGRAM (2 OF 2)

NOTE: *the success of any operation implying the submission of user-entered data to the server is subject to input validation. It is not the prerogative of a navigation diagram to specify the validation criteria for every single field; in general, validation will mainly consist of field coherence checks (e.g. alphabetic characters in numeric fields and vice-versa…).*

2.3 Component design

As was pointed out in Subsection 2.1, the Java EE platform imposes a stratified structure to any application being built pursuing to its standard. Such an architecture – along with the Java language itself – induces the enforcement of the **decoupling** design principle (among other ones): the modularity of software is in fact vital to its maintainability and expandability, especially when it comes to enterprise applications.

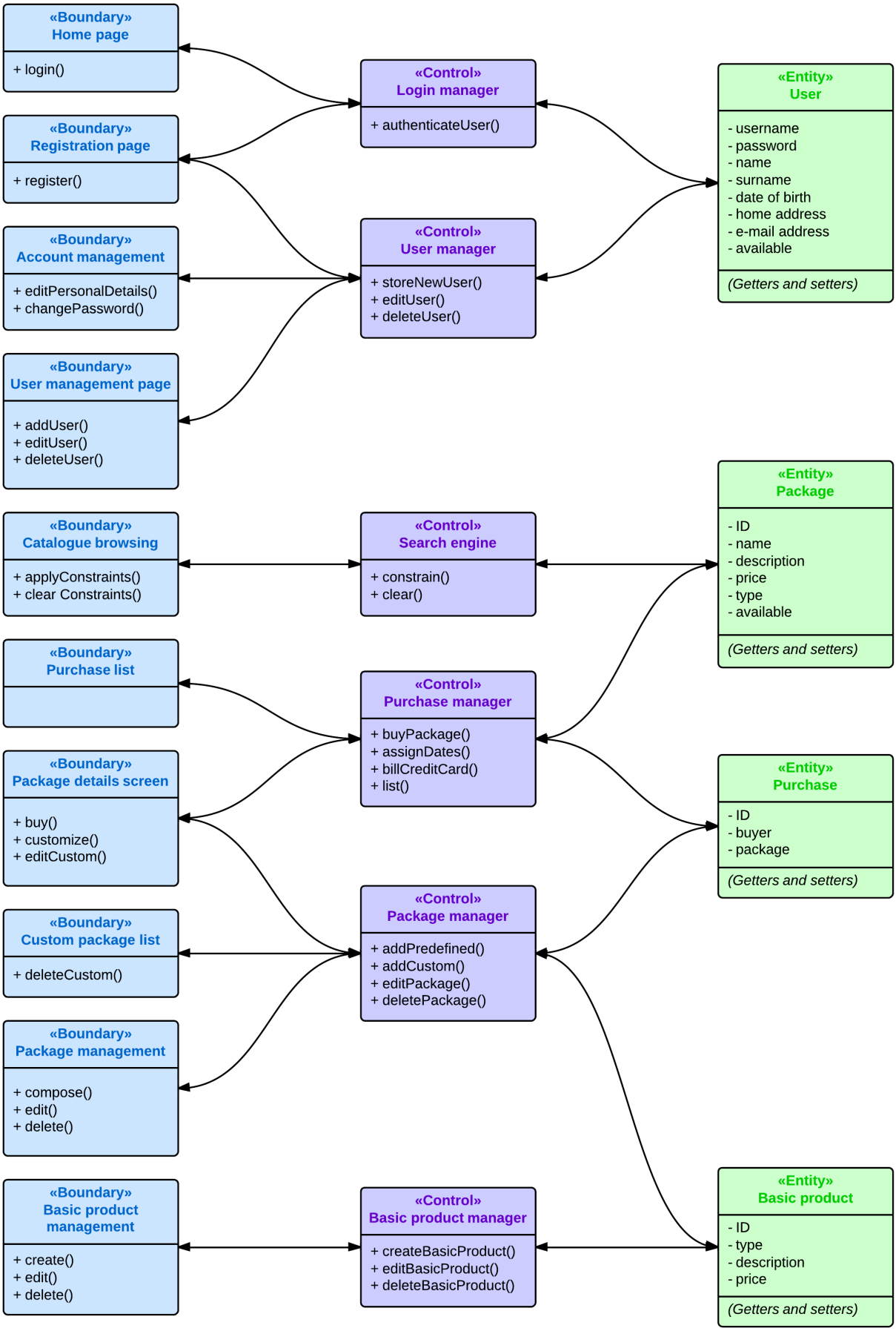
It is clear, furthermore, that Java EE's architecture rests on the Model-View-Controller paradigm:

- the **Model**, i.e. the representation of the fundamental data types the application will deal with, resides entirely in the so-called *Entities*, managed by the Business Tier;
- the **Controller**, consisting of the business logic, is also a prerogative of the Business Tier, but the pieces of software in charge of all processing tasks – namely, *SessionBeans* – are entirely disjoint from *Entities*;
- the **View**, meaning the graphical representation of data and of the interfaces allowing the user to perform their operations, is actualized by (X)HTML pages pertaining to the Web Tier.

The apportionment of responsibilities among the different components of the system-to-be is shown in the following diagram (Picture 11), whose elements belong to one of the following three UML stereotypes:

- **«Boundaries»** symbolize, as their name suggests, the frontier between the system and the outside world – i.e. the user. This accounts for the strong resemblance between «Boundaries» and the «Screens» found in the previous navigation models: they are nothing more than web pages enabling system-user interaction, collecting requests to be sent to the business logic and formatting responses to be shown on screen.
- **«Control»** elements are pieces of business logic. Their position in the component diagram testifies their role: they receive requests from users through the «Boundary» layer and process them, occasionally retrieving and/or storing the necessary data from «Entities» if needed. It is therefore clear that «Controls» will be mapped to *SessionBeans* in the software implementation.
- **«Entities»** are software components designed to the management of the data bases underlying the application, i.e. what Java EE also calls *Entities*. For the sake of simplicity, the component diagram features a subset of the Entities that will actually populate the final software: only the fundamental ones (*User*, *Package*, *Purchase* and *Basic product*) have been included.

The arrows linking the components have been made bidirectional in order to stress that their interactions are rigidly based on a request-response paradigm. It should be noted, finally, that drafting such a diagram implies a certain degree of familiarity with the Java EE platform which – in the author's humble opinion – cannot be fully achieved without "getting one's hands dirty" with some practice: the component diagram is therefore likely to be updated during the implementation phase, so as to reflect the real look of the software's structure.



PICTURE 11: COMPONENT DIAGRAM

Miscellanea

Changelog

Version 2 (February 2nd, 2014)

Added Subsection 1.4.

Version 1 (December 20th, 2013)

First release of the Design Document.

Software

The following software has been taken advantage of throughout the requirements phase of the project:

- **Microsoft Word for Mac 2011** enabled the creation of the document itself;
- **Lucidchart** (`https://www.lucidchart.com`) helped developing and rendering the entity-relationship, navigation and component diagrams.

Effort

A total of *34 hours and 30 minutes* were spent on the production of this document. In particular:

- 30 minutes for the Preamble;
- 15h30m for Section 1, including the development of the entity-relationship diagrams;
- 18h30m for Section 2, including the development of all diagrams therein.