TravelDream Project

Design Document

Riccardo B. Desantis - matr. 765106

Contents

Chapter 1

Introduction

1.1 Purpose

This document describes the general and specific architecture of the e-commerce system, called **TravelDream**, that is being developed for the TravelDream company to support its sale process.

This document will explain the architectural decisions and tradeoffs chosen in the design process and their justification.

1.2 Scope

The architectural descriptions provided concern the functional view, module view, deployment view, data layer, business logic and the user interface of the RASD. Hence the architecture will consider the following functionalities that TravelDream is going to handle:

- **Profiles**: TravelDream will manage personal data of the different types of users. Users can be customers, registered customers and employees.
- **Users**: TravelDream will manage registering, logging in/out of users.
- **Basic Components**: TravelDream will manage the list of basic component of which the travel packages are made of.
- **Travel Packages**: TravelDream will manage the list of travel packages that customers might buy.
- **Customized Travel Packages**: TravelDream will manage the list of travel packages that the customers have customized.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

Here are the definitions of the terms used:

- **Customer**: a customer of the company, a person willing to buy a travel package.
- **Registered Customer**: a registered customer in our system, that is a person which entered his/her data in our system.
- **Employee**: an employee of the TravelDream company.

- **Basic Components**: one of these elements:
  - a flight;
  - a car rental;
  - a train;
  - a hotel;
  - an excursion.
- **Travel Package**: a package composed of some basic products, created by an employee. We can assume that it can only be created or removed.
- **Customized Travel Package**: a travel package modified by a customer, with a specific flight (e.g., with a departing and returning date) etc.

1.3.2 Acronyms and abbreviations

The acronyms and abbreviations used are:

- **XML**: Extensible Markup Language.
- **XHTML**: Extensible HyperText Markup Language.
- **RASD**: Requirements Analysis and Specification Document.
- **QA**: Quality Attributes.
- **G**: Goal.
- **DBMS**: Database Management System.
- **RDBMS**: Relational Database Management System.
- **AS**: Application Server.
- **JEE**: Java Enterprise Edition.
- **EJB**: Enterprise Java Beans.
- **FR**: Functional Requirement.
- **NFR**: Non-functional Requirement.
- **QoS**: Quality of Service.
- **ER**: Entity Relationship.
- **LD**: Logical Design.
- **JAAS**: Java Authentication and Authorization Service.

1.4 Reference documents

The documents used for the writing of this RASD are:

1. Analysis document: rasd.pdf;
2. IEEE Standard for Information Technology-Systems Design-Software Design Description: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5167255.

1.5 Overview

This document specifies the architecture of TravelDream spreading from the general into the specific. The design was guided by a top-down process approach and the document structure reflects this tactic.

This document is organized as follows:

1. **Introduction**: provides a synopsis of the architectural descriptions.
2. **Design Overview**: provides a general description of TravelDream including its functionality and matters related to the overall system and its design.
3. **Design Considerations**: describes the design assumptions and constrains of TravelDream.
4. **Software Architecture**: specifies the general architecture, describes the basic structure and interactions of the main subsystems.
5. **Detailed System Design**: specifies in detail the components of the system through different architectural views.
6. **Appendixes**: provides supporting information and additional material.

Chapter 2

Design Overview

This section provides a general description of the software system including its functionality and concerns related to the overall system and its design.

2.1 Design context

The design context sets the limits for the system design, considering the functional and technological context.

2.1.1 Functionalities

The following functional requirements were identified during RASD. These functionalities are grouped by the following functional areas:

Managing profiles

- **Functional requirements**:
  - **FR1**: View personal information;
  - **FR2**: Modify personal information;

Managing users

- **Functional requirements**:
  - **FR3**: Register to the system;
  - **FR4**: Login;
  - **FR5**: Logout;
  - **FR6**: Modify password;
  - **FR7**: Recover password.

Managing components

- **Functional requirements**:
  - **FR8**: Add a component;
  - **FR9**: Remove a component;
  - **FR10**: Modify a component.
  - **FR11**: View a component.

Managing packages

- **Functional requirements**:
  - **FR12**: Add a package;
  - **FR13**: Remove a package;
  - **FR14**: Modify a package;
  - **FR15**: View a package;
  - **FR16**: Search packages.

Managing customized packages

- **Functional requirements**:
  - **FR17**: Customize a package;
  - **FR18**: Remove a customized package;
  - **FR19**: Update a customized package;
  - **FR20**: Search among customized packages.

2.1.2 System technologies

The TravelDream project will be designed considering the **client-server 3-tier distributed architectural style**. Each tier requires specific technologies as depicted below:

- **Web tier**:
  - Dynamic web pages containing XHTML, which are generated by web components.
  - Web components developed with Java Server Faces technology, which is a user interface component framework for web applications.
- **Business Logic tier**:
  - Java Enterprise Edition (JEE) 7 platform supports applications that provide enterprise services in the Java language. It is the common foundation for the various kinds of components in Java.
  - Enterprise Java Beans (EJB) 3.1, business components that capture the logic that solves or meets the needs of a particular business domain and persistence entities.
  - GlassFish AS 4.0, a server that provides services such as security, data services, transaction support, load balancing, and management of distributed applications, which supports the JEE7 platform.
- **Persistence tier**:
  - MySQL Server 5.6.14, a RDBMS.

2.2 General design description

This subsection presents the roadmap followed to model the architecture of TravelDream, including its functionality and matters related to the overall system and its design.

2.2.1 Design approach

The design approach is based on a client-server 3-tier distributed system, where each tier is described as follows:

- **Client tier**: this tier is responsible of translating user actions and presenting the output of tasks and results into something the user can understand.
- **Business Logic tier**: this tier coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the client and persistence tiers.
- **Persistence tier**: this tier holds the information of the system data model, and is in charge of storing and retrieving information from a database.

The design process followed a top-down process approach, so the outermost tiers were first identified and then broken into components that encapsulate the functionality. Hence each component is responsible for certain functionalities and interacts with others.

2.2.2 Overall design

This subsection presents the design model of TravelDream, specifying the basic relations between packages, use cases and users.

General package design

Since each tier is broken into components and each component is responsible for a set of functionalities that fulfill the requirements, there is a correlation between use cases (functionality) and package design. In the diagram in figure 2.1 we can identify three packages:

- **User UI**: this package contains the user interfaces. It is responsible for the interaction with the user such as getting UI requests, referring them to the Business Logic package and retrieving the data back for displaying.
- **Business Logic**: this package contains the business logic components. This package is responsible for handling the User UI package requests, processing them and accessing the Persistence package if required to provide a response.
- **Persistence**: this package is responsible for managing the data requests from the Business Logic package.

Employees, Registered and Unregistered customers access directly the User UI package and submit requests to accomplish their tasks.

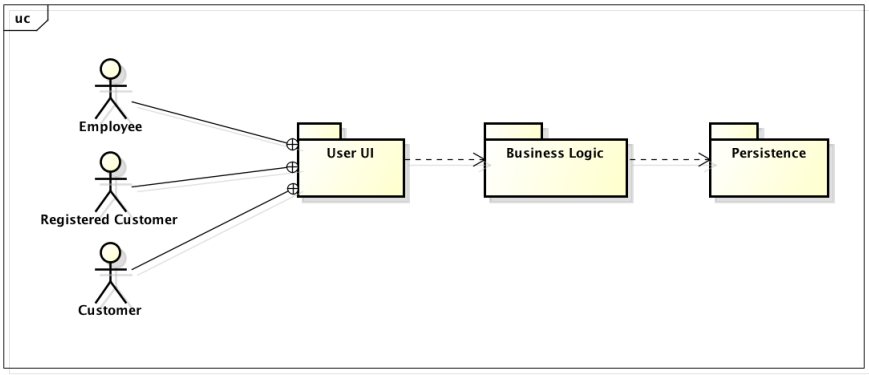

Figure 2.1: Basic package diagram.

Detailed package design

Given the functional requirements identified we can encapsulate them within specific components in the package diagram as follows:

- **User UI**: these set of sub packages are responsible for encapsulating the user actions and forwarding information requests to the Business Logic sub packages.
  - Profile pages: this package implements **FR1**, **FR2**;
  - User pages: this package implements **FR3**, . . . , **FR7**;
  - Components pages: this package implements **FR8**, . . . , **FR11**;
  - Packages pages: this package implements **FR12**, . . . , **FR16**;
  - Custom packages pages: this package implements **FR17**, . . . , **FR20**.
- **Business Logic**: these set of sub packages are responsible for handling requests from the User UI package, processing them and send back a response. These packages may access the Persistence package.
  - User manager: this package implements **FR1**, **FR3**, . . . , **FR7**;
  - Nomen manager: this package implements **FR2**, **FR4**;
  - Component manager: this package implements **FR8**, . . . , **FR11**;
  - Package manager: this package implements **FR12**, . . . , **FR16**;
  - Custom package manager: this package implements **FR17**, . . . , **FR20**.
- **Persistence**: this sub package contains the data model for the system. It accepts requests from the Business Logic package.
  - Entity manager: this package implements **F1**, . . . , **F20**.

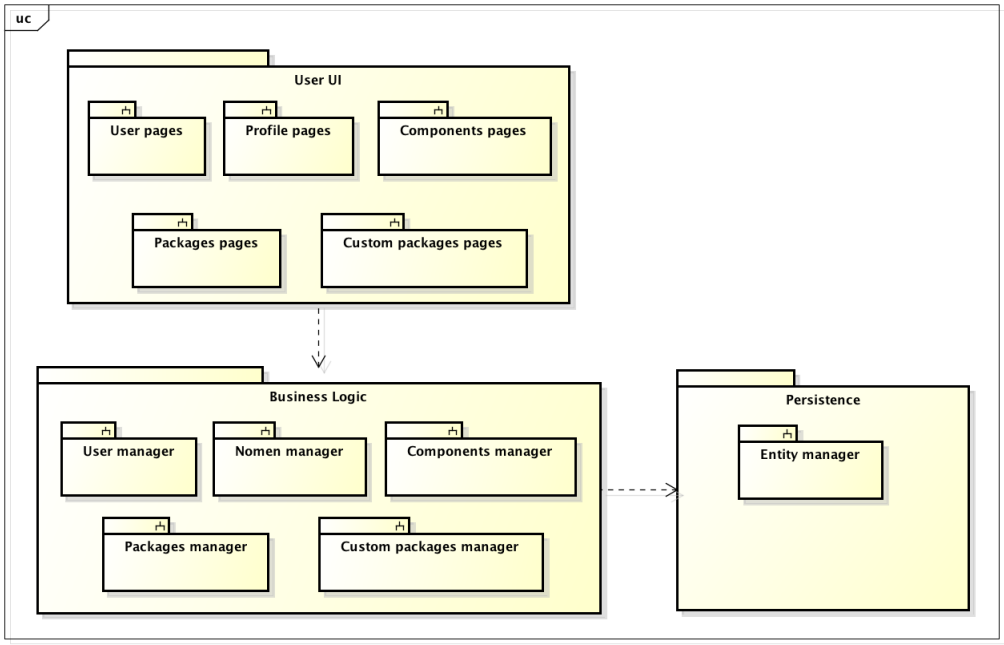

Figure 2.2: Detailed package diagram.

Chapter 3

Design Consideration

3.1 Assumptions

Assumption	Action
All the employees are administrators of the system.	At least one employee is registered at all times.

3.2 Dependencies

Dependency	Impact
Java virtual machine that supports JEE7 is already installed on the OS.	TravelDream only runs on operating systems that support the JEE7 platform.
the supported browsers will be Chrome and Safari.	TravelDream outputs XHTML code that requires browsers that support most of the web standards, elsewhere the UI experience will be affected.
A JEE7 AS is required on the server side.	TravelDream cannot operate if there is no AS that supports the JEE7 standard.

3.3 General constraints

This subsection describes the NFRs and the QoS details related to the design of the software product.

Element	Requirement
Memory	2+ GB
Database server	MySQL
Network	Internet Access, HTTP protocol
Security	The software product is controlled for each type of user. SSL is not supported.
Hard disk space	40+ GB

3.4 Performance requirements

3.4.1 Standard compliance

The software product does not have to meet any standard compliance.

3.4.2 Reliability

For assuring the reliability of the software product, it is mandatory to back up the database periodically.

3.4.3 Availability

An AS is used to guarantee availability of the software product. However in order to eliminate the downtime cause by the dependency on a single point of failure redundancy in the AS instances is recommended. For the first release of the software product we will assume that all the tiers run on the same physical server.

3.4.4 Security

The software product does not support SSL in AS. It supports the hashing of the user password according to sha1 algorithm in the database. It supports authorization according JAAS.

3.4.5 Maintainability

The architectural style and the component definition described contribute to low coupling and high cohesion of the software product.

3.4.6 Portability

The software product has been developed using the Java language and related dependent technologies. Java is specifically designed to have as few implementation dependencies as possible, meaning that code that runs on one platform does not need to be recompiled to run on another.

Chapter 4

Software Architecture

TravelDream shall be developed by using a general 3-tier JEE Architecture, as mentioned in section 2.2. We have conceptually identified the components of each, by dividing them in tiers which represent logically a clearer view of what each of them is responsible of. Section 4.1 gives a general description of the architecture we propose for our system and furthermore it provides a small description for each of the components. Detailed description is found in section 5. Detailed software design. Hereby we need to clarify that:

- Client Tier and Web Tier represent the Web Component;
- Business Logic Tier represents the Business Logic component;
- Persistence Tier represents Persistence Component;
- Database represents the data model.

4.1 Conceptual design

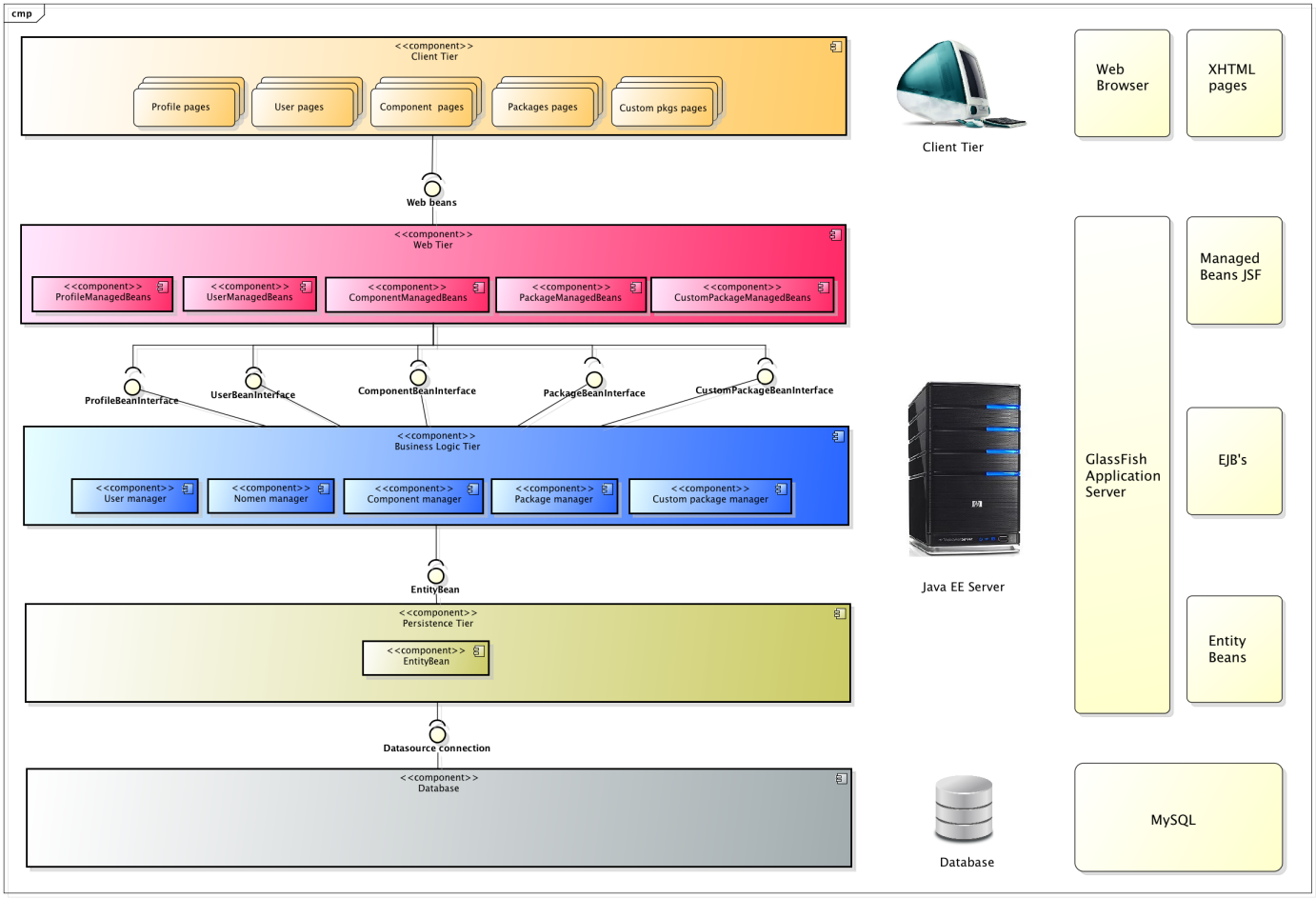

Figure 4.1: General architecture design.

The diagram in fig. 4.1 represents our conceptual design of TravelDream. This diagram depicts all the components of the designed software, by clarifying the logical separation between the tiers. It is clear that the user will interact with the XHTML pages, which in their side are implemented with beans to manage user interaction. This web interaction is then supported by the business tier, which holds on the information provided by the persistence layer. The persistence layer is the one in charge of the connection to the database and managing all the queries needed from the above layers.

Further explanation on the technologies used is found in section 4.2.

4.1.1 Client tier

The client tier is composed of the XHTML pages that the users will see. Actually this is strictly related to the Web Tier, and in section 5 it is detailed together with the Web Tier.

4.1.2 Web tier

Web Tier is composed of the web beans. This tier receives the requests of the user and has some specific beans which listen to these events and display data regarding the user requests. They interact with the beans in Business Tier, to retrieve the information. Since we are using JSF these beans are called Managed Beans.

4.1.3 Business logic tier

The business logic tier is composed of all the logic underlying our application; it is responsible of communication with Web Tier and Persistence Tier. Its components are the EJB Beans, named as Managers in section 5.1, just to differentiate from the web beans.

4.1.4 Persistence tier

The persistence tier is composed of the entity beans which represent the entities depicted from our RASD document and then further endorsed in our conceptual design. These entities are fundamental as they represent the connection to our database. Since in JEE we are interested in working in an object oriented environment, they represent a high level object view of the database of the application, which from its side connects to the Database Tier, to insert, update, delete, select.

4.1.5 Database

The database is composed of the tables composing the database we generated based on our assumptions and needs of the project.

4.2 System specification

The following table displays the technologies we will use during implementation. All of the technologies are free open source technologies.

Component Name	**Technology**
Client Tier Web Tier	XHTML integrated with Java Server Faces.
Business Tier	JEE with the AS GlassFish 4.0.
Persistence Tier Database Tier	MySQL 5.6

Chapter 5

Detailed Software Design

In this section we provide detailed insight into product structure and (intended) implementation regarding the general structure already detailed in section 4.1. We are releasing some of components from some implementation details since we are not very confident in JEE. In the following sections we have provided details about the general subcomponents, which are object of change in the further phase.

5.1 Implementation modules/components

Our project is composed of 3 main components, which shall be implemented during implementation phase:

- **Web component**;
- **Business logic component**;
- **Persistence component**.

5.1.1 Web component

In this section we provide the useful information for what needs to be implemented regarding this component. The following general diagram shows the subcomponents and their communication.

We have identified 4 main Subcomponents and their related Managed Beans:

- Profile pages,
- User pages,
- Component pages,
- Packages pages,
- Custom package pages.

We hereby remind that we are using JSF and that user events in the pages are managed by Managed Beans divided in 4 sections as well. The related beans Managed Beans are:

- ProfileManagedBeans,
- UserManagedBeans,
- ComponentManagedBeans,
- PackageManagedBeans,

- CustomPackageManagedBeans.

These beans represent the conceptual idea of the Managed Beans, as there will be more managed beans that will be needed during implementation.

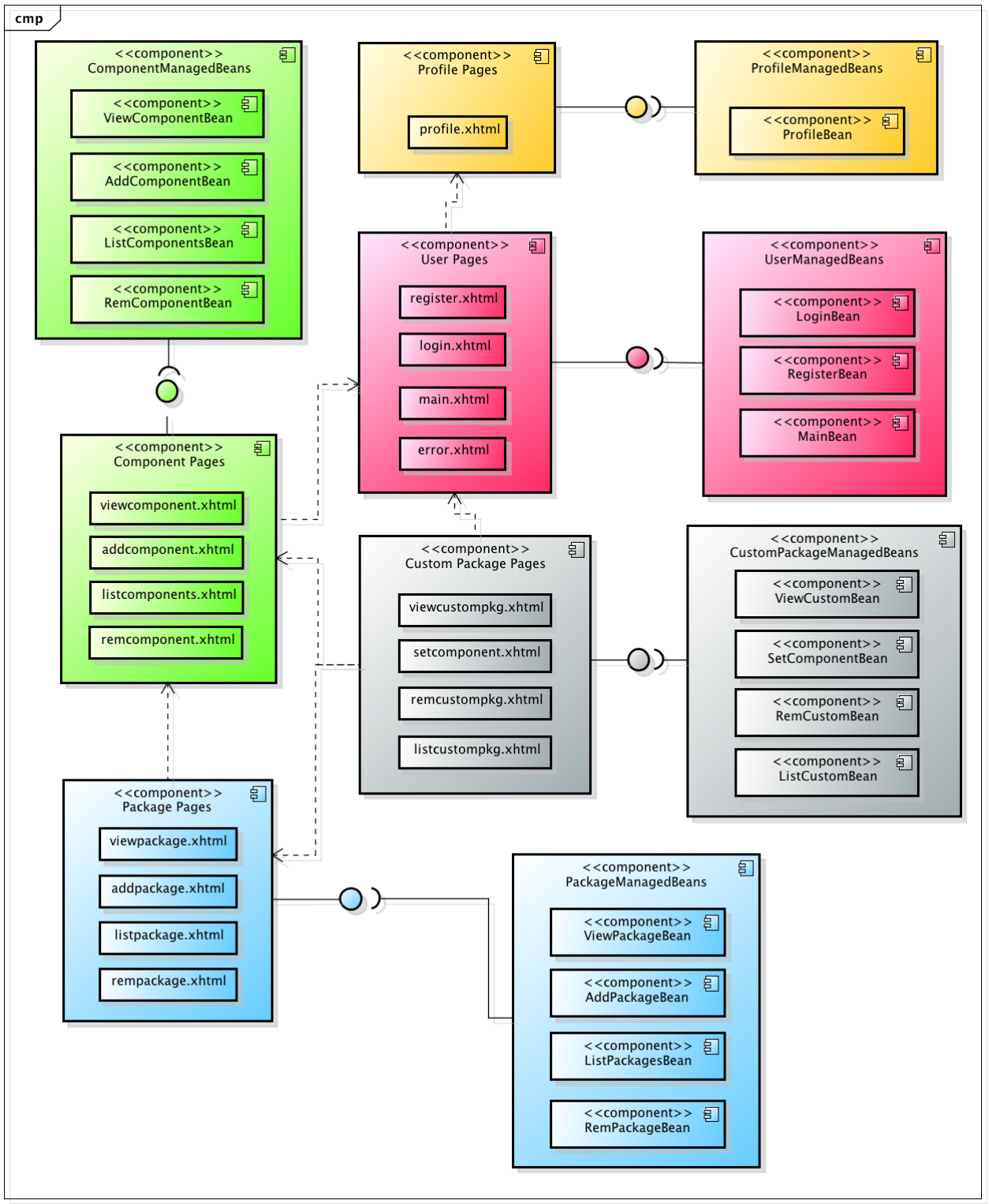

Figure 5.1: Web tier components.

Profile pages and managed beans

Pages and beans in this section are responsible for everything related to the profiles.

Component Name	**profile.xhtml**
Definition	User interface for displaying user profile information.
Responsibilities	• Display profile information. • Allow modification to the profile.

Component Name	**ProfileBean**
Definition	Managed bean for displaying user profile information.
Responsibilities	• Load profile information based on the user. • Handle modification to the profile.

User pages and managed beans

Pages and beans in this section are responsible for everything related to users.

Component Name	**register.xhtml**
Definition	User interface for user registration.
Responsibilities	• Load registration form.

Component Name	**login.xhtml**
Definition	User interface for user login.
Responsibilities	• Display login form.

Component Name	**main.xhtml**
Definition	User interface for displaying the main menu for a user.
Responsibilities	• Display user main homepage (according to the user role); • Display connections to Profile, Packages, Customized Packages pages.

Component Name	**error.xhtml**
Definition	User interface for displaying error messages.
Responsibilities	• Load type of error because of user input.

Component Name	**RegisterBean**
Definition	Managed bean for user registration.
Responsibilities	• Load registration form; • Check compulsory data; • Redirect to error.xhtml or main.xhtml.

Component Name	**LoginBean**
Definition	Managed bean for user login.
Responsibilities	• Load login form; • Check login credentials; • Redirect to error.xhtml or main.xhtml.

Component Name	**MainBean**
Definition	Managed bean for displaying the main menu for a user.
Responsibilities	• Load user main homepage (according to the user role); • Load connections to Profile, Packages, Customized Packages pages.

Component pages and managed beans

Pages and beans in this section are responsible for everything related to the components.

Component Name	**viewcomponent.xhtml**
Definition	User interface for displaying a component.
Responsibilities	• Display the details about a specific component; • Display the link to remcomponent.xhtml for removing such component if the user has the rights for doing it (e.g. is an employee).

Component Name	**addcomponent.xhtml**
Definition	User interface for adding a new component.
Responsibilities	• Display the form for adding a new component; • Redirects to viewcomponent.xhtml at the end of the adding process.

Component Name	**listcomponents.xhtml**
Definition	User interface for listing all the available components.
Responsibilities	• Display a list of all the available components, where for each of them there's a link to the specific viewcomponent.xhtml page; • A search bar is available for finding a subset of components.

Component Name	**remcomponent.xhtml**
Definition	User interface for removing a specific component.
Responsibilities	• Display a confirmation form for removing a specific component; • Redirects to listcomponents.xhtml or viewcomponent.xhtml either if the component was removed or not.

Component Name	**ViewComponentBean**
Definition	Managed bean for displaying a component.
Responsibilities	• Load the details about a specific component; • Load the link to remcomponent.xhtml for removing such component if the user has the rights for doing it (e.g. is an employee).

Component Name	**AddComponentBean**
Definition	Managed bean for adding a new component.
Responsibilities	• Load the form for adding a new component; • Redirects to viewcomponent.xhtml at the end of the adding process.

Component Name	**ListComponentsBean**
Definition	Managed bean for listing all the available components.
Responsibilities	• Load the list of all the available components, where for each of them there's a link to the specific viewcomponent.xhtml page; • Load a subset of the available components if a search query is provided.

Component Name	**RemComponentBean**
Definition	Managed bean for removing a specific component.
Responsibilities	• Load a confirmation form for removing a specific component; • Redirects to listcomponents.xhtml or viewcomponent.xhtml either if the component was removed or not.

Packages pages and managed beans

Pages and beans in this section are responsible for everything related to the packages.

Component Name	**viewpackage.xhtml**
Definition	User interface for displaying a package.
Responsibilities	• Display the details about a specific package; • Display a list of links to viewcomponent.xhtml for each component; • Display the link to rempackage.xhtml for removing such package if the user has the rights for doing it (e.g. is an employee); • Display the link for "buying" and customizing this package if the user has the rights for doing it (e.g. is a registered customer).

Component Name	**addpackage.xhtml**
Definition	User interface for adding a new package.
Responsibilities	• Display the form for adding a new component; • Display a list of components that can be added, and a link to addcomponent.xhtml; • Redirects to viewpackage.xhtml at the end of the adding process.

Component Name	**listpackages.xhtml**
Definition	User interface for listing all the available packages.
Responsibilities	• Display a list of all the available packages, where for each of them there's a link to the specific viewpackage.xhtml page; • A search bar is available for finding a subset of packages.

Component Name	**rempackage.xhtml**
Definition	User interface for removing a specific package.
Responsibilities	• Display a confirmation form for removing a specific package; • Redirects to listpackage.xhtml or viewpackage.xhtml either if the package was removed or not.

Component Name	**ViewPackageBean**
Definition	Managed bean for displaying a package.
Responsibilities	• Load the details about a specific package; • Load a list of links to viewcomponent.xhtml for each component; • Load the link to rempackage.xhtml for removing such component if the user has the rights for doing it (e.g. is an employee); • Load the link for "buying" and customizing this package if the user has the rights for doing it (e.g. is a registered customer).

Component Name	**AddPackageBean**
Definition	Managed bean for adding a new package.
Responsibilities	• Load the form for adding a new package; • Load a list of components that can be added, and a link to addcomponent.xhtml; • Redirects to viewpackage.xhtml at the end of the adding process.

Component Name	**ListPackagesBean**
Definition	Managed bean for listing all the available packages.
Responsibilities	• Load the list of all the available packages, where for each of them there's a link to the specific viewpackage.xhtml page; • Load a subset of the available packages if a search query is provided.

Component Name	**RemPackageBean**
Definition	Managed bean for removing a specific package.
Responsibilities	• Load a confirmation form for removing a specific package; • Redirects to listpackages.xhtml or viewpackage.xhtml either if the package was removed or not.

Custom packages pages and managed beans

Pages and beans in this section are responsible for everything related to the custom packages.

Component Name	**viewcustompkg.xhtml**
Definition	User interface for displaying a custom package.
Responsibilities	• Display the details about a specific custom package; • Display a list of links to setcomponent.xhtml for each component; • Display the link to remcustompkg.xhtml for removing such package.

Component Name	**setcomponent.xhtml**
Definition	User interface for customizing a component in a customized package.
Responsibilities	• Display the form for customizing a component (dependently of the type of the component, e.g. departing date, etc.); • Redirects to viewcustompkg.xhtml at the end of the customization process.

Component Name	**listcustompkg.xhtml**
Definition	User interface for listing all the customized packages.
Responsibilities	• Display a list of all the customized packages, where for each of them there's a link to the specific viewcustompkg.xhtml page; • A search bar is available for finding a subset of packages.

Component Name	**remcustompkg.xhtml**
Definition	User interface for removing a specific custom package.
Responsibilities	• Display a confirmation form for removing a specific custom package; • Redirects to listcustompkg.xhtml or viewcustompkg.xhtml either if the package was removed or not.

Component Name	**ViewCustomBean**
Definition	Managed bean for displaying a package.
Responsibilities	• Load the details about a specific custom package; • Load a list of links to setcomponent.xhtml for each component; • Load the link to remcustompkg.xhtml for removing such component.

Component Name	**SetComponentBean**
Definition	Managed bean for customizing a component in a customized package.
Responsibilities	• Load the form for customizing a component (dependently of the type of the component, e.g. departing date, etc.); • Redirects to viewcustompkg.xhtml at the end of the customization process.

Component Name	**ListCustomBean**
Definition	Managed bean for listing all the customized packages.
Responsibilities	• Load the list of all the customized packages, where for each of them there's a link to the specific viewcustompkg.xhtml page; • Load a subset of the customized packages if a search query is provided.

Component Name	**RemCustomBean**
Definition	Managed bean for removing a specific custom package.
Responsibilities	• Load a confirmation form for removing a specific custom package; • Redirects to listcustompkg.xhtml or viewcustompkg.xhtml either if the package was removed or not.

5.1.2 Business logic component

In this section we provide the useful information for what needs to be implemented regarding this component. The following general diagram shows the subcomponents and their communication. We have identified 5 main subcomponents:

- User manager,
- Nomen manager,
- Component manager,
- Package manager,
- Custom package manager.

Each of these beans will be defined as an EJB Bean by defining all the required methods in the specific beans.

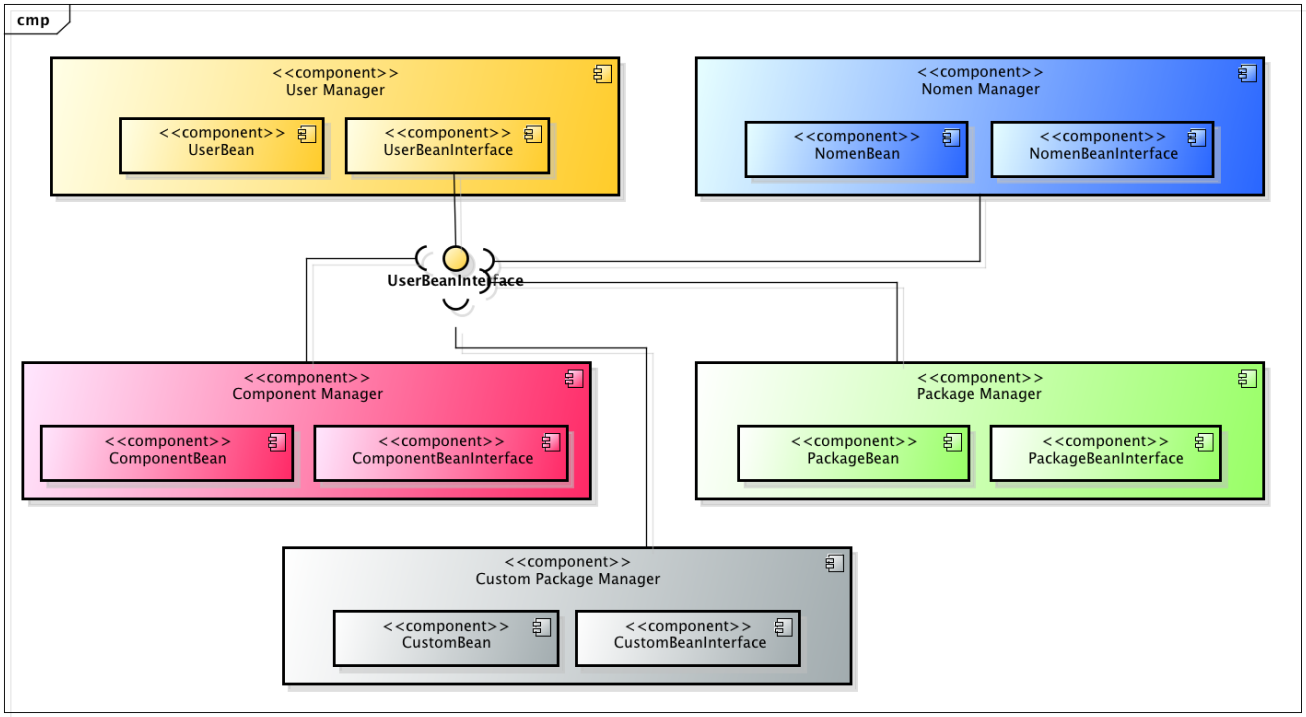

Figure 5.2: Business tier components.

User manager

Component Name	**UserBeanInterface**
Definition	Interface instantiated every time communication between beans is needed.
Responsibilities	• Communicate with NomenBeanInterface, ComponentBean-Interface, PackageBeanInterface, CustomBeanInterface.

Component Name	**UserBean**
Definition	Bean in charge for all the functionalities related to users.
Responsibilities	• Login user; • Logout user; • Register a new user; • Load profile information for the user.

Nomen manager

Component Name	**NomenBeanInterface**
Definition	Interface instantiated every time communication between beans is needed.
Responsibilities	• Communicate with UserBeanInterface.

Component Name	**NomenBean**
Definition	Bean in charge for all the functionalities related to nomenclature entities.
Responsibilities	• Load list of available countries, cities, etc.

Component manager

Component Name	**ComponentBeanInterface**
Definition	Interface instantiated every time communication between beans is needed.
Responsibilities	• Communicate with UserBeanInterface.

Component Name	**ComponentBean**
Definition	Bean in charge for all the functionalities related to the components.
Responsibilities	• Load list of available components; • Load the information on a single component; • Add a new component; • Remove a component.

Package manager

Component Name	**PackageBeanInterface**
Definition	Interface instantiated every time communication between beans is needed.
Responsibilities	• Communicate with UserBeanInterface.

Component Name	**PackageBean**
Definition	Bean in charge for all the functionalities related to the packages.
Responsibilities	• Load list of available packages; • Load the information on a single package; • Add a new package; • Remove a package.

Custom package manager

Component Name	**CustomBeanInterface**
Definition	Interface instantiated every time communication between beans is needed.
Responsibilities	• Communicate with UserBeanInterface.

Component Name	**CustomBean**
Definition	Bean in charge for all the functionalities related to the custom packages.
Responsibilities	• Load list of customized (by the specific user) packages; • Load the information on a single customized package; • Customize a component of the package; • Remove a customized package.

5.1.3 Persistence component

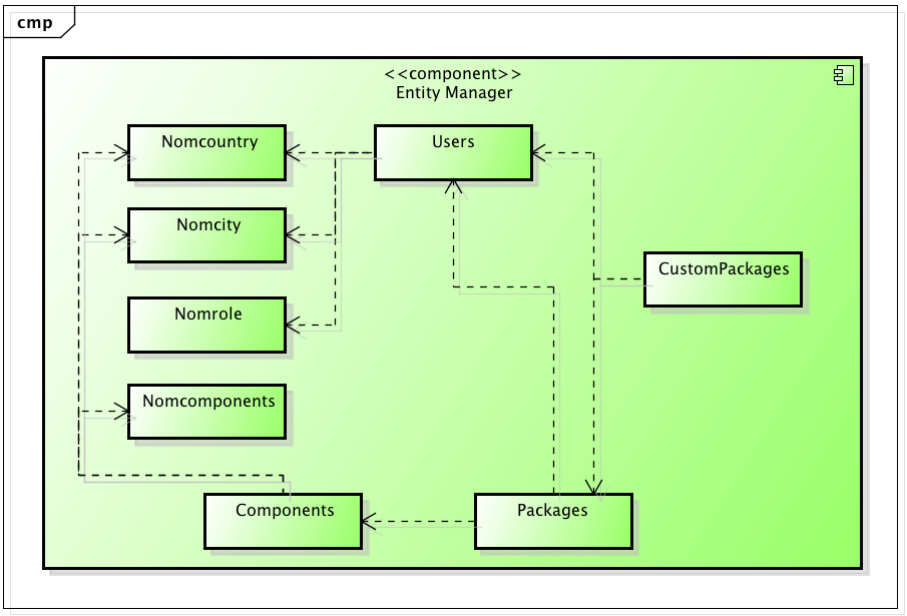

Figure 5.3: Persistence component diagram.

In the Persistence component diagram are shown all the entities in the application. All of these entities represent a high level object view of the tables in the database, and likewise we are providing an object oriented model for the database, that will be used in our application. So, all of the entities have inside all the attributes that are specified in the associated table in the database.

In the following table there is a short description of the responsibilities of each of the entities from above.

Entity	Responsibilities
Users	Represents the users in the system.
Nomcountry	Represents a nomenclature of all the countries in the system.
Nomcity	Represents a nomenclature of all the cities in the system.
Nomrole	Represents a nomenclature of all the roles in the system.
Nomcomponents	Represents a nomenclature of all the types of components in the system.
Components	Represents the components in the system.
Packages	Represents the packages in the system.
CustomPackages	Represents the custom packages in the system.

5.2 Database model

In the following section we will provide a detail description of the database by providing its both views: the **conceptual** and the **logical design diagrams**.

5.2.1 Conceptual design

The conceptual design of the database is represented with the Entity Relationship Diagram (ER) given bellow. In the following text we will provide a description of the elements in the diagram. Let us clarify that:

- 1:1 means one to one relation;
- 1:M means one to many relation;
- M:M means many to many relation.

Let's start by describing all the elements in the ER diagram. A common characteristic for all entities is that they have primary key named id. In the ER diagram, we can see that all entities have an underlined attribute id.

The diagram has four Nomen entities:

- Nomcity,
- Nomcountry,
- Nomrole,
- Nomcomponent,

each of them representing a specific form of data in our system, called **nomenclatures**. Specifically, Nomcomponent represent the type of components available (e.g. flight, train, etc.), and the parameters are going to be dealt with via a JSON specification:

```
{
    "param1": { "type": "string", "value": "" },
    "param2": { "type": "date", "value": "" },
    ...,
    "paramn": { "type": "int", "value": "" }
}
```

Nomrole has also a level because it could be possible to give access to same area based on the numerical level (`if (level > 5) ...`).

The Users entity represents the information for the users in the system. Besides the main attributes, we want to know which role the user belongs to, so we have a 1:M relation between the Users and Nomrole, because one user can has only one role, but many users can have that same role.

The Components entity represents the information for the components in the system. Other than the name and some other details, there are three 1:M relations with Nomcity, Nomcountry and Nomcomponent, because a component can have one city, one country and one type, but that same city, country and type can be the element in some other component. The defaults field will contain a copy of the parameters in the associated Nomcomponent, with some of the parameters set with a default value.

The packages entity represents the information for the packages in the system. The only interesting bit is the M:M relation with the Components entity, because a package is made of many component, but a component can be part of more than one package.

Lastly, the CustomPackages entity represents the information for the customized packages in the system. There's a 1:M relation with Users and Packages, because a CustomPackage is associated to one user and one package, but there can be a number of customized packages from the same package, and one user can have more than one customized package. The configuration field follows the JSON specification above, where the parameters will finally have the customized value per user.

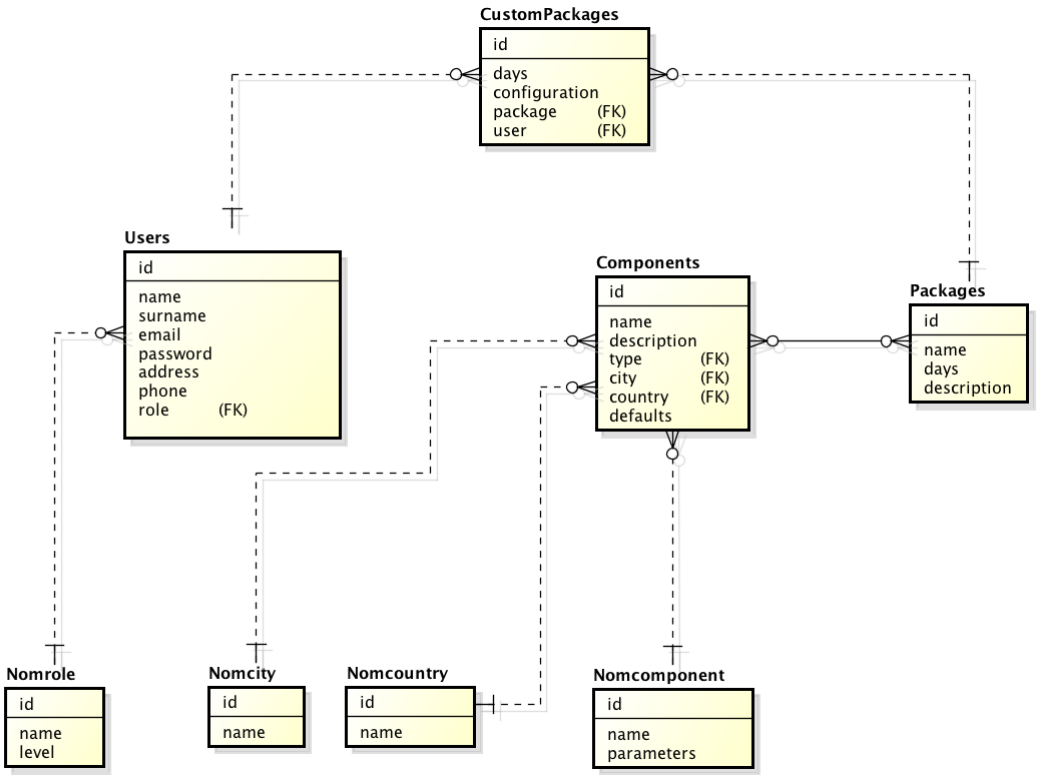

Figure 5.4: Conceptual ER of the database.

5.2.2 Logical design

The logical design (LD) diagram represents the final implementation of the database, and it is based on the conceptual ER diagram. The LD diagram, showed below, is generated automatically by using the MySQL workbench tool. Using the conceptual model to create the database will means that the entities will be translated into tables, and the attributes of each of the entities will become columns in the tables. Furthermore, the M:M relations will become tables in the LD diagram, and the 1:M relations will be model by using foreign keys.

In the LD diagram we see all the tables in the database. For all of them, the common characteristic is that the primary key is of type integer, and is an autoincrement column. That means that when inserting new tuples into the table, we are not going to take care about the value of the id column, because it will be assigned automatically by the database management system (DBMS).

The users table, besides the main attributes, has one foreign key to the table Nomrole, and likewise we are implementing the 1:M relation from the conceptual model of the database. This key has an integer column value associated (role), because we created all of the primary keys in the database as integer autoincrementing columns.

The components table, besides the main attributes, contains the foreign keys to the 1:M relations with Nomcity, Nomcountry and Nomcomponent (respectively with the integer valued fields city, country, type).

The packages table has nothing interesting other than the main attributes. The componentinpackage table is the one taking care of the relationship between the components and the packages (that is the M:M relation considered in the ER), using the two foreign keys to the components and packages table (respectively with the fields component and package).

The custompackages table has the two foreign keys to the associated package and the user that created it (represented in the 1:M relations), with the fields package and user.

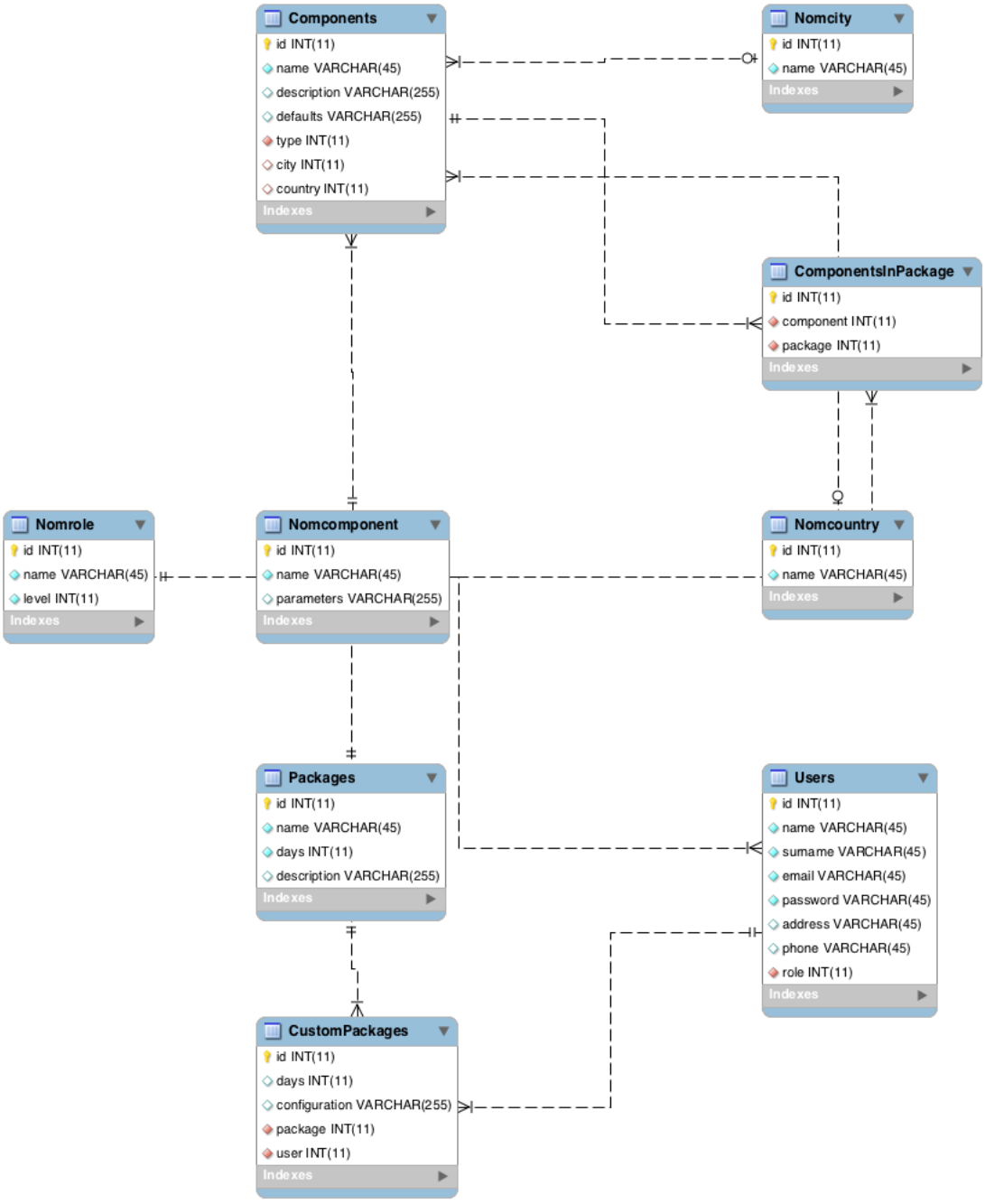

Figure 5.5: Logical design of the database.

5.3 Web site organization

The diagram in fig. 5.6 describes the site navigation in the TravelDream project.

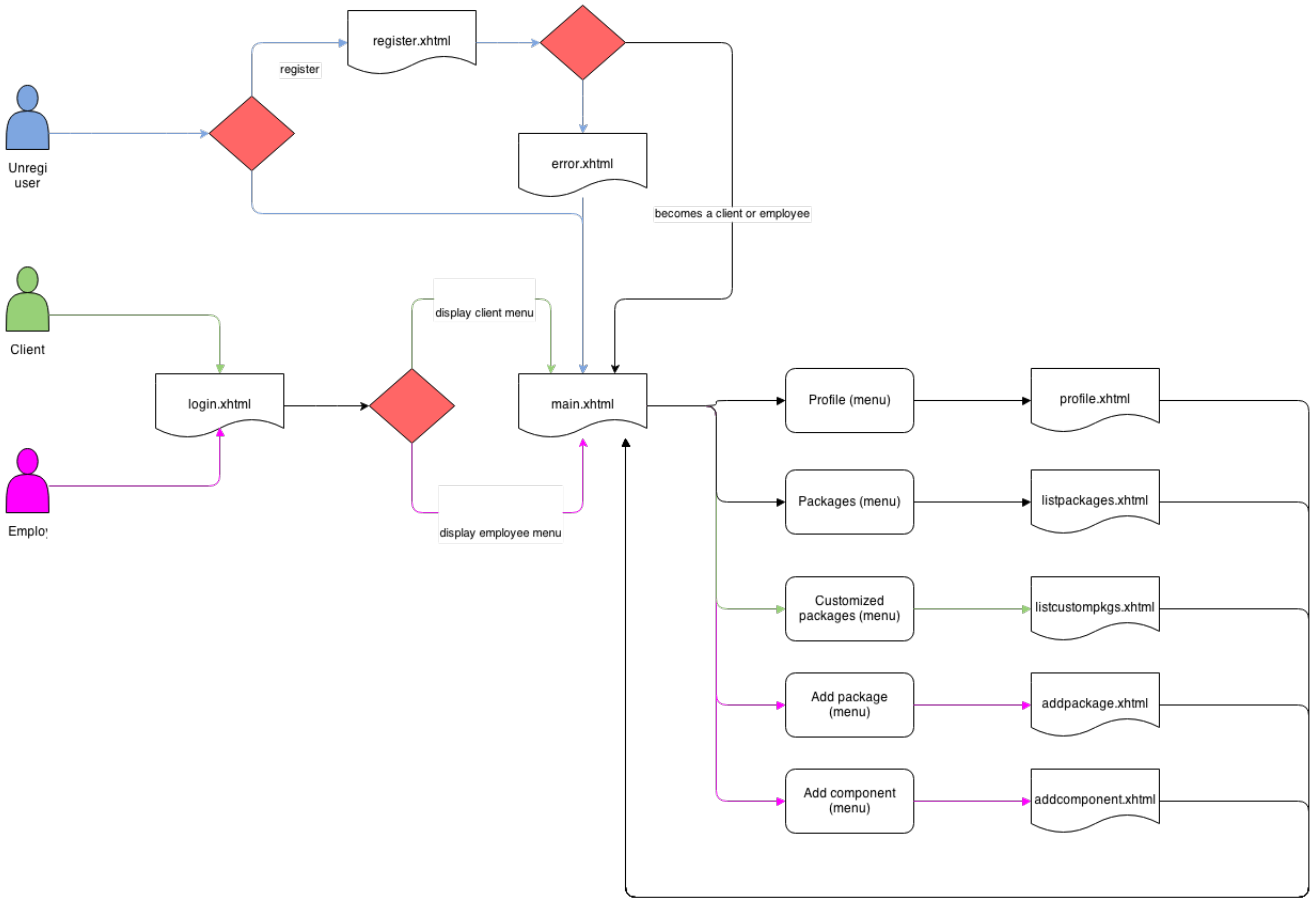

Figure 5.6: Web site navigation diagram.

5.4 Runtime view

The diagram in fig. 5.7 describes the runtime view of the TravelDream project. The software product that will be released will be traveldream.ear that can be deployed in any JEE Application Server.

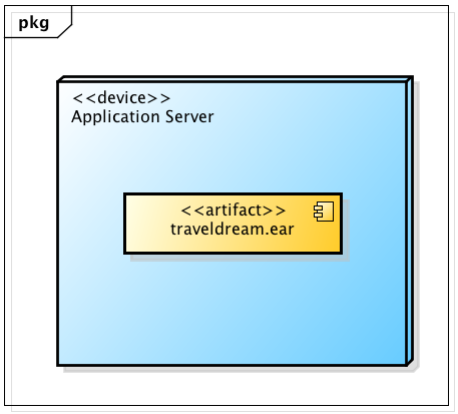

Figure 5.7: Runtime view of TravelDream.

5.5 Deployment view

The diagram in fig. 5.8 shows the deployment view of our software product.

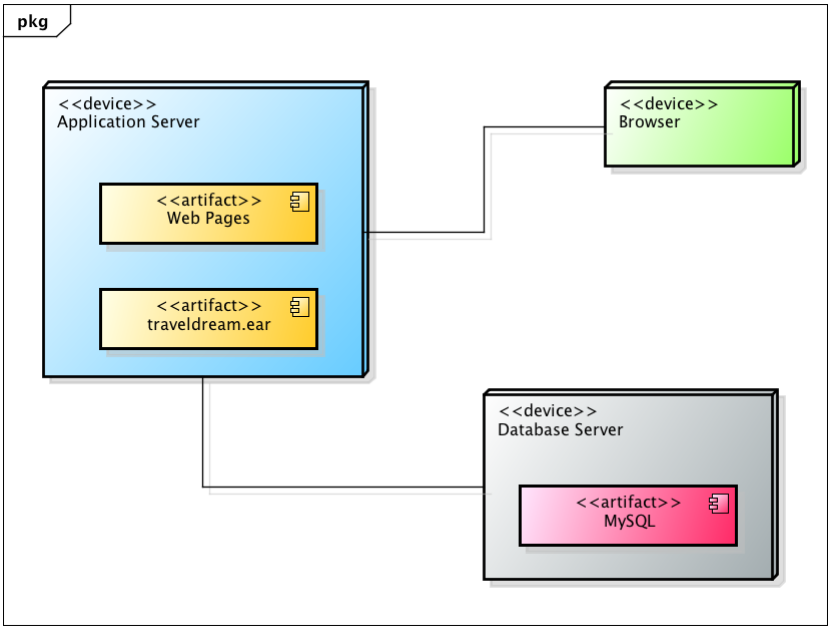

Figure 5.8: Deployment view of TravelDream.

5.6 Module view

The diagram in fig. 5.9 describes how the source code will be organized.

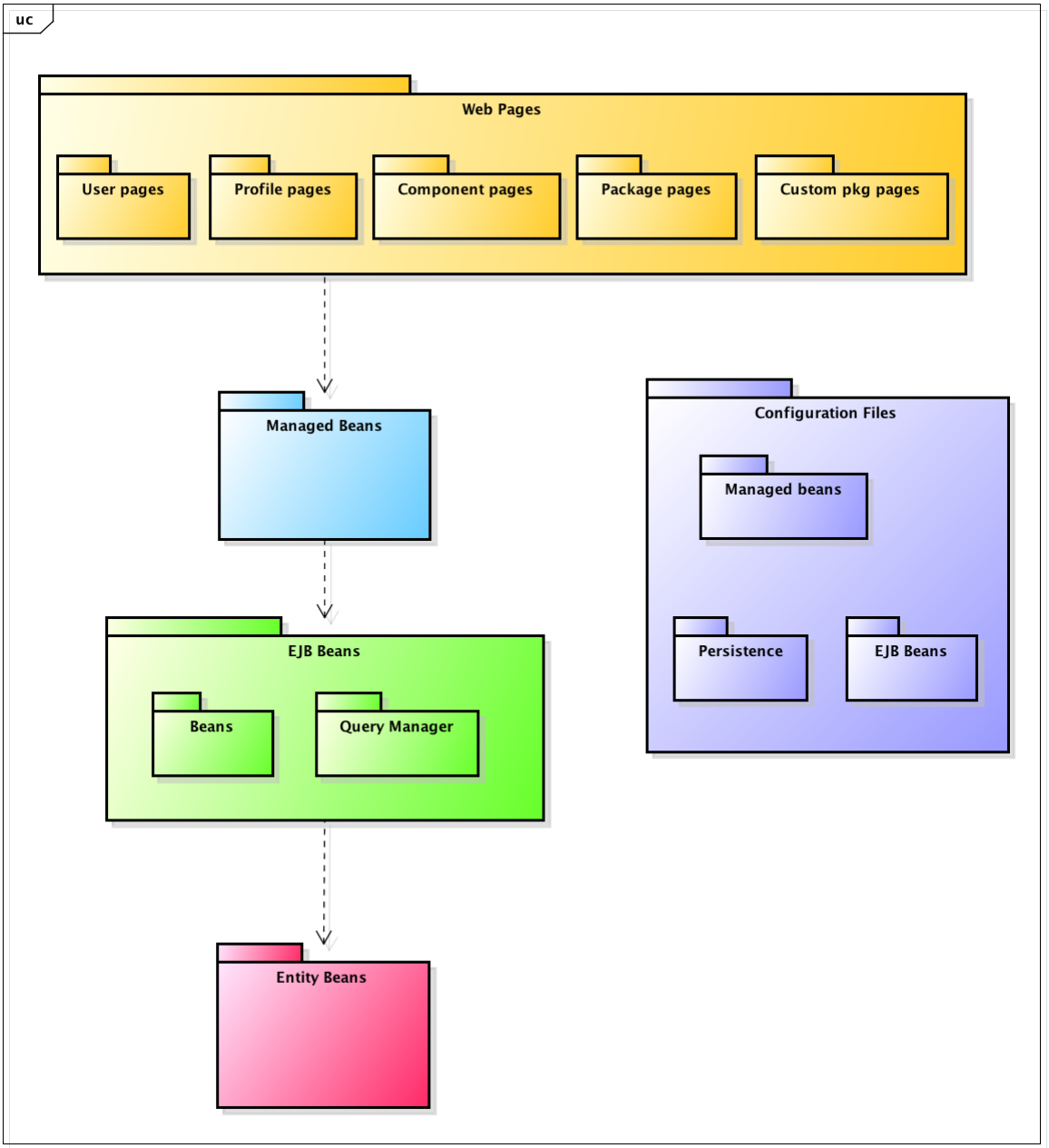

Figure 5.9: Source packages diagram.

Chapter 6

Appendixes

6.1 RASD modification

None at the moment.