Project #2

Enterprise Application Integration

João Ribeiro
David Cruz
Filipe Norte

18/11/2012

1 Introduction

The current report provides a description of the solution implemented for project 2. It includes static and dynamic perspectives of the architecture and an explanation of some of the major design decisions.

2 Architecture Description

Dynamic View

The figure below shows the dynamic view of the system:

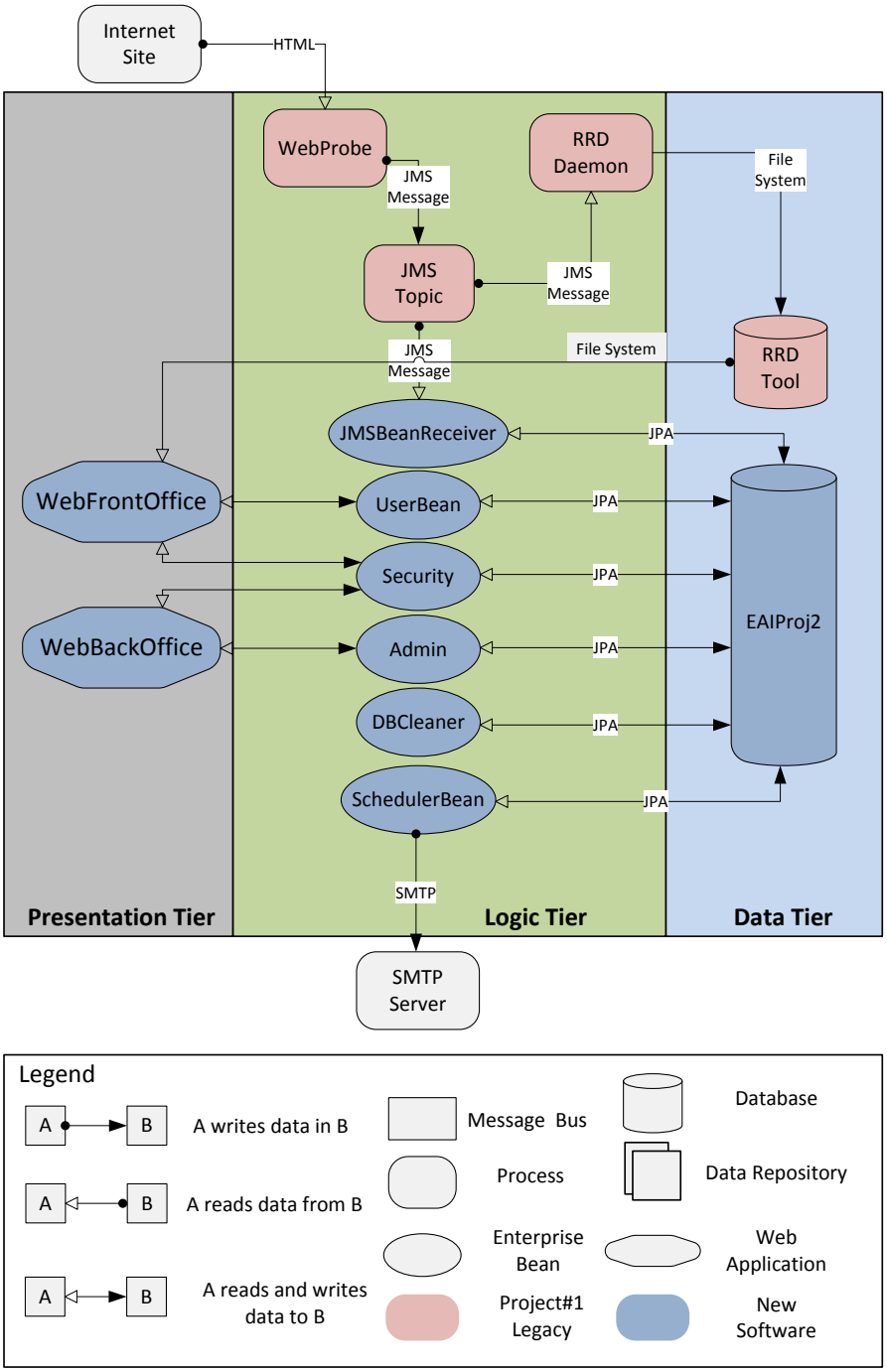

Figure 1 – Dynamic view of the system

Static View

The project is composed by different packages with different responsibilities on the overall project. The following table gives a brief overview of their responsibilities and relevant information:

Project	Package	Responsibilities
EAICommon	**eai.msejdf. config**	Handles all the configuration settings required by the system
	eai.msejdf. data	The data model object that was used internally by the system. Contains the xsd schema definition of the data model.
	eai.msejdf. utils	Package responsible for various utilities used by the system. Includes classes for handling the xml/xsl transformations, property loading, string operations and sending emails.
	eai.msejdf. web	Responsible for handling the session management
EJBServer	**eai.msejdf.admin**	Package to handle the BackOffice admin operations
	eai.msejdf.jms	This package is responsible for handling the Java Messaging System (JMS) interactions (receiving of messages through JMS)
	eai.msejdf.security	Package for security related calls, such as login, registration, etc.
	eai.msejdf.timer	Package to handle the operations that are timely scheduled
	eai.msejdf.user	Package to handle the FrontOffice user operations
EJBServerInterfaces	**eai.msejdf.admin**	Provides the bean interfaces for the admin operations
	eai.msejdf.exception	Implementation of specific exceptions
	eai.msejdf.security	Provides the bean interfaces for security related operations
	eai.msejdf.sort	The Enum that defines the order by clauses.
	eai.msejdf.user	Provides the bean interfaces for the user operations
JPAData	**eai.msejdf.persistence**	JPA Entity implementations
WebBackOffice	**eai.msejdf.web.backoffice.admin**	Admin related integrations with Logic Tier
	eai.msejdf.web.backoffice.security	Security related interactions with Logic Tier
	Webcontent	Web site definition
WebFrontOffice	**eai.msejdf.pagedata**	Core classes per data/operation type for interactions with Logic Tier. See design decisions
	eai.msejdf.validator	Data validation operations
	eai.msejdf.pages	Beans (non EJB) for each HTML page containing code for interactions with the EJBs. Derived from pagedata classes. See design decisions
	Webcontent	Web site definition

Table 1 – Package list and responsibilities

Some of these packages are composed of different structures and classes that define object instances used internally by the systems. A brief detail of their structure and design will be described in the following sections for the main or most structured packages.

eai.msejdf.persistence package

The persistence package contains all the POJO classes that are used with the JPA system for storing and retrieving all the data from the database. Some of the classes have direct representation to the database model, but others are simplification objects that persist into different tables than described in the POJO. One example of this is the StockInfo class that is associated with the company, but that is stored in the same table on the database schema defined. This approach helps on handling the data within the java code allowing to direct assignment on the database model.

The following diagram describes the structure of these classes:

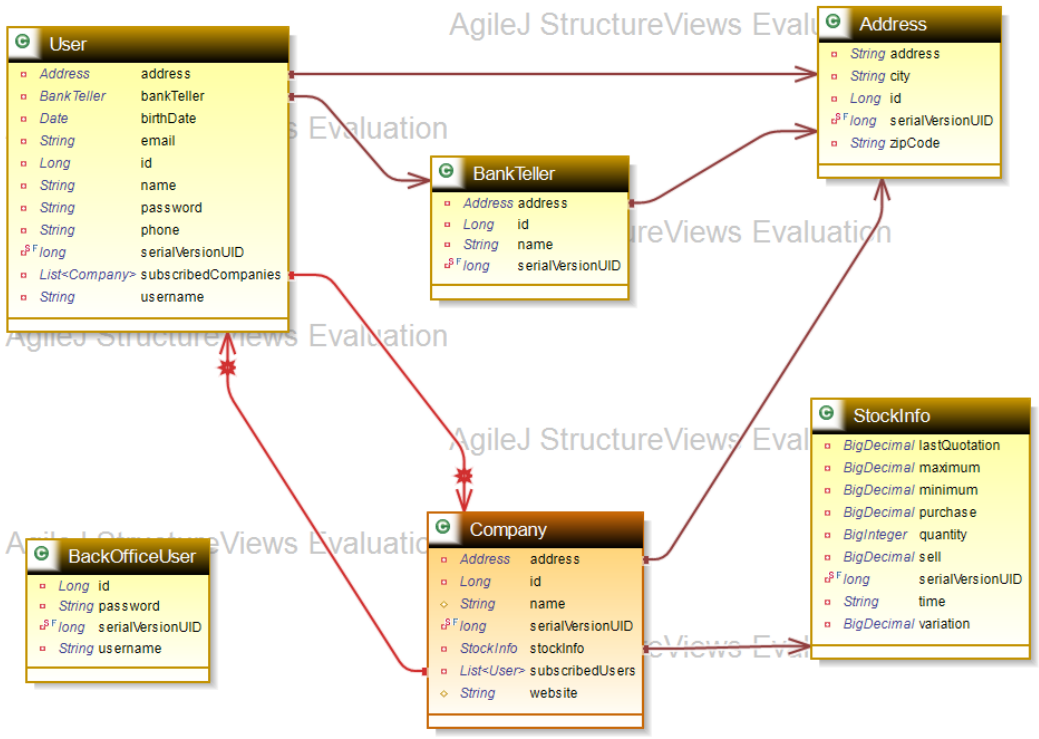

Figure 2 – Persistence Static view

EJBServerInterfaces and EJBServer

These projects define the overall structure of the EJB middle tier of the system. The EJBServer interfaces define the contract methods that provide the clients the ability to call any of the internal EJB implementations stored in the JBoss system. The following diagram describes the Server and client interfaces of the system.

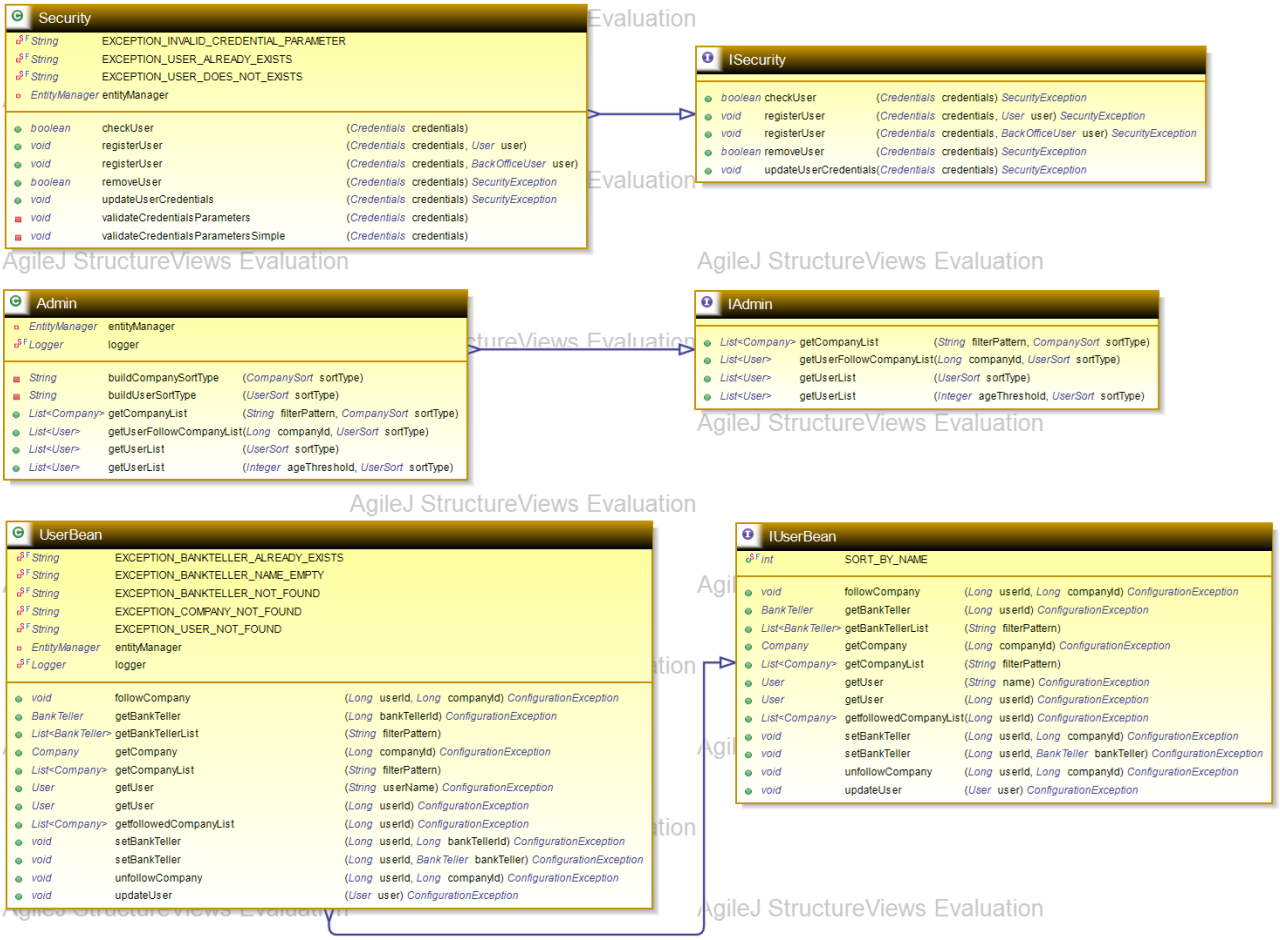

Figure 3 – EJB Static view

Design Decisions

The following sections list some major design decisions.

Use of three EJBs for user/administration operations

We decided to use one EJB for operations used by the user (*UserBean* EJB), one for administration operations (*Admin* EJB) and one for security related operations (*Security* EJB), such as, login and registration, for the following reasons:

- Admin operations are not used frequently. To avoid spending unnecessary resources, a dedicated bean is used, so that associated resources are used only when necessary
- Cleaner. Avoids "polluting" beans for specific operations with "unnecessary" interfaces
- Security. Although it is possible to control permissions at the method level of a EJB (dedicated roles for administrators and normal users), exposing administration interfaces is providing more than "need to know" information
- Separation of concerns. Operations related with access control (login, registration, etc.) are relatively common and might be shared in the future between other applications. Having a dedicated bean promotes a cleaner reuse, as other applications may require different types of operations as the ones required for this application. Having a same bean exporting user and companies operations together with access

control operations, would imply that other applications would be "exposed" to such interfaces potentially without needing them (causing additional resource waste).

- Simplicity. The user bean has all user, company and bank teller supporting operations as the amount of functionality is relatively low, not justifying a more logical separation into multiple beans. However, if the complexity would increase or we would like to have a better logical separation to promote reuse between other applications, we could use a bean for company related operations (get list, subscription control, etc.), for user related information (user list, user details, etc.) and bank teller (listing, new, etc.). However, depending on the usage, this could imply additional resource usage (threads, memory, etc.) in the EJBs.

In addition to these EJBs, we have three additional beans, but which do not export public interfaces. These are used for handling database cleanup operations (dangling bank tellers), mail digests sending and JMS topic data reception, respectively.

Stateless versus stateful EJBs

We opted for the use of stateless EJBs for the following reasons:

- With the designed interfaces, the EJB don't need to store any temporary information or be bound to a client, not justifying any state control. This allows the reuse of the beans by different applications, saving resources and promoting performance when an existing bean can be reused (avoids creation of a bean if it is still alive and available). Session control is performed automatically by the web part.
- Interfaces were designed with this in mind to avoid resource waste, which could occur with the increasing number of users if stateful beans would be used, beside the performance impacts due the need of creating the bean for every user accessing it.

Use of facelets for the Web part

- Keep presentation layer clean and promote code reuse by having wrapper classes (eai.msejdf.pagedata package) providing the interfaces that can be shared between pages to avoid code duplication of functionality.

  Note: Each HTML page has a class implementing the code required for it to interact with the EJBs. It inherits from one of the available parent classes (eai.msejdf.pagedata package) that implement common type of functionality (user, companies, bank teller and security operations), promoting reuse, as different pages may require same types of operations and can therefore reuse the parent classes. If no further functionality is required, nothing else needs to be implemented in the page class. However, if a page requires operations of a different type than the operations provided by the parent class, it can extend its implementation by referring to a class of another type for that page only (see RegisterUser.java for an example). This avoids polluting other pages with unnecessary info (with associated performance impacts).

Different sites for front and back office

- Simplifies design of web site by avoiding having to deal with concerns related with access control to different parts of the site related with different user permissions. The downside can be some code duplication, but as the operations are typically different, this duplication will be relatively low.

Database

- We decided to user lazy and eager load types depending on the database data we wanted to retrieve. Using the diagram in Annex B – DB Schema Diagram and classes in the code for context, we opted to set the load type for the Address to *eager* in the user and bank teller objects, because this data is usually accessed when referring to the user or bank teller in the application as designed. Subscribed companies and users in the user and company classes, respectively, are set to lazy to avoid fetching and passing data between tiers that is or might not be used. Instead, dedicated methods exist to retrieve such lists upon request (i.e. when required), favoring performance when the application accesses user or company data individually (typically the case).

  It shall be noted that if we would not use the *eager* load type, we would have to force the load in the business layer (less clean), as the application tier would not be able to retrieve the data without breaking the 3-tier design (persistence information would have to be known by the application tier to access the information directly).

Other decisions

We decided to use a configuration file instead of annotations for JPA objects. Besides providing a naturally cleaner code, the main focus is the separation of concerns, as the classes don't need to "know" anything related with their storage approach. In addition, this avoids changing existing/legacy classes to include annotations. To void SQL injection through user provided data, we decided to use parameters in the queries, which are set using the *setParameter()* methods.

3 Participations

The participation by each team member is listed as the packages they mainly worked on. However, each one also worked on other packages, although to less extend in order to update with functionalities or for corrections.

João Ribeiro (Time 84 hours, 20 minutes)

- EJBServer(Timer, JMS , User and Admin packages), WebFrontOffice and EJBServerInterfaces

David Cruz (Time 80 hours, 19 minutes)

- JPAData, EJBServerInterfaces and EJBServer(Timer, Security and Admin packages), WebBackOffice, EAICommon (Email classes)

Filipe Norte (Time 93 hours, 33 minutes)

- EJBServerInterfaces, EJBServer (Security and User packages), WebFrontOffice, EAICommon (web classes)

Annex A – Project deployment instructions

Prerequisites

In order to execute this software it is necessary to have JMS with JBoss AS 7 installed and MySql Server version 5 running with the configurations accordingly. The custom settings from the previous project should be set according to the previous report. Only the new ones will be described in this document, on the next section.

Deploy directory structure

The deploy directory structure is set in the following manner:

Dir	Meaning
Ear	Contains the EAR package to deploy to the JBoss 7 instance. Deploy all the components necessary to work with project (EJB, Web Frontend and Backend).
Sql	Contains the sql scripts that create the initial database schema and initial user data for running the application
Src	All the source code of whole the application

Table 2 – Deployment directory structure

Database configuration

The following steps must be taken in order to install the database required for the system:

1. Execute the script SchemaSetup.sql, that will create the original schema of the database and the user required to access the database.
2. Configure the JBoss server database connection to connect to the database. Add the following configuration to it:

```
        <datasources>
…
          <datasource jta="true" jndi-name="java:/mydb" pool-name="my_pool" enabled="true" use-java-context="true" use-ccm="true">
            <connection-url>jdbc:mysql://localhost:3306/EAIProj2</connection-url>
            <driver>mysql</driver>
            <security>
              <user-name>DBuser</user-name>
              <password>DBpass</password>
            </security>
            <statement>
              <prepared-statement-cache-size>100</prepared-statement-cache-size>
              <share-prepared-statements>true</share-prepared-statements>
            </statement>
          </datasource>
…
        </datasources>
```

3. Deploy the Ear (P2EARDeploy.ear) package into JBoss deployment directory and verify that the database schema is correctly filled.
4. Execute the script InitialData.sql, that will populate the database with the initial users required for the correct function of the system.
5. Run at least one time the webprobe application from the P1 project to populate the companies' tables on the database.

Settings and Configurations

The following section describes the settings stored in configuration files, their meaning and default values (the previous settings from the P1 project will not be handled in this report).

config.properties

This file contains the main application settings properties of the system. The following table describes the main keys of this file.

Setting	Default value	Meaning
mail.smtp.host	smtp.dei.uc.pt	The SMTP mail address to send email from
mail.smtp.user	<user>	A valid user on the SMTP serve for authentication on the server
mail.smtp.pass	<pass>	The password of the valid user to authenticate him on the SMTP server
mail.emailfrom.address	davidpc@dei.uc.pt	The email that will used in the from field for the digest emails sent for the users

Table 3 – Configuration settings

How to execute

The system is essentially web based, with enterprise beans running on the middle tier of the system. There are two different web applications depending on the purposes of the system. The frontend applications for the common usage can be open in a web browser with the following settings:

Setting	Value
Address	http://localhost:8080/WebFrontOffice/
Sample user	user
Sample password	user

Table 4 – Frontend access settings

For the back office application the following settings should be used:

Setting	Value
Address	http://localhost:8080/WebBackOffice/
Sample user	admin
Sample password	admin

Table 5 – Backend access settings

The legacy applications from the P1 project should also be run and setup in order to be used correctly with this system. As described in the previous report the tools can be run through the following commands:

File	Operation
WebProbe.bat	Executes the parser plugin
RRDDaemon.bat	Initiates the RRD Daemon

Table 6 – Legacy programs execution

NOTE : For correct display of the graphs on the companies pages shown on the frontend web site, the rrd tool should be set to output to output the images to a directory DataOut/rrd/ inside the welcome-content dir of the JBoss server. To do this just set the **app.base.outputdir** setting to the path where the welcome-content directory is plus the DataOut dir, something like **c:\servers\ jboss-as-7.1.1.Final\welcome-content\DataOut**

Annex B – DB Schema Diagram

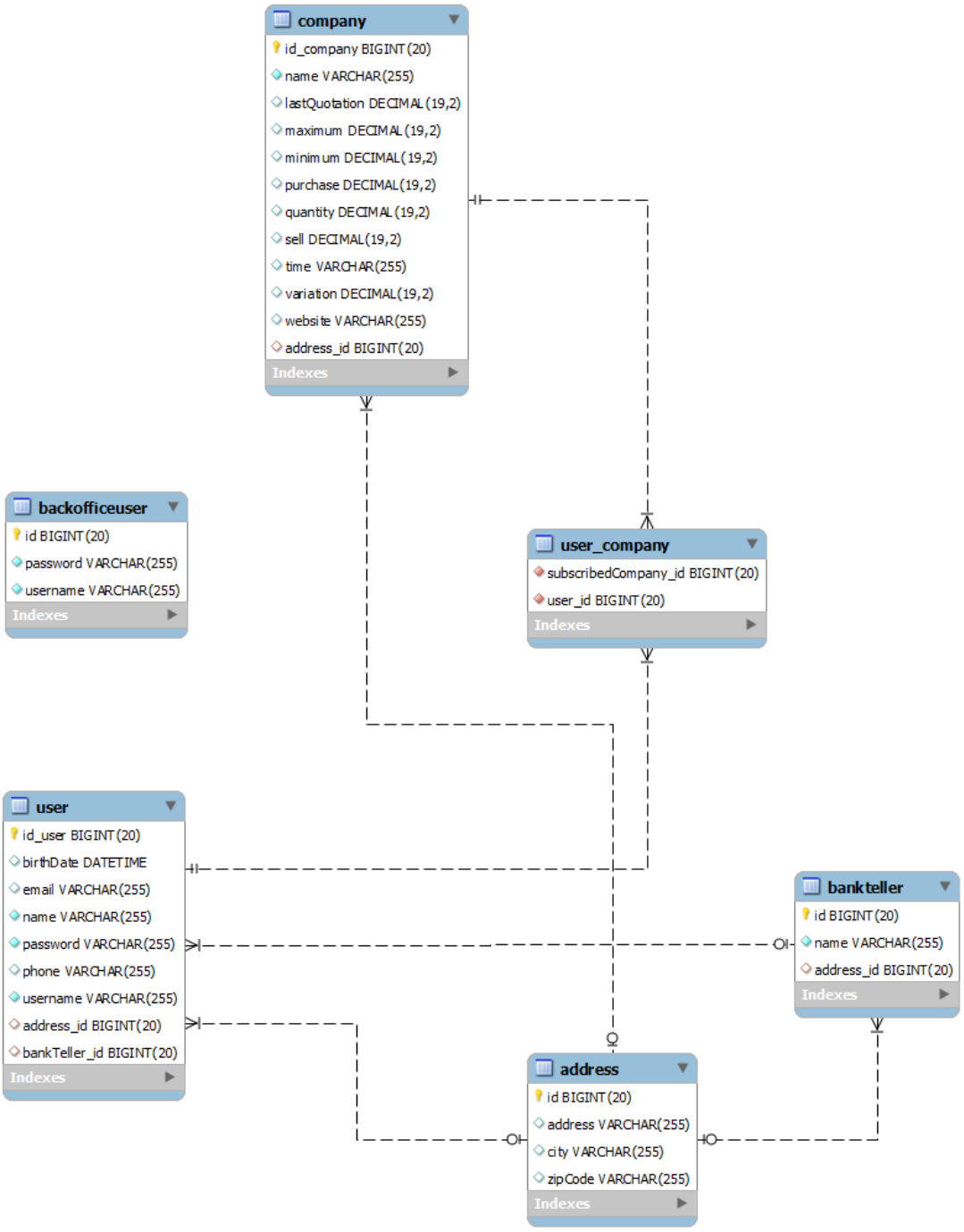

Figure 4 – Database schema of the system