< Bank Management>
Analysis and Design Document

Student: Sechel Raluca-Rodica
Group: 30233

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

This is an application for the front desk employees of a bank. The application has two types of user: a basic user which is the front desk employee, and an administrator user. All kinds of users need to provide a username and a password in order to use the application. The basic user can perform several operations such as add/update/view client information, create/update/delete/view client account, transfer money between accounts, process bills. The client information consists of name, identity card number, personal numerical code, address, and the account information can hold identification number, type, amount of money, date of creation. The administrator user can create/read/update/delete employees' information and generate reports for a particular period containing the activities performed by an employee.

1.2 Functional Requirements

In order to use the application, first you have to authenticate. You can choose one of the two possible users: basic user or administrator user. For the authentication you need a username and a password. After entering the username and password the application will validate the compatibility between them. If they match, you can perform the provided operations, if not, the application will show an error message: „Username and password don't match!". For username and password enter Strings.

If you want to authenticate as basic user with the right username and password, you can perform client and account operation. For each operation (view, add, update, delete) the application first searches the datas entered. If they don't exist, the application shows an error message. If the input is valid, CRUD operations can be performed on clients and accounts. If the task is to transfer money, amount is debited if sufficient balance is available, otherwise an error message is shown. Likewise, if the user chooses to process a bill, the availale balance is validated before. For ids and amounts enter Integer and for names enter String.

If you want to authenticate as administrator user with the right username and password, you can perform some operations on employees. For CRUD operations, first the application validates the datas and shows an error message for invalid data. An error message appears also when the user leaves some fields uncomplited. If not the case, the admin can add/view/update/delete employees and see employee report. For ids and amounts enter Integer and for names enter String.

1.3 Non-functional Requirements

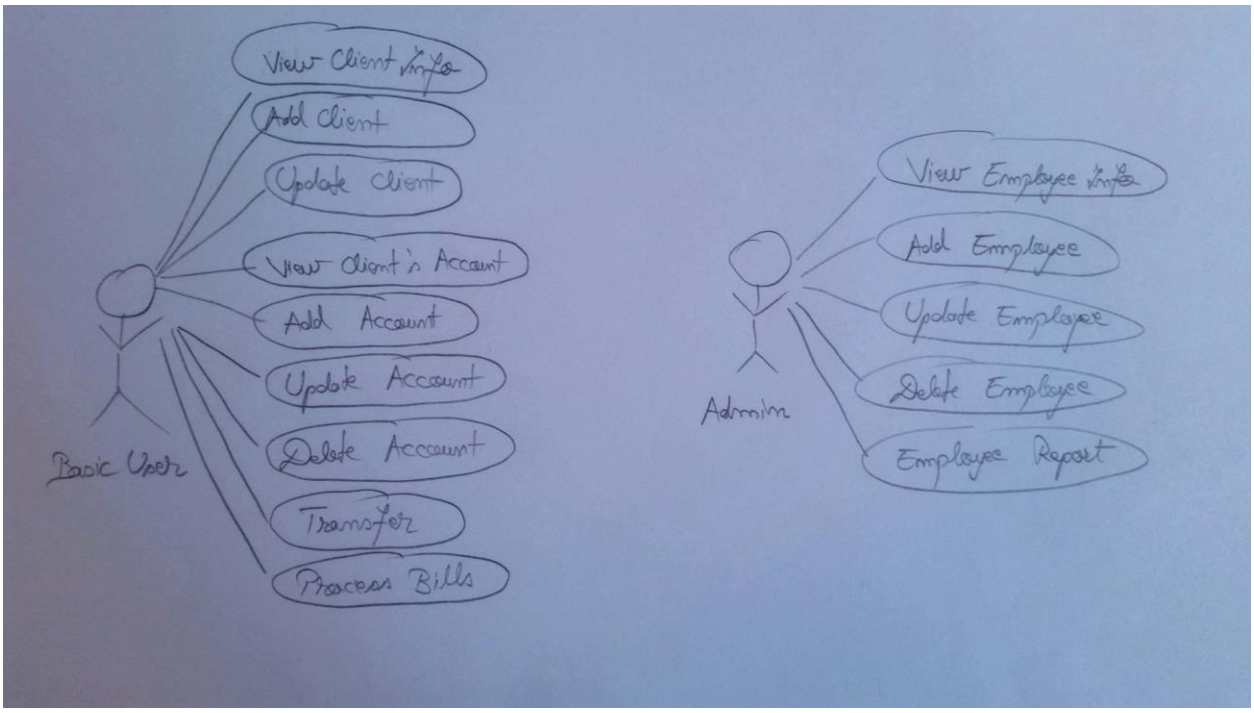
- Availability

My system is available because it is capable to provide it's intended services.

- Reusability

The application has some assets that can be used in some form within the software product development process. For example we can use classes that take information from database and use them in some different ways.

2. Use-Case Model



Use case: Transfer money
Level: Sub-function
Primary actor: Employee
Main success scenario: The amount to be transferred exists in the source account, and the amount in the destination account increases with the specified amount.
Extensions: It is possible that the specified amount to be greater than the amount in the source code, which leads to a failure.

3. System Architectural Design

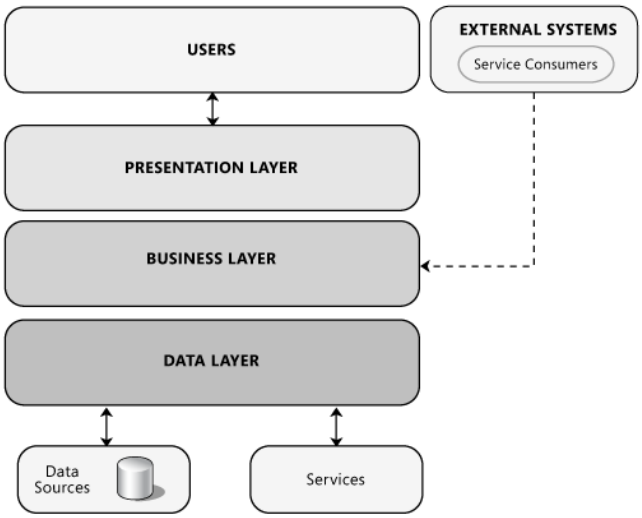
3.1 Architectural Pattern Description

For my application I used Layered Architectural Pattern. As the name says, the architecture is structured in layers. an application can consist of a number of basic layers. The common three-layer design consists of the following layers :

- **Presentation layer**. This layer contains the user oriented functionality responsible for managing user interaction with the system, and generally consists of components that provide a common bridge into the core business logic encapsulated in the business layer.

- **Business layer**. This layer implements the core functionality of the system, and encapsulates the relevant business logic. It generally consists of components, some of which may expose service interfaces that other callers can use
- **Data layer**. This layer provides access to data hosted within the boundaries of the system, and data exposed by other networked systems; perhaps accessed through services. The data layer exposes generic interfaces that the components in the business layer can consume.

3.2 Diagrams



The picture above describes the system's conceptual architecture. To implement this architecture I used two patterns: Table Module for Business layer and Table Data Gateway for Data layer.

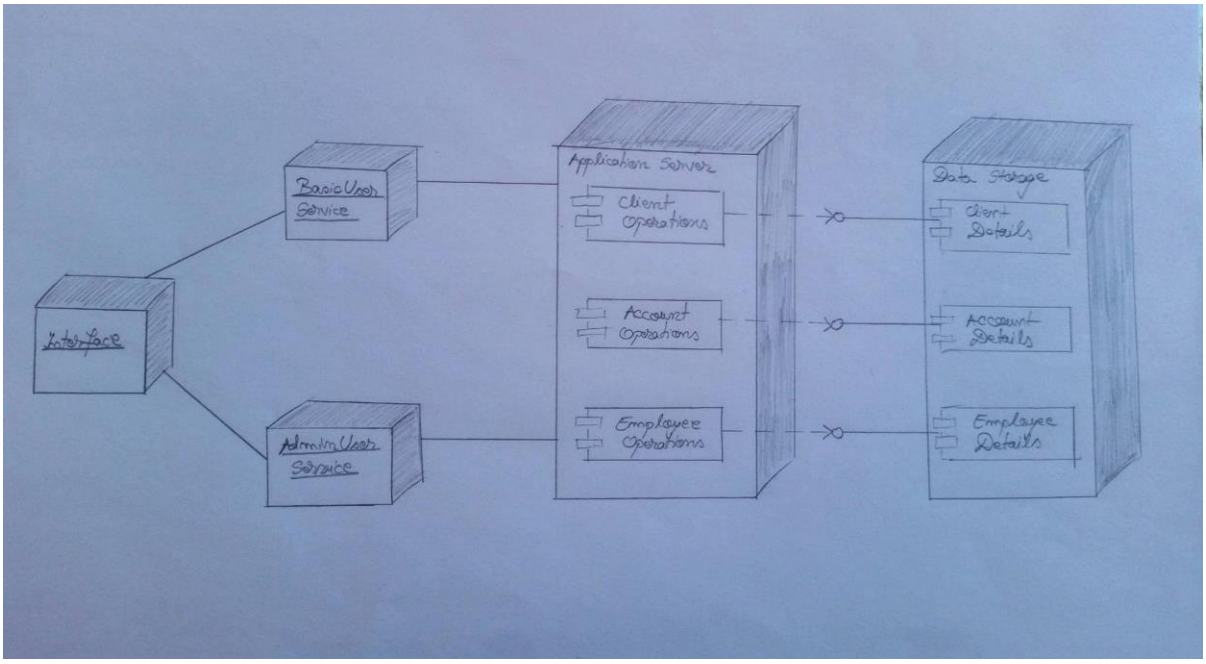
Table Module and Table Data Gateway are both table level patterns, but with a very fundamental difference.

A Table Module is a Domain Logic Pattern in the sense it can contain the business logic related to a particular table. A Table Module organizes domain logic with one class per table in the database, and a single instance of a class contains the various procedures that will act on the data. The strength of Table Module is that it allows you to package the data and behavior together and at the same time play to the strengths of a relational database. On the surface Table Module looks much like a regular object. The key difference is that it has no notion of an identity for the objects it's working with. I chose to use Table Module because I have backing data structure that's table oriented. The tabular data is normally the result of a SQL call and is held in a Record Set that mimics a SQL table. The Table Module gives you an explicit method-based interface that acts on that data. Grouping the behavior with the table gives you many of the benefits of encapsulation in that the behavior is close to the data it will work on.

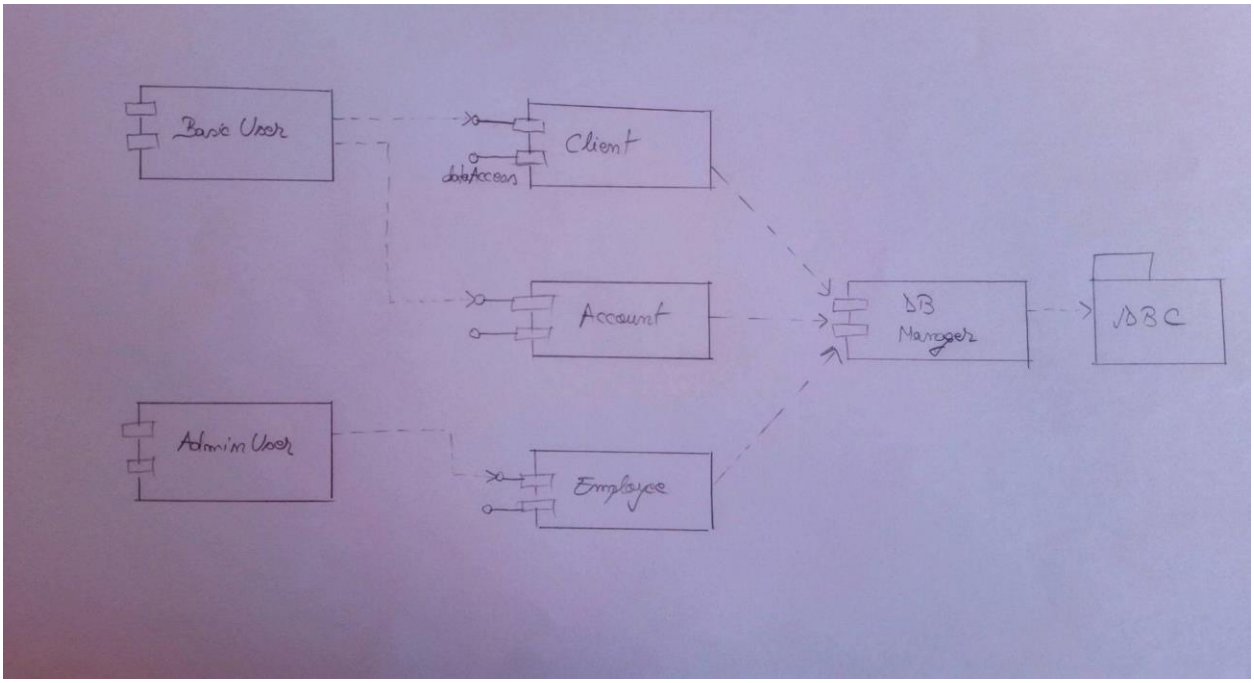
A Table Data Gateway is supposed to handle Database interface only and is not supposed to contain any Business Logic. We can look at Table Data Gateway as an object that acts as a Gateway to a database table and has one instance handles all the rows in the table.The

data access part is just an other level of abstraction which is the specific part that talks to the persistent level which can be easily replaced if you ever need to change your persistent level in your application. This way, the only part you have to rewrite is the data access level and the rest of the application is still working.

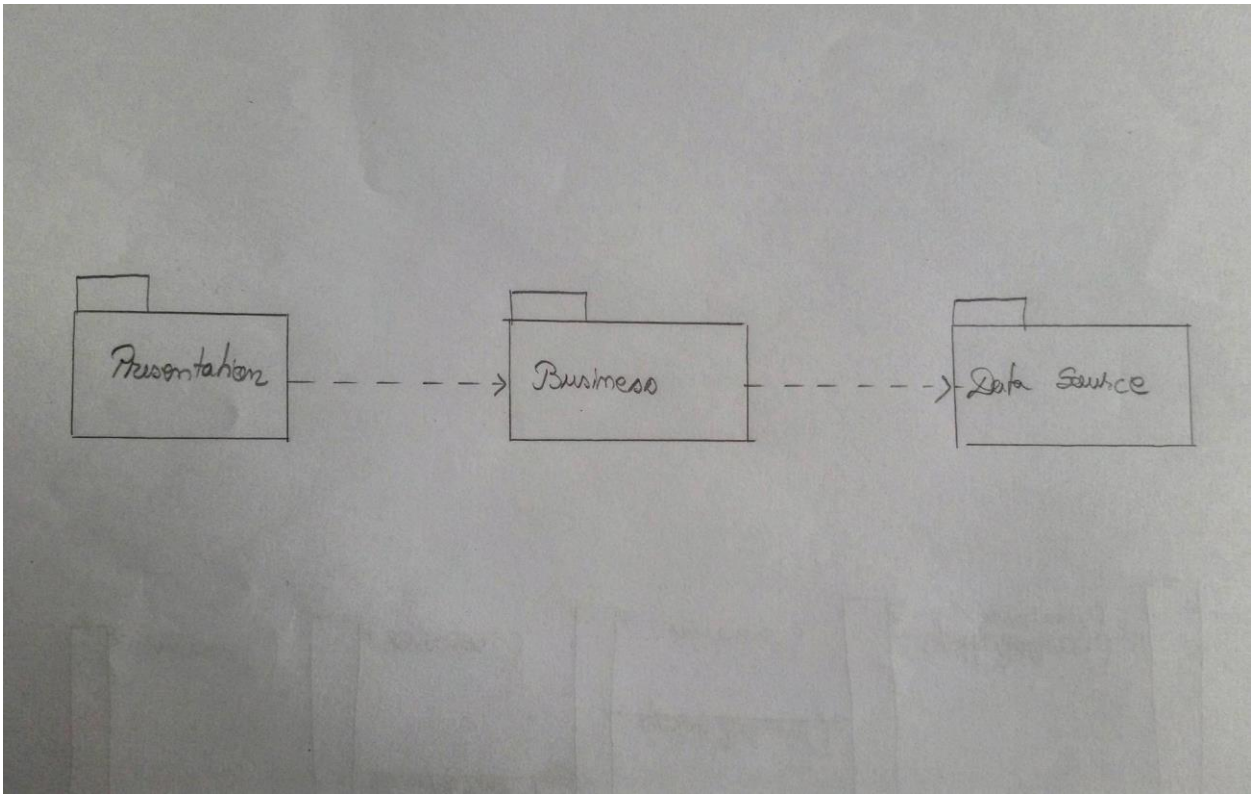
Here is the deployment diagram for my application:



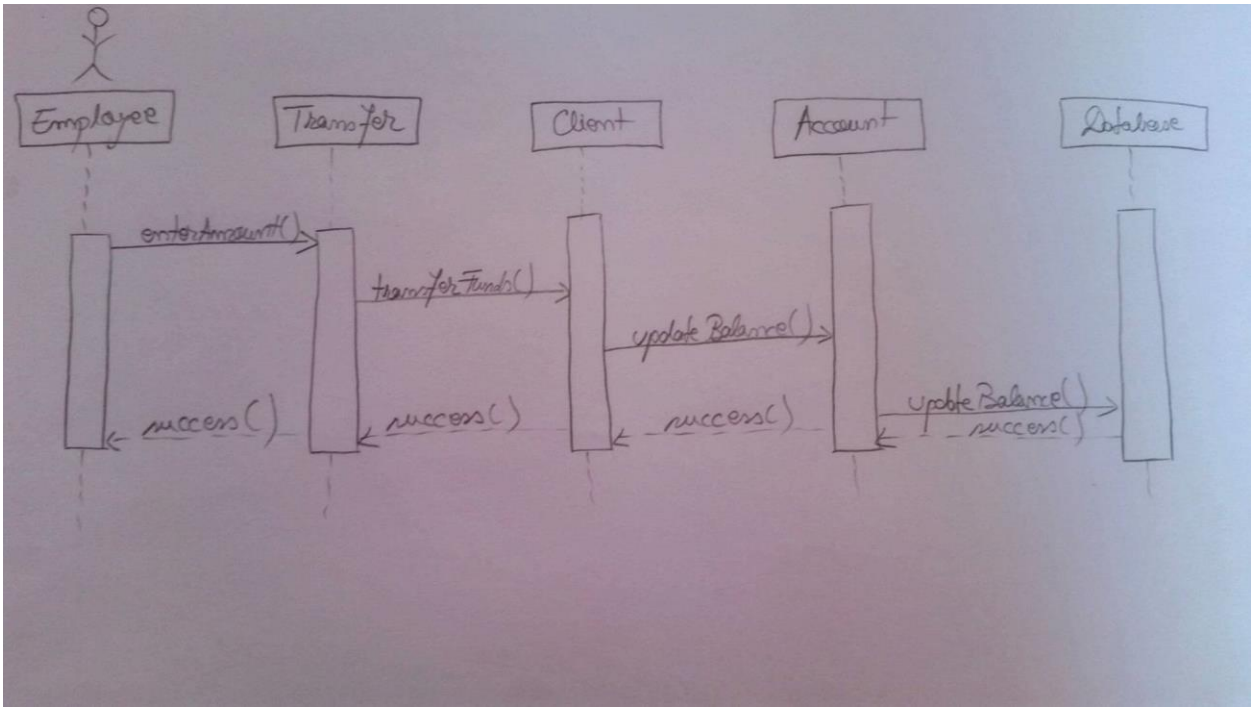
Here is the component diagram for my application:



Here is my package diagram:



4. UML Sequence Diagrams



5. Class Design

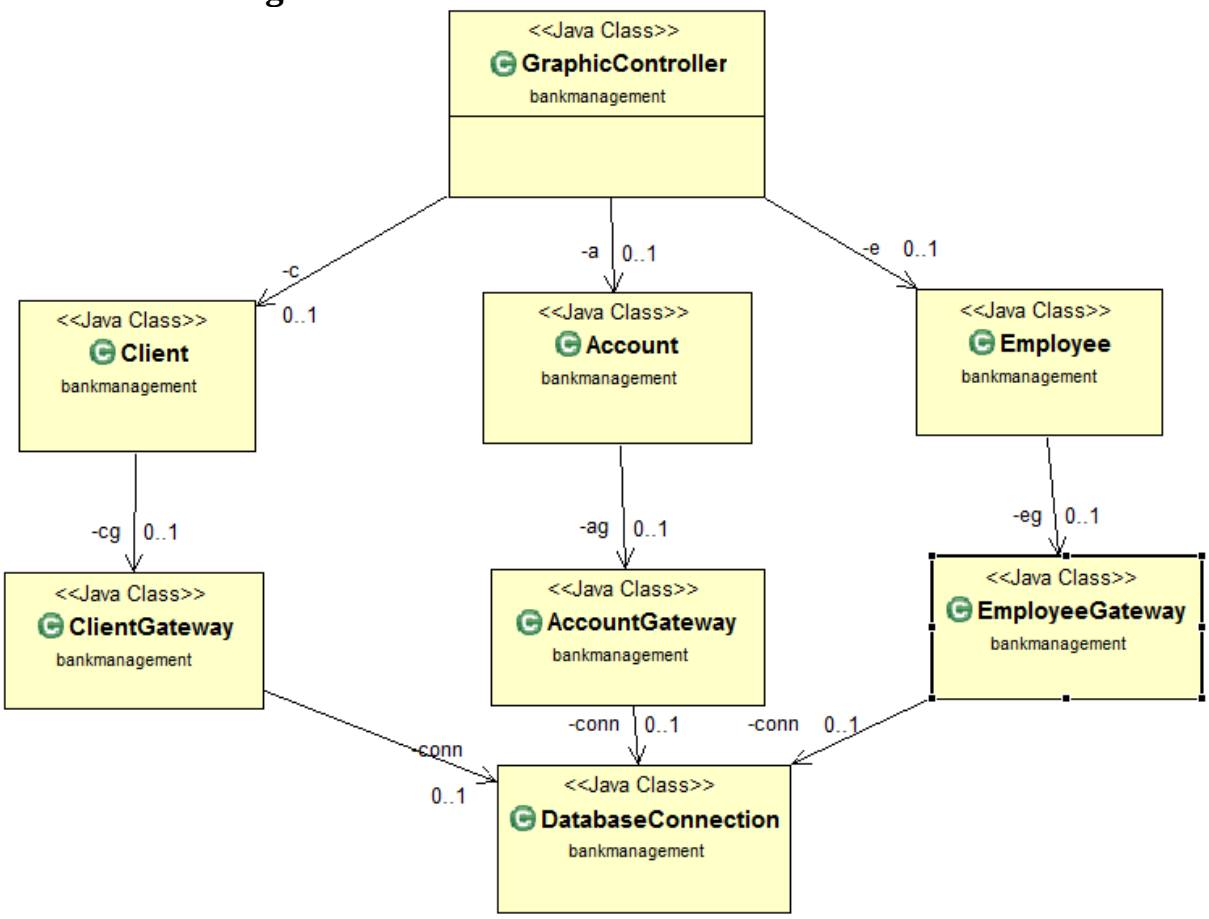
5.1 Design Patterns Description

As patterns used in my application I already talked about Table Module and Table Data Gateway.

Table Module is a Domain Logic Pattern and it describes the functional algorithms or business logic that handle the information exchange between a database and a user interfaces. A well organized Domain Logic components are easy to maintain and scale. A Table Module organizes domain logic with one class per table in the database and a single instance of a class contains the various procedures that will act on the data. Table Module will have one object to handle all orders.

Table Data Gateway is a design pattern in which an object acts as a gateway to a database table. The idea is to separate the responsibility of fetching items from a database from the actual usages of those objects. Users of the gateway are then insulated from changes to the way objects are stored in the database.

5.2 UML Class Diagram



6. Data Model

We need to know how the logical structure of our database is modeled, so we need a data model. Data models define how data is connected to each other and how they are processed and stored inside the system. I will describe two data models: Entity-Relationship Model and Relational Model.

Entity-Relatioship Model consists of the notion of real word entities and relationships between them. This model is based on entities and their attributes, relationships between entities, where an entitie is a real-world entity having properties called attributes and a relationship is the logical association among entities.

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. In this model, data is stored in tables called relations and relations can be normalized. In normalized relations, values saved are atomic values. On the other hand, each row in a relation contains a unique value and each column in a relation contains values from a same domain.

The data model used in my application is Relational Model.

7. System Testing

For each CRUD operation I checked first if the given data exists in database. For example, if you want to view a client's information, the application verifies if the name of that client is in the database and then, if it exists, calls the method that takes the information from database. On the other hand, you can't create the same client twice, because the application checks if the client exists in database before creating the client. Of course, if you want to update a client's information, that client has to be in the database and this is also verified in the application. You can't delete something that doesn't exist, so the system won' let you. Every time when you try to access something that doesn't exist in database, you get an error message. Same things happen for accounts and employees.

8. Bibliography

- Software Architecture Patterns – Mark Richards
- Patterns of Enterprise Application Architecture – Martin Fowler
- https://www.oreilly.com/ideas/software-architecture-patterns/page/2/layered-architecture