<Electronic Shop for Festival Tickets>

Analysis and Design Document

Student:Ispas Ovidiu Mihai

**Group:30433**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <01/04/18> | <1.0> | <First version> | <Ispas Ovidiu Mihai> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

# Design and implement a web application which could be used as an application for selling tickets for festivals.

The application should have two types of users : the regular customer and the administrator.

The regular customer can perform the following operations :

* View festivals, with all the specifications
* View all types of tickets available for one festival
* Purchase tickets for the festivals he/she wants
* Search for the wanted festival either by typing the name or by using some filters

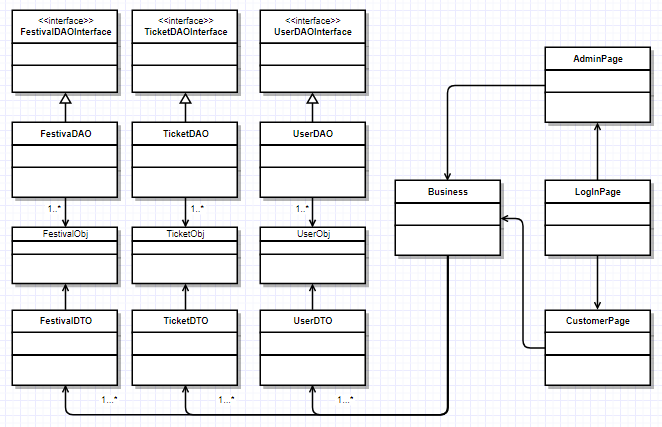
The administrator can perform the following operations :

* CRUD on festivals
* CRUD on tickets

# Elaboration – Iteration 1.1

# Domain Model

The domain model is a [conceptual model](https://en.wikipedia.org/wiki/Conceptual_model_(computer_science)) of the domain that incorporates both behavior and data. In [ontology engineering](https://en.wikipedia.org/wiki/Ontology_engineering), a domain model is a [formal representation](https://en.wikipedia.org/wiki/Knowledge_representation_and_reasoning) of a [knowledge domain](https://en.wikipedia.org/wiki/Knowledge_domain) with concepts, roles, data types, individuals, and rules, typically grounded in a [description logic](https://en.wikipedia.org/wiki/Description_logic).



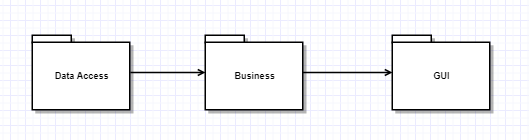
# Architectural Design

## Conceptual Architecture

Since it is a web application, one of the architectural design pattern used for the application would be the Client-Server architecture, client being the regular user and server being the application’s server.

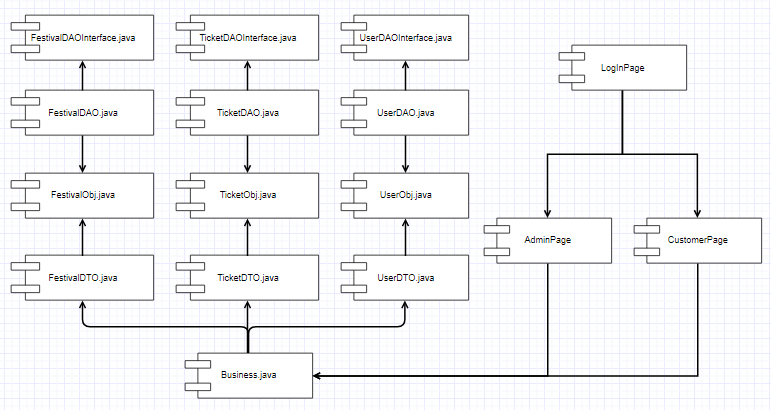
Another architectural pattern which the application will use will be the Layers pattern in order to have a better separation of the data access from the rest of the application and so the code to be more easy to change if it is the case.

## Package Design

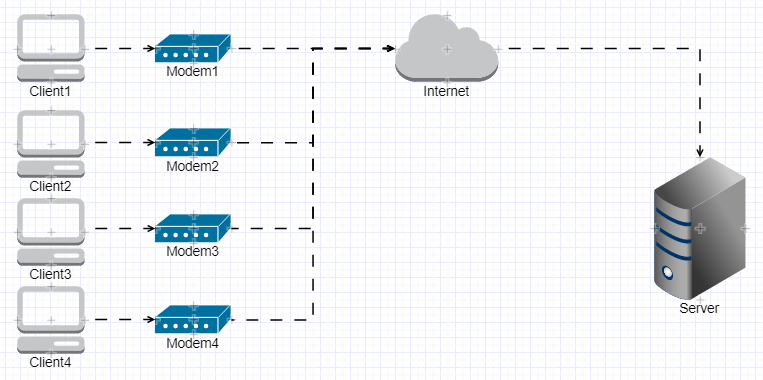


## Component and Deployment Diagrams

Component diagram :



Deployment diagram :



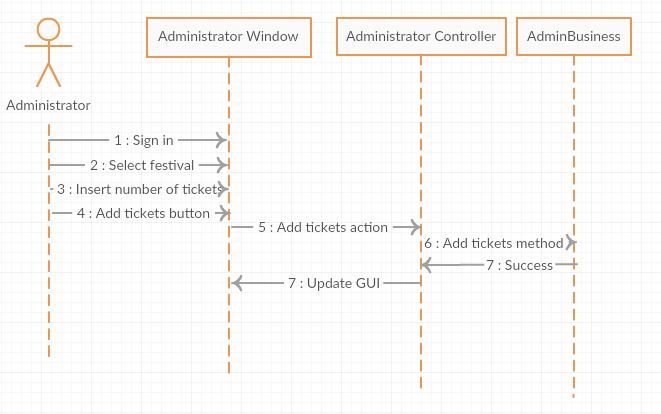
# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

Sequence diagram :

Scenario : add tickets

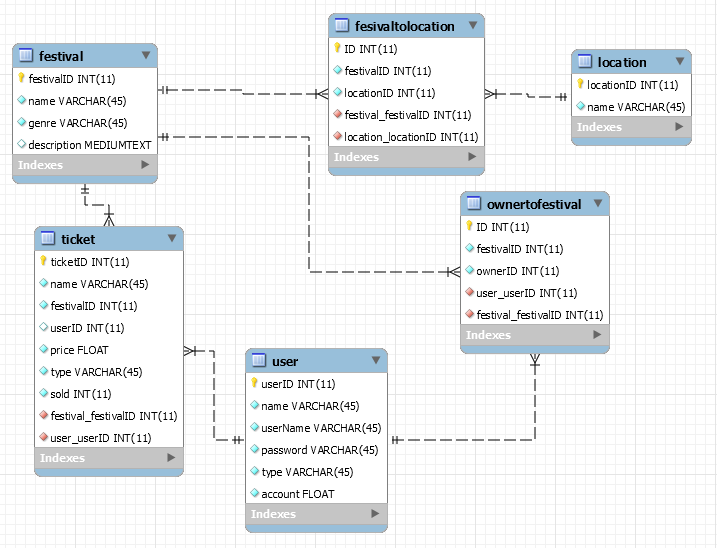


## Class Design

# Data Model

The Data Model I will use for organizing the data for this project is the Relational Model , which is based on storing all the necessary data in 2 dimensional tables which are linked between each other by using an additional field which contains the ID from the table you want to make the one-to-many relationship.

The purpose of the relational model is to provide a [declarative](https://en.wikipedia.org/wiki/Declarative_programming) method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.



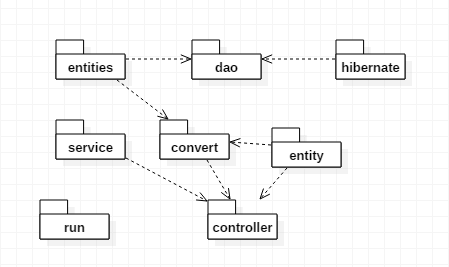
# Unit Testing

For the testing section, there are many different test case scenarios.

One of them would be adding a new owner account to the data base.

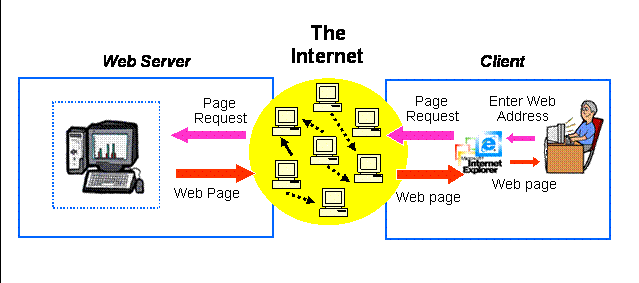
# Elaboration – Iteration 2

# Architectural Design Refinement



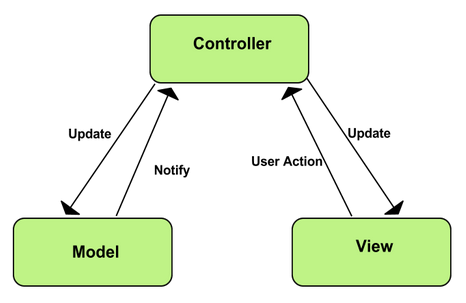
* **Client-Server Architecture**

Since it is a web application, the ESFT is a Client-Server one.



* **Model View Controller (MVC)**

Spring is using a MVC architecture to make the relationships between the backend and the view.



# Design Model Refinement

Spring framework is a complex framework which implements various design patterns in it’s. Some of them which were implemented in the Spring’s functionalities that I used were :

* Dependency Injection (IOC)
* Factory Design Pattern
* Singleton Design Pattern
* Proxy Design Pattern

# Construction and Transition

# System Testing

Testing was made gradually, as the application evolved. The most important test case would be for the creating festival test case scenarios, because it involved many-to-many relationships which had to work properly.

Another important test case scenario was the buying ticket scenario, because it was important that the account of the client decreases with the price of the according ticket.

# Future improvements

The application, as it is now, is quite basic. So there are plenty of future improvements that could be done :

* Better looking graphical user interface
* Application security, so that the application can not be accessed without logging in

# Bibliography