<Plane Tickets>

Analysis and Design Document

Student: Prata Mirela

**Group: 30233**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 25/04/2019 | 1.0 | Iteration 1.1 | Prata Mirela |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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

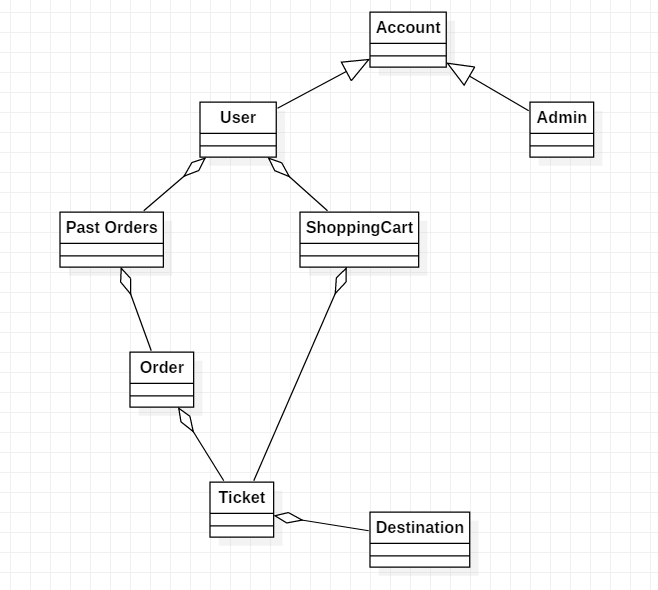
VI. Bibliography 5

# Project Specification

# The application is an online platform that helps the users in purchasing plane tickets by offering the list of vacation offers. The main purpose of the application is to give the user the most affordable and good deals to reduce as much as possible the search time. The system allows the user to see the description of each vacation offer and to add plane tickets to the chart and buy them. The system is developed with a user-friendly GUI, users have to first login into the platform to view the offers and destination.

# Elaboration – Iteration 1.1

# Domain Model



# Architectural Design

## Conceptual Architecture

**Model View Controller Architecture**

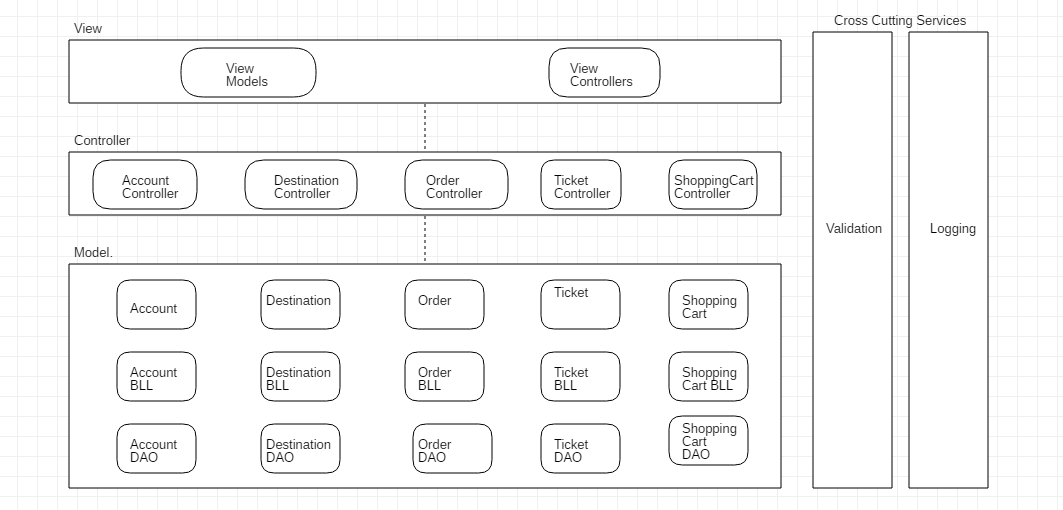
MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

**Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

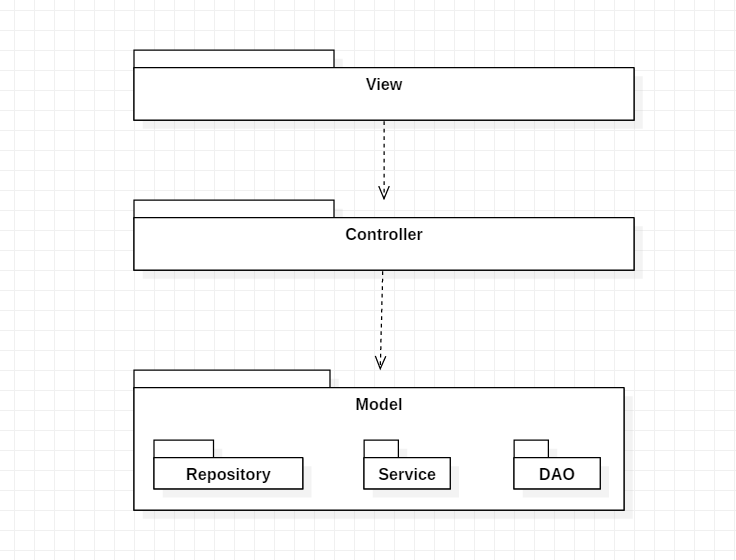
**View** - View represents the visualization of the data that model contains.

**Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

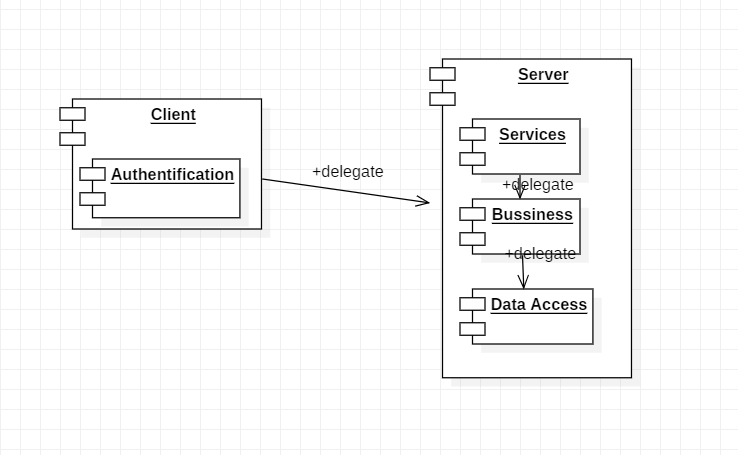
The application also has a mini layered architectural pattern. One of the powerful features of the layered architecture pattern is the separation of concerns among components. Components within a specific layer deal only with logic that pertains to that layer. Business layer**.** This layer implements 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.



## Package Design



## Component and Deployment Diagrams



# 

# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

*[Create the interaction diagrams (1 sequence, 1 communication diagrams) for 2 relevant scenarios]*

## Class Design

*[Create the UML class diagram; apply GoF patterns and motivate your choice]*

# Data Model

*[Create the data model for the system.]*

# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

# Design Model Refinement

## *[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]*

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

# Future improvements

*[Present future improvements for the system]*

# Bibliography