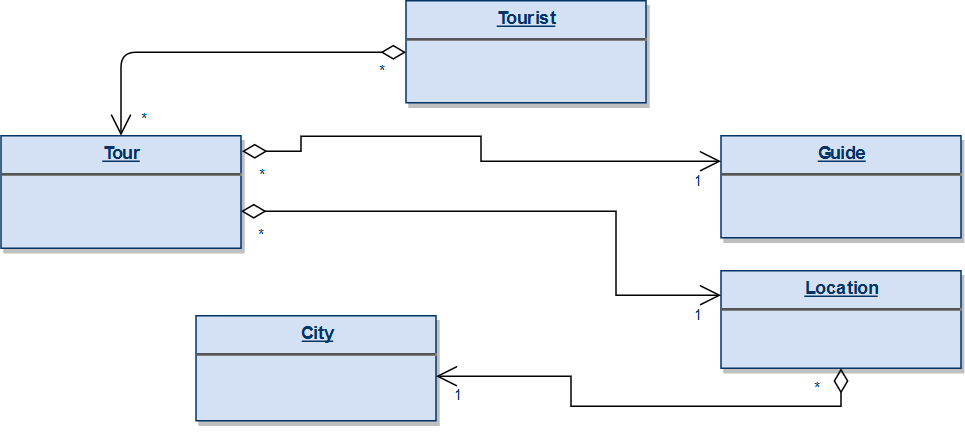
Tours & Travels

Analysis and Design Document

# Elaboration – Iteration 1.1

# Domain Model

**

My app will have as models the above entities. The relationships between these entities are the following:

* The tourist will be able to book multiple tours, while the tour needs to have more than one tourist so it can have multiple tourists as well. We are in the situation of a many-to-many relationship.
* The tour can only have one guide and among its attributes it also contains the guide’s id, but the guide can organize many tours, not just one, so we find ourselves in the situation of a one-to-many relationship.
* The tour can take place in a single location, while a location can have multiple tours being held.
* The location can have only one city, but the city can have more locations.

# Architectural Design

## Conceptual Architecture

Functional requirements

R1. The system is a travel planning app. The system can find the following optimal travel tours:

* Most economical
* Cheapest
* Most pleasant
  + Most sightseeing spots
  + Along rivers and through parks

R2. The places, the data regarding booking and personal data of the tourists and of the guides are all stored in a database

R3. The admin and the developer of the app can add/remove tours

R4. The user can search for cities, locations and through a wide range of tours for each chosen location

R5. The system needs to interact with external services (for example, if a specific tour includes a visit to the museum, it should buy tickets for that museum)

R6. The system needs to provide user management:

* Register new users
* Log-in/Log-out
* Store user preferences
* Store credentials for purchase

Architectural style

I am going to use the Layered architectural style given the following reasons:

* It enforces a better separation between the responsibilities the app should carry out
* Each layer of the layered architecture pattern has a specific responsibility within the application:
* A presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request
* It is easier to understand and to maintain for the developer
* It is easier to write because it is cleaner and better organized
* It is easier to extend because the process of adding new features or changing the current features is simplified

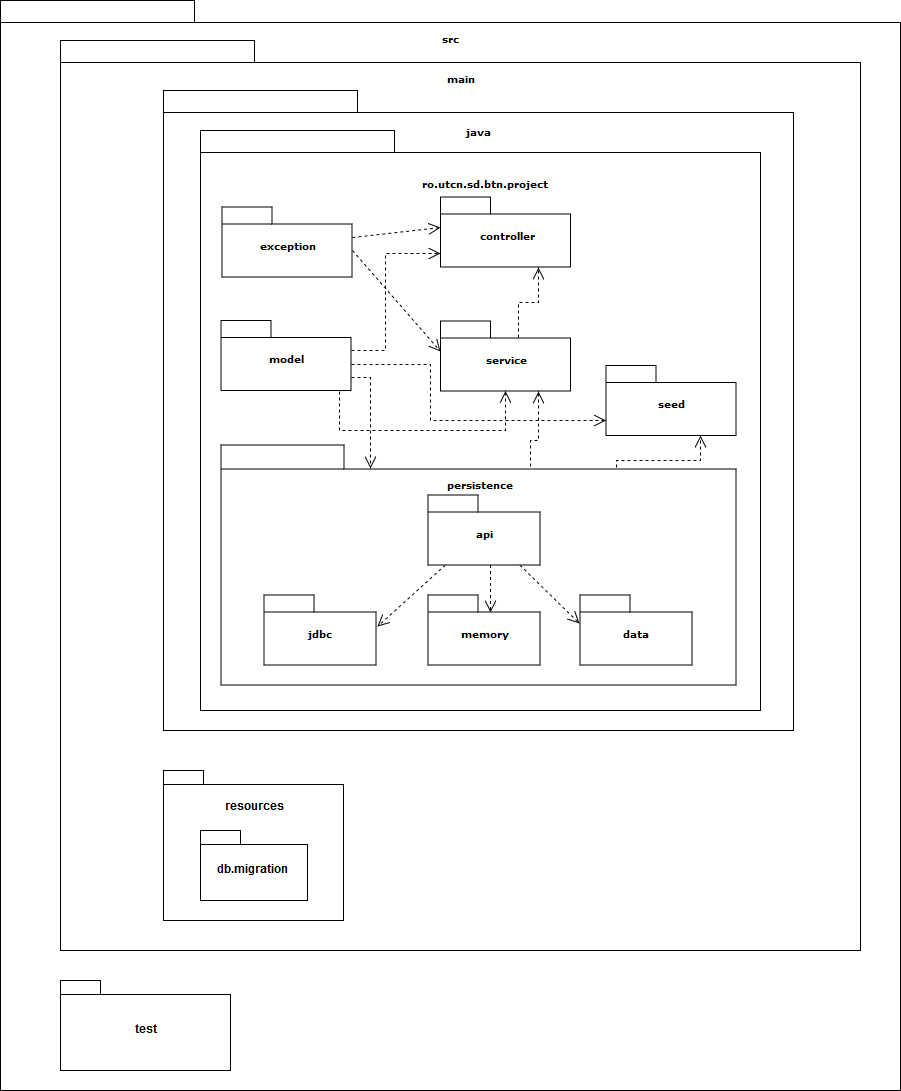
Architectural pattern

I am going to use the Model-View-Controller architectural pattern given the following reasons:

* There is a better separation between the different aspects of the app(input logic, business logic, and UI logic) :
  + The UI logic belongs in the view
  + Input logic belongs in the controller
  + Business logic belongs in the model
* This separation enables the possibility to focus on one aspect of the implementation at a time

## Package Design

One package depends on another if changes in the other could possibly force changes in the first.

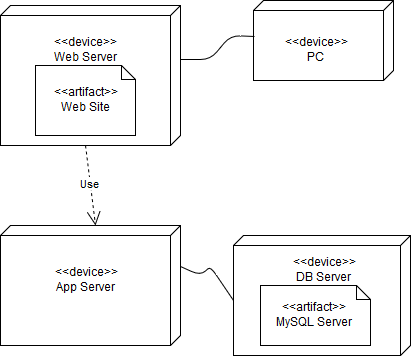
**

## Component and Deployment Diagrams

Component Diagram

# 

Deployment Diagram



# Elaboration – Iteration 1.2

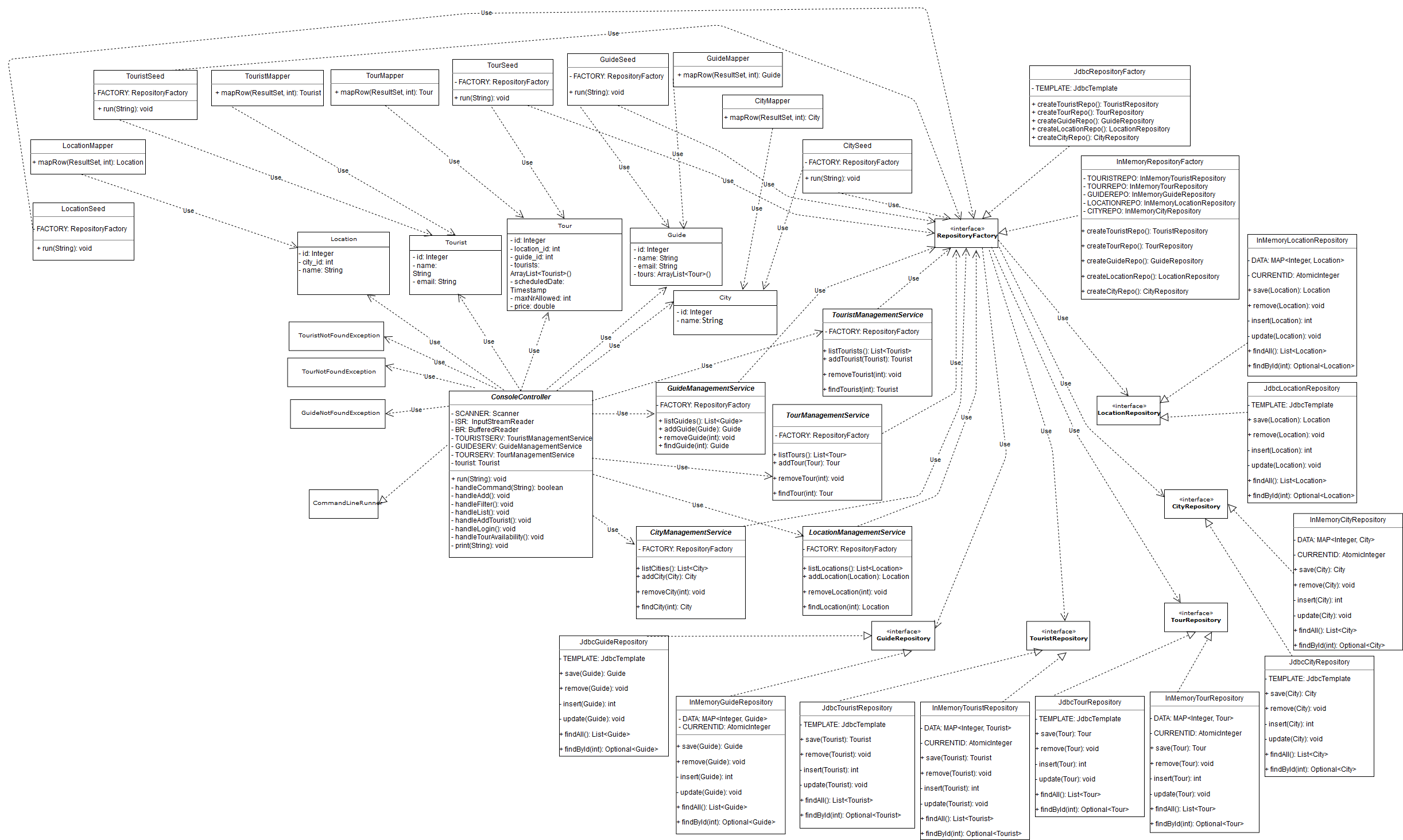
# Design Model

## 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.]*

# Test Strategy

*[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]*