Online Bus Ticket System

Object Design Document

V:1.0

28.12.2017

Didar Damla Turanlı

Hatice Dilara Turanlı

Ayşenur Bayram

Anıl Aydıner

Prepared for

SE301 Software Engineering



Table of Contents

[1. Introduction 1](#_Toc502259659)

[1.1. Object Design Trade-offs 1](#_Toc502259660)

[1.2. Interface Documentation Guidelines 1](#_Toc502259661)

[1.3. Definitions, Acronyms, and Abbreviations 2](#_Toc502259662)

[1.4. References 2](#_Toc502259663)

[2. Packages 2](#_Toc502259664)

[3. Class Interfaces 3](#_Toc502259665)

OBJECT DESIGN DOCUMENT

# Introduction

## Object Design Trade-offs

Firstly, our project name is Online Bus Ticket System (OBTS).This website is related to buying tickets and making ticket bookings. Users who use this system can purchase tickets at the time they want to go or they can booking tickets in advance. The purpose of this document is to describe design and architecture of the OBTS site. The codes used when designing this system and system design are products of our group work and they are unique to us.The OBTS system should be a simple and understandable system.It is also one of the most important features of this system if it is safe.In this system,we use database to store information and this database also increases the system’s security.System keep some information like address, password, SSN number, bus information,vacation information… etc. securely.This project includes some trade-off.As we mentioned earlier,the OBTS system uses a database,and it increase performance.Therefore database is very important for OBTS system. The user who uses the OBTS system ,waits get fast receive feedback from the system in order to search vacation, buy tickets, make reservations.Database provide fast response time. PhpMYADMİN can handle almost any amount of data, up to as much as 50 million rows or more PhpMYADMİN provide high scalability.

## Interface Documentation Guidelines

In this system, object design principle had applied. There are two groups of graphical design in this project. First, graphical design for User –Who is registered user or Visitor- side, and other for Admin side. These two designs are designed differently.

Users easily understand the site structure, and they can take easily actions on the site and with this they can easily make them reservations.In registered user side ,registered user can see reservations and ticket that done before, they can make a new reservation. By the way, registered user can buy ticket or delete tickets . Visitor side, Visitor can buy ticket but when s/he try to reserve ticket s/he have to be registered into the OBTS.

The admin panel, is more complex according to user-side. However, it also easy to useable for admin. Menus are clearly identified on the panel.Admin can check all users’ information. Also admin can delete bus and voyage anytime.

Coding standards are important in any development project, but they are particularly important when many developers are working on the same project. Coding standards help ensure that the code is high quality, has fewer bugs, and can be easily maintained. Function names must always start with a lowercase letter. Any file that contains PHP code should end with the extension ".php", with the notable exception of view scripts.

## Definitions, Acronyms, and Abbreviations

• OBTS:A name of the online bus ticket system.

• ODD : Object Design Document

• Login: to get access to an operating system or application, usually in a remote computer

• User: a person who use a computer (Visitor or RegisteredUser)

• Visitor:A person who can see ticket,buy ticket,register.

• RegisteredUser:A person who cen can login ,see ticket,buy ticket,booking, etc…

• Admin: A person who do some operations on database.

## References

* + 1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML,Patterns, and Java*, Prentice Hall, 3rd ed.
    2. Se301 course ODD template

# Packages

Large scale projects do not coding by one person, everyone who involves the coding side of project can see very easily, therefore packages are very significant . Before packaging we separate our system to subsystems in our SDD. Also subsystems must be relative with each other, when coding is finish, subsystems attach the code packages. Before the coding process first of all we create empty package, in this package we have necessary statements which is including in our use case. Inside our package user information’s stored. After that our package connect to the DB then retrieval data from DB. Our second package is about creating voyage. Another .php page these information’s taken and stored in DB. In this package we can edit anything payment, booking, see ticket or booking . Finally last package which provide user needs includes Log-in, Sign-up, paying ticket, booking, see ticket ,see booking and main page.

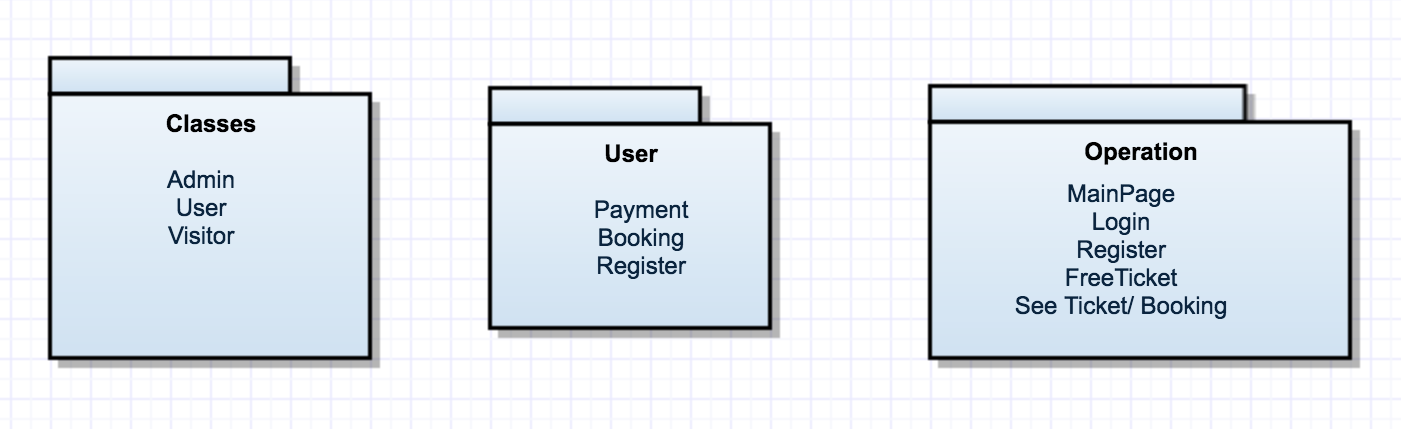


Figure 2.1- OBTS packages

# Class Interfaces

User and Visitor classes are extends from user class. Both of their information are stored in DB and user must be log-in for booking, payment ,ticket and using free ticket and update their information. All classes’ attributes are private. Some necessary part of methods have get and set methods.

**Database**

\*Invariants:  
\* These information cannot be null  
\*@invariant db\_name!=null   
\*@invariant server\_password!=null   
\*  @invariant server\_username!=null   
\*  @invariant host!=null

\*   
\*  PostConditions:   
\*  start(host,server\_username,server\_password,db\_name)  
\*  @post isConnected==true   
\*PreConditions:  
\* start(host,server\_username,server\_password,db\_name)  
\* @pre getServerUsername()!=null && getHost!=null && getServerPassword!=null && getServerUsername!=null && getDbName!=null

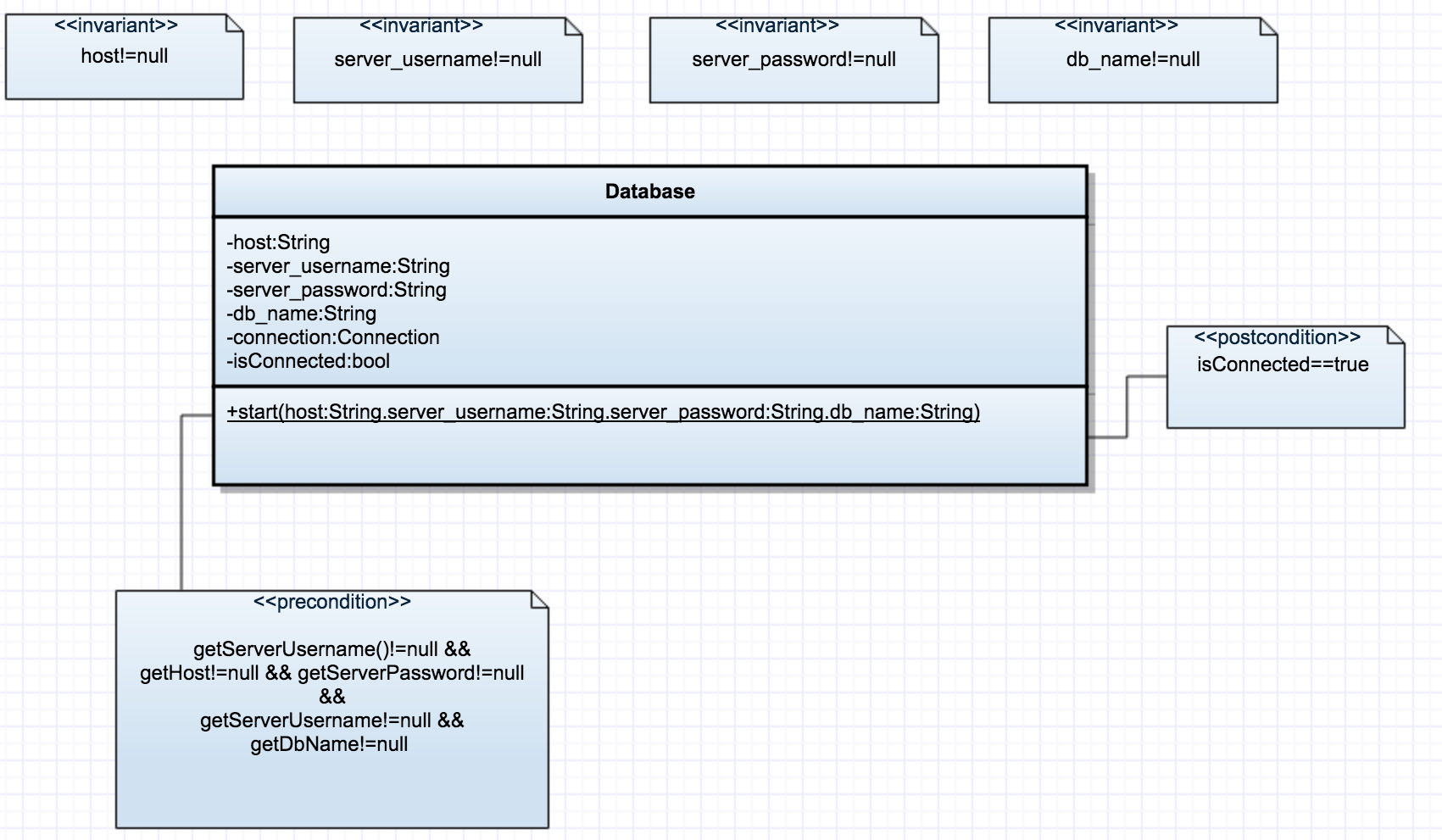
\*/

Figure 3.1- Database UML

**Voyage**

\*Invariants:  
\*@invariant voyage\_id!=null  
\*@invariant bus\_id!=null   
\*@invariant city\_id!=null   
\*@invariant startingLocation!=null   
\*@invariant destionalLocation!=null  
\*@invariant date!=null  
\*@invariant time!=null  
\*   
\*  Preconditions:   
\*  admin\_login(ssn, password)   
\*  @pre getpassword()!=null   
\*  PostConditions:   
\*  isLogin(ssn, password)   
\* @post isLogin == true

\*

\*

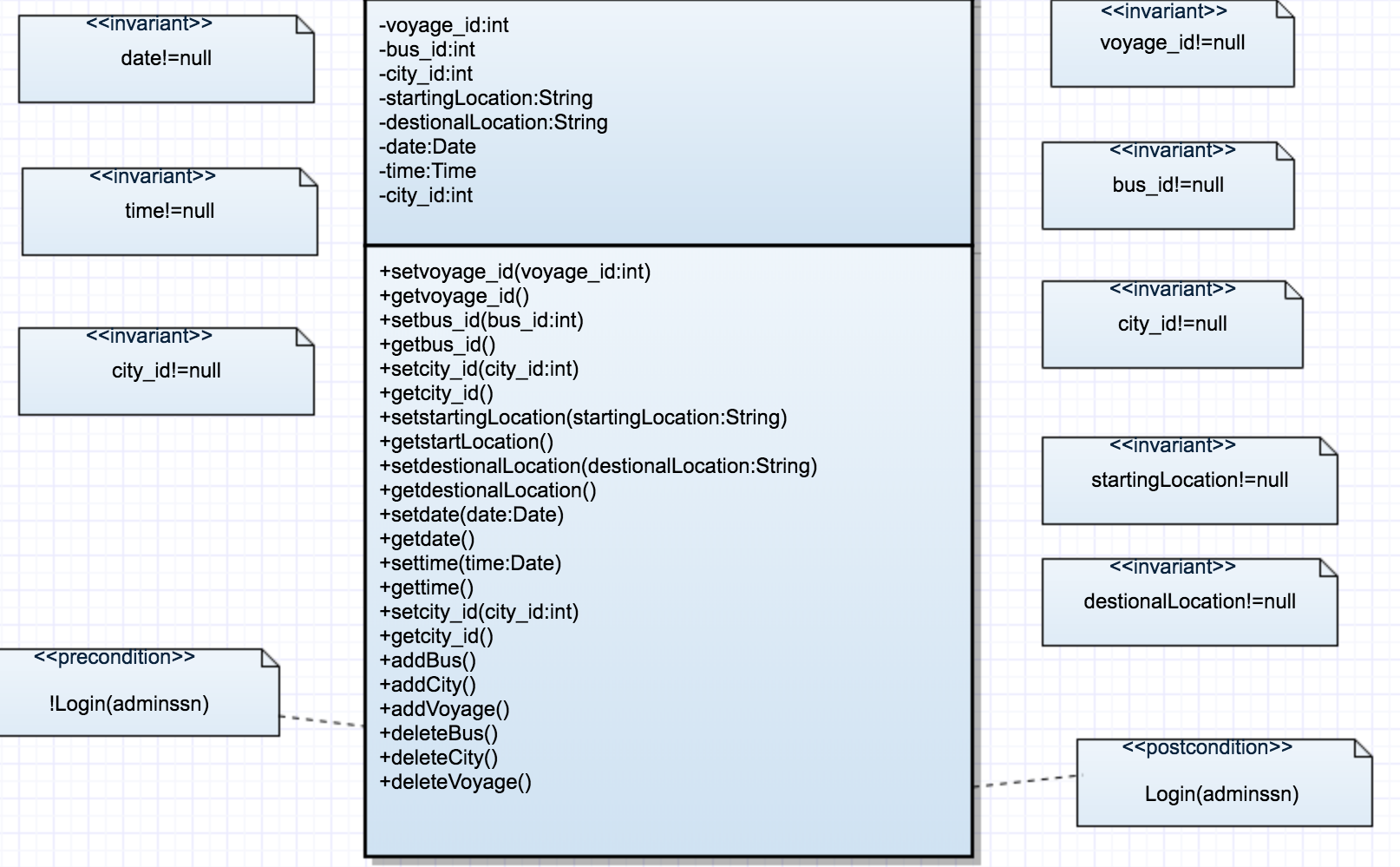


Figure 3.2- Voyage UML

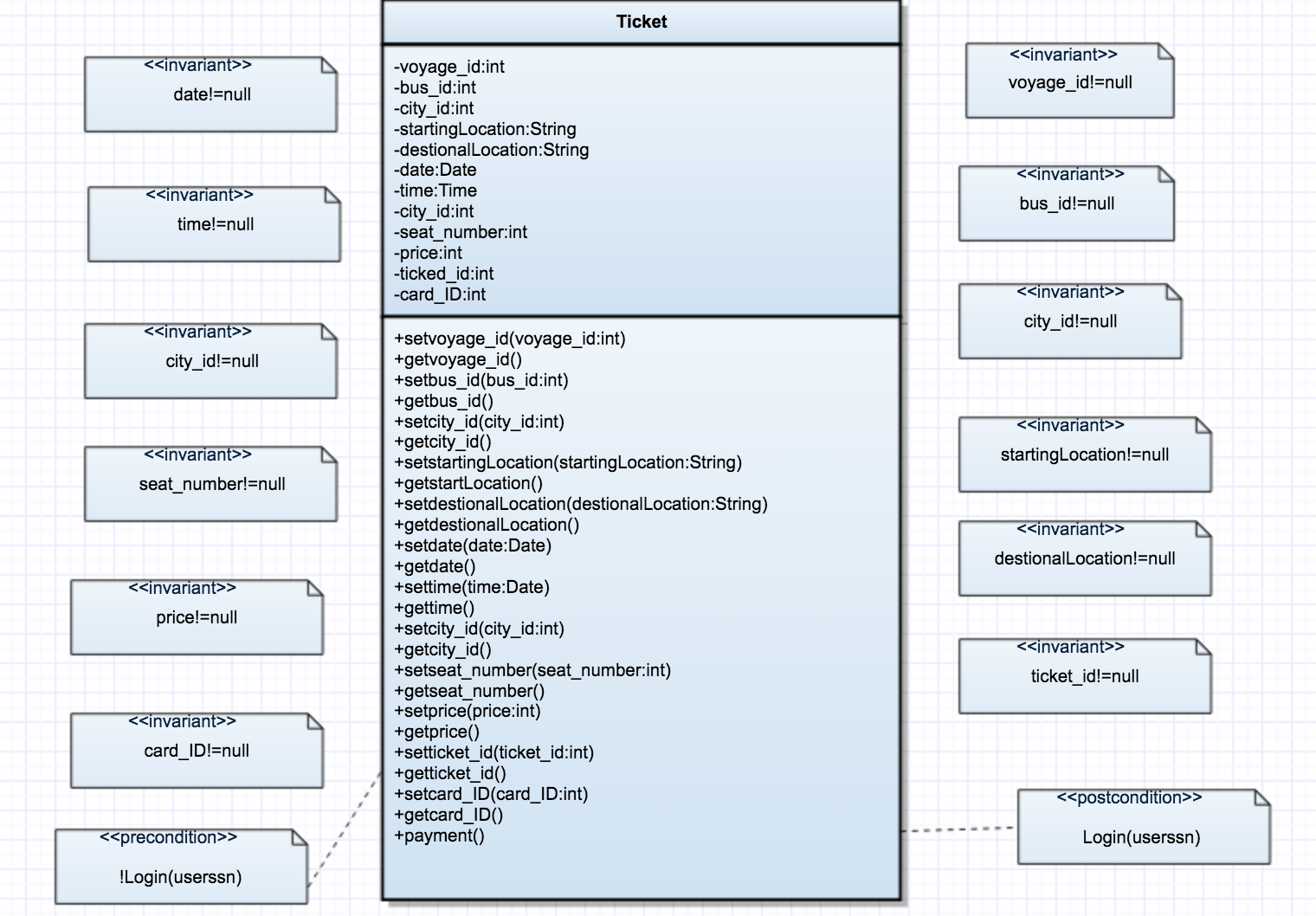
**Ticket**

\*Invariants:  
\*@invariant voyage\_id!=null  
\*@invariant bus\_id!=null   
\*@invariant city\_id!=null   
\*@invariant startingLocation!=null   
\*@invariant destionalLocation!=null  
\*@invariant date!=null  
\*@invariant time!=null  
\*@invariant seat\_number!=null  
\*@invariant price!=null  
\*@invariant ticket\_id!=null  
\*@invariant card\_id!=null

\*  
\*  Preconditions:   
\*  login(userssn, password)   
\*  @pre getpassword()!=null

\*   
\*  PostConditions:

\*  Login(userssn, password)   
\*  @post Login == true

\*  
 Figure 3.3- Ticket UML

**Booking**

\*Invariants:  
\*@invariant voyage\_id!=null  
\*@invariant bus\_id!=null   
\*@invariant city\_id!=null   
\*@invariant startingLocation!=null   
\*@invariant destionalLocation!=null  
\*@invariant date!=null  
\*@invariant time!=null  
\*@invariant seat\_number!=null  
\*@invariant ticket\_id!=null  
\*  Preconditions:   
\*  login(userssn, password)   
\*  @pre getpassword()!=null

\*   
\*  PostConditions:

\*  Login(userssn, password)

\*  @post Login == true

\*

\*

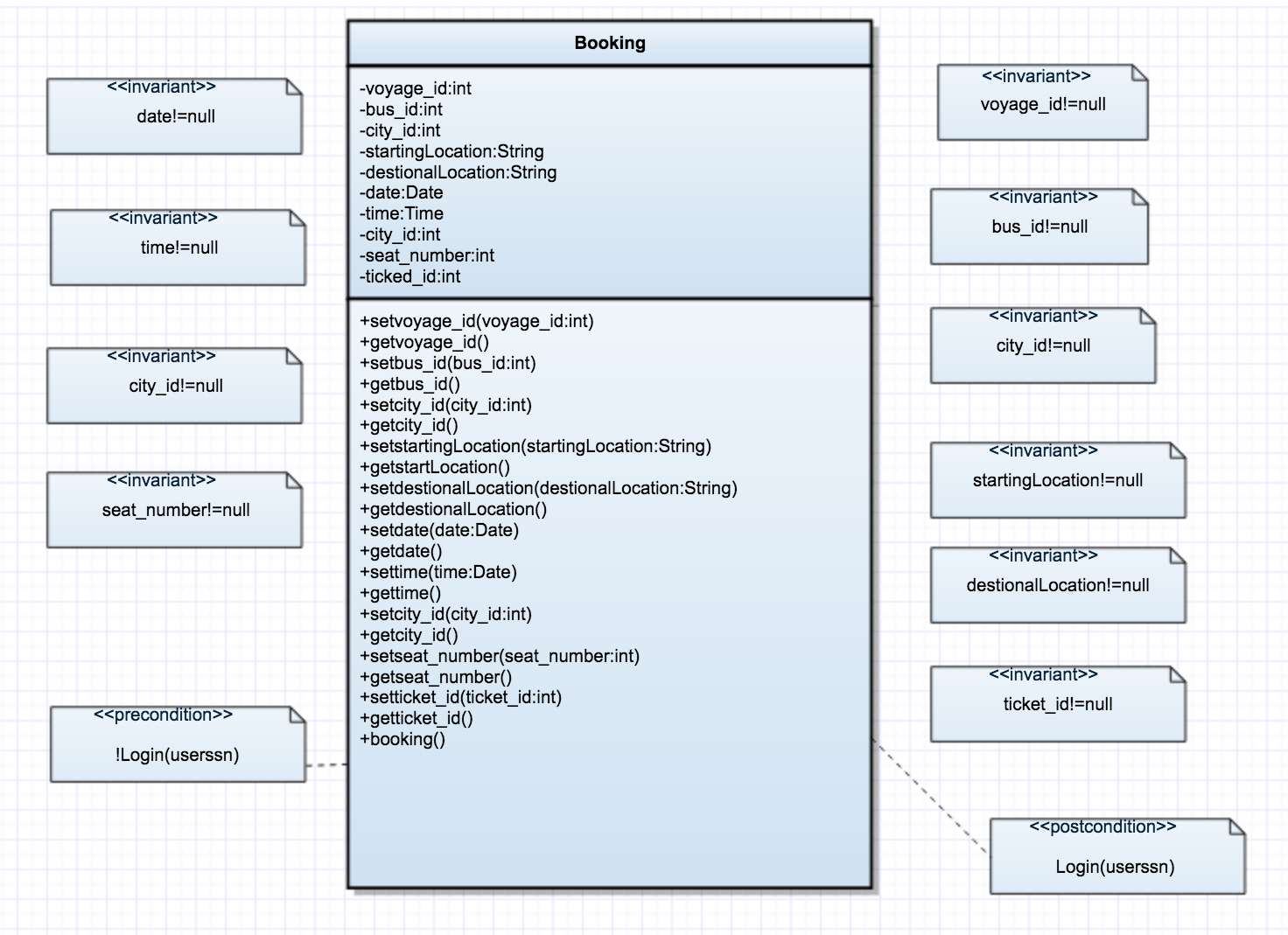


Figure 3.4- Booking UML

**User**

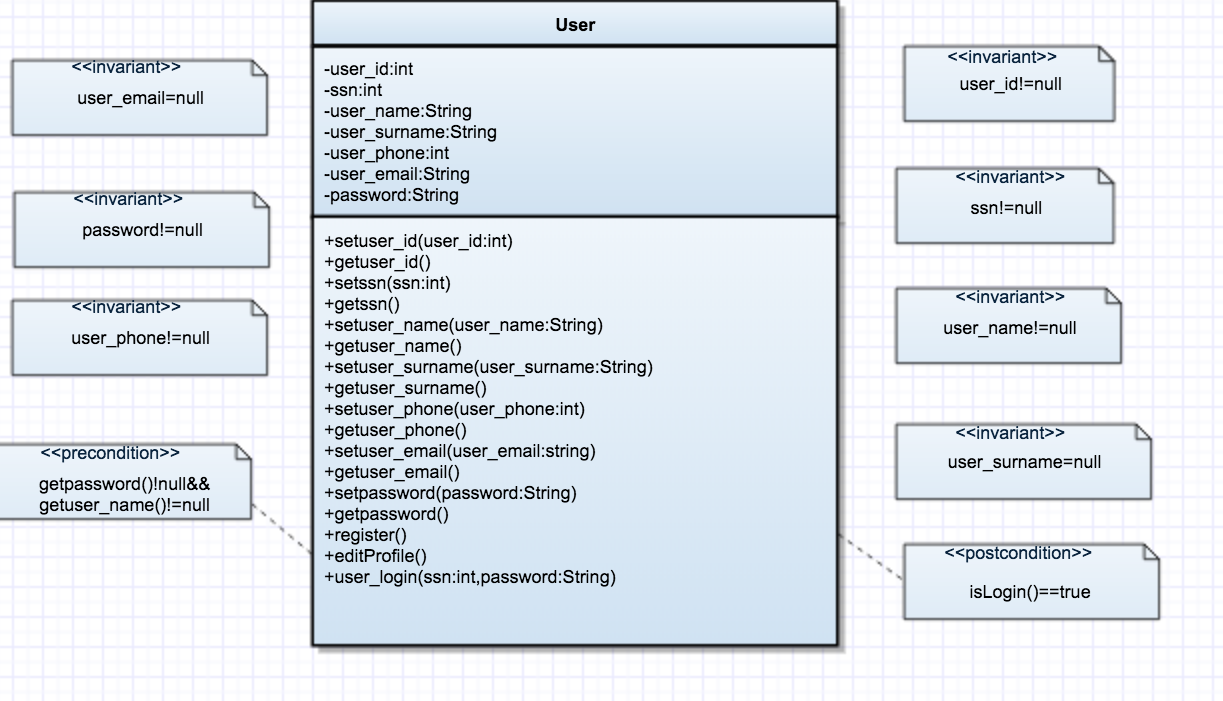
\*@invariant user\_id!=null   
\*@invariant ssn!=null  
\*@invariant user\_name!=null   
\*@invariant password!=null  
 \*@invariant user\_surname!=null  
\*@invariant email!=null  
\*@invariant phone!=null\*   
\*  Preconditions:   
\*  login(userssn, password)   
\*  @pre getpassword()!=null \*   
\*  PostConditions:   
\*  Login(userssn, password)   
\*  @post Login == true \*  


Figure 3.5- User UML