Assignment A1
Analysis and Design Document

Student: Linca Paul Tudor
Group: 30432

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

Use JAVA/C# to design and implement an application for an online Parking Request System. The main goal of the application is to provide clear and transparent way of requesting and assigning parking spots in the parking lots of Cluj-Napoca. Each citizen can make a new request for a parking spot. The request can be done only for one car that the citizen owns, but he can select multiple parking lots that would suit him good. Each parking lot has a certain number of parking spots. If the citizen has multiple cars, he must file multiple parking requests.

The application should have two types of users: a regular user represented by the citizen and an administrator user, represented by the city clerk responsible for assigning the free parking spots. Both kinds of uses have to provide an email and a password in order to access the application.

1.2 Functional Requirements

The main objective of this assignment is to become familiar with Layers architectural pattern. Furthermore, we aim to become familiar with the DAO pattern for database access and Transaction Script / Table Module for domain modeling. Store data in a MySQL database. Implement the Data Access Layer both with Hibernate and JDBC. Use the Abstract Factory pattern to switch between implementations.

1.3 Non-functional Requirements

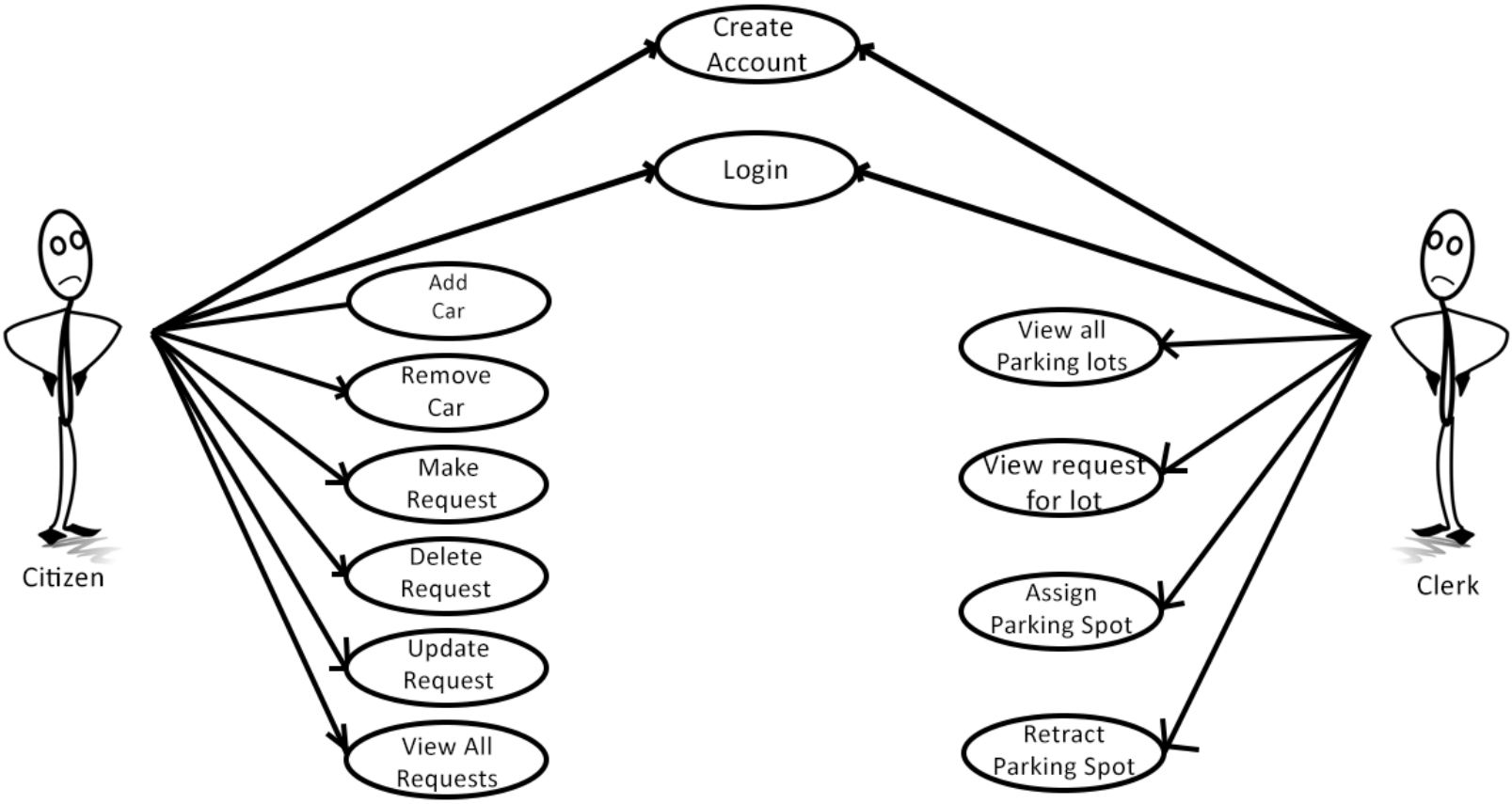
- The application should be accessible and easy to use.
- The application must be secure
- The user interface must update instantly (in under 1 second)
- The application must not jam

2. Use-Case Model

Use case: Create new parking request
Level: user-goal level
Primary actor: Citizen
Main success scenario:

1. Open Application.
2. Provide username and password
3. Press the “Login” button.
4. Select the “Make Request” tab.
5. Select the car and parking lots for the request.
6. Press the “Make request” button.
7. The system confirms the request and updates the database.

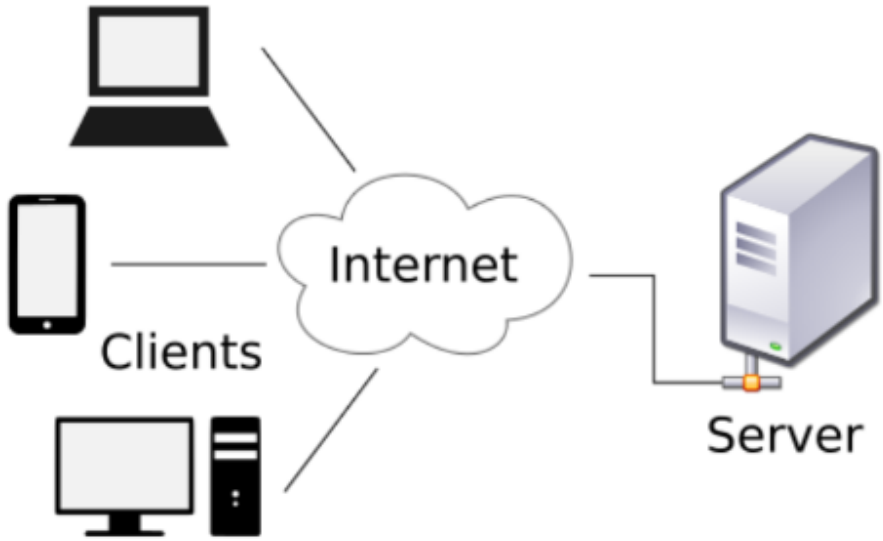
Extensions: If the PTI of the car is not valid, the system will reject the request and provide a message to the user.



3. System Architectural Design

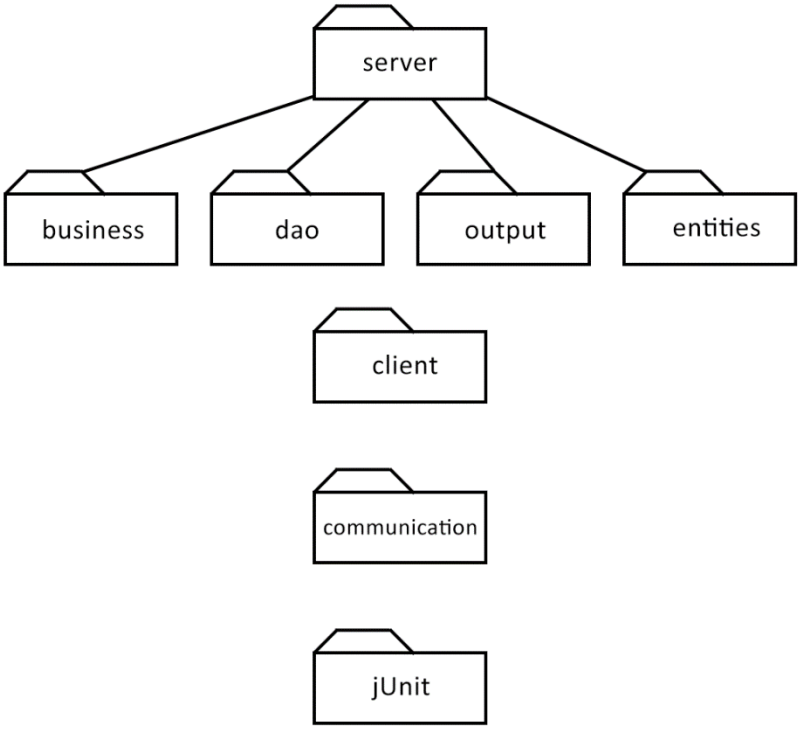
3.1 Architectural Pattern Description

The client–server model of computing is a distributed computing structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.

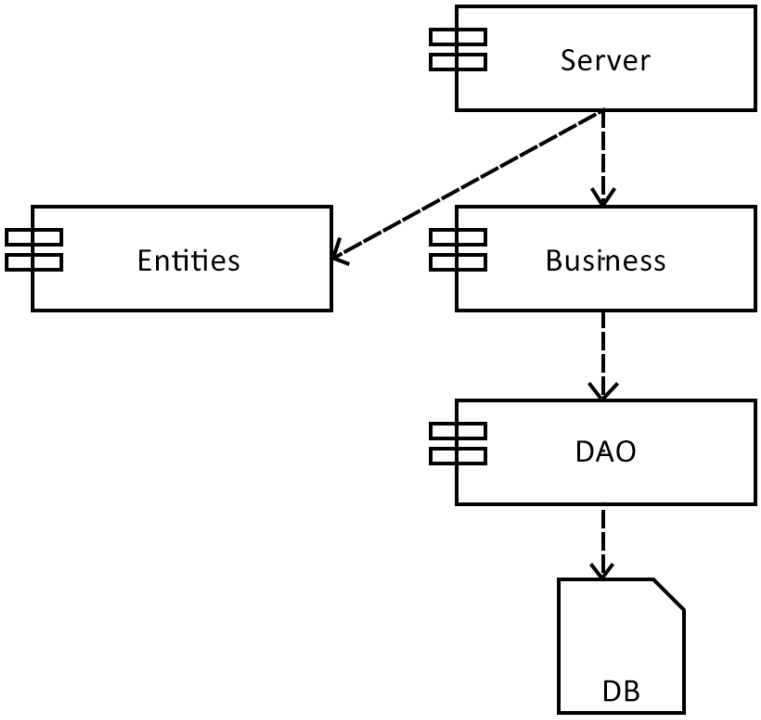


3.2 Diagrams

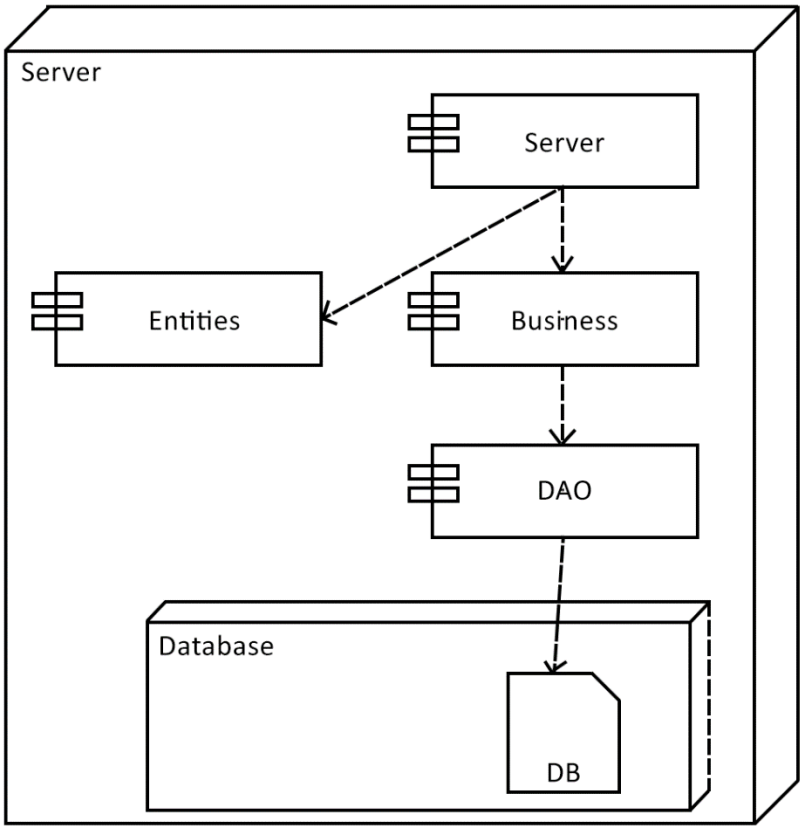
Package Diagram



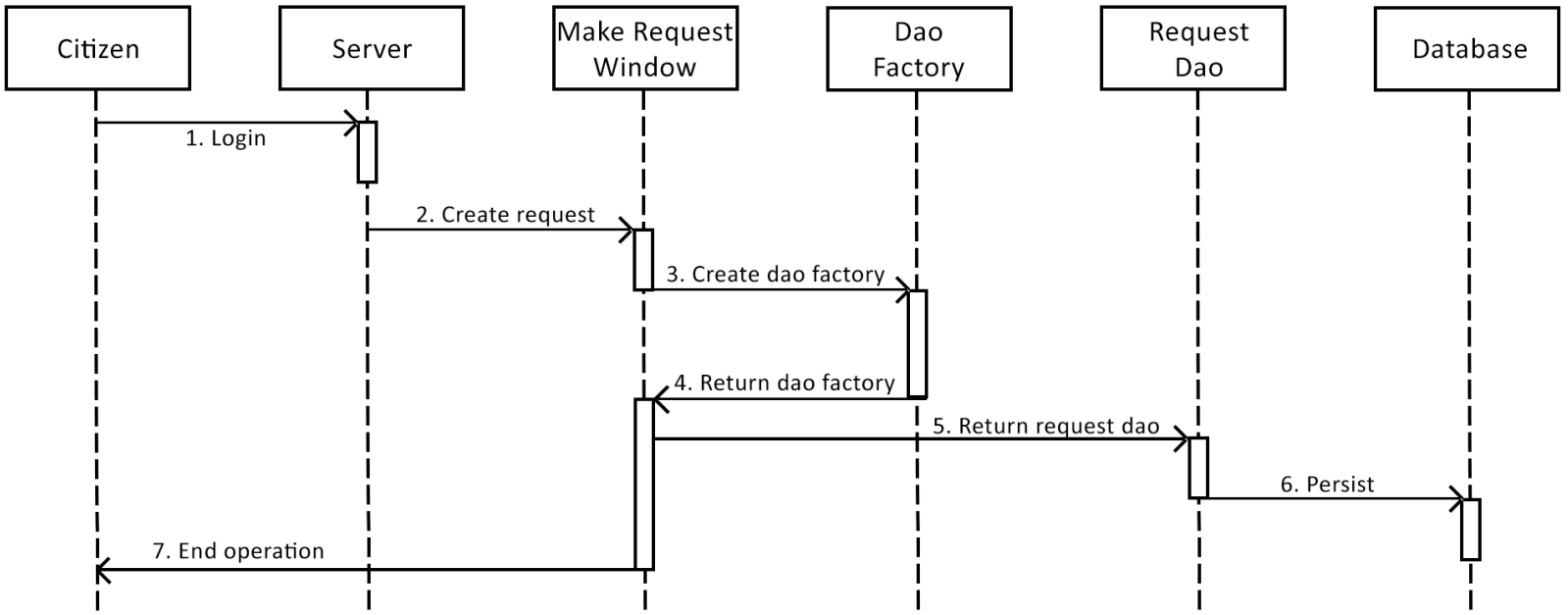
Component Diagram



Deployment diagram



4. UML Sequence Diagrams



5. Class Design

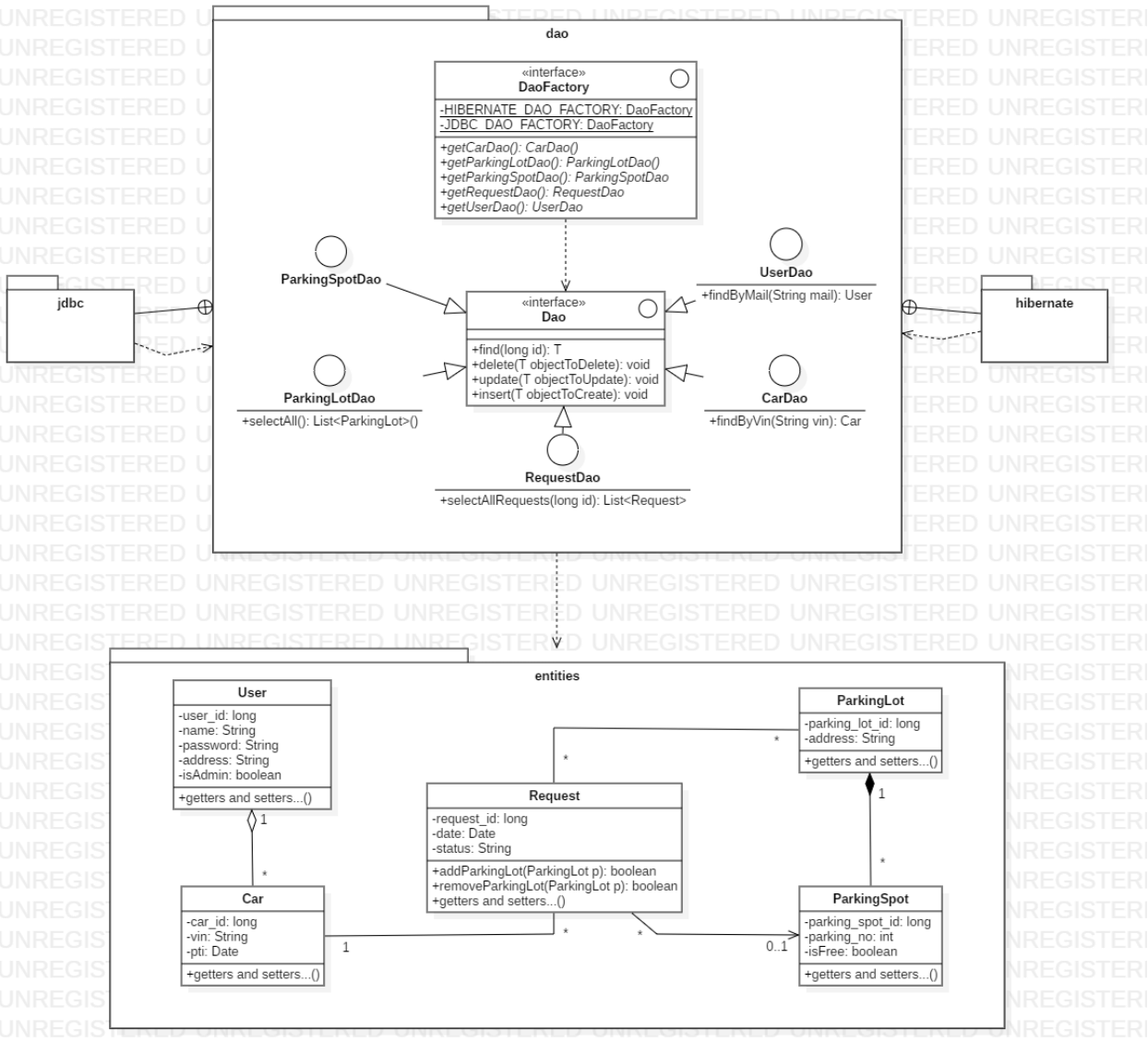
5.1 Design Patterns Description

The **Data Access Object (DAO) pattern** is a structural pattern that allows us to isolate the application/business layer from the persistence layer (usually a relational database, but it could be any other persistence mechanism) using an abstract API.

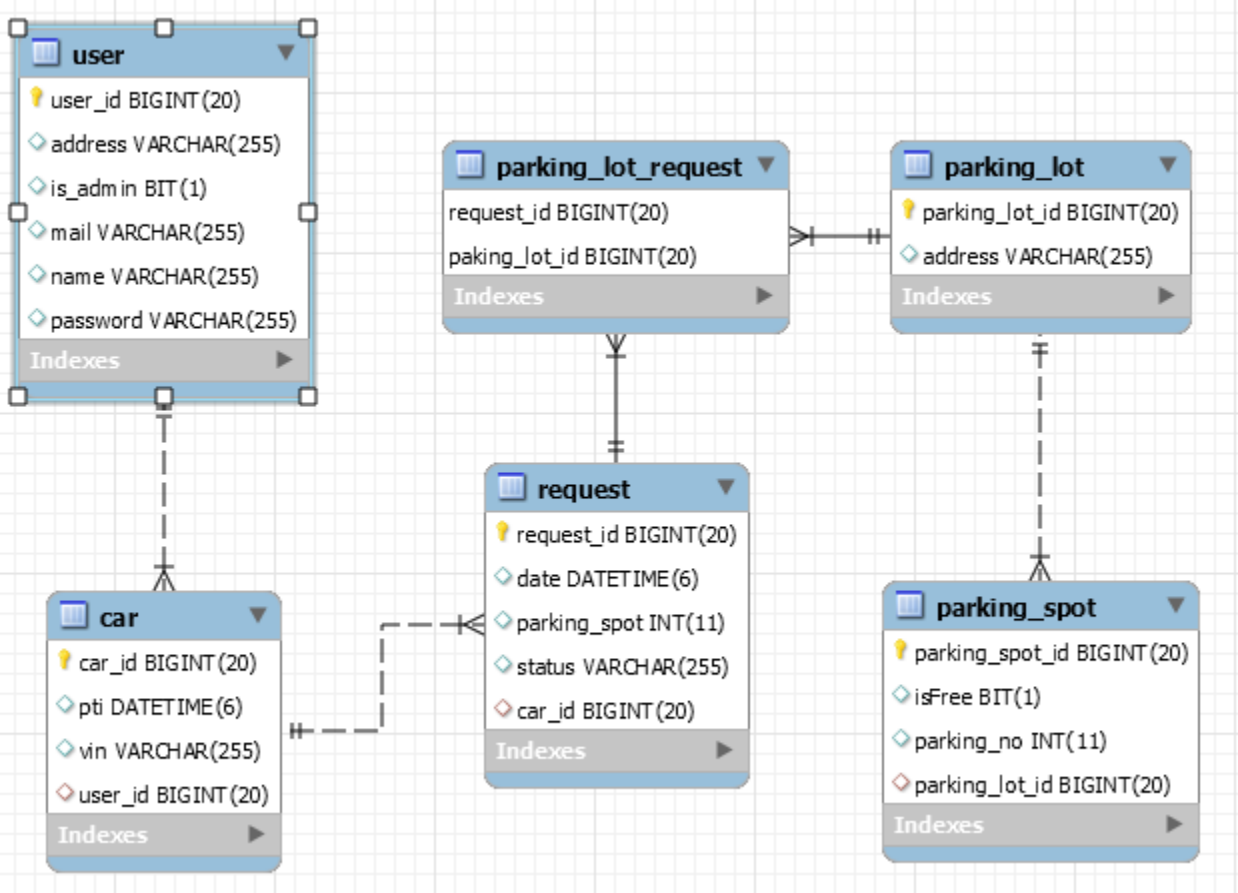
Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

5.2 UML Class Diagram



6. Data Model



7. System Testing

The system testing was performed using unit testing for the Data Mappers, integration testing for the integration of the client with the server, and validation testing for the finished system. The unit testing was performed using jUnit in the TestCases class. Integration testing was performed using the graphical interface by testing each functionality in turn for main success and fail scenarios (data-flow strategy). Each Use Case has been tested using the data-flow strategy.

8. Bibliography

https://www.tutorialspoint.com/design_pattern/abstract_factory_pattern.htm
https://www.baeldung.com/java-dao-pattern
https://github.com/buzea/SoftwareDesignLabResources