Driving School Management System

Analysis and Design Document

Student: Fazekas Vlad

Group:30234

Revision History

Date	Version	Description	Author
<dd/mmm/yy>	<x.x>	<details>	<name>

Table of Contents

I. Project Specification

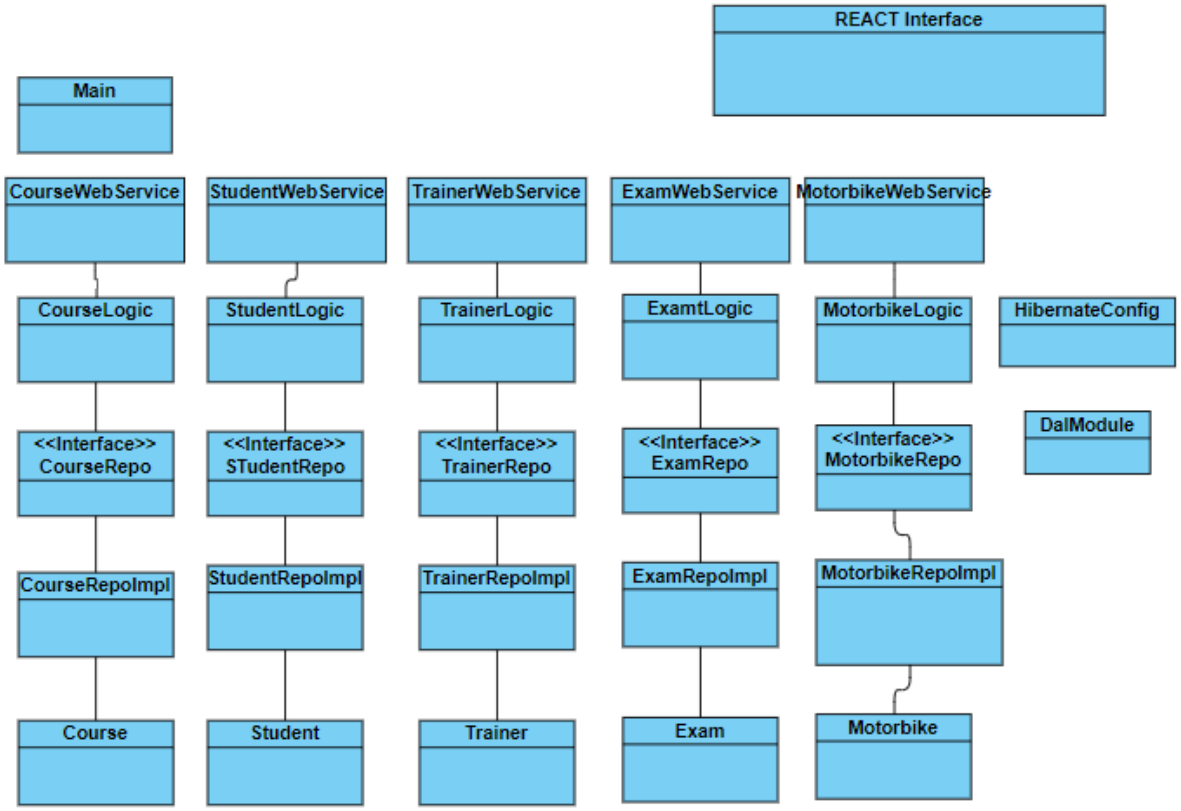
A web application for the management of a driving school, that includes the interaction from the user/students and from the administrators/trainers of the driving school.

There are two types of users:

- Trainers – will be able to see their students, see reports, see who is going to have the exams next week, add exam questions, keep track of the motorcycles
- Students – will be able to take exams, change their data, chose the motorcycles

II. Elaboration – Iteration 1.1

1. Domain Model



2. Architectural Design

2.1 Conceptual Architecture

The systems architecture will be based on the Layers architectural pattern and model-view-controller pattern.

Dividing an application into separate layers that have distinct roles and functionalities helps you to maximize maintainability of the code, optimize the way that the application works when deployed in different ways, and provides a clear delimitation between locations where certain technology or design decisions must be made.

That's why we will divide our application in three big layers:

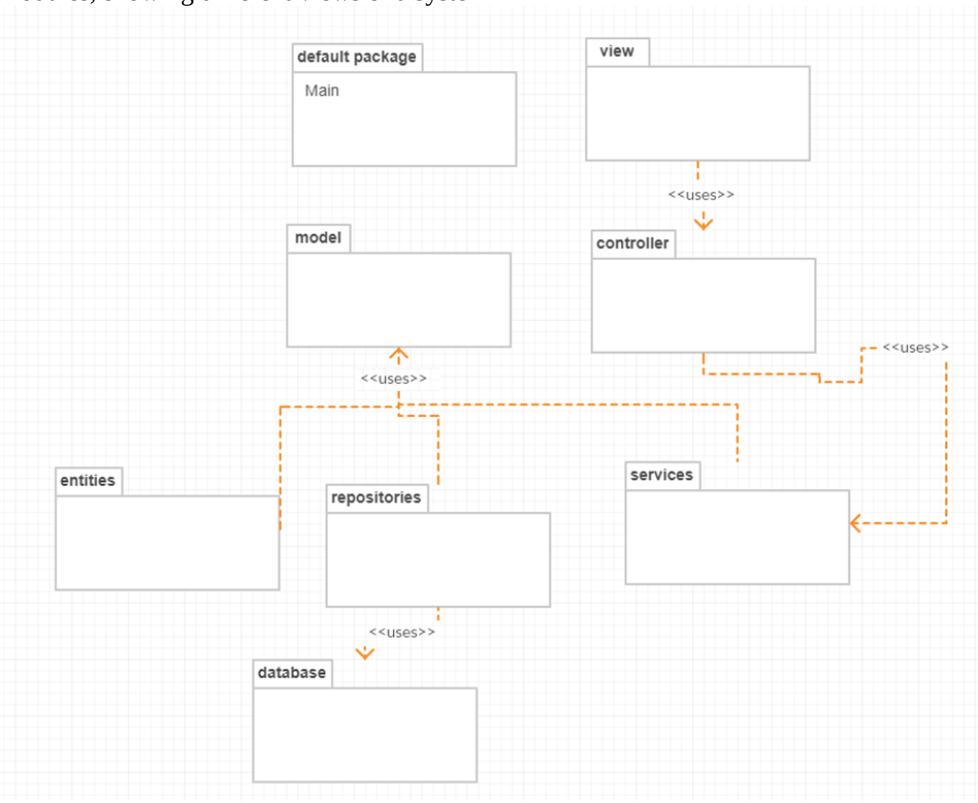
- Presentation layer (where all the interaction of the user with the system will reside)
- Business layer (where the core functionality and relevant logic of the system will be implemented)
- Data layer (where access to data is provided(generally through generic interfaces that the components in the business layer can consume)

Model–View–Controller (usually known as MVC) is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. It also provides a much clearer overall view of the system and a more structured code that is easier to expand with new functionalities. That's why we will divide our application in the following structure:

- Controller
- View
- Model
  - Entities
  - Repositories
  - Services
- Database

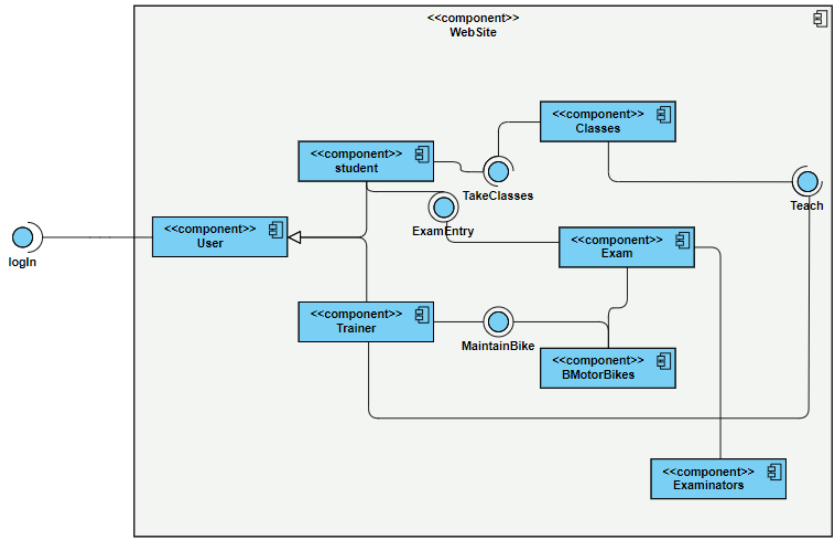
2.2 Package Design

Package diagram, a kind of structural diagram, shows the arrangement and organization of model elements in middle to large scale project. Package diagram can show both structure and dependencies between sub-systems or modules, showing different views of a system

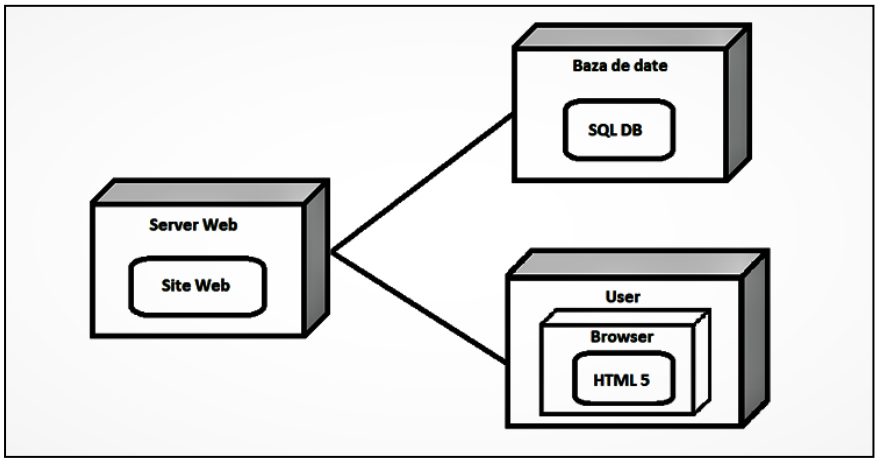


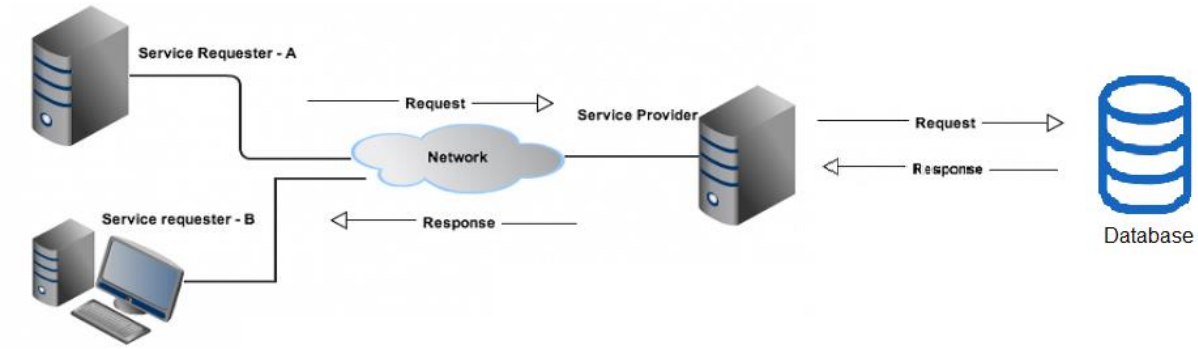
2.3 Component and Deployment Diagrams

Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.



A UML deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Deployment diagrams is a kind of structure diagram used in modeling the physical aspects of an object-oriented system. They are often be used to model the static deployment view of a system (topology of the hardware).



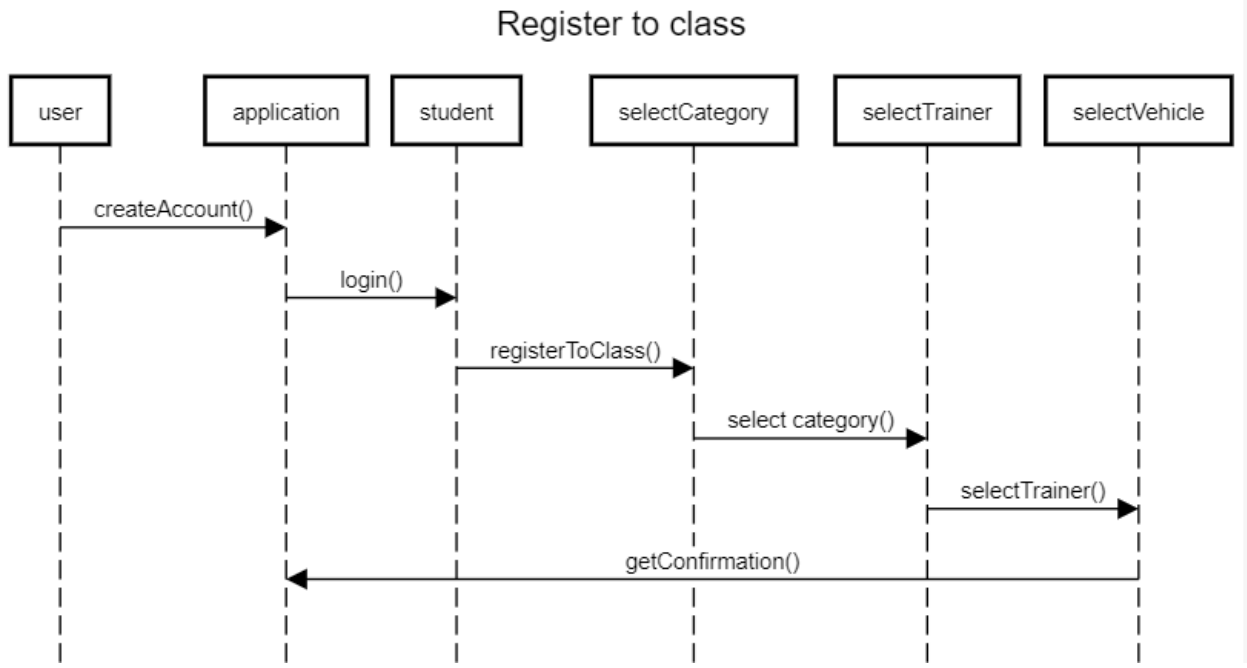


III. Elaboration – Iteration 1.2

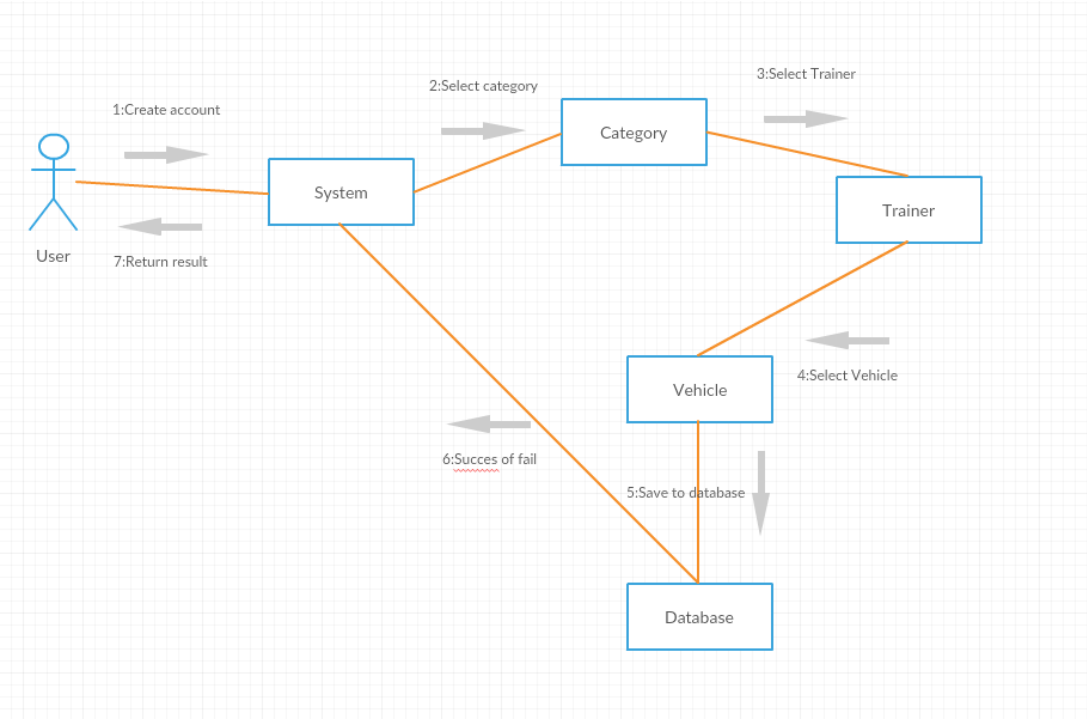
1. Design Model

1.1 Dynamic Behavior

A **sequence diagram** shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

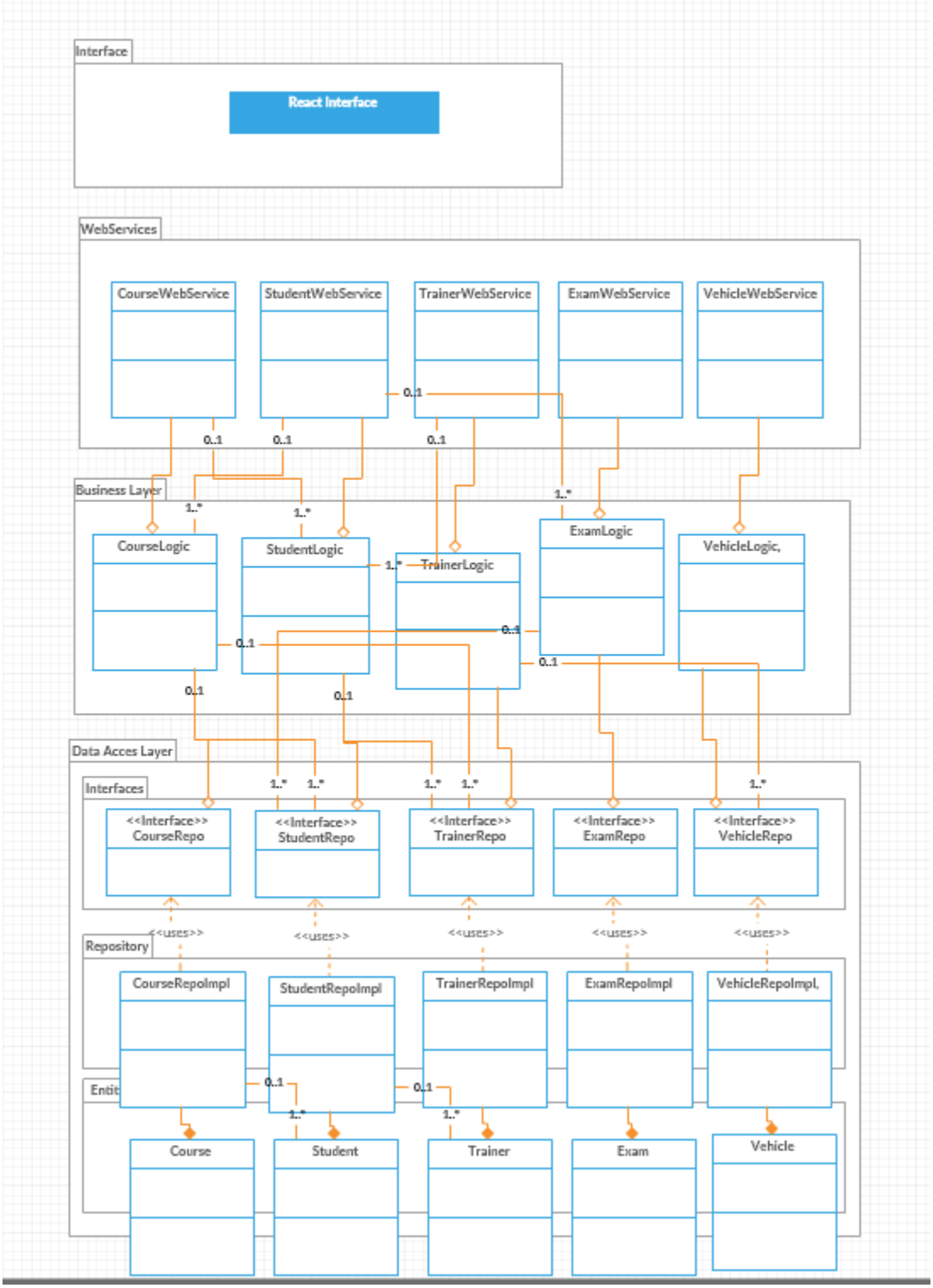


A Communication diagram models the interactions between objects or parts in terms of sequenced messages. Communication diagrams represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system. However, communication diagrams use the free-form arrangement of objects and links as used in Object diagrams.

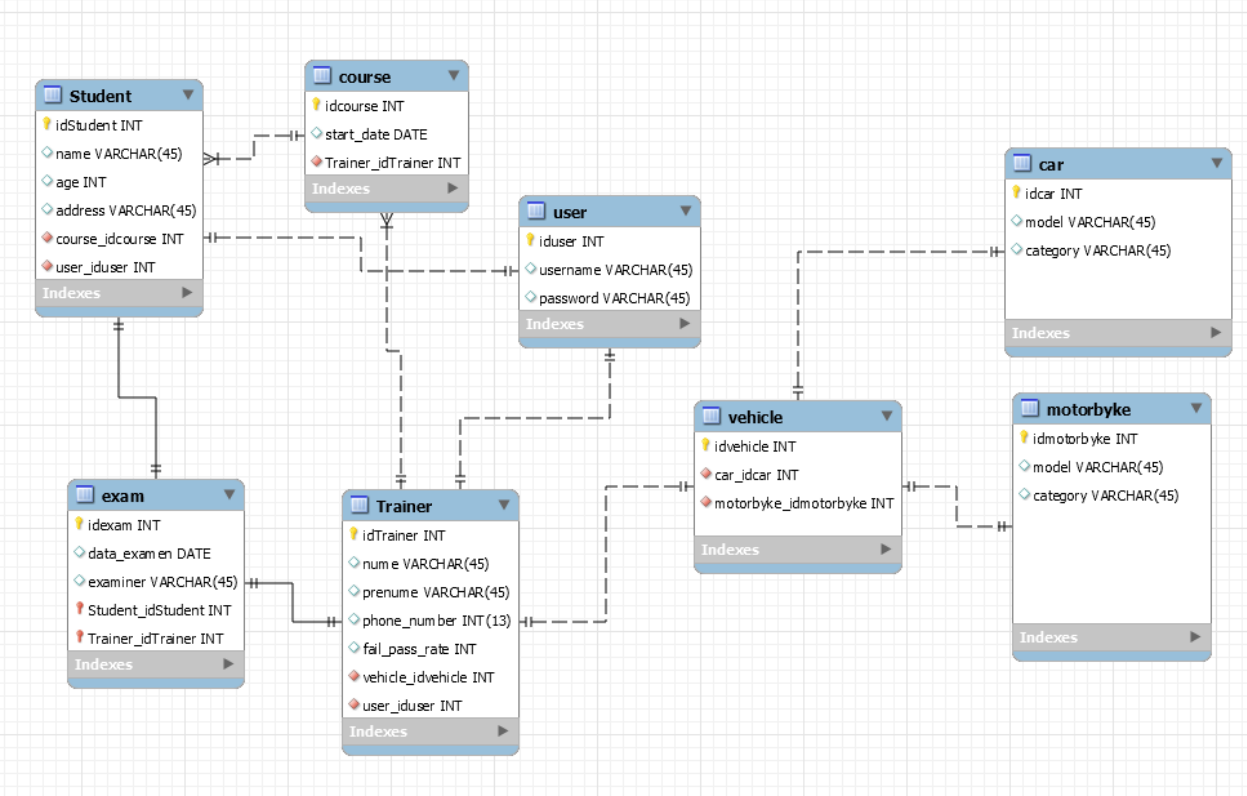

1:Create account
2:Select category
3:Select Trainer
Category
System
Trainer
User
7:Return result
4:Select Vehicle
Vehicle
6:Succes of fail
5:Save to database
Database

1.2 Class Design

The singleton pattern is used for the database connection. Also, the builder will be used (there will be a build method that will construct an instance of a class).

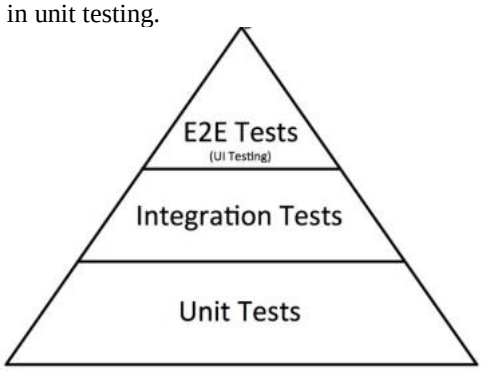


2. Data Model



3. Unit Testing

Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.



As unit tests are at the base of the pyramid of testing, we will come with some advantages to support why it is like that:

- Unit testing increases confidence in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change.
- Codes are more reusable
- Debugging is easy. When a test fails, only the latest changes need to be debugged.
- Etc

In this project we will use Junit and Mockito.

JUnit is a unit testing framework for Java programming language, and between the advantages that it provides are:

- Provides annotations to identify test methods.
- Provides assertions for testing expected results.
- Provides test runners for running tests.

Mockito is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. Mockito is used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing.

Mockito facilitates creating mock objects seamlessly. It uses Java Reflection in order to create mock objects for a given interface. Mock objects are nothing but proxy for actual implementations.

Some of the test case scenarios:

A Test Scenario is defined as any functionality that can be tested. It is a collective set of test cases which helps the testing team to determine the positive and negative characteristics of the project.

- Check the Login functionality
- Check the take exam functionality
- Check the registration of a new student functionality
- Check the bike management functionality

IV. Elaboration – Iteration 2

1. 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.]

2. Design Model Refinement

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

V. Construction and Transition

1. System Testing

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

2. Future improvements

[Present future improvements for the system]

VI. Bibliography