Manhattan School of Driving LLC, Management System

Date Here

Elaboration1 Document

By:
Author 1
Author 2
Author 3

Date	Version	Author	Summary of Changes Made
3/25/2013	1.0	Author 1	Document Creation
3/25/2013	1.1	Author 2	Added Supplementary Specifications
3/27/2013	1.2	Author 3	Modificati on

Contents

Vision and Business Case

Introduction

The Manhattan School of Driving LLC was established in 2008 on the upper east side of Manhattan. For the last 2 years, the school has devoted time to teach students from the age of 16 to 84 by providing instructions in a spotless, dual-brake, late model BMW. The school also offers many different driving courses such as the New York State 5-hour pre-licensing class in its comfortable, fully modern classroom. The 6-hour point and insurance-reduction class is also part of their curriculum.

Business Goals

During the last 2 years, the Manhattan School of Driving has successfully multiplied the number of students from its previous year, while forecasting a continuing growth for the next years to come. In order to guarantee such successful forecast, the school has set some goals and guidelines that they wish to accomplish. Some of these business goals are:

- Passing the DMV on-line training audit approval
- Providing more services for customers online

The Manhattan School of Driving is willing to invest time and money to accomplish their goals. Most importantly, by accomplishing these goals, customer and employee satisfaction will be reached.

Problem Statement

As the school continues to succeed and contract new students and employees, the company has faced a road block using their current paper based system. Keeping students records organized and accessible to both student and employees have become a hassle, error prone and cumbersome. Currently, the school has been able to manage its student and employee data with their current paper-base management system. It has been estimated that using the current paper-base system by the end of this year (2010) will no longer be an effective and efficient way to manage student and employee records. The following is a list of problems faced by students and employees due to the school's current paper-based system:

- Students don't have access to their own training records
- Employees must search piles of folders stored in file cabinets in order to retrieve a student's record
- All student and employee records are stored in file cabinets, unsecured and without backups
- Not all student records are filed correctly and inconsistent data exist
- There is no way to analyze the school data in order to generate valuable information or statistics about their students and performance
- Class schedules are kept in a notebook, which has been the reason for classes being scheduled incorrectly or having classrooms overbooked

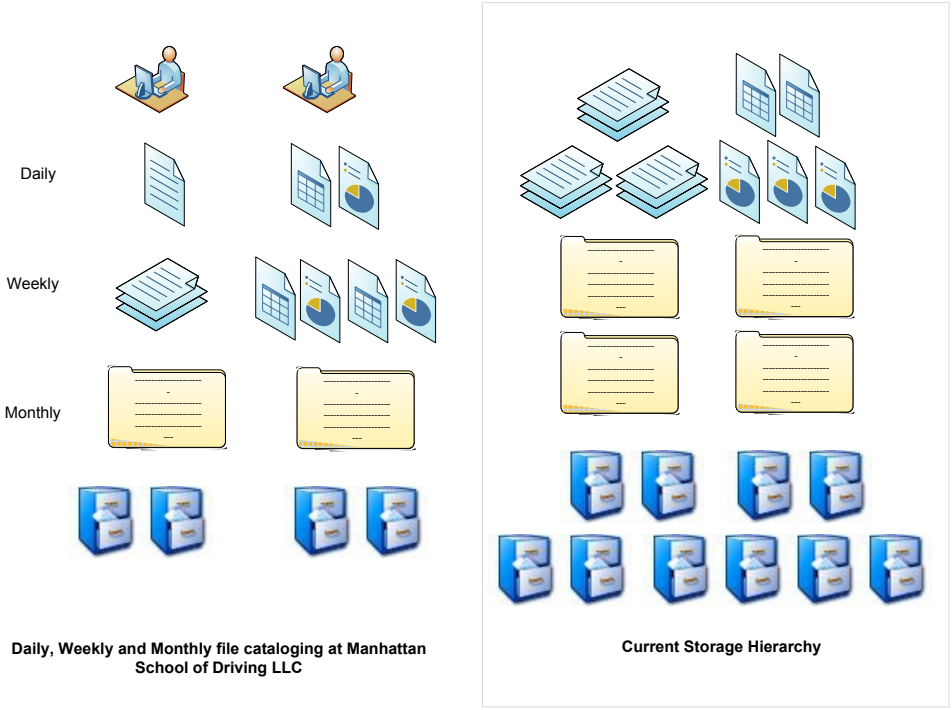


Figure 1: A visual representation of the file storage system currently in place for the Manhattan School of Driving LLC.

Business Case

Implementing the Manhattan School of Driving Management System has many positive outcomes for both the business and the developers. From the business point of view, a new system is definitely needed by the end of this year, the business success and its employees as well as customers satisfaction depends on this new system. Not all driving schools have been fortunate enough to have the large amount of success that Manhattan School of Driving is currently having. Because of this, not all driving schools need a system like the one being requested. Not only would Manhattan School of Driving be one of the few schools in the tri-state area with such advance management system but they will also set the standards for future driving schools and be one of the few to pass the DMV on-line training audit approval.

From the developer's point of view, the software system being requested by the business is not out of the ordinary. Many management systems have been built in the past, and some of these systems have shared their code to the public, allowing developers to reuse existing source code and modules that have been tested to work and proven efficient. Not only is this advantage for the developers but also to the business.

Business Constraints

With every project, for either a small or large business, there is always constraints that both the business and developer should be aware about. For this particular project, these are some possible identified constraints:

- Even if the change is made for the greater good and for the advantage of the greater population, there will always be a few that will dislike the change and will struggle adapting to these changes
- Not all customers and employees can be categorized as technically savvy. There will be a few customers (for example., elderly students) that may struggle learning and using the new system
- Not all students have computers outside the school, accessing their student records may still require either contacting the school directly or visiting the school
- Before the system can prove its advantages and success, the new management system will need to be advertised to all students. A system that is not in used have no positive effect
- The new management system will require a larger budget than the current paper-based system

Executive Summary

The Manhattan School of Driving Management System is meant to help the school reach its 2010 business goals. Accomplishing these business goals will guarantee that the school will remain as one of the best in the tri-state area. In order to accomplish its business goals, the school will have to go through a technological reform, which will include eliminating their current paper-based system and implementing a system that will better manage its growing needs. The current paper-based system has imposed many problems, some which have been described in the problem statement. Implementing the new management system should alleviate these problems and provide flexibility for future business growth.

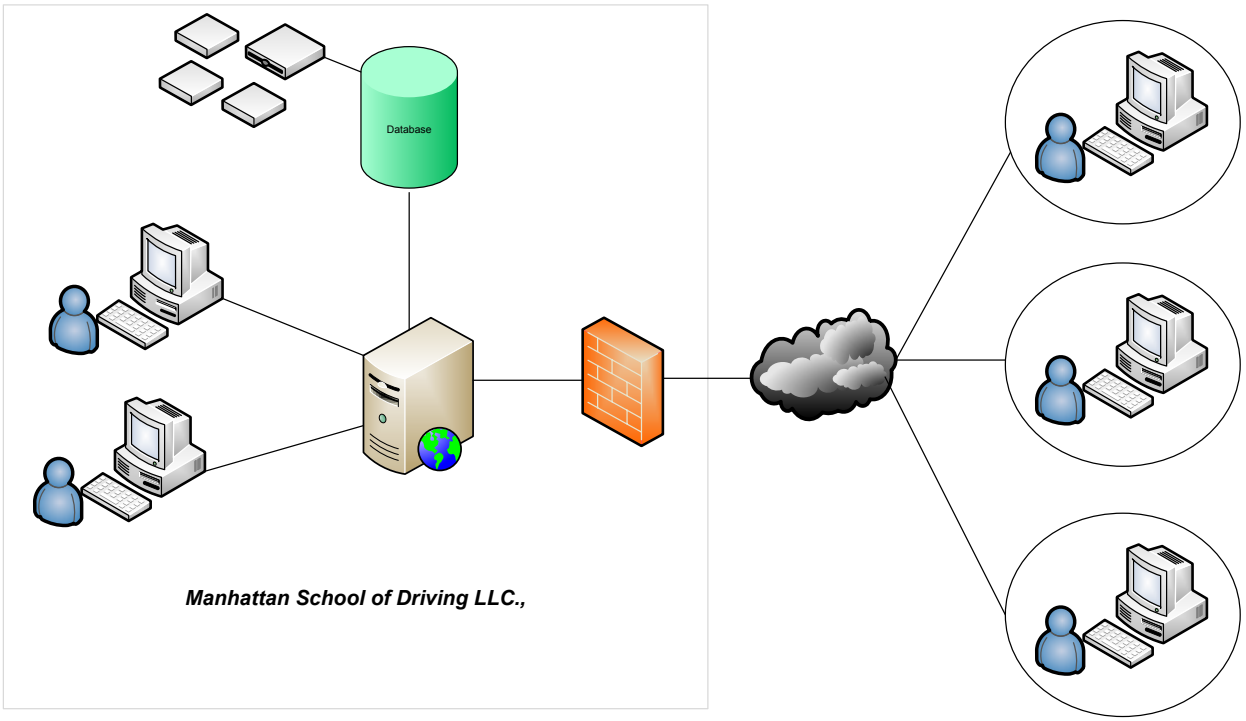


Figure 2: A visual representation of the technological reform

The advantages of implementing this system are many. Not only will the business get a system that they can use for many years to come, but this will also open a brand new market for all driving school if the system is implemented successfully. To the developer's advantage, the developers can make good use of software reusability for the current project and for future driving school management system projects as well.

Use-Case Models

Use Case UC1: Log-in To System

Primary Actor: Receptionist, Trainer, Student
Preconditions: Receptionist, Trainer or Student must have a username and password
Post conditions: Users log in to the system
Main Success Scenario:

1. Receptionist, Trainer or Student navigates to the system's web site using the link provided and a web browser that meets system requirements.
2. Receptionist, Trainer or Student enters his/her username and password and clicks OK.
3. Receptionist, Trainer or Student is redirected to his/her profile home screen.

Use Case UC2: Student Search

Primary Actor: Receptionist, Trainer
Preconditions: Receptionist or Trainer must log in to the system
Post conditions: Zero or many student records are display
Main Success Scenario:

1. Receptionist or Trainer clicks on 'Student Search' from the main menu and gets redirected to the Student Search page.
2. Receptionist or Trainer enters a combination of first name, last name or student number and selects Search.
3. A table containing zero or many records is displayed based on the search result.

Use Case UC3: Add a Lesson Taken

Primary Actor: Receptionist
Preconditions: Receptionist must log in to the system. A student must provide the receptionist with a 'Lesson Taken Receipt', signed by the trainer specifying the lesson type, lesson length, the lesson start time and lesson end time.
Main Success Scenario:

1. Receptionist searches for the student mentioned in the 'Lesson Taken Receipt'
2. The system displays a table list with the search results, which includes the student being searched
3. Receptionist clicks on the student name and is redirected to the student's page
4. From the 'Student Actions' menu, the Receptionist clicks on the 'Lesson Taken' link
5. The system redirects the client to a page where he/she can add a lesson taken and all its required information

Design Model

Design Class Diagram

The Class Diagram below is a representation of all the classes that are expected to be implemented in the Manhattan Driving School system. This is going to be the base of a larger scale project that we can hopefully expand upon in the future. These are the required elements that will need to be provided to the customer to make the software useable.

Figure 1: Class diagram depicting all classes that will be used in the Manhattan School of Driving system.

Domain Model

Current Domain Model Diagram

This Model shows the current classes that are implemented as of now in the driving school system. The main Student class holds basic information on the students of the school. This implementation allows us to show basic information on students and users in the system.

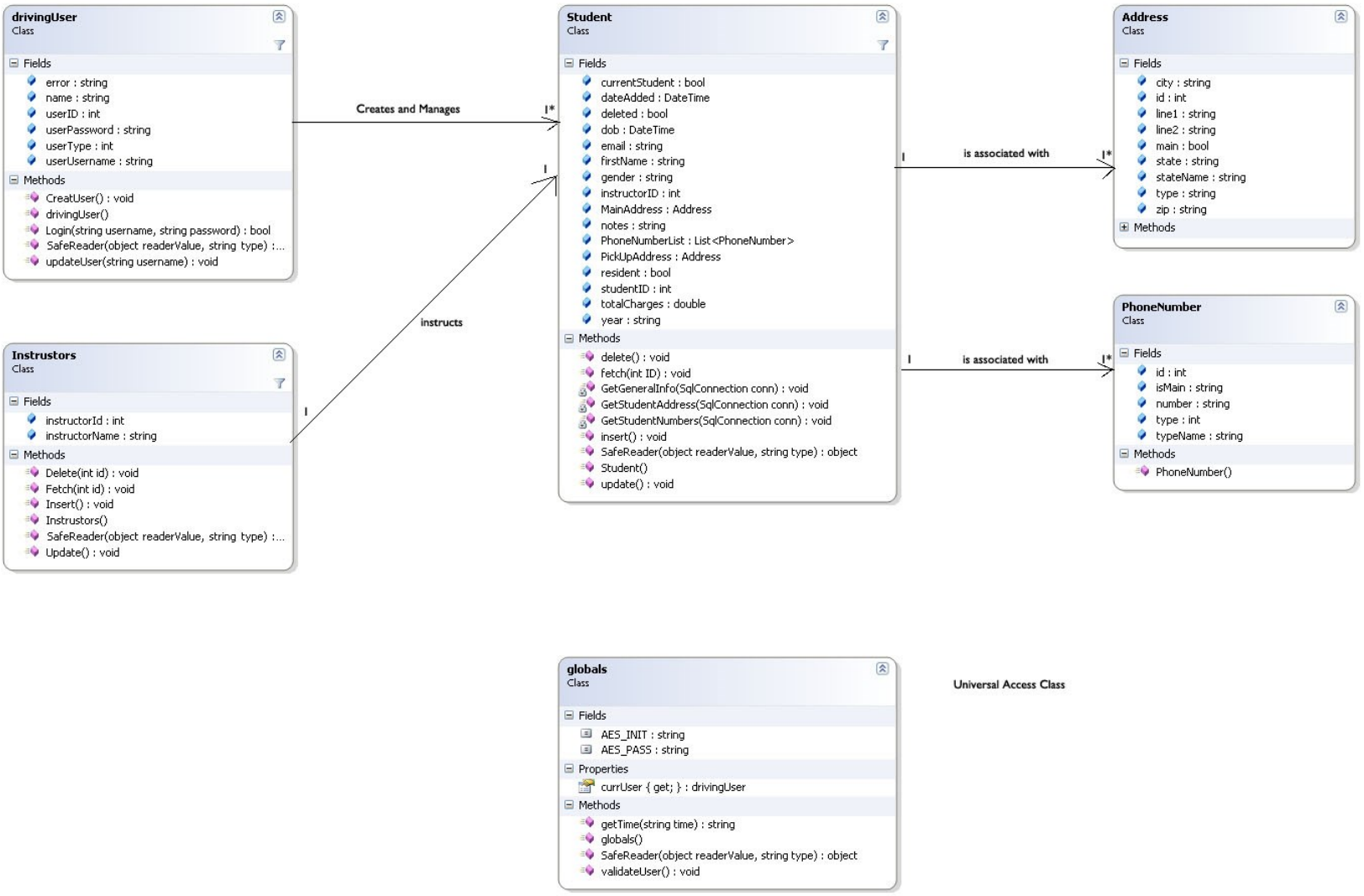


Figure 2: Domain Model diagram depicting all classes that are currently implemented in the Manhattan School of Driving system and their relationship with each other.

Data Model

Database ER Diagram

This Model shows the design of the entire database that is going to be implemented for this project. It shows all the tables along with their relationships with other tables in the database. This may change as the requirements of the user might change as we develop this application. As can be seen below the Student is the main focus in this application.

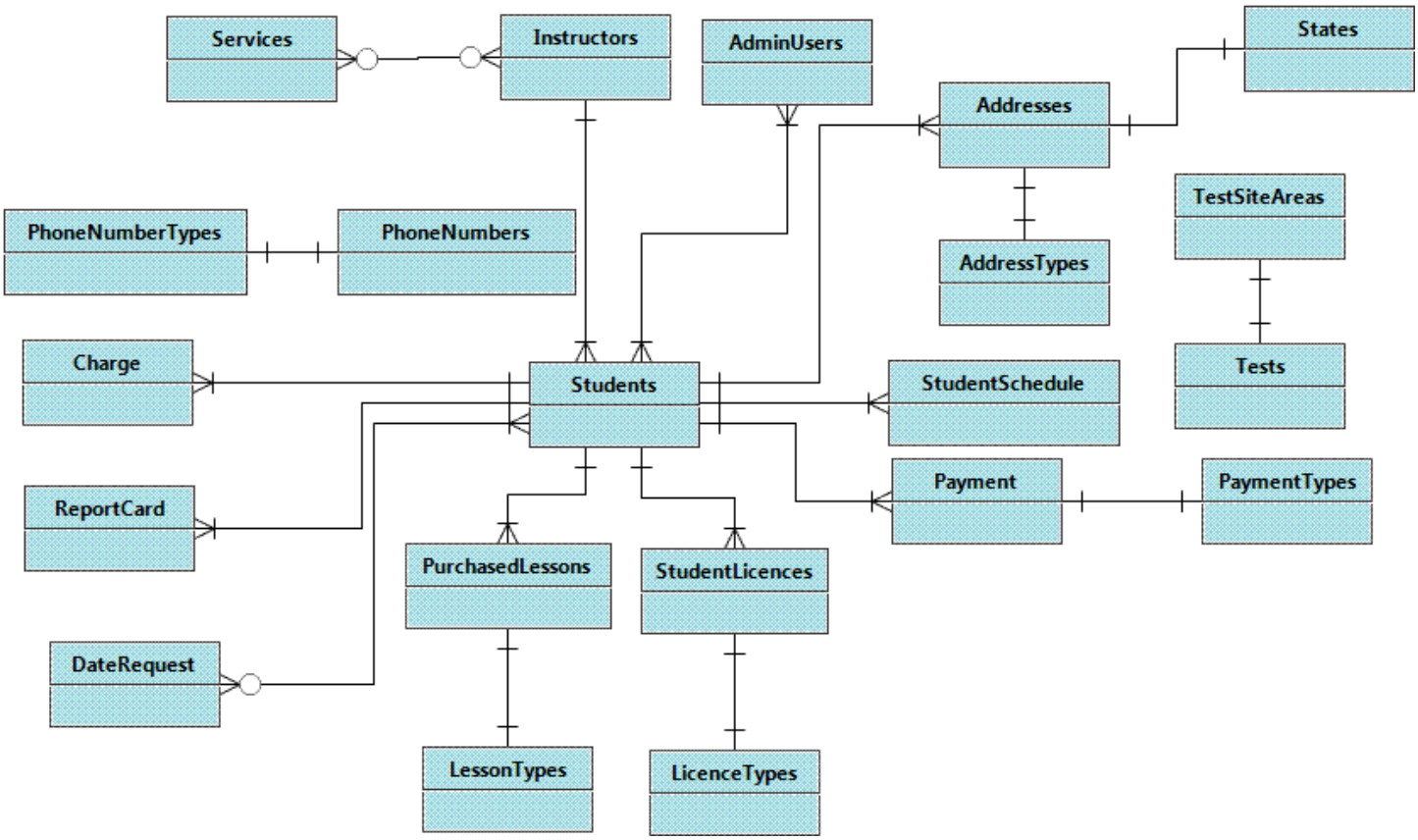
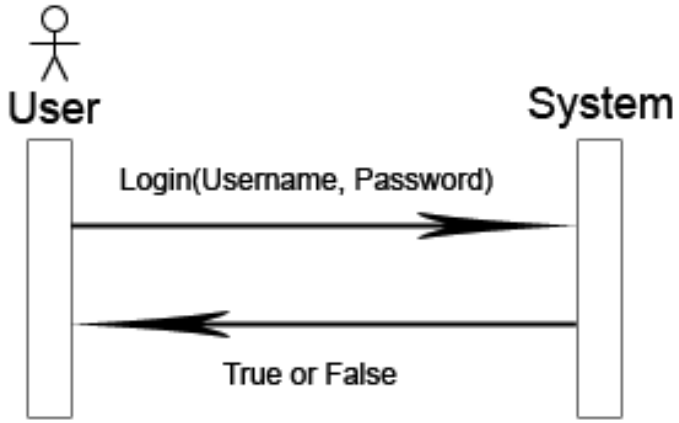


Figure 3: ER diagram depicting all tables that will be implemented in the Manhattan School of Driving system and their relationship with each other.

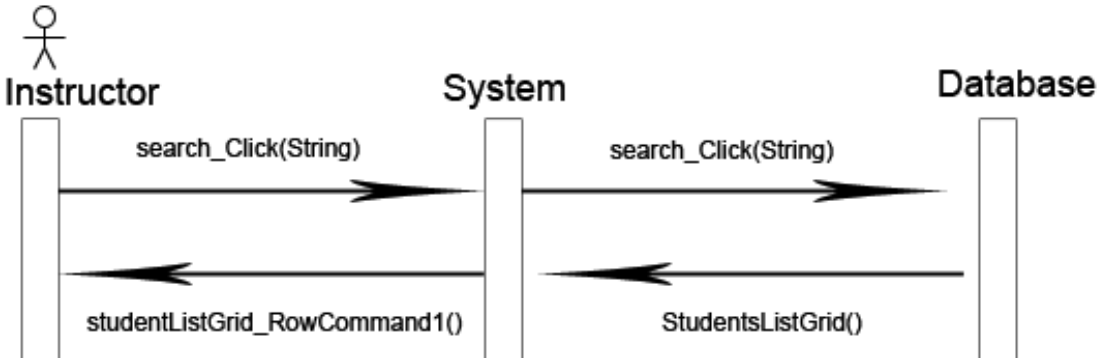
Sequence Diagram

User Login



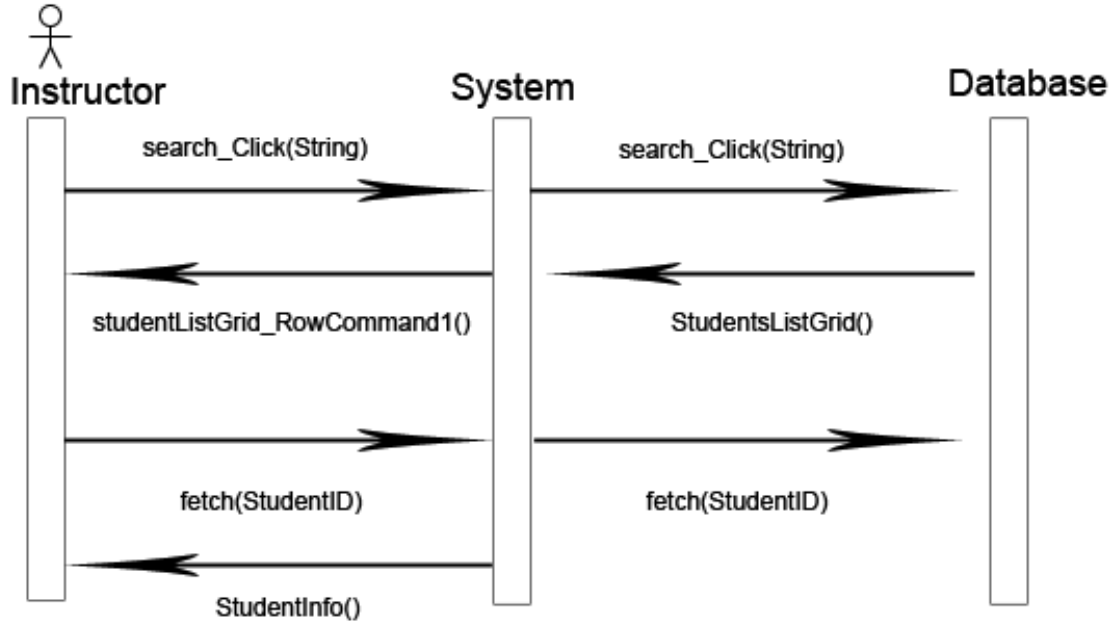
When the user logs in, their username/password will be passed to the system, the system will check to make sure it is valid and return a Boolean value, either granting or denying the user entry to the system

User Search



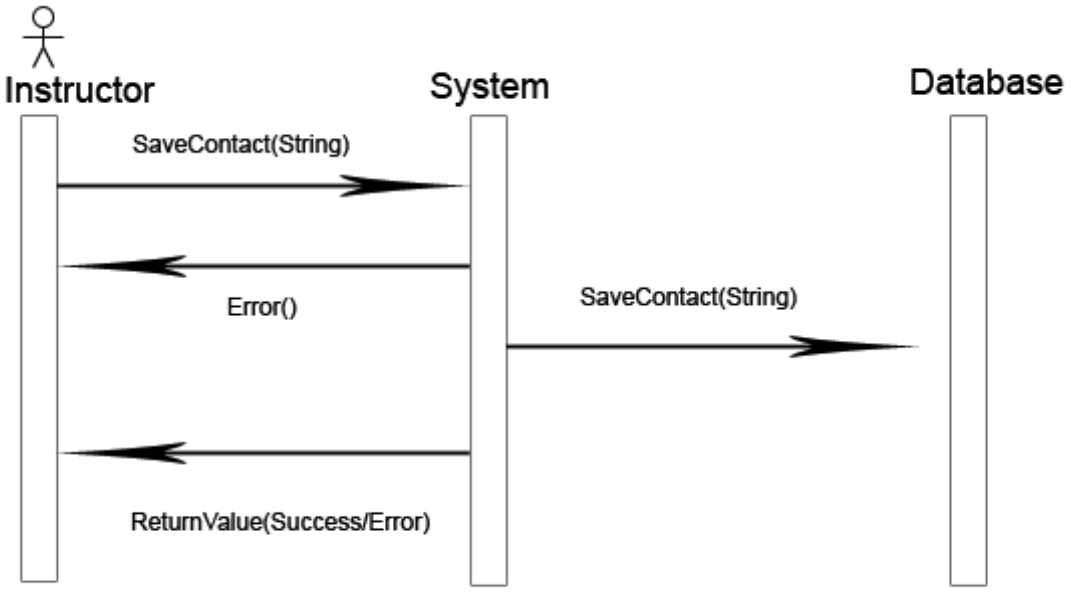
When the instructor searches for a student, they must fill out at least one of 3 fields (First name, Last name or Phone number). The user submits the information and the system passes it to the database. The database will then return a data list of all students that are generated from the query.

Student Info



When the instructor pulls a student profile, the method to search is similar to the search method. After the list of students is generated and the instructor selects the student, another request to the database will be made to pull the students information based on the submitted StudentID number. The information is then displayed on the Student.aspx page.

New Student



When the instructor creates a new student from the form, the information gets passed along to the server to compute a series of checks to make sure the information is valid and correct. If there is an error it will be returned to the instructor. If there is no error the new user will be inputted into the system and a confirmation will be returned to the instructor.

Software Architecture Document

Architectural Representation

The architecture of Manhattan School of Driving System will be described using technical memos and architectural views.

Technical Memo #1

Issue: Functionality

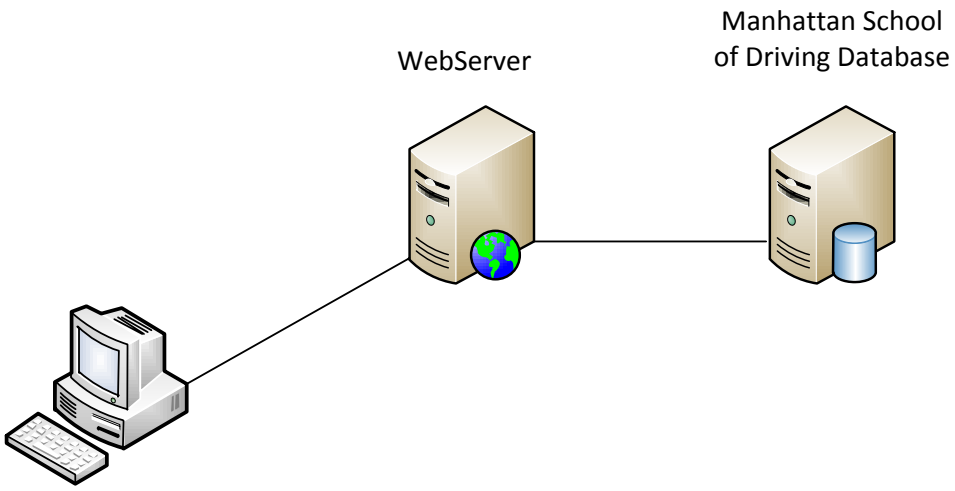
Solution Summary: Student records are saved in a database in order to provide permanence and accessibility.

Factors

- Manhattan School of Driving System must store all student records for the entire company. These records include current and past students.

Solution

All Student records are entered by an admin. When the admin has completed entering the student information and clicks "Save", the Student class through its sets of methods verifies that the data entered is correct before calling the Insert method which takes as arguments the student information that was entered. All data is then stored on the database and becomes accessible for future use.



Solution for Technical Memo #1

Technical Memo #2

Issue: Security

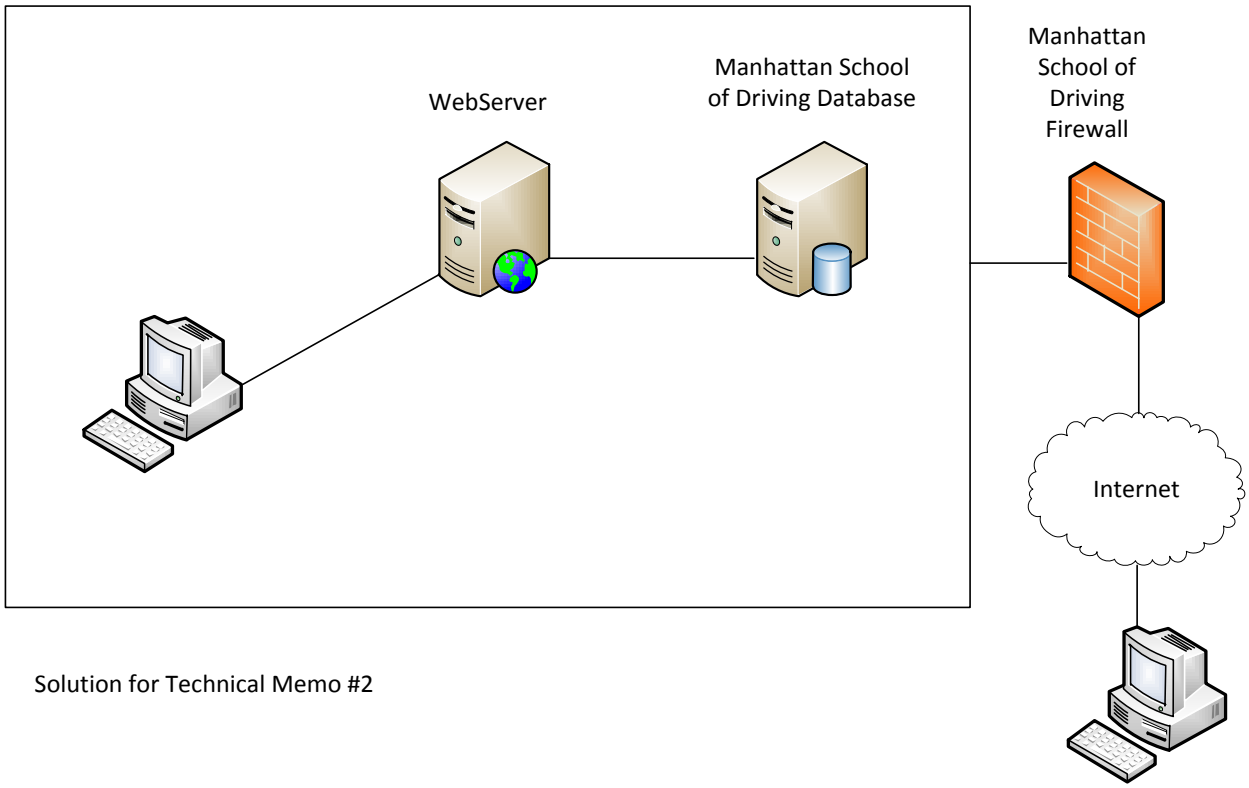
Solution Summary: Security is provided by a mandatory login, and through a firewall.

Factors

- Users of the Manhattan School of Driving must be shielded from those who would try to gain authorized access to the system.
- Users must be assured that their system passwords cannot be discovered by potential intruders.
- The system must be immune to SQL injections

Solution

Security is provided through a call to the database, to check the provided username and password against the database. If a match is found for the username, the password is validated against the matching password for the stored username. If no match is found for the username or if the password is not correct for the given username, the system should not allow the unauthorized intruder to log in to the system. If more than 3 unsuccessful attempts are made, the system should lock that username from any access to the system for duration of 24 hours unless a system administrator unlocks the account. To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, parameterized statements must be used (preferred), or user input must be carefully escaped or filtered.



Solution for Technical Memo #2

Technical Memo #3
Issue: Usability

Solution Summary: Ease of use and a comfortable web-based interface are provided.

Factors
- User must be comfortable working with the system.
- System must be easy to learn
- All major functionalities should be accessible in less than 3 clicks

Solution
The Manhattan School of Driving system is a web-based system that uses the same terminology that was previously used in their paper-based form. The interface is friendly, using colors that are eye pleasing and that represent the company colors. All major functionalities are 3 clicks away after login.

Technical Memo #4
Issue: Reliability and Recovery from Failure

Solution Summary: The Manhattan School of Driving system will be built to be fault tolerant, if for any given reason, there were to be a failure, redundancy will be provided as well as a list of available back up of its database.

Factors
- The Manhattan School of Driving System must always be accessible
- If the system fails, recovery must be made immediately

Solution
The Manhattan School of Driving System will provide a backup mechanism for its database that will be performed daily. At the end of the week a weekly backup will be stored. A total of four weekly backups will be stored in a month. Monthly, a disk checkup will be performed where the System Administrator may determine in advance if there is a faulty hardware disk currently in use. In order to have the system available at all times, the system will be connected to the internet. Redundancy could be implemented but is currently in negotiation and may be implemented in a future elaboration.

Test Models

Classes of Tests

- Admin login to the system
- Adding, editing and removing student profiles from the database system
- Searching for ***only current*** students based on the following criteria:
    - Student's first name
    - Student's last name
    - Student's phone number
- Searching for ***all*** students in the database based on the following criteria:
    - Student's first name

    - Student's last name
    - Student's phone number
- Displaying Student's record information with the following columns:
    - Student's name
    - Student's DOB
    - Student's Email
    - Student's Main Phone Number
    - Date of when student was added to the database
    - Student's status
- Validating student records before it's saved in the database
- Error Handling

Expected Software Response

- System allows an admin with a valid username and password to login to the system
- Admin is able to add, edit and remove students from database
- Admin is able to search for a student record based on any of the following information given:
    - Student's first name
    - Student's last name
    - Student's phone number
- Admin is able to view a list table of student records with the following information:
    - Student's name
    - Student's DOB
    - Student's Email
    - Student's Main Phone Number
    - Date of when student was added to the database
    - Student's status
- System prevents admin from adding two students with exact name, DOB and email address
- System prevents admin from adding a student with incorrect values for the requested fields
- System saves all data inputted on the system when a new user is added to the database

Performance Bounds

- Major functionalities should be accessible with a minimum number of clicks. Major functionalities for elaboration1 includes:
    - Student Search
    - Add New Student
    - Quick Student Add
- All major functionalities should be responsive when clicked by a user. The estimated time for the system website to load should be less than 2 seconds on DSL or Cable modem and 3 to 4 seconds on 56K modem. The proposed metrics were derived from the average time that it takes to load some popular social websites as well as some commercial websites, such as; Facebook.com (1.35 seconds), Craiglist.com (1.49 seconds), Yahoo.com (1.49 seconds) and LinkedIn.com (1.49 seconds).

Implementation Model: Source Code

The following class that is provided is the current Student class that we are implementing in our system. Currently the collecting, editing and storing of student data is the most important focus for this project. Below you can see the first implementation of a student in this system, It is able to create, update and delete basic student information.

Namespaces used:

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Reflection;
// <summary>
// Summary description for Student
// </summary>
```

Student.cs

```
public class Student
{
public string firstName;
public string lastName;
public string instructorName;
public string notes;
public string email;
public string year;
public int studentID;
public int age;
public Address MainAddress = new Address();
public Address PickUpAddress = new Address();
public List<PhoneNumber> PhoneNumberList = new List<PhoneNumber>();
public int instructorID;
public string gender;
public DateTime dateAdded;
public DateTime dob;
public bool currentStudent;
public bool deleted;
public bool resident;
private string stringType = "string";
private string intType = "int";
private string doubleType = "double";
```

```
private string boolType = "bool";
private string dateTimeType = "dateTime";
public Student(){
firstName = String.Empty;
lastName = String.Empty;
age = -1;
email = String.Empty;
notes = String.Empty;
year = "0000";
instructorID = 1;
gender = "";
currentStudent = true;
deleted = false;
resident = true;
dob = Convert.ToDateTime("01 /01/1800");
dateAdded = DateTime.Now;
}
public void fetch(int ID) {
studentID = ID;
SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["Magnum"].ToString());
using (conn) {
conn.Open();
// Get All General Student Info
GetGeneralInfo(conn);
// GET THE CURRENT BALANCE
GetStudentBalance(conn);
// Get the addresses
GetStudentAddress(conn);
// Get the phoneNumber
GetStudentNumbers(conn);
conn.Close();
}
}
public void insert() {
SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["Magnum"].ToString());
using (conn)
{
conn.Open();
// Insert Basic Info
SqlCommand cmd = new SqlCommand("Insert into studentInfo
(firstname,lastname,age,email,notes,year,instructorId,gender,dob,resident,dateAdded)" +
"values(@firstname,@lastname,@age,@email,@notes,@year,@instructorId,
@gender,@dob,@resident,@dateAdded)", conn);
cmd.Parameters.AddWithValue("@firstname",firstName);
cmd.Parameters.AddWithValue("@lastname",lastName);
cmd.Parameters.AddWithValue("@age",age);
cmd.Parameters.AddWithValue("@email",email);
cmd.Parameters.AddWithValue("@notes",notes);
cmd.Parameters.AddWithValue("@year",year);
cmd.Parameters.AddWithValue("@instructorId",instructorID);
cmd.Parameters.AddWithValue("@gender",gender);
cmd.Parameters.AddWithValue("@dob",dob);
cmd.Parameters.AddWithValue("@resident",resident);
cmd.Parameters.AddWithValue("@dateAdded",dateAdded);
```

```
cmd.ExecuteNonQuery();
SqlCommand cmdID = new SqlCommand("select top 1 studentID from studentInfo order by studentID
desc ;",
conn);
SqlDataReader reader = cmdID.ExecuteReader();
// Get ID
if (reader.HasRows)
{
reader.Read();
studentID = (int)SafeReader(reader["studentID"], intType);
}
if (!(MainAddress.line1 == ""))
{
if (!(MainAddress.line1 == null))
{
// insert Address
SqlCommand addr1cmd = new SqlCommand("Insert into addresses
(studentID,line1,line2,city,state,zip,addressType)
values(@studentID,@line1,@line2,@city,@state,@zip,@addressType)",conn);
addr1cmd.Parameters.AddWithValue("@studentID",studentID);
addr1cmd.Parameters.AddWithValue("@line1",MainAddress.line1);
addr1cmd.Parameters.AddWithValue("@line2",MainAddress.line2);
addr1cmd.Parameters.AddWithValue("@city",MainAddress.city);
addr1cmd.Parameters.AddWithValue("@state",MainAddress.state);
addr1cmd.Parameters.AddWithValue("@zip",MainAddress.zip);
addr1cmd.Parameters.AddWithValue("@addressType",MainAddress.type);
addr1cmd.ExecuteNonQuery();
}
}
if (!(PickUpAddress.line1 == ""))
{
if (!(PickUpAddress.line1 == null))
{
SqlCommand addr2cmd = new SqlCommand("Insert into addresses
(studentID,line1,line2,city,state,zip,addressType)
values(@studentID,@line1,@line2,@city,@state,@zip,@addressType)",
conn);
addr2cmd.Parameters.AddWithValue("@studentID",studentID);
addr2cmd.Parameters.AddWithValue("@line1",PickUpAddress.line1);
addr2cmd.Parameters.AddWithValue("@line2",PickUpAddress.line2);
addr2cmd.Parameters.AddWithValue("@city",PickUpAddress.city);
addr2cmd.Parameters.AddWithValue("@state",PickUpAddress.state);
addr2cmd.Parameters.AddWithValue("@zip",PickUpAddress.zip);
addr2cmd.Parameters.AddWithValue("@addressType",PickUpAddress.type);
addr2cmd.ExecuteNonQuery();
}
}
// insert Phone
foreach (PhoneNumber phone in PhoneNumberList)
{
SqlCommand phonecmd = new SqlCommand("Insert into phoneNumbers(studentId, phoneNumber,
phoneType,
isMain) values(@studentId, @phoneNumber, @phoneType, @isMain)", conn);
phonecmd.Parameters.AddWithValue("@studentID", studentID);
phonecmd.Parameters.AddWithValue("@phoneNumber", phone.number);
phonecmd.Parameters.AddWithValue("@phoneType", phone.type);
```

```
phonecmd.Parameters.AddWithValue("@isMain", phone.isMain);
phonecmd.ExecuteNonQuery();
}
conn.Close();
}
}
public void update() {
SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["Magnum"].ToString());
using (conn)
{
conn.Open();
SqlCommand cmd = new SqlCommand("Update studentInfo set firstname= @firstname, lastname=
@lastname,
year= @year," +
" instructorID= @instructorID, " +
" email= @email, notes= @notes , age= @age ,gender= @gender ,dob= @dob, " +
" currentStudent= @currentStudent, " +
" resident= @resident, dateAdded= @dateAdded where studentId=@id", conn);
cmd.Parameters.AddWithValue("@firstname", firstName);
cmd.Parameters.AddWithValue("@lastname", lastName);
cmd.Parameters.AddWithValue("@age", age);
cmd.Parameters.AddWithValue("@email", email);
cmd.Parameters.AddWithValue("@notes", notes);
cmd.Parameters.AddWithValue("@year", year);
cmd.Parameters.AddWithValue("@instructorId", instructorID);
cmd.Parameters.AddWithValue("@gender", gender);
cmd.Parameters.AddWithValue("@dob", dob);
cmd.Parameters.AddWithValue("@resident", resident);
cmd.Parameters.AddWithValue("@dateAdded", dateAdded);
cmd.ExecuteNonQuery();
SqlCommand cmd2 = new SqlCommand("Delete from addresses where studentid=@id", conn);
cmd2.Parameters.AddWithValue("@id", studentID);
cmd2.ExecuteNonQuery();
// insert Address
if (!(MainAddress.line1 == ""))
{
if (!(MainAddress.line1 == null))
{
SqlCommand addr1cmd = new SqlCommand("Insert into addresses
(studentID,line1,line2,city,state,zip,addressType)
values(@studentID,@line1,@line2,@city,@state,@zip,@addressType)",
conn);
addr1cmd.Parameters.AddWithValue("@studentID", studentID);
addr1cmd.Parameters.AddWithValue("@line1", MainAddress.line1);
addr1cmd.Parameters.AddWithValue("@line2", MainAddress.line2);
addr1cmd.Parameters.AddWithValue("@city", MainAddress.city);
addr1cmd.Parameters.AddWithValue("@state", MainAddress.state);
addr1cmd.Parameters.AddWithValue("@zip", MainAddress.zip);
addr1cmd.Parameters.AddWithValue("@addressType", MainAddress.type);
addr1cmd.ExecuteNonQuery();
}
}
if (!(PickUpAddress.line1 == "")){
if (!(PickUpAddress.line1 == null))
{
```

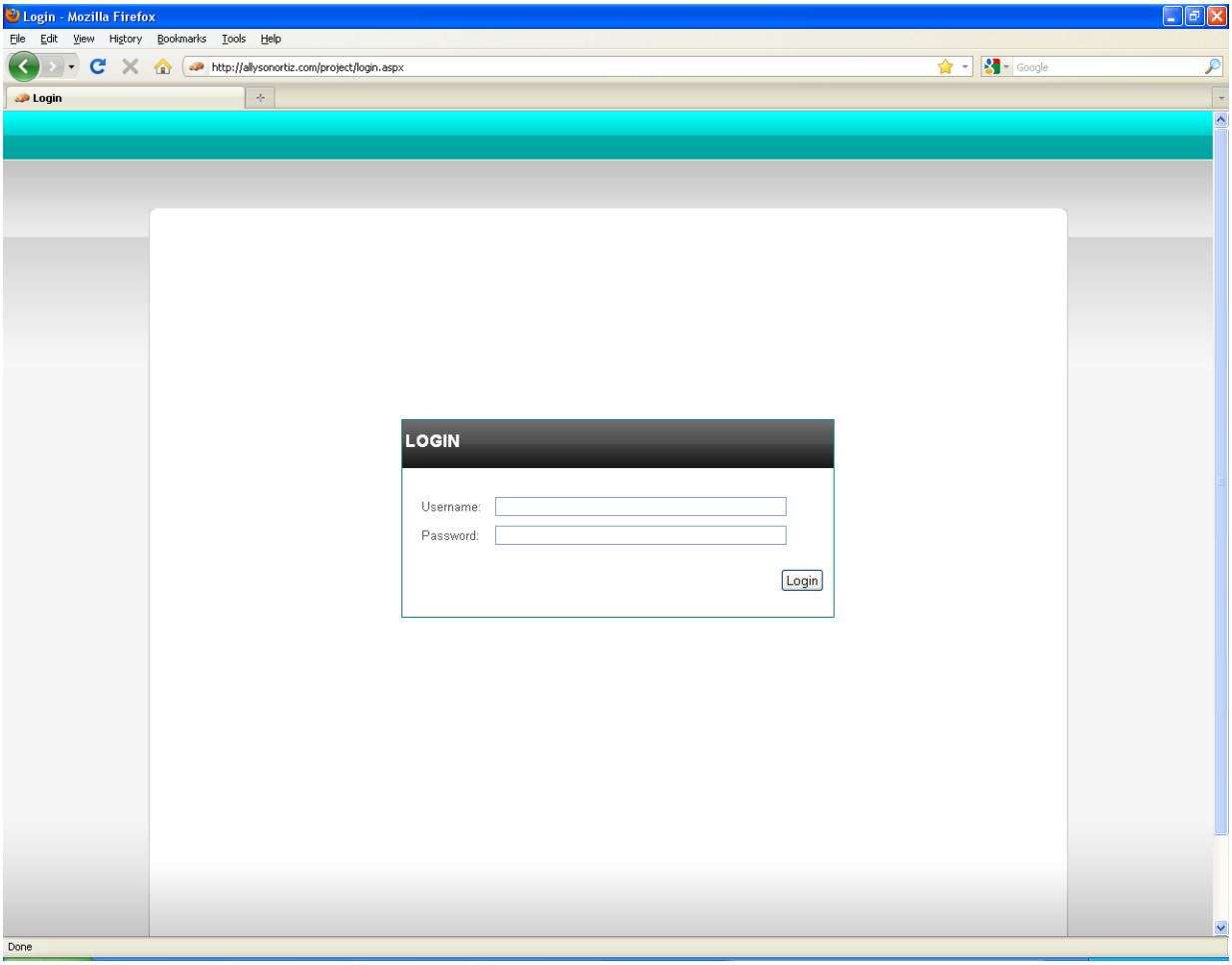
```
SqlCommand addr2cmd = new SqlCommand("Insert into addresses
(studentID,line1,line2,city,state,zip,addressType)
values(@studentID,@line1,@line2,@city,@state,@zip,@addressType)",
conn);
addr2cmd.Parameters.AddWithValue("@studentID", studentID);
addr2cmd.Parameters.AddWithValue("@line1", PickUpAddress.line1);
addr2cmd.Parameters.AddWithValue("@line2", PickUpAddress.line2);
addr2cmd.Parameters.AddWithValue("@city", PickUpAddress.city);
addr2cmd.Parameters.AddWithValue("@state", PickUpAddress.state);
addr2cmd.Parameters.AddWithValue("@zip", PickUpAddress.zip);
addr2cmd.Parameters.AddWithValue("@addressType", PickUpAddress.type);
addr2cmd.ExecuteNonQuery();
}
}
SqlCommand cmd3 = new SqlCommand("Delete from phoneNumbers where studentid=@id", conn);
cmd3.Parameters.AddWithValue("@id", studentID);
cmd3.ExecuteNonQuery();
// insert Phone
foreach (PhoneNumber phone in PhoneNumberList)
{
if (phone.isMain.ToLower() == "true" || phone.isMain == "1")
{
phone.isMain = "1";
}
else
{
phone.isMain = "0";
}
SqlCommand phonecmd = new SqlCommand("Insert into phoneNumbers(studentId, phoneNumber,
phoneType,
isMain) values(@studentId, @phoneNumber, @phoneType, @isMain)", conn);
phonecmd.Parameters.AddWithValue("@studentID", studentID);
phonecmd.Parameters.AddWithValue("@phoneNumber", phone.number);
phonecmd.Parameters.AddWithValue("@phoneType", phone.type);
phonecmd.Parameters.AddWithValue("@isMain", phone.isMain);
phonecmd.ExecuteNonQuery();
}
conn.Close();
}
}
public void delete() {
SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["Magnum"].ToString());
using (conn)
{
conn.Open();
SqlCommand cmd = new SqlCommand("Update studentInfo set deleted=1 where studentId=@id", conn);
cmd.Parameters.AddWithValue("@id", studentID);
cmd.ExecuteScalar();
conn.Close();
}
}
private void GetGeneralInfo(SqlConnection conn){
SqlCommand cmd = new SqlCommand("select * from studentInfo left join instructors on
instructors.instructorID =
studentInfo.instructorID where studentId=@id", conn);
```

```
cmd.Parameters.AddWithValue("@id", studentID);
SqlDataReader reader = cmd.ExecuteReader();
if (reader.HasRows)
{
reader.Read();
firstName = SafeReader(reader["firstname"], stringType).ToString();
lastName = SafeReader(reader["lastname"], stringType).ToString();
instructorName = SafeReader(reader["instructorName"], stringType).ToString();
year = SafeReader(reader["year"], stringType).ToString();
notes = SafeReader(reader["notes"], stringType).ToString();
email = SafeReader(reader["email"], stringType).ToString();
age = Convert.ToInt32(SafeReader((reader["age"]), intType));
instructorID = (int)SafeReader(reader["instructorID"], intType);
gender = SafeReader(reader["gender"], stringType).ToString();
dateAdded = Convert.ToDateTime(SafeReader(reader["dateAdded"], dateTimeType));
dob = Convert.ToDateTime(SafeReader(reader["dob"], dateTimeType));
currentStudent = (bool)SafeReader(reader["currentStudent"], boolType);
resident = (bool)SafeReader(reader["resident"], boolType);
}
}
private void GetStudentAddress(SqlConnection conn)
{
SqlCommand cmdAddr = new SqlCommand("select * from addresses inner join states on states.stateid =
addresses.state inner join addressTypes on addressTypes.addressTypeID = addresses.addressType
where studentId=@id",
conn);
cmdAddr.Parameters.AddWithValue("@id", studentID);
SqlDataReader reader = cmdAddr.ExecuteReader();
while (reader.Read())
{
Address addr = new Address();
addr.line1 = SafeReader(reader["line1"], stringType).ToString();
addr.line2 = SafeReader(reader["line2"],stringType).ToString();
addr.city = SafeReader(reader["city"], stringType).ToString();
addr.state = SafeReader(reader["state"], stringType).ToString();
addr.zip = SafeReader(reader["zip"], stringType).ToString();
addr.type = SafeReader(reader["addressType"], stringType).ToString();
addr.id = (int)reader["id"];
addr.stateName = SafeReader(reader["stateName"], stringType).ToString();
if (addr.type.Equals("2"))
{
PickUpAddress = addr;
}
else
{
MainAddress = addr;
}
}
}
private void GetStudentNumbers(SqlConnection conn)
{
SqlCommand cmdPhone = new SqlCommand("select * from phoneNumbers inner join
phoneNumberTypes on
phoneNumberTypes.phoneNumberTypeID = phoneNumbers.phoneType where studentId=@id", conn);
cmdPhone.Parameters.AddWithValue("@id", studentID);
SqlDataReader reader = cmdPhone.ExecuteReader();
```

```
while (reader.Read())
{
PhoneNumber phone = new PhoneNumber();
phone.number = SafeReader( reader["phoneNumber"],stringType).ToString();
phone.type = (int)SafeReader(reader["phoneType"], intType);
phone.typeName = SafeReader(reader["phoneNumberTypeName"], stringType).ToString();
phone.id = (int)SafeReader(reader["id"],intType);
phone.isMain = SafeReader(reader["isMain"], stringType).ToString();
PhoneNumberList.Add(phone);
}
}
public object SafeReader(object readerValue, String type)
{
if (readerValue == DBNull.Value)
{
switch (type)
{
case "string":
readerValue = String.Empty;
break;
case "int":
readerValue = 0;
break;
case "double":
readerValue = 0.0;
break;
case "dateTime":
readerValue = "1 / 1 / 1800";
break;
case "bool":
readerValue = false;
break;
}
}
return readerValue;
}}
```

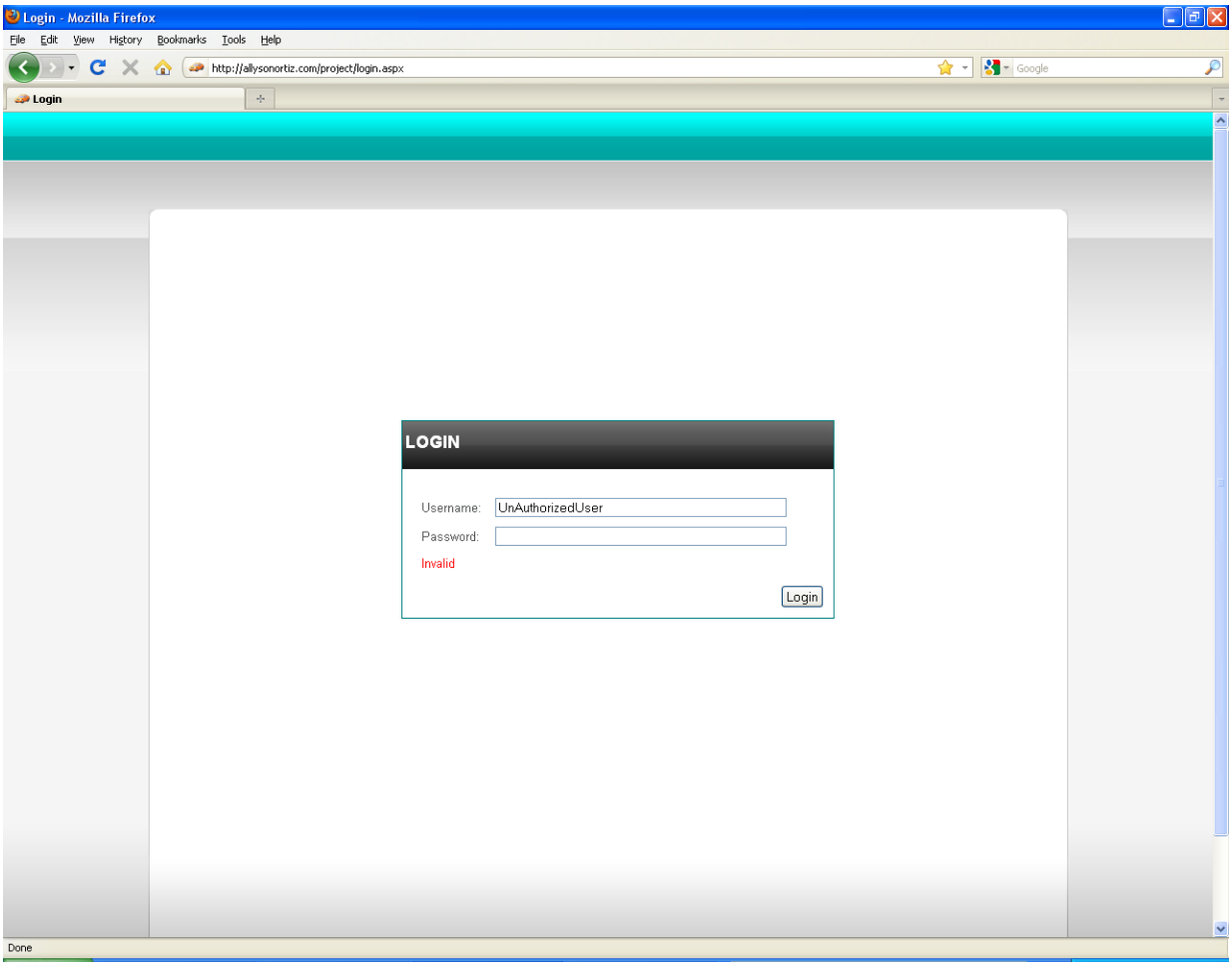
User Interface

Login Page



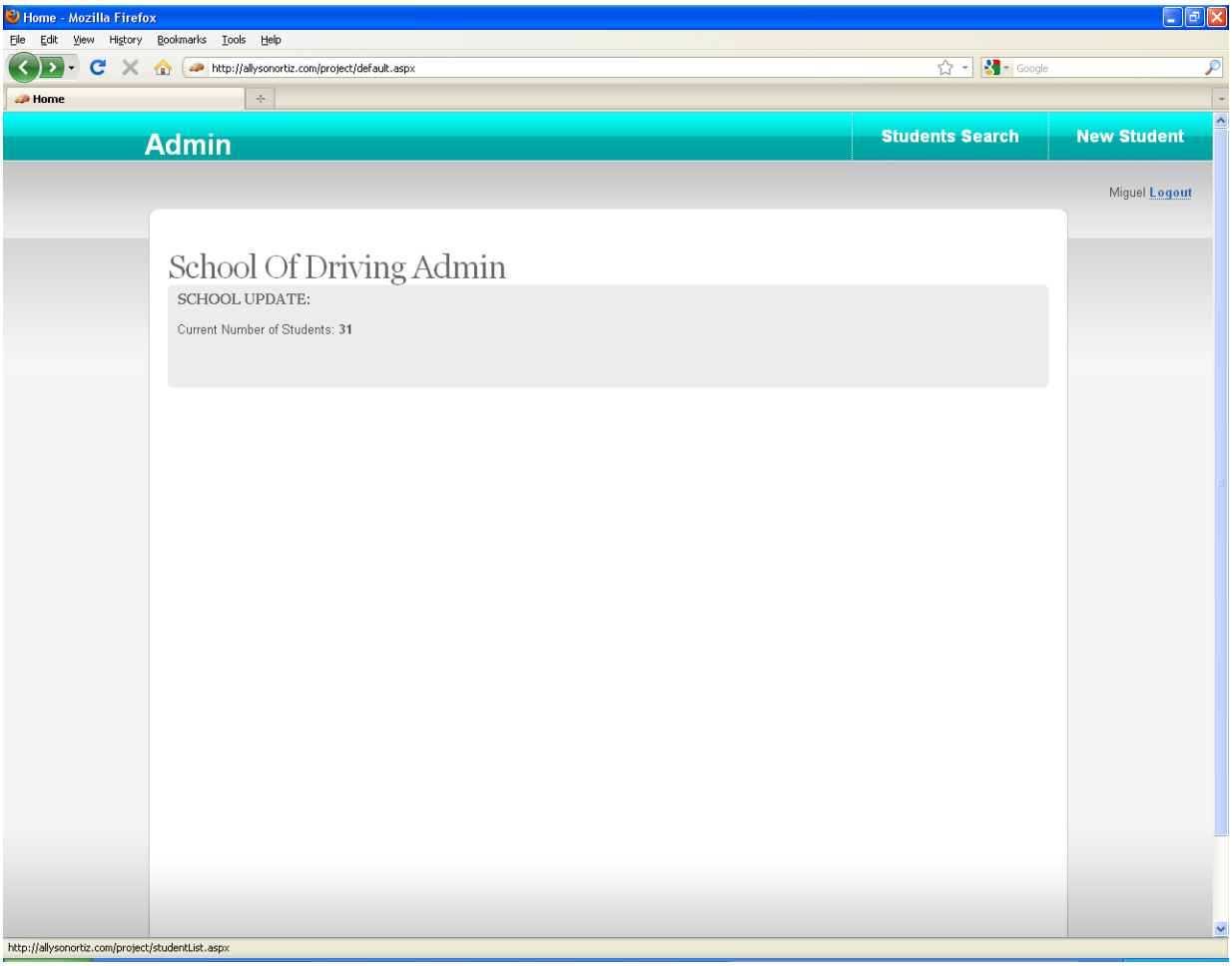
The Main Login page is the entry point to the Manhattan School of Driving system. For the first elaboration, the main goal of the developers was to get a system up and running with little or no aesthetics enhancements. Based on the customer feedback, future logos, change of color or any other aesthetics changes may occur.

Unsuccessful Login



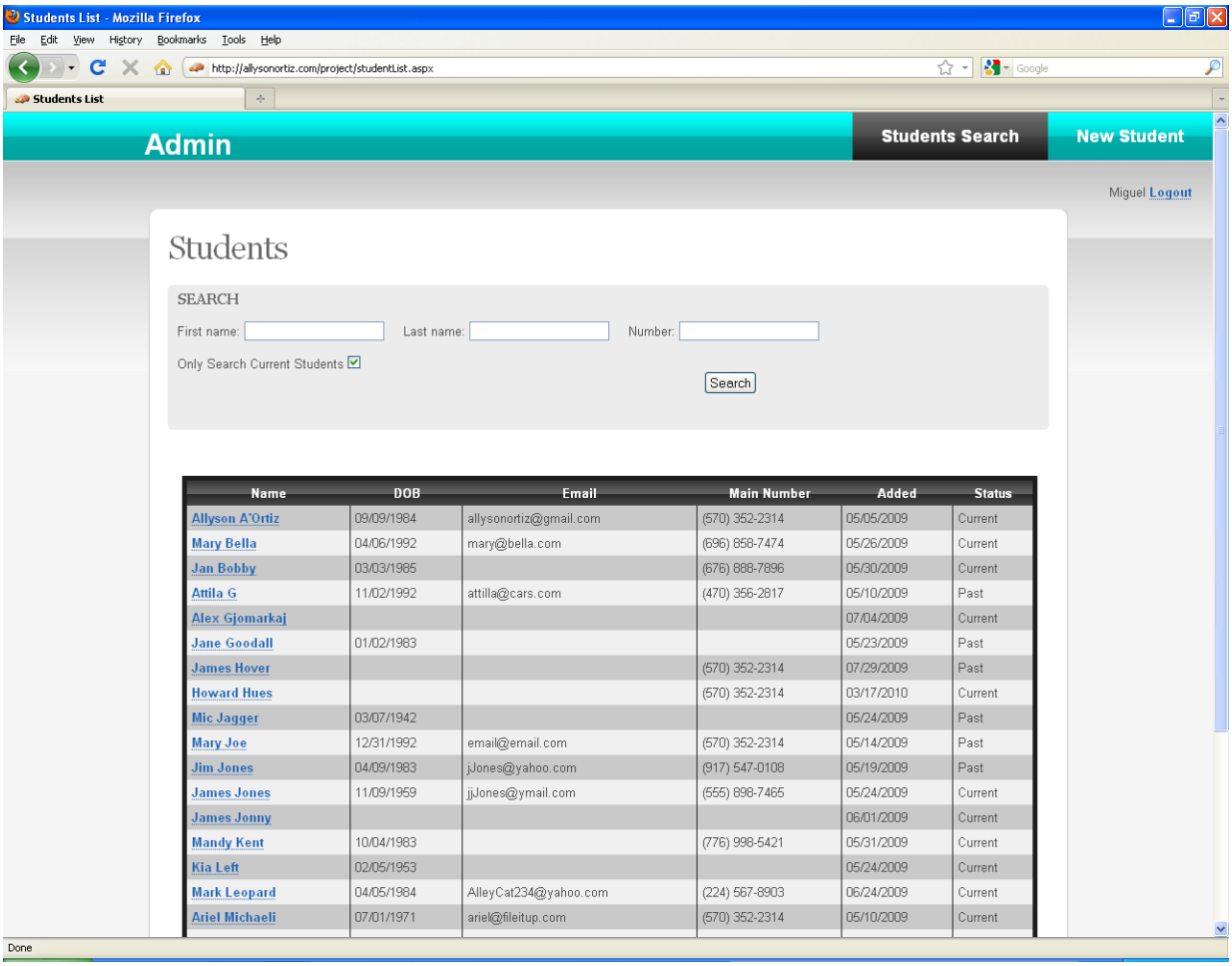
For any unauthorized user who attempts to login, or for any user who erroneously types in their wrong username or password, the system will display an error message.

Admin Page



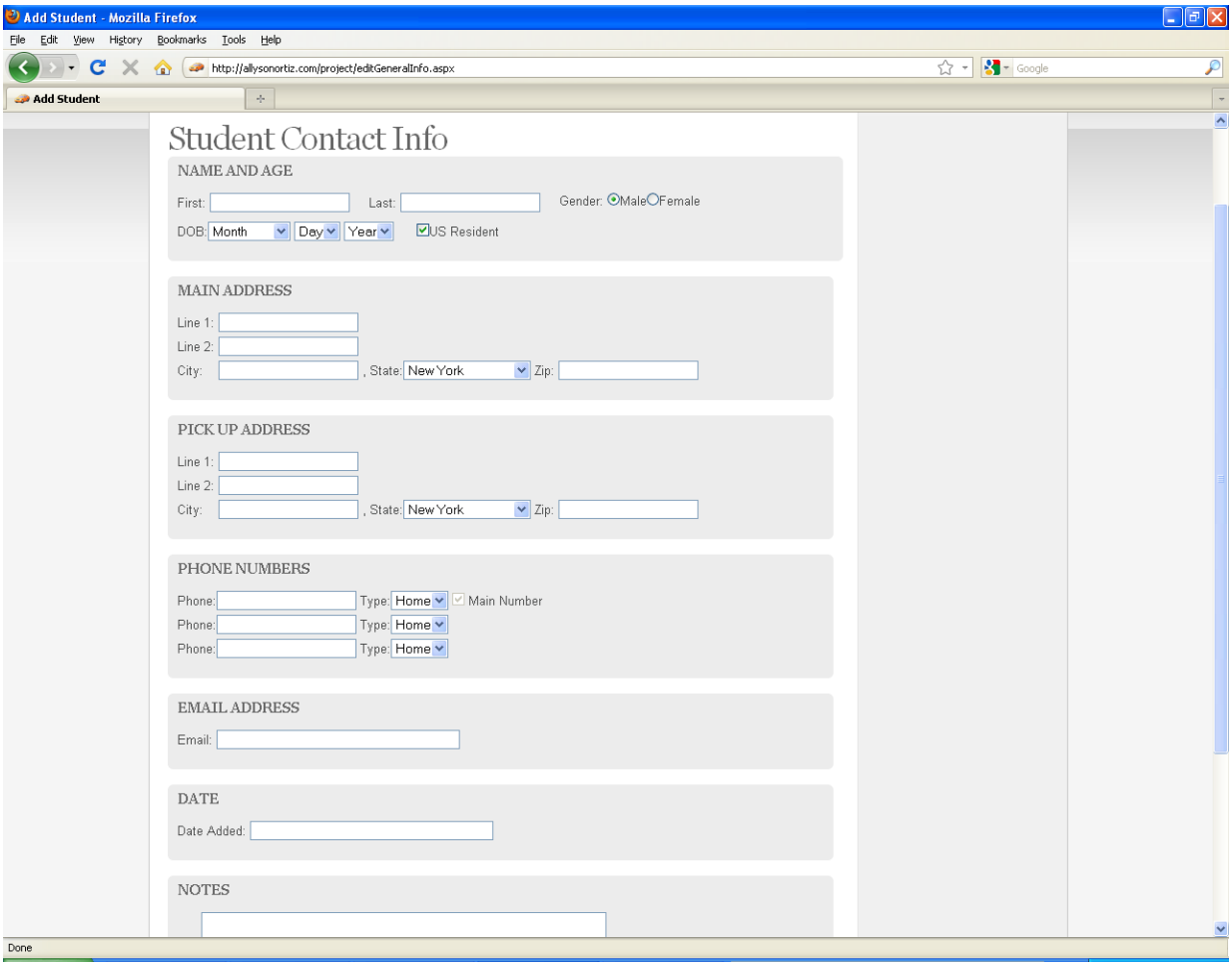
The admin page has the major functions that are required for any administrator to perform. On the admin page header, there are two tabs: “Students Search” and “New Student”.

Students Search



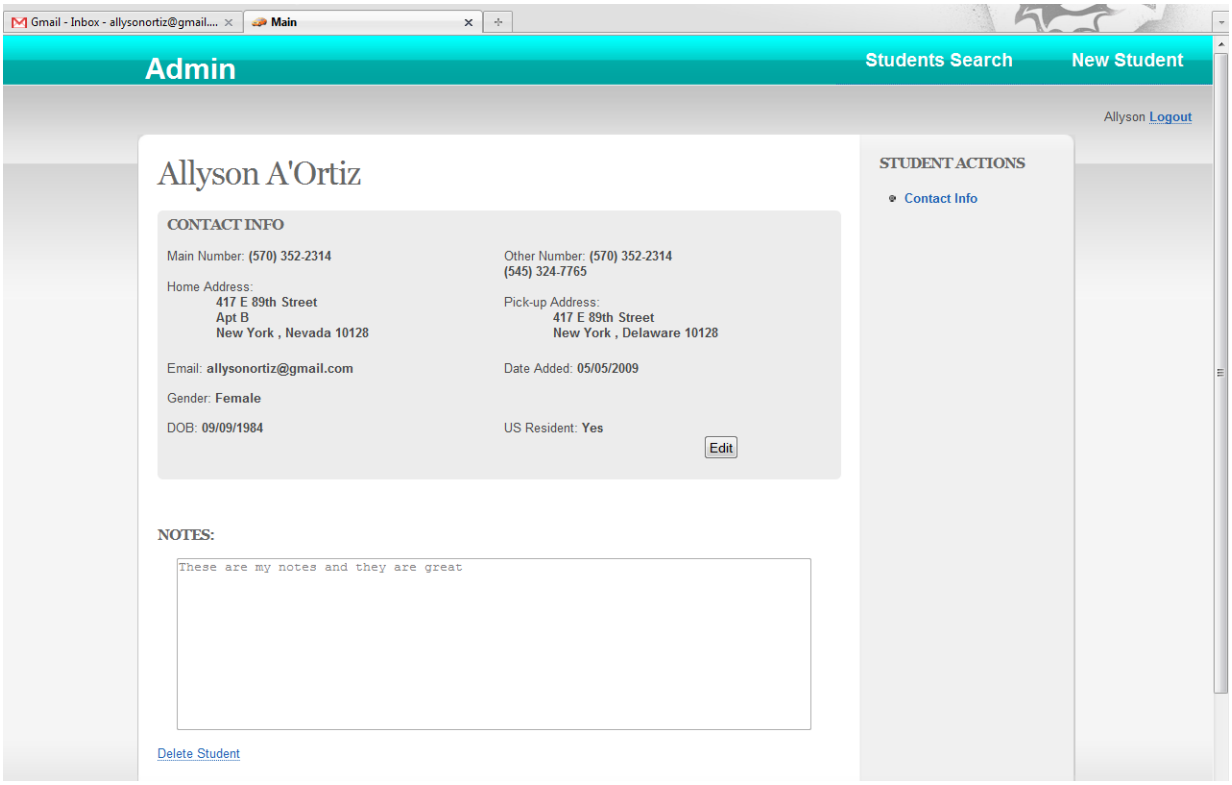
The Students Search page is the main search engine for student look-ups. Searches can be made by using a Student's first name, last name or student phone number. The returned results include a table that consists of zero or many rows (students) with the following columns: Name, DOB, Email, Main Number, Added, and Status.

New Student



The New Student page, allows an admin to input all the information necessary for a new student registering with the school.

View General Student Info



The Student Contact Info page, allows an admin to View information necessary for a students.

Supplementary Specifications

Introduction

This document will serve as a repository for all requirements not captured in the use cases.

Functionality

Logging and Error Handling

The management system should log the following events:

- Failed log-in attempts
- Failed data-backups

Security

The usage of the management system should be available to receptionist, trainers and students who have active usernames and passwords. There is no alternate access to the management system without proper system authentication using the system's main website. This includes no backdoor access from developers to any user (authorized or unauthorized).

Usability

Human Factors

The Manhattan School of Driving Management System will be used in an office environment by employees (receptionist and trainers) and students. The system will also be available outside the office environment. Since this is a web based application, students and employees can access the system online by typing the system URL link on their web browser.

The target audience is employees and students which may vary in age from 16 to 84. Since this web based application will be used by many different people with different ages, the system must be user friendly and easy to learn. The term "User Friendly" can be very vague and generic, so a formal definition can consist of the following:

- Adequate Text Size – The text size used by the system should be large enough to read from arm's length.
- Descriptive Menus – All menus must have a descriptive name that is useful in indicate its purpose. For example, Log-off, would obviously indicate that you would like to terminate your session and Print, would indicate that you would like to print the page currently in view.
- Pleasing Colors – Some colors may be hard for certain people to see. For example, a bright red or pink might not be pleasant to someone's eyes while a color that is too light might be hard for someone to notice.

- Tool-Tips – certain forms may require a “tip” to indicate how the data should be entered on the system or what the purpose is for a particular button or field box. Where ever it might seem required, a tool-tip may be placed to help the user navigate and use the system.

Reliability

The Manhattan School of Driving Management System must behave in a reliable manner in its working environment. A “reliable manner” is a specified level of performance that is determined only by specified conditions. These specified conditions will be mentioned in further detail during the elaboration phase. For now we will consider the following characteristic:

- Maturity – the system should avoid failure as a result of faults in the software by being in a state of being fully developed
- Fault Tolerance – the system should be capable of maintaining a specified level of performance in cases of software faults or of infringement of its specified interface
- Recoverability – the capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure
- Compliance – the capability of the software product to adhere to standards, conventions or regulations relating to reliability

Efficiency

The system should be capable of providing appropriate performance, relative to the amount of resources used, under stated conditions. The specified conditions will be mentioned in further detail

- Time Behavior – the system should be capable of providing appropriate response and processing times and throughput rates when performing its function, under stated conditions
- Resource Utilization – the system should use an appropriate amount of resources when the system performs its function under stated conditions
- Compliance – the system should adhere to standards or conventions relating to efficiency

Maintainability

The system should be modifiable. Modifications may include corrections, improvements or adaptation of the system to changes in environment, and in requirements and functional specifications. The following characteristics will define if a system is maintainable:

- Changeability – the system should easily enable a specified modification to be implemented
- Stability – the system should be able to avoid unexpected effects from modifications of the software
- Testability – the system should easily enable modified software to be validated
- Compliance – the system should adhere to standards or conventions relating to maintainability

Portability

The system should be developed in a way that it is easy for someone to transfer it from one environment to another. The following characteristics will define if a system is portable:

- Adaptability – the system should be able to adapt to different specified environments without applying actions or means other than those provided for this purpose for the system considered
- Install ability – the system should be capable of being installed in a specified environment
- Co-existence – the system should co-exist with other independent software in a common environment sharing common resources
- Replace ability – the system should be used in place of another specified software product for the same purpose in the same environment
- Compliance – the system should adhere to standards or conventions relating to portability

Glossary

Term	Definition	Aliases
Management System	is a collection of procedures used to manage work flow in a collaborative environment	
DMV	Department of Motor Vehicle	
Backdoor	is a method of bypassing normal authentication, securing remote access to a computer, obtaining access to plaintext, and so on, while attempting to remain undetected	
Function	is a structured representation of the functions, activities or processes	Action, Event
Environment	state which provides software services for processes or programs while a computer is running	infrastructure