|  |  |
| --- | --- |
| \\cetus.ece.missouri.edu\users\fischerjd\Desktop\Current Semester\common\logo-clear.png | University of Missouri – Columbia  Department of Electrical Engineering and Computer Science  *“A Proud Tradition, Engineering the Future!”* |

Railroad Car System

CS 3380 Group Project  
Group Report

April 24, 2017

— Engineering Team —

Michael Brown

Christopher Whestel

Akshay Chavakula

Timothy Shea

Joseph Crowe

— Course Instructor —

Dr. Joe Guilliams

Table of Contents

[README 1](#_Toc409448156)

[Abstract i](#_Toc409448157)

[Chapter 1.  Introduction 1](#_Toc409448158)

[Report Scope and Purpose 1](#_Toc409448159)

[Problem Statement 1](#_Toc409448160)

[Project Concept 1](#_Toc409448161)

[Background 1](#_Toc409448162)

[Report Overview 1](#_Toc409448163)

[Chapter 2.  Implementation 1](#_Toc409448164)

[Implementations Considered 1](#_Toc409448165)

[Proposed Implementation 1](#_Toc409448166)

[Theory of Operation 1](#_Toc409448167)

[System Operation (Heading 3) 1](#_Toc409448168)

[Set “<set\_name>” (Heading 3) 1](#_Toc409448169)

[Unit 1 “<unit1\_name>” (Heading 3) 1](#_Toc409448170)

[Assembly 1A1 “<assy\_1a1\_name>” Operation (Heading 4) 1](#_Toc409448171)

[Assembly 1A1 Software Architecture (Heading 4) 1](#_Toc409448172)

[Assembly 1A2 “<assy\_1a2\_name>” Operation (Heading 4) 1](#_Toc409448173)

[Subassembly 1A2A1 “<assy\_1a2a1\_name>” Operation (Heading 4) 1](#_Toc409448174)

[Unit 1 Software Architecture (assuming Unit 1 has software) (Heading 3) 1](#_Toc409448175)

[<software\_element1> (Heading 4) 1](#_Toc409448176)

[<software\_element2> (Heading 4) 1](#_Toc409448177)

[Unit 2 “<unit2\_name>” Operation (Heading 3) 1](#_Toc409448178)

[Contingency Plan 1](#_Toc409448179)

[Delivered Implementation 1](#_Toc409448180)

[Theory of Operation 1](#_Toc409448181)

[Safety Information 1](#_Toc409448182)

[Work Assignments 1](#_Toc409448183)

[Chapter 3.  Project Scope 1](#_Toc409448184)

[Specifications 1](#_Toc409448185)

[System Specifications 1](#_Toc409448186)

[Set Specifications 1](#_Toc409448187)

[Group Specifications 1](#_Toc409448188)

[Unit Specifications 1](#_Toc409448189)

[Assembly Specifications 1](#_Toc409448190)

[Interface Specifications 1](#_Toc409448191)

[Software Specifications 1](#_Toc409448192)

[Constraints 1](#_Toc409448193)

[Goals and Objectives 1](#_Toc409448194)

[Customer-centric Goal and Objectives 1](#_Toc409448195)

[<ENGINEERING\_GOAL1> Goal and Objectives 1](#_Toc409448196)

[<ENGINEERING\_GOAL2> Goal and Objectives 1](#_Toc409448197)

[<ENGINEERING\_GOAL3> Goal and Objectives 1](#_Toc409448198)

[<ENGINEERING\_GOAL4> Goal and Objectives 1](#_Toc409448199)

[Assumptions 1](#_Toc409448200)

[Chapter 4.  Evaluation 1](#_Toc409448201)

[Project Outcome 1](#_Toc409448202)

[Goals and Objectives Evaluation 1](#_Toc409448203)

[Customer-centric SMART Goal and Objectives 1](#_Toc409448204)

[<ENGINEERING\_GOAL1> SMART Goal and Objectives 1](#_Toc409448205)

[<ENGINEERING\_GOAL2> SMART Goal and Objectives 1](#_Toc409448206)

[<ENGINEERING\_GOAL3> SMART Goal and Objectives 1](#_Toc409448207)

[<ENGINEERING\_GOAL4> SMART Goal and Objectives 1](#_Toc409448208)

[Timeline Evaluation 1](#_Toc409448209)

[Proposed Timeline 1](#_Toc409448210)

[Actual Timeline 1](#_Toc409448211)

[Budget Evaluation 1](#_Toc409448212)

[Proposed Budget 1](#_Toc409448213)

[Actual Budget 1](#_Toc409448214)

[Chapter 5.  Conclusions 1](#_Toc409448215)

[Successes 1](#_Toc409448216)

[Limitations 1](#_Toc409448217)

1. Introduction

## Report Scope and Purpose

The railroad car system is a website developed to provide access to a complete system created using HTML, PHP, and a Mysql database. This system will provide full functionality to users and employees to complete various tasks such as, a customer renting equipment, administrators updating employee’s information, and a logging system to keep all information stored. We used this website development to give us a method to connect all concepts of the course, we utilize our knowledge of entity relationship diagrams to create a model of our system, then utilizing information provided throughout most labs we created a database, and connected our HTML and PHP to the website which can be viewed [here](http://cs3380.rnet.missouri.edu/~GROUP1/home/index.php). To access the website, a table has been provided in Appendix A. Overall, this project utilized the knowledge provided in the course to develop a fully functional and dynamic website and database through the use of HTML, PHP, and Mysql. We utilized HTML to retrieve information from the user, such as login information, PHP to translate that information into statements that Mysql can understand, and then used the Mysql database to store the information for future use. Our final design was very successful, allowing for a fully functional website that could be used as a starting system. It allows room for substantial improvement, but meets all the basic requirements for a new system.

1. Task Delegation

To delegate tasks appropriately, during our first two meeting we had everyone list what they were comfortable doing, what they understood, and what they we not comfortable with. We then went through the project description and listed out all of the tasks necessary to complete, after listing these we split them between those who felt more comfortable with each task. This was outlined through project milestones. Of course, as with every project, we all worked together in the end to ensure a completed and well developed system, to do this we shared knowledge with others so for future attempts they will help build a stronger plan of execution without additional assistance. Below, in table 2.1, we identify all aspects of the project and who is assigned to which task.

Table 2.1 – Group member assignments: software elements

|  |  |  |
| --- | --- | --- |
| Software Element | Software Element | Assigned To |
| 1A1U1 | User Authentication | Chris Whetsel, Akshay Chavakula, Michael Brown |
| 1A1U2 | Website Styling | Michael Brown & Emily Temple |
| 1A1U3 | Login Page | Michael Brown |
| 1A2U1 | Administration | Akshay Chavakula |
| 1A3U1 | Engineer | Chris Whetsel |
| 1A3U2 | Conductor | Chris Whetsel |
| 1A4U1 | Customer | Joseph Crowe |
| 1A5U1 | Logging | Timothy Shea |
| 1A6U1 | Mysql Table Creation | Group Effort |
| 1A6U2 | Mysql Insert Statements | Michael Brown |

1. Data Model Diagram

### C:\Users\mhb44\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ERD_G01_ver2.png

Figure 1: Group 1 ERD

1. Implementation

**Admin Search, Insert, Update, and Delete Trains**

**Authored By**: Akshay Chavakula

**Files Discussed:** adminSearchTrain.php, adminInsertTrain.php, adminUpdateTrain.php, adminHandleUpdate.php, adminDeleteTrain.php

One of the requirements for the admins is they should be able to search all the trains in the database, insert trains into the database and update/delete trains from the database. The solution is as follows: create a html form that has a search box and 3 radio buttons as the criteria that the admin can search the table. In this case, the criteria to search the trains table is train number (primary key), destination and start location. Each page required checkLogin.php to start a session and the session variable “id” was set to make sure employee was logged in. Then the page checked if the correct type of employee (admin) was accessing the page. If admin is not accessing the page then it redirects to index.php. The php code checks if the submit button is clicked and if it is then it connects to the database using a mysqli object. Once it is connected to the database, use if/else statements to see what criteria (train number, destination, start location) is selected. If train number is selected then the sql statement would be “SELECT \* FROM `trains` WHERE `trainNumber` LIKE ?”. The question mark represents prepared statements being used to clean the user input. Similarly, the sql statement if destination is selected is “SELECT \* FROM `trains` WHERE `destination` LIKE ?”. If start location is selected, sql = “SELECT \* FROM `trains` WHERE `startLocation` LIKE ?”. Then we dynamically print the table to the interface using echo. The columns are dynamically loaded using the mysqli\_fetch\_field() function. The rows are also dynamically loaded using the fetch\_array() function. Each row in the table printed on the interface has two buttons: update and delete. If the update button is clicked, it will go to a different php script called “**adminUpdateTrain.php**”. This php script contains a form with all the columns from the train table besides the primary key (train number) to be editable by the administrator. The form is also pre filled with the data from the row that is selected using hidden input types. The primary key which is train number in this case cannot be changed so it will be “readonly” in the form. The action for this form is called “**adminHandleUpdate.php**”. “**adminHandleUpdate.php**” checks if the update button is clicked and connects to the database using mysqli object. This page requires **core/log.php** to enter into the log every time an admin updates a train. The query for the update is “UPDATE `trains` SET `destination`=?, `startLocation`=?, `days`=?, `departureTime`=?, `arrivalTime`=? WHERE `trainNumber`=?”. This changes the values for all the columns in the train table besides the train number for the particular row selected based on the values from the update form. The addLog() function is called and employee id, description and action type are passed to this function so that it can be entered into the log. If the update is successful then the program prints “Train has been updated” to the screen using echo and the page redirects to the search train pages so admin can search the table to see if the update is reflected in the database.

When the delete button is clicked, the action on the form for delete is “**adminDeleteTrain.php**”. **adminDeleteTrain.php** requires **core/log.php** so that the addLog() function can be called. In this page, the php code checks if the delete button is clicked and connects to the database using a mysqli object. First, we have to delete records from conductor\_history with the same trainNumber because trainNumber is a foreign key in the conductor\_history table. The query from deleting from the conductor\_history table is “DELETE FROM `conductor\_history` where `trainNumber`=?”. As always prepared statement is used to preprare the query and the bind\_param() and execute() functions are called to execute the query. The query for deleting a train from the trains table is “DELETE FROM `trains` WHERE `trainNumber`=?”. This query is prepared and the bind\_param() function is called to handle ? by passing the trainNumber from the row that delete is being performed on.  The addLog() function is called and the employee id, description and action type are passed to this function so whenever an admin deletes a train, the log is updated. Similar to the update, the page prints “Delete Successful!” and it redirects to the search trains page so that the admin can check if the delete is reflected in the database. The search trains page consists of a link to the insert train page (“**adminInsertTrain.php**”). **adminInsertTrain.php** also requires **checkLogin.php** to see if an admin is logged in or the page will be redirected to index.php. This page also requires **core/log.php** so that the log can be updated every time an admin adds a train into the database. The insert train page consists of a form with all the columns from the trains table as input. The php checks if the submit button is clicked and connects to the database using mysqli object. Then it checks if a train with the same train number already exists in the database so the primary key is not duplicated. The query is “SELECT \* FROM `trains` WHERE `trainNumber`=?”. If a train with the same train number already exists in the database then it is printed to the screen using echo so the administrator is informed that the train already exists. If the train number is unique then it is inserted into the database using the following sql statement: “INSERT INTO `trains` (trainNumber, destination, startLocation, days, departureTime, arrivalTime) VALUES(?,?,?,?,?,?)”. The question marks are binded using stmt()->bind\_param function. The addLog() function is called and the employee id, description and action type are passed to this function so that the log can be updated whenever an admin inserts a train into the database. “Train has been updated” will be printed to the screen using echo so the admin is informed whether the train is added to the database. A link to the search trains page is provided so the admin can search the database to see if the insertion of the train is reflected in the database.

**Assign Equipment to Trains**

**Authored By**: Akshay Chavakula

**Files Discussed:** adminSearchEquipment.php, adminAssignEquipment.php, adminHandleAssign.php

Similar to the other admin pages, **checkLogin.php** is required to see if an admin is logged in. If not, the page will be redirected to index.php. **adminSearchEquipment.php** contains a search box and three radio buttons underneath which are the criteria for the search. The equipment table can be searched based on serial number, load capacity and type. The php code checks if the submit button is clicked then connects to the database using a mysqli object. If mysqki throws out an error then “Failed to connect to database” is printed to the screen to inform the user. Then it check to see what criteria (serial number, load capacity, type) is selected for the search. The default criteria is serial number which is always checked. If the search box is empty then, everything in the equipment is printed in a table on the website. The sql statements for the three criteria are as follows:

SELECT \* FROM `equipment` WHERE `serialNumber` LIKE ?, SELECT \* FROM `equipment` WHERE `loadCapacity` LIKE ?, SELECT \* FROM `equipment` WHERE `type` LIKE ?

Prepared statements are used to prepare the query and bind\_param() and execute() function are called to execute the query. The columns from the equipment table are dynamically loaded using the mysqli\_fetch\_field() function and the rows are loaded using the fetch\_array() function. Each row has a button called assign/update to assign the equipment to the train. Admin is able to edit all information for the equipment so the page redirects to **adminAssignEquipment.php** which contains a form to edit all the columns for the particular row that is selected. Serial number cannot be changed and it is going to be “readonly” because it is the primary key for the equipment table. The form will be pre-filled with the data from the row that the update button is clicked on. The admin will be able to edit this information including the trainNumber field to assign equipment to trains. Once this form is submitted, the page will redirect to **adminHandleAssign.php**. **adminHandleAssign.php** requires **core/log.php**, starts the session using the session\_start() function and checks to if the submit button is clicked using the isset() function. Then connects to the database using mysqli object. The query to update equipment:

UPDATE `equipment` SET `trainNumber`=?, `loadCapacity`=?, `type`=?, `location`=?, `manufacturer`=?, `price`=? WHERE `serialNumber`=?

This query is initialized and prepared using the stmt\_init() and prepare() functions and executed using the execute() function. The addLog() function is called and employee id, description and action type are passed to this function so the log can be updated whenever an admin assigns equipment to trains or changes any of the fields for equipment. Once the statement executes, “Equipment has been updated!” is printed to the web site using echo to inform the admin and the page is redirected to **adminSearchEquipment.php** so the admin can search the equipment table to see if the changes are reflected in the database.

Assign Conductors to Trains by Akshay Chavakula

**Files Discussed:** adminSearchCondHistory.php, adminUpdateCondHistory.php adminHandleCondHistory.php

**adminSearchCondHistory.php** requires **checkLogin.php** to use the checkLogin() function to see if an admin is logged in. The super global $\_SESSION is used to check the role is administrator and if not, the page will be redirected to index.php. Similar to the other admin pages, there is a form with a search box and two radio buttons underneath which are the criteria for the search. The two main criteria for searching the conductor\_history table are train number and id. The php code checks if the form is submitted using the isset() function. It connects to the database using mysqli object. The sql statements for the two criteria are as follows:

SELECT \* FROM `conductor\_history` WHERE `trainNumber` LIKE ?, SELECT \* FROM `conductor\_history` WHERE `id` LIKE ?

Prepared statements are used to prepare the query and bind\_param() and execute() function are called to execute the query. The columns from the conductor\_history table are dynamically loaded using the mysqli\_fetch\_field() function and the rows are loaded using the fetch\_array() function. Each row has a button called update to assign a conductor to a train. Admin is able to edit all information for the conductor\_history table so the page redirects to **adminUpdateCondHistory.php** which contains a form to edit all the columns for the particular row that is selected. **adminUpdateCondHistory.php** requires **checkLogin.php** to use the checkLogin() function to see if the admin is logged in and if not the page will be redirected to index.php. There will be pre-filled from with all the data from the row that is selected. Once the submit button is clicked, the page will be redirected to **adminHandleCondHistory.php**. This page require **core/log.php** and uses the session\_start() function to start a session. It checks to see if the submit button is clicked using the isset() function. Then it will connect to the database using mysqli object. The query to update conductor\_history table is:

UPDATE `conductor\_history` SET `startDate`=?, `endDate`=?, `trainNumber`=? WHERE `id`=?

This query is initialized and prepared using the stmt\_init() and prepare() functions and executed using the execute() function. The addLog() function is called and employee id, description and action type are passed to the function so that the log can be updated whenever an admin assigns a conductor to a train or updates any field for conductor\_history table. Once the statement executes, “Conductor history has been updated!” is printed to the web site using echo to inform the admin and the page is redirected to **adminSearchCondHistory.php** so the admin can search the conductor\_history table to see if the changes are reflected in the database.

**Assign Engineers to Trains**

**Authored By**: Akshay Chavakula

**Files Discussed:** adminSearchEngHistory.php, adminUpdateEngHistory.php, adminHandleEngHistory.php

**adminSearchEngHistory.php** requires **checkLogin.php** to use the checkLogin() function to see if an admin is logged in. The super global $\_SESSION is used to check the role is administrator and if not, the page will be redirected to index.php. Similar to the other admin pages, there is a form with a search box and two radio buttons underneath which are the criteria for the search. The two main criteria for searching the engineer\_history table are train number and id. The php code checks if the form is submitted using the isset() function. It connects to the database using mysqli object. The sql statements for the two criteria are as follows:

SELECT \* FROM `engineer\_history` WHERE `trainNumber` LIKE ?, SELECT \* FROM `engineer\_history` WHERE `id` LIKE ?

Prepared statements are used to prepare the query and bind\_param() and execute() function are called to execute the query. The columns from the engineer\_history table are dynamically loaded using the mysqli\_fetch\_field() function and the rows are loaded using the fetch\_array() function. Each row has a button called update to assign an engineer to a train. Admin is able to edit all information for the engineer\_history table so the page redirects to **adminUpdateEngHistory.php** which contains a form to edit all the columns for the particular row that is selected. **adminUpdateEngHistory.php** requires **checkLogin.php** to use the checkLogin() function to see if the admin is logged in and if not the page will be redirected to index.php. There will be pre-filled from with all the data from the row that is selected. Once the submit button is clicked, the page will be redirected to **adminHandleEngHistory.php**. This page requires **core/log.php** so the addLog() function can be called and uses the session\_start() function to start a session. It checks to see if the submit button is clicked using the isset() function. Then it will connect to the database using mysqli object. The query to update engineer\_history table is:

UPDATE `engineer\_history` SET `startDate`=?, `endDate`=?, `travelTime`=?, `trainNumber`=? WHERE `id`=?

## This query is initialized and prepared using the stmt\_init() and prepare() functions and executed using the execute() function. The addLog() function is called and the employee id, description, and action type are passed to the function so the log can be updated whenever an admin assigns an engineer to a train or updates any field in the engineer\_history table. Once the statement executes, “Engineer history has been updated!” is printed to the web site using echo to inform the admin and the page is redirected to **adminSearchEngHistory.php** so the admin can search the engineer\_history table to see if the changes are reflected in the database.

## **Employee and Admin View/Edit Information**

**Authored By**: Christopher Whetsel

**Files Discussed**: employeeInfo.php, condcutorInfo.php, adminInfo.php, engineerInfo.php, engineerEditInfo.php, conductorEditInfo.php, adminEditInfo.php, adminEditEmployeeInfo.php, prepareAdminEmployeeUpdate.php, employeeUpdate.php

The overall design of the employee information section of the project was relatively simple. Each page required **checkLogin.php** which started a session and checked to make sure the session variable ‘*id’* was set, meaning that an employee was logged in. Then each page further checked to make sure the correct type of employee (engineer, conductor, and admin) was accessing the page. If the employee was not supposed to access this page, it redirected them to the **employeeHome.php**.  Each type of employee had their own information page uniformly named **engineerInfo.php**, **conductorInfo.php**, and **adminInfo.php** which all similarly displayed the employee’s information by querying the group1 database. The SQL queries to access each employee’s information followed the form

"SELECT users.id, users.firstName, users.lastName, employee.username, users.phoneNumber, employee.address, {$role}.status, {$role}.rank FROM users JOIN employee ON (users.id = employee.id) JOIN {$role} ON (users.id = {$role}.id) where users.id = {$id}";

with slight differences based on the role so the query matched the database. For engineers and conductors, their train assignment history was also displayed. These files were linked from **employeeInfo.php** which used the session variable ‘*role’* to check what type of user was logged in and would redirect the user to the correct Info page based on their role. This was helpful for making the nav-bar so that no matter what employee was logged in the link in the nav-bar could be **employeeInfo.php** instead of having three different nav-bars, one for each type of employee.

The info pages also link to a page that allows the employee to edit their information through a form named (**employeeType)EditInfo.php**. This page includes **employeeInfo.php** to display the employee’s information so they know what they need to change. The PHP also checks to see if the user is an admin. On the conductor and engineer edit info pages, if it is not an admin, the session variables ‘*IDtoUpdate’*, ‘*RoleToUpdate’*, and ‘*UserToUpdate’* are set the values of the session variables ‘*id’*, ‘*role’*, and *’user’* respectively which were set by the login module. These update variables are used so that when an admin is editing other employees’ info, their ‘*id’* and ‘*role’* stored in the session are not lost. This is important for the reuse of these pages so that admins can update employee info using these same forms and not have to have an extra page for admins editing other employees’ information. The form allows them to edit any of their information except their employee id and username. Once submitted, all forms go to **employeeUpdateInfo.php** to handle the form action which checks each form value to see if it’s set and updates the appropriate records which allows the user to only enter values in the edit info form for values they actually want to update. It also allows the same PHP script to be used to handle all the EditInfo forms. It uses the session variables ‘*IDtoUpdate’* and ‘*RoleToUpdate’* for the ‘*id’* and ‘*role’* to find which employee is being updated. It stores any errors or failed updates in the session variable ‘*updateErrors’* which is an array. Once all updates are completed either successfully or not, a log entry is created using the log function from **core/log.php** and the user is redirected to **employeeInfo.php.** Admins could update other employee’s information (Except other admins) though the **adminEditEmployeeInfo.php** page which allowed admins to search conductors and engineers and choose them for update. Once chosen, the chosen employee’s id is passed to **prepareAdminEmployeeUpdate.php** using a hidden form input and used query the database for the employee’s role and username. Then the admin is redirected to the corresponding **EditInfo.php** page for either conductors or engineers based on what role the employee they wish to update is. Then they can use the same form as a conductor or engineer uses to update their info and the process is the same.

## **Website Style and Design**

**Authored By**: Christopher Whetsel

**Files Discussed**: core/style.css, home/style.css, home/index.php, index.php, bootstrapCDN.php, customerNavBar.php, homeNavBar.php, employeeNavBar.php

The design and style for the website was originally the role of our sixth member Emily Temple until the beginning of April when she informed us that she was going to drop the class and switch majors away from Computer Science. Since I had most of my work done for the employee information pages and had some knowledge of bootstrap, I volunteered to take over the responsibility of coming up with a style for the website to adhere to and of turning the different modules into a functioning and cohesive web application.

I began with what Emily had started which was the home page of the project and decided to use bootstrap extensively for the layout of website. She had some css styling done with the background and logo which I decided to keep and are still used on the home page, **home/index.php**, currently. For the look of the site, we decided to go with dark grey nav-bar, lighter grey or white text, red buttons, and light grey background for most div elements.

Each page of the site has a nav-bar at the top with links appropriate to type of user currently using the site. The nav-bars all use bootstrap classes for nav-bar styling. The home and the customer navbars are very similar and only differ in the fact that the home navbar has a link to an about paragraph located on the **home/index.php page**. The home navbar is used on the pages in the home/ directory while the **customerNavBar.php** is used on all pages in the customer/ directory and the **employeeNavBar.php** is used on all pages in the employee/ directory. The customer and home navbars have the functionality that allows users to log in. On the navbar, there is a button which says login with a bootstrap glyphicon for login that, when pressed, toggles the collapse property of a div with a login/register form. The Login form has fields for password and username and two buttons to login or to register. When login is pressed, the username and password are evaluated using **loginLogic.php** which is included at the top of the home page and customer pages. The register button toggles the displayed form to be the register form which has fields for employee id, username, password, and confirm password. It has two buttons, register and cancel. Cancel toggles back to the login form and register will submit the form to the same page where the **loginLogic.php** will handle the registration. The **employeeNavBar.php** checks what type of employee is logged in and displays an appropriate navbar for them. Engineers and conductors both get the same navbar with links to the **employeeHome.php** homepage, a page to view their information, a link to view the log, and a link to logout. The admin navbar has the same links but also has links to edit trains, equipment, engineer and conductor assignment, and employee info since they have full access to the site.

The general style for all the sites in the web application was achieved with a simple style sheet core/style.css which provided style for the login form, background color of the body, and the color of the text. The homepage **home/index.php** had its own stylesheet home/style.css with additional styling. Further styling for the site was achieved using bootstrap. All body elements were contained within a div with *class=”container-fluid*” and another div with *class=”well”.* Forms were styled with *class=”form-group”* on the html form element and *class=”form-control”* on input elements. Buttons were styled with *class=”btn btn-lg* *btn-danger*” to give them a red color that fit the scheme of the site. Tables were given the bootstrap class=”table table-responsive”. Bootstrap was added to the pages using a CDN link and the links were included on each page by including **core/bootstrapCDN.php.**  **public\_html/Index.php** simply redirects to **home/index.php**.

## **Customer Planning and Design**

**Authored By**: Joseph Crowe

For the customer task, the customer had to be able to browse through cars, choose his or her car type and location, enter his or her company id, and fill out various fields to reserve the car. Customers had to be able to apply filters in the search. Customers could book multiple cars at once if there were cars of the same type and location. Customers must also have a 5% sales tax added on to the total price. Once they approved of the reservation, the equipment must be booked for the customer and the log must be updated accordingly.

The first subtask was filtering the search. In order to display the available categories of train cars, I used an SQL query to find all the unique values of the two categories, city and equipment type. I also excluded the locomotive type as the group decided that customers would not be able to reserve locomotives. I used the SQL query to create a drop-down menu with all the possible equipment types and cities. The use of the dynamic drop-down creation was chosen because if the number of car types changed or the number of locations changed, the website would still be up to date. An ‘All’ category was also added and this led to a small challenge. I discovered that I could add another ‘AND’ condition if the value selected was not ‘All’.  The price filter was implemented using two number types. Since the number type is not immune to SQL injection, a prepared statement was required to execute this filter.

After the filter was applied, the customer had to see all available train cars. This was easy to execute as all the available train cars had a null value for the id field.

After the customer selected the car, they also had the option to order multiple cars at once if they were the same type and had the same location. This was accomplished by finding the number of rows where the city and type were the same and the id was null. This allowed customers to book multiple cars at once and was very convenient because it would not allow customers to attempt to order more cars than available. Several of the other fields were also read-only so the customer could not change information.

After the customer ordered the number of cars as well as inputted their customer ID and cargo type, the customer could then see a confirmation page to review and confirm his or her order. This confirmation page would include the price with the sales tax included. The confirmation page was also read-only so the customer could not edit any information. Once the customer confirmed his or her order, the equipment table would be updated with the customer id for the cars ordered and a log with the total and the individual car bookings would be created.

Overall, I learned mostly about the integration of the front end and back end in this section of the project. I discovered that a well implemented back end made the front end of the website much easier. The design of the database allowed for relatively simple queries. I also discovered that sometimes, complexity on the front end is required. Discovering the number of cars with the same city and equipment that were not reserved had to be handled on the front end. This project overall led to an appreciation of balance between back end and front end. The back end should simplify things as much as practical, but at some point, complexity will be required by the front end to enforce database rules. The front end must be responsible for only allowing relevant and reliable information into the database. This task was made easier within the web application with drop-down menus and read-only fields. The back end of the database was responsible for storing and connecting all of the information.

Enumerated Algorithms for Customer:

Type filter: "SELECT DISTINCT type FROM equipment WHERE id IS NULL AND type != 'locomotive'";

City filter: "SELECT DISTINCT location FROM equipment WHERE id IS NULL";

Table Query: $sql = "SELECT serialNumber, loadCapacity, type, location, manufacturer, price from equipment WHERE type != 'locomotive' AND id IS NULL AND price < ? AND price > ?";

                       if ($\_POST['type'] != 'all'){

                               $sql = $sql .  " AND type = '" . $\_POST['type'] . "'";

                           }

                           if ($\_POST['location'] != 'all'){

                            $sql = $sql .  " AND location = '" . $\_POST['location'] . "'";

                            }

Number of Cars Query: "SELECT type FROM equipment WHERE id IS NULL AND type = '" . $\_POST['type'] . "' AND location = '" . $\_POST['location'] . "'"; Used number of rows from result.

Check for valid Customer ID:    SELECT \* FROM users WHERE id = ?

Find cars for updating:   "SELECT serialNumber FROM equipment WHERE id IS NULL AND type = '" . $\_POST['type'] . "' AND location = '" . $\_POST['location'] . "'";

Update Cars: "UPDATE equipment SET id = ? WHERE serialNumber = '" . $row[0] . "'";

Log Statement: 'INSERT INTO log (logNumber, ipAddress, ocassionTime, description, id, actionType) VALUES (DEFAULT, ?, ?, ?, ?, 2)')

## **Website Logging Function**

**Authored By**: Timothy Shea

Many actions taken by users on the website needed to be stored into a table in the database. The users would then be able to visit a page to view specific actions related to them. To view the log, first navigate to the viewLog page. If you are logged in as a conductor or engineer, you will be prompted to select a number of logs to view, and an option to sort them by date of occurrence (newest and oldest). If you are not logged in, you will be treated as a customer and given the same prompts as the conductor and engineer users, but with the addition of a field to enter in a customer ID. If you are logged in as an administrator you will be given the field for the number of logs to view, the field to sort them by date of occurrence, and an additional field to sort them by the type of action that occurred on the website. Hitting the submit button will grab all logs based on the user type and the specified fields.

The option to sort by type of action that occurred is only given to administrators, as it does not seem wise to display sensitive information about the things that happen on the website to anyone who could get to the page and view the drop-down menu.

There are two php files associated with the log: log.php and viewLog.php. The first file, log.php, includes a function that, when called and passed a user ID, a description of the action that took place, and an action type variable, will add the data associated with the action to the log table in the database. The function itself handles getting the IP address of the user, as well as the time that the action took place.

The second file, viewLog.php, has an HTML form that displays options and information depending on the type of user that is logged in. If no user is logged in, it defaults to displaying a field to enter a customer ID. When the submit button is pressed, it checks to see what user has requested to view the log, then checks to see that their ID is in a valid format and displays a message if it is not. The user ID is then checked against the database to verify what type of user wants to view the log, and to verify that the user is a registered user or customer. If the user is not registered in the database, an error message is displayed. If the user ID passes the security checks, the database is queried using the information submitted from the form, and the user ID. The query that is ran on the database again depends on what type of user submitted the form. An administrator has a query that selects everything, while conductors, engineers, and customers use the same query that only returns entries associated with their ID. The result of the query is handed off to a function to be printed to the web page. If no logs were found in the database, an error message is displayed.

The bulk of the work for the log was for validating user ID's as well as other security checks to make sure that users could not see information that was not related to them. Because customers do not log in and instead enter a user ID in the form, I had to be certain that they could not enter the ID of an employee or administrator and view the logs associated with that ID. Before the database is queried, another check is performed on the user ID to verify that belongs to an existing customer or employee. This is needed primarily for the employees, because if their ID was removed from the database for some reason, lets say their employment was terminated, we wouldn't want them to be able to check the log if their account was still signed into the website. You might think that the query would return no results for their ID if it was removed from the users table, but that ID should still exist in the log table.

The most difficult part of the log task was coming up with ways to try and get information from the database by using invalid or mismatching user IDs or by providing none at all. It was necessary to try and break in myself so that I could implement fixes to prevent these security flaws from happening when the website is ready to be used. Unfortunately, these fixes often involved restructuring large portions of the program as well as rewriting multiple functions, because changing the way one function worked often meant that I had to tweak another. Being absolutely sure that everything worked together and that I hadn't left something out took up a good portion of the time that I had dedicated to the task.

From doing this project, I learned that on a large-scale project that has many interconnecting parts planning and communication is critical to success. Planning out the project is not something to be taken lightly, as having a solid plan of how the website will fit together and function will save you a lot of work and hassle in the future. If everything fits together according to the plan, you won't find yourself having to rewrite large portions of code or having to implement large patches and fixes. If you don't have a plan to begin with, or don't follow the agreed upon plan, the project can get very confusing and end up being much more complicated than it needs to be.

Instruction Manual

## Customer Users

Customer are allowed to use the site without logging in to reserve train cars and view their reservations using their assigned customer number. When your business registered with Missouri Rail, you should have received a customer number. You may access the site using this link [here](http://cs3380.rnet.missouri.edu/~GROUP1/home/index.php) to the homepage. From this homepage you may reserve cars by pressing the link on the navbar at the top of your page that reads “Trains.” Once pressed, you will be presented with a page that allows you to filter all available train cars for browsing. Enter the information you wish to filter by into the form and click filter. The results will be displayed on the lower half of the same page. On the right side of the results table there will be a button that reads “Order.” Once you have decided which car you would like to reserve, click the order button. You will then be presented with a form displaying the information about the car you wish to reserve. Enter your Customer ID, a description of the cargo, and choose how many of this car you would like to reserve (if only one is available, you will only be able to reserve one). Click order once your information has been entered. You will then have a chance to confirm your order. If all the information looks correct, click the confirm button at the bottom of the form.

If you wish to view your past reservations, click the “Reservations” link on the navbar at the top of your screen. You will be presented with a form asking for your Company ID (the same as your Customer ID) and for criteria to filter your reservations. Enter the required information and press the submit button. Your reservations, if you have any, will be displayed at the bottom of the page.

## Employee Users

Conductors, Engineers and Administrators will need to login to use the web application. You may log in at this link [here](http://cs3380.rnet.missouri.edu/~GROUP1/home/index.php). Click the login link in the top right of your page. This will display a panel with a login form with buttons to login and to register. If you have not registered previously to use the site, you will need to click register. Once pressed, a form to register will be displayed. Enter the required information and click “Register.” If your employee number is correct and the username is not already taken, you will be registered and logged in. To login, enter your username and password in the login form and click the “Login” button. If your username and password are correct, you will be logged in and redirected to the employee home page.

Admins, conductors and engineers, will be able to view and edit their information by clicking the link “View/Edit Your Info” on the navbar at the top of your page. Once pressed you will be displayed with your information including past train assignments if applicable. You can edit your information by following the link at the bottom of the page. You will then be presented with a form that allows you to enter in new information into the fields you choose. You only need to enter information in the fields you wish to change. If you do not want to update a field, leave it blank. Please note that you cannot change your username or ID. Click submit and your changes will be saved. Employees can view the log of their actions by following the link “Action Log” on the navbar. You will be presented with criteria to filter the log based on. Enter your criteria and click submit to view your log entries. You can logout of the site by clicking the logout link in the top right of your screen on the navbar.

For administrators, to view or edit an employee’s information, following the “Edit Employee Info” link on the navbar. Enter criteria in the form to filter for employees and click the Submit button. The results of your search will be displayed below the form. Once you have found the correct employee, click the “Update” button to the right of the employee you wish to update. Enter the information you wish to change in the displayed form and click sumbit. Leave the fields you do not wish to change blank. Please note that you cannot edit other administrator’s information.

Administrators can also edit conductor and engineer train assignments. Click “Conductor Assignments” on the navbar. Enter the information you wish to search conductor based on and click the Search button. Locate the record you wish to update and click the Update button to the right of that record. Enter the updated information into the displayed form and click Update. The process is the same for updating Engineer Assignments.

Trains can be added, updated, or deleted by following the “Trains” link on the navbar. To add a trains, follow the “Add train” link at the top of the page. Then, enter the information for the train and click submit. To update or delete trains, use the form to provide criteria to search the trains and click the “Search” button. The results will be displayed by below the form. Once you locate the trains you wish to update or delete, click the “Update” or the “Delete” button depending on which action you wish to complete. If you wish to update, fill out the fields to change in the displayed form and click the “Update Trains” button.

To update equipment, click the link “Equipment” on the navbar. Enter the criteria you wish to search the equipment on and click the “Search” button. Once you locate the equipment car you are looking for, click the “Assign/Update” button that corresponds to that car. Enter the information you wish to change into the displayed form and click the “Update” button.

Project Difficulties and Learning Experience

This project overall went smoothly, there were points in time where we would have minor inconveniences. A large portion of our project was delayed while finding a time that everyone could meet, as well as a team member dropping from the course. Due to our meeting times being typically an hour, we often spent the whole time attempting to find and fix issues, then on our individual times we would apply the fix. This was incredibly time consuming, especially on the ERD, which after completion, had no feedback provided, leading to continuous updates and changes throughout the project. These changes were hard to handle due to not building our programs together, and more individually, we had to consider that everyone codes differently, none of us had used a unified way to error test, this caused some errors to have a significant testing time, or nearly impossible to replicate unless under certain situations. Coding differences also caused it to be much harder to debug, especially when PHP code was used to change pages. Changing the name of a PHP file would require an update throughout all files, which could take a couple tests before completely updating the files this was increasingly difficult due to needing to learn more about the usage of PHP and how it interacts using prepared statements and integrating PHP into HTML. During sections of the project where we would all need to decide how an interaction would occur, such as the abilities for customers, where we couldn’t conclude to what capabilities we would like them to utilize, like creating an account. In the end, we would make a decision, but typically it was made based on who would design the element that was under debate.

As far as what was learned, we found it incredibly important to learn how to work with a team, towards the end of the project, it seemed more as though we had multiple individual assignments and connecting them lead to quite a few more errors that had to be corrected and took a bit of extra time to be completed. We learned a lot about utilizing the capabilities of each individual to ensure that we all assigned tasks to be completed in a timely manner while still being completed in a well-organized and coherent method.

Conclusion

Throughout this project, we have utilized the knowledge provided by the course to create a fully operational and functional website that is used to provide users the ability to manage and use a railroad cart system. This system had four user types to be accounted for, these were customers, engineers, conductors, and administrators. Each user type was required to provide different functionality to the railroad system. Customers should be able to search the database for all available carts, this should be able to be sorted based on the price, location and type of carts. The price is then posted for the carts, and when selected they will be taken to a reservation page that shows the price plus tax, on this page they will be provided with a final chance to modify reservations or continue, at which point it would be logged into the database.

The engineer and conductor have very similar requirements, they should be able to create and manage an account on the website, for engineers this should store their basic information such as first name, last name, rank, status, and hours traveled they should be able to view their log and view current and previous work assignments. Conductors require the same access to the website, but have no requirement to keep track of their hours traveled.

Administrators will have access to all information of the website, they will be able to edit any information of workers and customers, they can reset passwords and monitor all trains. They are the only ones that can add individuals to the engineer and conductors, though the engineer must register using their employee number to set their username. They will also be able to manage all trains and equipment that are within the system, as well as update and insert additional equipment and trains.

All of the login information will be ran through a user authentication module, this will store all usernames and password, the passwords will be hashed and then stored within a table. This system will be used by the administration, engineers, and conductors within our system. As well as all authentication, there is a web logging system that will keep track of all activities on the website, these include activities such as logging in, reserving trains, password changes, and employee modifications.

Overall this project worked very well, the website provides a very user friendly system that met and exceeded all expectations set in the requirements. This project was incredible in helping teach that even though a document is open ended it is best to get as much information as possible from the consumers. This information should be used to develop a system that can be easily updated and expandable, especially since requirements may change, additional information is required or set, and as times change user requirements may expand. This project did an exceptional job to tie all links of the class together to provide a unified understanding of simple databases using HTML, PHP, and Mysql.

Website Information

1. <http://cs3380.rnet.missouri.edu/~GROUP1/home/index.php>

|  |  |  |  |
| --- | --- | --- | --- |
| Registered User Credentials | | | |
| Username | Password | Role | ID |
| admin | pass | Administrator | 4 |
| engineer | pass | Engineer | 3 |
| conductor | pass | Conductor | 2 |

|  |  |
| --- | --- |
| Unregistered User Credentials | |
| Role | ID |
| Administrator | 5 |
| Engineer | 6 |
| Engineer | 7 |
| Engineer | 8 |
| Conductor | 9 |
| Conductor | 10 |

|  |
| --- |
| Customer IDs |
| 1 |
| 11 |
| 12 |