BUS MANAGEMENT SYSTEM

CPSC 471 Final Project Report

Group 20

Hannah MacDonald, Madisson Carle, Brianna Cyr

ABSTRACT

For this project, we designed and implemented a system for a company, henceforth referred to as “Company A”, to store information about bus routes and their passengers. The purpose of creating this system was to enable effective contact tracing in the event that a passenger tested positive for COVID-19. To design our system, we used a variety of concepts that we learned throughout the semester in CPSC 471. Ultimately, we produced a functional bus route and passenger records system, with an interactive interface via a website backed by a database.

INTRODUCTION

The current paper manifest system for bus transportation at Company A has proven to be unreliable and inefficient. This is due to the high expectations for the bus driver to know which seats are taken and where passengers should sit when boarding. This paper log system is also difficult for administrators to manage because it lacks easy searching capability. This is an issue, since it makes it difficult to take proper contact tracing measures if any passengers on the bus test positive for COVID-19.

We realized that the existing system could be improved with a digital management system, comprised of a database and a corresponding website, which we then created. This new system takes most of the responsibility of assigning seats away from the drivers. It also allows for a more accurate log of passenger trip information to be accumulated and accessed by administrators whenever needed. We began the process of designing this system by making an extended entity relationship diagram. This diagram was then translated into a relational model, which was used to make an object-oriented model of the system. Together, these diagrams and models served as the basis for the relational database, API and website that we created.

PROJECT DESIGN

The following sections discuss the details of our project’s design.

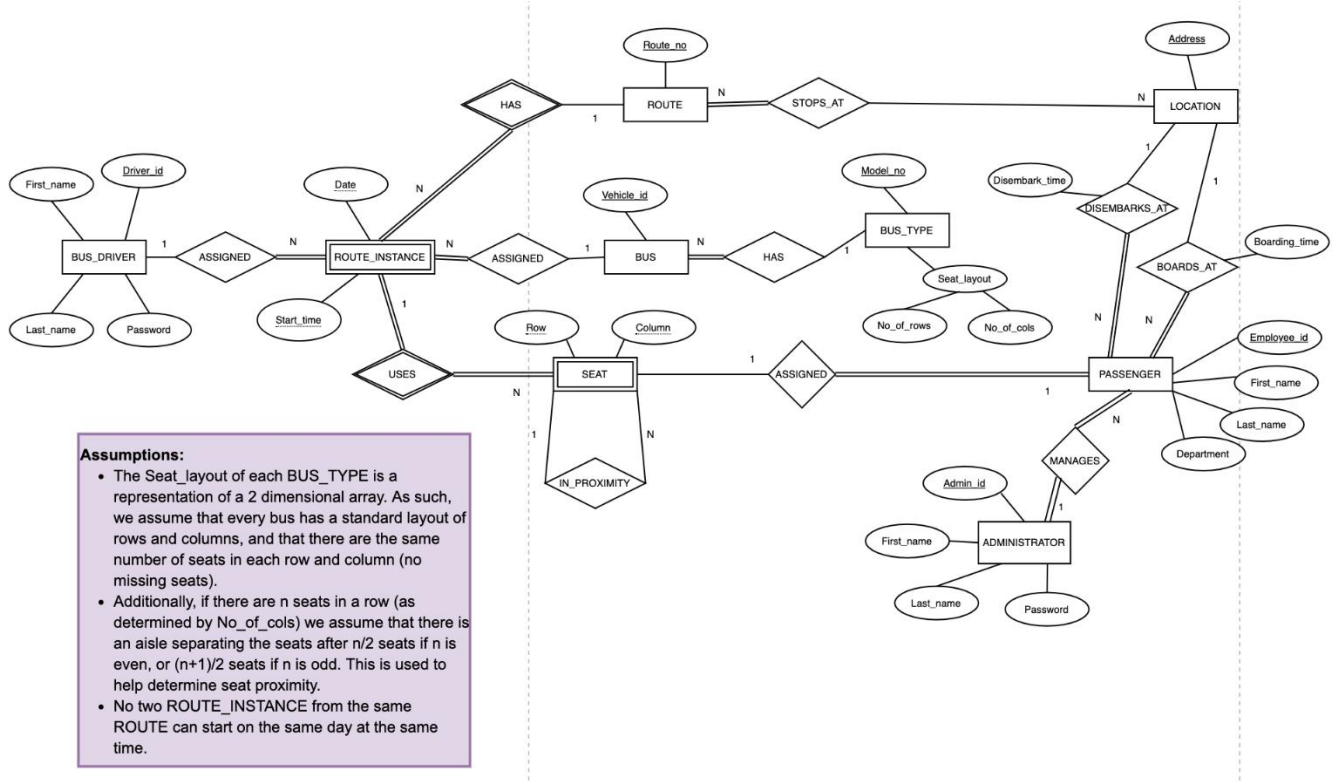
USERS

The system we created supports two types of end-users: administrators and bus drivers. Bus drivers are able to enter the information about the bus they are driving, the route they are taking, and their own login information to create a new route instance. This route instance represents a particular trip that a bus would take, and has information about the associated route as well as the route date and start time. It is also associated with a particular bus, which has a bus type. This bus type describes how many seats are on the bus using the number of rows and number of columns on the bus. It is assumed that the seats on the bus are arranged like a two-dimensional array, and thus the number of seats would simply be the number of rows multiplied by the number of columns. The seats are then created and associated with a

particular route instance, with their particular row and column being stored to determine where exactly the seat would be on the bus. The seats that are in close proximity to one another are also determined at this stage. A seat is said to be in close proximity if it is located one column above or below another seat, one column to the left or right of another seat, or both. Once this new route instance is created, a bus driver can enter the employee identification number of a passenger in order to automatically assign them to a seat, which sets their boarding time and location. This information is stored for later use by the administrators and is also viewable by the bus driver at the time of boarding. Finally, the bus driver can end the route, which sets the disembark location and time for all passengers on that route instance. This system allows the bus driver to not have to manually assign seats to passengers, and to have all relevant information for an administrator be stored with little hassle.

Administrators first have to login with accurate credentials. After login, administrators have two main functionalities to use. The first functionality allows administrators to get information on an employee in the database by entering their first and last name. If the name entered is correct, the system will output all stored information on the employee and allow the user to return to the previous page. If the entered name is incorrect the system will present a button that will take the user back to the administration main page. The second functionality allows administrators to find information on an employee's previous bus trips. This is used for cases when an individual has been found contagious for COVID-19. This allows for other passengers near the contagious individual to take the necessary precautions without rendering everyone on that bus unable to work. By entering the contagious individuals ID all passengers that were in close proximity on any route will be listed with all their information. If the employee ID is invalid the system will present a button that will take the user back to the administration main page. The final option on the main view is to allow the administration user to logout of their account. By selecting this option, the user will be rerouted to the initial shared view.

EER DIAGRAM

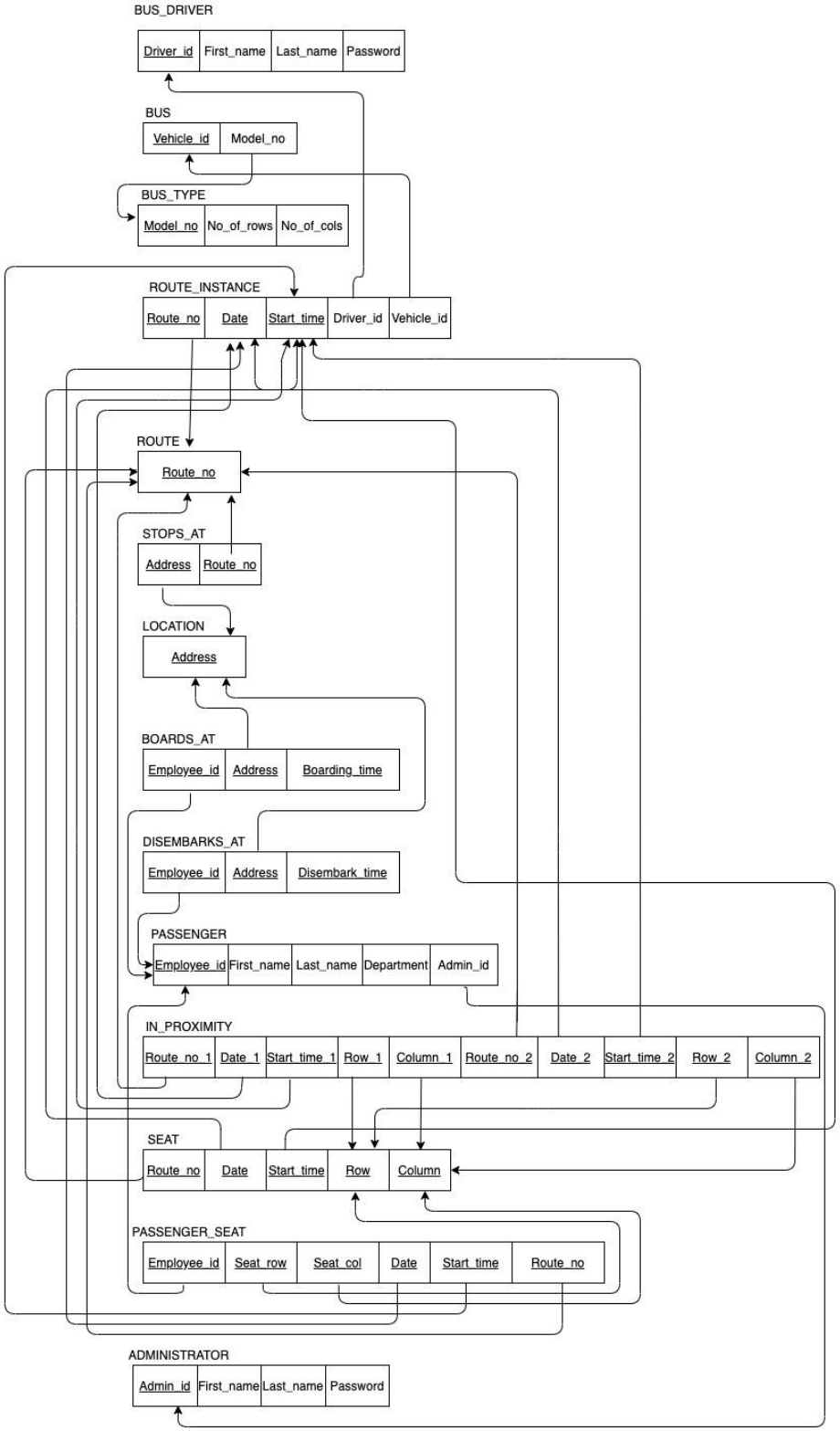


The above EER diagram was the first step in our design process. No changes have been made to it since its initial creation. In addition to the assumptions listed in the diagram, we also assume that two seats that are IN_PROXIMITY must have the same Route_no, Date, and Start_time.

IMPLEMENTATION

The following sections discuss the details of our project's implementation.

RELATIONAL MODEL

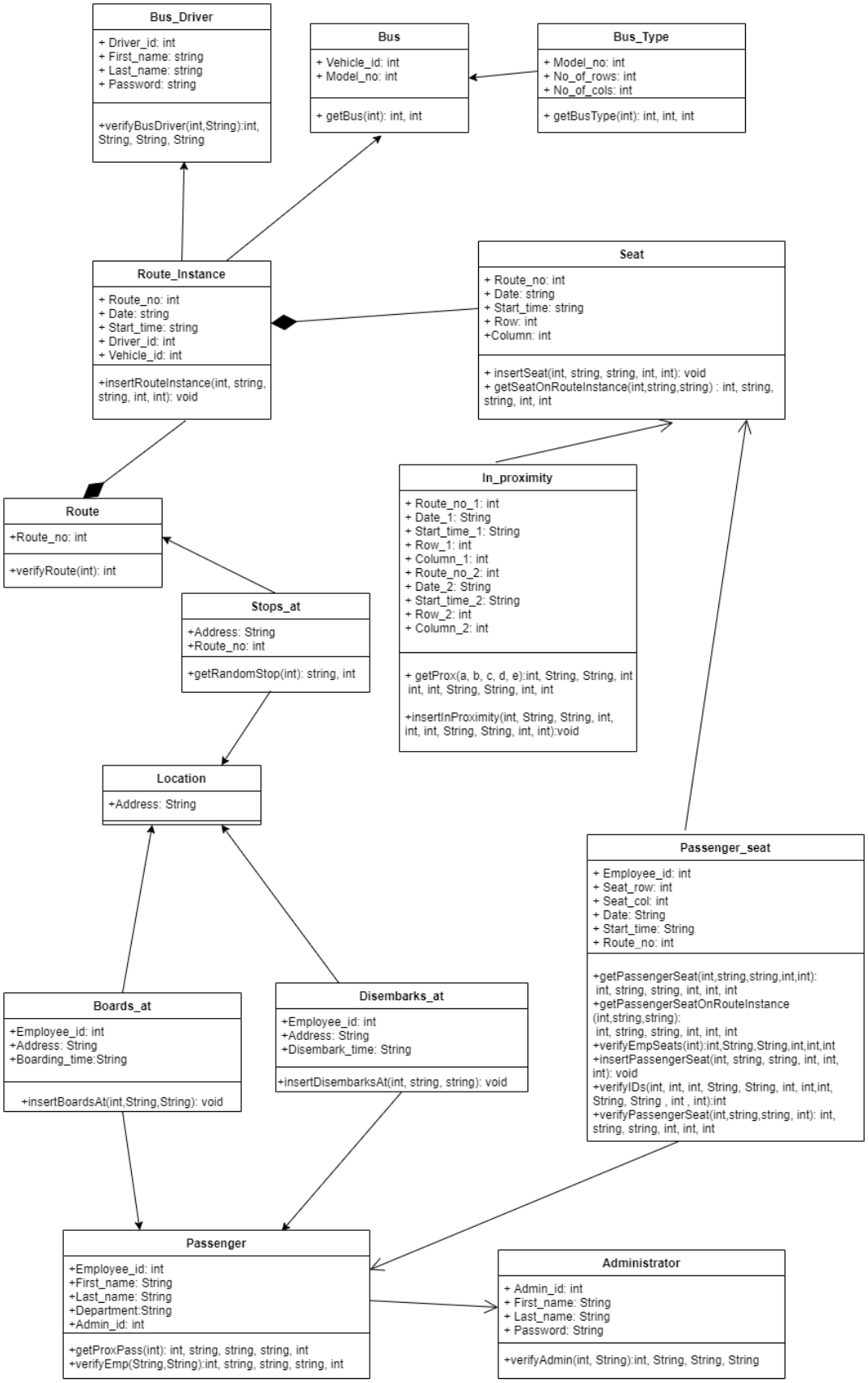


The above relational model diagram was the second step in our design process. To create it, we followed the procedure for converting an EER diagram to a relation model outlined in the CPSC 471 course content. We made two minor changes to it after its creation. We changed Boarding_time in BOARDS_AT from a normal attribute to part of the primary key. We also changed Disembark_time in DISEMBARKS_AT from a normal attribute to part of the primary key. These changes were made so that each employee could only board or disembark at a single location at a single time.

SQL STATEMENTS

To implement our bus management system, we used MySQL 8 to create a relational database. This database was the primary component.

On the next page is the object oriented model of our system, with all of the SQL statements named as functions. Input and output types, such as integer or string, are specified on the diagram, while the exact contents of each input are explained with the statements that follow.


Bus_Driver
+ Driver_id: int
+ First_name: string
+ Last_name: string
+ Password: string
+verifyBusDriver(int,String):int, String, String, String
Bus
+ Vehicle_id: int
+ Model_no: int
+ getBus(int): int, int
Bus_Type
+ Model_no: int
+ No_of_rows: int
+ No_of_cols: int
+ getBusType(int): int, int, int
Route_Instance
+ Route_no: int
+ Date: string
+ Start_time: string
+ Driver_id: int
+ Vehicle_id: int
+insertRouteInstance(int, string, string, int, int): void
Seat
+ Route_no: int
+ Date: string
+ Start_time: string
+ Row: int
+Column: int
+ insertSeat(int, string, string, int, int): void
+ getSeatOnRouteInstance(int,string,string) : int, string, string, int, int
Route
+Route_no: int
+verifyRoute(int): int
In_proximity
+ Route_no_1: int
+ Date_1: String
+ Start_time_1: String
+ Row_1: int
+ Column_1: int
+ Route_no_2: int
+ Date_2: String
+ Start_time_2: String
+ Row_2: int
+ Column_2: int
+ getProx(a, b, c, d, e):int, String, String, int int, int, String, String, int, int
+insertInProximity(int, String, String, int, int, int, String, String, int, int):void
Stops_at
+Address: String
+Route_no: int
+getRandomStop(int): string, int
Location
+Address: String
Passenger_seat
+ Employee_id: int
+ Seat_row: int
+ Seat_col: int
+ Date: String
+ Start_time: String
+ Route_no: int
+getPassengerSeat(int,string,string,int,int): int, string, string, int, int, int
+getPassengerSeatOnRouteInstance (int,string,string): int, string, string, int, int, int
+verifyEmpSeats(int):int,String,String,int,int,int
+insertPassengerSeat(int, string, string, int, int, int): void
+verifyIDs(int, int, int, String, String, int, int,int, String, String , int , int):int
+verifyPassengerSeat(int,string,string, int): int, string, string, int, int, int
Boards_at
+Employee_id: int
+Address: String
+Boarding_time:String
+insertBoardsAt(int,String,String): void
Disembarks_at
+Employee_id: int
+Address: String
+Disembark_time: String
+insertDisembarksAt(int, string, string): void
Passenger
+Employee_id: int
+First_name: String
+Last_name: String
+Department:String
+Admin_id: int
+getProxPass(int): int, string, string, string, int
+verifyEmp(String,String):int, string, string, string, int
Administrator
+ Admin_id: int
+ First_name: String
+ Last_name: String
+ Password: String
+verifyAdmin(int, String):int, String, String, String

The SQL statements below are presented along with the filename of the files they appear in. A '?' indicates an input parameter, which is described below the statement. What the statement should return is also described, if anything.

verifyBusDriver.php:

SELECT * FROM bus_driver where Driver_id=? AND Password =?

Parameters (in order): the driver's id, the driver's password

Returns: the bus driver with the given id and password (if such a bus driver exists)

verifyRoute.php:

SELECT * FROM route where Route_no=?

Parameters (in order): the route number

Returns: the route with the specified route number (if such a route exists)

getBus.php

SELECT * FROM bus where Vehicle_id=?

Parameters (in order): the vehicle id, also known as the bus number

Returns: the bus with the specified vehicle id, including its model number (if such a bus exists)

insertRouteInstance.php

INSERT INTO route_instance (Route_no, Date, Start_time, Driver_id, Vehicle_id) VALUES (?, ?, ?, ?,?)

Parameters (in order): the route number, the date, the start time, the driver id, and the vehicle id of the route instance to be added

Returns: nothing (creates a route instance with the given information if there is no existing route instance with the same route number, date, and start time)

getBusType.php

SELECT * FROM bus_type where Model_no=?

Parameters (in order): the model number

Returns: the bus type with the given model number (if such a bus type exists), including no_of_rows and no_of_cols

insertSeat.php

INSERT INTO seat VALUES (?, ?, ?, ?, ?)

Parameters (in order): the route number, the date, the start time, the row number, and the column number of the seat to be added

Returns: nothing (creates a seat with the given information if there is no existing seat with the same route number, date, start time, row number, and column number)

insertInProximity.php

INSERT INTO in_proximity (Route_no_1, Date_1, Start_time_1, Row_1, Column_1, Route_no_2, Date_2, Start_time_2, Row_2, Column_2) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

Parameters (in order): the first seat's route number, the first seat's date, the first seat's start time, the first seat's row number, the first seat's column number, the second seat's route number, the second seat's date, the second seat's start time, the second seat's row number, and the second seat's column number of the in-proximity entry to be added

Returns: nothing (creates an in-proximity entry with the given information if there is no existing in-proximity entry with the same first seat and same second seat)

getSeatOnRouteInstance.php

SELECT * FROM seat where Route_no=? AND Date=? AND Start_time=?

Parameters (in order): the route number, date, and start time of the route instance

Returns: all seats on the given route instance with the route number, date, start time, row number and column number (if such seats exist)

getPassengerSeat.php

SELECT * FROM passenger_seat where Route_no=? AND Date=? AND Start_time=? AND Seat_row=? AND Seat_col=?

Parameters (in order): the route number, date, start time, row number, and column number of the seat

Returns: the seat and its assigned passenger with the given route number, date, start time, row number, column number, and employee id (if such a seat-passenger combination exists)

getPassenger.php

SELECT * FROM passenger where Employee_id=?

Parameters (in order): the passenger's employee id.

Returns: the employee id, first name, last name, department, and administrator id for that passenger (if such a passenger exists)

verifyPassengerSeat.php

SELECT * FROM passenger_seat where Route_no=? AND Date=? AND Start_time=? AND Employee_id=?

Parameters (in order): the seat's route number, the seat's date, the seat's start time, and the passenger's employee id

Returns: the seat and its assigned passenger with the given route number, date, and employee id (if such a seat-passenger combination exists)

insertPassengerSeat.php

INSERT INTO passenger_seat VALUES (?,?,?,?,?,?)

Parameters (in order): the seat's route number, the seat's date, the seat's start time, the seat's row number, the seat's column number, and the passenger's employee id

Returns: nothing (creates a seat-passenger combination with the given information if there is no existing seat-passenger combination with the same seat and passenger)

getRandomStop.php

SELECT * from stops_at WHERE Route_no =? ORDER BY rand() LIMIT 1

Parameters (in order): the route number

Returns: a stops-at entry with a random address that route stops at and the given route number (if such a stops-at entry exists)

insertBoardsAt.php

INSERT INTO boards_at (Employee_id,Address,Boarding_time) VALUES (?,?,?)

Parameters (in order): the passenger's employee id, the location's address, and the time at which the passenger boards at that location

Returns: nothing (creates a boards-at entry with the given information if there is no existing boards-at entry with the same passenger and location)

getPassengerSeatOnRouteInstance.php

SELECT * FROM passenger_seat where Route_no=? AND Date=? AND Start_time=?

Parameters (in order): the route number, date, and start time of the route instance

Returns: all seats and their assigned passengers on the given route instance with the route number, date, start time, row number, column number, and employee id (if such seats exist)

insertDisembarksAt.php

INSERT INTO disembarks_at (Employee_id,Address,Disembark_time) VALUES (?,?,?)

Parameters (in order): the passenger's employee id, the location's address, and the time at which the passenger disembarks at that location

Returns: nothing (creates a disembarks-at entry with the given information if there is no existing disembarks-at entry with the same passenger and location)

getProx.php

SELECT DISTINCT * FROM in_proximity WHERE

(Route_no_1=? AND Date_1=? AND Start_time_1=? AND Row_1=? AND Column_1=?) OR

(Route_no_2=? AND Date_2=? AND Start_time_2=? AND Row_2=? AND Column_2=?)

Parameters (in order): The route number, the date, the start time, the seat's row and column, the route number, the date, the start time, the seat's row and column

Returns: returns in_proximity tuples where the passenger seat information matches (if such in_proximity tuple exists)

getProxPass.php

SELECT * FROM passenger WHERE Employee_id=?

Parameters (in order): The employee ID

Returns: a passenger tuple where the ID matches (if such tuple exists)

verifyAdmin.php

SELECT * FROM administrator where Admin_id=? AND Password =?

Parameters (in order): The admin ID entered, the password entered

Returns: Returns an administrator tuple with the login information matching it (if such tuple exists)

verifyEmp.php

SELECT * FROM passenger WHERE First_name =? AND Last_name=?

Parameters (in order): The passenger First name, the passenger last name

Returns: returns a passenger tuple with the first and last name matching it (if such tuple exists)

verifyEmpSeats.php

SELECT * FROM Passenger_seat WHERE Employee_id =?

Parameters (in order): The employee ID

Returns: returns a passenger_seat tuple where the ID matches (if such tuple exists)

verifyIDs.php

SELECT Employee_id FROM passenger_seat WHERE

Employee_id <> ? AND

(Route_no=? AND Date=? AND Start_time=? AND Seat_row=?AND Seat_col=?) OR

(Route_no=? AND Date=? AND Start_time=? AND Seat_row=? AND Seat_col=?)

Parameters (in order): (all from a tuple of in_proximity) The first route number, the first date, the first start time, the first seat's row and column, the second route number, the second date, the second start time, the second seat's row and column

Returns: returns an employee ID if a passenger_seat was found that matches (if such ID exists)

API DOCUMENTATION

This is included as the file “471 API Documentation.pdf” as well as “471 API collection.postman_collection.json” for the raw export from Postman. Note that some of the example returned json objects get cut off in the pdf, and are thus included at the end in full. A link to the online version of this documentation is also available here: https://documenter.getpostman.com/view/13736092/TVmS6aMt

USER GUIDE

The following files are part of the website we created and not the API:

- addPassengerSeat.php
- adminMainView.php
- busDriverLogin.php
- CheckAdmin.php
- CheckDriver.php
- endRoute.php
- index.php
- LoginAdmin.php
- logoutAdmin.php
- searchEmp.php
- searchProximity.php
- showBusLayout.php

The following files are the API files:

- config.php
- getBus.php
- getBusType.php
- getPassenger.php
- getPassengerSeat.php
- getPassengerSeatOnRouteInstance.php
- getProx.php
- getProxPass.php
- getRandomStop.php
- getSeatOnRouteInstance.php
- insertBoardsAt.php
- insertDisembarksAt.php
- insertInProximity.php
- insertPassengerSeat.php

- insertRouteInstance.php
- insertSeat.php
- verifyAdmin.php
- verifyBusDriver.php
- verifyEmp.php
- verifyEmpSeats.php
- verifyIDs.php
- verifyPassengerSeat.php
- verifyRoute.php

To set up the website/API, you will need to do the following:

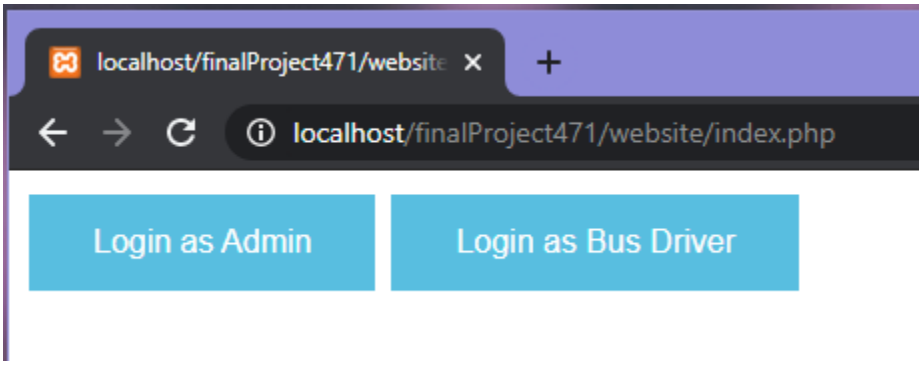
1. Host the folder 'finalProject471' at the root of your localhost using a webserver like Apache, and ensure you have PHP installed (my version was PHP 7.4.13). This folder should contain a folder called 'website', and inside that folder should be all of the .php files listed above.
2. Run 'DatabaseCreationAndData.sql' on your localhost using mySQL version 8 or above (the file was exported from version 8.019 using the engine InnoDB, so we can not guarantee that it will work on older versions of mySQL). This will create a new database called '471Project' at the root of your localhost.
3. Finally, go into the file 'config.php' and set your localhost username and password for the database. 'DB_USERNAME' is the username (usually root) and 'DB_PASSWORD' is the password.

```php
<?php
/* Database credentials. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'root');
define('DB_PASSWORD', 'root');
define('DB_NAME', '471project');

?>
```

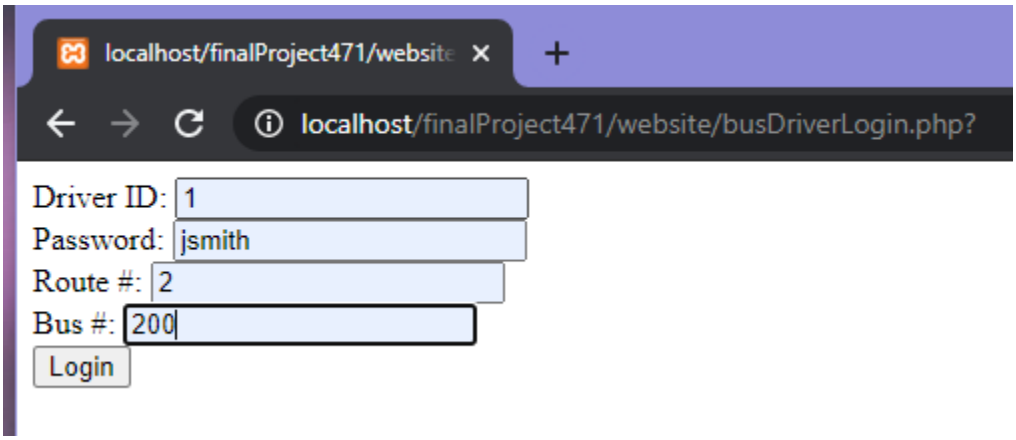
To start the website, open your browser and navigate to "http://localhost/finalProject471/website/index.php"

This will bring you to the following page, where you can choose to login as an administrator or a bus driver:

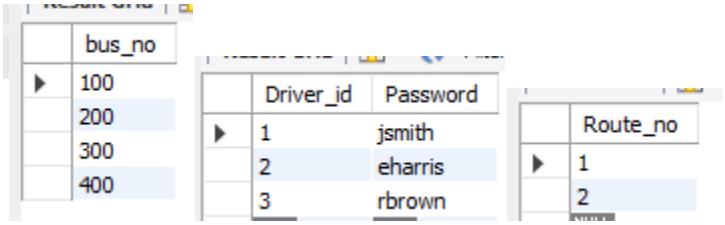


Bus Driver View Guide:

Select 'Login as Bus Driver'. This will direct you to the following page, where you can login with a valid combination of a driver id, password, route number and bus number (which is 'Vehicle_id' in our relational model, but is renamed here):



For the sample data given, here are a valid list of routes, bus driver logins, and bus numbers:

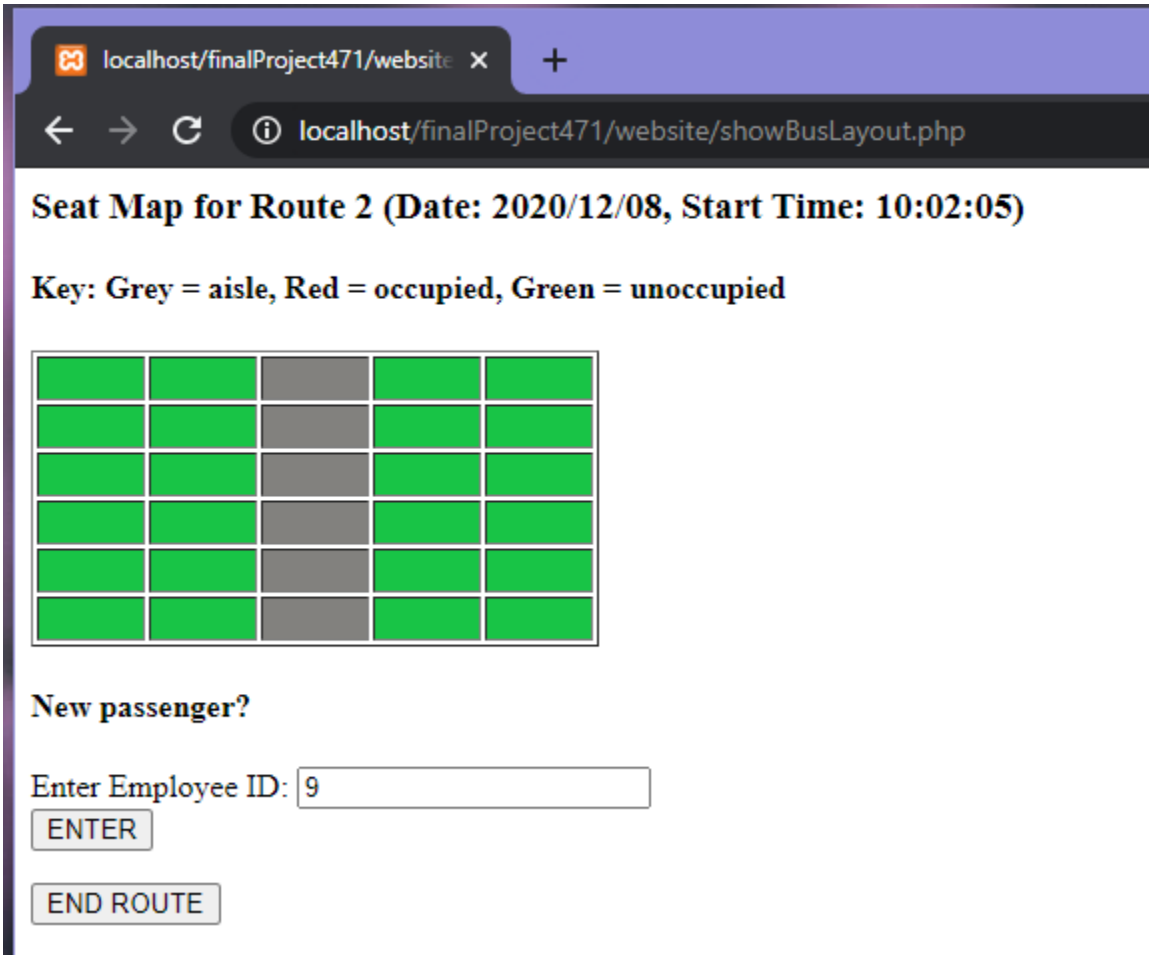


bus_no
100
200
300
400

Driver_id	Password
1	jsmith
2	eharris
3	rbrown

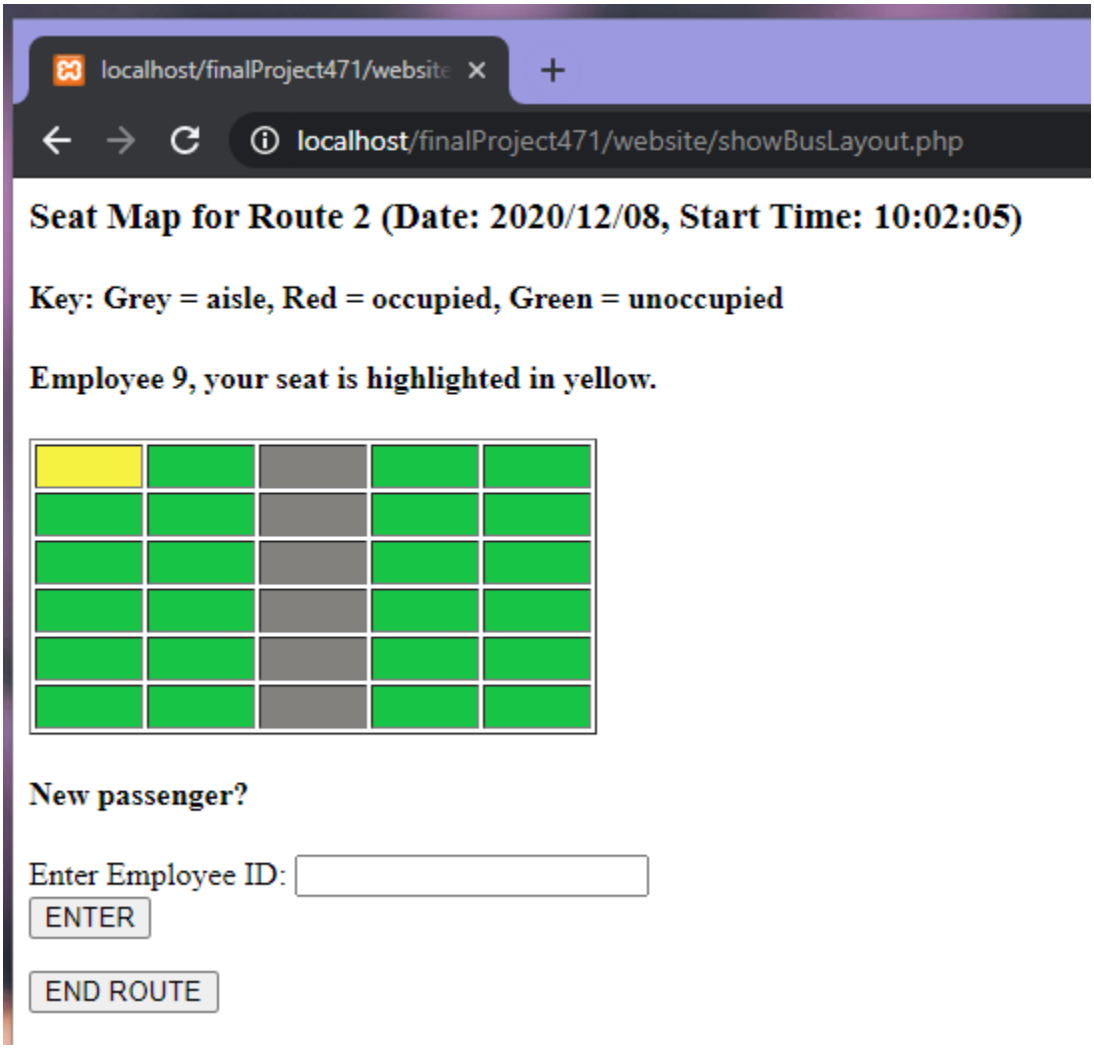
Route_no
1
2

Click ‘Login’. If you entered valid information, this will create a new route instance and associated seats for the current date and time, and redirect you to the following page:

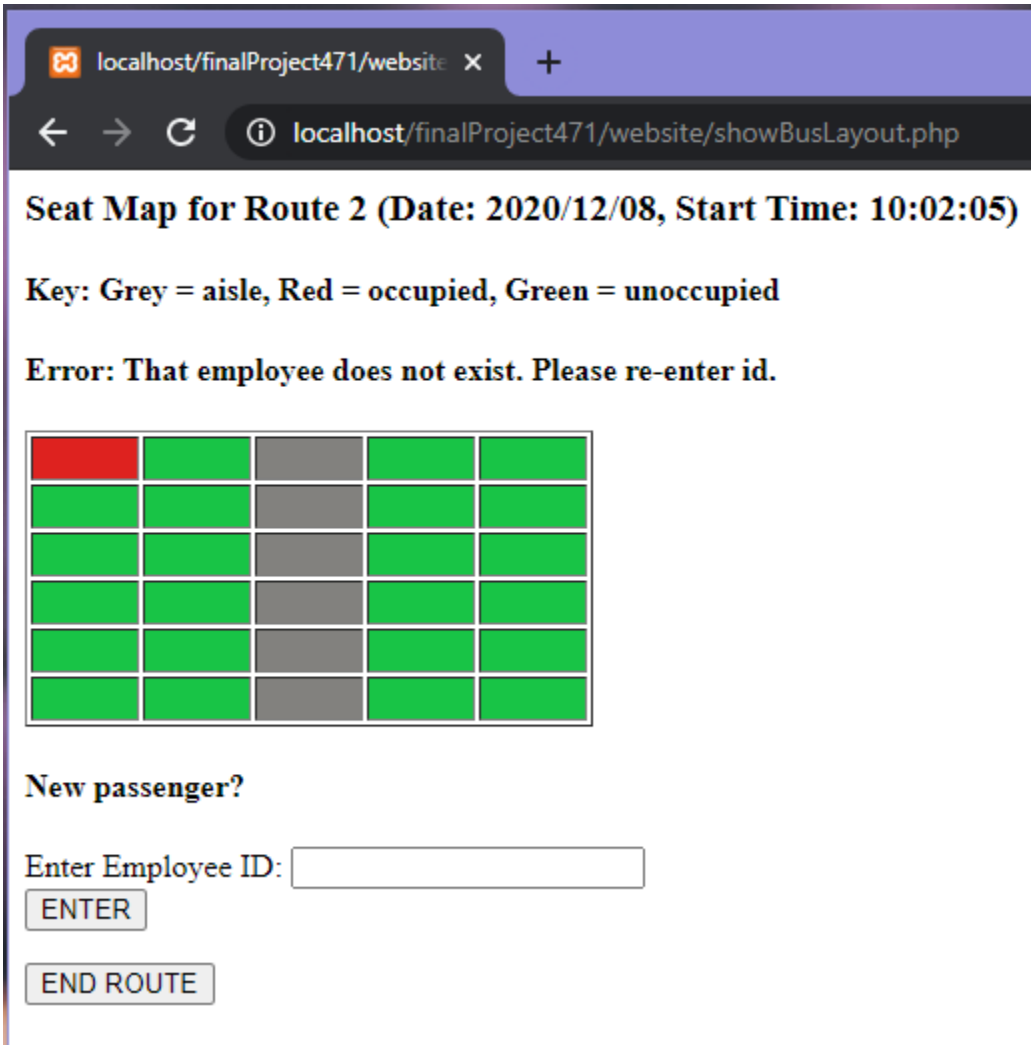


This displays a map of occupied and available seats on the bus. To add an employee to the bus, enter their employee id into the form and click ‘ENTER’. Valid employee ids for the sample data provided are between 1 and 18, inclusive.

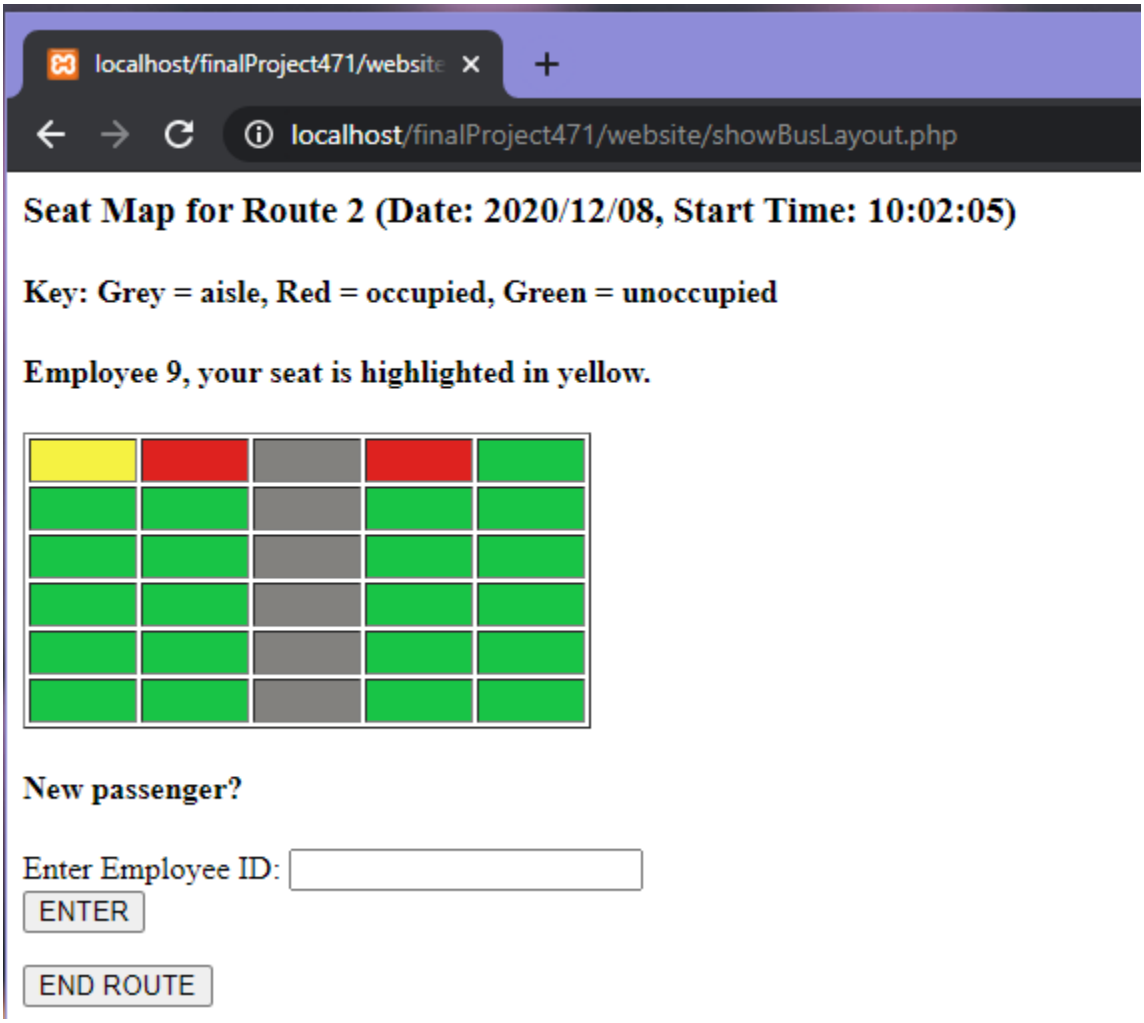
This will add the employee to the bus if they were not already on the bus, and set their boarding location, and boarding time to the current system time. It will also display where their assigned seat is in yellow, like so:



If you try to enter an invalid id, the following is displayed:



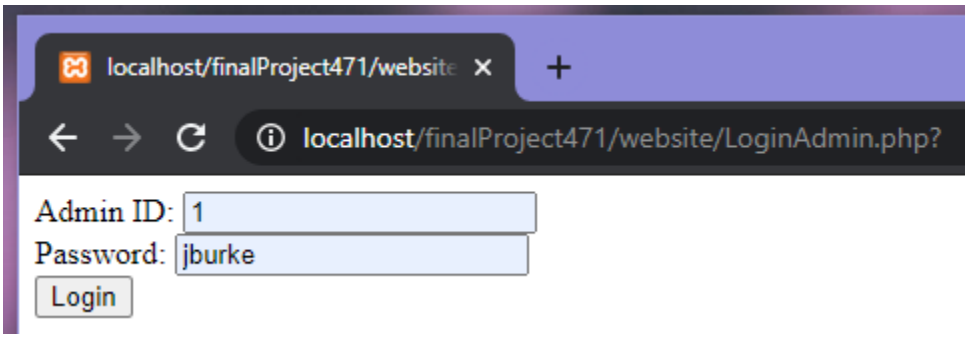
After adding a few more employees, if you accidentally enter a duplicate, it will just show you where their seat was originally, like so:



Finally, to end the route and return to the main page, click 'END ROUTE'. This will set the disembark location and time for all passengers on the current route, using the current system time.

Administrator View:

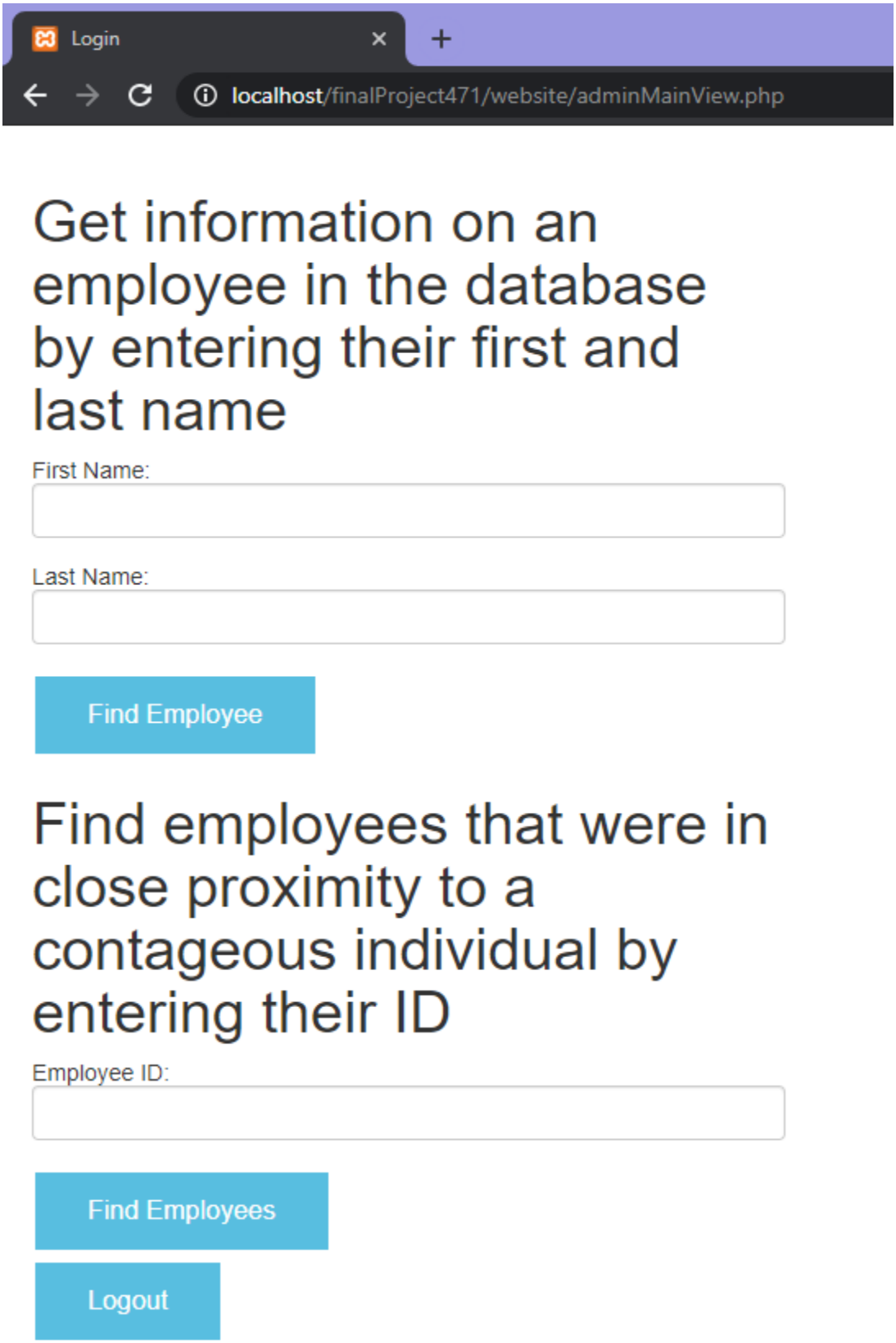
Select 'Login as Admin' from the main page. This will direct you to the following page, where you can login with a valid combination of an admin id and password:



For the sample data given, here is a valid list of admin logins:

Admin_id	Password
1	jburke
2	mhudson

Click ‘Login’. This will redirect you to the following page if you entered a valid login:

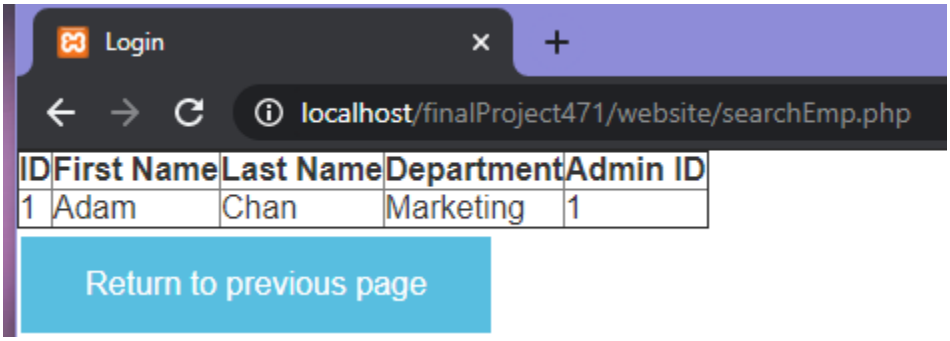


To search for an employee by name, enter their first and last name into the boxes and select 'Find Employee'. Note that it must be the full name and not a portion of the name.

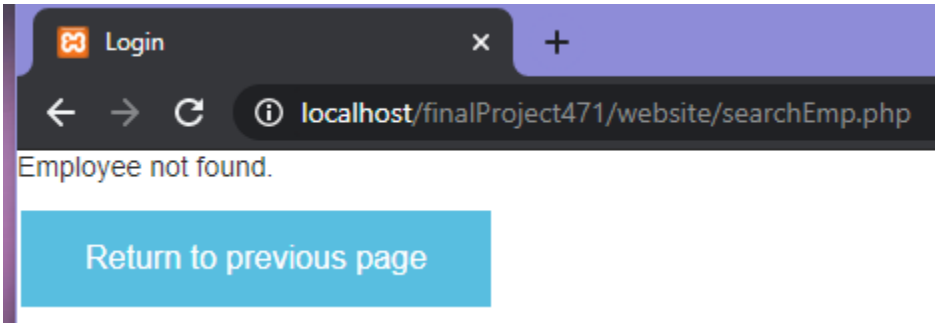
Here is a list of valid employee first and last name combinations from the sample data:

First_name	Last_name
Adam	Chan
Rebecca	Moore
Katie	Fox
Harry	Lewis
Chris	Ford
Charles	Scott
Linda	White
Max	Hill
Jemma	Shepard
Mitchel	Davis
Clint	Slater
Ruth	Bauer
Sienna	Cassidy
Stan	Hastings
Cody	Burton
Lucia	Kane
Tyler	Curtis
Anna	Piper

If an employee is found with that name combination, it will display their information, like so:

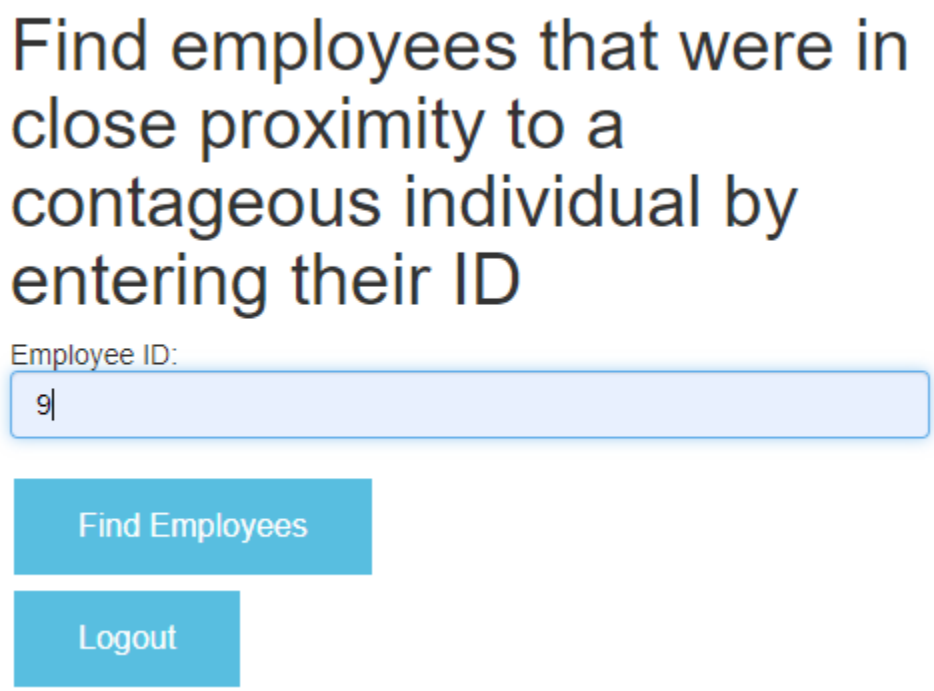


If an employee is not found, the following will be displayed:

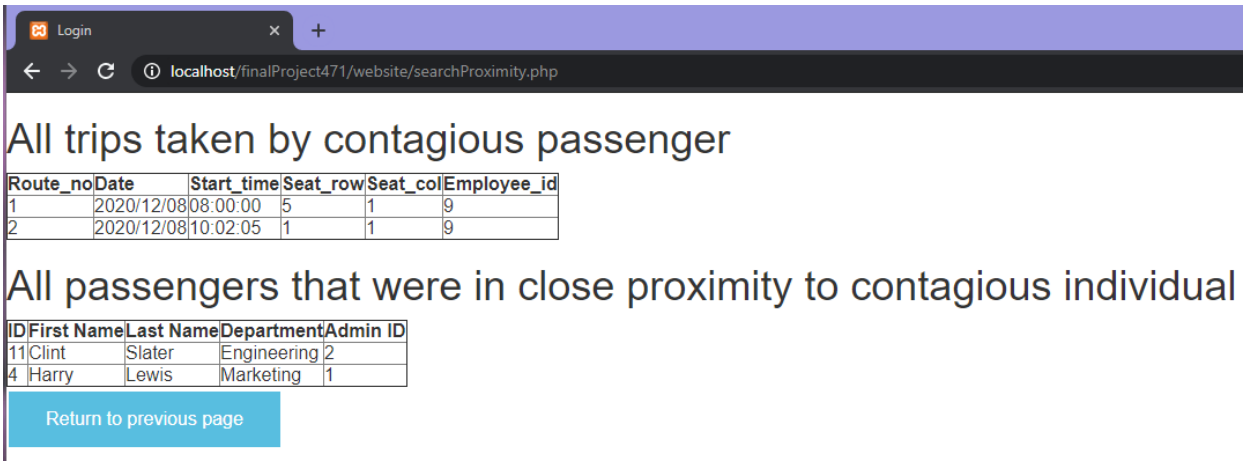


Clicking 'Return to previous page' will return you to the main view.

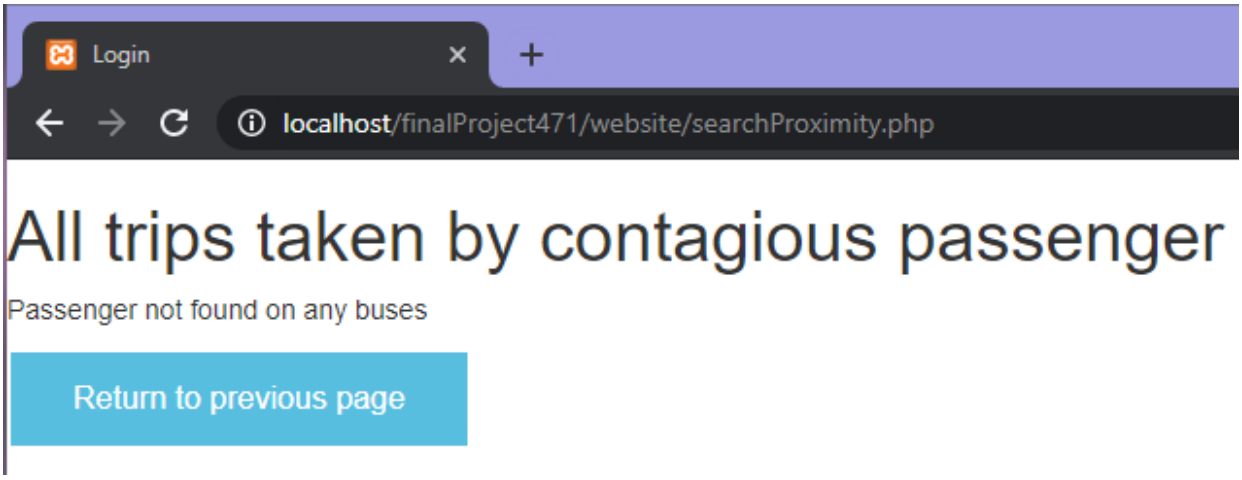
Back on the main view, to find what employees were in close contact with a particular employee, enter the employee id of an 'infected' employee into the Employee ID field like so, and click 'Find Employees':



This will display information about the trips that the passenger took and any employees they were in close proximity to like so. Note that there is a proximity entry for employee 4 here because we used employee 4 in the earlier example to fill up the seat map, for the trip taken at 10:02:05.



If you enter an invalid employee id, or that employee has not taken any trips, the following will be displayed:



Selecting 'Return to previous page' will return you to the administrator main view.

Finally, back on the administrator main view, selecting 'Logout' on the bottom of the screen will return you to this screen: