

CPS630 Iteration 1 & 2 Technical Report

P2S ONLINE SYSTEM

CPS630 - Winter 2021

Group 12

Nikan Afshar - 500807513

Mohamed Elbadry - 500860249

Table of Contents

Introduction	3
UI Design	3
Ride Services	3
Delivery Services	4
Checkout	6
Login / Register	6
About Us	7
Contact Us	7
DB Maintain	8
Technologies Used	9
Database Design	10
User	10
Car	12
Order	13
OrderProductMap	14
Coffee	14
Flower	14
Payment	15
Trip	16
Database ER Diagram	17
MVC Design Pattern - Component Architecture	17
System Architecture (Class Diagram)	18
Workload Distribution	18

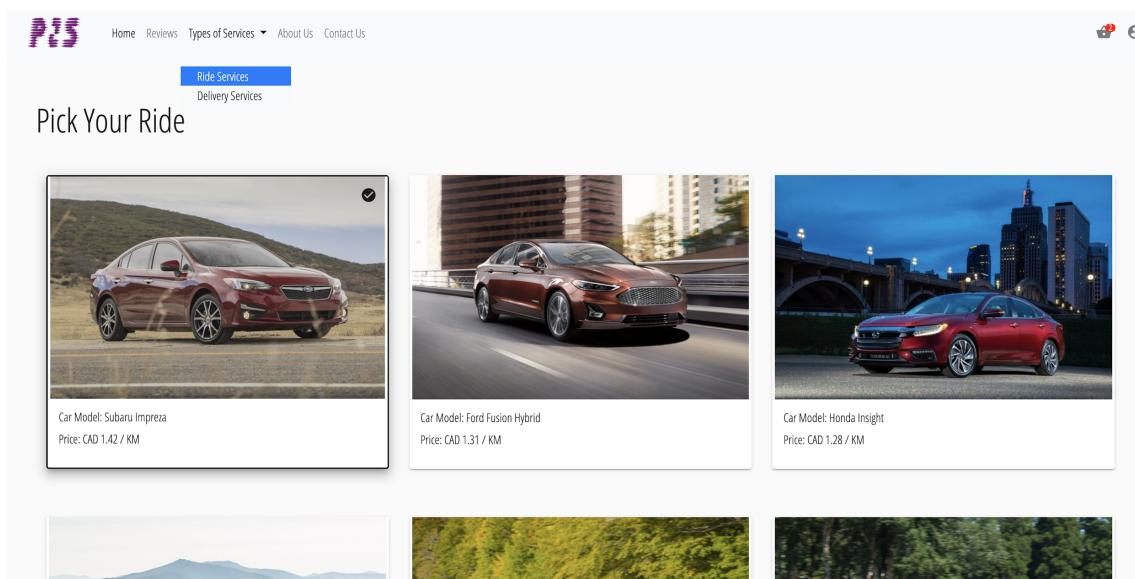
Introduction

As part of this project for our CPS630 term project we are building a Web Application that allows customers to create accounts on our system, login and then book rides or order delivery online. We then use the google Geolocation API to display the directions from source to destination, give them a price estimate and after choosing the product/car model they will be able to place the order in our shopping cart using HTML5 and JS drag and drop functionality. Currently however our system only supports florist and coffee shops as per the requirements.

UI Design

Ride Services

Using our system users can book their rides online. First they need to pick the car model and after they can pick their source and destination, pick up and drop off time and finally review the order and if everything is good they will be able to drag and drop the order review card into the shopping card.



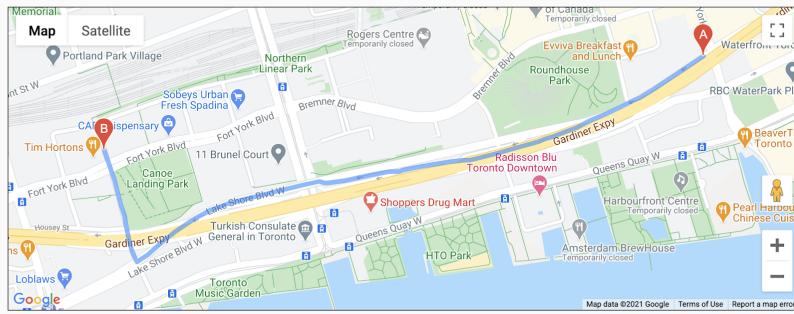
Source & Destination

Source
12 York Street, Toronto, ON, Canada

Destination
151 Dan Leckie Way, Toronto, ON, Canada

GET ESTIMATE

Distance:	1.7 km
Duration:	3 mins
Total Estimate:	CAD 7.17



Schedule Trip

Choose a date
3/26/2021

Choose a time
12:55 PM

REVIEW ORDER

Order Review

Car Details

Car Model	Car Colour
Ford Fusion Hybrid	Beige

Pickup Time

Date	Time
Mar 26, 2021	12:55

Fees

Ride Fare	CAD 7.17
Driver Tip (5%)	CAD 0.36
HST	CAD 1.10
Total	CAD 7.17

Delivery Services

Our system also allows users to get their favorite items delivered to their doors by choosing a destination and a dropoff time. At the moment the system only supports two stores:

- A florist store
- A coffee shop



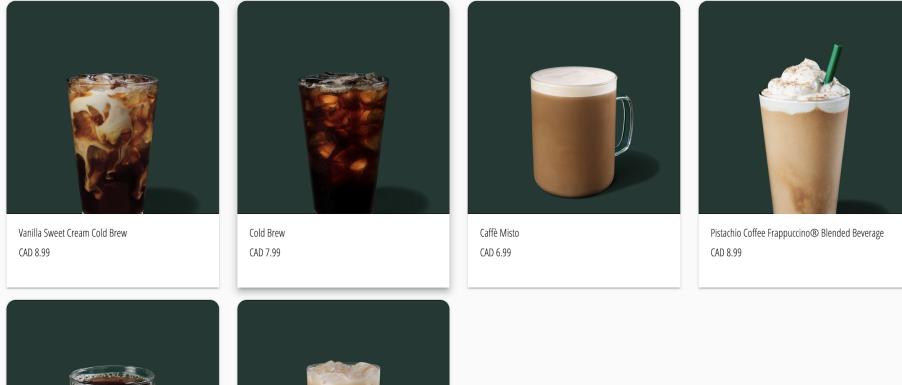
[Home](#) [Reviews](#) [Types of Services](#) [About Us](#) [Contact Us](#)

Featured Stores



After picking the store, users will be able to view the products in that store.

Coffee Store



After picking the product they will be able to input their destination and drop off date and time. Finally they will be able to review the order and if everything looks good they will be able to drag the order and drop it into the shopping cart.

Schedule Delivery

Destination: 12315 Coleraine Drive, Bolton, ON, C

Choose a date: 3/10/2021

Choose a time: 07:50 PM

REVIEW ORDER

Map | Satellite

Map data ©2021 Google | Terms of Use | Report a map error

Order Review

Order Items

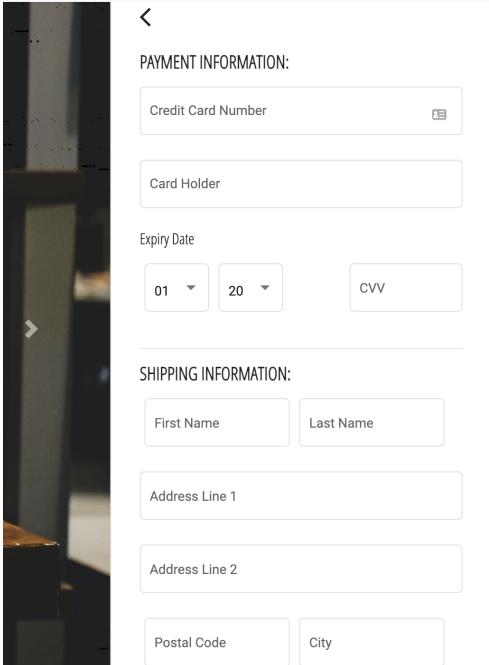
Cold Brew S	CAD 7.99 X
---------------	------------

Delivery Fees

Delivery Fee	CAD 66.53
HST	CAD 11.42
Total	CAD 85.94

Checkout

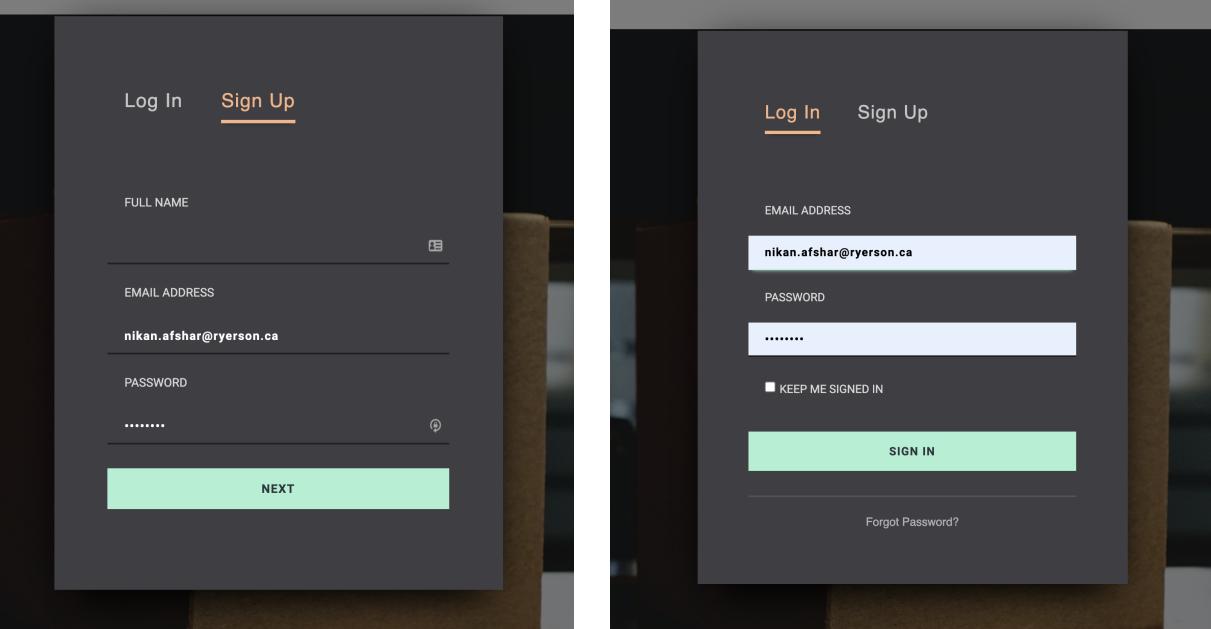
After adding all of the items to the cart our users can checkout by inputting their billing and shipping information. If they input their information correctly they will receive a confirmation message indicating that their order has been received.



A screenshot of a mobile application's payment and shipping information screen. At the top, there is a back arrow icon. Below it, the "PAYMENT INFORMATION:" section contains fields for "Credit Card Number" and "Card Holder". Underneath these are dropdown menus for "Expiry Date" (set to 01/20) and a "CVV" field. A large, dark rectangular redaction box covers the middle portion of the screen. Below the payment section is the "SHIPPING INFORMATION:" section, which includes fields for "First Name" and "Last Name", "Address Line 1" and "Address Line 2", "Postal Code", and "City".

Login / Register

Our system also allows users to sign up on our system and later they can log into the system. In future releases we are going to use this feature to load the past orders of our users.



Two screenshots of a mobile application's login/register screen. Both screenshots show a dark-themed interface with a navigation bar at the top containing "About Us" and "Contact Us" links. The first screenshot shows the "Sign Up" tab selected. It includes fields for "FULL NAME" (with a placeholder icon), "EMAIL ADDRESS" (containing "nikan.afshar@ryerson.ca"), and "PASSWORD" (containing "*****"). A green "NEXT" button is at the bottom. The second screenshot shows the "Log In" tab selected. It includes fields for "EMAIL ADDRESS" (containing "nikan.afshar@ryerson.ca"), "PASSWORD" (containing "*****"), a "KEEP ME SIGNED IN" checkbox, and a green "SIGN IN" button. Below the "SIGN IN" button is a "Forgot Password?" link.

About Us

Additionally we have developed an about us page displaying our information and what we stand for.

Our Team

Nikan Afshar
500807513



4th year computer science co-op student with interests in full-stack development and data science.

Mohamed Elbadry
500860249



4th year Computer Science Student at Ryerson University with interest in Software/Web development

"Plan for Smart Services" (P2S) Web-Application.

P2S is an online system that aims to plan for smart green trips inside the city and its neighborhood through sharing vehicles (like Uber). Considering the traffic as a serious threat to the quality of life these years, the world has been looking for various solutions to decrease the stress, frustration, delays and terrible air pollutions being caused through it. P2S attempts to provide a smart green solution on this regard by matching up drivers who live, work, and finally drive in the same neighborhood and would like to provide trio services.

Contact Us

We have developed a contact us screen that would allow the users to reach out to us in case of any technical or non-technical problem.

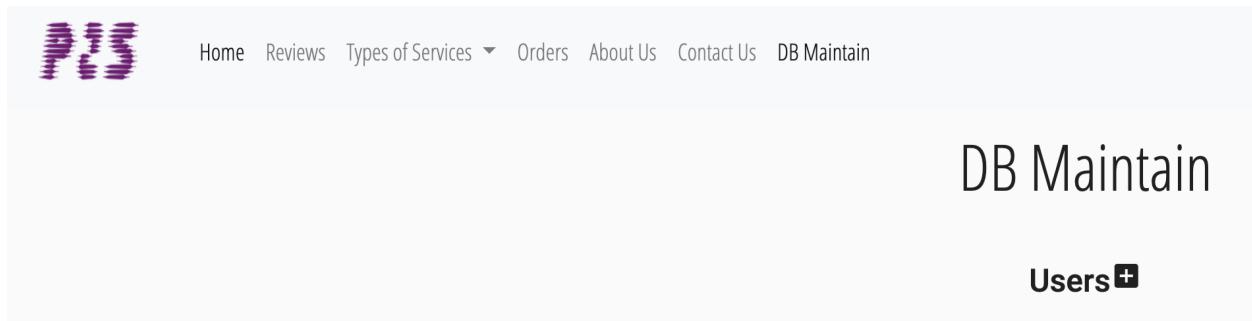
LET'S GET IN TOUCH

Message

DB Maintain

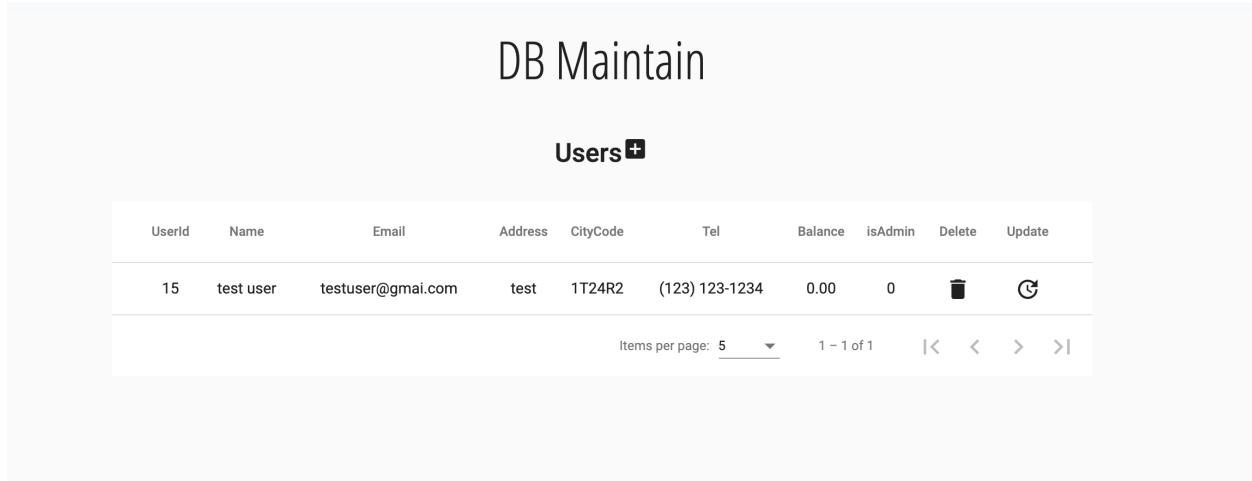
We have also developed the DB Maintain component for our system admins. At the moment only the User table is manageable from this component. Once a user logs in we check whether they are an admin or not by checking for the value stored in isAdmin flag. If admin they will be able to see the “db maintain” item on the navbar.

```
<li class="nav-item" *ngIf="user.isAdmin == 1">
    <a class="nav-link" [routerLink]=["dbMaintain"]>
        DB Maintain
    </a>
</li>
```

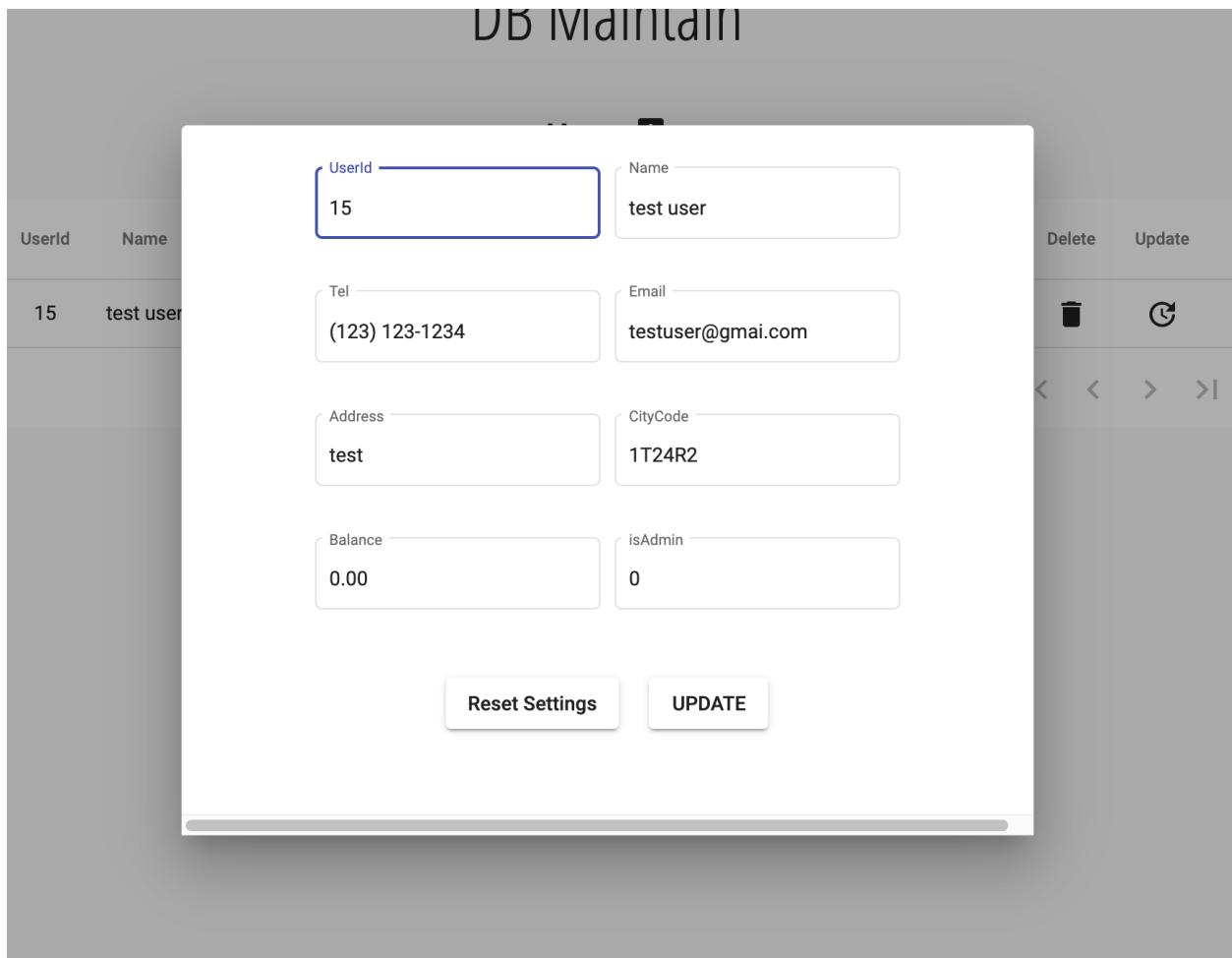


The screenshot shows the PES website's navigation bar at the top, featuring links for Home, Reviews, Types of Services (with a dropdown arrow), Orders, About Us, Contact Us, and DB Maintain. Below the navigation, the page title "DB Maintain" is centered. Underneath the title, there is a button labeled "Users" with a plus sign icon. The main content area is currently empty, showing only the page title and the "Users" button.

Using the DB Maintain feature users will be able to insert/delete/update records on our database system without the need of writing any SQL queries.



The screenshot shows the "DB Maintain" component's interface. The title "DB Maintain" is at the top, followed by a "Users" button with a plus sign. Below is a table with columns: UserId, Name, Email, Address, CityCode, Tel, Balance, isAdmin, Delete, and Update. A single row is visible, representing a user with ID 15, name "test user", email "testuser@gmai.com", address "test", city code "1T24R2", phone "(123) 123-1234", balance "0.00", isAdmin "0", and edit/delete icons. At the bottom, there is a pagination bar with "Items per page: 5" and navigation arrows.



Technologies Used

In order to develop our web application we used the following technologies:

- Backend:
 - Object oriented PHP
 - MySQL
- Frontend:
 - Bootstrap
 - Angular
 - HTML
 - CSS
 - Font Awesome
 - Google Fonts

- Google Geolocation API
- Angular Material Library (used for the styling of our form input fields)

Database Design

We use the following tables to store our data in our MySql database:

- User

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	UserId 📃	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	Name	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	Tel	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
4	Email	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
5	Address	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
6	CityCode	varchar(10)	utf8mb4_general_ci		Yes	NULL			Change Drop More
7	Password	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More
8	Balance	decimal(6,2)			Yes	0.00			Change Drop More
9	isAdmin	tinyint(1)			No	None			Change Drop More

When users sign up on our system we use the information inputted by the user to register the user in our system. After data validation, we make a post request to our PHP API and insert the record into our database.

```
/**
 * Takes an object with fullName, email and password as parameters and makes a post
request to register.php endpoint to register a new user.
 * @param param0
 */
registerUser({fullName, email, password, phoneNumber, address, postal}) {
  return this.httpClient.post(this.baseUrl + '/user/register.php', {'fullName':
  fullName, 'email': email, 'password': password, 'phoneNumber': phoneNumber, 'address':
  address, 'postal': postal});
}
```

The PHP backend will ensure the POST data is set and then instantiate the User Object, set the properties and call the register method to insert into the database.

```

<?php

    header("Access-Control-Allow-Origin: *");
    header("Access-Control-Allow-Methods: PUT, GET, POST, DELETE");
    header("Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type,
Accept");

    include_once '../config/database.php';
    include_once '../objects/user.php';

    $postdata = file_get_contents("php://input");
    $request = json_decode($postdata);

    $database = new Database();
    $db = $database->getConnection();

    $user = new User($db);

    if(isset($postdata) && !empty($postdata)) {
        $user->Name = $request->fullName;
        $user->Email = $request->email;
        $user->Password = $request->password;
        $user->Tel = $request->phoneNumber;
        $user->Address = $request->address;
        $user->CityCode = $request->postal;

        if ($response = $user->register()) {
            http_response_code(200);
            echo json_encode(array("message" => "User was created."));
        } else {
            http_response_code(400);
            echo json_encode(array("message" => "User was not created."));
        }
    }
?>

```

By default all users are not admin (isAdmin = false) and Balance is set to 0 (Balance = 0). The UserId column is an auto increment column (i.e. the ID will get incremented by 1 automatically after each insert).

PHP API code for insert into the User table:

```
public function register() {
    $query = "INSERT INTO User(Name, Tel, Email, Address, CityCode, Password)
VALUES ('$this->Name', '$this->Tel', '$this->Email', '$this->Address',
'$this->CityCode', '$this->Password')";

    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt;
}
```

- Car

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	CarId 	int(11)		No	None			AUTO_INCREMENT	 Change  Drop  More
2	CarModel	varchar(255)	utf8mb4_general_ci	Yes	NULL				 Change  Drop  More
3	CarCode	varchar(100)	utf8mb4_general_ci	Yes	NULL				 Change  Drop  More
4	AvailabilityCode	varchar(100)	utf8mb4_general_ci	Yes	NULL				 Change  Drop  More
5	CarColour	varchar(255)	utf8mb4_general_ci	Yes	NULL				 Change  Drop  More
6	ImageURL	varchar(255)	utf8mb4_general_ci	Yes	NULL				 Change  Drop  More
7	CarPrice	decimal(6,2)		Yes	NULL				 Change  Drop  More

The car table is used to display the car models available for ride services. CarId is the PRIMARY KEY and set to AUTO_INCREMENT. When the Car component on our angular app is loaded the car records will be requested and shown in the component.

```
/*
 * Returns the cars that we have for ride services.
 */
getCars() {
    return this.httpClient.get(this.baseUrl + '/car/read.php');
}
```

```
<?php

class Car {
```

```

private $conn;
private $tblName = "Car";

public $CarId;
public $CarModel;
public $CarCode;
public $AvailabilityCode;
public $CarColour;
public $ImageURL;
public $CarPrice;

public function __construct($db) {
    $this->conn = $db;
}

public function read() {
    $query = "SELECT
        * FROM
        " . $this->tblName;

    $stmt = $this->conn->prepare($query);
    $stmt->execute();
    return $stmt;
}

?>

```

- Order

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	OrderId 📄	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	UserId	int(11)			No	None			Change Drop More
3	OrderTotal	varchar(10)	utf8mb4_general_ci		No	None			Change Drop More
4	Timestamp	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More

Order component is used to store the orders placed by our users. OrderId is used as PRIMARY KEY and is set to AUTO_INCREMENT. We use UserId as FOREIGN KEY to link the table to the USER table. For every order, we only create a single record in the ORDER table even if the user orders multiple products. To link the ORDER table to the PRODUCT table we use the OrderProductMap table.

- OrderProductMap

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
□ 1	OrderProductId 	int(11)			No	None		AUTO_INCREMENT	 Change  Drop  More
□ 2	OrderId	int(11)			No	None			 Change  Drop  More
□ 3	ProductId	int(11)			No	None			 Change  Drop  More

OrderProductId is the PRIMARY KEY of the table and we use ProductId and OrderId to map the ORDER table to the FLOWER/COFFEE table.

- Coffee

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
□ 1	ProductId 	int(11)			No	None		AUTO_INCREMENT	 Change  Drop  More
□ 2	Name	varchar(255)	utf8mb4_general_ci		No	None			 Change  Drop  More
□ 3	Price	varchar(10)	utf8mb4_general_ci		No	None			 Change  Drop  More
□ 4	ImageURL	varchar(255)	utf8mb4_general_ci		No	None			 Change  Drop  More
□ 5	StoreCode	varchar(10)	utf8mb4_general_ci		No	None			 Change  Drop  More

The coffee table is used to store the coffee products that we display in our Delivery Services component of our application. ProductId is set to PRIMARY KEY. ImageURL field is used to store the image URL used on our website.

- Flower

The FLOWER table is used to store the flower products that we display in our Delivery Services component of our application. ProductId is set to PRIMARY KEY. ImageURL field is used to store the image URL used on our website.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
□ 1	ProductId 	int(11)			No	None		AUTO_INCREMENT	 Change  Drop  More
□ 2	StoreCode	varchar(6)	utf8mb4_general_ci		No	None			 Change  Drop  More
□ 3	Name	varchar(255)	utf8mb4_general_ci		No	None			 Change  Drop  More
□ 4	Price	varchar(10)	utf8mb4_general_ci		No	None			 Change  Drop  More
□ 5	ImageURL	varchar(255)	utf8mb4_general_ci		No	None			 Change  Drop  More

- Payment

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	PaymentId ⚒	int(11)			No	None		AUTO_INCREMENT	Change Drop ▾ More
2	OrderId	int(11)			Yes	NULL			Change Drop ▾ More
3	CardNumber	varchar(20)	utf8mb4_general_ci		No	None			Change Drop ▾ More
4	ExpiryMonth	varchar(10)	utf8mb4_general_ci		No	None			Change Drop ▾ More
5	ExpiryYear	varchar(10)	utf8mb4_general_ci		No	None			Change Drop ▾ More
6	CardHolderLastName	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
7	CardAddressLine1	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
8	CardAddressLine2	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop ▾ More
9	PostalCode	varchar(30)	utf8mb4_general_ci		No	None			Change Drop ▾ More
10	City	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
11	StateOrProvince	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
12	Country	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
13	CardHolderFirstName	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More

Payment table is used to store the users payment information when they create a new order on our system. PaymentId is set as the PRIMARY KEY of this table. We store the payment information to be displayed on the checkout component when they log into the system. So that they don't have to input their payment information twice (to be implemented in future iterations).

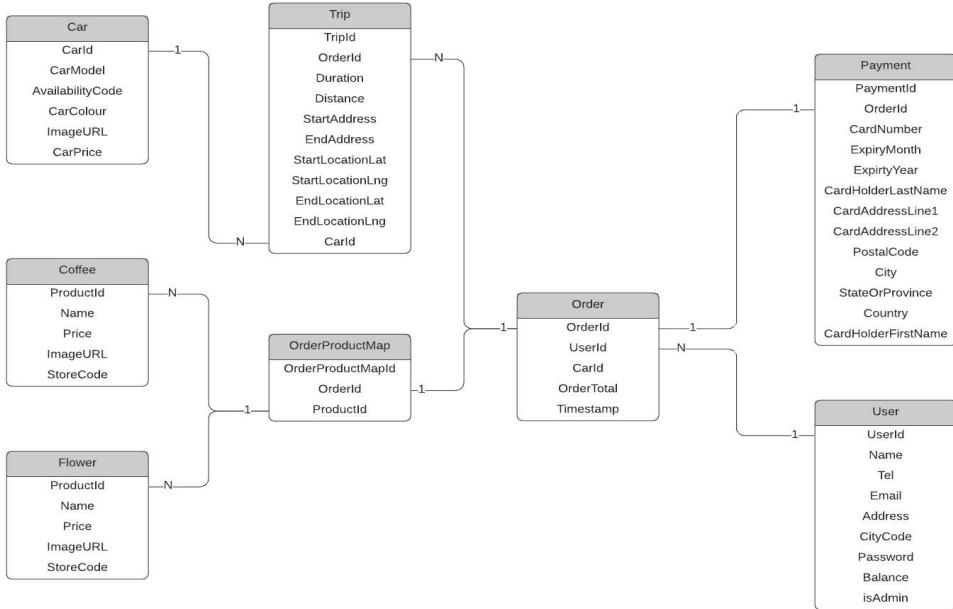
```
if ($response = $order->create()) {
    $payment->OrderId = $db->lastInsertId();
    $trip->OrderId = $payment->OrderId;
    $orderProductMap->OrderId = $payment->OrderId;
    if ($response = $payment->create()) {
        if ($response = $trip->create()) {
            if ($response = $orderProductMap->create()) {
                http_response_code(200);
                echo json_encode(array("message" => "Order was created."));
            } else {
                http_response_code(400);
                echo json_encode(array("message" => "Order was not created."));
            }
        }
    }
}
```

- Trip

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	TripId 🚗	int(11)			No	None		AUTO_INCREMENT	Change Drop ▾ More
2	OrderId	int(11)			No	None			Change Drop ▾ More
3	Duration	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
4	Distance	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
5	StartAddress	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
6	EndAddress	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
7	StartLocationLat	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
8	StartLocationLng	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
9	EndLocationLat	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More
10	EndLocationLng	varchar(255)	utf8mb4_general_ci		No	None			Change Drop ▾ More

The TRIP table is used to record the trip information for our users. TripId is set as the PRIMARY KEY of the table and OrderId and FOREIGN KEY to link it to the ORDER table. Each order could be made up of 1 or more trips. We store the start location / coordinates as well as the address / coordinates of the destination. On top of that we store the duration and the distance of the trip.

- Database ER Diagram



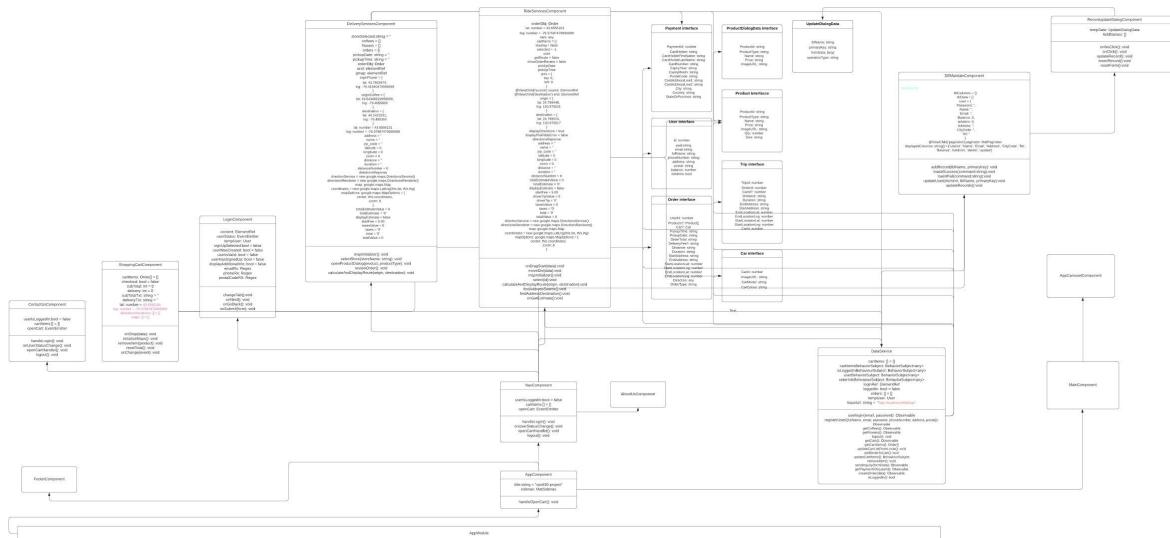
MVC Design Pattern - Component Architecture

Angular makes use of MVVM design patterns. MVVM design pattern is the refinement of the MVC design pattern and facilitates the separation of the development and the graphical user interface. MVVM design pattern includes 3 main features:

1. Model: is the data that is represented in the HTML part.
2. View: is the HTML part that displays the data.
3. View-Model: is the part that handles the flow from model to the view. In Angular that is done through data binding between the TypeScript files and the HTML files.

System Architecture (Class Diagram)

Note: The class diagram image with higher resolution has been submitted as well.



Workload Distribution

Group Member	Percentage of the total work completed
Nikan Afshar	50%
Mohamed Elbadry	50%