CPS630 Iteration 3 & 4 Technical Report

P2S ONLINE SYSTEM

CPS630 - Winter 2021

Group 27

Nikan Afshar - 500807513

Mohamed Elbadry - 500860249

Table of Contents

Introduction **3**

A. Security Issues Detected (Part A) **3**

Hash credentials using 'SHA1' with salt: 3

Keep track of user login attempts to detect suspicious activity: 4

Redirect the user to login on page refresh: 6

"Keep me signed in" feature implemented using browser cookies: 7

Validate credit card information on checkout: 9

Mask credit card number before storing in database: 10

Prevent user from registering if the email address entered already exists in User table: 11

B. Service C and Shopping Cart Implementation (Part B - 2 & 3) **11**

Ride Green Service Component Design: 11

Shopping Cart Development: 13

B. Browsers Supported (Part B - 4) **14**

B. The New Demanding Service Added - Service D (Part B - 5) **16**

C. SPA Architecture (Part C - 1) **18**

Workload Distribution **20**

Introduction

As part of iteration 3 we improved upon the ‘maintain’ feature that was developed as part of iterations 1 & 2. With this improvement now users will be able to maintain all aspects of our database and UI. They will be able to add new products, change cars, drivers, addresses, payment information, user permissions and much more. Sign up feature was already implemented during previous iterations. However, we were able to improve upon this feature as well. To enhance the security of our system, users now have only 5 attempts and if they enter wrong credentials they will get blocked and only a system administrator will be able to unblock them. As part of iterations 1 & 2, we designed the system based on two types of users, ordinary users and admins. Admins are able to maintain the system while using all other features whereas ordinary users will not have access to the ‘dbmaintain’ feature. Furthermore, two additional services, service C (i.e. Ride Green) and service D (i.e. Ride Social) were added to the system. Additionally, we were able to implement the user reviews component to all users to leave reviews about our products and services. As part of this iteration, we added a mechanism that allows us to detect the user's browser and display it as a dialog. To implement all of these features we used Angular framework, Bootstrap for CSS styling and Google’s API to implement our map features.

A. Security Issues Detected (Part A)

During iterations 3 & 4 we were able to detect some security flaws in our design and rectify those issues. Here we will discuss a few of these issues detected and how we attempted to resolve them:

1. **Hash credentials using ‘SHA1’ with salt:**

In iterations 1 & 2, upon user registration we would store the user's password directly into the database. However, this is very insecure for a few reasons:

- If the database is compromised, the hacker will have access to all the passwords.
- Storing credentials using plaintext is prone to SQL injection attacks and attackers will be able to get passwords on the fly.

In order to rectify this security flaw, we used PHP md5 crypt hash function to hash our passwords using randomly generated salt values.

```
function getSalt() {
        $charset =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789/][{}\
;:?.>,<!@#$%^&*()-_=+|';
        $randStringLen = 64;
        $randString = "";
        for ($i = 0; $i < $randStringLen; $i++) {
            $randString .= $charset[mt_rand(0, strlen($charset) - 1)];
        }
        return $randString;
}
```

```
$user->Password = crypt($request->Password, $salt);
```

We then store this salt with each user record in our database.
On login we retrieve this salt value from the database and use it to encrypt the user password using the same md5 hash function, if the hashed value is the same as the hashed password stored on the database, the user credentials are correct and the user can login.

2. **Keep track of user login attempts to detect suspicious activity:**

In order to detect suspicious user activity we decided to keep track of user's login attempt activity. When users click the login button, in the PHP backend we retrieve the

Password	Balance	isAdmin	Salt
.=rYNuMOiUF4M	0.00	1	.=b5l^<Bg&s<cQ? TjYX1p7cisZF(L#3DbVpOk7.1(=#1n8Gz0...

entered email address. We fetch the user record by email. If the user exists we then

create a new record inside the LoginAttempt table. We store the user's IP address, entered username, login timestamp and a flag indicating whether the user entered the right password or not. Below you can see an example of this record.

+ Options

	LoginAttemptId	Email	IPAddress	InvalidCreds	Timestamp
Edit Copy Delete	1	nikan.afshar@ryerson.ca	127.0.0.1	0	2021-04-13 20:10:15
Edit Copy Delete	2	nikan.afshar@ryerson.ca	127.0.0.1	1	2021-04-13 20:10:25
Edit Copy Delete	3	nikan.afshar@ryerson.ca	127.0.0.1	1	2021-04-13 20:10:30
Edit Copy Delete	4	nikan.afshar@ryerson.ca	127.0.0.1	1	2021-04-13 20:10:33
Edit Copy Delete	5	nikan.afshar@ryerson.ca	127.0.0.1	1	2021-04-13 20:10:36
Edit Copy Delete	6	nikan.afshar@ryerson.ca	127.0.0.1	0	2021-04-13 20:10:39
Edit Copy Delete	7	nikan.afshar@ryerson.ca	127.0.0.1	1	2021-04-13 20:10:46
Edit Copy Delete	8	nikan.afshar@ryerson.ca	127.0.0.1	1	2021-04-13 20:10:48
Edit Copy Delete	9	nikan.afshar@ryerson.ca	127.0.0.1	0	2021-04-13 20:10:52
Edit Copy Delete	10	nikan.afshar@ryerson.ca	127.0.0.1	0	2021-04-13 20:10:57
Edit Copy Delete	11	nikan.afshar@ryerson.ca	127.0.0.1	0	2021-04-13 20:11:02

If the credentials are valid the user can login. If not we set the InvalidCreds flag in our database to 1. We then fetch the number of user login attempts within the past 15 minutes where InvalidCreds = 1 (meaning user entered wrong credentials) and then subtract that from 5. You can find the SQL query we are using below.

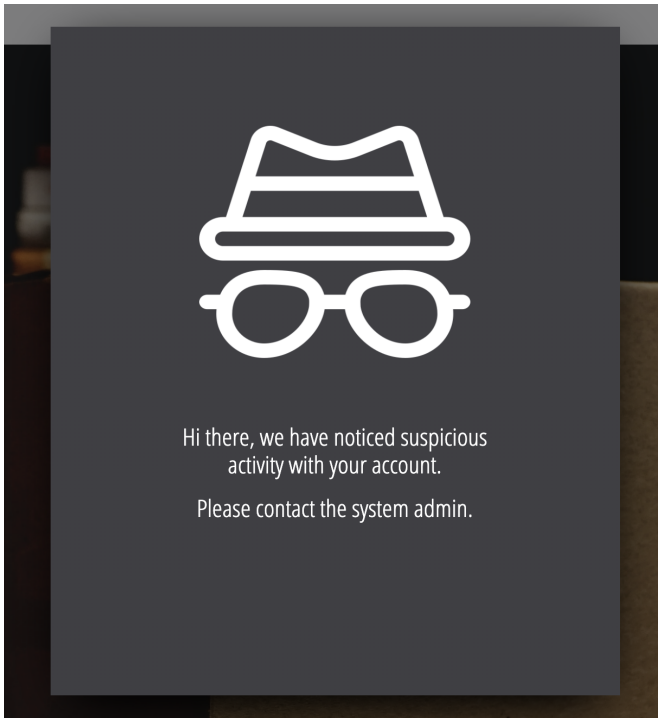
```
SELECT CASE WHEN 5 - COUNT(LoginAttemptId) <= 0 THEN 0 WHEN 5 - COUNT(LoginAttemptId)
> 0 THEN 5 - COUNT(LoginAttemptId) END AS RemainingAttempts FROM LoginAttempt WHERE
TImestamp >= CURRENT_TIMESTAMP - INTERVAL 5 MINUTE && Email = '$this->Email' &&
IPAddress = '$this->IPAddress' && InvalidCreds = 1
```

If the result is less than 0, then we block the user by setting the Blocked flag on our User table to 1 because that means the user from that particular IP Address has entered their credentials incorrectly 5 times in the past 15 minutes.

```
if ($remainingAttempts == '0') {
      $user->lockUser();
}
```

CityCode	Password	Balance	isAdmin	Salt	Token	Blocked
M5B3A8	.=rYNuMOiUF4M	0.00	1	.=b5l^<Bg&s<cQ? TjYX1p7cisZF(L#3DbVpOk7.1(=#1n8Gz0...	ba4ba51506c5e2a8799e65905bfa8fa5	1

On login we first check whether the user is blocked or not. If blocked the PHP code will exit and the Angular Client will check the HTTP request response and if blocked sets the isBlocked flag to true on the client side and prevent the user from logging in. If the user is blocked even if they enter the password correctly they will be prevented from logging in to the system and they will have to contact the system admin to unblock them. The admin will be able to directly unblock the user by setting Blocked flag to 0 on dbmaintain/usermaintain.

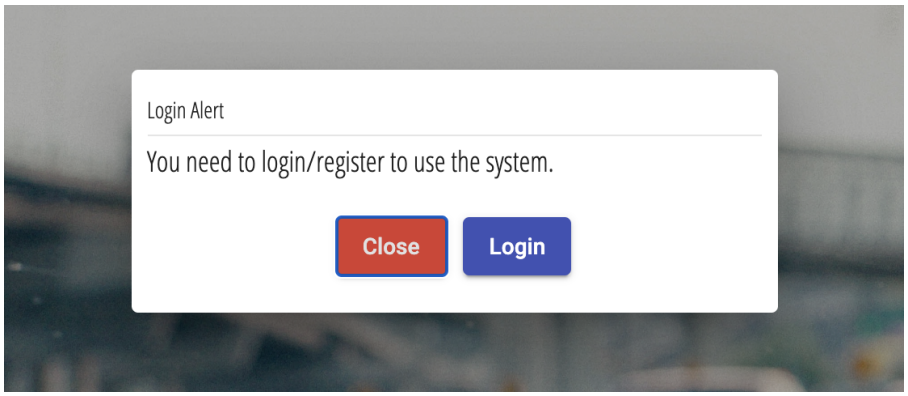



```
fail => {
  if (fail.error.message === 'Locked') {
    this.suspiciousActivity = true;
  } else if (fail.error.remainingAttempts) {
    this.remainingAttempts = fail.error.remainingAttempts;
  }
}
```

3. **Redirect the user to login on page refresh:**

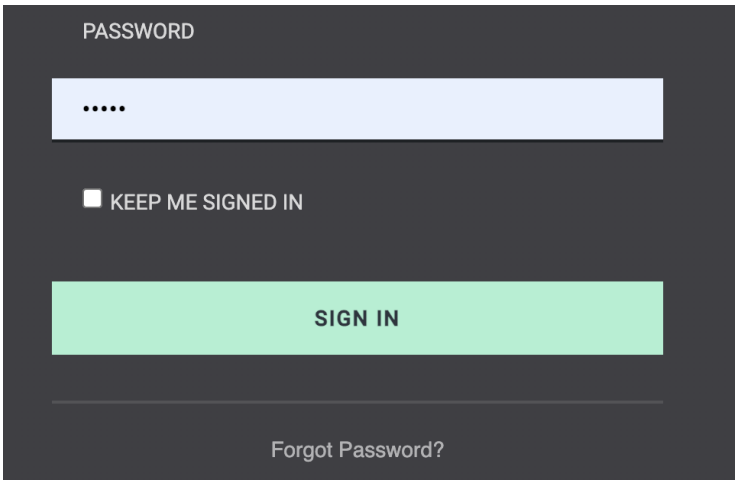
The way that we had implemented our design in iterations 1 & 2 would allow for users to access different pages of our application such as the service pages without having to login. However, as per the requirements of iterations 3 & 4 we had to prevent users from being able to access service pages or the reviews page if they are not logged in. To implement that functionality we used the Angular's OnInit lifecycle hook and on component initialization we check whether the user is logged in or has a cookie setup, if

so then we allow the user to access the service if not we redirect the user to the home page and open the login dialog.



4. **“Keep me signed in” feature implemented using browser cookies:**

This is a security feature that we intended to implement in iterations 1 & 2, however due to the lack of time we were not able to get it functional. However, during iterations 3 & 4, we were able to implement this functionality using cookies and Angular CookieService. First we bind the KEEP ME SIGNED IN input checkbox to a variable called keepMeLoggedIn which will be sent as part of the HTTP Login request to login.php in our backend. This flag is then retrieved in the backend if the flag is set to 1 then and the user is not blocked we then generate a random token (i.e. unique string value every time) we concatenate the token to the email value entered by the user to get the cookie and then use ‘SHA1’ hash function with user salt to create a hash of all these values and call it ‘mac’. We then concatenate this hashed value (i.e. mac) to the cookie. Additionally, we need to store the token on our database inside the Token field. Finally, this cookie will be sent as a response to the client side to be stored on the browser.



```
$token = generateRandomToken();
$cookie = $Email . ':' . $token;
$mac = hash_hmac('sha256', $cookie, $Salt);
$cookie .= ':' . $mac;
setcookie('rememberme', $cookie);
if ($token) {
  $user->Token = $token;
  $result = $user->storeToken();
}
```

lmin	Salt	Token
1	.=b5l^<Bg&s<cQ? TjYX1p7cisZF(L#3DbVpOk7.1(=#1n8Gz0...	90c48fd0f9137fc81bccaa4cf5f383ba

After the cookie is received, it will be stored on the browser as ‘rememberme’ using Angular’s CookieService. We also set the expiration time to 15 minutes to ensure it gets erased after 15 minutes.

```
if (this.keepMeLoggedIn) {
   const dateNow = new Date();
   dateNow.setMinutes(dateNow.getMinutes() + 10);
   this.cookieService.set('rememberme', response['records'][0].cookie,
dateNow);
}
```

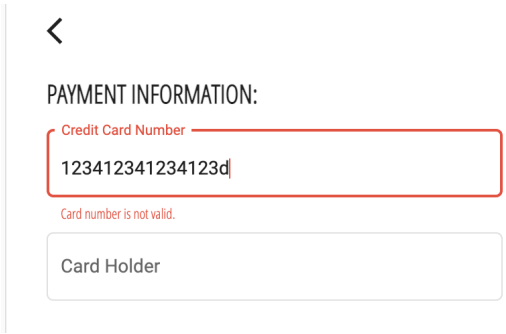
Name	Value	Domain	Path
1P_JAR	2021-04-06-02	.gstatic.com	/
rememberme	nikan.afshar%40ryerson.ca%3A99c5ab929efa00421ffd372d9d8fcb91%3A296...	localhost	/

Once the user refreshes the application, we check the browser storage to see if there is a cookie on the user's browser, if so we make a HTTP request to rememberUser.php to

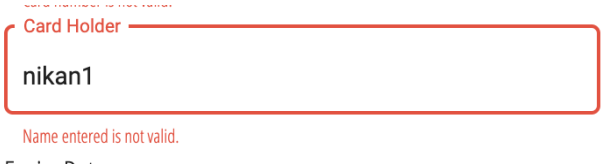
validate the user's cookie. Once we get the cookie in rememberUser.php we explode the string on ':', in order to separate the concatenated variables (Email, Token and Mac). Once we get the user email we check whether the user exists if so we get user's Salt and Token to be compared with the Token in the cookie. We also recalculate the mac using sha256 with user Salt and if the mac in the cookie and the one just calculated are equal we then check whether the user records Token and the token in the cookie are equal if so the user cookie is validated. We then generate a new token, calculate a new hash (i.e. mac and we return that alongside the user record to the client side and now the user is logged in. The new cookie returned will then replace the old one stored on the browser. This process is repeated every time the user refreshes the browser page to ensure the cookie is valid.

5. **Validate credit card information on checkout:**

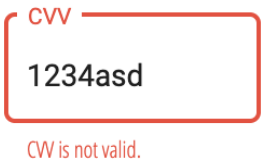
As part of our efforts to secure our system we decided to validate the user's credit card information before allowing the user to checkout in our system. To validate the credit card number we make sure all the characters are numbers. We then check the length and make sure it's not more than 16 characters long.



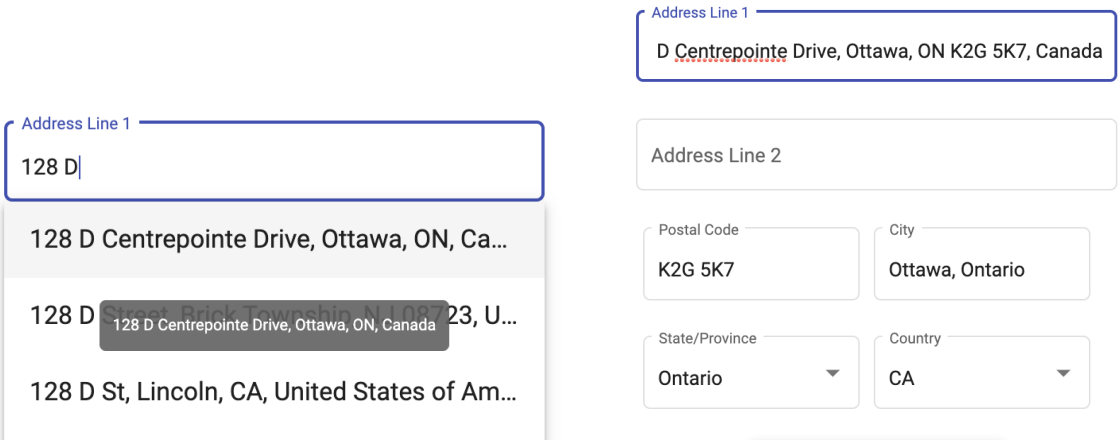
For Card Holder's First and Last name we ensure that the inputted values do not contain any digits.



For CVV we make sure that the input only contains characters and no digits and the length is not greater than 4 or less than 3.



For the shipping information we make use of geoapify api to get “place suggestions”. After the user inputs the first 3 characters of the address we make get request to geoapify api using Ajax and based on the place suggestions populate other fields.



6. Mask credit card number before storing in database:

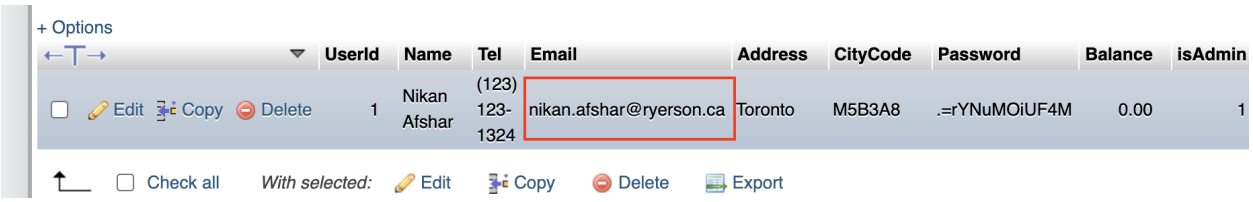
In addition to the previous step, in order to further secure our systems we decided to mask credit card information on the client side before sending the data to the backend php server. Storing credit card information is an extremely dangerous practice. If the database gets compromised hackers will have access to everyone’s credit card information. They may also be able to get access to credit card information by SQL injection and other techniques. Therefore, it is always recommended not to store credit card information in the database as plaintext. Credit card information should be stored by the third-party credit card provider that is integrated into the system such as PayPal, DataCash, etc. However, since we had to pay for these services we decided to simply mask credit card information after checkout. We therefore, replace all digits except the last 4 digits with *. We also avoid storing CVV numbers on our tables.

ions

	PaymentId	OrderId	CardNumber	ExpiryMonth	ExpiryYear	CardHolderLastName	CardAddressLine1	CardAddressLine2	PostalCode	City
Edit Copy Delete	29	39	**********1234	1	20	test	75 Dalhousie St	1004	M5B 2R9	Toronto

7. **Prevent user from registering if the email address entered already exists in User table:**

Another security feature that was added as part of iterations 3 & 4 was preventing users from registering with an existing email address. The reason for this decision is because we use email addresses to verify users passwords and cookies on login, therefore it is important for email addresses to be unique. For instance, nikan.afshar@ryerson.ca email address already exists in our User table.

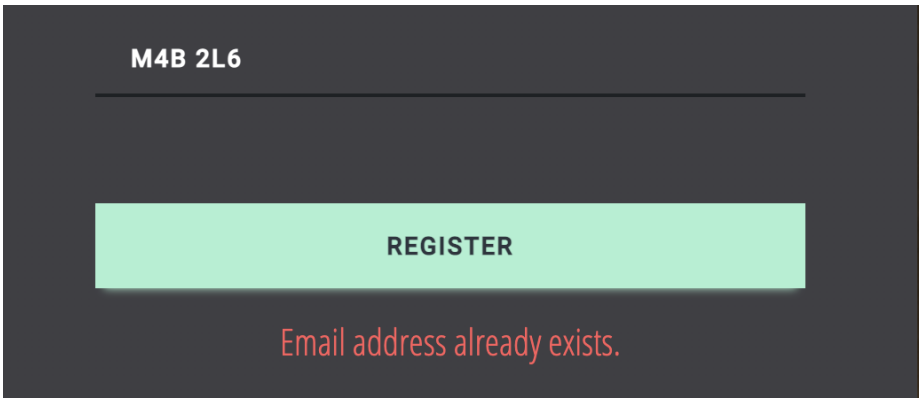


+ Options

	UserId	Name	Tel	Email	Address	CityCode	Password	Balance	isAdmin
Edit Copy Delete	1	Nikan Afshar	(123) 123-1324	nikan.afshar@ryerson.ca	Toronto	M5B3A8	.=rYNuMOiUF4M	0.00	1

Check all *With selected:* Edit Copy Delete Export

Now we try to register a new user with the same email address.



Now the user is prevented from registering until they enter another unique email address.

B. Service C and Shopping Cart Implementation (Part B - 2 & 3)

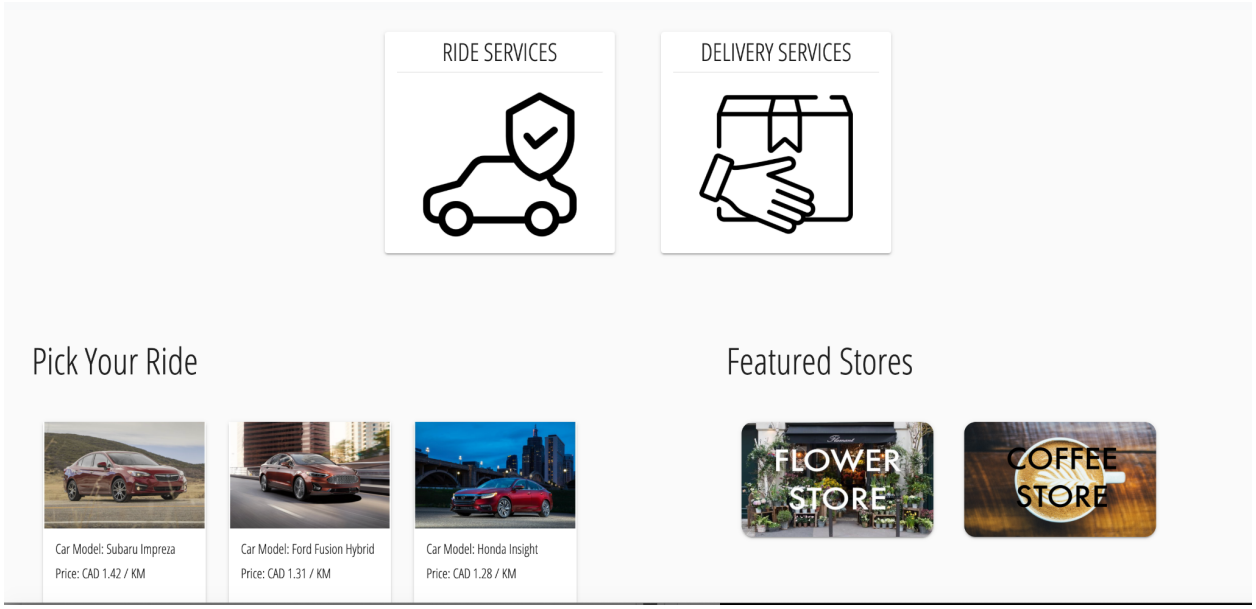
1. **Ride Green Service Component Design:**

In iteration 3, we developed a new service, service C called “Ride Green”. As part of this new service we developed a comparison table of the different services that our system offers. We also use the user reviews and average them for each service to show for each service in the comparison table. By using the comparison table below users will be able to see how prices are calculated (ex. Based on KM or Minutes). You can find a picture of the comparison table below.

Ride Green (Service Comparator)

Services	Customize Car	Customize Driver	Set Ride Preferences	Select Custom Source	Select Custom Destination	Price Per KM/M	Price Guarantee	Average Customer Rating
Ride Services	✓	✗	✗	✓	✓	KM	✓	5
Delivery Services	✗	✗	✗	✗	✓	KM	✓	5
Ride Green	✗	✗	✗	✓	✓	KM	✓	5
Ride Social	✗	✓	✓	✓	✓	Minutes	✗	4

After users compare different services they will be able to pick one of the ride or delivery services (i.e. services A and B) or both of these services, select source and destination, schedule their trip and add the orders to their cart.



As can be seen in the screenshot above, using Angular's SPA architecture we add both services as children to the parent component RideGreenComponent.

```
<div class="row service-container">
        <div class="col-md-6 service" *ngIf="servicesSelected.indexOf('ride-service') >
-1">
            <ride-services-component></ride-services-component>
        </div>
        <div class="col-md-6 service"
*ngIf="servicesSelected.indexOf('delivery-service') > -1">
            <delivery-services></delivery-services>
        </div>
```

```
</div>
```

Using this design architecture we don't need to copy the code inside the delivery service component or ride services component inside the ride green component. Each component will be added by Angular as a child component and each component will have its own controller. However, the main controller will be inside ride-green.component.ts.

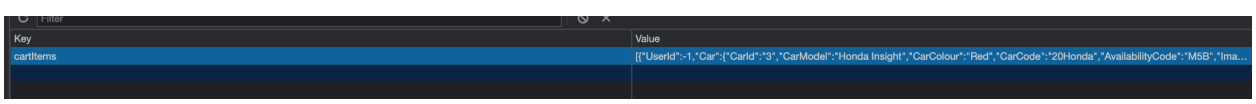
2. **Shopping Cart Development:**

The shopping cart was developed in iterations 3 & 4 to allow for users to add different order types with different parameters to the shopping cart at the same time. Users can add as many different orders as they want to the same shopping cart by drag and drop. The items will be pushed to the cart array on insert which happens inside the data.service.ts that is shared between all different components. In addition, our order interface that was developed in iterations 1 & 2 has a parameter called OrderType. This parameter can take 3 values, namely, ride, delivery or social. Depending on the type of this parameter we use different HTML templates using Angular's *ngIf structural directives.

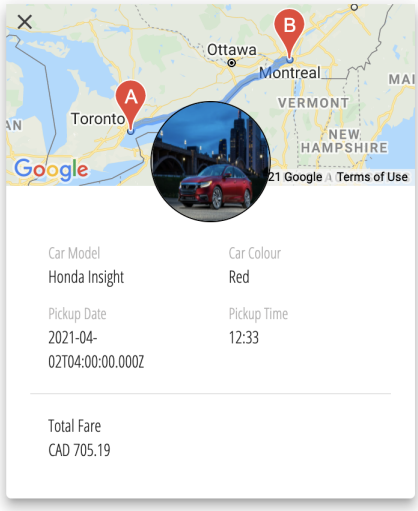
```
export interface Order {
    UserId: number;
    Products?: Product[];
    Car?: Car;
    Driver?: Driver;
    PickupTime: string;
    PickupDate: string;
    OrderTotal: string;
    DeliveryFee?: string;
    DriverFee?: string;
    Distance: string;
    Duration: string;
    StartAddress: string;
    EndAddress: string;
    StartLocationLat: number;
    StartLocationLng: number;
    EndLocationLat: number;
    EndLocationLng: number;
    Direction: any;
    OrderType: string;
}
```

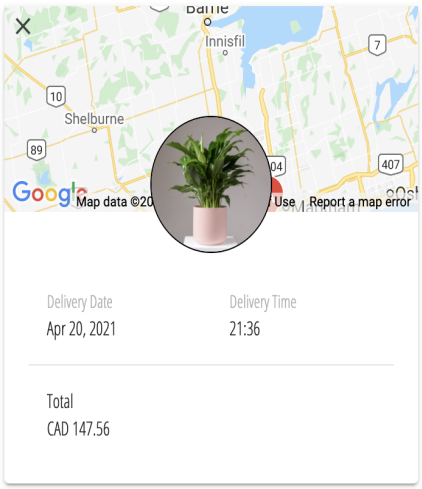
For instance, below is an example of the code we are using to achieve this goal. As part of this implementation we also store the items added by the user to cart in localStorage to persist User's cart items. As a result, users can refresh the website and the items will still remain in the cart. Users can close the browser and when they come back the cart items will be there.

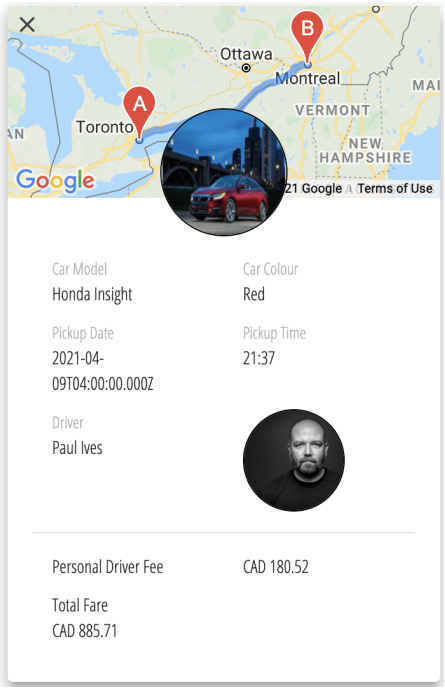
```
<div class="card-content" *ngIf="item.OrderType === 'delivery'">
    <div class="row">
        <span class="material-icons delete-icon"
            (click)="removeItem(item); $event.stopPropagation()">clear</span>
        <!-- <div class="col-md-6">
            <span class="sub-title">Car Model</span><br>
            <span>{{item.Car.CarModel}}</span>
        </div> -->
        <!-- <div class="col-md-6">
            <span class="sub-title">Car Colour</span><br>
            <span>{{item.Car.CarColour}}</span>
        </div> -->
    </div>
    <div class="row">
```



Below are examples of different cart items that can be added to the cart for ride-services, delivery-services and ride-social. As demonstrated in the examples below all these cart items are different depending on the OrderType but they can all be added to the same shopping cart.







B. Browsers Supported (Part B - 4)

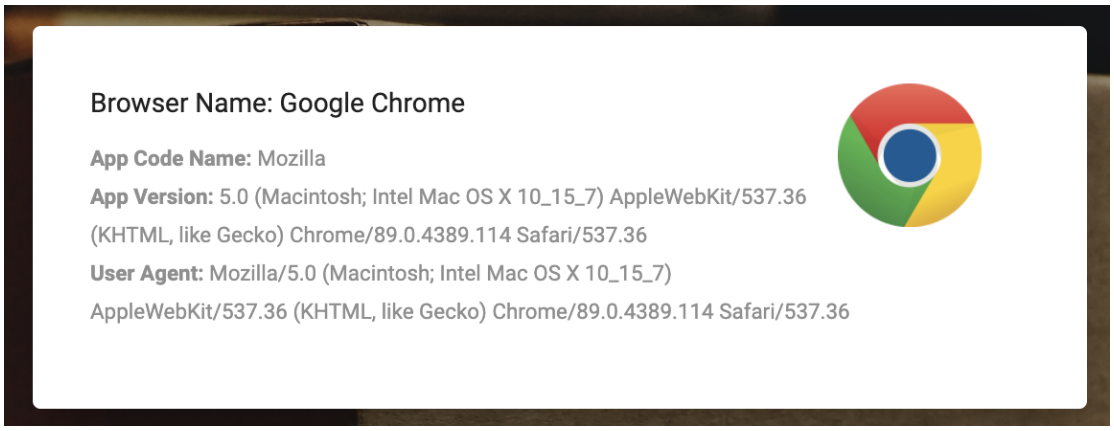
Our application supports most browsers and has been fully tested on 3 browsers Google Chrome, Mozilla Firefox and Apple Safari. We capture browser information using window navigator information. We use the vendor property to check for Google Chrome, user vendor and userAgent properties to check for Apple Safari, and userAgent to check for Mozilla Firefox. Below demonstrates how we use this object to detect user's browser:

```
if (window.navigator.vendor === 'Google Inc.') {
  this.browserName = 'Google Chrome';
} else if (window.navigator.vendor === 'Apple Computer, Inc.' &&
window.navigator.userAgent.toLowerCase().indexOf("safari") !== -1) {
  this.browserName = 'Apple Safari';
```

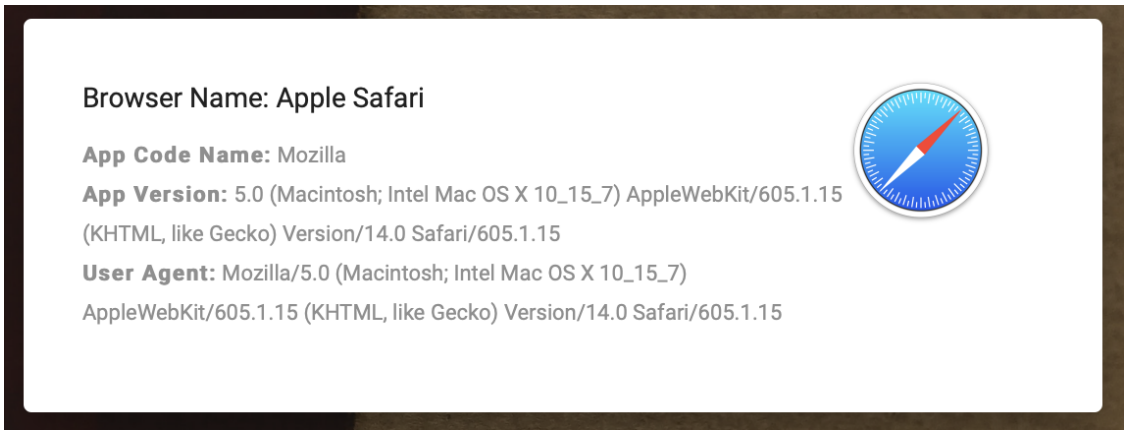
```
} else if (window.navigator.userAgent.toLowerCase().indexOf('firefox') !== -1)
{
  this.browserName = 'Mozilla Firefox';
} else if (window.navigator.userAgent.toLowerCase().indexOf('edge') > -1 ||
window.navigator.userAgent.toLowerCase().indexOf('edg') > -1) {
  this.browserName = 'Edge';
}
```

After the browser information is detected we open the BrowserInfoDialog to show browser information. Below are the screenshots of the dialog when using Google Chrome, Apple Safari and Mozilla Firefox:

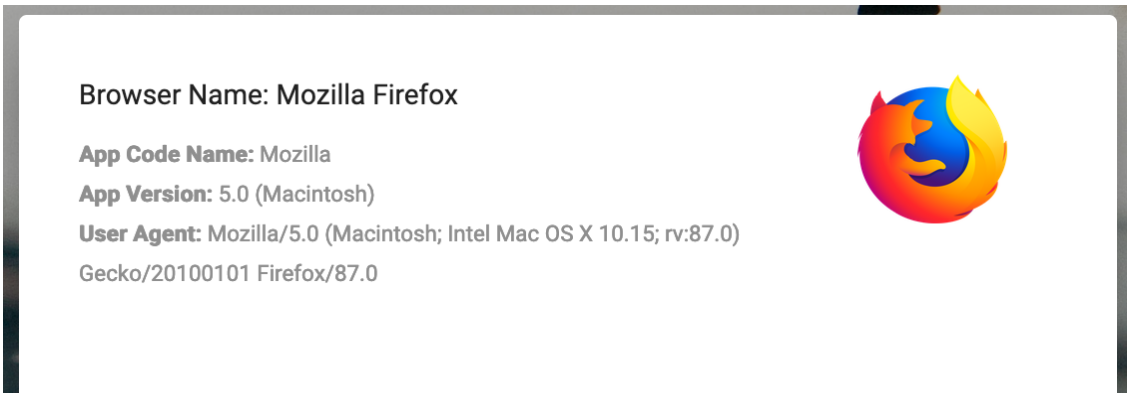
Google Chrome:



Apple Safari:

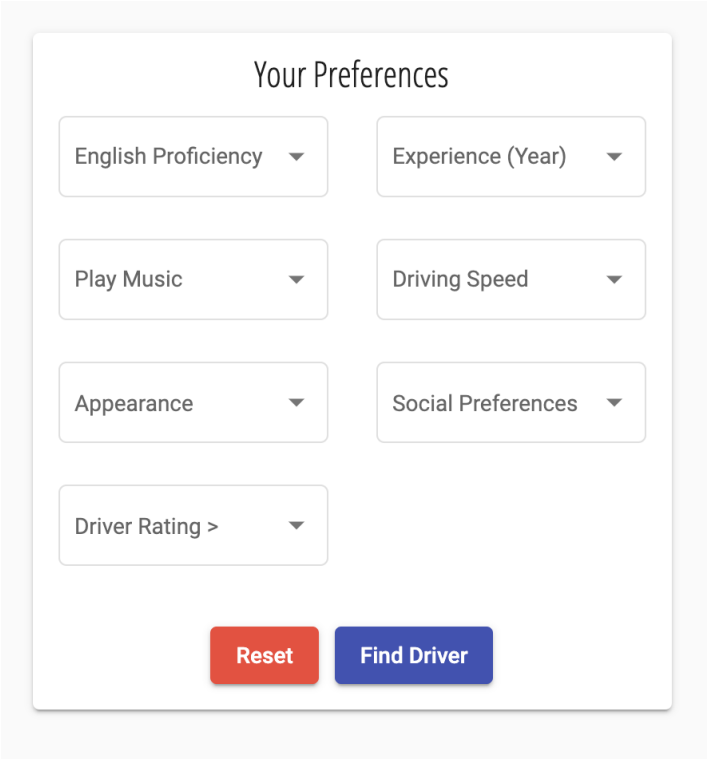


Mozilla Firefox:

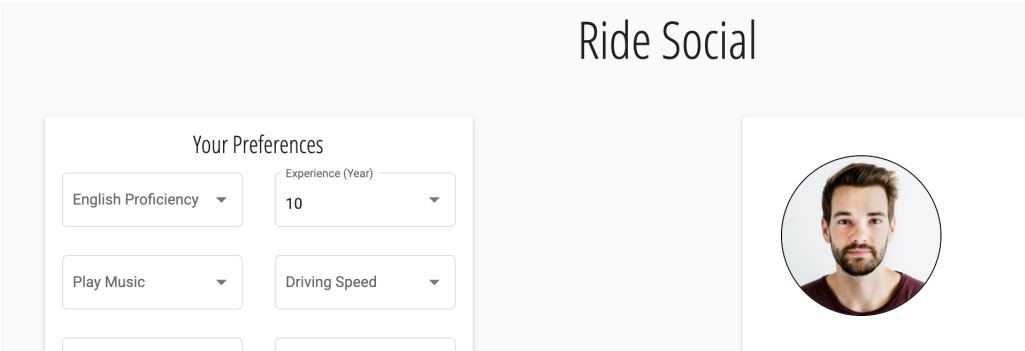


B. The New Demanding Service Added - Service D (Part B - 5)

As part of iteration 3 we developed a new service called “Ride Social” (i.e. Service D). Ride social service allows users to pick their own personal driver according to their own preferences. There are many parameters that users can configure to find the driver that is best matched with them. Below you can see the parameters that users can change to find their driver.



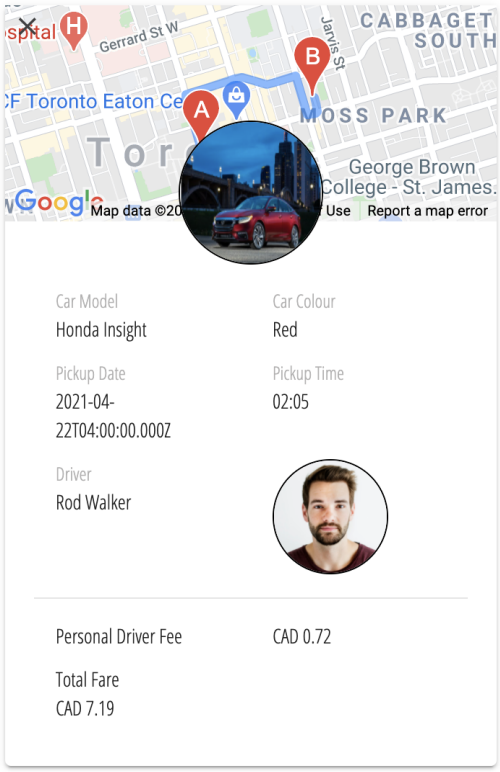
For instance if the user sets Driver Experience to 10 years, they will be able to see drivers with 10 years of experience.



At the moment we only have one driver with 10 years of experience. Users then will be able to select that driver and book their ride service as usual. Using the SPA architecture we reuse the RideServicesComponent so users can pick their own car as well. In order to include the driver fees in the final calculations we pass in the driver as an argument to the RideServicesComponent.

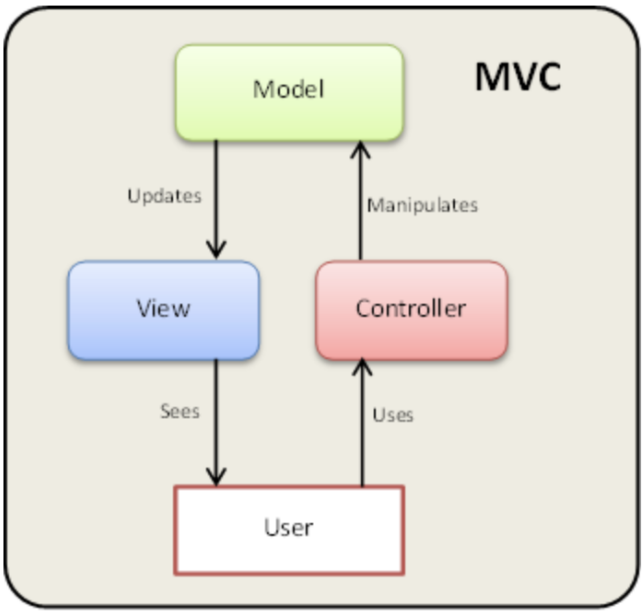
```
<ride-services-component *ngIf="selectedDriverId !== -1"
[driver]="filteredDrivers[selectedDriverId]"></ride-services-component>
```

If the driver is passed into the RideServicesComponent we use if statements and *ngIf directives to add driver fee to the Order Review Card. The Order Review can be added to the shopping cart the same way other Order Review cards (ride-services and delivery-services) can be added to the shopping cart. As can be seen, we use the OrderType parameter to determine the type of the order and if the order is a 'ride-social' service we use *ngIf directive to show the image of the driver in the order card as well as the personal driver fee. Ride social service is calculated based on KM and that means customers have a price guarantee meaning they will know in advance how much their trip would cost and be ensured that will not change at the end of their trip.



C. SPA Architecture (Part C - 1)

We used Angular framework to develop our SPA architecture. SPA or Single Page Application is a web application architecture where instead of the old-fashioned way of reloading each page on user request, the page gets dynamically rewritten. This is way more efficient than the traditional approach of the multi-page applications. SPA does not require page reload and as a result is much more user friendly since it does not require reloading the page. It is basically just one webpage and then all other contents will be loaded using JavaScript. Additionally, our design is based on the MVC design pattern. Angular is a full-featured SPA framework that supports the principles behind the Model-View-Controller architecture. MVC design pattern is used to separate the user's view or template from the actual logic of the application. Below is the diagram showing how MVC in Angular works.

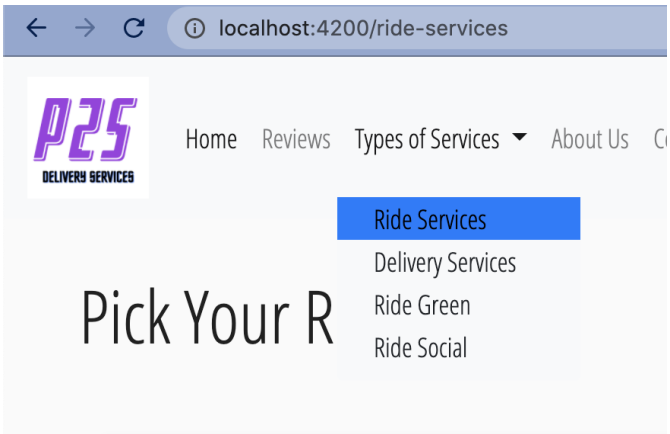


As demonstrated in the diagram above, MVC divides the application into 3 main parts, view, model and controller. View presents the data to the user, model is the logic of the application and controller connects the logic or model with the template or view part. Angular is implemented using a subset of the MVC design pattern called MVVM. MVVM consists of Model, View and ViewModel. These parts can be mapped to angular's main components and directives. For instance, View is implemented using the templates or

HTML files and Model is implemented using TypeScript. The two parts are then connected using directives and data binding.
Our application consists of 33 different components, 21 accessible by routes and we have no separate pages that would require page reload. All these components are loaded into a single HTML page by Angular itself. Since we don't want all components to be visible to the user at the same time we need to use Angular Router to determine which component should be displayed based on the user's decision. In order to implement SPA in Angular we need to define the routes to our components inside routes. Below are the routes we have defined in our application:

```
export const routes: Routes = [
 { path: 'home', component: MainComponent },
 { path: 'ride-services', component: RideServicesComponent },
 { path: 'reviews', component: ReviewsComponent },
 { path: 'delivery-services', component: DeliveryServices },
 { path: 'ride-green', component: RideGreenComponent },
 { path: 'ride-social', component: RideSocialComponent },
 { path: 'contact', component: ContactComponent },
 { path: 'orders', component: OrdersComponent },
 { path: 'about', component: AboutComponent },
 { path: 'dbMaintain/loginattempt-maintain', component: LoginAttemptMaintainComponent
},
 { path: 'dbMaintain/driver-maintain', component: DriverMaintainComponent },
 { path: 'dbMaintain/inquiry-maintain', component: InquiryMaintainComponent },
 { path: 'dbMaintain/trip-maintain', component: TripMaintainComponent },
 { path: 'dbMaintain/payment-maintain', component: PaymentMaintainComponent },
 { path: 'dbMaintain/order-maintain', component: OrderMaintainComponent },
 { path: 'dbMaintain/reviews-maintain', component: ReviewsMaintainComponent },
 { path: 'dbMaintain/user-maintain', component: UserMaintainComponent},
 { path: 'dbMaintain/car-maintain', component: CarMaintainComponent},
 { path: 'dbMaintain/store-maintain', component: StoreMaintainComponent},
 { path: 'dbMaintain', component: DbmaintainComponent},
 { path: '**', component: MainComponent }
];
```

These routes map URLs to their respective components. For instance when the user click on a link (i.e. anchor tag <a href=""></a>) on the navbar to access ride-services, the URL in the browser URL box will change to URL defined inside anchor tag's href attribute and this will be detected by Angular and will load the respective component.




```
<div class="dropdown-menu" aria-labelledby="navbarDropdown">
    <a class="dropdown-item" [routerLink]="['ride-services']" routerLinkActive="active">Ride Services</a>
    <a class="dropdown-item" [routerLink]="['delivery-services']" routerLinkActive="active">Delivery Services</a>
    <a class="dropdown-item" [routerLink]="['ride-green']" routerLinkActive="active">Ride Green</a>
```

Once the routes are defined we need to put the <router-outlet></router-outlet> inside the parent component we want these routes to be rendered inside.

```
<mat-sidenav-content>
    <div class="main-container">
        <router-outlet></router-outlet>
    </div>
</mat-sidenav-content>
```

We have placed our router outlet inside our app.component.html. This will allow angular to dynamically load components based on the URL. We also use ** to set the default route. So if none of the previous routes match the URL, Angular will load MainComponent.

Workload Distribution

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