# IFB299 Personal Portfolio

Chris Martin - n9434631

October 28, 2016

# Contents

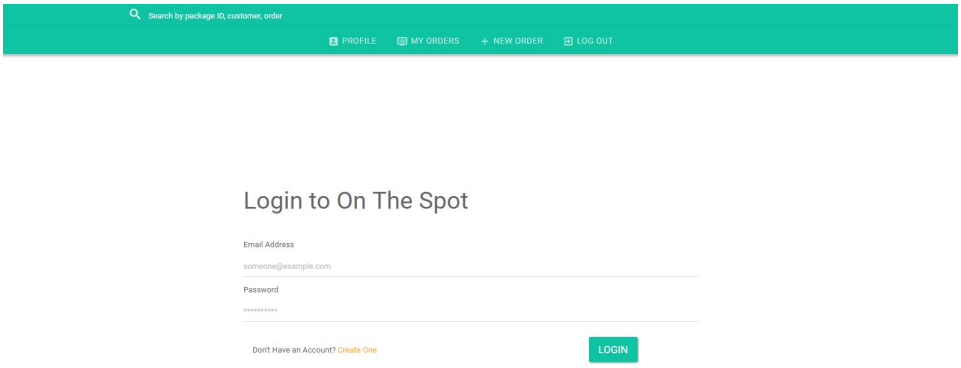
# 1 Release 1

## 1.1 Artefact 1 - Login Page

### Description Of Artefact

The login page was created with HTML and CSS, using the bootstrap CSS library. The styles and layout were created in accordance to existing UI mockups.

### Evidence Of Contribution



Listing 1: Login Page(client/auth/login/login.view.html)

```
<div class="container">
  <div class="form-wrapper">
    <form class="login-form" ng-submit="vm.onSubmit()">
       <div>
            <span>
              <img src="/img/logo-64.png" alt="logo">
              <h3 id="login-title">On The Spot</h3>
            </span>
       </div>
       <div>
            <p id="login-error-msg"
               class="error-msg">Username or password
               incorrect</p>
         <!-- email input field -->
         <label class="control-label" for="email" >Email
            Address</label>
         <input class="form-control" id="email" type="text"
            placeholder="someone@example.com"
            ng-model="vm.credentials.email">
       </div>
       <div>
         <!-- password input field -->
         <label class="control-label"
            for="password">Password</label>
         <input class="form-control" id="password"
            type="password" placeholder="**********"
            ng-model="vm.credentials.password">
       </div>
       <div class="row">
         <div class="col-md-9 col-lg-9 col-xs-12
            reset-link">Don't Have an Account? <a
            href="register">Create One</a></div>
         <button type="submit" class="login-button btn
            btn-primary btn-lg">LOGIN</button>
       </div>
    </form>
  </div>
</div>
```

## 1.2 Artefact 2 - User Object Model

**Description Of Artefact**

The database object to build our users from was designed in a way that allowed us to re-use it for different user types. Based on the email address used during registration, certain fields on the user object in the database were set to true or false.

**Evidence Of Contribution**

Listing 2: Database Object Model(server/models/orders.js)

```
var packageUsers = new mongoose.Schema({
  firstName: {
    type: String,
    required: true
  },
  lastName: {
    type: String,
    required: true
  },
  email: {
    type: String,
    unique: true,
    required: true
  },
  streetNumber: {
    type: String,
    required: true
  },
  streetName: {
    type: String,
    required: true
  },
  suburb: {
    type: String,
    required: true
  },
  postCode: {
    type: String,
```

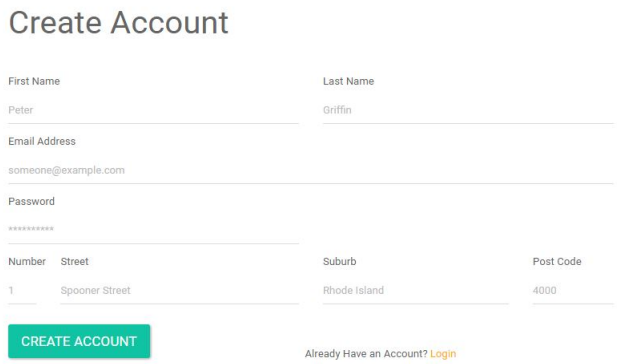
```
    required: true
  },
  isDriver: {
    type: Boolean,
    default: false
  },
  isAdmin: {
    type: Boolean,
    default: false
  },
  hash: String,
  salt: String
});
```

## 1.3 Artefact 3 - Registering Users

**Description Of Artefact**

This artefact includes the registration form as well as the logic to store the user into the database. The registration form was created with HTML and CSS, the logic was written on both the client-side with angular and the server-side with express and mongoose. This code includes the logic to set a user type based on the email address provided.

**Evidence Of Contribution**




Listing 3: Creating User JSON Object to send to server (client/auth/register.controller.js)

```
//create object to be populated with form data, this will
   be sent in http request to server
  vm.credentials = {
    firstName : "",
    lastName : "",
    email : "",
    password: "",
    streetNumber: "",
    streetName: "",
    suburb: "",
    postCode: ""
  };

 /*
```

```
When the user submits the, do all validation and
   transport data given to the back end
 */
 vm.onSubmit = function () {
  console.log('Submitting registration');
    if (validateFields()) { //check for valid fields
      functionService
        .register(vm.credentials)
        .error(function(err){
          toastr.error('There\'s already an account
             registered with that email address',
             'Error');
        })
        .then(function(){
          $location.path('/profile');
      });
    }
 };
```

Listing 4: Making HTTP Request from Client (client/common/services/functionService.js)

```
 //send form data to server for registration
 register = function(user) {
  console.log('register being called');
  return $http.post('/api/register',
     user).success(function(data){
    toastr.success('Account created', 'Success');
    saveToken(data.token);
  });
 };
```

Listing 5: Saving New User Into Database (server/controllers/functionService.js)

```
module.exports.register = function(req, res) {

  //create new User Object Instance
  var user = new User();

  //set object field from form data
  user.firstName = req.body.firstName;
  user.lastName = req.body.lastName;
  user.email = req.body.email;
  user.streetNumber = req.body.streetNumber;
  user.streetName = req.body.streetName;
  user.suburb = req.body.suburb;
  user.postCode = req.body.postCode;

  //split user email address to determine the user type
   var userEmail = (req.body.email).split('@');

  //set User Object Fields based on email address used for
     signup
   var driverOrAdmin = function(username, domain) {
     if (domain.includes('onthespot.com')) {
       user.isDriver = true;
     } else if (username.includes('admin')) {
       user.isAdmin = true;
     }
   };
   driverOrAdmin.apply(null, userEmail);

  //hash password before storing to database
  user.setPassword(req.body.password);

  //save user to database
  user.save(function(err) {
    if (err){
      res.status(500).json(err);
    } else {
      var token;
      token = user.generateJwt();
```

```
    res.status(200).json({ "token": token });
   }

  });
};
```

## 1.4 Artefact 4 - Saving Orders To Database

**Description Of Artefact**

I wrote the code which allowed orders to be saved into the database. This included sending our form data from the client to the server and then building the database object from the data in the http request body.

**Evidence Of Contribution**

Listing 6: Calling our functionService from new order controller (client/orders/newOrder.controller.js)

```
    vm.onSubmit = function () {

      if (validateFields()) {
        console.log('Placing Order');
        console.log(vm.newOrder);
        functionService
          .placeOrder(vm.newOrder)
          .error(function(err){
            console.log(err);
            toastr.error(err, 'Error');
        })
        .then(function(){
          $location.path('orders')
        });
      }
    };
```

Listing 7: Sending new order as HTTP request body (client/common/services/functionService.js)

```
placeOrder = function(order){
  return $http.post('/api/orders/new',
     order).success(function(data){
    toastr.success('Order placed', 'Success');
    toastr.hidden('Hidden', 'Hidden');
  });
};
```

Listing 8: Receiving new order as HTTP request body and saving to database (server/controllers/functionService.js)

```
/*
Places an order into the database by adding the
 */
module.exports.placeOrder = function(req, res) {

  var order = new Order();

  //populate new order instance with form data
  order.userID = req.body.userID;
  order.userName = req.body.userName;
  order.pickUpNumber = req.body.pickUpNumber;
  order.pickUpName = req.body.pickUpName;
  order.pickUpSuburb = req.body.pickUpSuburb;
  order.pickUpPostcode = req.body.pickUpPostcode;
  order.dropOffNumber = req.body.dropOffNumber;
  order.dropOffName = req.body.dropOffName;
  order.dropOffSuburb = req.body.dropOffSuburb;
  order.dropOffPostcode = req.body.dropOffPostcode;
  order.notes = req.body.notes;
  order.isFragile = req.body.isFragile;
  order.isExpress = req.body.isExpress;

  order.state = req.body.state;

  order.save(function(err) {
    if (err){
```

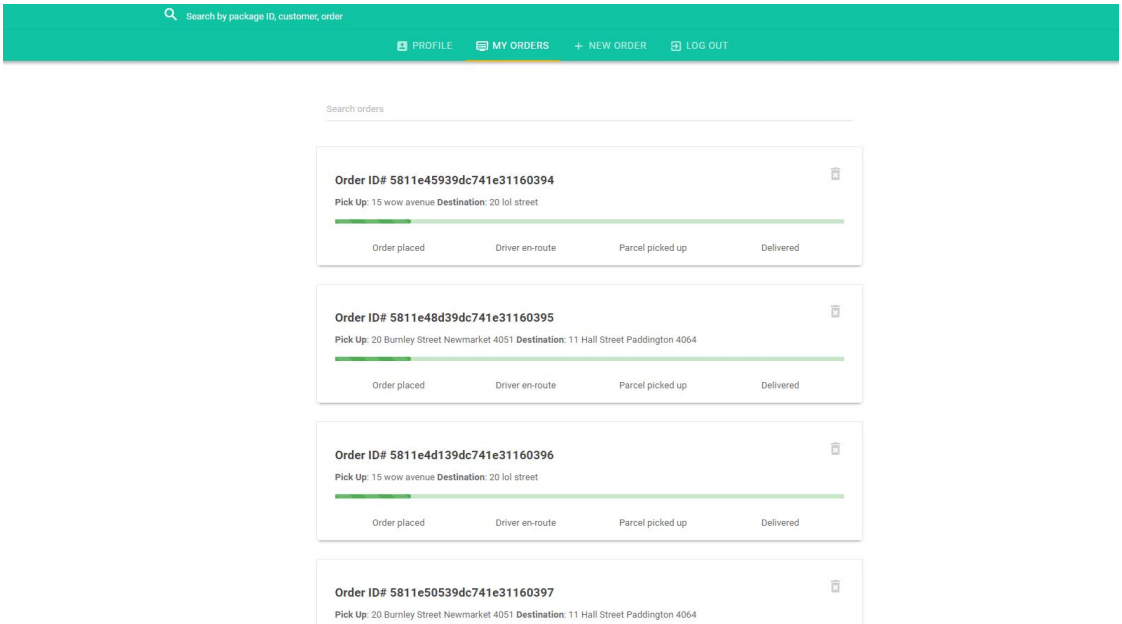
```
      console.log(err);
      res.json({
        error:err
      });
    } else {
        console.log('order saved');
        res.sendStatus(200);
      }
  });
};
```

## 1.5 Artefact 5 - Loading User Orders Into Order View

### Description Of Artefact

This artefact is what allowed a logged in user to view the orders they had placed. This worked by querying the database as soon the 'orders' view was navigated to in the browser. This was later modified to load orders differently based on the logged in user type.

### Evidence Of Contribution



Listing 9: Tell our client functionService to fetch orders for this user (client/orders/order.controller.js)

```
functionService
  .getUserOrders(vm.currentUser.email)
  .error(function(err){
    if (err){
      alert(err);
    }
  })
  .then(function(){
    $location.path('orders');
    console.log('finished getting orders');
    $scope.orders = functionService.loadOrders();
  });
```

Listing 10: Make HTTP call to server passing logged in user as request body (client/common/services/functionService.js)

```
  getUserOrders = function(user){
    return $http.get('/api/orders', {params: {user :
        user}}).success(function(data){
      console.log(data);
      orders = data;
    });
  };
```

Listing 11: Query database for orders based on logged in user type (server/controllers/functionService.js)

```
module.exports.getUserOrders = function(req, res){
 var user = JSON.parse(req.query.user);
 var userEmail = user.email.split('@');
  //if logged in user is a driver, find orders where
     driver field is equal to their name
  if ((userEmail[1] == 'onthespot.com') && (userEmail[0]
     != 'admin')){
    console.log('fetching orders assigned to ' +
```

```
        req.query.user);
    Order.find({ driver: userEmail[0].toLowerCase() },
        function (err, orders) {
      if (err) console.log(err);
      console.log(orders);
      res.send(orders);
    });
  }
 //if they arent a driver, get orders that are placed by
    them
  else{
    console.log('fetching orders for ' + req.query.user);
    Order.find({ userID: user._id }, function (err,
        orders) {
      if (err) {
        res.status(404).json(err); // Error in case the
            server does not reply
      }
      console.log(orders);
      res.send(orders);
    });
  }
};
```

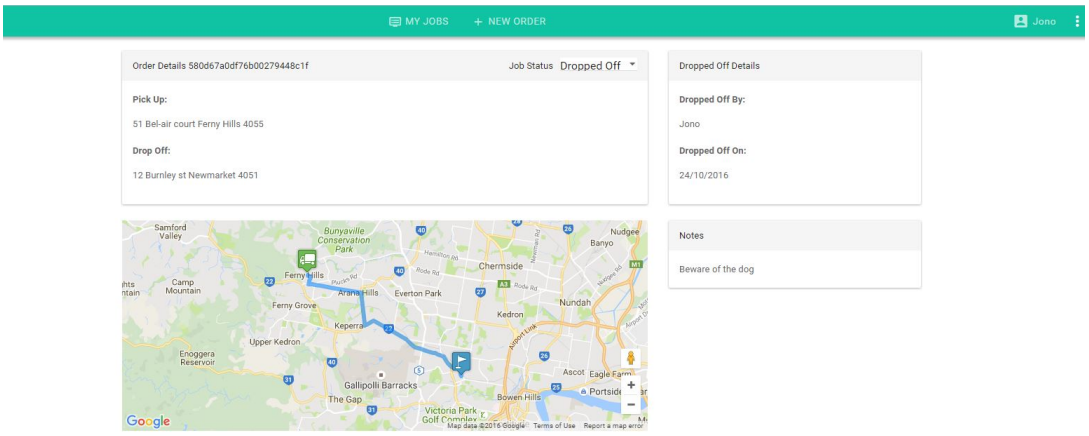
# 2 Release 2

## 2.1 Artefact 1 - Individual Order View

### Description Of Artefact

Users of the application needed to be able to view more information about an order by opening it in it's own view. This was done by using our server as a RESTful API, allowing us to extract parameters from the url and query the database from them. Once this view was loaded, detailed information and a map view of the order was displayed to the user.

### Evidence Of Contribution



Listing 12: Redirect to individual order view passing orderID through URL (client/orders/orders.controller.js)

```
vm.openOrder = function(order){
       if (functionService.loggedInUserType() === 'admin')
          {
         $location.path('admin/dashboard/' + order._id);
       } else {
         $location.path('order/' + order._id);
       }
   };
```

Listing 13: Call server with orderID from URL when user arrives at single order view (client/orders/singleOrder.controller.js)

```
  functionService
    .getSingleOrder(vm.orderID)
    .error(function(err){
      if (err){
      alert(err);
      }
    })
    .then(function(){
      vm.order = functionService.loadSingleOrder();
      if (vm.order.pickedUpAt) {
        vm.pickedUpAtStatus = 'complete';
      }
      if (vm.order.droppedOffAt) {
        vm.droppedOffAtStatus = 'complete';
      }
      if (vm.order.paidAtStatus) {
        vm.paidAtStatus = 'complete';
      }
    }).then(function(){
      //do some logic to get our view variables
      vm.pickUpAddress = vm.order.pickUpNumber+'
         '+vm.order.pickUpName+'
         '+vm.order.pickUpSuburb+'
         '+vm.order.pickUpPostcode;
      vm.dropOffAddress = vm.order.dropOffNumber+'
```

```
        '+vm.order.dropOffName+'
        '+vm.order.dropOffSuburb+'
        '+vm.order.dropOffPostcode;
      vm.state = vm.order.state;
      // Driver name
      vm.driverName =
         (vm.order.driver).charAt(0).toUpperCase() +
         vm.order.driver.slice(1);
      //if we are logged in as a driver, set isSeen to
         true on order object
      if (vm.loggedInUserType() == 'driver' &&
         vm.order.seenByDriver === false){
        console.log('calling mark job as seen');
        functionService.markJobAsSeen({_id :
           vm.order._id});
      }
    });

    NgMap.getMap();
```

Listing 14: Query database with orderID from http request (server/controllers/functionService.js)

```
module.exports.getSingleOrder = function(req, res){
  Order.findOne({ _id: req.query.orderID }, function (err,
     order) {
    if (err) {
      console.log(err);
    }
    else{
      res.send(order);
    }
  });
};
```

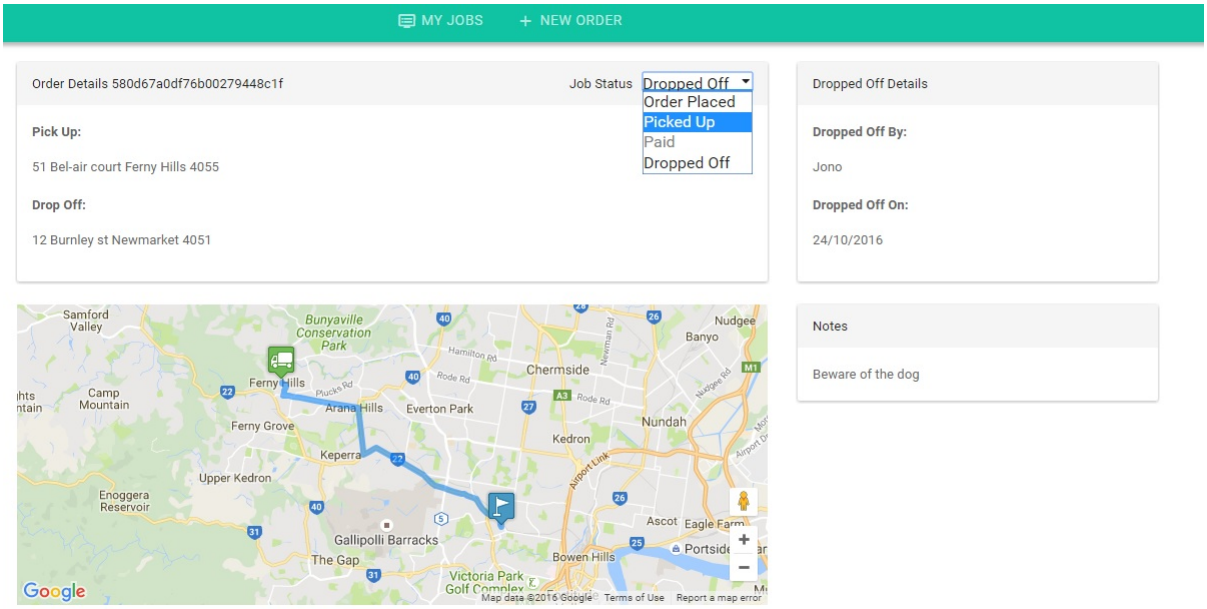
## 2.2 Artefact 2 - Updating Job States

### Description Of Artefact

This functionality was crucial to the way the application worked. Drivers needed to be able to update states of jobs assigned to them. The ability to update jobs was implemented by having a dropdown in the individual order view from which the driver would select a state to change the job to.

Once this selection was made, a request was fired off to the server and the corresponding order in the database would have its 'state' field updated. This functionality is only available to users with a type of 'driver' or 'admin'.

### Evidence Of Contribution



Listing 15: Fetch state from dropdown selection and call functionService with that value
(client/orders/singleOrder.controller.js)

```
vm.updateJobState = function(newState){
       if (newState == 'Paid'){
         if(!vm.validatePaymentDetails()){
           return false;
         }
       }

       update = {
         _id: vm.orderID,
         state: newState,
         pickedUpAt: vm.order.pickedUpAt,
         droppedOffAt: vm.order.droppedOffAt,
         paidAt: vm.order.paidAt
       };

       if (newState.toLowerCase().replace(' ', '') ==
          'pickedup'){
         update.pickedUpAt = Date.now();
       } else if (newState.toLowerCase().replace(' ', '')
          == 'droppedoff'){
         update.droppedOffAt = Date.now();
       } else  if (newState.toLowerCase().replace(' ', '')
          == 'paid'){
         update.paidAt = Date.now();
         //todo format date
       };

       functionService.updateJobState(update).then(function(){
         functionService
         .getSingleOrder(vm.orderID)
         .error(function(err){
           if (err){
             alert(err);
           }
         })
         .then(function(){
           vm.order = functionService.loadSingleOrder();
```

```
        })
        .then(function(){
          $route.reload();
        });
      });
    };
```

Listing 16: Make HTTP request to server with updated job state (client/common/services/functionService.js)

```
  updateJobState = function(update){
    return $http.put('/api/update/jobstate',
       update).success(function(data){
      toastr.success('Job State Changed', 'Success');
      toastr.hidden('Hidden', 'Hidden');
    });
  };
```

Listing 17: Update job state in database (server/controllers/functionService.js)

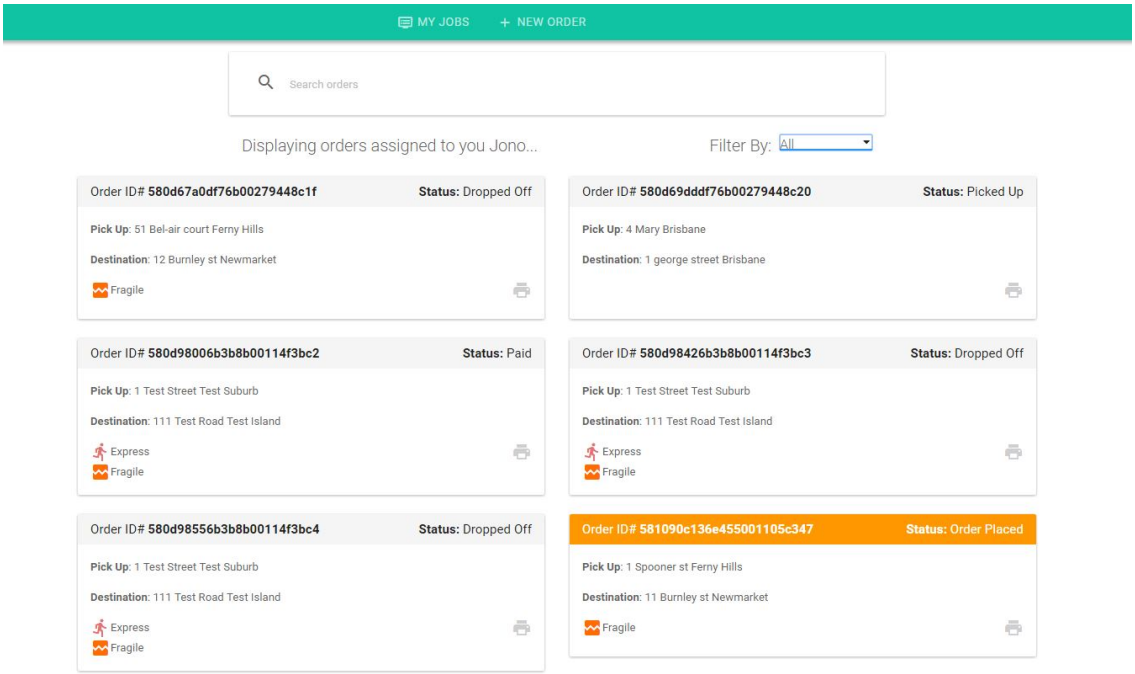
```
module.exports.updateJobState = function(req, res){
  // update job state
  Order.findOneAndUpdate({_id:req.body._id}, req.body,
     {mutli:false, new:true}, function(err, doc){
    if(err) {
      res.status(500).json(err);
    }
    res.status(200).json({ job: doc });
  });
};
```

## 2.3 Artefact 3 - Highlighting Unseen Orders

### Description Of Artefact

This artefact ensured that new jobs were seen by the drivers they were assigned to. The order object model was modified to include a 'seenByDriver' field. While this field was false, the order was highlighted in bright orange until the driver opened the order in its individual view.

### Evidence Of Contribution



Listing 18: Apply css class to orders when loading them into orders view (client/orders/orders.controller.js)

```
  .then(function(){
    if (functionService.loggedInUserType() === 'admin') {
      functionService.getCurrentOrders()
        .then(function() {
          vm.orders = functionService.loadOrders();
        })
        .then(function() {
          vm.orders.forEach(function(item, index){
            if (item.seenByDriver === false){
              vm.orders[index].panelClass = 'panel
                 panel-warning';
            }
            else{
              vm.orders[index].panelClass = 'panel
                 panel-default';
            }
          });
        })
```

Listing 19: Call functionService to mark job as seen when opened by the assigned driver (client/orders/singleOrder.controller.js)

```
if (vm.loggedInUserType() == 'driver' &&
   vm.order.seenByDriver === false){
         console.log('calling mark job as seen');
         functionService.markJobAsSeen({_id :
            vm.order._id});
     }
}
```

Listing 20: Update job in the database to be 'seenByDriver' (server/controllers/functionService.js)

```
module.exports.markJobAsSeen = function(req, res){
  Order.findOneAndUpdate({ _id:req.body._id},
     {seenByDriver: true}, function (err, order){
   if (err) {
```

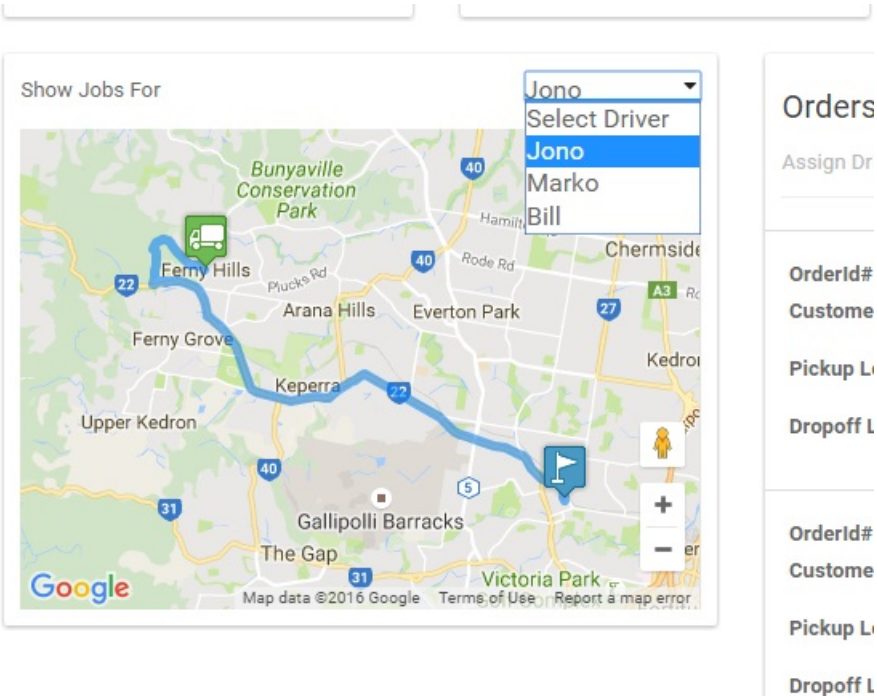
```
      console.log(err);
    }
    else{
      console.log('job marked as seen');
      res.send(order);
    }
  });
};
```

## 2.4 Artefact 4 - Admin Map

**Description Of Artefact**

This map was one of the features on the admin dashboard. The map allowed Bill to view the routes for the jobs his drivers were currently assigned to. This functionality was a requirement for our applcation.

**Evidence Of Contribution**



Listing 21: Add map and load in map routes based on dropdown selection (client/admin/dashboard/dashboard.view.html)

```
<div class="col-md-5">
      <div class="panel panel-default">
        <div class="widget-heading">
          <p>
            <span>Show Jobs For</span>
            <select ng-model='vm.selectedDriver'
               class='selectpicker dropdown pull-right
               jobstate-dropdown'>
              <option selected>Select Driver</option>
              <option ng-repeat='driver in vm.drivers'
                value='{{driver.firstName}}'>{{driver.firstName}}
              </option>
            </select>
          </p>
          <ng-map center="-27.46,153.02" zoom='11'>
            <!-- pick up marker -->
            <marker ng-repeat='order in vm.placedOrders
               | filter : {driver: vm.selectedDriver}'
              position="{{order.pickUpNumber+'
                 '+order.pickUpName+'
                 '+order.pickUpSuburb+'
                 '+order.pickUpPostcode}}"
              title="{{order.pickUpNumber+'
                 '+order.pickUpName+'
                 '+order.pickUpSuburb+'
                 '+order.pickUpPostcode}}"
              icon="img/pickUpMarker.png">
            </marker>
            <!-- drop off marker -->
            <marker ng-repeat='order in vm.placedOrders
               | filter : {driver: vm.selectedDriver}'
              position="{{order.dropOffNumber+'
                 '+order.dropOffName+'
                 '+order.dropOffSuburb+'
                 '+order.dropOffPostcode}}"
              title="{{order.dropOffNumber+'
```

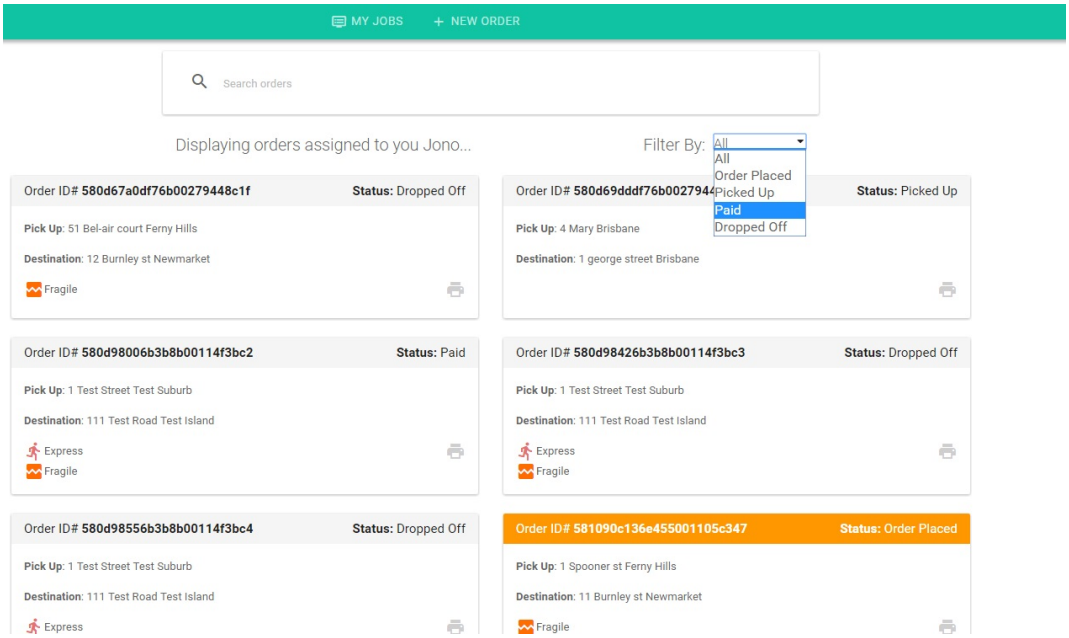
```
                '+order.dropOffName+'
                '+order.dropOffSuburb+'
                '+order.dropOffPostcode}}"
            icon="img/dropOffMarker.png">
        </marker>
        <!-- directions -->
        <directions ng-repeat='order in
           vm.placedOrders | filter :
           {driver:vm.selectedDriver}'
        suppress-markers="true"
        origin="{{order.pickUpNumber+'
           '+order.pickUpName+'
           '+order.pickUpSuburb+'
           '+order.pickUpPostcode}}"
        destination="{{order.dropOffNumber+'
           '+order.dropOffName+'
           '+order.dropOffSuburb+'
           '+order.dropOffPostcode}}">
        </directions>
      </ng-map>
    </div>
  </div>
</div>
```

## 2.5 Artefact 5 - Filtering Orders Based On State

### Description Of Artefact

This important functionality allowed the user to show or hide orders based on the job states. This was a requirement as our users would otherwise have to sift through all jobs to find the one they wanted. The user is able to filter jobs by selecting a desired job state from a dropdown menu on the orders view.

### Evidence Of Contribution



Listing 22: Add map and load in map routes based on dropdown selection (client/orders/orders.view.html)

```
<div class="row">
  <div class='col-md-8 col-md-offset-2 col-xs-10
     col-xs-offset-1'>
    <p class="lead"><span>{{vm.ordersMessage}}...</span>
      <select ng-model='vm.filter' class='selectpicker
         dropdown pull-right jobstate-dropdown'>
        <option selected="vm.filter == ''"
           value=''>All</option>
        <option selected="vm.filter == 'Order Placed'"
           value='Order Placed'>Order Placed</option>
        <option selected="vm.filter == 'Picked Up'"
           value='Picked Up'>Picked Up</option>
        <option selected="vm.filter == 'Paid'"
           value='Paid'>Paid</option>
        <option selected="vm.filter == 'Dropped Off'"
           value='Dropped Off'>Dropped Off</option>
      </select>
      <span class='pull-right'>Filter By: </span>
    </p>
  </div>
</div>

<div class='row'>
  <div class="col-md-6" ng-repeat="order in vm.orders |
     filter:{state : vm.filter} | orderBy: placedAt :
     reverse">
    <div class="{{order.panelClass}}">
      <div class="panel-heading ">
        <h3 class="panel-title clickable"
           ng-click="vm.openOrder(order)">Order ID#
           <strong>{{order._id}}</strong>
          <span class='card-dest-text
             pull-right'><strong>Status:
             </strong>{{order.state}}</span>
        </h3>
      </div>
      <div class="panel-body">
        <p class='card-dest-text'><strong>Pick
```

```
            Up</strong>: {{order.pickUpNumber + ' ' +
            order.pickUpName + ' ' +
            order.pickUpSuburb}}</p>
         <p
            class='card-dest-text'><strong>Destination</strong>:
            {{order.dropOffNumber + ' ' +
            order.dropOffName + ' ' +
            order.dropOffSuburb}}</p>
         <span class='float-right'>
           <a href='/label/{{order._id}}'
              target="_blank"><i
              class="material-icons">print</i></a>
         </span>
         <div>
           <!--Metadata Icons-->
           <div ng-if='order.isExpress'>
           <span class='float-left' style="display:
              flex;">
             <i class="material-icons
                express-icon">directions_run</i>
             <span>Express</span>
           </span>
           </div>

           <div ng-if='order.isFragile'>
           <span class='float-left' style="display:
              flex;">
             <i class="material-icons
                valuable-icon">broken_image</i>
             <span>Fragile</span>
           </span>
           </div>
         </div>
       </div>
     </div>
   </div>
 </div>
```