Lab 1.2 - How to capture custom events

As has been mentioned earlier, the MobileFirst Platform provides the ability to generate custom log events, which can be used to provide a better understanding of how your users are interacting with an application.

In the following lab, we will explore the logging API and add it to our application in several key places. The logged events will be collected by the Analytics server, and will be available for inspection and charting to give us very specific insight into usage patterns by our community.

Add WL.Analytics.log() calls to the controller logic in controllers.js

Open the controllers.js file.

With angular / ionic MVC processing, a controller is called each time the view is loaded by user navigation. In our app we have 3 views - splash, main (or employee), and details, along with 3 controllers - splashCtrl, mainCtrl, and employeeDetailCtrl. We will add our logging event to each controller, capturing an event each time one of these views is loaded.

Modify the controller code in controllers.js and add the following code, which will log the **'viewLoad'** event with the name of the view (login view in this screenshot):

```
// Adding custom event
var event = {viewLoad: 'login view'};
WL.Analytics.log(event, 'Login view - loaded');
```

Locate the **controllers.js** file under **IBMEmployeeApp/www/js/controllers.js** and start with the **mainCtrl** and add the custom event logic to the bottom of the controller block to log entry into the main view.
The mainCtrl is called every time the employee.html (employee list) view is displayed.

Make sure you change the log message to reference the employee view as shown in the after block below:
Your code should look like this

Before

```
ibmApp.controller('mainCtrl', ['$scope', 'employees',  function ($scope, employees) {
    console.log(">> in mainCtrl ... ");
    //ionicMaterialInk.displayEffect();
    $scope.employees = employees;

    }])

```

After

```
ibmApp.controller('mainCtrl', ['$scope', 'employees',  function ($scope, employees) {
    console.log(">> in mainCtrl ... ");
    //ionicMaterialInk.displayEffect();
    $scope.employees = employees;

    var event = {viewLoad: 'Employee List View'};
    WL.Analytics.log(event, 'Employee List View - loaded');

}])
```

Repeat for **employeeDetailCtrl**, changing the view name in the logged message from "login" to "detail" as shown in the after block below. This message will log each time the details.html view is displayed.

Before

```
 ibmApp.controller('employeeDetailCtrl', function($scope, EmployeeService,
                              employeeDetailList , empId ,$ionicHistory) {
   $scope.employee = {
         "first_name" : "",
         "last_name" : "",
         "_id" : ""
   }
   $scope.employeeDetails = {}
   console.log(">> in - employeeDetailCtrl:" + employeeDetailList);
   //Employee service cached the list of employee
   $scope.employee = EmployeeService.getEmployeeById(empId);
   $scope.employeeDetails = employeeDetailList;
   $scope.employeeDetails.email =  angular.lowercase($scope.employeeDetails.email);

 })
```

After

```
ibmApp.controller('employeeDetailCtrl', function($scope, EmployeeService,
                                employeeDetailList , empId ,$ionicHistory) {
  $scope.employee = {
        "first_name" : "",
        "last_name" : "",
        "_id" : ""
  }
  $scope.employeeDetails = {}
  console.log(">> in - employeeDetailCtrl:" + employeeDetailList);
  //Employee service cached the list of employee
  $scope.employee = EmployeeService.getEmployeeById(empId);
  $scope.employeeDetails = employeeDetailList;
  $scope.employeeDetails.email =  angular.lowercase($scope.employeeDetails.email);

  var event = {viewLoad: 'Details View'};
  WL.Analytics.log(event, 'Details View - loaded');

})
```

Repeat once more for **splashCtrl**, which is called when the application start, please add it within the timeout callback *Make sure you change the log message to reference the employee view as shown in the after block below*:

Before

```
    $timeout(function(){
        $scope.moveSplashBox();
    }, 3000);

 }])
```

After

```
    $timeout(function(){
        $scope.moveSplashBox();
        var event = {viewLoad: 'Splash View'};
        if(WL!=null && WL!=undefined) WL.Analytics.log(event, 'Splash View - loaded');
    }, 3000);

}])
```

Add WL.Analytics.send() to push queued events to the server

The MobileFirst client will automatically forward collected log messages when the log threshold is reached, but for this exercise we would like to push them to the mfp server as soon as they are logged, so that we can see them immediately and continue with creating our custom charts.

Open the **app.js** Add `WL.Analytics.send();` to the wlCommonInit() method, just at the end of the method.

Before

```
function wlCommonInit() {
  console.log(">> wlCommonInit() ..." );
  var serverUrl = WL.App.getServerUrl(function(success){
      console.log(success);
  }, function(fail){
      console.log(fail);
  });

  //Calling to the MobileFirst Server
  WLAuthorizationManager.obtainAccessToken().then(
        function (accessToken) {
          console.log(">> Success - Connected to MobileFirst Server");
        },
        function (error) {
          console.log(">> Failed to connect to MobileFirst Server");
        }
  );
};
```

After

```
function wlCommonInit() {
  console.log(">> wlCommonInit() ..." );
  var serverUrl = WL.App.getServerUrl(function(success){
      console.log(success);
  }, function(fail){
      console.log(fail);
  });

  //Calling to the MobileFirst Server
  WLAuthorizationManager.obtainAccessToken().then(
        function (accessToken) {
          console.log(">> Success - Connected to MobileFirst Server");
        },
        function (error) {
          console.log(">> Failed to connect to MobileFirst Server");
        }
  );

  WL.Analytics.send();

};
```

Save your changes!

In the next lab, we will run the app to generate the data, then use the Analytics console to create a custom chart showing the distribution of views being loaded.

Summary

With these simple additions, your application now delivers custom events to the MFP Analytics server to track usage patterns in the application and help with problem diagnosis, usability and app health.

In case you got lost on the way

You can easily get to this stage by running the following command :

```
git checkout -f step-8
```