Setting up the Web development environment

Overview

Developing and testing web applications is as easy as previewing a local HTML file in your web browser of choice.
Developers can use their IDE of choice, and any framework(s) that suits their needs.

However one thing may stand in the way of developing web applications. Web applications might encounter errors due to same-origin policy (https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy) violation. Same-origin policy is a restriction embosed on web browsers. For example, if an application is hosted on the domain **example.com**, it is not allowed for the same application to also access contect that is available on another server, or for that matter, from the MobileFirst Server.

Web apps that are to use the MobileFirst Web SDK (../../../application-development/sdk/web) should be handled in a supporting topology, for example by using a Reverse Proxy to internally redirect requests to the appropriate server while maintaining the same single origin.

The policy requirements can be satisfied by using either of the following methods:

- Serving the web application resources' from the same WebSphere Full/Liberty profile application server that also hosts the MobileFirst Server.
- Using Node.js as a proxy to redirect application requests to the MobileFirst Server.

Prerequisites:
The following requires either Apache Maven or Node.js installed on the developer's workstation.
For instructions, refer to the installation guide (../mobilefirst/installation-guide/).

Using WebSphere Liberty profile to serve the web application resources

In order to serve the web application's resources, these need to be stored in a Maven webapp (a **.war** file).

Creating a Maven webapp archetype

1. From a **command-line** window, navigate to a location of your choosing.
2. Run the command:

   ```
   mvn archetype:generate -DgroupId=MyCompany -DartifactId=MyWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
   ```

   - Replace **MyCompany** and **MyWebApp** with your own values.
   - To enter the values one-by-one, remove the `-DinteractiveMode=false` flag.

Building the Maven webapp with the web application's resources

1. Place the web application's resources (such as the HTML, CSS, JavaScript and image files) inside the generated **[MyWebApp] → src → Main → webapp** folder.

> From here on, consider the **webapp** folder as the development location for the web application.

2. Run the command: `mvn clean install` to generate a .war file containing the application's web resources.
   The generated .war file is available in the **[MyWebApp] → target** folder.

> **Important:** `mvn clean install` must be run each time you update a web resource.

Adding the Maven webapp to the application server

1. Edit the **server.xml file** of your WebSphere application server.
   If using the MobileFirst Development Kit, the file is located in: [**MobileFirst Development Kit] → mfp-server → user → servers → mfp** folder. Add the following entry:

```
<application name="MyWebApp" location="path-to/MyWebApp.war" type="war"></application>
```

   - Replace **name** and **path-to/MyWebApp.war** with your own values.
   - The application server is automatically restarted after saving the changes to the **server.xml** file.

Testing the web application

Once you are ready to test your web application, visit the URL: **localhost:9080/MyWebApp**. - Replace **MyWebApp** with your own value.

Using Node.js

Node.js can be used as a reverse proxy to tunnel requests from the web application to the MobileFirst Server.

1. From a **command-line** window, navigate to your web application's folder and run the following set of commands:

```
npm init
npm install --save express
npm install --save request
```

2. Create a new file next to the **node_modules** folder, for example **proxy.js**.
3. Add the following code to the file:

```
var express = require('express');
var http = require('http');
var request = require('request');

var app = express();
var server = http.createServer(app);
var mfpServer = "http://localhost:9080";
var port = 9081;

server.listen(port);
app.use('/myapp', express.static(__dirname + '/'));
console.log('::: server.js ::: Listening on port ' + port);

// Web server - serves the web application
app.get('/home', function(req, res) {
   // Website you wish to allow to connect
   res.sendFile(__dirname + '/index.html');
});

// Reverse proxy, pipes the requests to/from MobileFirst Server
app.use('/mfp/*', function(req, res) {
   var url = mfpServer + req.originalUrl;
   console.log('::: server.js ::: Passing request to URL: ' + url);
   req.pipe(request[req.method.toLowerCase()](url)).pipe(res);
});
```

- replace the **port** value with your preferred one.
- replace `/myapp` with your preferred path name for your web application.
- replace `/index.html` with the name of your main HTML file.
- if needed, update `/mfp/*` with the context root of your MobileFirst runtime.

4. To start the proxy, run the command: `node proxy.js`.
5. Once you are ready to test your web application, visit the URL: **server-hostname:port/app-name**, for example: **http://localhost:9081/myapp**
   - Replace **server-hostname** with your own value.
   - Replace **port** with your own value.
   - Replace **app-name** with your own value.

Next steps

To continue with MobileFirst development in Web applications, the MobileFirst Web SDK need to be added to the Web application.

- Learn how to add the MobileFirst SDK to Web applications (../../../application-development/sdk/web/).
- For applications development, refer to the Using the MobileFirst Foundation SDK (../../../application-development/) tutorials.
- For adapters develpment, refer to the Adapters (../../../adapters/) category.

Last modified on