Design Document and Impact Analysis

Prepared by:

Jessy Jacob Mesapam, Prachi Lotake, Deepthi Yanamala, Jashitha Boyapati, Vamsi Akula

Contents:

1. Data Report Charts functionality

1.1. Concept Location and Approach:

1.1.1. During the preparation of requirements book, we already began searching for files in which changes need to be made to attain required features. We started looking in mango src folder and webapps folder with key word "report." In search results we found 'reports.jsp' to be most significant among several other class files and folders.

1.1.1.1. *Input data fields files:* The reports.jsp file consists of functions/methods that collect and process the input values which are collected through user input for the report generation. Some of the functions are addPointToReport()
addToReportPointsArray()
writeReportPointsArray()

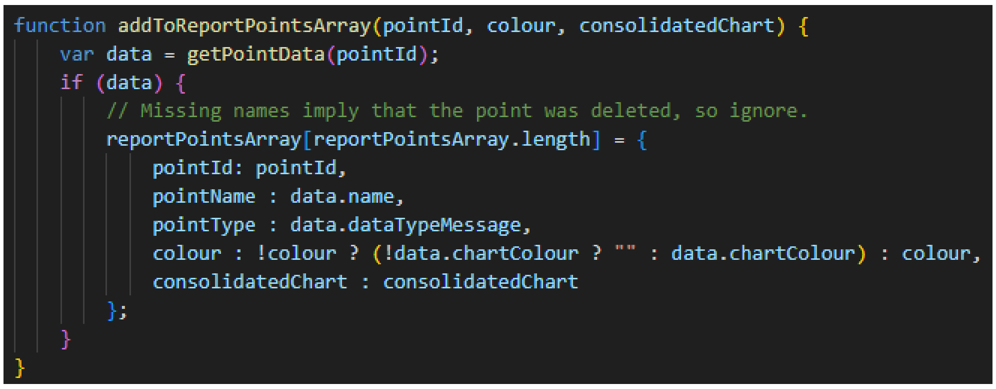
1.1.2. We also found an entire folder named 'report' in the path mango\WEB-INF\classes\com\serotonin\mango\vo\report dedicated to generate charts and reports for the data points selected. Significant changes could be seen in

1.1.2.1. ReportPointVO.java & ReportPointInfo.java – contains all getter and setter methods for the report attributes as per current functionality

1.1.2.2. ReportChartCreator.java – enables us to set start time and end time of chart data and datatype

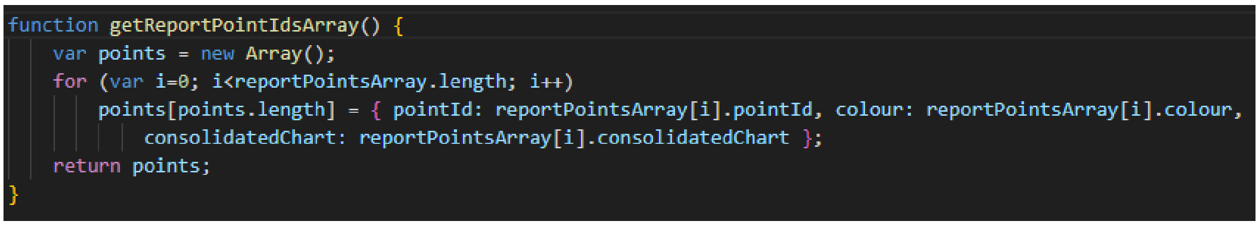
1.1.2.3. ImageChartUtils.java – contains attributes like height, width etc for the charts generated

1.1.3. *DB query location:* 'reportDao.java' found in path stated in 1.1.2 consists of all database calls for the attributes like pointName, pointType as per current chart functionality.



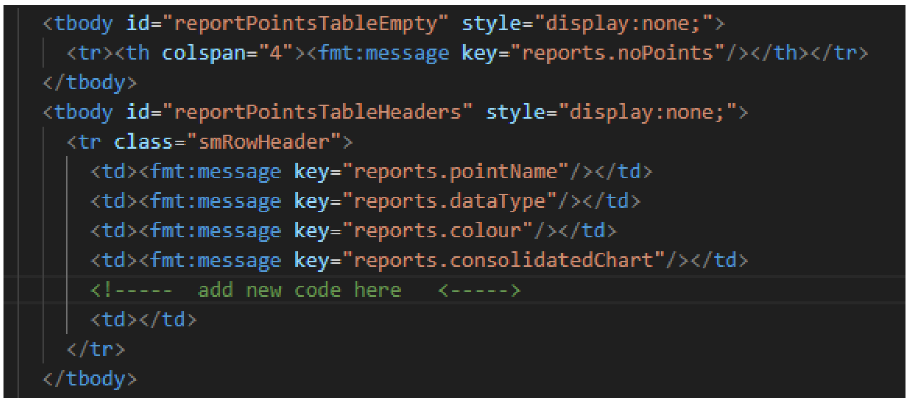
```
function addToReportPointsArray(pointId, colour, consolidatedChart) {
    var data = getPointData(pointId);
    if (data) {
        // Missing names imply that the point was deleted, so ignore.
        reportPointsArray[reportPointsArray.length] = {
            pointId: pointId,
            pointName : data.name,
            pointType : data.dataTypeMessage,
            colour : !colour ? (!data.chartColour ? "" : data.chartColour) : colour,
            consolidatedChart : consolidatedChart
        };
    }
}
```

1(a)



```
function getReportPointIdsArray() {
    var points = new Array();
    for (var i=0; i<reportPointsArray.length; i++)
        points[points.length] = { pointId: reportPointsArray[i].pointId, colour: reportPointsArray[i].colour,
            consolidatedChart: reportPointsArray[i].consolidatedChart };
    return points;
}
```

1(b)



```
<tbody id="reportPointsTableEmpty" style="display:none;">
  <tr><th colspan="4"><fmt:message key="reports.noPoints"/></th></tr>
</tbody>
<tbody id="reportPointsTableHeaders" style="display:none;">
  <tr class="smRowHeader">
    <td><fmt:message key="reports.pointName"/></td>
    <td><fmt:message key="reports.dataType"/></td>
    <td><fmt:message key="reports.colour"/></td>
    <td><fmt:message key="reports.consolidatedChart"/></td>
    <!-----  add new code here   <----->
    <td></td>
  </tr>
</tbody>
```

1(c)

Fig1: Snippets of code from reports.jsp file (a)addToReportArray(); (b)getReportPointIdsArray(); (c)reportPointsTableHeaders

1.2.Impact Analysis:

1.2.1. The new feature data input fields are to be added to the functions and other fields wherever inputs are referenced in the reports.jsp file.

1.2.1.1. The reports.jsp file will also need additional functions such as updatePointChartType(), updatePointTitle(), updatePointXaxisLabel(), updatePointYaxisLabel(), updatePointYReferenceLine() to update the inputs from the user.

1.2.1.2. These added data fields should also be included under the <tbody id = “reportPointsTableHeaders” tag and increase colspan from 4 to 9 which display them to the user in the report criteria table from reports tab.

1.2.1.3. Getter and Setter methods for new attributes – ChartType, title, AxisLabels and YreferenceLine shall be added in ReportPointVO.java and ReportPointInfo.java.

1.2.2. With the current knowledge we possess and taking into consideration the complexity of the file interactions, we could only come up with one solution in the given time constraints.

2. Landing Page Functionality

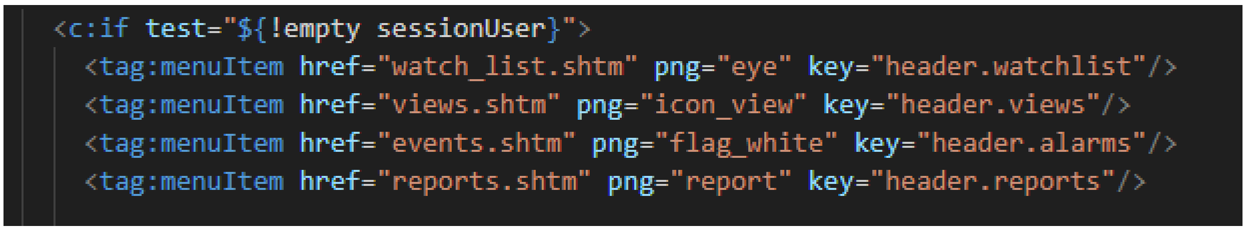
2.1.Concept Location

2.1.1. We started searching with keywords such as home, page, url, login etc and found ‘page.tag’ file. In this file, we can design the navigation bar by using png files, and link a URL to it. Each icon has a key which is linked to a header that provides name based on the language chosen. As English is selected by default the message_en.properties is used to get the respective values. There are six ‘.properties’ files for each language setting used in Mango software.

2.1.1.1. We also found references to MiscDwr.class file within this, where functions like getHomeUrl() and setHomeUrl() are used in setting a default homepage. When no URL is provided the getHomeUrl() takes ‘watch_list.shtm’ by default.

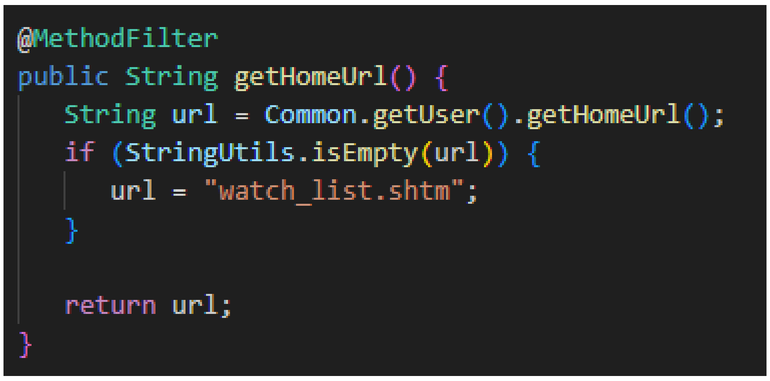
2.1.2. We found another file springDispatcher-servlet.xml in WEB-INF folder which contains all the URL’s mapping to respective Controllers.

2.1.2.1. Within this file we found loginController with successUrl value set to watch_list.shtm which is a URL linked to watchlistController(the current page after logging in).



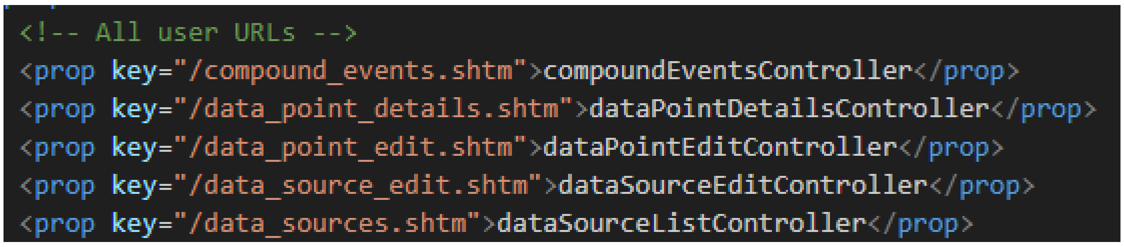
```
<c:if test="${!empty sessionUser}">
  <tag:menuItem href="watch_list.shtm" png="eye" key="header.watchlist"/>
  <tag:menuItem href="views.shtm" png="icon_view" key="header.views"/>
  <tag:menuItem href="events.shtm" png="flag_white" key="header.alarms"/>
  <tag:menuItem href="reports.shtm" png="report" key="header.reports"/>
```

2(a)



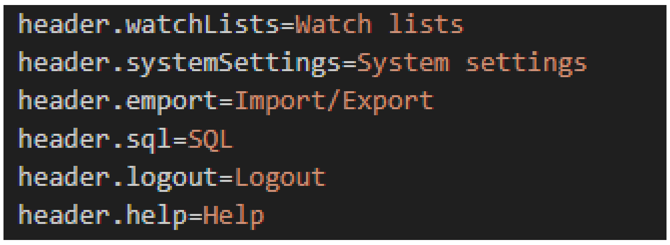
```
@MethodFilter
public String getHomeUrl() {
   String url = Common.getUser().getHomeUrl();
   if (StringUtils.isEmpty(url)) {
      url = "watch_list.shtm";
   }

   return url;
}
```

2(b)



```
<!-- All user URLs -->
<prop key="/compound_events.shtm">compoundEventsController</prop>
<prop key="/data_point_details.shtm">dataPointDetailsController</prop>
<prop key="/data_point_edit.shtm">dataPointEditController</prop>
<prop key="/data_source_edit.shtm">dataSourceEditController</prop>
<prop key="/data_sources.shtm">dataSourceListController</prop>
```

2(c)



```
header.watchLists=Watch lists
header.systemSettings=System settings
header.emport=Import/Export
header.sql=SQL
header.logout=Logout
header.help=Help
```

2(d)

Fig2: Snippets of code from (a)page.tag; (b)MiscDwr.class; (c)springDisplatcher-servlet.xml; (d) messages_en.properties

2.2. Impact Analysis

2.2.1. *Solution 1:* Creating a new landingpage.jsp and landingpageController.class files. In page.tag file add a new icon png with key name as landingpage and a header linked to it in message_en.properties files named as landingpage.

2.2.1.1. The default URL in the MiscDwr.class file and successUrl for loginController in springDispatcher-servlet.xml shall also to be changed to the ‘landingpage.shtm’.

2.2.1.2. The default URL must be changed to ‘landingpage.shtm’, if it is cross-referenced in other files depending on MiscDwr.class, springDispatcher-servlet.xml.

2.2.2. *Solution 2:* Implementing Dynamic Customization, where the user can choose from variety of pre-made themes or widgets to customize their landing page.

2.2.2.1. Create frontend interfaces for customization, interact with pre-existing navigation and layout elements, and develop backend capabilities for managing user preferences and templates.

2.2.3. *Decision:* After carefully considering the complexity of the two approaches, we decided to implement Solution 1 as it is more feasible and can be easily implemented whereas Solution 2 takes longer to implement.

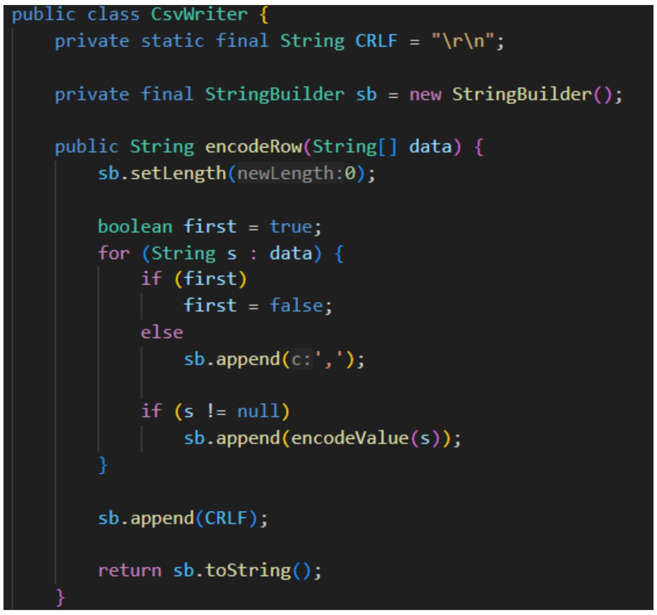
3. Horizontal CSV functionality

3.1. Concept Location

3.1.1. We started searching Mango Source file folder using keyword CSV and found 38 files out of which we found CsvWriter.java, ReportCsvStreamer.java, ChartExportServlet.java.

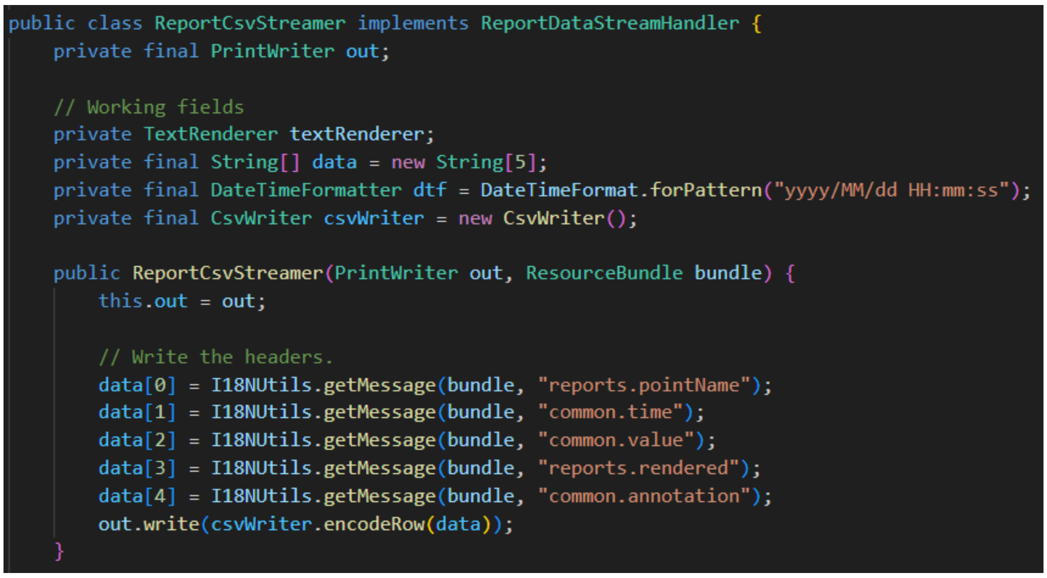
3.1.1.1. The ReportCsvStreamer.java takes in the necessary data and formats this data into .CSV file format using CsvWriter.java which contains functions that take in array of strings and output CSV file. ReportCsvStreamer.java organises the data, writes headers and footers and manages report generation.

3.1.1.2. The ChartExportServlet.java calls the ReportCsvStreamer.java in the report generation process.



```java
public class CsvWriter {
    private static final String CRLF = "\r\n";

    private final StringBuilder sb = new StringBuilder();

    public String encodeRow(String[] data) {
        sb.setLength(newLength:0);

        boolean first = true;
        for (String s : data) {
            if (first)
                first = false;
            else
                sb.append(c:',');

            if (s != null)
                sb.append(encodeValue(s));
        }

        sb.append(CRLF);

        return sb.toString();
    }
```

(a)



```java
public class ReportCsvStreamer implements ReportDataStreamHandler {
    private final PrintWriter out;

    // Working fields
    private TextRenderer textRenderer;
    private final String[] data = new String[5];
    private final DateTimeFormatter dtf = DateTimeFormat.forPattern("yyyy/MM/dd HH:mm:ss");
    private final CsvWriter csvWriter = new CsvWriter();

    public ReportCsvStreamer(PrintWriter out, ResourceBundle bundle) {
        this.out = out;

        // Write the headers.
        data[0] = I18NUtils.getMessage(bundle, "reports.pointName");
        data[1] = I18NUtils.getMessage(bundle, "common.time");
        data[2] = I18NUtils.getMessage(bundle, "common.value");
        data[3] = I18NUtils.getMessage(bundle, "reports.rendered");
        data[4] = I18NUtils.getMessage(bundle, "common.annotation");
        out.write(csvWriter.encodeRow(data));
    }
```

(b)

Fig3: Snippets of code from (a)CsvWriter.java, (b)ReportCsvStreamer.java

3.2. Impact Analysis

3.2.1. *Solution 1:* Update the existing CsvWriter.java file

3.2.1.1. This solution will modify the data array found in the constructor of the Report csv file. If we were to create a loop around the initial creation of this array, we would be able to expand the list horizontally for as many different points that are.

3.2.1.2. The condition for this loop would be to compare the current "PointName" with previous ones. If there is a new "PointName" then a new set of horizontal columns would be added.

3.2.1.3. The other methods within the class would also need to be altered to store the proper data in the correct elements in the new array.

3.2.2. Solution 2: Writing a new CsvWriter file which directly accommodates the condition to store each sensor horizontally in the file generated.

3.2.2.1. This requires an new set of utilities, encodeRow() and encodeValue() methods ensuring that the output file generated still follows the CSV file format standards and has the 'PointName' separated horizontally.

3.2.3. Decision: We decided on implementing Solution 1 as it is easier to incorporate a new logic on the existing code rather than to build an entirely new file with the required constraints.