Software Design Document
for
the Bad Area Detector (BAD)

Version 2.0.0 approved

Prepared by

Alex Rodriguez
Eduardo Gutierrez
Ka-Yee Yeung
Robert Martinez
Tanner Hoeflicker

California State University Los Angeles
March 15, 2019
# Table of Contents

- Revision History
  - pg 4

## 1. Introduction
  - pg 5
    - 1.1. Purpose
    - 1.2. Document Conventions
    - 1.3. Intended Audience and Reading Suggestions
    - 1.4. System Overview

## 2. Design Considerations
  - pg 7
    - 2.1. Assumptions and dependencies
    - 2.2. General Constraints
    - 2.3. Goals and Guidelines
    - 2.4. Development Methods

## 3. Architectural Strategies
  - pg 9
    - 3.1. Node.js Architectural Strategies
    - 3.2. Socket.io Architectural Strategies
    - 3.3. Cloud Object Storage (COS) Architectural Strategies
    - 3.4. React Architectural Strategies

## 4. System Architecture
  - pg 10

## 5. Policies and Tactics
  - pg 11
    - 5.1. Choice of which Specific Products Used
    - 5.2. Plans for ensuring requirements traceability
    - 5.3. Plans for testing the software
    - 5.4. Interfaces for end-users
    - 5.5. How to build and generate the system’s deliverables
    - 5.6. Plans for maintaining software

## 6. Detailed System Design
  - pg 13
    - 6.1. User Interface Module
    - 6.1.1. Responsibilities
    - 6.1.2. Constraints
    - 6.1.3. Composition
    - 6.1.4. Uses/Interactions
    - 6.1.5. Resources
    - 6.1.6. Interface/Exports
    - 6.2. IBM Cloud Analytics Module
    - 6.2.1. Responsibilities
    - 6.2.2. Constraints
    - 6.2.3. Composition
    - 6.2.4. Uses/Interactions
    - 6.2.5. Resources
    - 6.2.6. Interface/Exports
    - 6.3. User Account Authentication Module
    - 6.3.1. Responsibilities
    - 6.3.2. Constraints
    - 6.3.3. Composition
6.3.4 Uses/Interactions.................................................................................... pg 16
6.3.5 Resources............................................................................................... pg 16
6.3.6 Interface/Exports.................................................................................... pg 16
6.4 BAD Web App Service Module................................................................. pg 16
   6.4.1 Responsibilities................................................................................... pg 16
   6.4.2 Constraints......................................................................................... pg 16
   6.4.3 Composition......................................................................................... pg 16
   6.4.4 Uses/Interactions................................................................................. pg 16
   6.4.5 Resources............................................................................................ pg 16
   6.4.6 Interface/Exports................................................................................ pg 17
6.5 Cloud Object Storage Module................................................................... pg 17
   6.5.1 Responsibilities................................................................................... pg 17
   6.5.2 Constraints......................................................................................... pg 17
   6.5.3 Composition......................................................................................... pg 17
   6.5.4 Uses/Interactions................................................................................. pg 17
   6.5.5 Resources............................................................................................ pg 17
   6.5.6 Interface/Exports................................................................................ pg 17
6.6 Simulated Police Dispatch Module............................................................ pg 18
   6.6.1 Responsibilities................................................................................... pg 18
   6.6.2 Constraints......................................................................................... pg 18
   6.6.3 Composition......................................................................................... pg 18
   6.6.4 Uses/Interactions................................................................................. pg 18
   6.6.5 Resources............................................................................................ pg 18
   6.6.6 Interface/Exports................................................................................ pg 18
7. Detailed Lower level Component Design.................................................. pg 19
7.1 Index.js...................................................................................................... pg 18
   7.1.1 Classification....................................................................................... pg 18
   7.1.2 Processing Narrative(PSPEC)............................................................... pg 18
   7.1.3 Interface Description........................................................................... pg 18
   7.1.4 Processing Detail.................................................................................. pg 18
      7.1.4.1 Design Class Hierarchy................................................................. pg 18
      7.1.4.2 Restrictions/Limitations............................................................... pg 18
      7.1.4.3 Performance Issues..................................................................... pg 18
      7.1.4.4 Design Constraints..................................................................... pg 18
      7.1.4.5 Processing Detail For Each Operation........................................ pg 20
7.2 Node_modules............................................................................................ pg 20
   7.2.1 Classification....................................................................................... pg 20
   7.2.2 Processing Narrative(PSPEC)............................................................... pg 20
   7.2.3 Interface Description........................................................................... pg 20
   7.2.4 Processing Detail.................................................................................. pg 20
      7.2.4.1 Design Class Hierarchy................................................................. pg 20
      7.2.4.2 Restrictions/Limitations............................................................... pg 20
      7.2.4.3 Performance Issues..................................................................... pg 20
      7.2.4.4 Design Constraints..................................................................... pg 20
# Revision History

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Reason For Changes</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>11/30/2018</td>
<td>Initial Version</td>
<td>v1.0</td>
</tr>
<tr>
<td>Update</td>
<td>12/14/2018</td>
<td>Updated Version</td>
<td>v1.1</td>
</tr>
<tr>
<td>Final</td>
<td>3/15/2019</td>
<td>Final Version</td>
<td>v2.0</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Purpose

The purpose of this software is to assist the police in keeping the public safe by delivering valuable statistics to the general public. The software functions on an industrial internet of things (IIoT) platform called IBM Cloud which is a Platform as a Service (PaaS) developed by International Business Machines (IBM). This will ensure data is delivered in real time to the public. With this information, the general public and transportation services will be able to plan routes ahead of time to stay clear of dangerous areas.

1.2 Document Conventions

- Black highlighted words contain terminal/command prompt text.
- Bolded words contain sections that references to other sections
- Each section is intended for providing a different topic and documentation
- There are no priorities in this document as this structure is best read sequentially
- UI - User Interface
- BAD - Bad Area Detector

1.3 Intended Audience and Reading Suggestions

This documentation is intended for developers and testers. Developers can read the documentation to gain an understanding of how data is processed from the initial police dispatch call to the display of data on the UI. That makes this document a good source to use when troubleshooting the BAD system. The best way to read this document is in sequential order of the sections. Section 4 will provide the best insight to developers as it details IBM Cloud Services used in the software. This section also provides the major responsibilities of the software. Section 1.4 is best read by project managers and marketing staff as it provides general overview of the software.
1.4 System Overview

The main purpose of BAD is to increase public safety by alerting the general public of where dangerous police dispatch calls are being made. BAD will process the dispatch calls in real time to run various data analytics and show the user on a map where the calls were made so that they may avoid the area(s). This will in turn help the police as well by keeping citizens clear of their scene.

The modules essential to accomplishing these tasks are the User Interface Module (UI), Cloud Object Storage Module (COS), BAD Web App Module (WA) and the User Account Authentication Module (UAA). The COS takes the real time police dispatch data and stores it into IBM’s database and provides fast data transfer into the BAD WA. The BAD WA queries the COS to send required information to the UI. The UI provides the interface to the end user to see information about dispatch calls. The user is able to access this module by going through the UAA URL which provides the security for the entire BAD system.
2. Design Considerations

This section will highlight a few of the general constraints and how it impacted the development of the BAD system.

2.1 Assumptions and Dependencies

The only dependency that the user will need to have installed prior to operate this software is Node.js. The software uses Node.js as a runtime environment for JavaScript server side applications. The assumption that the user will run the software with an internet connection on their device to successfully connect to the database. The user is assumed to also have the latest version of their web browsers:

- Firefox 63.0+
- Google Chrome 70+
- Safari 12.0+

2.2 General Constraints

The design is heavily taken into consideration by utilizing different frameworks. The current design does not allow for displaying all markers onto the map at once. There will be a shift in using Cloud Object Storage (COS) with Express to resolve this design issue. Converting from a local database to COS will offer seamless filtering in the backend, because Ajax will be used to load thousands of data points onto the map. Ajax is used best when a software has to handle countless of data and displaying it onto a UI.

IBM provides a concise documentation of how to install the UI components onto a React project, however changing the css of the UI components was not provided in the documentation. This caused a constraint of having to manually change the code within the UI component to have it display the tables and graphs properly on different devices.

The software can only be accessed via website with the most updated browser, which includes mobile, laptops and tablets.
2.3 Goals and Guidelines

The goal of this software is to assist police in keeping the public safe by analyzing police dispatch calls and relaying the information to the general public. Other software goals include the following:

- Providing a robust and fast software
- Allows for filtering data based on set attributes
- A friendly UI with animated gauges, graphs and more
- Provides recent dispatched calls for the user
- Delivering a toggle functionality that will allow the user to switch between a heat map and the default map

2.4 Development Methods

The Agile software development method is used due to the ongoing process improvement. We self organize by dividing tasks in order to maximize workflow. However, we also have set leadership roles such as documentation lead, UI lead, UAA lead, data analytics lead and group lead. This method allows for flexible response to change, due to the self-organization which allows for team members to help each other when the need arise.
3. Architectural Strategies

The design goal is to make BAD a fast and robust software that informs the user of dispatched calls using a multitude of unique services provided by IBM such as UI components, asset services, etc. With the implementation of these various services, each one provides a function that help towards the end goal and the entirety of the system design. All libraries support JavaScript.

3.1 Node.js Architectural Strategies

Node.js is used as our runtime environment that utilizes Express, a web application framework, to build our system. Node.js also provides several libraries that make it much easier for back-end development, such as the require method, which is used to import files locally into your web application. Moreover, it provides libraries and functionality which makes it fast and simple to create robust web API’s.

3.2 Socket.io Architectural Strategies

The backbone of this architectural system comes from Socket.io. This JavaScript library provides real-time communication between web applications (server and client). Socket.io seamlessly does the communication between the server and the client in the backend and works great with large amounts of data.

3.3 Cloud Object Storage (COS) Architectural Strategies

Cloud Object Storage is a highly scalable cloud storage service provided by IBM. It provides storage, management, and access of data. Data stored in COS can be easily accessed through IBM’s self service portal or HTTP using a REST API with the help from IBM’s API for interacting with data stored in their object storage.

3.4 React Architectural Strategies

The front end of BAD is utilizing React, a JavaScript library for building the UI. React will allow for efficient update and rendering for the right components when data changes, which is key for BAD. React is also component-based, which allows for easy encapsulation and an organized directory.
4. System Architecture

The BAD architecture is summarized in the Context Diagram (DFD Level 0) given below. A more complete functional description is given in Section 6 of this document. The Context Diagram provides the overall structure of the software modules and all its inputs and outputs. The notation used corresponds to that defined for any Data Flow Diagram (DFD).

BAD Data Flow Diagram: Level 0

Figure 1
5. Policies and Tactics

This section will provide details of the tactics used for to create the BAD system.

5.1 Choice of which specific products used

- IDE
  - Microsoft Visual Studio Code
- Package Manager
  - Node Package Manager (npm)
- Libraries
  - Node.js
  - Express
  - Socket.io
  - React
- Database
  - IBM Cloud Object Storage

5.2 Plans for ensuring requirements traceability

Requirements are up to date and contain everything said in the following documentation. If immediate change is required, it will be evaluated and tested to ensure that it works best for the system. Upon successfully implementing an alternative approach or change, a follow up revision will be planned.

5.3 Plans for testing the software

As described earlier in section 2.4, the agile software development method is used by continuously testing the product through each stage of development.

5.4 Interfaces for end-users

As mentioned in section 2.1, the use of a browser is needed, so there will be multiple test done throughout the development of the BAD system to ensure the are no end-user issues. Testing the end-user interface is one of the main test that will be looked at thoroughly and is one of the many plans for maintaining the the software.
5.5 How to build and generate the system’s deliverables

As mentioned earlier in section 2.1, the user will need to have Node.js installed in their system. Also, the assumption that they have the most updated web browser.

1. The user will need to download this specific software from github, specifically the following link: https://github.com/rmartinez213/carbon-san_diego-webapp-react/tree/rob_branch(UI_V2)
2. Upon successfully installing the software, go into the root directory of the project and open command prompt/terminal based on your operating system.
3. Once in the command prompt/terminal type the following command: npm install. This will install the IBM’s Carbon UI components and a few other libraries needed to make the application run.
4. Then in the command prompt/terminal type the following command: npm start. If steps have been successful then the following message will appear on the command line/terminal: Starting the development server...
5. A web browser will then open up with the web application running locally on your device on port 3000.

5.6 Plans for maintaining software

BAD will be maintained by internal review of the modules every 6 months to ensure all the libraries and components are up to date and to prevent any form of errors in the application. If changes are made, the documentation will provide the information of the changes along with a date. The github repository will also provide the date of change.
6. Detailed System Design

**BAD Data Flow Diagram: Level 1**

![Diagram](image)

### 6.1 User Interface Module

#### 6.1.1 Responsibilities

The UI module responds to display data from past and current crime events. The UI displays this information via Google Maps that the user can interact with. The maps can be translocated spatially or the user can zoom in and out of a targeted area. There are gauges, tables and graphs that display the simulated data.

#### 6.1.2 Constraints

There is no constraints placed on this module.
6.1.3 Composition

BAD’s front end UI is composed of IBM’s design system that help with consistency and quality. Google Map React is also one of the major components in our software as it provides the ability for the user to interact with the map and displays markers based on crime location.

6.1.4 Uses/Interactions

The map will allow user to zoom in and out of a targeted area, and click on the markers of each crime. That marker will allow the user to see the information of that specific crime. The map will also provide the user with the option to switch from the normal map to a heat map. There will be options that will allow the user to filter different types of data, such as priority, call type and time simulation.

6.1.5 Resources

The resources required for this product are provided through the IBM’s Cloud services. The resources included are the Carbon UI components, COS, IBM CA and the UAA. The details of the backend of these services are provided in the documentation.

6.1.6 Interface/Exports

At the top of the UI, there is a bar graph and line graph that displays the crime based on priority, month and other attributes. These UI components will update visually as the data is simulated. The mid section contains gauges that will categorize the simulated calls based on call type. The bottom of the UI will contain a list that provides a detailed description of the call such as the address from where the call was made, call type, priority and more. The other bottom half of the UI will also display information via Google Maps that display description markers as the calls are simulated. The entire UI is displayed using the React library.

6.2 IBM Cloud Analytics Module

6.2.1 Responsibilities

The CA module uses data from past police dispatch calls stored in the COS to determine the total number of calls made over a given time period, analyze the frequency of call types and analyze days of the week with the most calls made.
6.2.2 Constraints

An instance of this service must first be instantiated then linked to a cloud-ready application.

6.2.3 Composition

CA allows for ingestion, analyzing and correlation of information as it arrives from data sources.

6.2.4 Uses/Interactions

By using the IBM CA Service, data scientist are allowed to run multiple analytics from real-time data sources. The result will allow you to monitor the information, configure and manage the events as they unfold.

6.2.5 Resources

Express and COS are needed because COS module will be delivering the data to the WA or the CA service as it is simulating.

6.2.6 Interface/Exports

Streaming Analytics v2 API will allow the application to programmatically interact with IBM Streams in the cloud providing the ability to start, stop and manage Streaming Analytic instances.

6.3 User Account Authentication Module

6.3.1 Responsibilities

The primary responsibilities of behavior of this module is used for centralized identity management to issue and validate users via IBM’s App ID service.

6.3.2 Constraints

This module shall be created before any other modules.

6.3.3 Composition

App ID (UAA) is comprised of OAuth2 and OpenID Connect, which are the authorization and authentication process.
6.3.4 Uses/Interactions
Client, as an application client, can be created in this module. An authorized grant type, scopes, authorities and redirect URI can be assigned when creating the client. User and group will be created in this module as well. Provides the ability to setup authentication, user management and distinct profiles from App ID service.

6.3.5 Resources
Valid email needed to setup users for BAD.

6.3.6 Interface/Exports
There are no interface/exports on this module.

6.4 BAD Web App
6.4.1 Responsibilities
The primary responsibilities of behavior of this module acts as a medium to allow the back-end information to be sent to the front-end UI.

6.4.2 Constraints
This module requires an instance of a Cloud Object Storage Service Module in the IBM Cloud.

6.4.3 Composition
The WA is comprised of an Express/React Node.JS application with a tool chain to the COS.

6.4.4 Uses/Interactions
This module is used to push the data of all the police calls to the UI.

6.4.5 Resources
Requires app to be created or deployed in the IBM Cloud.
6.4.6 Interface/Exports

There are no interface/exports on this module.

6.5 Cloud Object Storage

6.5.1 Responsibilities

The primary responsibilities of behavior of this module acts as a cloud storage that stores the police calls in the form of JSON data. Buckets are used as containers for storing objects that can be accessed over HTTP using a REST API.

6.5.2 Constraints

This module requires an instance of Cloud Object Storage to be instantiated into a cloud-ready application.

6.5.3 Composition

The composition of the Cloud Object Storage module follows a unique storage structure. Buckets are first created in order to store objects (data) into the cloud. Buckets also provide an organized way of storing data by allowing users to choose a unique name. Data is then stored into these buckets in the form of objects.

6.5.4 Uses/Interactions

This module is used to store the data of all the police calls and be made accessible via REST API.

6.5.5 Resources

The police data itself must be ready in JSON format in order to upload the data to the buckets in COS.

6.5.6 Interface/Exports

All the objects in the buckets are accessible by SQL query or HTTP using REST API.
6.6 Simulated Police Dispatch Module
6.6.1 Responsibilities

The primary responsibility of behavior for this module is simulating police dispatch calls by receiving data from the Cloud Object Storage in JSON format.

6.6.2 Constraints

This module requires the use of a Raspberry Pi 3 or hardware with similar specifications.

6.6.3 Composition

A police dispatch call simulates calls received at a 911 center. The data will include information about the location of crime, threat level, etc.

6.6.4 Uses/Interactions

This module is used to simulate police dispatch calls.

6.6.5 Resources

No resources available on the module.

6.6.6 Interface/Exports

SPD will export information in a JSON format.
7. Detailed Lower level Component Design

This section will describe the purpose of the files used to create the BAD system.

7.1 Index.js

7.1.1 Classification

Class

7.1.2 Processing Narrative (PSPEC)

Processes the server and instantiates a server connection.

7.1.3 Interface Description

Does not provide an interface as this is the main class for rendering the main component (App.js), which contains all of its subcomponents.

7.1.4 Processing Detail

This parent class instantiates the interface by rendering all of its sub components.

7.1.4.1 Design Class Hierarchy

Parent class

7.1.4.2 Restrictions/Limitations

No restrictions or limitations.

7.1.4.3 Performance Issues

No performance issues has occurred in this class.

7.1.4.4 Design Constraints

There were no design constraints.
7.1.4.5 Processing Detail For Each Operation

Processes all the back-end server operations including the parsing of JSON text into objects.

7.2 Node_modules

7.2.1 Classification

Node package.

7.2.2 Processing Narrative (PSPEC)

Node_modules contains all the libraries that are were discussed in section 5.1.

7.2.3 Interface Description

This package has no interface and is meant for holding all the libraries in a single directory.

7.2.4 Processing Detail

Processes all the libraries into a single directory that allows npm to access.

7.2.4.1 Design Class Hierarchy

Parent class.

7.2.4.2 Restrictions/Limitations

This restrictions of node_modules is to consist of only libraries that are used in this software.

7.2.4.3 Performance Issues

There were no form of performance issues experienced.

7.2.4.4 Design Constraints

There were no design constraints when working with this package.
7.2.4.5 Processing Detail For Each Operation

Processes all the node modules in a single package for easy access to perform operations.

7.3  src

7.3.1 Classification

Subsystem.

7.3.2 Processing Narrative (PSPEC)

Src is a subsystem that contains HTML files, JavaScript files and CSS files.

7.3.3 Interface Description

This subsystem has no interface and contains only files that are used for the development of the BAD system.

7.3.4 Processing Detail

Processes all the front-end development files into a single directory that are accessed from the root folder.

7.3.4.1 Design Class Hierarchy

Parent class.

7.3.4.2 Restrictions/Limitations

Restrictions of the subsystem only applies.

7.3.4.3 Performance Issues

There were no form of performance issues experienced.

7.3.4.4 Design Constraints

There were no design constraints when working with this package.
7.3.4.5 Processing Detail For Each Operation

Processes all the front-end interfaces, such as the HTML, JavaScript and CSS files.
8. Database Design

IBM’s COS is used as the database that holds all the police data that will be used in BAD. IBM has provided developers with an API that is meant for their COS and it is intended to help developers interact with the data they have stored in IBM’s cloud storage. All data that is returned from the database is in the form of JSON. COS also provides a REST API that allows for interaction between the database and the application.
9. User Interface

This section will provide further details of the user interface.

9.1 Overview of User Interface

The user interface can display data from past and current events. There is a menu bar at the bottom of the UI screen which will include the following buttons:

- **Simulate all data:**
  - This will display all police dispatch call information, due to the amount of the data, it will have some delay issue

- **Refresh 1:**
  - This button will allow user to decide to load the data every 1 second

- **Refresh 5:**
  - This button will allow user to decide to load the data every 5 seconds

- **Refresh 10:**
  - This button will allow user to decide to load the data every 10 seconds

- **Clear simulation:**
  - This button function as a reset button, it will regenerate the data from initial

- **Months:**
  - This is a drop down list and allow user to filter out the dispatch call data by month

- **Locations**
  - This will show markers via the zip code (beats) and display the total crimes of that police beat

Below the menu bar there are 4 sections. The top will display statistical data in the form of bar graph and line graphs. The middle will display information in the form of gauges on major crimes reported. The bottom left will display a list of all the total calls with detailed information about those calls. There are four different priority levels which are represented by different colors on a bar graph. The bottom right section displays crime events on a Google Maps.
9.2 Screen Frameworks or Images
9.3 User Interface Flow Model

The main UI page will provide a combination of all the interface components which include major crimes, priority call statistics and map components.

1. Web Components
   a. UI components from Carbon Design such as threshold bar, data visualization chart and a data table that logs each crime in a list.

2. Major Crimes Components
   a. Includes 5 gauges that takes the information of crimes being displayed.
   b. Will display the percentage of a certain category of crime between gauges.

3. Priority Call Stats Components
   a. Displays a horizontal bar graph that increments based on the refresh rate selected by the user, the crimes are divided by months and by 4 priorities in each month.

4. Map Components
   a. Displays the default and heat maps via Google Maps Api.
   b. Provides zoom functionality on the map object.
   c. Displays different colors for different priority call types placed on the map.
   d. Will display markers that groups crimes in groups of a police beat in one infowindow.
## 10. Requirements Validation and Verification

(Requirements 6.X reference 4.X in the SRD for the BAD Web App software)

<table>
<thead>
<tr>
<th>Requirement No.</th>
<th>Requirement Description</th>
<th>Description of the Testing Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.1</td>
<td>The UI will consist of multiple independent components.</td>
<td>The testing method requirement has been met. Tested and verified by standard software practices.</td>
</tr>
<tr>
<td>6.1.2</td>
<td>The UI shall include a toggle button on the menu to change between a heat map and the default map.</td>
<td>The testing method requirement has been met. The user can press the button to toggle the switch between the heatmap and the feature.</td>
</tr>
<tr>
<td>6.1.3</td>
<td>The UI shall have option to toggle analytical maps.</td>
<td>The testing method requirement has been met as there are multiple toggles that change the UI of the map.</td>
</tr>
<tr>
<td>6.1.4</td>
<td>The UI shall contain refresh options of 3, 5 and 10 seconds.</td>
<td>The testing method requirement has been met as there are 1, 5, and 10 second refresh rates available.</td>
</tr>
<tr>
<td>6.1.5</td>
<td>The UI shall allow the user to interact with the displayed map by moving it.</td>
<td>The testing method requirement has been met as there is map that provides interaction. The user can interact with the map when scrolling left or right to other cities or states.</td>
</tr>
<tr>
<td>6.1.6</td>
<td>The UI shall allow the user to zoom in and out of the map and click on the infowindow of each marker.</td>
<td>The testing method requirement has been met as it allows users to maneuver the map using zoom in and out functions provided by Google Maps API.</td>
</tr>
<tr>
<td>6.1.7</td>
<td>The UI shall have the option to display analyzed data.</td>
<td>The testing method requirement has been met as there are gauges and bar graphs.</td>
</tr>
</tbody>
</table>
### 6.1.8
The UI shall allow the user to search and filter any call from the list of police calls.  
The testing method has been met as there is a functioning search feature.

### 6.2 Cloud Analytics (CA)

<table>
<thead>
<tr>
<th>Requirement No.</th>
<th>Requirement Description</th>
<th>Description of the Testing Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.1</td>
<td>The CA shall analyze the total number of dispatch calls.</td>
<td>The testing method requirement has been tested and verified through statical calculations and output on the UI.</td>
</tr>
<tr>
<td>6.2.2</td>
<td>The CA shall analyze the frequency of each call type.</td>
<td>The CA was tested and verified by comparing with known values.</td>
</tr>
<tr>
<td>6.2.3</td>
<td>The CA shall analyze the days of the week with the most calls.</td>
<td>This requirement was tested and verified by selecting the corresponding menu option in a UI dropdown menu.</td>
</tr>
<tr>
<td>6.2.4</td>
<td>The CA shall send analyzed data to the RMD Data Source.</td>
<td>Requirement not implemented.</td>
</tr>
<tr>
<td>6.2.5</td>
<td>The CA shall make REST calls to the COS to request data in real-time and analyze them.</td>
<td>Requirement verified but not implemented into current UI.</td>
</tr>
</tbody>
</table>

### 6.3 User Account Authentication (UAA)

<table>
<thead>
<tr>
<th>Requirement No.</th>
<th>Requirement Description</th>
<th>Description of the Testing Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.1</td>
<td>The UAA shall provide a login page for the user.</td>
<td>The testing method requirement has been met. Tested and verified by standard user login-access trials as well as token expiration times provided by the UAA service.</td>
</tr>
</tbody>
</table>
6.3.2  The UAA shall send an email to the user for verification upon completing the registration form. The requirement has been functionally tested through multiple registered users.

6.3.4  The UAA shall redirect the user to the UI upon successful login. This requirement has been functionally validated through redirection of user to alternate URI which would substitute for the

<table>
<thead>
<tr>
<th>Requirement No.</th>
<th>Requirement Description</th>
<th>Description of the Testing Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.1</td>
<td>The WA shall retrieve information from the COS.</td>
<td>This requirement has not been met as of the current date.</td>
</tr>
<tr>
<td>6.4.2</td>
<td>The WA shall run on the IBM Cloud.</td>
<td>This requirement has been met and tested via a web browser by going the address of the WA and verifying REACT components function.</td>
</tr>
<tr>
<td>6.4.4</td>
<td>The WA shall send information to the UI.</td>
<td>This requirement has been tested via hardcoding information into the WA and verifying visually in the UI.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement No.</th>
<th>Requirement Description</th>
<th>Description of the Testing Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5.1</td>
<td>The COS shall receive data from the Watson application.</td>
<td>Once the SPD starts simulating data REST calls are made to the COS to see if data is present.</td>
</tr>
<tr>
<td>6.5.2</td>
<td>The COS shall store information in the form of JSON.</td>
<td>REST calls are made to the COS to ensure proper data format.</td>
</tr>
<tr>
<td>6.5.4</td>
<td>The COS shall have buckets for storing objects.</td>
<td>This requirement was verified at the creation of the COS Module in the IBM Cloud.</td>
</tr>
</tbody>
</table>
### 6.6 Simulated Police Dispatch (SPD)

<table>
<thead>
<tr>
<th>Requirement No.</th>
<th>Requirement Description</th>
<th>Description of the Testing Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.1</td>
<td>The SPD shall simulate a live police dispatch calls to the BAD.</td>
<td>Launch Watson Dashboard and check to see if data is streaming in on a CARD.</td>
</tr>
<tr>
<td>6.6.2</td>
<td>The SPD shall convert police dispatch convert call information to JSON file.</td>
<td>This requirement has been fulfilled through standard software assertion best practices.</td>
</tr>
<tr>
<td>6.6.3</td>
<td>The SPD shall make calls to the Watson IoT Service.</td>
<td>Service has been created in IBM Cloud. In the Watson Dashboard under devices the SPD RP is registered.</td>
</tr>
</tbody>
</table>
11. Glossary

**AM** – Asset Manager

**API** – Application Programming Interface

**BAD** – Bad Area Detector

**CA** – Cloud Analytics

**COS** – Cloud Object Storage

**CSS** – Cascade Style Sheet

**DFD** – Data Flow Diagram

**HTML** – Hypertext Markup Language

**IBM** – International Business Machines

**IoT** - Industrial Internet of Things

**JSON** – JavaScript Object Notation

**OS** – Operating System

**PaaS** – Platform-as-a-Service

**REST API** – RESTful Application Programming Interface

**SPD** – Simulated Police Dispatch Module

**UI** – User Interface

**UAA** – User Account Authentication

**WCAG** – Web Content Accessibility Guideline
12. References

- Setup IBM Cloud CLI for IBM Cloud:
  - https://console.bluemix.net/docs/cli/reference/ibmcloud/download_cli.html#install_use

- Understanding Deployment: How to Deploy Applications
  - https://console.bluemix.net/docs/cli/idt/index.html#developing

- Cloud Object Storage Starter

- Exploring Application Security: IBM Cloud ID
  - https://console.bluemix.net/docs/services/appid/index.html# gettingstarted

- Exploring IBM Analytics Services: Deploying an Analytics Framework instance
  - https://cloud.ibm.com/docs/services/StreamingAnalytics?topic=StreamingAnalytics -t_deploytocloud#t_deploytocloud

- Google Geocoding API:
  - https://developers.google.com/maps/documentation/geocoding/start