Crime Buster Application

System Design Document

Presented to

Mr. Jonathan Pautsch of

Next Century Corporation

Software Engineering I

CMSC 447

By

Angel Cheng

Sam Mendimasa

Katelyn Seitz

Zach Vance

6 April 2018

Crime Buster Application
System Design Document

Table of Contents

1. Introduction

1.1 Purpose of This Document

The purpose of this document is to describe the design of the Crime Buster web application. The important topics covered in this document include: a high level view of the system architecture, lower level class designs, and the data design. The intended audience for this document are the Baltimore Police Department, Next Century Corporation, Dr. Wilson of the University of Maryland, Baltimore County, and anyone who is interested in using or implementing the Crime Buster application.

1.2 References

1. Crime Buster Web Application System Requirements Specification Document
2. Crime Buster Web Application User Interface Design Document
3. Baltimore Police Department. "BPD Part 1 Victim Based Crime Data | Open Baltimore | City of Baltimore's Open Data Catalog." *City of Baltimore*, Socrata, data.baltimorecity.gov/Public-Safety/BPD-Part-1-Victim-Based-Crime-Data/wsfq-mvij.

2. System Architecture

2.1 Architectural Design

The CrimeBuster application was build using the Vue.js framework. The basic features of Vue.js include templates, reactivity in the form of models, components, and transitions.

Templates refers to the various parts of the page, such as body, footer, and etc. The major templates in the CrimeBuster application, are the side bar menu, the header, the footer, and the body. Vue.js uses html-based syntax, which will allow us to declaratively bind the DOM to our underlying data. We also used some CSS to style the templates in our application.

Reactivity is basically a form of state management as we manipulate various visualizations in our application. This is accomplished with Vue.js by Models, which are essentially javascript objects that, when modified, update the views. Also, with Vue.js, each component keeps tracks of it's reactive dependencies, making it easier for re-rendering.

Components in Vue are custom elements that we can attach a specific behavior to. This will correspond to various small portions of our application that we will be able to easily reuse, without the need of retyping everything. The components of CrimeBuster will be created based on the information our our database.

Transitions in Vue is how the system reacts to events such as inserting, updating, or removing elements from the DOM. This includes automatically applying classes for CSS, using third-party libraries, and using javascript to directly manipulate the DOM. Vue automatically locates and applies the target elements at the appropriate timing as transitions occurs.

2.2 Decomposition Description

Each Model corresponds to a javascript object. The objects represent the various visualizations that CrimeBuster has. Each object is able to interface with the component in which it resides, which is able to directly interact with our database, to pull the require data, as our users interact with it. For example, when a user re-renders the Map based on the Crime Type, the models are updated automatically using Vue.js, and since each model will keep track of it's own dependencies, our system will seem streamless to our users. This concept applies to the other models, in the Models Diagram above.

Crime Buster View Decomposition Diagram

Each model in the decomposition diagram corresponds to a visualization that the user can interact with. Each model is also attached to a specific javascript function that is called as the user interacts with the model, which generates events. The corresponding javascript function handles the events and determines which change was made. Using ajax and PHP, we are then able to obtain the corresponding data from the database, return it, and re-render the model to display the updated changes.

3. Persistent Data Design

3.1 Database Descriptions

The database for the CrimeBuster application is simple and straightforward. In order to ensure that we can update and render our visualizations quickly, we are storing the data pulled from the Open Baltimore website using their API into a relational database, SQLite. Our database will have three tables: crimes, users, and comments. A field called crimeID will serve as the primary key in the crimes table, as well as a foreign key in the comments table. The comments will have a field called commentsID as the primary key, and will have the userID, the primary key of the users table, as a foreign key. This will ensure that we can link the users, comments, and crimes in our system to the corresponding crimes that they comment on. The crimes table will contain all of the data about crimes in Baltimore, which is publicly available. This table will be updated biweekly, which is when the dataset on the OpenBaltimoe site is updated. This will ensure that our application is up to date and that our visualizations will portray current crime trends in baltimore city.

The comments table on the other hand, will be used to store user comments, timestamp of the comments, and the user's name and id that made the comment. Since our application is being built to work within a corporate SSO, once a new user access our site, we will have access to their ID, and as they comment on various crimes, we can then save their comments. The purpose of the users table is to store the prefered dashboard and default settings for a particular user. The schema for our database is below.

3.2 File Descriptions

This provides the organizational structure of our application and details of the purpose of each document in our application.

README.md: description of our project
Index.php: Home and main page of the application. Has all the code for the templates
Map.html: The Map visualization template for our application.
DB: Contains all files which serves as endpoints to retrieve information from the database

- db.php: setup and connect to the database for our application
- db/getActualTableData.php: Endpoint for retrieving dataset from the table visualization
- db/getActualMapLocations.php: Endpoint for retrieving dataset for the Map visualization

Scripts: Contains javascript functions for the application

- Script.js: contains functions that are used throughout crimebuster
- Vue.js: Contains scripts for using Vue framework

Styles: Contains css styles for classes and element ID's define in the application

- styles/w3.css: css styling taken from the w3schools site
- styles/Style2.css: additional css styling for our home page
- styles/Style.css: css styling for our home page

Images: folder that contains all of the images for the CrimeBuster application.

SSO: folder that contains files for setting up corporate SSO and getting users information

- sso/UMBCUtils.php: file that sets up sso for umbc.edu

Docs: Folder that contains all the written documents, such as the requirements and design for the CrimeBuster

4. Requirements Matrix

Please refer to the System Requirements Specification for details regarding the corresponding use cases.

Appendix A – Agreement Between Customer and Contractor

The customer agrees to a Crime Buster system with data visualizations, filtering options, and crime comments as some of the capabilities. Additional features will be provided in further development spirals. When and if future changes to this document occur a drafted new document will be created. Both a hard and electronic copy of both versions will be presented to the client for review. Upon approval, the draft will be finalized and signed off by both parties.

Client

Name: Date:

Signature

Comments

Team

Name: Date:

Signature

Name: Date:

Signature

Name: Date:

Signature

Name: Date:

Signature

Appendix B – Team Review Sign-off

This document has been collaboratively written by all members the team. Additionally, all team members have reviewed this document and agree on both the content and the format. Any disagreements or concerns are addressed in team comments below.

Team:

Name: Date:

Signature

Comments

Name: Date:

Signature

Comments

Name: Date:

Signature

Comments

Name: Date:

Signature

Comments

Appendix C – Document Contributions

Throughout the development of this document, each team member contributed in some way, hence the overall work distribution split evenly across all members (Angel 25%, Sam 25%, Katelyn 25%, and Zach 25%). The specific breakdown of work contribution is also divided across all sections, as we all worked on this document together.