**CrimeHawk**

Revision 1.0

CMSC 495: 6380

Group 5

Andrew Landis

Brent Phillips

Eske Gizaw

Lauren Rapp

[Project Plan 3](#_Toc45161668)

[**Revision History** 3](#_Toc45161669)

[**Requirement Specifications** 3](#_Toc45161670)

[**System Specification** 3](#_Toc45161671)

[**Development Platform** 3](#_Toc45161672)

[**Operating Platform** 3](#_Toc45161673)

[**Sub-Systems** 4](#_Toc45161674)

[**Software Management** 4](#_Toc45161675)

[**Project Schedule** 4](#_Toc45161676)

[Project Requirements 6](#_Toc45161677)

[Project Analysis 8](#_Toc45161678)

[**Input Data** 8](#_Toc45161679)

[**Sources of Input Data** 8](#_Toc45161680)

[**Output Data** 8](#_Toc45161681)

[**Destination of Output Data** 9](#_Toc45161682)

[**Convert Input Data to Output Data** 9](#_Toc45161683)

[**Systems that Interface with CrimeHawk** 9](#_Toc45161684)

[**Sub-Systems** 11](#_Toc45161685)

[**Sub-Systems Description** 11](#_Toc45161686)

[**Possible Enhancements** 13](#_Toc45161687)

[**Possible Risks & Mitigations** 13](#_Toc45161688)

[Project Design 14](#_Toc45161689)

[**Class Diagram** 15](#_Toc45161690)

[**Start-Up Scenario** 15](#_Toc45161691)

[**Shut-Down Scenario** 16](#_Toc45161692)

[**Sequence Diagram** 16](#_Toc45161693)

[**Normal Scenarios** 16](#_Toc45161694)

[**Error-Handling Scenarios** 19](#_Toc45161695)

[**Pseudocode for All Classes** 22](#_Toc45161696)

[**Unresolved Risk and Risk Mitigation** 23](#_Toc45161697)

[Project Interface Control (ICD) 24](#_Toc45161698)

[Interfaces 25](#_Toc45161699)

[**Heroku → NodeJS** 25](#_Toc45161700)

[**NodeJS ↔ Open Baltimore API** 25](#_Toc45161701)

[**NodeJS ↔ ReactJS** 26](#_Toc45161702)

[**NodeJS ↔ MongoDB** 26](#_Toc45161703)

[**User ↔ ReactJS** 27](#_Toc45161704)

**[User → Heroku](#_Toc45161705)** [27](#_Toc45161705)

[Project Test Plan 29](#_Toc45161706)

[User Guide 32](#_Toc45161707)

**[Introduction](#_Toc45161708)** [32](#_Toc45161708)

**[Requirements](#_Toc45161709)** [32](#_Toc45161709)

**[Installation](#_Toc45161710)** [33](#_Toc45161710)

**[Operation](#_Toc45161711)** [35](#_Toc45161711)

# **Project Plan**

## **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Date** | **Description** | **Name** |
| 1.0 | 6/3/2020 | Initial Draft | Brent Phillips, Andrew Landis, Lauren Rapp |
| 1.0.1 | 6/8/2020 | Sub-Systems | Brent Phillips, Lauren Rapp |

## **Requirement Specifications**

CrimeHawk is an application that aggregates crime statistical data in order to inform the general public of various crimes in a specific geographic location.

## **System Specification**

## **Development Platform**

**Hardware -** Minimum requirements are: a 1.6 GHz or faster processor, 1GB of RAM, 10GB of disk space, Windows, Linux or MAC operating systems.

**Software -** Visual Studio Code IDE, Node Package Manager, GIT and any modern browser (Mozilla Firefox, Safari, Google Chrome or Microsoft Edge).

## **Operating Platform**

**Hardware -** Minimum requirements are: a 1.6 GHz or faster processor, 1GB of RAM, 10GB of disk space, Windows, Linux or MAC operating systems.

**Software -** A modern browser (Mozilla Firefox, Safari, Google Chrome or Microsoft Edge) with JavaScript Enabled and a reliable internet connection.

## **Sub-Systems**

**User -** User can make selections by crime type and year, which returns crime data visualizations through interaction with ReactJS.

**Heroku** - The Heroku server is visited and starts service to NodeJS.

**NodeJS**  - NodeJS makes a request to the Open Baltimore API and receives output.

**MongoDB**  - MongoDB database management system accepts data, and provides data from NodeJS when requested.

**Open Baltimore API** - Open Baltimore API accepts requests from NodeJS and provides crime data.

**ReactJS**  - ReactJS builds the user interface responsible for accepting input from the user, requesting data from NodeJS, and returning data to the user.

## **Software Management**

The team will be using git as our versioning software. We have a private repository that is being hosted by github.com that the team has access to in order to create, read, update and delete project files. For documentation purposes, the team will utilize the Google docs service which will allow for real time collaboration.

## **Project Schedule**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | **Duration** | **Start Date** | **End Date** | **Personnel** |
| 1.   Project Requirements       a.       Writing       b.       Review       c.       Revise Document | 7 days 4 days 2 days 1 day | 05/27 | 06/02 | Andrew, Brent Eske, Lauren, Brent Andrew, Brent |
| 2.   Project Analysis       d.       Analyzing       e.       Review       f.        Revise Document | 7 days 4 days 2 days 1 day | 06/03 | 06/09 | Lauren, Eske All Brent, Andrew |
| 3.   Project Design       g.       Write Outline       h.       Review       i.         Revise Document | 7 days 4 days 2 days 1 day | 06/10 | 06/16 | Andrew, Lauren All Brent, Eske |
| 4.   Project test plan and ICD       j.         Create ICD       k.       Review       l.         Revise | 7 days 5 days 1 days 1 day | 06/17 | 06/23 | Brent, Eske All Lauren, Andrew |
| 5.   Sprint 1 – Implementation and Testing       m.     Sprint 1       n.       Review       o.       Retrospective | 7 days  5 days 1 day 1 day | 06/24 | 06/30 | All participate  Back-end – Andrew, Eske, Front-end – Lauren, Both - Brent |
| 6.   Sprint 2 – Implementation and Testing       p.       Sprint 2       q.       Review       r.        Retrospective | 7 days  5 days 1 day 1 day | 07/01 | 07/07 | All participate  Back-end – Andrew, Eske, Front-end – Lauren, Both - Brent |
| 7.   Final Sprint - Delivery (Code, Binaries, Test Data and User’s Guide)       s.        Final Sprint       t.        Documentation       u.       Review       v.       Revise | 7 days  3 days 2 days 1 day 1 day | 07/08 | 07/14 | All participate   Brent, Andrew, Eske Lauren, Brent All All |

# **Project Requirements**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Date** | **Description** | **Name** |
| 1.0 | 6/3/2020 | Initial Draft | Andrew Landis, Eske Gizaw |
|  |  |  |  |

**Topic**

CrimeHawk displays Burglary, Homicide, and Theft crimes in Baltimore city using Table, pie chart, and graphs. The data displayed is decided by dropdowns selected by the user.

**Project Requirements**

Figure 1 shows the top-level requirements for CrimeHawk.



**Figure 1** - CrimeHawk Top-Level Requirements

Table 1 lists the functional requirements for CrimeHawk.

**Table 1** - CrimeHawk Functional Requirements

|  |  |
| --- | --- |
| **Req ID** | **Requirement Text** |
| 1.0 | CrimeHawk shall display crime data. |
| 1.1 | CrimeHawk shall display homicide crime data. |
| 1.1.1 | CrimeHawk shall display homicides per month over the last 12 months. |
| 1.1.2 | CrimeHawk shall display homicide locations. |
| 1.1.3 | CrimeHawk shall display details of homicides over the last month. |
| 1.1.4 | CrimeHawk shall display the types of weapon used in homicides over the last 12 months. |
| 1.2 | CrimeHawk shall display burglary crime data. |
| 1.2.1 | CrimeHawk shall display burglaries per month over the last 12 months. |
| 1.2.2 | CrimeHawk shall display burglary locations. |
| 1.2.3 | CrimeHawk shall display details of burglaries over the last month. |
| 1.3 | CrimeHawk shall display theft crime data. |
| 1.3.1 | CrimeHawk shall display thefts per month over the last 12 months. |
| 1.3.2 | CrimeHawk shall display theft locations. |
| 1.3.3 | CrimeHawk shall display details of thefts over the last month. |
| 1.3.4 | CrimeHawk shall display the types of thefts over the last 12 months. |
| 1.3.5 | CrimeHawk shall display crime data based on the year selected by the user. |
| 1.3.6 | CrimeHawk shall display crime data based on the crime type selected by the user. |
| 2.0 | CrimeHawk shall get Crime data. |
| 2.1 | CrimeHawk shall get Crime data from the linked endpoint: [link](https://data.baltimorecity.gov/Crime/Crimes-By-Neighborhood/nhwe-7c7x). |
| 3.0 | CrimeHawk shall store Crime data. |
| 3.1 | CrimeHawk shall store Crime data in a database. |
| 3.2 | CrimeHawk shall store homicide data. |
| 3.3 | CrimeHawk shall store burglary data. |
| 3.4 | CrimeHawk shall store theft data. |

# **Project Analysis**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Date** | **Description** | **Name** |
| 1.0 | 6/3/2020 | Initial Draft | Andrew Landis, Brent Phillips, Eske Gizaw, Lauren Rapp |
| 1.0.1 | 6/8/2020 | Input/Output Data | Brent Phillips, Lauren Rapp, Andrew Landis, Eske Gizaw |

### **Input Data**

* User
  + Year selection
  + Crime type selection
* Open Baltimore API
  + Crime Date
  + Crime Type
  + Crime Location
  + Crime Description
  + Crime Weapon
  + Crime District
  + Crime Neighborhood

### **Sources of Input Data**

* User
* CrimeHawk
* Open Baltimore API

### **Output Data**

* Crime Data
* Crime Data Visualizations

### **Destination of Output Data**

* The output data will be displayed on the CrimeHawk User Interface. It will be displayed in two forms. The first form will be in the form of a crime data table and the other will be a crime data visualization.

### **Convert Input Data to Output Data**

* When the CrimeHawk application is initiated, a generic call is made to the Open Baltimore API to retrieve crime data.
* That data will then be stored in a cloud based version of MongoDB.
* The User will then see an interface with two different select box options and a button.
* Option one will be for crime type.
* Option two will be for crime year.
* If the User selects a crime type and clicks the button, a call to the Open Baltimore API is made. This data will be returned and the User interface will be updated accordingly.

### **Systems that Interface with CrimeHawk**

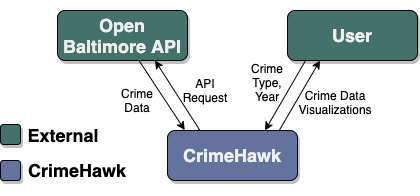


Figure 1: Context Diagram

#### **Outside Systems**

**Open Baltimore API** - This API is the data source for CrimeHawk. CrimeHawk will be making GET calls only.

**MongoDB Atlas** - MongoDB Atlas is a cloud based NoSQL database solution. The data is stored in a cloud environment and CrimeHawk will be able to access.

**Heroku** - Heroku is a free cloud based server that we are using to run CrimeHawk as a web application. It is integrated with our code repository and has been configured to automatically build and deploy subsequent versions of CrimeHawk.

**NodeJS** - NodeJS is a JavaScript library that makes it possible to write server (backend) code for an application entirely in JavaScript. CrimeHawk uses NodeJS as a backbone to the appliaction.

**ReactJS** - ReactJS is a JavaScript library that allows developers to create real time single page applications.  CrimeHawk will use ReactJS to handle when something gets rendered by the application.

#### **Input Data and Sources**

* The input selection for Crime Type (ReactJS Application)
* The input selection for Crime Year (ReactJS Application)

The inputs from the ReactJS application will be submitted as a GET request to the Open Baltimore API. The results will differ based on the inputs passed by the ReactJS application.

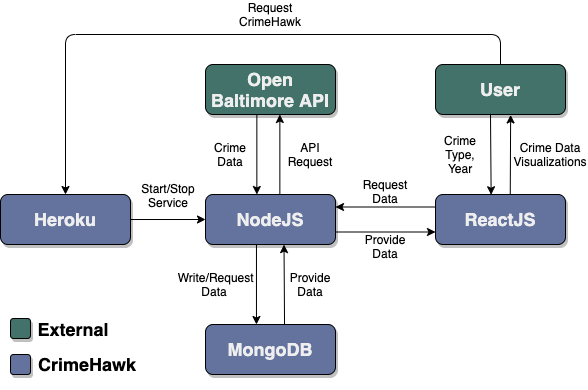
#### **Output Data and Destinations**

* Crime Data Visual Display - The ReactJS application will make a request to the Open Baltimore API. Based on the response from the API, a visual chart will be generated and displayed on the CrimeHawk application.
* Crime Data Table - The ReactJS application will make a request to the Open Baltimore API. Based on the response from the API, a crime data table will be generated and displayed on the CrimeHawk application.

#### **Data Processing**

CrimeHawk will take input given by the User and make a request to the Open Baltimore API. The Open Baltimore API will return a result as JavaScript notation (JSON). NodeJS will provide this data to the ReactJS application. CrimeHawk will then display the results with a crime data visualization and a crime data table.

### **Sub-Systems**



### **Sub-Systems Description**

**User -** User can make selections by crime type and year, which returns crime data visualizations through interaction with ReactJS.

**Heroku** - The Heroku server is visited and starts service to NodeJS.

**NodeJS**  - NodeJS makes a request to the Open Baltimore API and receives output.

**MongoDB**  - MongoDB database management system accepts data, and provides data from NodeJS when requested.

**Open Baltimore API** - Open Baltimore API accepts requests from NodeJS and provides crime data.

**ReactJS**  - ReactJS builds the user interface responsible for accepting input from the user, requesting data from NodeJS, and returning data to the user.

Sub-Systems Interface Data

**User –**User can input crime type and year to ReactJS and User receives data output.

**NodeJS**  - NodeJS inputs a request to the Open Baltimore API and

receives output.

**MongoDB**  - MongoDB accepts data input that NodeJS receives from Open Baltimore

API and stores it until output is requested from NodeJS.

**Open Baltimore API** – This database

provides output to NodeJS.

**ReactJS**  - ReactJS receives input from the user and gives input to NodeJS, then receives output from NodeJS and relays that crime visualization data to the user.

**Table 1** - CrimeHawk Sub-System Requirement Mapping

|  |  |  |
| --- | --- | --- |
| **Req ID** | **Requirement Text** | **Sub-Systems** |
| 1.0 | CrimeHawk shall display crime data. | All |
| 1.1 | CrimeHawk shall display homicide crime data. | All |
| 1.1.1 | CrimeHawk shall display homicides per month over the last 12 months. | All |
| 1.1.2 | CrimeHawk shall display homicide locations. | All |
| 1.1.3 | CrimeHawk shall display details of homicides over the last month. | All |
| 1.1.4 | CrimeHawk shall display the types of weapon used in homicides over the last 12 months. | All |
| 1.2 | CrimeHawk shall display burglary crime data. | All |
| 1.2.1 | CrimeHawk shall display burglaries per month over the last 12 months. | All |
| 1.2.2 | CrimeHawk shall display burglary locations. | All |
| 1.2.3 | CrimeHawk shall display details of burglaries over the last month. | All |
| 1.3 | CrimeHawk shall display theft crime data. | All |
| 1.3.1 | CrimeHawk shall display thefts per month over the last 12 months. | All |
| 1.3.2 | CrimeHawk shall display theft locations. | All |
| 1.3.3 | CrimeHawk shall display details of thefts over the last month. | All |
| 1.3.4 | CrimeHawk shall display the types of thefts over the last 12 months. | All |
| 1.3.5 | CrimeHawk shall display crime data based on the year selected by the user. | All |
| 1.3.6 | CrimeHawk shall display crime data based on the crime type selected by the user. | All |
| 2.0 | CrimeHawk shall get Crime data. | NodeJS, Open Baltimore API, MongoDB |
| 2.1 | CrimeHawk shall get Crime data from the linked endpoint: [link](https://data.baltimorecity.gov/Crime/Crimes-By-Neighborhood/nhwe-7c7x). | NodeJS, Open Baltimore API |
| 3.0 | CrimeHawk shall store Crime data. | MongoDB, NodeJS |
| 3.1 | CrimeHawk shall store Crime data in a database. | MongoDB |
| 3.2 | CrimeHawk shall store homicide data. | MongoDB, NodeJS |
| 3.3 | CrimeHawk shall store burglary data. | MongoDB, NodeJS |
| 3.4 | CrimeHawk shall store theft data. | MongoDB, NodeJS |

### **Possible Enhancements**

* Allow the user to select a different state and city.
* Export crime data tables as Excel and CSV files.
* Add a User registration system.
* Add a Login system.
* Track User’s search history.
* Allow Users to create a favorite search.
* Allow User to see side-by-side comparisons

### **Possible Risks & Mitigations**

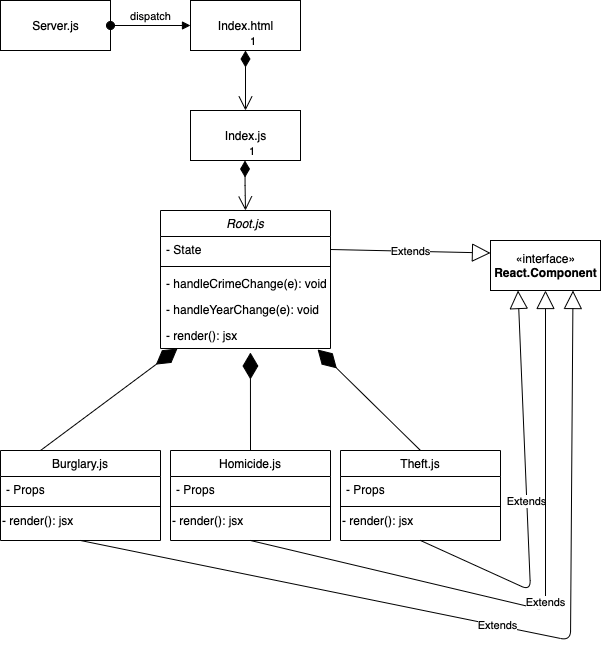
* The application needs a constant data update and since this application uses API to get the data from a state database, there will be a copy of data that will be available in case there is a system issue.
* The Cloud based database system may go offline.
* The front-end uses dropdowns to determine the data that should be displayed to the user, this mitigates an input errors from the user

# **Project Design**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Date** | **Description** | **Name** |
| 1.0 | 6/10/2020 | Initial Draft | Andrew Landis, Brent Phillips, Eske Gizaw, Lauren Rapp |
| 1.0.1 | 6/12/2020 | Diagrams and Pseudocode Code | Andrew Landis |
| 1.0.2 | 6/13/2020 | Start Up, Shut Down, Scenarios | Brent Phillips |
| 1.0.3 | 6/13/2020 | Error-Handling Scenarios and Unsolved Risk and Risk Mitigation | Eske Gizaw, Andre Landis |

## **Class Diagram**



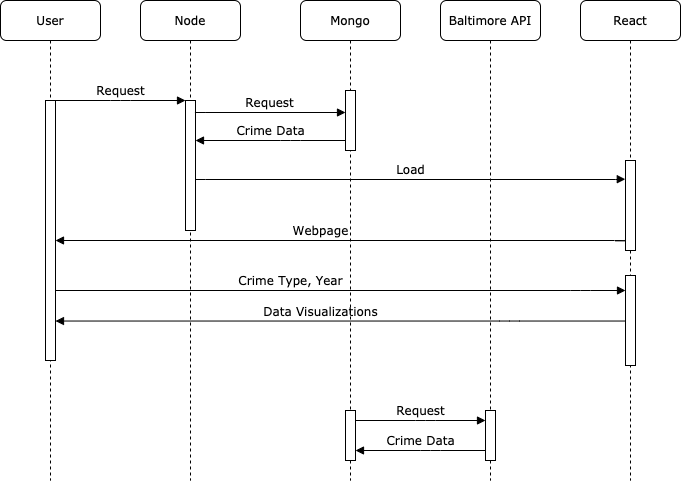
## **Start-Up Scenario**

1. The User visits the application.
2. The Heroku server receives a request to display the application.
3. The Heroku server starts the application.
4. NodeJS Attempts to connect to the MongoDB Database.
5. ReactJS Attempts to connect to the Open Baltimore API.
6. ReactJS Sends data from Open Baltimore API to NodeJS.
7. NodeJS Sends data to the MongoDB Database.
8. NodeJS Makes the data available to ReactJS.
9. ReactJS Displays the data.

## **Shut-Down Scenario**

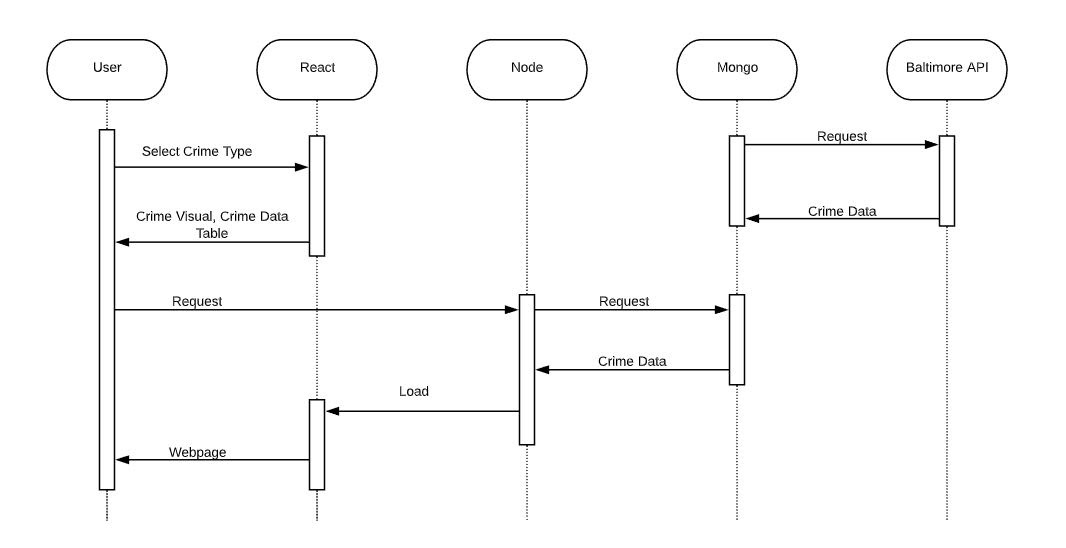
1. The User closes or exits the application.
2. NodeJS Closes the connection with the MongoDB Database.
3. NodeJS Stops all processes.
4. ReactJS Stops all processes.
5. The Heroku server Stops all processes.

## **Sequence Diagram**

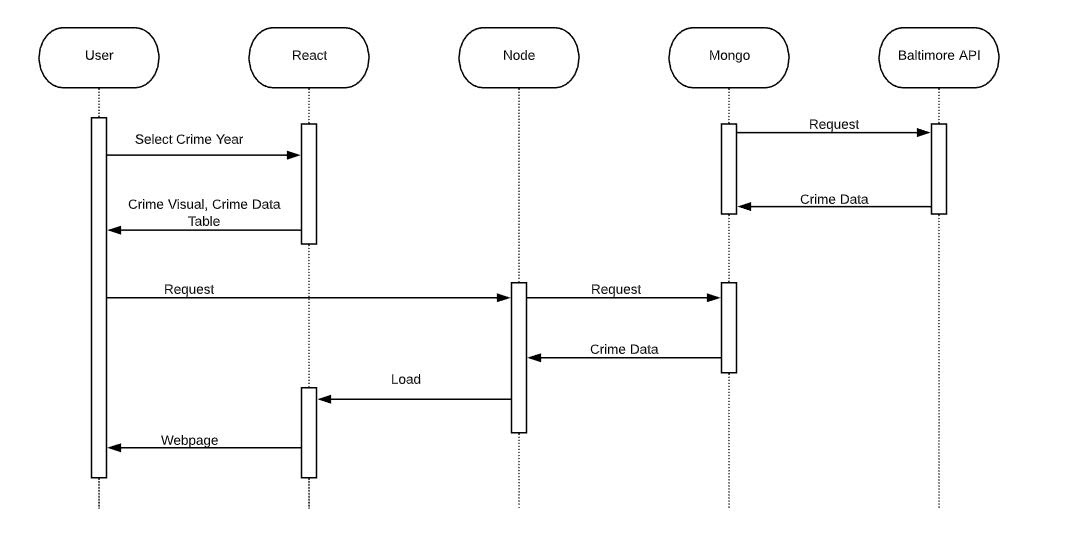


## **Normal Scenarios**

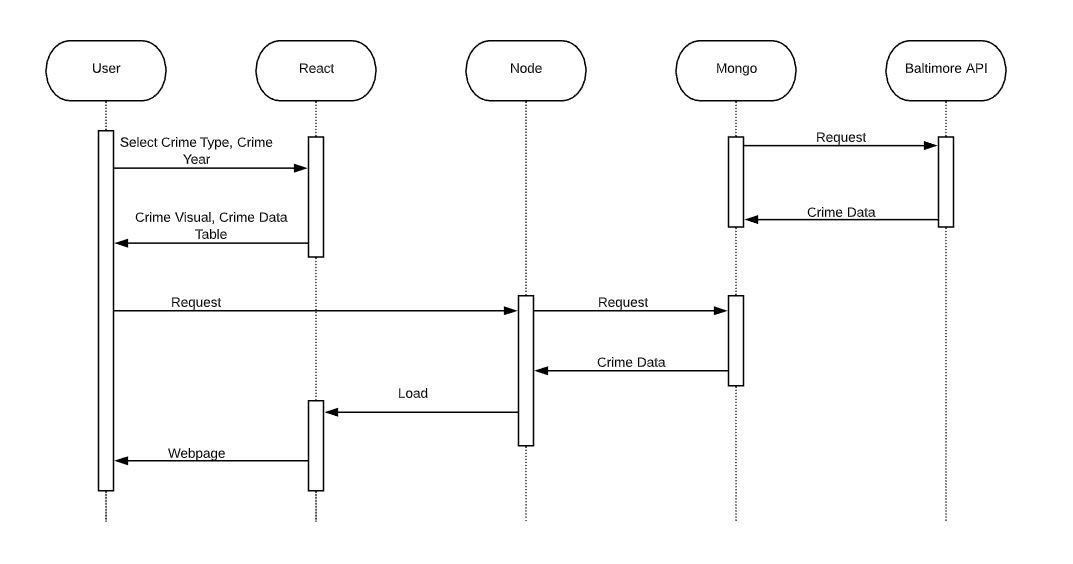
* Scenario 1
  + Pre-Condition: User chooses a Crime Type
  + Post-Condition: User will see a Crime Visual and a Crime Data Table
  + Description: The User will choose a Crime Type from the available selection element. The User will then click on the GO button. CrimeHawk will make a request to the Open Baltimore API. CrimeHawk will take the results of the request and display the Crime Visual and Crime Data Table to the User.



* Scenario 2
  + Pre-Condition: User chooses a Crime Year
  + Post-Condition: User will see a Crime Visual and a Crime Data Table
  + Description: The User will choose a Crime year from the available selection element. The User will then click the GO button. CrimeHawk will make a request to the Open Baltimore API. CrimeHawk will take the results of the request and display the Crime Visual and Crime Data Table to the User.

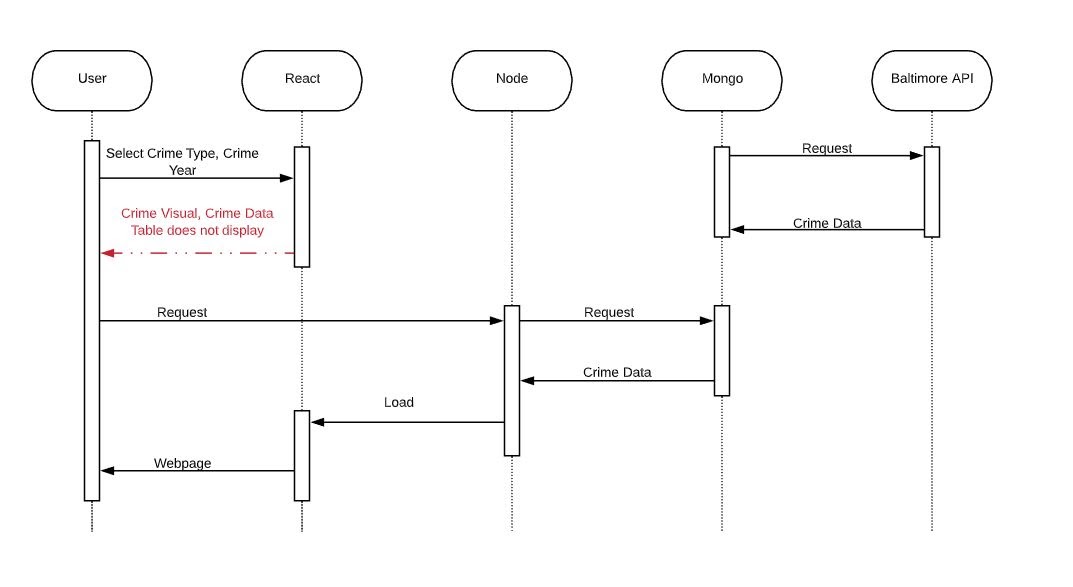


* Scenario 3
  + Pre-Condition: Users chooses a Crime Type and Crime Year
  + Post-Condition: User will see a Crime Visual and a Crime Data Table
  + Description: The User will choose a Crime Type and Crime Year from the available selection elements. The User will then click the GO button. CrimeHawk will make a request to the Open Baltimore API. CrimeHawk will take the results of the request and display the Crime Visual and Crime Data table to the User.

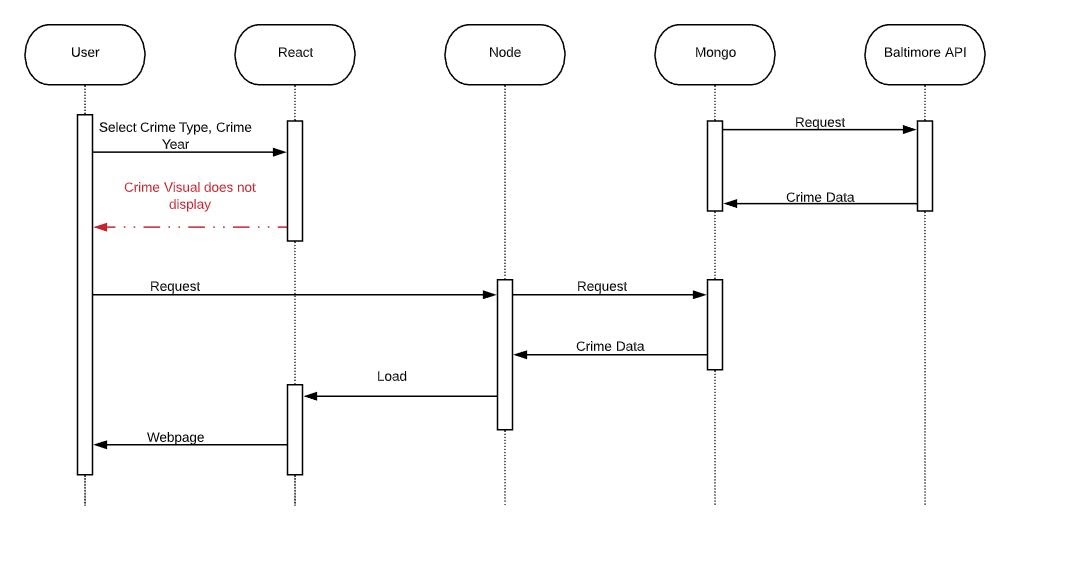


## **Error-Handling Scenarios**

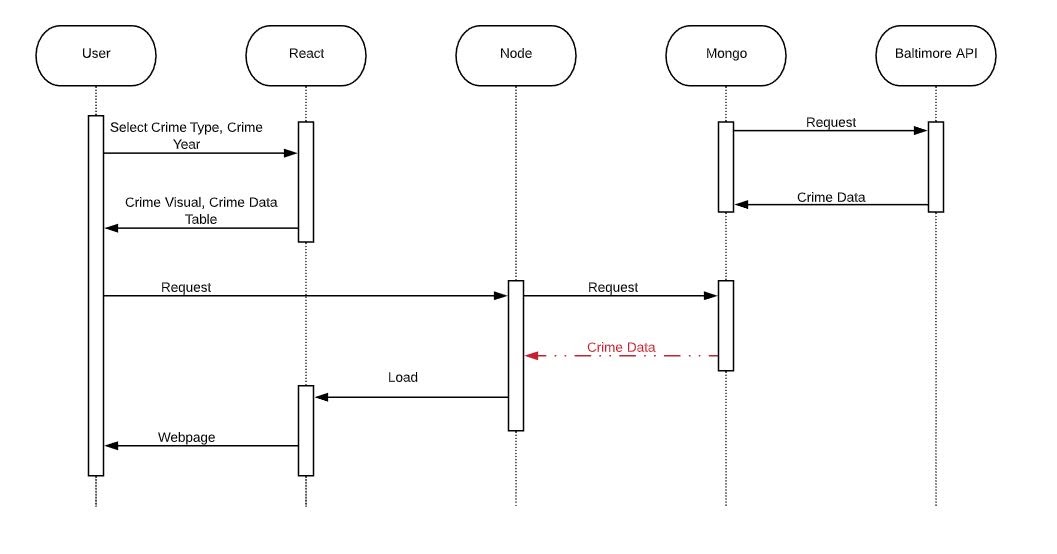
* Scenario 1
  + Pre-Condition: User chooses a Crime Type and Crime Year from dropdown
  + Post-Condition: User doesn’t see current Crime Visual and Crime Data Table
  + Description: The user chooses Crime Type and Crime Year. The User gets live updated data from Open Baltimore API and mongoDB will be used as a backup that updates at midnight.



* Scenario 2
  + Pre-Condition: User chooses a Crime Type and Crime Year
  + Post-Condition: User only sees Crime Data Table
  + Description: The user chooses Crime Type and Crime Year, CrimeHawk makes the request to the Open Baltimore API but the website only displays the Crime Data Table.



* Scenario 3
  + Pre-Condition: User chooses a Crime Type and Crime Year
  + Post-Condition: User sees live Crime Type and Crime Year updated data, but Cloud based database goes offline
  + Description: The user will choose a Crime Type and Crime Year and CrimeHawk displays a Crime Visual and Crime Data Table.



## **Pseudocode for All Classes**

Node:

* app = express()
* app.get(<send CrimeHawk>)
* app.get(<send crime data to react>)
* MongoClient.connect(<create crime collection>)
* getCrimeData(

fetch(baltimoreAPI, crimeData)

saveCrimeData(crimeData)

)

* saveCrimeData(

MongoClient.collection(‘CrimeHawk’).updateOne({upsert: true})

)

Mongo:

* External system, responds to MongoClient requests in Node

Baltimore API:

* External system, responds to Node requests with crime data

React:

* Index.js
* crimeData = fetch(to node for crime data)
* ReactDOM.render(<Root data=crimeData />)
* Root:
* Props: crimeData
* State: crimeType, year
* handleCrimeChange(e) => {

setState(user input crime type)

}

* handleYearChange(e) => {

setState(user input year)

}

* render() {

this.state.crimeType && <Burglary crimeData year />

this.state.crimeType && <Homicide crimeData year />

this.state.crimeType && <Theft crimeData year />

}

* Burglary:
* Props: crimeData, year
* render() {

<HighCharts />

}

* Homicide:
* Props: crimeData, year
* render() {

<HighCharts />

}

* Theft:
* Props: crimeData, year
* render() {

<HighCharts />

}

## **Unresolved Risk and Risk Mitigation**

* The application needs a constant data update and since this application uses API to get the data from a state database, there will be a copy of data that will be available in case there is a system issue.

**Risk mitigation**: Cloud based database, mongoDB will be used to store backup data that updates at midnight.

The cloud based database may go offline.  
**Risk mitigation:** CrimeHawk website uses cloud based database as a backup. Website may still be functional and Crime Type and Crime Year data will still be available via live update from Open Baltimore API.

# **Project Interface Control (ICD)**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Date** | **Description** | **Name** |
| 1.0 | 6/20/2020 | Initial Draft | Andrew Landis, Eske Gizaw |
| 1.0.1 | 6/21/2020 | Introduction Added | Andrew Landis |
| 1.0.2 | 6/22/2020 | Interfaces Added | Andrew Landis, Eske Gizaw |

Figure 1 below shows the interfaces that are detailed in this document.

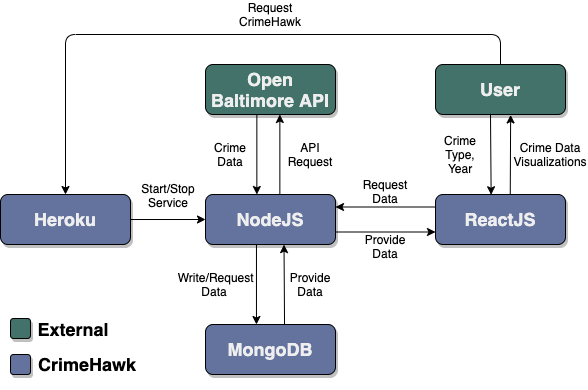


Figure 1 - CrimeHawk Interfaces

The interfaces will be broken down in the following order:

* Heroku → NodeJS
* NodeJS ↔ Open Baltimore API
* NodeJS ↔ ReactJS
* Node JS ↔ MongoDB
* User ↔ ReactJS
* User → Heroku

# Interfaces

## **Heroku → NodeJS**

Heroku is responsible for starting and stopping the server that runs CrimeHawk and is accessible via the internet. Table 1 details the interface from Heroku to NodeJS. NodeJS does not communicate back to Heroku.

Table 1 - Interface for Heroku and NodeJS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| Heroku | NodeJS | Start/Stop service | npm command | N/A | npm start |

## **NodeJS ↔ Open Baltimore API**

NodeJS is the server that runs CrimeHawk and requests data from the Open Baltimore API via an HTTP GET request. The Open Baltimore API responds with a JSON object of the Crime Data. Table 2 details the interface between NodeJS and the Open Baltimore API.

Table 2 - Interface for NodeJS and the Open Baltimore API

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| NodeJS | Open Baltimore API | HTTP GET | HTTP Request | N/A | Method: GET |
| Open Baltimore API | NodeJS | CrimeData | Array of:  {    crimeDate: string    crimeCode: string    location: string    description: string  } | Null or string for each field | {    crimeDate: null    crimeCode: null    location: null    description: null  } |

## **NodeJS ↔ ReactJS**

NodeJS serves the CrimeHawk ReactJS front-end with the crime data when a request comes in for the CrimeHawk URL. ReactJS then provides a user interface to the user with the crime data. Table 3 details the interface between NodeJS and ReactJS.

Table 3 - Interface for NodeJS and ReactJS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| NodeJS | ReactJS | CrimeData | Array of:  {    crimeDate: string    crimeCode: string    location: string    description: string  } | Null or string for each field | {    crimeDate: null    crimeCode: null    location: null    description: null  } |
| ReactJS | NodeJS | HTTP GET | HTTP Request | N/A | Method: GET  Path: /crime\_data |

## **NodeJS ↔ MongoDB**

Once the NodeJS server receives the data from the Open Baltimore API, it writes the data to a MongoDB Atlas server. When NodeJS receives a request from CrimeHawk, it requests the JSON object from the same MongoDB Atlas server. Table 4 details the interface between NodeJS and MongoDB.

Table 4 - Interface for NodeJS and MongoDB

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| NodeJS | MongoDB | CrimeData | Array of:  {    crimeDate: string    crimeCode: string    location: string    description: string  } | Null or string for each field | {    crimeDate: null    crimeCode: null    location: null    description: null  } |
| MongoDB | NodeJS | CrimeData | Array of:  {    crimeDate: string    crimeCode: string    location: string    description: string  } | Null or string for each field | {    crimeDate: null    crimeCode: null    location: null    description: null  } |

## **User ↔ ReactJS**

When the user makes Crime Type and year selection, CrimeHawk ReactJS user interface displays crime data. Table 5 details  the interface between user and ReactJS.

Table 5- Interface for User and ReactJS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| User | ReactJS | Select Crime Type, and Crime Year | Array of:  {    crimeType: string    crimeYear: Year  } | String | {    crimeType: null    crimeYear: null  } |
| ReactJS | User | Crime Visual, Crime Data Table | Array of:  {    crimeDate: string    crimeCode: string    location: string    description: string  } | Null or string for each field | {    crimeDate: null    crimeCode: null    location: null    description: null  } |

## **User → Heroku**

User visits the CrimeHawk application, a request is sent to the Heroku server and responds with CrimeHawk.

  Table 6 - Interface for User and Heroku

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| User | Heroku | GET | HTTP Request | N/A | / |
| Heroku | User | CrimeHawk | HTML, CSS, and JS | CrimeHawk Application | N/A |

# **Project Test Plan**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Date** | **Description** | **Name** |
| 1.0 | 6/10/2020 | Initial Draft | Andrew Landis, Brent Phillips, Eske Gizaw, Lauren Rapp |
| 1.0.1 | 6/20/2020 | Added Test Cases | Lauren Rapp |
| 1.0.2 | 6/21/2020 | Added Test Cases | Brent Phillips |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test** | **Requirement** | **Description** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| 1 | 1.0 | User visits the application and CrimeHawk application is displayed | CrimeHawk application will display and User can select Crime Type and Crime Year from dropdowns to display Crime Data | CrimeHawk application displays and User can select Crime Type and Crime Year from dropdowns to display Crime Data | Pass |
| 2 | 1.0 | User visits the application and CrimeHawk application is displayed, but url() filepath doesn not load CrimeHawk background image | The filepath does not work, but a secondary background image will display. CrimeHawk application will display and User can select Crime Type and Crime Year from dropdowns to display Crime Data | A secondary image displays when the first image does not. User can select Crime Type and Crime Year | Pass |
| 3 | 1.1.1, 1.1.2, 1.1.3, 1.1.4 | User Selects Homicide from the dropdown | Crime Data Visualization and Table will display homicides over the last 12 months, homicide locations, weapons used; details of homicides in last month | Crime Data Visualization and Table display homicides over the last 12 months, homicide locations, weapons used; details of homicides in last month | Pass |
| 4 | 1.2, 1.2.1, 1.2.2, 1.2.3 | User Selects Burglary from the dropdown | Crime Data Visualization and Table will display burglaries over the last 12 months, locations, and details of burglaries in last month | Crime Data Visualization and Table display burglaries over the last 12 months, locations, and details of burglaries in last month | Pass |
| 5 | 1.3, 1.3.1, 1.3.2, 1.3.3, 1.3.4 | User Selects Theft from the dropdown | Crime Data Visualization and Table will display thefts over the last 12 months, types of thefts, locations, and details of thefts in the last month | Crime Data Visualization and Table display thefts over the last 12 months, types of thefts, locations, and details of thefts in the last month | Pass |
| 6 | 1.3.5 | User Selects Year  from the dropdown | Crime Data Visualization and Table will display for selected year | Crime Data Visualization and Table display for selected year | Pass |
| 7 | 1.3.6 | User Selects Crime Type and Year from the dropdowns | Crime Data Visualization and Table will display for the selected Crime Type and Year | Crime Data Visualization and Table display for the selected Crime Type and Year | Pass |
| 8 | 2.0 | CrimeHawk will make a request to an External Data Source | The CrimeHawk Application will make a request to an External Data Source | The CrimeHawk Application makes a successful request to an External Data Source | Pass |
| 9 | 2.1 | CrimeHawk will make a request to the Open Baltimore API and receive data | CrimeHawk will make a request to a specific endpoint. After the request is made, CrimeHawk will receive data from the Open Baltimore API endpoint. | CrimeHawk makes a request to a specific endpoint. The request is made, CrimeHawk received data from the Open Baltimore API endpoint. | Pass |
| 10 | 3.0 | CrimeHawk will connect to Atlas MongoDB database server | CrimeHawk connects to the Atlas MongoDB database without producing an error. | CrimeHawk connects to the Atlas MongoDB database without producing an error. | Pass |
| 11 | 3.1 | CrimeHawk will take data from the Open Baltimore API and store it in the Atlas MongoDB database | CrimeHawk gets data from the Open Baltimore API and then stores it into the Atlas MongoDB database | CrimeHawk gets data from the Open Baltimore API and then stores it into the Atlas MongoDB database | Pass |
| 12 | 3.2 | CrimeHawk will receive homicide data from the Open Baltimore API and store it within the Atlas MongoDB Database | CrimeHawk gets homicide  data from the Open Baltimore API and then stores it into the Atlas MongoDB database | CrimeHawk gets homicide  data from the Open Baltimore API and then stores it into the Atlas MongoDB database | Pass |
| 13 | 3.3 | CrimeHawk will receive burglary data from the Open Baltimore API and store it within the Atlas MongoDB Database | CrimeHawk gets burglary  data from the Open Baltimore API and then stores it into the Atlas MongoDB database | CrimeHawk gets burglary  data from the Open Baltimore API and then stores it into the Atlas MongoDB database | Pass |
| 14 | 3.4 | CrimeHawk will receive theft data from the Open Baltimore API and store it within the DB | CrimeHawk gets theft  data from the Open Baltimore API and then stores it into the Atlas MongoDB database | CrimeHawk gets theft  data from the Open Baltimore API and then stores it into the Atlas MongoDB database | Pass |

# **User Guide**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Date** | **Description** | **Name** |
| 1.0 | 6/20/2020 | Initial Draft | Andrew Landis, Eske Gizaw, Lauren Rapp, Brent Phillips |
| 1.1 | 7/8/2020 | Introduction and Operation | Lauren Rapp |
| 1.2 | 7/8/2020 | Requirements/Installation/Screenshots | Brent Phillips |

## **Introduction**

Welcome to the CrimeHawk web application, your link to crime data for the city of Baltimore. Here we take a sample of crime data from the Open Baltimore Crime by Neighborhood victim-based crime data that is available to the public.

The User can make selections from two different drop-downs with three different crime types: burglary, homicide, and theft, and select a year between 2014-2020. CrimeHawk then displays three different crime data visuals based on the User selection. The first visual is a pie chart displaying information about how many crimes of the selected type occurred in each neighborhood. The second is a line chart displaying the number of selected crimes per month within the selected year. In the case of theft, a bar graph displaying the number of different types of larcenies, and in the case of homicide, a pie chart will display the weapons used. Lastly, a table displaying details for each crime of that type sorted by date, including crime code, description, location, and district. Data for 2020 is being updated and stored on a nightly basis, so the User can come back for updates.

## **Requirements**

CrimeHawk is designed to be used as a live web application. However, you can also view CrimeHawk locally on your machine without needing to be connected to the internet. The following are the requirements for the live and local instances:

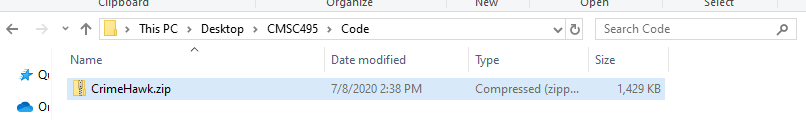
**Live Instance** - In order to view the latest live version of CrimeHawk you must be connected to the internet and be using a modern browser. For example, you can use the CrimeHawk application on your phone if you are connected to the internet, open your devices browser (Safari for Apple, Chrome for Android) and visit <https://crimehawk.herokuapp.com>. This will also work for any other device and modern browser. Older browsers like Internet Explorer version 10 and below are not fully supported. The Application may work but the creators of CrimeHawk prefer that it be used on modern browsers like Safari, Google Chrome and Mozilla Firefox.

**Local Instance** - Running CrimeHawk on your local computer requires a lot more work. First, you must have NodeJS installed on your local system. You can visit <http://nodejs.org> to get the latest version for your operating system. After you have installed NodeJS you will have to restart your machine. In addition to NodeJS, you must have a code text editor or integrated development environment. The CrimeHawk developer team prefers to use Microsoft Visual Studio Code which is cross platform. After you have these two things, you will have to have the source code. The source code can be obtained upon request from the CrimeHawk team.

## **Installation**

The live instance of CrimeHawk can viewed on our website <https://crimehawk.herokuapp.com>. The following installation instructions are for running the local instance of CrimeHawk.

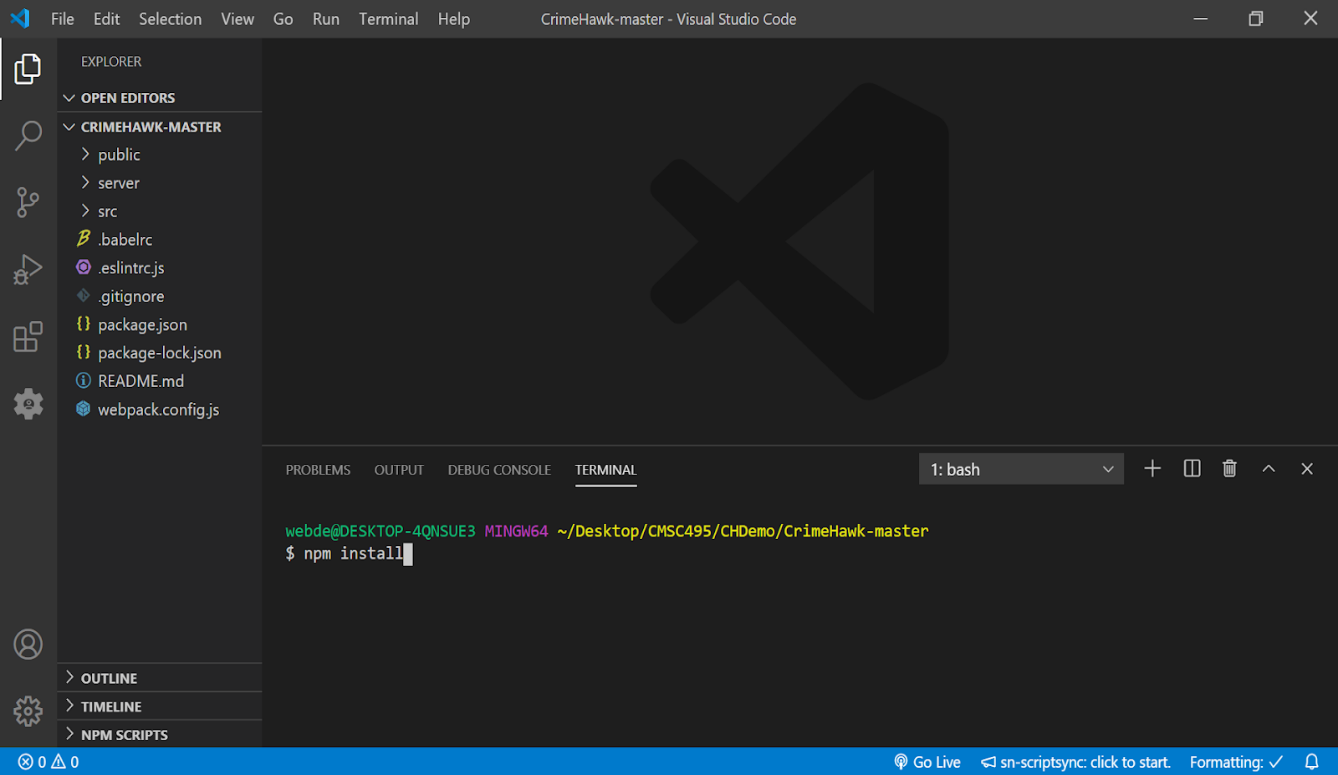
Unzip the CrimeHawk source code into a known directory.



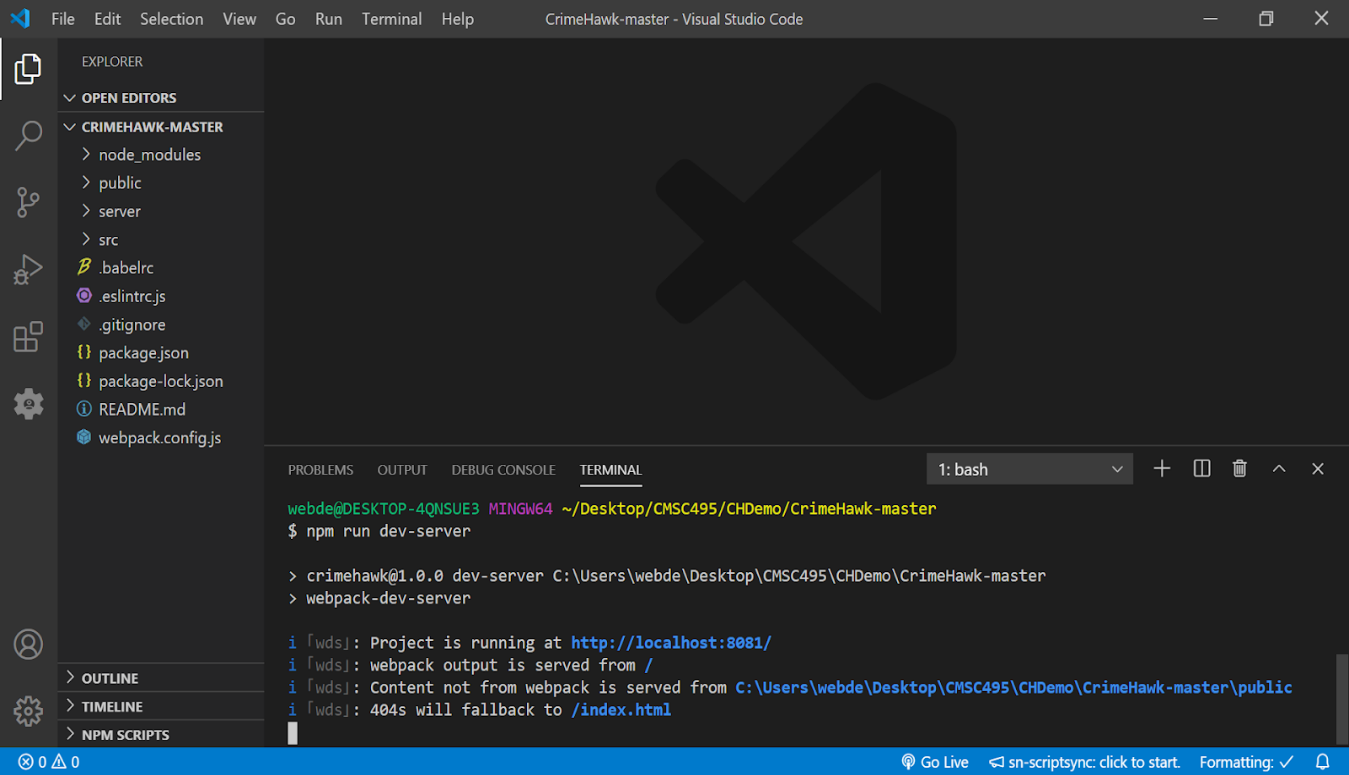
Check to make sure that you have NodeJS installed. Run the following command in a terminal on your machine.



Run the npm install command

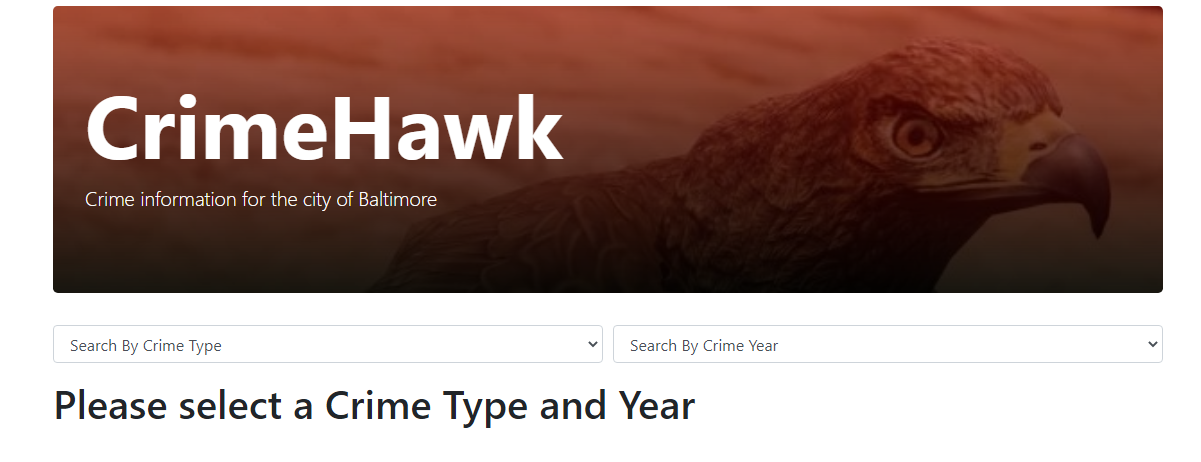


Run the npm run dev-server command



Now if you visit <http://localhost:8801> you will see the CrimeHawk Application

## **Operation**

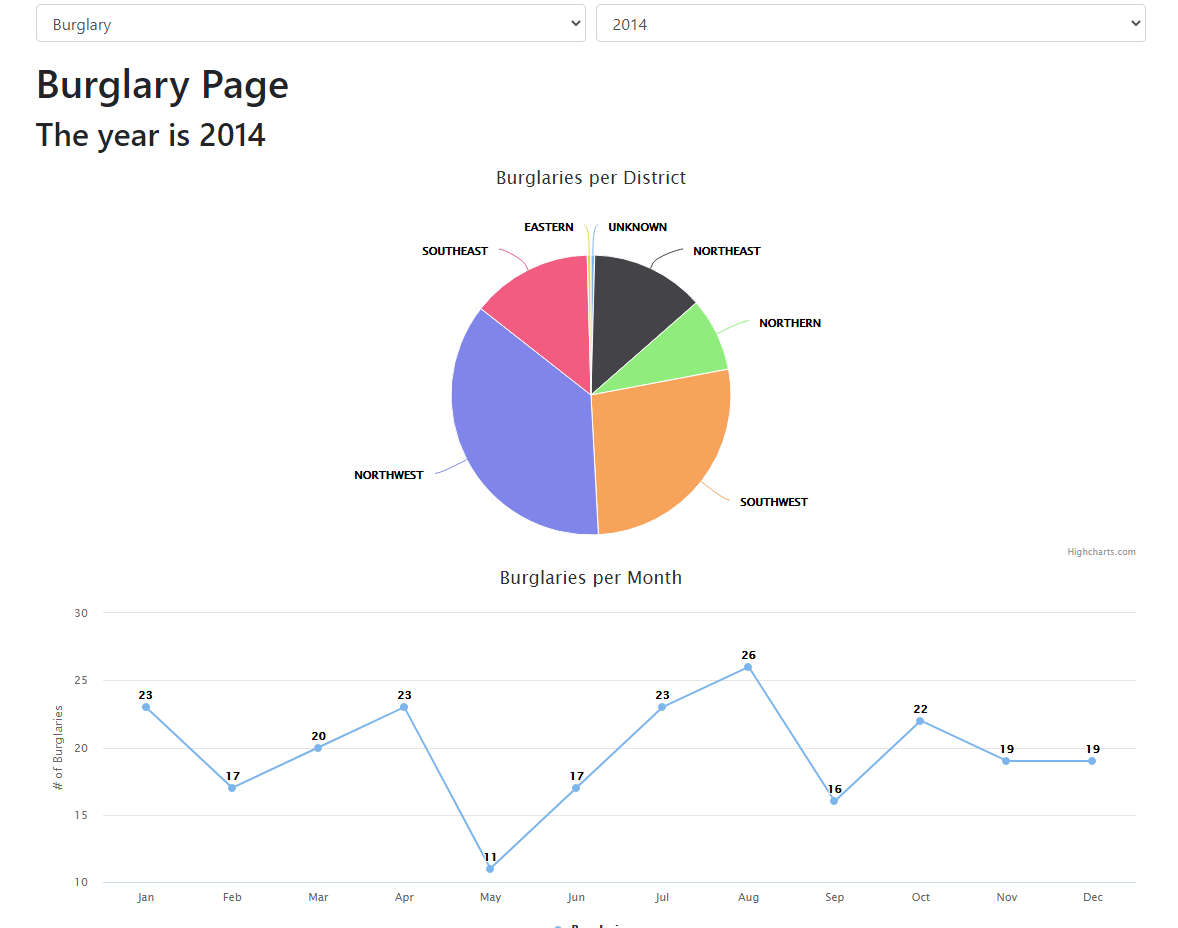


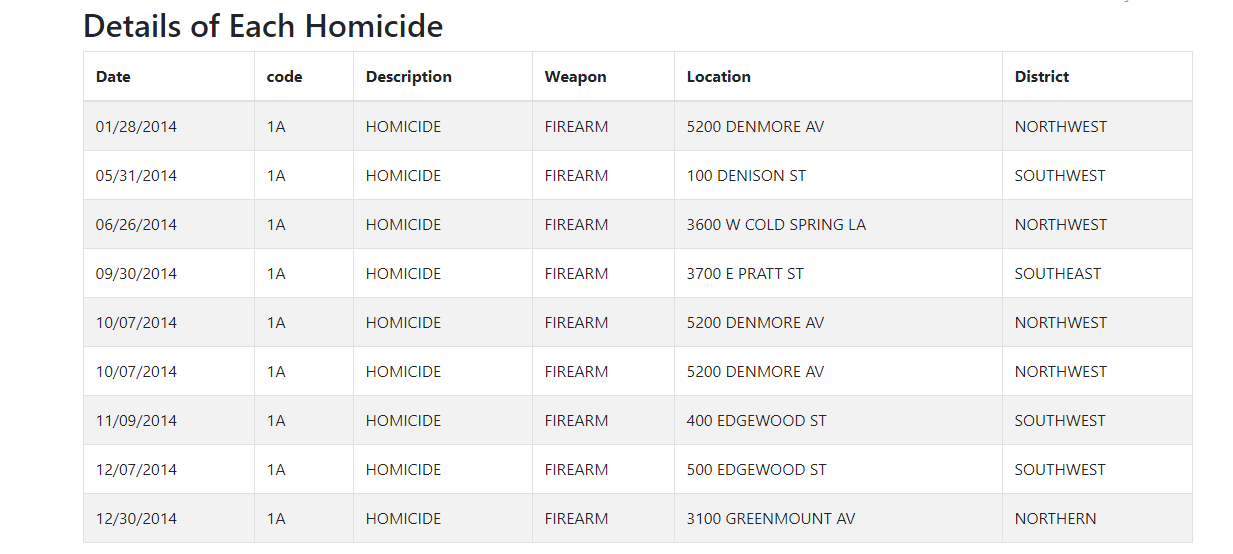
CrimeHawk is designed to be simple for the User to operate. Start by visiting the CrimeHawk web application where the User will select from the following drop-down options:

1)      Crime Type: Burglary, Homicide, or Theft

2)      Crime Year: 2014, 2015, 2016, 2017, 2018, 2019, or 2020

Once the User makes both selections, CrimeHawk will display results to the User in the following three crime data visuals:





1)      A pie chart displaying data number of crimes by neighborhood in selected crime type and year

2)      A line graph will be displayed with crimes by month for selected crime type and year. If the User selects theft, a bar graph displays broken down by the type of larceny. If the User selects homicide, a pie chart will display the type of weapon used.

3)      A Table with details of the selected crime type and year sorted by date, including crime code, description, location, and district.

If the User would like to change selections, simply make another selection from the drop-downs and the page will display new results without need to refresh.

Results will only be displayed if both crime type and crime year are selected.

When the User is finished exploring data with CrimeHawk, simply close the application.

The User can check back with CrimeHawk for up-to-date results for the current year, as the application collects data on a nightly basis.