CAB230 Web Computing Assignment

Getting Started with the API

Version of 11 April 2019, 11:00PM.

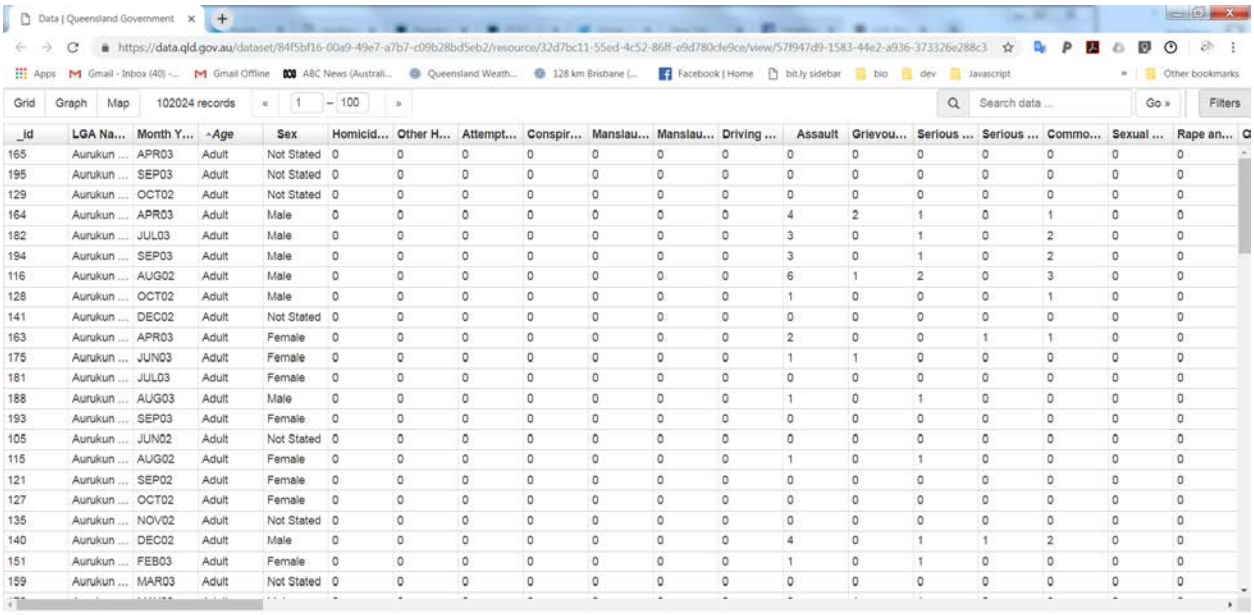
Note that the guide and the code have been substantially updated to reflect new filtering options and the switch to https. The final URL is: https://cab230.hackhouse.sh/.

This document presents a short guide to help you get started with the API you will be using over the course of the assignment. Initially, you will work with this site purely as a data source. In what follows we will work through the process of hitting this API using `fetch` based JavaScript calls as we did in the lecture and the prac for week 5.

As discussed in the lectures, we will be working with a dataset based on the following crime statistics made available by the state government:

https://data.qld.gov.au/dataset/lga_reported_offender_numbers/resource/32d7bc11-55ed-4c52-86ff-e9d780cfe9ce

As you can see if you select the visualisation tab, there are a massive number of columns listed, each reflecting an offence or range of offences covered by the criminal justice system.



_id	LGA Na...	Month Y...	Age	Sex	Homicid...	Other H...	Attempt...	Conspir...	Manslau...	Manslau...	Driving ...	Assault	Grievou...	Serious ...	Serious ...	Commo...	Sexual ...	Rape an...
165	Aurukun ...	APR03	Adult	Not Stated	0	0	0	0	0	0	0	0	0	0	0	0	0	0
195	Aurukun ...	SEP03	Adult	Not Stated	0	0	0	0	0	0	0	0	0	0	0	0	0	0
129	Aurukun ...	OCT02	Adult	Not Stated	0	0	0	0	0	0	0	0	0	0	0	0	0	0
164	Aurukun ...	APR03	Adult	Male	0	0	0	0	0	0	0	4	2	1	0	1	0	0
182	Aurukun ...	JUL03	Adult	Male	0	0	0	0	0	0	0	3	0	1	0	2	0	0
194	Aurukun ...	SEP03	Adult	Male	0	0	0	0	0	0	0	3	0	1	0	2	0	0
116	Aurukun ...	AUG02	Adult	Male	0	0	0	0	0	0	0	6	1	2	0	3	0	0
128	Aurukun ...	OCT02	Adult	Male	0	0	0	0	0	0	0	1	0	0	0	1	0	0
141	Aurukun ...	DEC02	Adult	Not Stated	0	0	0	0	0	0	0	0	0	0	0	0	0	0
163	Aurukun ...	APR03	Adult	Female	0	0	0	0	0	0	0	2	0	0	1	1	0	0
175	Aurukun ...	JUN03	Adult	Female	0	0	0	0	0	0	0	1	1	0	0	0	0	0
181	Aurukun ...	JUL03	Adult	Female	0	0	0	0	0	0	0	0	0	0	0	0	0	0
188	Aurukun ...	AUG03	Adult	Male	0	0	0	0	0	0	0	1	0	1	0	0	0	0
193	Aurukun ...	SEP03	Adult	Female	0	0	0	0	0	0	0	0	0	0	0	0	0	0
105	Aurukun ...	JUN02	Adult	Not Stated	0	0	0	0	0	0	0	0	0	0	0	0	0	0
115	Aurukun ...	AUG02	Adult	Female	0	0	0	0	0	0	0	1	0	1	0	0	0	0
121	Aurukun ...	SEP02	Adult	Female	0	0	0	0	0	0	0	0	0	0	0	0	0	0
127	Aurukun ...	OCT02	Adult	Female	0	0	0	0	0	0	0	0	0	0	0	0	0	0
135	Aurukun ...	NOV02	Adult	Not Stated	0	0	0	0	0	0	0	0	0	0	0	0	0	0
140	Aurukun ...	DEC02	Adult	Male	0	0	0	0	0	0	0	4	0	1	1	2	0	0
151	Aurukun ...	FEB03	Adult	Female	0	0	0	0	0	0	0	1	0	1	0	0	0	0
159	Aurukun ...	MAR03	Adult	Not Stated	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The site is quite sophisticated, and allows the user to sort and filter based on a range of data source selections and criteria. There is also the option to graph particular fields or to show the incidence of offences by geographical region on a map. If you want a grade of 6 or 7 standard mark for the client side of this assignment, then you will have to do the same. Start using this site to explore your ideas.

As you now know, Tylor took this CSV file and 'wrangled' the data into a form which is more suitable for a small, well-defined API. There are a range of endpoints available to you, and these are listed briefly, and without a lot of detail, in the bullet points below. The difference from the earlier version is that we now have available a range of other parameters to use as filters in the offences search.

It is easier now to group the endpoints into three categories:

Getting Started:

- `/register (POST) – Register using email and password`
- `/login (POST) – Logging into your account using the details provided. This allows access to authenticated routes via a JSON Web Token.`

Information (Open Endpoints):

- `/offences (GET) – returns an array of offences recorded. (Open endpoint).`
- `/areas (GET) – returns array of Local Government Areas. (Open endpoint).`
- `/ages (GET) – returns an array of age categories. (Open endpoint).`
- `/genders (GET) – returns an array of gender categories. (Open endpoint).`
- `/years (GET) – returns an array of years recorded. (Open endpoint).`

All of the information endpoints provide a list of legal values for the parameters of the search endpoint – usually just drop the 's' at the end of the word.

Search (Requires Authentication):

- `/search?offence=xxx (GET) – The primary search route. Note that query params need to be url encoded. (Requires authentication).`
- `/search?offence=xxx&param1=xxxx&param2=xxxx (GET) – The primary search route, but this time with optional filter parameters, which again must be url encoded. (Requires authentication).`
- `Params must be one of: area, age, gender, year, month`
- `Information routes are available for all params except month, which runs from 1,2,…,12`

Examples are provided below for `offences` and `offence`. Here we begin with the example of `Armed Robbery`, and this is used as the basis for subsequent filtering.

Information Endpoint: `/offences`
Search Parameter: `offence (required)`
Example values: `"Armed Robbery","Attempted Murder","Bomb Possess and/or use of"`
Example use: `https://cab230.hackhouse.sh/search?offence=Armed%20Robbery`

Note that this is the standard search – we have provided an offence (which is required) and we will see an array of records which cover all of the armed robbery offences in the database. We end up returning this (heavily edited) JSON:


```
{"query":{"offence":"Armed Robbery"},

    "result":[
      {"LGA":"Aurukun Shire Council","total":124},
      {"LGA":"Balonne Shire Council","total":13},
      {"LGA":"Banana Shire Council","total":5},
      {"LGA":"Barcaldine Regional Council","total":0},
      {"LGA":"Barcoo Shire Council","total":0},
      {"LGA":"Blackall Tambo Regional Council","total":0},
      {"LGA":"Boulia Shire Council","total":0},
      {"LGA":"Brisbane City Council","total":4049},

      …

      {"LGA":"Winton Shire Council","total":0},
      {"LGA":"Woorabinda Shire Council","total":10},
      {"LGA":"Wujal Wujal Shire Council","total":0},
      {"LGA":"Yarrabah Shire Council","total":6}
    ]
}
```


The role of the other (filter) parameters is then to limit these results based on some criteria. For example, an armed robbery query with filtering to the `Moreton Bay Regional Council` Area yields the following:


```
{"query":{"offence":"Armed Robbery","area":"Moreton Bay Regional
Council"},"result":[{"LGA":"Moreton Bay Regional Council","total":958}]}
```


In the code examples, I have considered a number of filter parameters: `area`, `age` and `year`. Note that the example for `year` shows the use of multiple values, with the query returning results for `2006`, `2007` and `2008`. Multiple values are comma separated, and the comma acts as a logical OR. The examples use only a single filter, but the API allows chaining of filters as in the examples on the root page at https://cab230.hackhouse.sh/.

These endpoints will be fixed for the remainder of the assignment. We will improve the documentation at the root and share this with you when we release the server side specification.

Some Code

The goal of this document is not to provide you with a comprehensive API reference, but to get you started in using the API to the point that you can register, and login and do some queries. The material relies heavily on the week 5 lecture and the week 5 prac. We are here to hit a public API with a series of POST and GET calls. We will be using the modern `fetch()` approach introduced in the lecture, and there will not be a single `XMLHttpRequest` in sight.

Our approach will be to build on the earlier prac work, defining buttons for each of the tasks we want to manage. The account registration should be executed only once for each email address. Please just use your QUT email and register once and once only.

In what follows we are going to use the `innerHTML` on the page only to show the good bits. For the error messages, you will need to open up the console in the page as you did in the debugging exercises in the prac.

The code is available in the `starter-files.zip` archive as usual. These are very basic calls and are intended solely to get you started. In particular, I am not doing any of the form processing here – for basic HTML you will find this at W3Schools. For React, please see the React Forms guide available near to this file on BB. The code provided here will be a good basis for the first stages of the assignment.

The HTML page is very simple. We begin with a cute heading at the top, we include two sets of buttons, and we then follow with a single result div which we call `app`. Each button

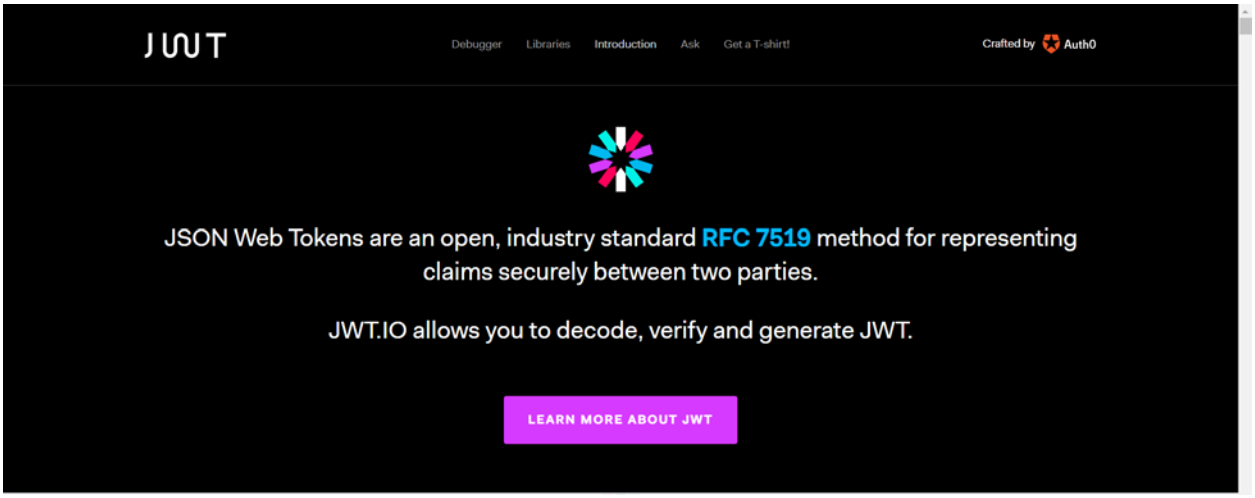
affects that div. We do not maintain the result of any previous calls. We will, however, need to keep track of a token, and that is something we will consider in a moment. The basic html looks like this:

```
<!DOCTYPE html>
<html>

<head>
   <title>API Starter Code</title>
   <link rel="stylesheet" href="./src/styles.css" />
   <meta charset="UTF-8" />
</head>

<body>
   <h1>Getting Started with Crime (Updated)</h1>
   <div id="menu">
      <button id="regBtn">Register</button>
      <button id="logBtn">Login</button>
      <button id="offBtn">Offences</button>
      <button id="serBtn">Armed Robbery</button>
   </div>

   <h2>Armed Robbery Offences - Filtered</h2>
   <div id="filter">
      <button id="area">area</button>
      <button id="age">age</button>
      <button id="year">year</button>
   </div>

   <h2>Query Responses</h2>
   <div id="app"></div>


   <script src="src/index.js">
   </script>
</body>
```

Registration and Login:

Registration is the first step in working with the API. If the database doesn't know you, then you can't perform a login to the system. If you can't login, then you can't acquire an access token. If you can't acquire an access token, then you can't really do all that much.

The API site here doesn't formally maintain a session. The way this works is that registered users may login to the site using their previously supplied credentials. On login, the system will return a JSON Web Token (see https://jwt.io/ for details):



This JWT provides you with authorisation to use the API. You do not need the JWT to access the `/offences` route as it is an open endpoint. However, if you want to access the interesting data using `/search`, then you need to provide this token in the request header. This is both dead simple and surprisingly tricky, in the sense that it is easy to make dumb errors that will cost you a lot of time. We will walk you through this below.

But first, we need to register. In practice, and in your assignment, we will expect that registration and login will be handled through a form. People will enter their details – we require an email and a password – and you will need to grab these data from the text boxes and then send them to the server using a POST request. The code below is essentially that POST request, only here we have provided the examples in their urelencoded form:

```
const regButton = document.getElementById("regBtn");
regButton.addEventListener("click", () => {

  fetch("https://cab230.hackhouse.sh/register", {
    method: "POST",
    body: 'email=woof%40dog.com&password=WalksAndFoodForever',
    headers: {
      "Content-type": "application/x-www-form-urlencoded"
    }
    })
```

```
    .then(function(response) {
      if (response.ok) {
        return response.json();
      }
      throw new Error("Network response was not ok");
    })
    .then(function(result) {
      let appDiv = document.getElementById("app");
      appDiv.innerHTML = JSON.stringify(result);
      regButton.disabled = true;
    })
    .catch(function(error) {
      console.log("Problem with fetch ",error.message);
    });
});
```

Much of this code is familiar to you, but some is not. The fetch request is a POST rather than a GET and so we must include a second argument to the function containing the parameters of the request. So the brace after the URL is the opening brace of a JSON object which contains the `method`, the `body`, and the `headers` for the request. The headers object here contains a single content type header which indicates that the body contains urlencoded data – this is typical of text grabbed from a form.

You need to focus on the body and directly edit it to show the email address and password you want to use. The `%40` is the '@' character from the email address. In the file I have supplied for your use, the email address and password are edited down to placeholder characters. You will need to fix them up for yourselves.

```
fetch("https://cab230.hackhouse.sh/register", {
  method: "POST",
  body: 'email=woof%40dog.com&password=WalksAndFoodForever',
  headers: {
    "Content-type": "application/x-www-form-urlencoded"
  }
})
```

The remaining code is boilerplate and very similar to that which you saw in the lectures. Open the index.js file in your editor and update the email and password to the values you want to use. Scroll down in the file to the next call, which hits the `/login` route. Again update the body, making sure that the email and password here are exactly the same as those you use in the registration request. Save the file and reload the html page in the browser. You should see something like the following (this is the old page image):

When you hit the registration button, you will get a simple stringified JSON response telling you that the registration has been successful. The Register button is then disabled, at least until the next time you reload the page.

If you do not get a success message, there will be something wrong with the email address or password that you have used. You can find out more by inspecting the page, opening up the console, and interrogating the response object.

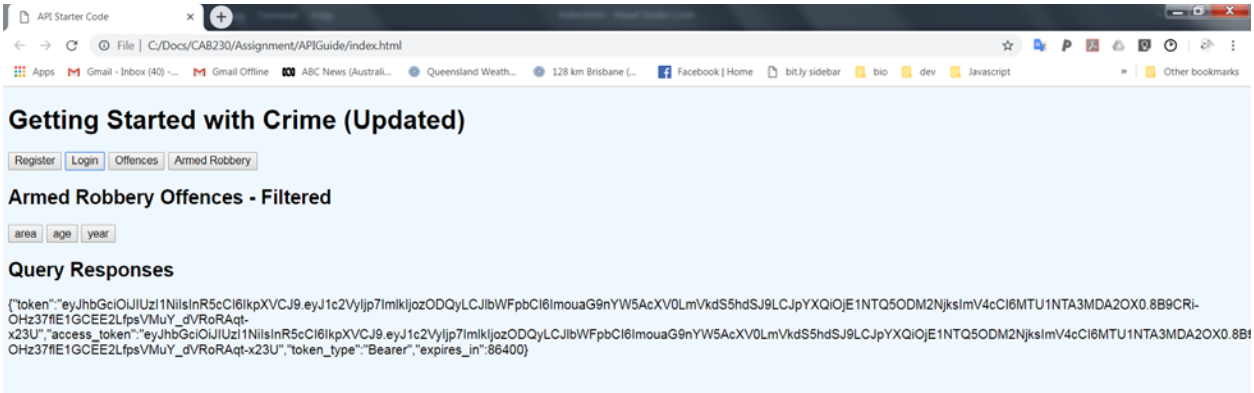
In the normal course of events, we now login. This doesn't happen on registration and it is a separate task. As you would expect, you must use the credentials you supplied during registration, and we did this on the previous page. The call is again a POST, and it is very similar in shape to the registration request. The details are on the following page. Note initially the declaration at the top of the file. JWT will hold the token when we get it back from the login.

```
let JWT = null;

//regButton code goes here – removed for space

const logButton = document.getElementById("logBtn");
logButton.addEventListener("click", () => {
    fetch("https://cab230.hackhouse.sh/login", {
      method: "POST",
      body: 'email=woof%40dog.com&password=WalksAndFoodForever',
      headers: {
            "Content-type": "application/x-www-form-urlencoded"
      }
    })
      .then(function(response) {
          if (response.ok) {
              return response.json();
          }
```

```
            throw new Error("Network response was not ok.");
        })
        .then(function(result) {
            let appDiv = document.getElementById("app");
            appDiv.innerHTML = JSON.stringify(result);
            JWT = result.token;
        })
        .catch(function(error) {
            console.log("Problem with fetch ",error.message);
        });
});
```

The parameters to the request are almost identical. The only difference is that we are hitting a different endpoint.

In the second `then` clause you will see the usual AJAX update, but here we also assign the token to the `JWT` variable. Click on the login button and you should see the image below. The assignment statement accesses the token field from the JSON and stores the token for future use. The token is valid for 24 hours, after which you need to login again.



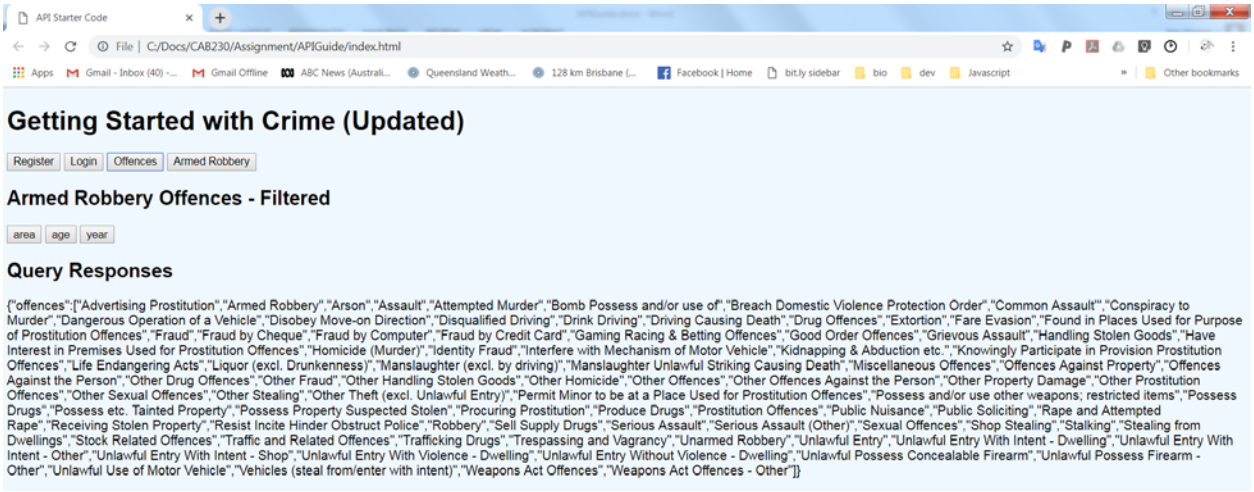
Getting Data:

We now grab some data, which is after all why we are here. We will begin with a simple GET request with no authentication to one of the information endpoints. The others are very, very similar, and you can work them out from this example.

```
const offButton = document.getElementById("offBtn");
offButton.addEventListener("click", () => {
    fetch("https://cab230.hackhouse.sh/offences")
        .then(function(response) {
            if (response.ok) {
                return response.json();
            }
            throw new Error("Network response was not ok.");
        })
```

```
        .then(function(result) {
            let appDiv = document.getElementById("app");
            appDiv.innerHTML = JSON.stringify(result);
        })
        .catch(function(error) {
            console.log("There has been a problem with your
fetch operation: ",error.message);
        });
});
```

This is exactly the same as so many of the examples you have seen and requires little explanation. It is a simple GET hitting the `/offences` endpoint, which provides a list of the offence categories used in the dataset. Click on the Offences button and you will see:



The final tasks are much more complicated. We are going to hit the `/search` endpoint, and then we are going to return and hit it again with filter parameters. The basic structure of the two sections is very similar, but when filtering is involved we need to manage additional parameters, and in our case, additional buttons.

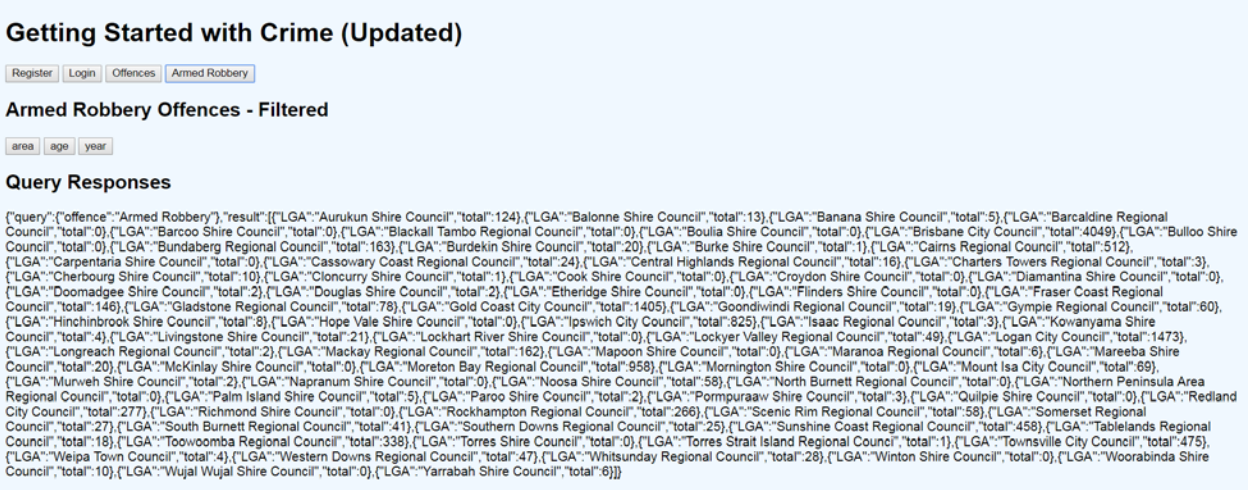
```
const searchButton = document.getElementById("serBtn");
searchButton.addEventListener("click", () => {

    //The parameters of the call
    let getParam = { method: "GET" };
    let head = { Authorization: `Bearer ${JWT}` };
    getParam.headers = head;

    //The URL
    const baseUrl = "https://cab230.hackhouse.sh/search?";
    const query = 'offence=Armed Robbery';
    const url = baseUrl + query;
```

```
    fetch(encodeURI(url),getParam)
        .then(function(response) {
            if (response.ok) {
                return response.json();
            }
            throw new Error("Network response was not ok.");
        })
        .then(function(result) {
            let appDiv = document.getElementById("app");
            appDiv.innerHTML = JSON.stringify(result);
        })
        .catch(function(error) {
             console.log("Problem with fetch: ",error.message);
        });
});
```

The key lies in the construction of the request parameters and the URL for the query request. Note that the original version of this document requested stats on murder. Here we have changed to armed robbery. The request parameters show that the request is a GET, but most importantly here they provide the mechanism for authentication. We create an `Authorization` header, which follows the usual: `Authorization: <type> <credentials>` format. Here we use a `Bearer`, with the token providing the credentials. This is accomplished using the lines below, where the second assignment adds this JSON object as a field on the `getParam` object:

```
let head = { Authorization: `Bearer ${JWT}` };
getParam.headers = head;
```

We build the URL and include the query string, yielding the results shown below:



Here we use string concatenation and then call the `encodeURI` method to ensure that the parameters are urlencoded. We then click on the search button and the event handler hits the authenticated API endpoint.

Our final task is to deal with a filtered version of the search. None of this is difficult, and the code is a very straightforward extension of what we have just seen. In this case, I have created three filter buttons, each of which has an associated label and example query. The selection is made simply as follows:

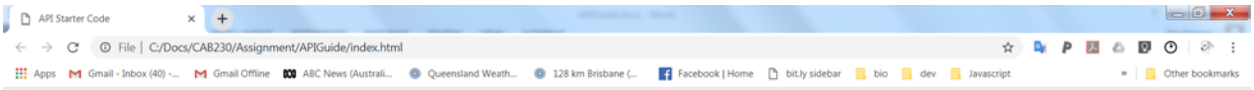
```
const filterDiv= document.getElementById("filter");
filterDiv.addEventListener("click", (event) => {
    const param = event.target.innerHTML;
    let filter = "";

    //Example filter strings
    if (param === "area") {
        filter = "area=Moreton Bay Regional Council";
    } else if (param === "age") {
        filter = "age=Juvenile"
    } else if (param === "year") {
        filter = "year=2006,2007,2008";
    }
```

Here we use the trick of attaching the event handler to the containing filter div, and then using the target property of the event to find the source button. The variable filter is then set to the example filter string. This is then combined with the mandatory `offence` parameter in the query string:

```
//The URL
const baseUrl = "https://cab230.hackhouse.sh/search?";
const query = 'offence=Armed Robbery';

const url = baseUrl + query + "&" + filter;
```

The remaining code is the same as for the earlier search example. We have seen the result for `area` above – on the next page you will see results for juvenile armed robbery offences. Note that the code is very straightforward and you can modify these parameters to explore a range of alternatives.

And so we are done…



Getting Started with Crime (Updated)

Register | Login | Offences | Armed Robbery

Armed Robbery Offences - Filtered

area | age | year

Query Responses

{"query":{"offence":"Armed Robbery","age":"Juvenile"},"result":[{"LGA":"Aurukun Shire Council","total":68},{"LGA":"Balonne Shire Council","total":3},{"LGA":"Banana Shire Council","total":2},{"LGA":"Barcaldine Regional Council","total":0},{"LGA":"Barcoo Shire Council","total":0},{"LGA":"Blackall Tambo Regional Council","total":0},{"LGA":"Boulia Shire Council","total":0},{"LGA":"Brisbane City Council","total":1112},{"LGA":"Bulloo Shire Council","total":0},{"LGA":"Bundaberg Regional Council","total":46},{"LGA":"Burdekin Shire Council","total":10},{"LGA":"Burke Shire Council","total":1},{"LGA":"Cairns Regional Council","total":220},{"LGA":"Carpentaria Shire Council","total":0},{"LGA":"Cassowary Coast Regional Council","total":2},{"LGA":"Central Highlands Regional Council","total":4},{"LGA":"Charters Towers Regional Council","total":1},{"LGA":"Cherbourg Shire Council","total":7},{"LGA":"Cloncurry Shire Council","total":1},{"LGA":"Cook Shire Council","total":0},{"LGA":"Croydon Shire Council","total":0},{"LGA":"Diamantina Shire Council","total":0},{"LGA":"Doomadgee Shire Council","total":0},{"LGA":"Douglas Shire Council","total":1},{"LGA":"Etheridge Shire Council","total":0},{"LGA":"Flinders Shire Council","total":0},{"LGA":"Fraser Coast Regional Council","total":47},{"LGA":"Gladstone Regional Council","total":14},{"LGA":"Gold Coast City Council","total":349},{"LGA":"Goondiwindi Regional Council","total":3},{"LGA":"Gympie Regional Council","total":15},{"LGA":"Hinchinbrook Shire Council","total":2},{"LGA":"Hope Vale Shire Council","total":0},{"LGA":"Ipswich City Council","total":238},{"LGA":"Isaac Regional Council","total":1},{"LGA":"Kowanyama Shire Council","total":1},{"LGA":"Livingstone Shire Council","total":3},{"LGA":"Lockhart River Shire Council","total":0},{"LGA":"Lockyer Valley Regional Council","total":11},{"LGA":"Logan City Council","total":527},{"LGA":"Longreach Regional Council","total":0},{"LGA":"Mackay Regional Council","total":46},{"LGA":"Mapoon Shire Council","total":0},{"LGA":"Maranoa Regional Council","total":3},{"LGA":"Mareeba Shire Council","total":13},{"LGA":"McKinlay Shire Council","total":0},{"LGA":"Moreton Bay Regional Council","total":308},{"LGA":"Mornington Shire Council","total":0},{"LGA":"Mount Isa City Council","total":30},{"LGA":"Murweh Shire Council","total":2},{"LGA":"Napranum Shire Council","total":0},{"LGA":"Noosa Shire Council","total":12},{"LGA":"North Burnett Regional Council","total":0},{"LGA":"Northern Peninsula Area Regional Council","total":0},{"LGA":"Palm Island Shire Council","total":5},{"LGA":"Paroo Shire Council","total":1},{"LGA":"Pormpuraaw Shire Council","total":0},{"LGA":"Quilpie Shire Council","total":0},{"LGA":"Redland City Council","total":77},{"LGA":"Richmond Shire Council","total":0},{"LGA":"Rockhampton Regional Council","total":66},{"LGA":"Scenic Rim Regional Council","total":16},{"LGA":"Somerset Regional Council","total":4},{"LGA":"South Burnett Regional Council","total":9},{"LGA":"Southern Downs Regional Council","total":6},{"LGA":"Sunshine Coast Regional Council","total":116},{"LGA":"Tablelands Regional Council","total":5},{"LGA":"Toowoomba Regional Council","total":130},{"LGA":"Torres Shire Council","total":0},{"LGA":"Torres Strait Island Regional Counci","total":0},{"LGA":"Townsville City Council","total":169},{"LGA":"Weipa Town Council","total":4},{"LGA":"Western Downs Regional Council","total":16},{"LGA":"Whitsunday Regional Council","total":8},{"LGA":"Winton Shire Council","total":0},{"LGA":"Woorabinda Shire Council","total":8},{"LGA":"Wujal Wujal Shire Council","total":0},{"LGA":"Yarrabah Shire Council","total":4}]}