Web Component JSON Response using AppInventor

App Inventor has a component called Web which gives you the functionality to send and fetch data from a server or a website through GET and POST requests. This component can decode both JSON and HTML formatted data. We will be writing an app called iRead that will ask a user to type in a full or partial book name and query Google’s database of books using Books API and retrieve the most relevant book info. We will parse the Title and Author of the book, we will get the image URL of the book cover, and finally the book URL which can be launched through a browser. This is how our app would look like-

Before we start, I advise you to read on anything that is unfamiliar to you but mentioned in the paragraph above like Books API, JSON, HTML, GET, POST, etc.
First thing first, build your user interface (UI) in the Designer window. Mine looks like this-

As you can see I renamed default names of components App Inventor provides. It’s upto you what you want to name the components you use. If you are having hard time designing the UI, download the source file iRead and take a look. What matters is that you have all the essential components as shown in the UI screenshot above.

Have you tried executing a Book API query request yet? Click on the link below to see the API response.
https://www.googleapis.com/books/v1/volumes?q=Harry&maxResults=1&projection=lite
This gives you a response that looks similar to this-
Our concern is not the actual book info (Over time you might not get the same book info using the same query), our concerns are the tokens that we’ll need to identify and parse specific data. Take a look at the tokens like “title”: that ends with a comma and a new line, “authors”: that ends with ], etc.

Now let’s start defining actions and interactions in the blocks editor.
We have **three** global variables defined:

- **googleBookSearchURL** – That’s where we store the API URL address for book search.
- **tempData** – We will use that for saving temporary data.
- **bookURL** – That’s where we will parse and store the URL (“infoLink” token) of the book information returned by the API.

When a user enters a book name and hits the Search button, we call **GetBookInfo** procedure where we simply construct our URL with user inputted text and **encode** it for using it as a URL, and finally feed it to **Web component’s URL property**. We want to get only one book info (API may return many); so we used **maxResults=1**. If we didn’t use **projection=lite**, we would get a lot of other info that we are not going to use in this tutorial app. At the end we call **Web.get** to execute our query.

How do we know when the server returns a response to our query? Well, we use **GotText** block of Web component for this. Whenever our app receives something, GotText block will automatically wake up for us to validate and process the data. Let’s **parse the JSON** response from the API that we receive.
We only can proceed if `responseCode` is 200 which means our query didn’t fail along the way. We have another procedure called `ParseBookResult` that actually parses the response. Let’s take a look at it-
In the **ParseBookResult** procedure, we split the response content using the **start tag** and **end tag**. For an example, to parse title of the book, we need to know the beginning and the ending tags of a title. If you have noticed in the query, it returns-

"title": "Harry Potter and the chamber of secrets",
"authors": [
 "J. K. Rowling"
],

So we know, it begins with “**title**”: and ends with a comma followed by a new line. That is why we used those tags when we called ParseBookResult from GotText procedure above. In our example raw data, the value is “Harry potter and the chamber of secrets”. Then we decode the JSON value by calling Web’s **JsonTextDecode**. And finally decode the **HTML characters** by calling **HtmlTextDecode**. We call HtmlTextDecode to replace unreadable HTML characters with readable characters. For instance, &amp; is actually & and we want to show & instead of &amp;.

When a user taps on the Read button, we simply launch the browser with the book URL.
For launching a browser, we use `ActivityStarter` component. We set the action first. You can see a list of actions in this page. We also set the URL we retrieved. Then we just call the `StartActivity` of `ActivityStarter` component to launch the browser with the specified URL. If our `bookURL` doesn’t contain a valid URL, we simply inform the users. 

Download the source file of this app iRead.
Parsing Multiple Results

What if you want to parse multiple results? Well, in that case you define a higher `maxResults` when you query like this-

As you can see in the screenshot above, we defined `maxResults` to be three meaning we want three results. If Google cannot find three matches for our book query, in that case we wouldn’t be receiving three results obviously. But if it does find three or more matches for our query, we will receive three.

For demonstration purpose, we will only parse the titles. Maybe you already guessed, we would need for each block to do the job. If you are not familiar with `for each`, read this tutorial first.
We created a procedure called `GetAllTitles` that has three arguments - `apiResponse`, `start`, and `end`. We call this procedure with data like this-

As you can see, we passed the `responseContent`, `start` and `end` tags like we did before for parsing a single title. `GetAllTitles` uses a `for each` loop and go through all the titles and do the same for parsing, exactly the way we did before. Before we didn’t have the `for each` loop as we had only one title to parse and now we have a loop because we have many to parse. Note that `GetAllTitles` procedure `stores` the `result` after parsing each title to a global variable named `allTitles` and `returns` it.

We used a Label component named `OtherTitlesLabel` to display the result that is returned by the `GetAllTitles` procedure. Our app now looks like this-
We created two apps. iRead for single result and iRead2 for multiple. You can download both apps – iRead and iRead2.