TCS — Blockchain Tutorial 3

# Build a Bitcoin Currency Exchange with API Endpoints

Ok, so far in the Blockchain Tutorial 1 you set up a simple MVC framework with which to implement your blockchain explorer code. In Blockchain Tutorial 2 you explored more deeply the concept of the blockchain by writing some JavaScript code, and you may have in Part 3 of this tutorial converted this JavaScript to a PHP implementation that integrates with your MVC framework.

In this Blockchain Tutorial Part 3 we are going to investigate public API endpoints to explore bitcoin currency exchanges. But, rather than go through a long-winded tutorial on how to develop a front end with Bootstrap, I will give you the front end into which you can begin to integrate your results for all parts of this tutorial series, Figure 1

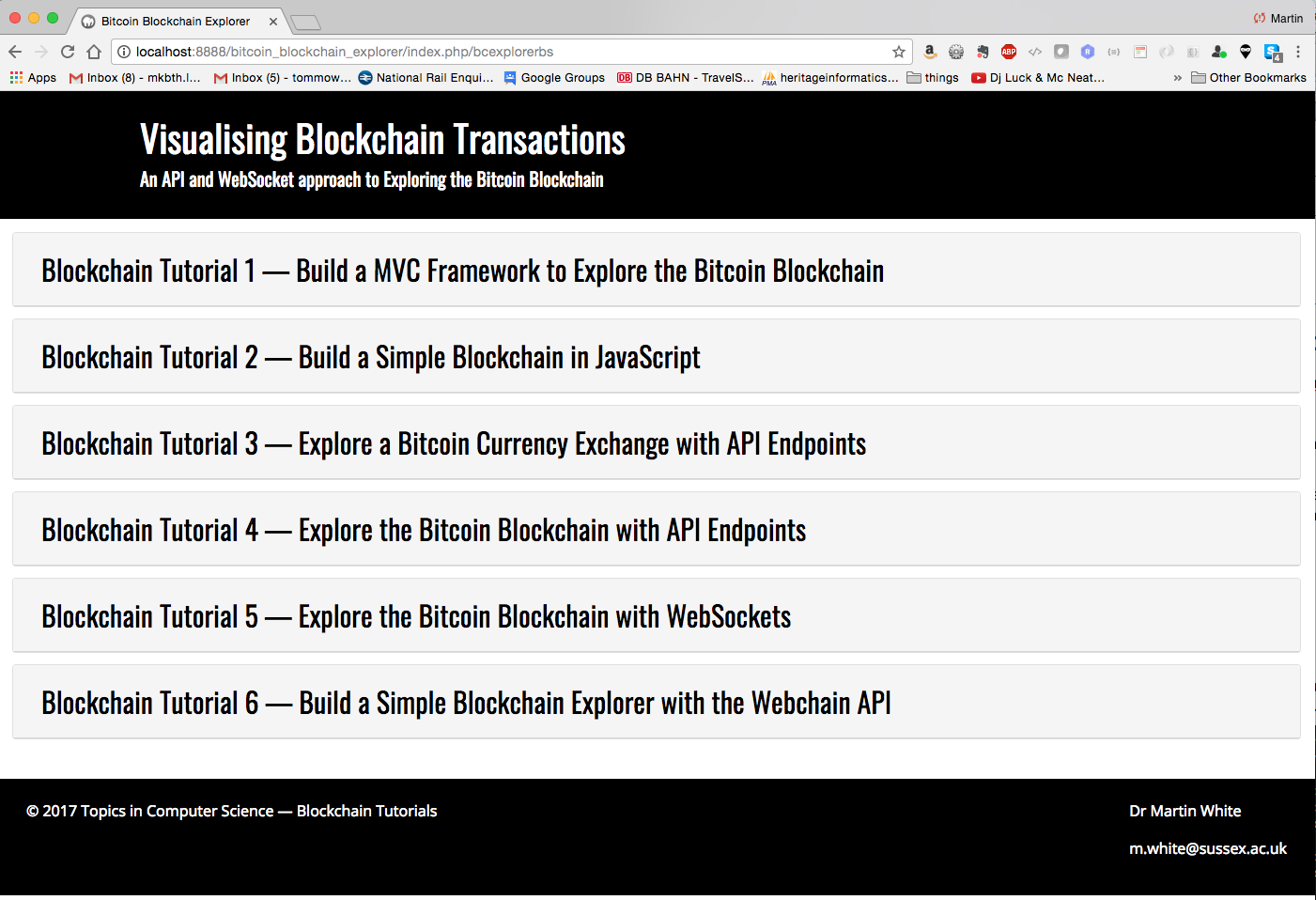


Figure 1: A Bootstrap front end for the Blockchain Tutorial Series …

Now, you have already completed Blockchain Tutorial 1, where we gave you another view (Dreamweaver) to visualise the data in the SQLite table, so all the Bootstrap view above is link to that view and back again. I am using the term visualise here rather loosely, but when, and if you get to the latter tutorials you should see that it would be a logical step to feed real-time or historical blockchain data into a graph function, for example.

To see your result for the Blockchain Tutorial 1 simply link the button BITCOIN\_DW, Figure 2, to the view shown in Figure 3. You can link up the navigation bar in Figure 3 at the same time.

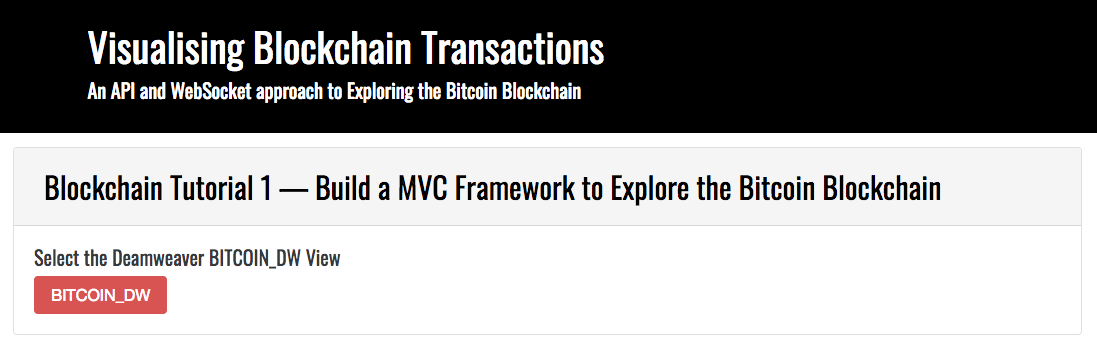


Figure 2: Bootstrap front end to link to Blockchain Tutorial 1 result

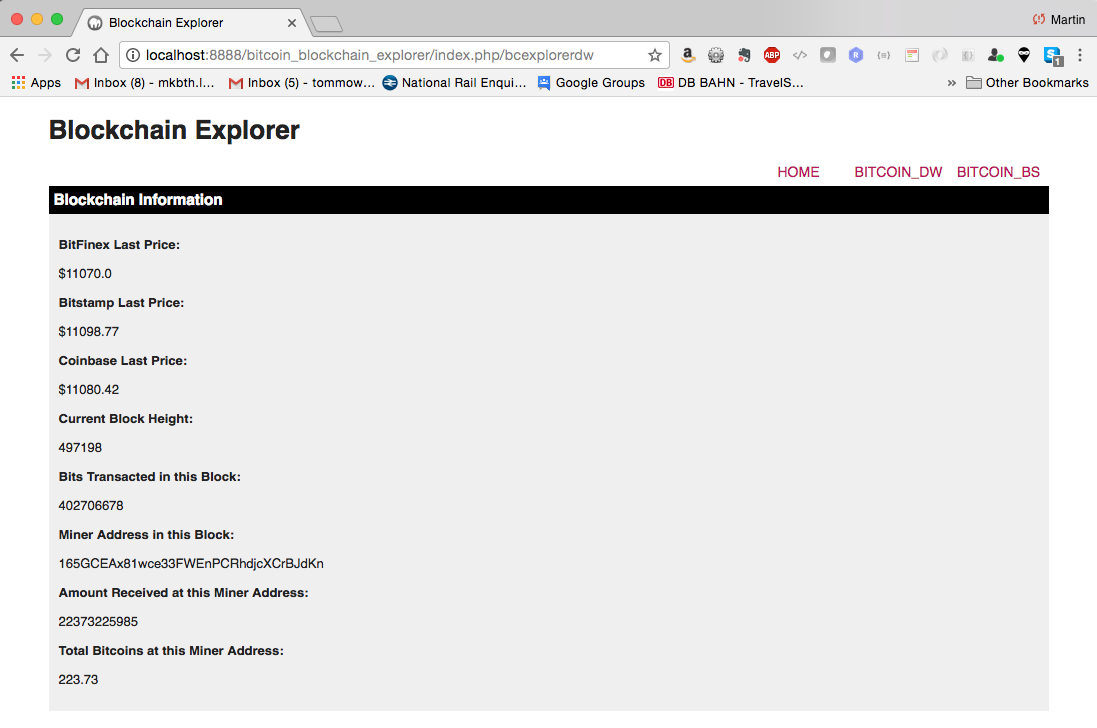


Figure 3: Blockchain Tutorial 1 result

Similarly, if you look at the Blockchain Tutorial 2 view in Figure 4, you can see it is empty because up to Part 2 of this tutorial we were looking at the JavaScript output with the console.log() function. You may recall that we suggested you convert your Part 2 JavaScript blockchain into a PHP blockchain and build a simple, very simple, wallet and simulation front end to play with it — our front end would go here.

So, before we move onto the Blockchain Tutorial 3 — Explore a Bitcoin Exchange with API Endpoints, lets make sure we have this new front end implemented in your MVC framework.

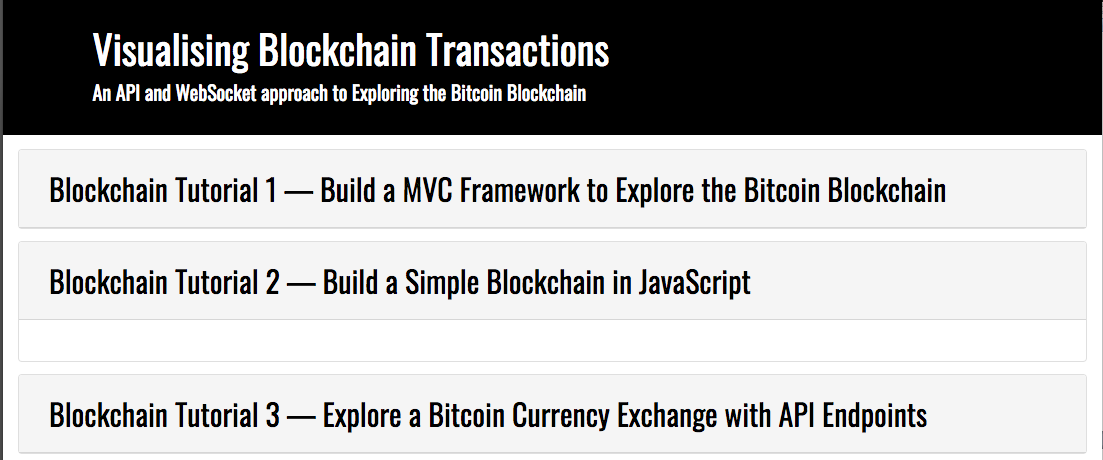


Figure 4: The view for Blockchain Tutorial 2 is empty waiting for you to use for Part 3

## Part 1 — Implement the Bootstrap Front End

You should already have downloaded the code for all these Blockchain Tutorials 1 to 6 for GitHub so that you can cut and paste elements as you go along. I am currently using Visual Studio Code as my development editor; see Figure 5, which shows the bcExplorerBs.php view highlighted. I will list all the required files needed this tutorial and highlight those you have already created, those you need to copy and those that you will update in this tutorial.

* **MVC** folder — This is your model, view, controller folder that you have already created in the Blockchain Tutorial 1. At the same level as this folder you should have the mvc.php file.
* **Controller** folder — This contains your controller.php file with your PHP controller class and associated methods.
* **Model** folder — This folder contains your model.php file with its Model class and associated methods. You may have updated these if you completed Part 3 of the Blockchain Tutorial 2. There are some other files here too: data.json, you will need to copy this from the GitHub download. This is a simple JSON file to allow dynamic update of information at the frontend view — if you look at it, it will be obvious what it does. You can ignore the bitcoin\_model.php file — that was just some procedural PHP code that I wrote to test the blockchain data access with API endpoints. You will use the same API endpoints, but we will use object oriented PHP later in this Blockchain Tutorial 3. Finally, there is also a db folder that contains your SQLite database. — again, you may have already updated this if you did the optional Part 3 of the Blockchain Tutorial 2.
* **Vendor** folder — ignore this folder, I originally installed composer as a PHP package manager, we are not using this at the moment. Associated with this is the composer.lock and composer.json files, again ignore.
* **View** folder —In here we find the bcExplorerBs.php view, which we just saw in Figure 1. You need to copy this file from the GitHub download. This file is already created for the complete set of tutorials, because we will be creating or updating new controller methods and new model methods using PHP and API endpoints in this Blockchain Tutorial 3, rather than focusing on HTML5 and CSS3 development. Where necessary I will refer to elements of this view to illustrate. Also, you already have the bcExplorerDw.php view, and the load.php, and the view\_simple\_message.php views. I can’t recall if you already have the home.php view, if not copy that over too.

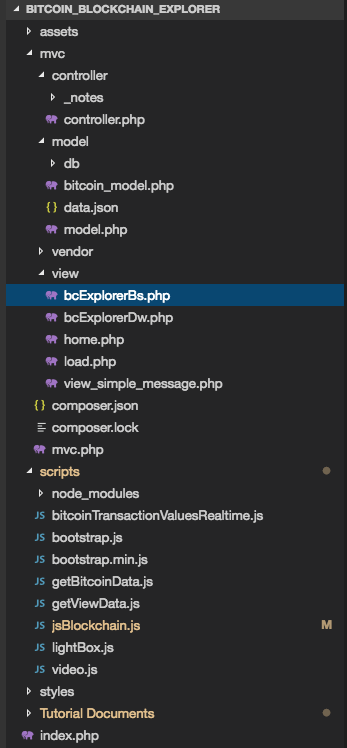


Figure 5: MVC framework structure showing the bcExplorerBs.php view highlighted

* **Scripts** folder — This folder contains all your JavaScript files and the node\_modules folder that you installed in the Blockchain Tutorial 2, when you used the JavaScript SHA256 hash function for creating the current block hash, previous block hash and mining a block. The rest of the files are JavaScript files, copy over the getViewData.js file, you can use this as it is, because we won’t have time to build it. Have a look at it anyway; it simply uses JQuery’s .getJSON() function to dynamically get view information from the data.json file in the Model folder to save cluttering up the view with hardwired text. The lightBox.js and video.js you can copy over, we might be using those later — you should note, that the iews I am giving you are based on views I created for other projects (code reuse, etc.). Obviously you will need the bootstrap JavaScript files, copy those over. You already created the jsBlockchain.js file in the last Blockchain Tutorial 2 — this is your simple JavaScript blockchain that you created to reinforce your knowledge around this subject. That leaves us one more JavaScript file for us to build as part of this Blockchain Tutorial 3, i.e. the getBitcoinData.js file. Finally, there is also a bitcoinTransactionsValuesRealtime.js file, which you can ignoire for now. We will create that in the Blockchain Tutorial 5 where we explore wWebSockets to access real-time bitcoin data.
* **Styles** folder — This folder contains all the CSS files needed to style the views. Just copy over this complete folder. I think you may already have it.

Ok, that is most, if not all, of the stuff needed for this Blockchain Tutorial 3 — Explore a Bitcoin Exchange with API Endpoints.

## Part 2 —Create PHP Model Methods with the API Endpoints

Let’s start by adding some more PHP methods to the Model class to access bitcoin exchange rate data form various bitcoin exchanges.

1. Open your model.php file in your mvc/model folder. You should have already created your Model class with:
   * The constructor that connects to your SQLite database,
   * A public function (i.e. method) dbCreateBitcoi Data() where you created the database table and inserted some test bitcoin data. Interestingly, at the time of writing this code, early November I think, bitcoin was worth around $5,665 and early today, 4 Dec 2017 it is worth $11,448 on the same Bitfinex exchange. So, if I had bought one bitcoin then and sold it now I would be about $6,000 better off.
   * A public function (i.e. method) dbReadBitcoinData() to get the bitcoin data from the SQLite database and return it to the controller class method dbReadBitcoinData().
   * A public function (i.e. method) dbDelete() to drop the bitcoin data table.
   * So, check you still have this PHP code.
2. In the GitHub downloaded code you will see that there is another method called apiReadBitcoinData(). A quick scan of this shows you that it uses a selection of API endpoints to read the bitcoin data values from the three exchanges (Bitfinex, Bitstamp, Coinbase) as well as using the blockchain.info API endpoits to get values back from the bitcoin blockchain. We will use this PHP method as a basis on which to create some new methods, i.e. break this method up to explore the APIs further starting with the exchange APIs. Using the apiReadBitcoinData() method as a guide create a new method to read just the three exchange bitcoin values from Bitfinex, Bitstamp and Coinbase.

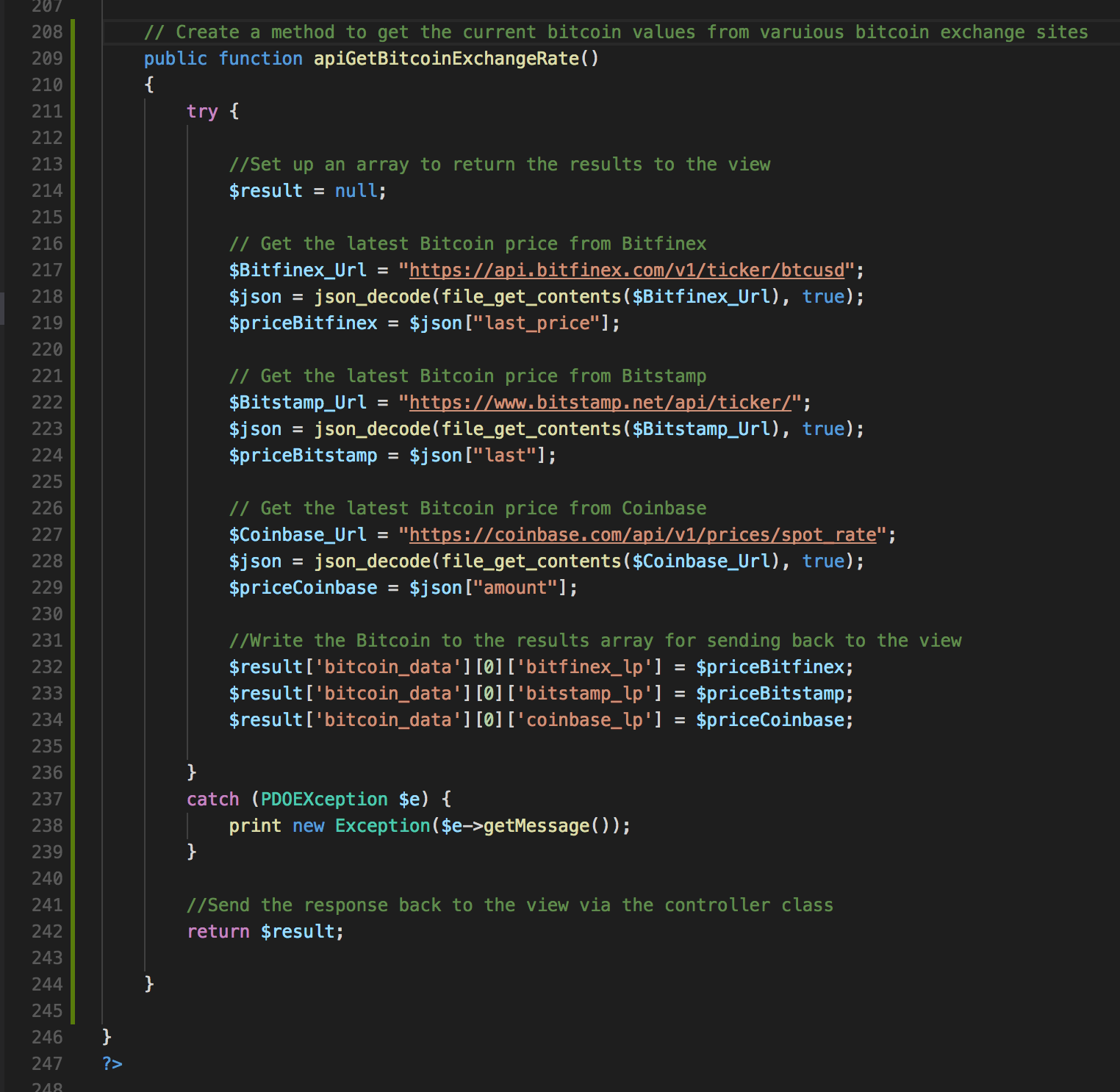


Figure 6: PHP method to get the latest bitcoin dollar rate for three separate bitcoin exchanges

* + Code explanation
    - Line 209 – 244: This is the method apiGetBitcoinExchnage Rate() that uses three different API endpoints from Bitfinex, Bitstamp and Coinbase and returns all three to the fri=ont end view via the controller. Why get three? Well, most API providers publish a guaraunteed uptime for their API servers, which could be something like 95% to 99%. But, as you may find out, sometime one goes down so you don’t get an response back. We could take an average of these three or return the first one that cam back, which would necessitate checking we actually had a value returned to work with. But, in the spirit of moving such code closer to the front end we simply return all exchange rate values.
    - 211 – 239: We use a try catch block to execute the API calls and catch any errors.
    - Line 214: Set up an array to return the results to the controller
    - Line 217 – 219: Use the bitfinex endpoint (btcusd — which I assume stands for bitcoin US dollar rate) to get the latest price from their ticker API and assign the value returned to the $bitfinex variable.
    - Line 218: Use the PHP file\_get\_contents() method to read the file returned into a string by passing it the file variable $bitfinex. This fubnction will either return the read data or FALSE on failure. If you run the bitfinex API endpoint in a browser, Figure 7, you can see that the returned form is JSON and there are several prices of useful bitcoin data. For the moment, we are interested in the last price data. So, because the returned data is now a JSON encoded strung, we need to convert this to a PHP variable, which we call $json, using the PHP json\_decode() function
    - Line 219: looking at Figure 7, we can see that the current exchange rate (there are other rates, mid, bid, ask) is the last\_price, so we can use this to index the value of the last\_price and assign the value to the $priceBitfinex variable. Later on we may use other data such as the timestamp to build a graph of changing exchange rates over time.
    - Line 232: We load the $bitfinex last price into the result array
    - Line 242: and return the result to the controller.



Figure 7: Returned bitcoin last\_price in JSON format

Clearly, the other two exchange rate codes work the same way, the only difference being the name given by each exchange for the current exchange rate. Bitstamp call it ‘last’, Figure 8, and Coinbase returns: {"amount":"11347.70","currency":"USD"}, which shows the current exchange rate as ‘amount’. You can echo these results out if you want to test your PHP code now, however it will be probably just as easy to move on and write the controller method as you have only really just copied the code.

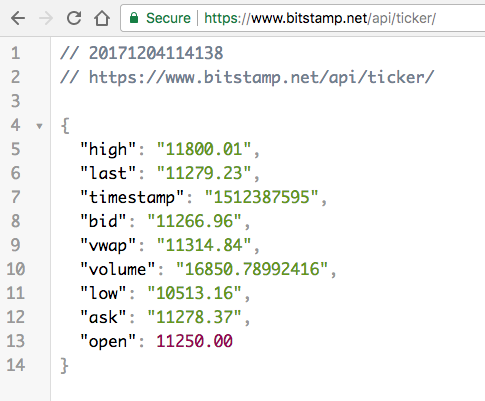


Figure 8: Bitstamp ticker API endpoint response

## Part 4 —Create PHP Controller Methods to Call the PHP Model API Endpoints

Ok, we have just written the PHP method to get some bitcoin change rates. Let’s now write the PHP controller method to call this PHP method, Figure 9.

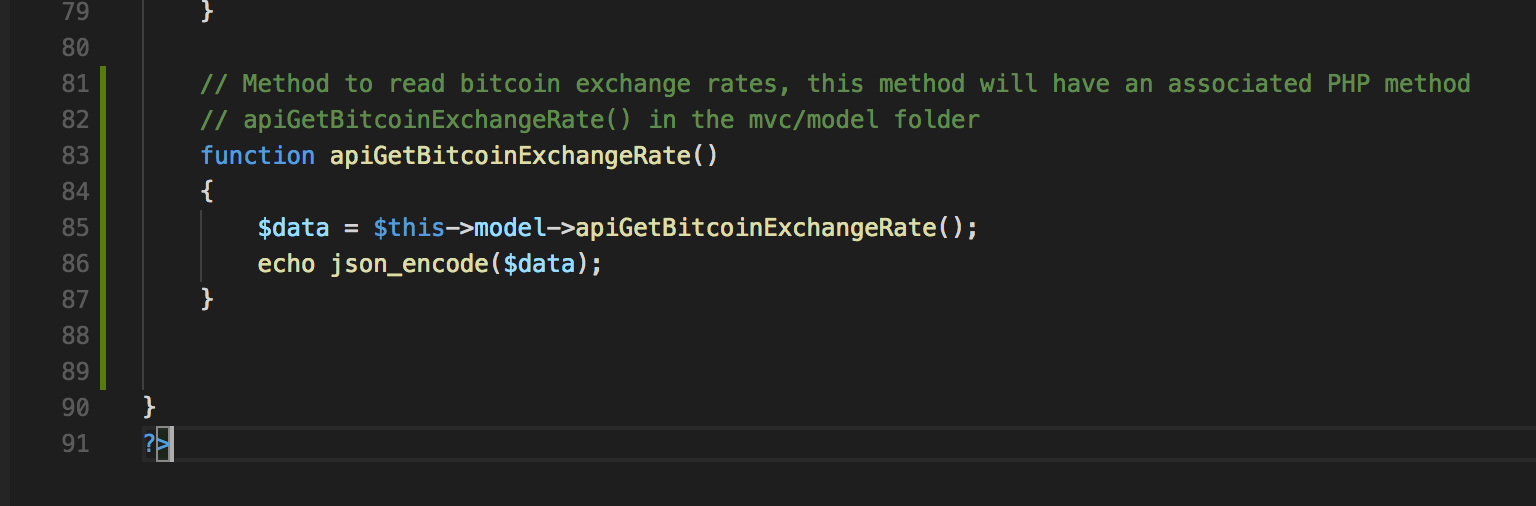


Figure 9: PHP controller method to request exchange rate data from the model

* + Code explanation
    - Line 83 – 87: Well, nothing new here, we simply robbed up one of the other methods you already wrote and converted it. We gave the method a meaningful name, i.e. the same name as its counterpart in the Model class, and we assign the return data value to the $data variable, we then use the json\_encode() function to return a string contacting the JSON presentation of the supplied $data value. This is then echo’d out so it can be picked up by an AJAX based JavaScript frontend.

## Part 5 — Create AJAX Requests for Bitcoin Exchange Data

Let’s have a look at the JavaScript frontend. You have already built the JavaScript file getBitCoinData.js used to call the controller method dbReadBitcoinData(), Figure 26 in the Blockchain Tutorial 1. We will adapt the same file by writing another JavaScript function apiBitcoinExchangeRate(). Figure 10 shows the addition of this apiBitcoinExchangeRate() to the getBitCoinData.js file.

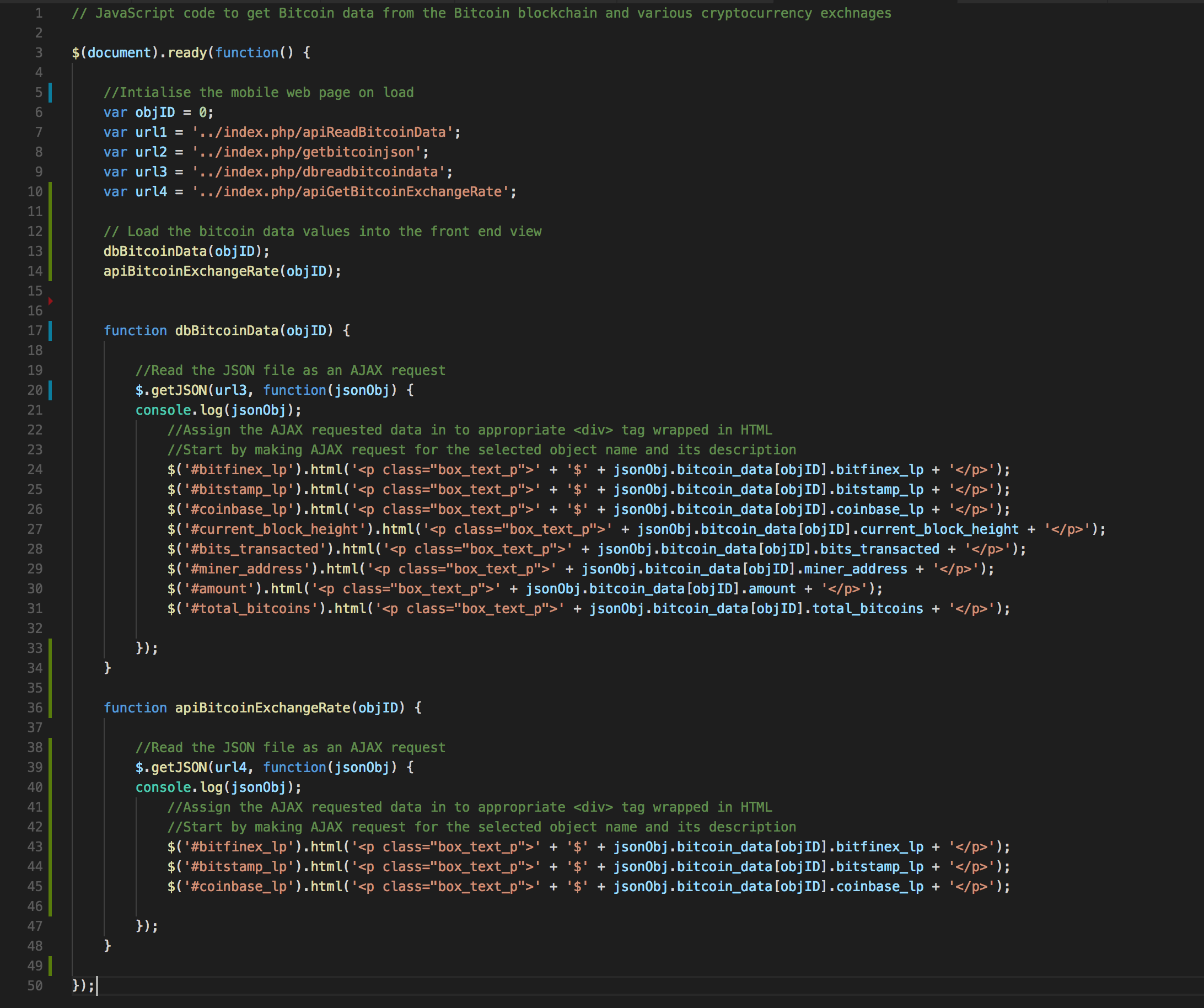


Figure 10: The new JavaScript apiBitcoinExchangeRate() function

* + Code explanation
    - Line 36 – 48: You can see here that we have simply reused the code from the dbBitcoinData() function for the three bitcoin exchanges. We have done some tidying up and renamed the old update() function in Blockchain Tutorial 1 to call it dbBitcoinData(), which better reflects where the bitcoin data is coming from.
    - Line 7 – 10: We have added another URL, i.e. url4, which is the path to the new controller method you just wrote
    - Line 13 - 14: These call the two functions in sequence. We could put a switch round these function invocations, and select them from a button in the front-end view — I’ll leave you to do that. However, I left it like this because it is interesting to observe the results, which can be a bit confusing if an API endpoint is slow to respond. We will consider this in the next section.

## Part 6 — Connect Bitcoin Exchange Data to the View

Make sure that you have downloaded the latest bcExplorerBS.php and bcExplorerDw.php views for GitHub. As mentioned before, we are not going to discuss these in any detail, but provided your PHP code is correct in the Model, Controller and the getBitcoinData.js file (considered part of the view), and you have the correct PHP view files, then your bitcoin exchange rate data should connect to the view.

Consider Lines 13 to 14 in Figure 10. First we invoke the dbBitcoinData() function, which will populate both the bcExplorerBS.php and bcExplorerDw.php views with bitcoin data stored in the database. Then the exchange rate data will be overwritten by the data returned for the exchange rate API endpoints when apiBitcoinExchangeRate() function is invokeed next. You can observe this.

If you switch the invocations round you see still see the database exchange rates come up first, which you might not be expecting because you may forget that the API call is making a round trip and waiting for a server response. Based on the invocation sequence you might be expecting the API exchange rate to be overwritten by the database exchange rate. But, this is not the case because the API may be slower to respond than the local the SQLite database, which allow the database rate to be assigned first, followed by the API rate some time later.

So, be careful what you doing! Invoking either way ends up with the same result (the API rate), after a time lapse, but this may not be what you wanted. You may need to implement some control logic at this point to make sure you get the result you want.

Once you have connected your bitcoin exchange rate data using the API endpoints, your bcExplorerBs.php view should look like that shown in Figure 11. Next, we will add a check button to each of the exchange rate options to select one of them for a currency conversion application.

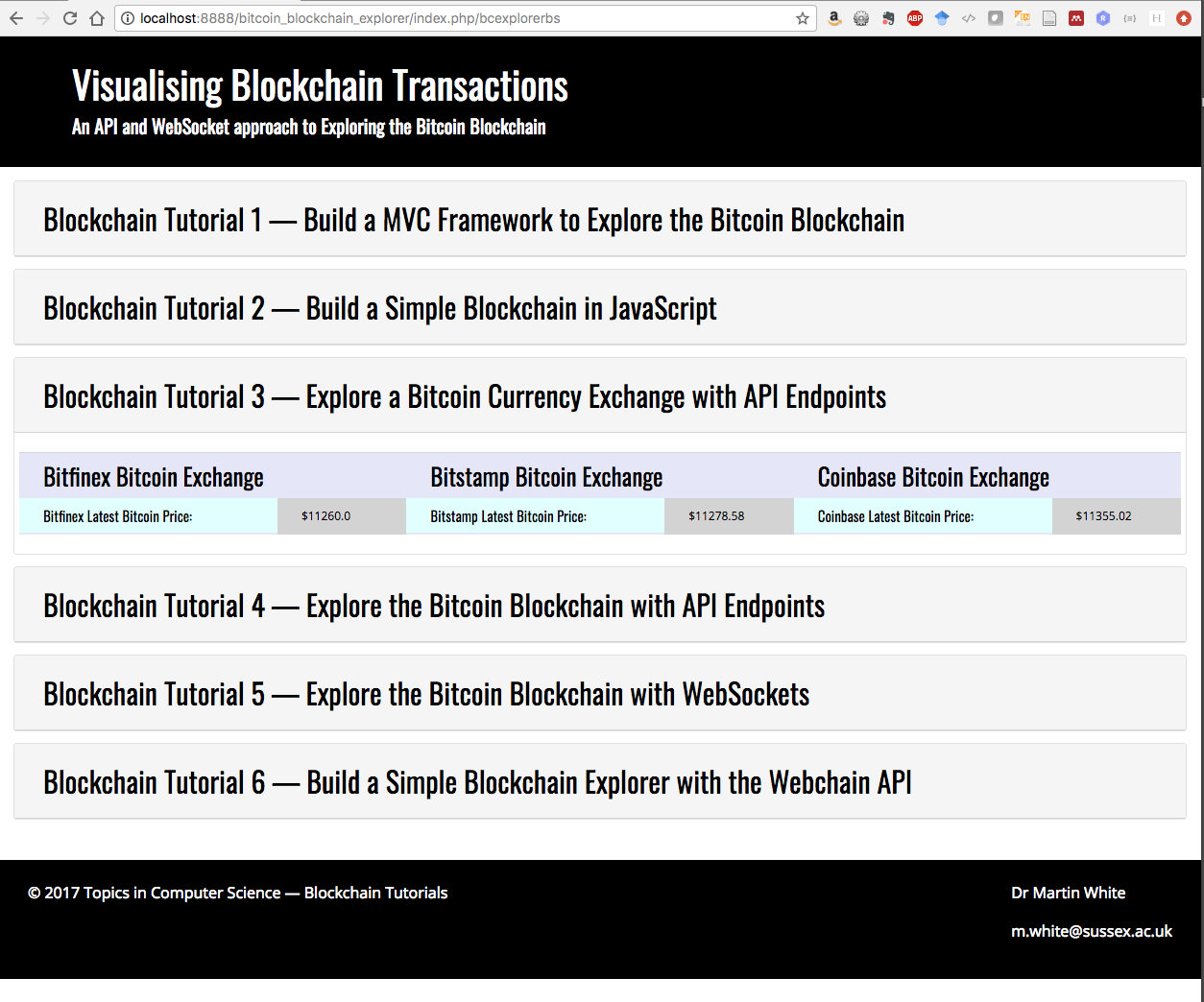


Figure 11: Blockchain Tutorial 3 Bootstrap view showing the returned bitcoin exchange rates using API endpoints

**Part 7 — Build a Bitcoin Currency Exchange Rate Application**

This is an optional exercise. Now that you have a basic MVC framework, with database connectivity and you can exploit bitcoin exchange rate API endpoints you could build a simple bitcoin exchange application to convert from bitcoin to USD and vice versa.

I have added Bootstrap radio button inputs to select between the different bitcoin exchange rate API endpoints, and also added Bootstrap form inputs to input either USD or BTC for conversion. I can tell you, lining up the radio buttons in Bootstrap was bad karma! However, notice we have defaulted to one checked. The idea now is to call either a JavaScript function to convert BTC to USD or a JavaScript function to convert USD to BTC based on the chosen bitcoin rate. So, for example, if we enter a BTC value in the BTC input box it will trigger the btcConvertUsd() function, which will calculate the USD rate and write this back to the USD input box. Similary, if we type a USD figure into the USD input box we want to trigger a JavaScript usdConvertBtc() function to calculate how many BTC we would get at the current exchange rate, and then write this value into the BTC input box. Finally, we will provide a submit and reset buttons for the currency inputs, Figure 12.

We will use JQuery to get our input values for the view, in particular we need to get the form input for the radio button that is checked, and the value of any input box. JQuery has a .val() function to do this.

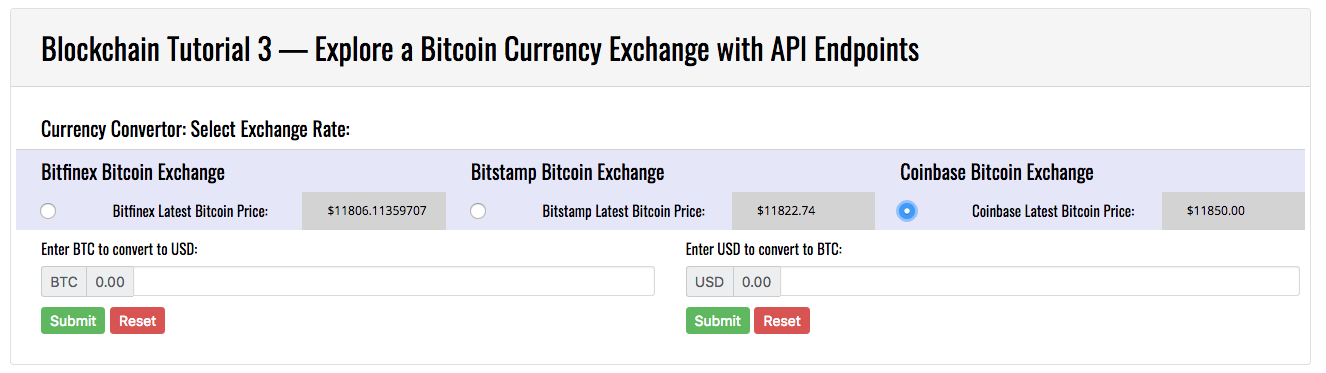


Figure 12: The currency convertor view exploiting the latest bitcoin price

1. We will use the JQuery function .val() to gather the inputs from the view illustrated in Figure 12. We need to know which Bitcoin exchange rate has been selected and we need to get hold of either the BTC value or the USD value typed into the input boxes, compute the currency conversion and write the result back to the other input box. We will write the JavaScript code for this in the getBitcoinData,js file, so open this file in your editor environment and create the JavaScript code to capture the input for the currency convertor, Figure 13.
   * Code explanation
     + Line 51 – 57: These are just stubs for the two currency convertor functions that we will write in a minute.
     + Line 60 – 74: This JavaScript code uses the JQuery .click() function to read the value of the input in the BTC input box using the JQuery .val() function.
     + Line 63: Grabs the BTC value from the input box using the id selector (#btcValue) for that input box and assigns it to the btcValue variable.
     + Line 63 – 70: Uses an ‘If else’ statement to check if a valid input BTC value has been input. Actaully, that is not true, we ar just checking that a value has been input for now. If not not, and the submit button is pressed, then we alert to ask for a valid input value.
     + Line 68: This is the interesting part. Here we need to make sure that input is actually a valid BTC value — this could be any text string, but clearly, we want a BTC value to two decimal places. If we have a valid BTC value, then we will input this to the btcConvertUsd() function and write the returned USD value to the USD input box using its #usdValue selector id and the .val() function. But, for now we simply alert the user and assume that a valid BTC value is input.
     + Line 72: We use the JQuery .click() function to detect a reset and hence reset the input box when the reset button is pressed by assigning a blank string to the id selector
     + Line 77 – 91: This is the same as the previous function, except here we are nor processing the USD input box. As before, we use the JQuery .click() function attached to the submit button id selector #usdInput, and so on.
     + Line 95 to 99: This time we used the JQuery .change() function to capture the value of the radio button selected. Each radio button is associated with a particular bitcoin exchange, so we need to process the value so that we know which bitcoin exchange rate to use in the conversion functions. We may need to put this code in a function, or make the variable global or something, not sure yet?

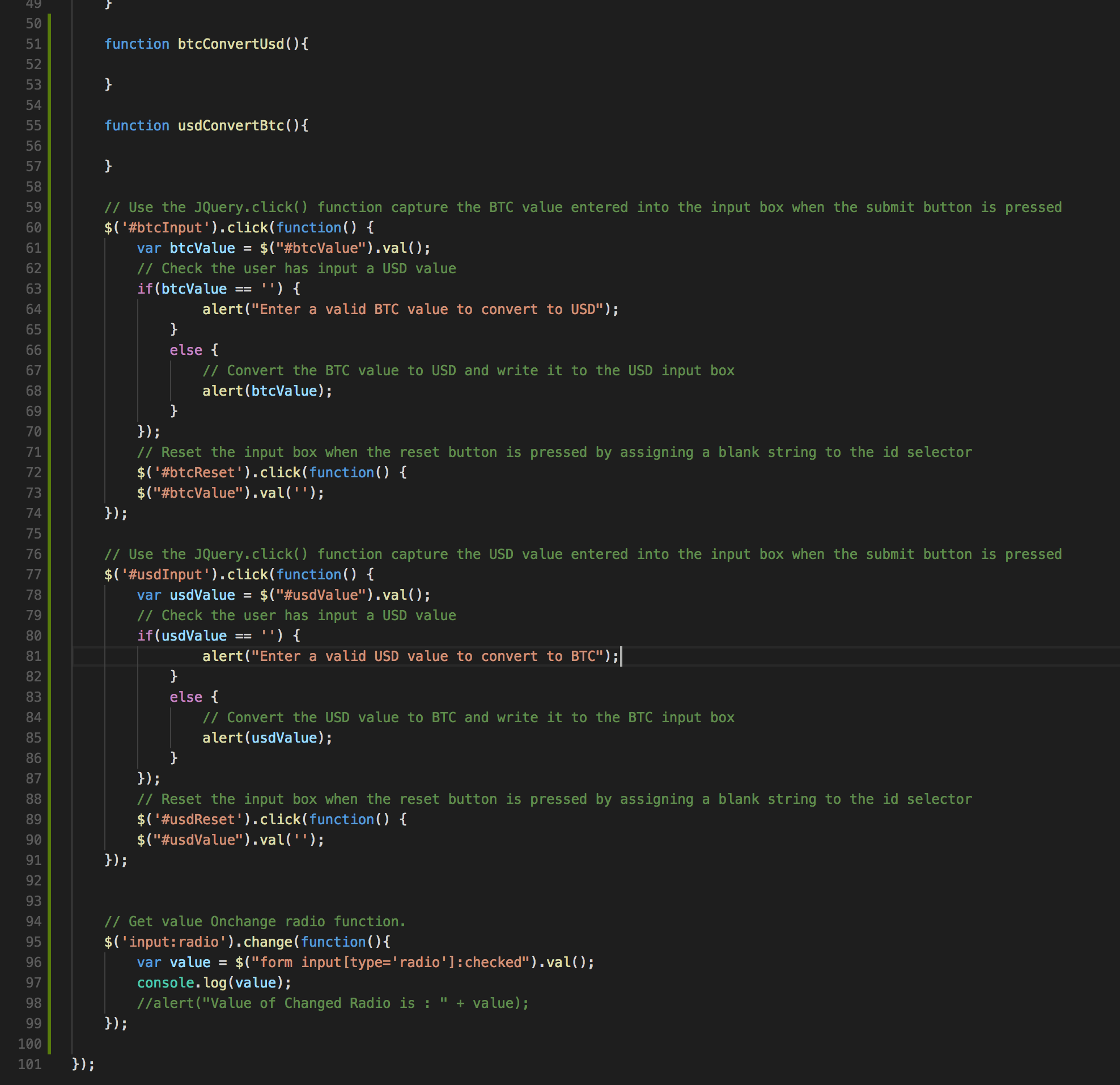


Figure 13: JavaScript code to capture the currency convertor form inputs

1. Ok, once you have this code entered (or copied form the GitHub download, etc.) you can test it works, if not debug it. Make sure you have the given view (bcExplorer.php) connected up, etc. Open the Google Chrome Inspect panel and click on Console. Test the radio buttons in turn, you should see the console return a message for each button, Figure 14. You will need to probably change this message in the view to something you can easily test and store a variable against on line 97 in Figure 13, to replace the console.log() function.

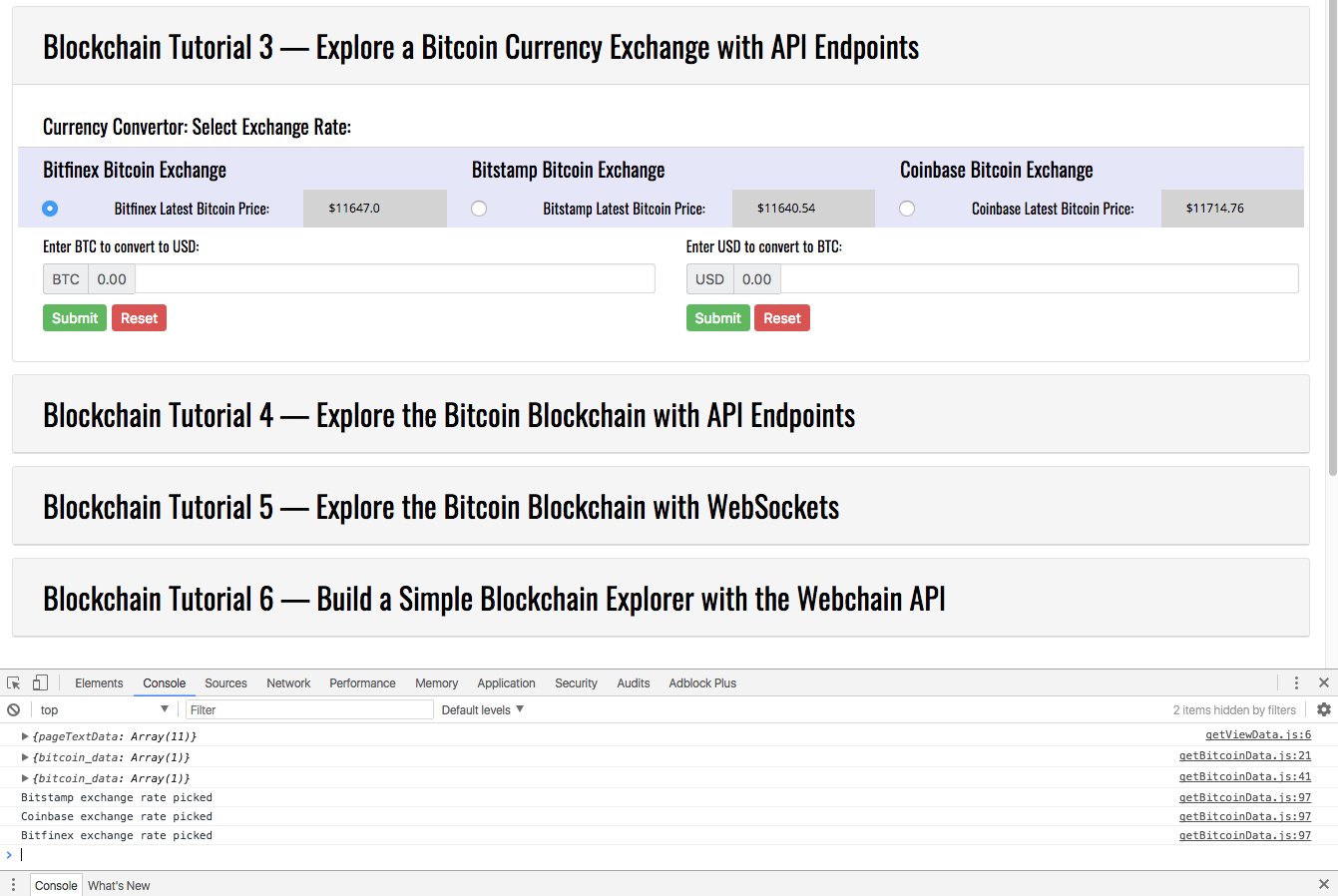


Figure 14: Test the radio buttons

1. Test also the BTC and USD input boxes. Press the submit button for each without entering a value. You should get an alert dialogue message back, Figure 15. Similarly, an alert will output the value you enter.

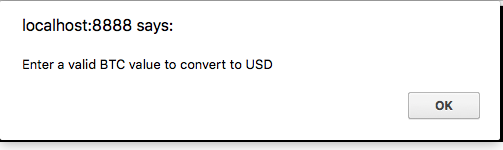


Figure 15: Alert if submitting an empty input value

1. Ok, once you have tested the code we need to complete the conversion functions and hook them up. Let’s start with the btcConvertUsd() function, Figure 16.

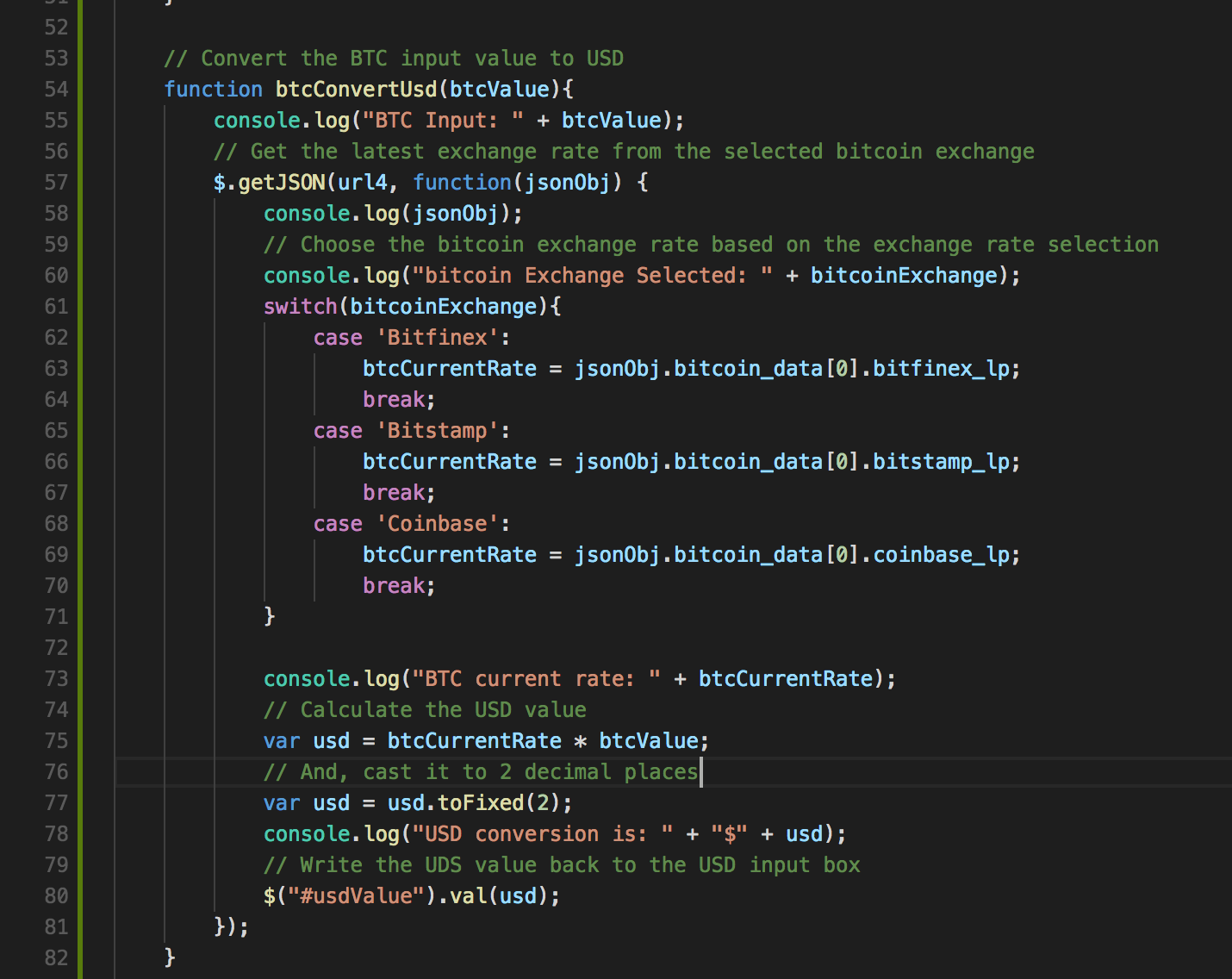


Figure 16: JavaScript function to convert BTC to UDS based on a selected bitcoin exchange last price.

* + Code explanation
    - Line 54 – 82: This is the btcConvertUsd() function that takes as an input value the BTC value you type into the BTC input box — remember we have no error checking, so you must type in a proper value! Here we are passing a parameter, btcValue, by value into the JavaScript function. We could have made btcValue a global variable
    - Line 57 – 81: We execute the .getJSON() function to get the three bicoin exchange rates using the API endpoints indicated by url4. We then write the handler to proves the returned jsonObj between lines 58 and 81. You can ignore the console.log() statements, these were added to trace the programme execution as I developed the code.
    - Line 61 – 71: The first thing we need to do is find out which bitcoin exchange rate was selected using the radio buttons on the view. You will recall from above that we used the JQuery .change() function to do this. Look at Figure 13, lines 95 to 99, here we are trying to find the ‘value’ of the checked radio button. When you tested this above it returned a message string, I just shortened these message strings to ‘bitfinex’, ‘bitstamp’ and ‘coinbase’ so we can test against these in the currency convertor functions. Further, I have changed variable name ‘value’ to something more expressive, i.e. ‘bitcoinExchange’, to hold the string pertaining to the bitcoin exchange selected, and I have made this variable global (and defaulted to ‘Bitfinex’) otherwise I would have to incorporate this code into the bitcoin currency convertor functions, Figure 17.
    - Line 61 – 71: We use a switch statement to resolve which bitcoin exchange has been selected based on the value of the variable bitcoinExchange, and set the variable btcCurrentRate to the selected last price returned by the appropriate API endpoint.
    - Line 75: We calculate the USD value (variable usd) where the bitcoin last price (dollar exchange rate) is multiplied by the amount of BTC enterted in the BTC input box (btcValue).
    - Line 77: Convert the USD value (usd) to 2 decimal places.
    - Line 80: Write the USD value into the USD Input box identified by the id selector #usdValue using the JQuery .val() function. Note, this will effectively overwrite any existing USD value in the USD input box.

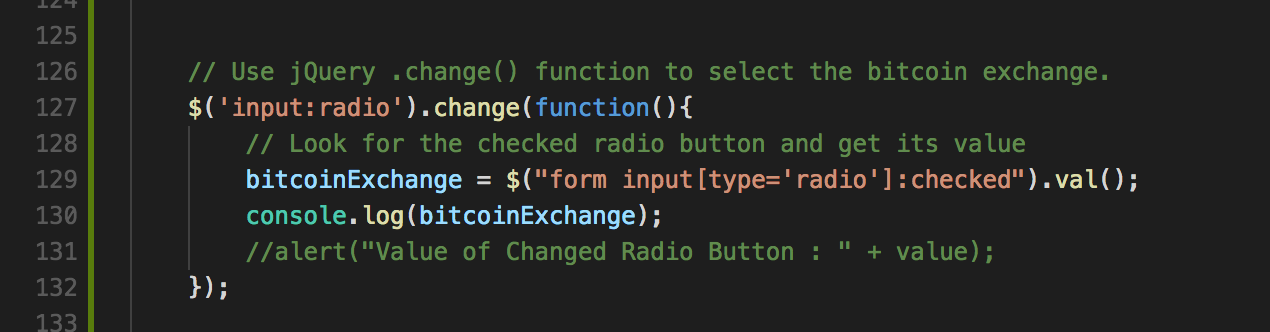


Figure 17: Updated JavaScript to select the bitcoin exchange

1. Test the BTC to USD conversion works as designed. We can see from Figure 18 that we have selected the Coinbase exchange rate, we have entered 1 BTC into the BTC input box. But, we have returned 12868.85 dollars. Why is this? The reason is because, the three exchange rates are being called via the API endpoints as before, but they have not received their responses back yet, so the value inserted comes from the database. In reality, you might add some more PHP code in the Model to update the database with the last value you retrieved from the API endpoints. If we wait a fraction more, the exchnages rates will update when they get the response. However, as we just saw for writing the btcConvertUsd() function, we called the API endpoint again and this got a response back before the radio button versions and thus we returned the USD value. I don’t know how the different bitcoin exchange servers are scheduling their responses. If we run the test again, shows that this time we got the exchange rates overwriting the database values and we can see that the coinbase values retrieved match, Figure 19. However, because the exchange rate (last price) can update between two AJAX requests, sometimes you see slightly different exchange rates. In reality, you probably would not use an API endpoint in this way, i.e. expose to the customer these differences. You would probably get the exchange rate in real-time using a WebSocket, store the value you want to offer, and update this depending on the market forces. You can imagine how banks make money on currency exchange, they publish an exchange rate for a certain time, then have to bet that it goes in their favour.

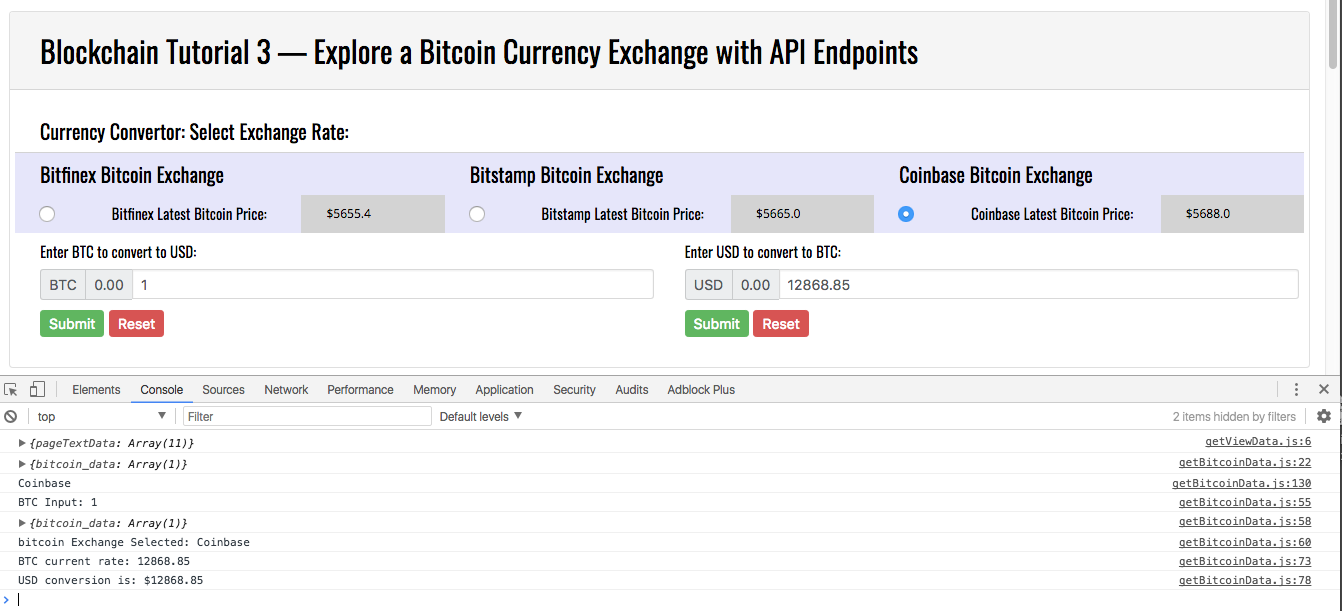


Figure 18: Test the BTC to USD conversion works

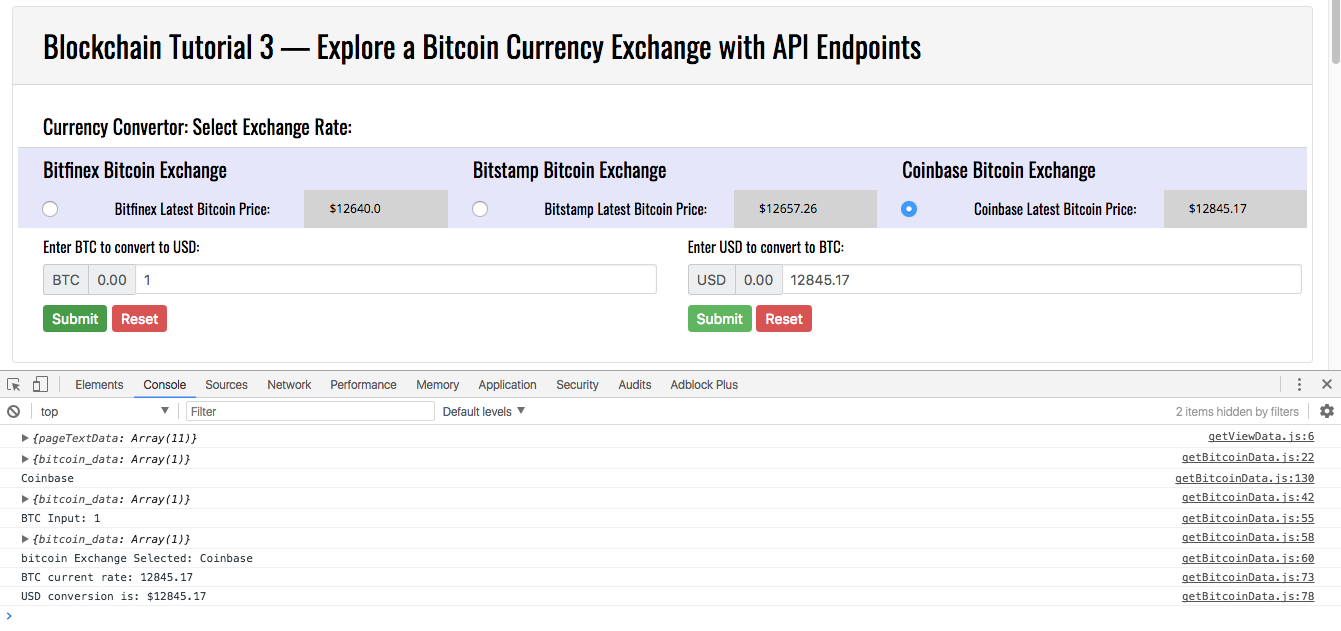


Figure 19: Test the BTC to USD conversion works, again

1. Next, create the usdConvertBtc() function and make sure that work. S. Clearly, this is virtually the same as the btcConvertUsd() function, except that we now divide to get BTC form the USD, and we write he results back to the BTC input box. I will leave you to do this, but Figure 20 shows the working USD to BTC conversion. Note, that we have set the BTC to 8 decimal places.

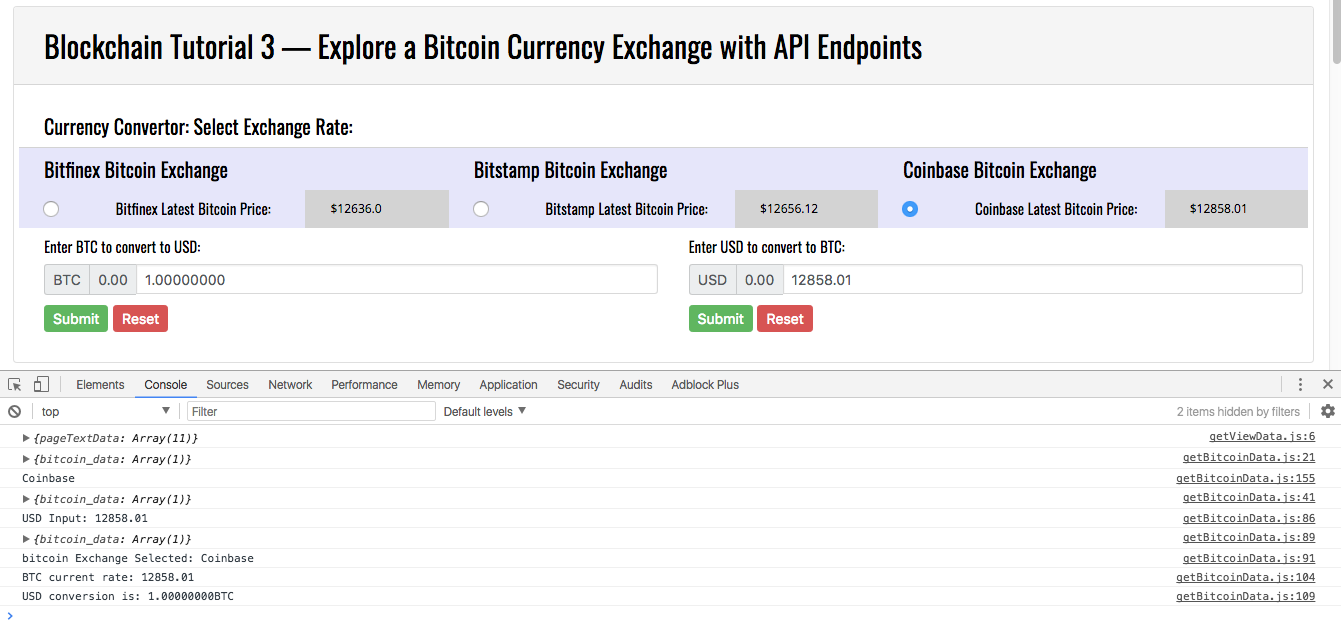


Figure 20: The working USD to BTC conversion

Once you have tested that this works you have concluded the Blockchain Tutorial 3.

Next, in Blockchain Tutorial 4, we will examine other API endpoints to explore the blockchain in a bit more depth. We will investigate the miner who solves the current block, see how many BTC this miner holds, etc. — this will be a shorter tutorial intended to reinforce your knowledge of the bitcoin blockchain structure.

Finally, we are not actually exploring a bitcoin currency exchange, we have built one, albeit a simple one, so change the Bootstrap Card title to reflect, Figure 21. There are, of course many other features that you could include, some already alluded to above, like storing exchange rates for a time. In general, currency markets are volatile, with exchange rates changing by the second. Such currency risk can be effectively hedged by locking in an exchange rate using something called currency futures, forwards and options, which is beyond the scope of this tutorial[[1]](#footnote-1). Suffice to say a when you buy a currency to go on holiday, for example, you usually buy at that day’s rate, you don’t see the value going up and down like a yo yo! However, even though bitcoin is very volatile it makes sense to use the same rate that is advertised in the currency convertor. At the moment, the currency convertor only loads the rate when the app is loaded so you could add a feature to lock that in, i.e. store that value in SQLite as if that was the rate for that period. You could add a feature to refresh that rate. And, you could then take the exchange rate for the store values. You could also add other exchange rates, e.g. other fiat currencies, which would require you to find another API endpoints to grabs those exchange rates. I am sure there is lots more you could play with.

The next Blockchain Tutorial 4 is focused on exploring the Bitcoin blockchain with API endpoints in a bit more depth, just to make sure you are more familiar with the Bitcoin blockchain and block structure.

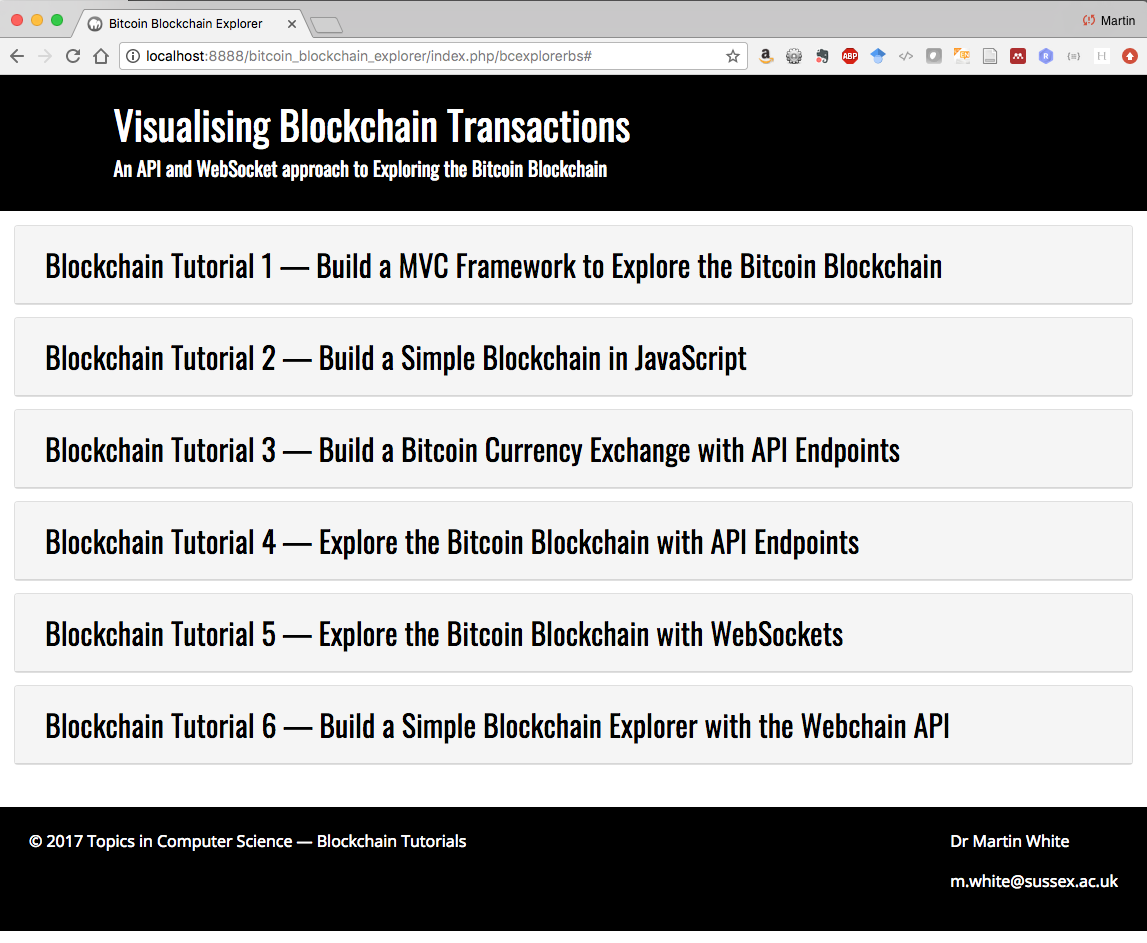


Figure 21: Change the title to Build, rather than Explore for Blockchain Tutorial 3

1. https://www.investopedia.com/articles/forex/051415/how-lock-exchange-rate.asp [↑](#footnote-ref-1)