



LAB 4 :

Web Application

JavaScript and JQuery / Ajax

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX

Objective:

In this lab we are going to develop a dashboard for restaurant table management.

Steps:

1. Use Case:

You are given a requirement for a restaurant to develop an ordering system using an electronic menu for its customer. At the dining table, customers will find an electronic menu controlled by a microcontroller with a menu linked to its touchpads.

Your job is to develop an embedded application such that, whenever a customer touches the selected touchpad designated with a menu set, the microcontroller automatically places the order onto the restaurant dashboard. The customer can make repeated orders by touching different touchpads designated with other sets of menus, or repeatedly touches the same touchpad to make multiple orders.

The microcontroller relies on WiFi connection available in the restaurants to connect to the dashboard server that manages the order. Cooks in the kitchen will use the information to cook the meals based on the orders displayed on the dashboard.

The dashboard developer has given the endpoints details and methods to use in order for the IoT to be able to send data successfully to the back-end system. It used REST technology to accomplish this data communication process.

16:41:07

Restaurant Table Management 7-3-2020

Tables Arrangment

Table A	Table B	Table C
Food	Food	Food
Drinks	Drinks	Drinks
Table D	Table E	Information
Food	Food	
Drinks	Drinks	

copy-right by tuzishariff@gmail.com

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX

14:00:11
Restaurant Table Management
8-3-2020

The order data sent by IoT device to the restaurant dashboard server will automatically displayed on the dashboard for cooks in the kitchen

Tables Arrangement

Table A

Serving
Sun Mar 8 07:04:47 2020

FOOD
• beef burger and cheese

DRINKS
• iced pepsi

Table B

Serving
Sun Mar 8 07:06:49 2020

FOOD
• lamb chop in barbeque sauce

DRINKS
• hot coffee with cream

Table C

Booked by WSkill
Sun Mar 8 07:10:25 2020

FOOD

DRINKS

Table D

Complete
Sun Mar 8 07:08:39 2020

FOOD
• lamb chop in black-pepper sauce
• beef burger and cheese

DRINKS
• hot chocolate with cream
• iced pepsi

Table E

BOOKED by WSkills
Sun Mar 8 05:52:10 2020

FOOD

DRINKS

Information

2. The below endpoint format that we will use to access the restaurant web application API.

`http://<servername_ip>:8000/status/<table_ID>`

3. Here is the list of REST Endpoints.

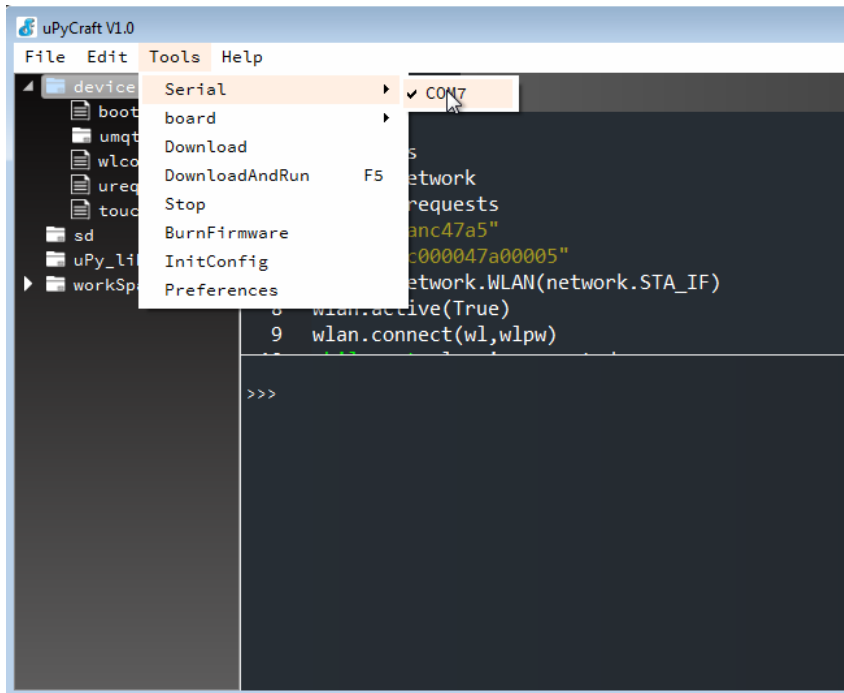
No	End Point (URI)	methods	Data (JSON)
1	<code>http://<servername_ip>:8000/table/<table_ID></code>	POST	data: <pre>{ "food":["name food of menu"], "Drinks":["name of drink"] }</pre> Successful Response: <pre>{ "Status": "idle", "drinks": ["hot chocolate with cream"], "food": ["lamb chop in black-pepper sauce"], "status": "Booked by WSkill", "table": "C", "time": "Sun Mar 8 07:10:25 2020" }</pre>
2	<code>http://<servername_ip>:8000/status/<table_ID></code>	POST	Data: <pre>{ "status": "BOOKED by WSkills" }</pre> Successful response: Updated: table C, BOOKED by WSkills

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX

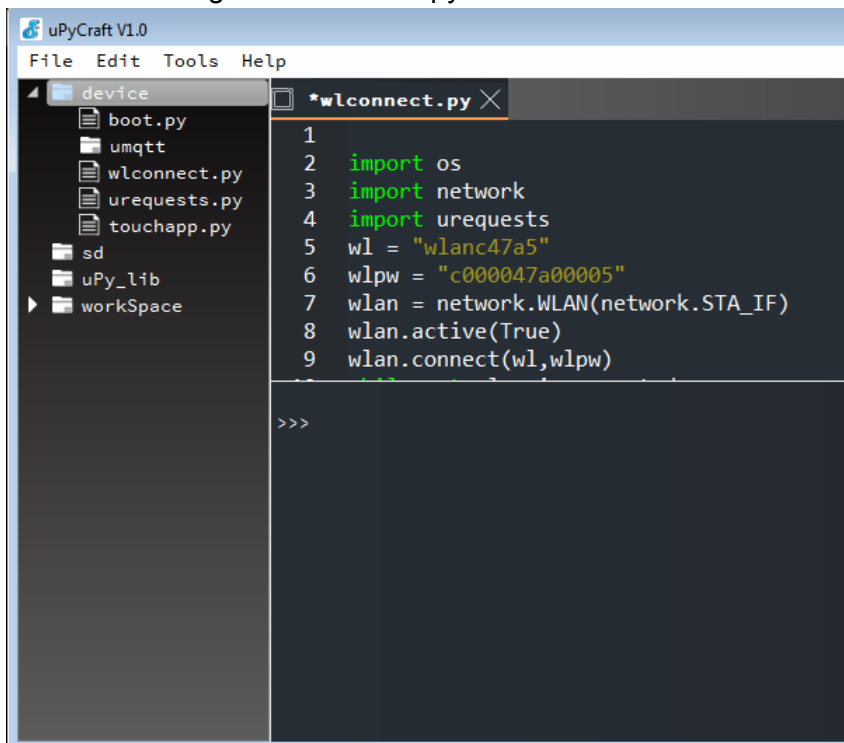
3	http://<servername_ip>:8000/main page	GET	Successful response: Table layout HTML page
4	http://<servername_ip>:8000/tables	GET	Successful response: JSON data: <pre>{ "A": { "drinks": ["iced pepsi"], "food": ["beef burger and cheese"], "status": "Serving", "table": "A", "time": "Sun Mar 8 07:04:47 2020" }, "B": { "drinks": ["hot coffee with cream"...} }</pre>
5	http://<servername_ip>:8000/menu	GET	application/JSON response with menu database. <pre>{ "dnr-1": { "drinks": "Iced Fresh Pineapple Juice", "food": "Grilled Lamb Chop in Black Pepper Sauce with Baked Potato Salad", "price": 22.5 }, "dnr-2": { "drinks": "Iced Kasturi Lime Juice with Asam Boi", "food": "Spicy Fried Rice with Grilled Beef in Percik Sauce ", "price": 19.5 }, } }</pre>

- Now let's start by programming the microcontroller to connect to WiFi and perform some http requests from the dashboard server by using urequest.py library.
- Connect your microcontroller to the USB port and allow it to boot properly.
- Make sure the uPyCraft IDE is connecting to the correct com port...In this case COM7

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX



7. Click on the device, list of older programs may exist...
8. Create a new python file and give it a name wlconnect.py.
9. Ensure that you have the necessary libraries copied to the device directory as well...
10. In this case we need “urequests.py”.
11. Now start editing the “wlconnect.py”



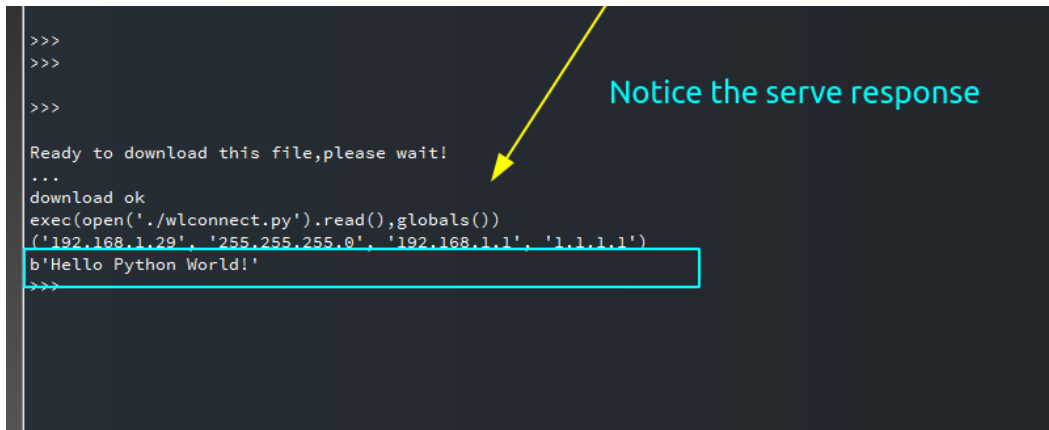
LAB 4 : WEB APPLICATION JAVAScript AND JQUERY / AJAX

Let test a HTTP requests to a sample web application server

```
import os
import network
import urequests
wl = "wlanc47a5"
wlpw = "c000047a00005"
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(wl,wlpw)
while not wlan.isconnected:
    pass
print(wlan.ifconfig())

websvr = "http://192.168.1.23:8000"

rsps = urequests.get(websvr)
print(rsps.content)
```



```
>>>
>>>
>>>
Ready to download this file,please wait!
...
download ok
exec(open('./wlconnect.py').read(),globals())
('192.168.1.29', '255.255.255.0', '192.168.1.1', '1.1.1.1')
b'Hello Python World!'
>>>
```

Notice the serve response

12. Congratulations ! You have successfully programmed an embedded application using an IoT device.
13. Now let us test the REST endpoint related to restaurant application....
14. Let's try to access the data from the server using HTTP / GET requests from one of the end-points.

Use this endpoint: "http://<servername_ip>:8000/tables"

```

>>> uri = websvr + "tables"
>>> uri
'http://192.168.1.23:8000tables'
>>> uri = websvr + "/tables"
>>> uri
'http://192.168.1.23:8000/tables'
>>> rsps = urequests.get(uri)
>>> print(rsps.content)
b'{\n  "A": {\n    "drinks": [\n      "iced pepsi"\n    ], \n    "food": [\n
"beef burger and cheese"\n    ], \n    "status": "Serving", \n    "table": "A",
\n    "time": "Sun Mar  8 07:04:47 2020"\n  }, \n  "B": {\n    "drinks": [\n
"hot coffee with cream"\n    ], \n    "food": [\n      "lamb chop in barbeque
sauce"\n    ], \n    "status": "Serving", \n    "table": "B", \n    "time": "Sun
Mar  8 07:06:49 2020"\n  }, \n  "C": {\n    "Status": "idle", \n    "drinks":
[\n      "hot chocolate with cream"\n    ], \n    "food": [\n      "lamb chop in
black-pepper sauce"\n    ], \n    "status": "BOOKED by WSkills", \n    "table":
"C", \n    "time": "Sun Mar  8 14:17:37 2020"\n  }, \n  "D": {\n    "drinks":
[\n      "hot chocolate with cream", \n      "iced pepsi"\n    ], \n    "food":
[\n      "lamb chop in black-pepper sauce", \n      "beef burger and cheese"\n
], \n    "status": "Complete", \n    "table": "D", \n    "time": "Sun Mar  8
07:08:39 2020"\n  }, \n  "E": {\n    "drinks": [\n      "iced pepsi"\n    ], \n
"food": [\n      "beef burger and cheese"\n    ], \n    "status": "BOOKED by
WSkills", \n    "table": "E", \n    "time": "Sun Mar  8 05:52:10 2020"\n  }, \n
"information": ""\n}\n'
>>>

```

15. From the example above, we use micropython to build the endpoint based on its existing variable values in memory...

```
uri = websvr + "tables"
```

Here uri will be our endpoint variable, we concatenate the value server address in variable websvr and test the value:

```
'http://192.168.1.23:8000tables'
```

This URI value seems to be correct, but there is a slight mistake.... It is missing one “/” before the parameter “tables”

It is easily fixed in the code follows :

```
uri = websvr + "/tables"
```

The code below re-execute REST request to the server:

```
rsps = urequests.get(uri)
```

16. After execution, REPL prompted with no error, this means that the REST request has been successful .

The “rsps” variable is the variable corresponding to the request object, now it holds the data being sent by the server response.

```
print(rsps.content)
```

This command prints the content or data available in rsps object as shown below...

```
b'{"A": {"drinks": ["iced pepsi"], "food": ["beef burger and cheese"], "status": "Serving", "table": "A", "time": "Sun Mar 8 07:04:47 2020"}, "B": {"drinks": ["hot coffee with cream"], "food": ["lamb chop in barbeque sauce"], "status": "Serving", "table": "B", "time": "Sun Mar 8 07:06:49 2020"}, "C": {"Status": "idle", "drinks": ["hot chocolate with cream"], ...}
>>>
```

17. The data is a large string object that can be processed as a JSON object. Now let's try to post some data according to REST API endpoint and data structure format.

Let say, the customer wants to order as set for:

Food: "Fish and Chips with Salads"
Drink: "hot coffee latte"

The the JSON data format show look like as follows:

```
{
  "food":["fish and chips with salads"],
  "drinks":["hot coffee latte"]
}
```

18. Now we need to create a python object to hold this data.

```
menu = {"food":["fish and chips with salads"],
        "drinks":["hot coffee latte"]}
```

19. To achieve this we can use python built in function “dict”, the function that builds a dictionary object which has a similar structure as JSON data.

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX

```
>>> menu = dict(food = ["fish and chips with salads"], drinks = ["hot coffee latte"])
>>>
>>> menu
{'drinks': ['hot coffee latte'], 'food': ['fish and chips with salads']}
>>> |
```

20. It looks like “food” and “drinks” switch positions, but luckily it doesn’t matter to python.

Now the data is ready to be posted to the server. What we need now is the correct server address and end-point that will be able to process the data..

Checking the REST end-points of the web application we the end point is:

`http://<servername_ip>:8000/table/<table_ID>`

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX

22. So let's build the URI for this end-point, and assign it to a variable.
We choose table A, since the Table is available...

Table A	
idle	
FOOD	DRINKS

```
>>> menu = dict(food = ["fish and chips with salads"], drinks = ["hot coffee latte"])
>>>
>>> menu
{'drinks': ['hot coffee latte'], 'food': ['fish and chips with salads']}
>>> uri
'http://192.168.1.23:8000/tables'
>>> websvr
'http://192.168.1.23:8000'
>>> uri = websvr + "/table" + "/A"
>>> uri
'http://192.168.1.23:8000/table/A'
>>> |
```

23. With the python code shown above , now the “uri” object holds the correct end-point value that we need.

```
'http://192.168.1.23:8000/table/A'
```

24. Next we will post the menu data to the endpoint using micro python “urequests” REST client library.
Another piece of information is required to inform the web application server that we want the data to be processed as a JSON application. Or else, it assumes XML/HTML which is designed for human visualization.

So the information is fed to header parameter of urequests object like shown below:

```
headers = {'content-type': 'application/json'}
```

25. Now the complete instruction to use for micropython will be as follows:

```
>>> rps = urequests.post(uri, json=menu, headers = headers)
```

26. Lets run the command.... You should receive confirmation from the web application server the response like shown in the screenshot below.

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX

```
uri = websvr + "/table" + "/" + A"
>>> uri
'http://192.168.1.23:8000/table/A'
>>> menu
{'drinks': ['hot coffee latte'], 'food': ['fish and chips with salads']}
>>> headers
{'content-type': 'application/json'}
>>> rps = urequests.post(uri, json=menu, headers=headers)
>>> rps.content
b'{"drinks": ["hot coffee latte"], "food": ["fish and chips with salads"], "status": "idle", "table": "A", "time": ""}'
>>>
```

27. Response from successful POST requests to end-point for ordering a dinner set of the restaurant web application. We can now check how it appears on the dashboard.

Table A	
idle	
FOOD	DRINKS
<ul style="list-style-type: none">fish and chips with salads	<ul style="list-style-type: none">hot coffee latte

28. We also need to send another piece of JSON data to set the table status as being served...

The endpoint should be:

`http://<servername_ip>:8000/status/<table_ID>`

29. Let's set the status as "Serving now..."

Our JSON data should be:

```
Data:
{
  "status": "Serving now..."
}
```

```
uri = websvr + "/status" + "/" + A"
>>> uri
'http://192.168.1.23:8000/status/A'
>>> tblsts = dict(status = "Serving now..."
... )
>>> tblsts
{'status': 'Serving now...'}
>>> rps = urequests.post(uri, json = tblsts , headers=headers)
>>> rps.content
b'Updated: table A, Serving now...'
>>>
```

30. So now let's take a look on the dashboard again.

<h3>Table A</h3> <p>Serving now...</p> <p>Sun Mar 8 22:18:32 2020</p>	
FOOD <ul style="list-style-type: none"> fish and chips with salads 	DRINKS <ul style="list-style-type: none"> hot coffee latte

As you can see, the information about the menu being ordered and the status of the table being served are displayed correctly for human visualization.

31. Congratulations ! Your IoT is capable of using REST data communication protocol to work as required by the web application API.

32. Finally you need to put these commands or codes into a python function or module so that it can be called by other python functions such as a function that runs when touchpad is activated.

Alright... now lets us put all the pieces of code together to perform as one embedded application in an IoT device.

33. Let's begin with the menu codes:

You can be creative in this task, just imagine your favorite food...
Here are four suggested menus that water your mouth...!

SET ID	Servings	Category
snk-1	food: Pineapple Shrimp Sandwich with Cheese and Pepper drinks: Hot black coffee	Snack - I'm not hungry
lch-1	food:BBQ Chicken Maryland and Steamed Butter Rice and Curry Gravy drinks: Iced lemon tea	Lunch - I'm hungry
dnr-1	food:Grilled Lamb Chop in Black Pepper Sauce with Baked Potato Salad drinks:Iced Fresh Pineapple Juice	Dinner - I'm very hungry
dnr-2	food:Spicy Fried Rice with Grilled Beef in Percik Sauce drinks:Iced Kasturi Lime Juice with Asam Boi	Dinner - I'm very hungry

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX

35. We must assume that the developer of the web application will be able to provide us with this data via its API interface and the way to access to the resources by using the endpoint:

`http://<servername_ip>:8000/menu`

We can test the API service from IoT itself...

This is how we do it...

```
>>> websvr
'http://192.168.1.23:8000'
>>> rps = urequests.get(websvr + "/menu")
>>> rps.content
b'{\n  "dnr-1": {\n    "drinks": [\n      "Iced Fresh Pineapple Juice"\n    ], \n    "food": [\n      "Grilled Lamb Chop in Black Pepper Sauce with\n      Baked Potato Salad"\n    ], \n    "price": 22.5\n  }, \n  "dnr-2": {\n    "drinks": [\n      "Iced Kasturi Lime Juice with Asam Boi"\n    ], \n    "food": [\n      "Spicy Fried Rice with Grilled Beef in Percik Sauce "\n    ], \n    "price": 19.5\n  }, \n  "lch-1": {\n    "drinks": [\n      "Iced lemon tea"\n    ], \n    "food": [\n      "BBQ Chicken Maryland\n      and Steamed Butter Rice and Curry Gravy"\n    ], \n    "price": 17.5\n  }, \n  "snk-1": {\n    "drinks": [\n      "Hot black coffee"\n    ], \n    "food": [\n      "Pineapple Shrimp Sandwich with Cheese and Pepper"\n    ], \n    "price": 9.5\n  }\n}'
>>> menus=json.loads(rps.content)
>>> menus
{'snk-1': {'drinks': ['Hot black coffee'], 'price': 9.5, 'food': ['Pineapple Shrimp Sandwich with Cheese and Pepper']}, 'dnr-1': {'drinks': ['Iced Fresh Pineapple Juice'], 'price': 22.5, 'food': ['Grilled Lamb Chop in Black Pepper Sauce with Baked Potato Salad']}, 'dnr-2': {'drinks': ['Iced Kasturi Lime Juice with Asam Boi'], 'price': 19.5, 'food': ['Spicy Fried Rice with Grilled Beef in Percik Sauce ']}, 'lch-1': {'drinks': ['Iced lemon tea'], 'price': 17.5, 'food': ['BBQ Chicken Maryland and Steamed Butter Rice and Curry Gravy']}}
>>>
>>> menus.keys()
dict_keys(['snk-1', 'dnr-1', 'dnr-2', 'lch-1'])
>>> menus['lch-1']
{'drinks': ['Iced lemon tea'], 'price': 17.5, 'food': ['BBQ Chicken Maryland and Steamed Butter Rice and Curry Gravy']}
```

36. Now we can see that we can access each menu information available on the web application server via REST API using an IoT microcontroller. This means that this information can be used to place orders on restaurant applications by using python codes that will respond to customer input such as a touchpad.

Here is the combined code into one IoT touchpad order application utilizing two touchpad sensors as inputs...

```
#This application is for embedded program to utilize touchpad sensors for restaurant ordering system
```

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX

```
#author fuzi shariff
#email: fuzishariff@gmail.com

import wlconnect    #your separate module to connect to WiFi
import urequests
from machine import TouchPad, Pin
from utime import sleep
led = Pin(15,Pin.OUT)
tch = TouchPad(Pin(4))
tch.config(2000)
tch2 = TouchPad(Pin(13))
tch2.config(2000)
import json

websvr ="http://192.168.1.23:8000"
headers = {"content-type":"application/json"}

def getMenu():
    rps = urequests.get(websvr+"/menu")
    menus = json.loads(rps.content)
    rps.close()
    return(menus)

def orderMenu(tableID, menuKey):
    menus = getMenu()
    rps = urequests.post(websvr + "/table/" + tableID, json=menus[menuKey], headers=headers)
    print(rps.content)
    rps.close()

while True:
    d = tch.read()
    if d < 490:
        led.on()
        print("You touched ESP32 sensor !")
        menulis = getMenu()
        orderMenu("A",'lch-1')
        sleep(1)
    led.off()
    sleep(0.2)

    d = tch2.read()
    if d < 490:
        led.on()
        print("You touched ESP32 sensor !")
        menulis = getMenu()
        orderMenu("B",'dnr-1')
        sleep(1)
    led.off()
    sleep(0.2)
```

LAB 4 : WEB APPLICATION JAVASCRIPT AND JQUERY / AJAX

38. The dashboard application displayed the orders as shown below:

12:24:49

Restaurant Table Management

10-3-2020

Tables Arrangment

Table A	
idle	
FOOD	DRINKS
• BBQ Chicken Maryland and Steamed Butter Rice and Curry Gravy	• Iced lemon tea

Table B	
idle	
FOOD	DRINKS
• Grilled Lamb Chop in Black Pepper Sauce with Baked Potato Salad	• Iced Fresh Pineapple Juice

Table C	
FOOD	DRINKS

Table D	
idle	
FOOD	DRINKS

Table E	
idle	
FOOD	DRINKS

Information	

copy-right by fuzisharif@gmail.com

References:

1. <https://appinventor.mit.edu/>