UNIVERSITY OF MICHIGAN-DEARBORN

Department of Computer and Information Science

CIS 571 Web Services Project 2

BY

Siyu Yang

UMID: 76998080

Content

Link for my online presentation: https://youtu.be/WJ19Kmr0V-A

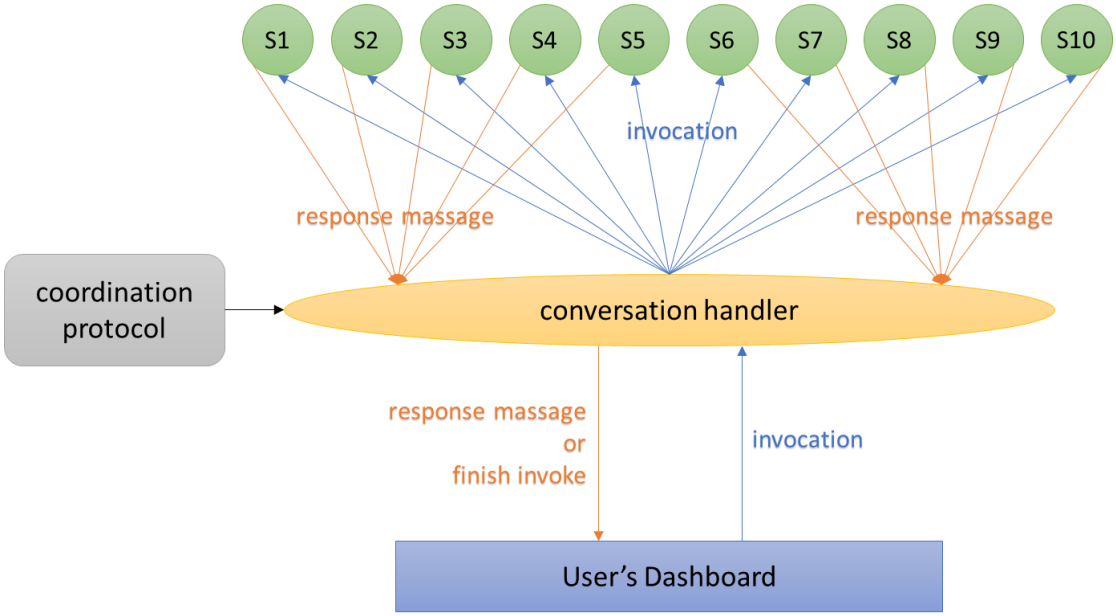
1. Introduction

1.1 Goal

The aim of this project is to design and implement a conversation handler for coordination protocols. This web application consists of 5 major tasks, they are:

- Implement 10 different REST Web services.
- Take a file with state diagram as the coordination protocol.
- Implement a Web interface.
- Invoke the services that the user will be able to invoke from current state until the end of the coordination protocol.
- Show the massage after invoking on the User's Dashboard.

1.2 Architecture



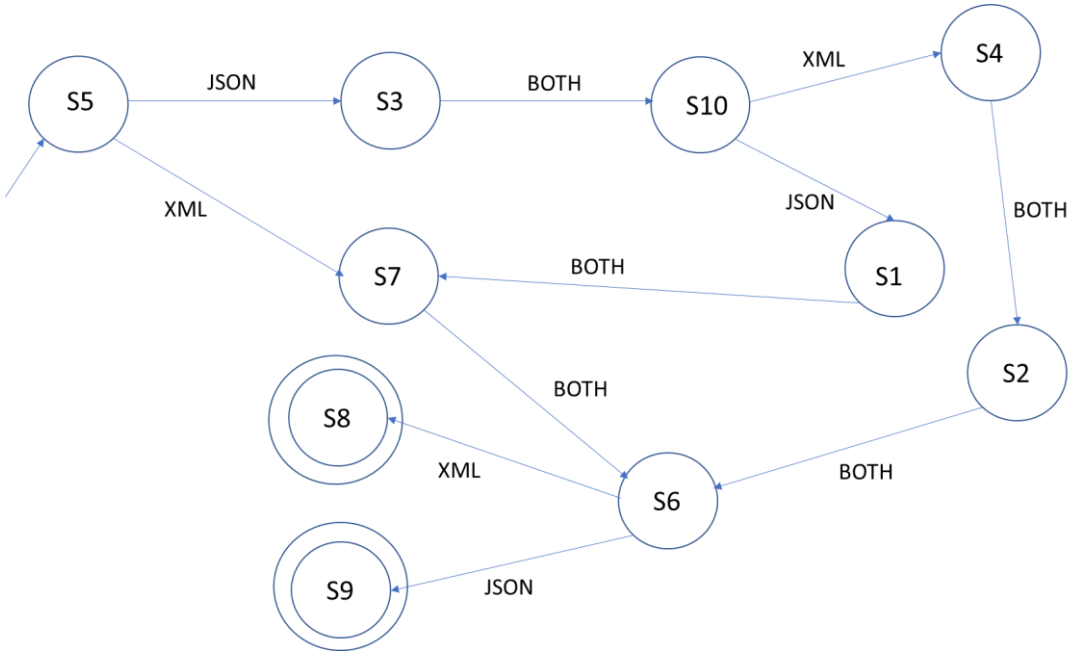
1.3 Technologies

- Back-end: Python Flask Web Framework
- Front-end: HTML, CSS, JavaScript

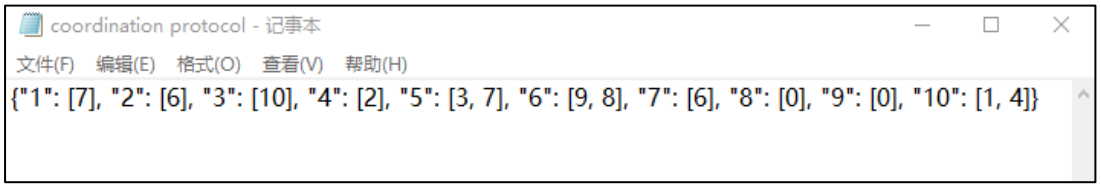
2. Design

2.1 State Diagram

I choose to use a plain file with JSON data format to describe the state diagram and the coordination protocol. Take the following diagram as an example:



The plain file for this state diagram is:

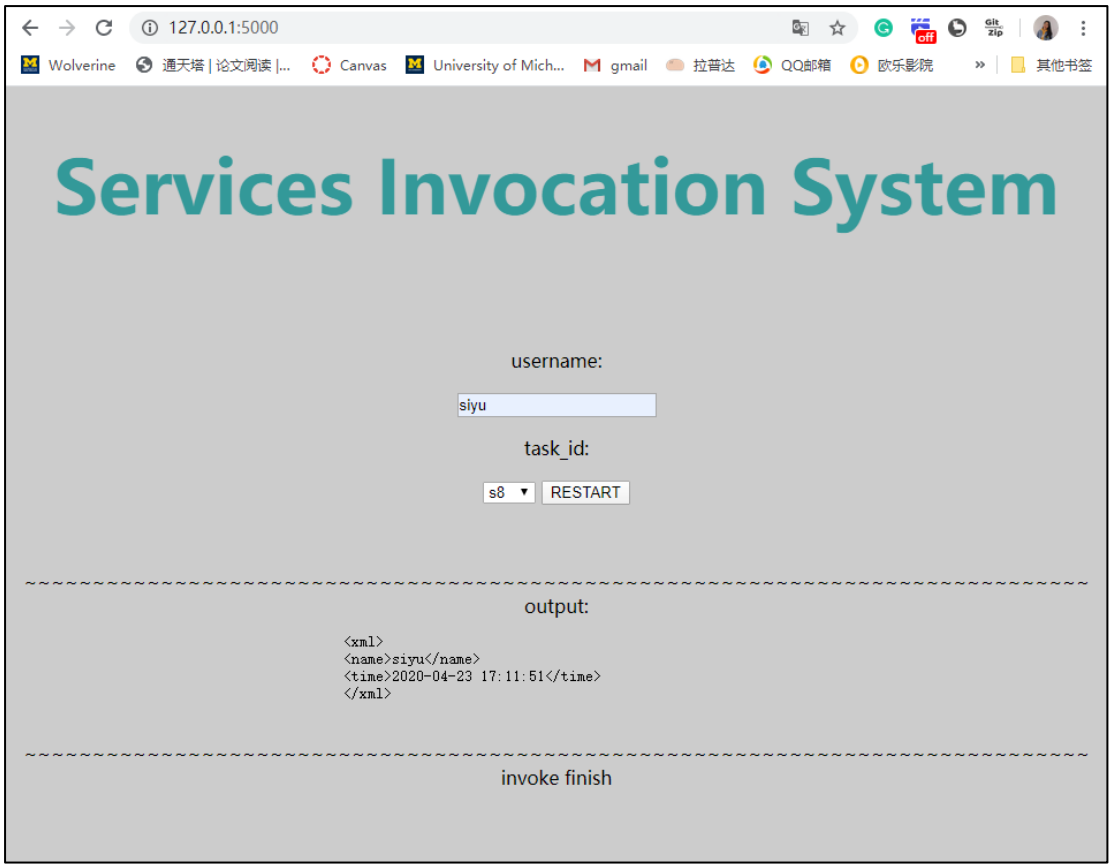


The meaning is: S1 will invoke S7 no matter which data format massage; S2 will invoke S6 no matter which data format massage; S3 will invoke S10 no matter which data format massage; S4 will invoke S2 no matter which data format massage; S5 will invoke S3 for JSON data format massage and S7 for XML data format massage; S6 will invoke S9 for JSON data format massage and S8 for XML data format massage; S7 will invoke S6 no matter which data format massage; S8 will result in the end of the invocation for XML data format massage; S9 will result in the end of the invocation for JSON data format massage; S10 will invoke S1 for JSON data format massage and S4 for XML data format massage.

2.2 Dashboard

2.2.1 HTML/CSS Part

Here is my own design for dashboard. I put components neatly and use the green and gray color to make the whole interface simple and clean. The input part and output part are separated by a line. I also design several options with task_id for users which will avoid the input errors.



2.2.2 JavaScript Part

I implement 6 functions in JavaScript. They are used to:

- Submit username and start service id

```
function tpformsubmit() {
    var username = $('input[name=username]').val();
    var next_invoke_func = $('#sel option:selected').val();
    url = '/index?nextfunc=' + next_invoke_func + '&username=' + username;
    $.ajax({
        url: url,
        method: 'GET',
```

```
            success: function (res) {
                if (res.errno == 1001) {
                    // finish invoke hint restart
                    $('.restart_button').show();
                    $('.sub_button').hide();
                    return
                }
                $('.recv_content').html('');
                $('.output').attr('style', 'display:block;text-align:center;');
                console.log(res);
                if (username.match('^[a-mA-M]')) {
                    var data = JSON.stringify(res, null, 2);
                    $('.recv_content').html(data);
                } else {
                    var xml_str = new XMLSerializer().serializeToString(res);
                    $('.recv_content').text(xml_str);
                }
                set_next_invoke();
            },
            error: function (res) {
                $('.recv_content').html(res)
            }
        })
}
```

- Determine the format of massage is JSON or XML

```
function isJSON(str) {
    if (typeof str == 'string') {
        try {
            var obj = JSON.parse(str);
            if (typeof obj == 'object' && obj) {
                console.log('is JSON');
                return true;
            } else {
                return false;
            }
        } catch (e) {
```

```
                console.log('error：' + str + '!!!' + e);
                return false;
            }
        }
}
```

- Get Cookie

```
function getCookie(name) {
    var strcookie = document.cookie;// get cookie str
    var arrcookie = strcookie.split("; ");// split cookie str
    for (var i = 0; i < arrcookie.length; i++) {
        var arr = arrcookie[i].split("=");
        if (arr[0] == name) {
            return arr[1];
        }
    }
    return "";
}
```

- Set next invoke service id

```
function set_next_invoke() {
    next_invoke = getCookie('next');
    if (next_invoke == '') {
        $('.sub_button').val('start');
    } else {
        if (next_invoke == 's0') {
            $('.sub_button').hide();
            $('.restart_button').show();
            $('.invoke_finish').attr('style', 'display:block');
            return
        }
        $('.sub_button').val('next');
        $('.invoke_finish').hide();
        $("#sel").val(next_invoke);
    }
}
```

- Remove cookie

```
function removeCookie(name) {
    var d = new Date();
    d.setTime(d.getTime() - 10000);
    document.cookie = name + '=1; expires=' + d.toGMTString();
}
```

- Restart

```
function restart() {
    $('.invoke_finish').hide();
    removeCookie('next');
    $("#sel").val('s1');
    $('input[name=username]').val('');
    $('.output').hide();
    $('.restart_button').hide();
    $('.sub_button').val('start');
    $('.sub_button').show();
}
```

3. APIs

3.1 Service API

3.1.1 Design

The service APIs are all the same. I will describe the S1 as an example.

The request is the username and the current state. The massage format is JSON or XML. If the username's starts with A-M, the response is a simple JSON message stating: user's name, invocation date, and invocation time. If the username's starts with N-Z, the response is a simple XML message, as:

Request: { "username": "kate" "state": "s5" }	Response: { "name": "kate" "time": "2020-04-23 17:32:13" }

Request:
{
"username": "siyu"
"state": "s5"
}
	Response:
	<xml>
	<name>kate</name>
	< time >2020-04-23 17:32:13</ time >
	</xml>

3.1.2 Implement

Service API:

```
def s1(username, map_dict):
    now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    return_dict = {
        "name": username,
        "time": now
    }
    if re.match(r'[a-mA-M]', username[0]):
        response = make_response(jsonify(return_dict))
        next_invoke = map_dict.get('json', '')
    else:
        xml = []
        for k in return_dict.keys():
            v = return_dict.get(k)
            if k == 'detail' and not v.startswith('<![CDATA['):
                v = '<![CDATA[{}]]>'.format(v)
            xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
            xml.append('\n')

        response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
        response.headers["Content-Type"] = "text/xml; charset=utf-8"
        next_invoke = map_dict.get('xml', '')
    response.set_cookie(key='next', value=next_invoke)
    return response
```

I import the time package to get the invoke date and time, the re package to match the username by regular expression. The output of name and time will be saved in a dictionary first. If *re.match(r'[a-mA-M]', username[0])*, the dictionary will transfer into JSON format and the response is the JSON massage. Else, I will create a list to store the keys and values of the dictionary in XML format. The response headers are assigned as "text/xml". Also, I use cookie to store the next service's id. There is a mapping dictionary which can be used to calculate the id of next service. The key of the cookie is 'next' and value is the next service's id. At last, each service will return the response.

3.2 Conversation Handler API

3.2.1 Design

The conversation handler API has HTTP GET and HTTP POST two protocols. It will get the input and cookie from user's dashboard. If it is the first invocation, there will be no cookie and the response will invoke next service with the invoke map. If the task_id input is 's0', which means the invocation will be finished, the response will be 'error' with an error massage. And the cookie will be deleted. For other situations, the response will invoke next service with the invoke map and the next service id from cookie.

As for the invoke map, it is generated from the plain file with state diagram. I create a function to transfer the txt file into a map like following:

```
def get_map(file_name):
    map_dict = {}
    try:
        with open(file_name, 'r')as f:
            content = ''
            while True:
                read_content = f.read(1024)
                if read_content:
                    content += read_content
                else:
                    break
        text_dict = eval(content)
        if type(text_dict == dict):
            for key, value in text_dict.items():
                map_dict['s' + str(key)] = {}
                map_dict['s' + str(key)]['this'] = 's' + str(key)
                map_dict['s' + str(key)]['json'] = 's' + str(value[0])
                if len(value) == 1:
                    map_dict['s' + str(key)]['xml'] = 's' + str(value[0])
                else:
                    map_dict['s' + str(key)]['xml'] = 's' + str(value[1])
    except Exception as e:
```

```
        print(e)
    else:
        return map_dict
```

The input of this function is a filename, the output is the invoke map. First, read the file and store all information in s string called content. After that, I use the eval() function to get the value from content and store them in a text dictionary. Then, read the text dictionary and store the values into map dictionary in the expected format. Here is the expected map dictionary from this function:

```
{
    's1': {'this': s1, 'json': 's7', 'xml': 's7'},
    's2': {'this': s2, 'json': 's6', 'xml': 's6'},
    's3': {'this': s3, 'json': 's10', 'xml': 's10'},
    's4': {'this': s4, 'json': 's2', 'xml': 's2'},
    's5': {'this': s5, 'json': 's3', 'xml': 's7'},
    's6': {'this': s6, 'json': 's8', 'xml': 's9'},
    's7': {'this': s7, 'json': 's6', 'xml': 's6'},
    's8': {'this': s8, 'json': 's8', 'xml': 's8'},
    's9': {'this': s9, 'json': 's9', 'xml': 's9'},
    's10': {'this': s10, 'json': 's1', 'xml': 's4'},
}
```

3.2.2 Implement

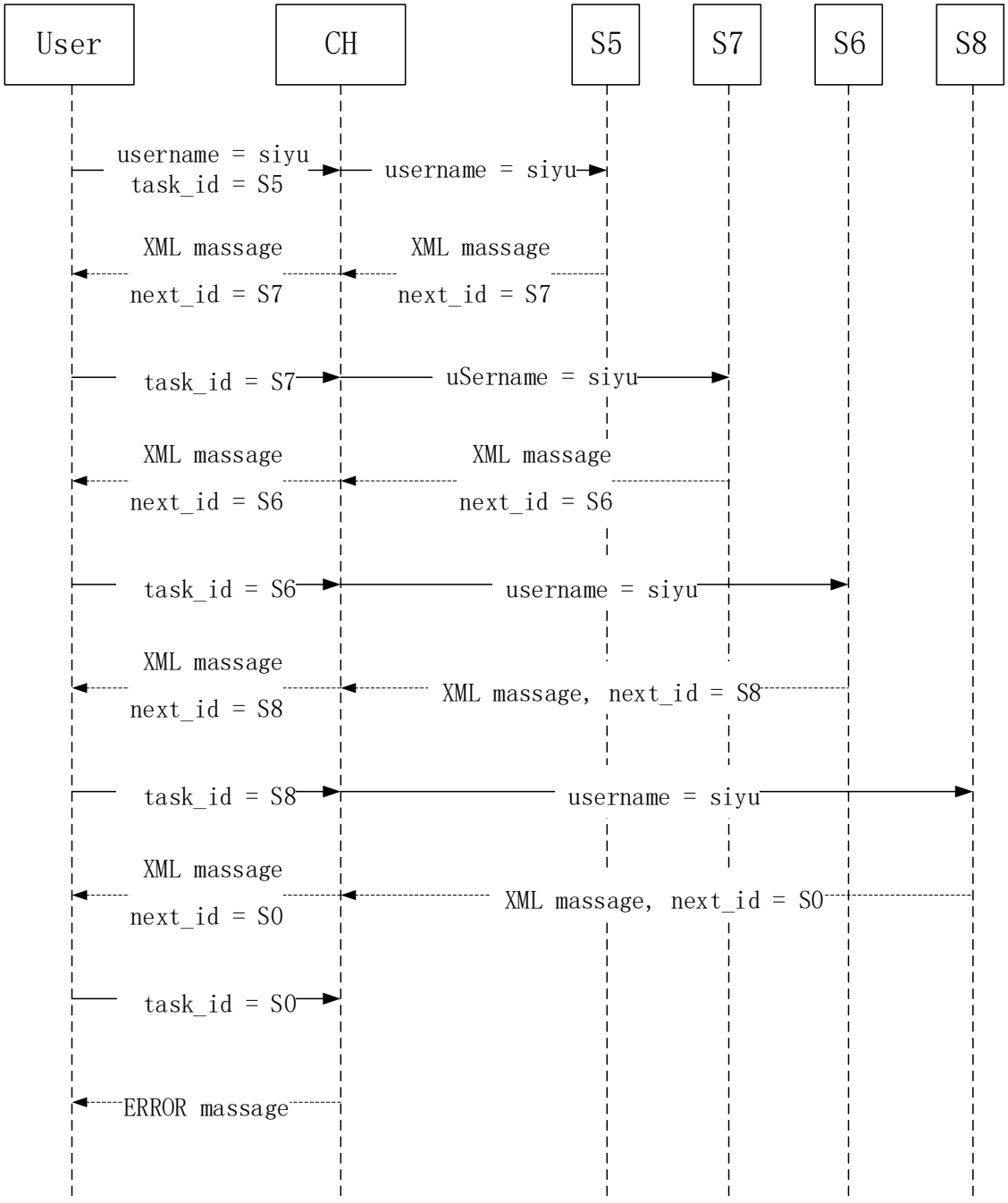
Conversation Handler API:

```
@app.route('/index', methods=["GET", "POST"])
def conversation_handler():
    username = request.args['username']
    nextfunc = request.args.get('nextfunc')    # user select
    next = request.cookies.get("next", '')    # browser saved
    invoke_map = get_map('./invoke_logic.txt')
    if next == '':
        response = eval(invoke_map.get(nextfunc)['this'])(username,
map_dict=invoke_map.get(nextfunc))
    elif next == 's0':
        response = make_response(jsonify({'errno': 1001, 'errmsg': "invoke
finish"}))
        response.delete_cookie('next')
    else:
        response = eval(invoke_map.get(next)['this'])(username,
```

```
map_dict=invoke_map[next])
    return response
```

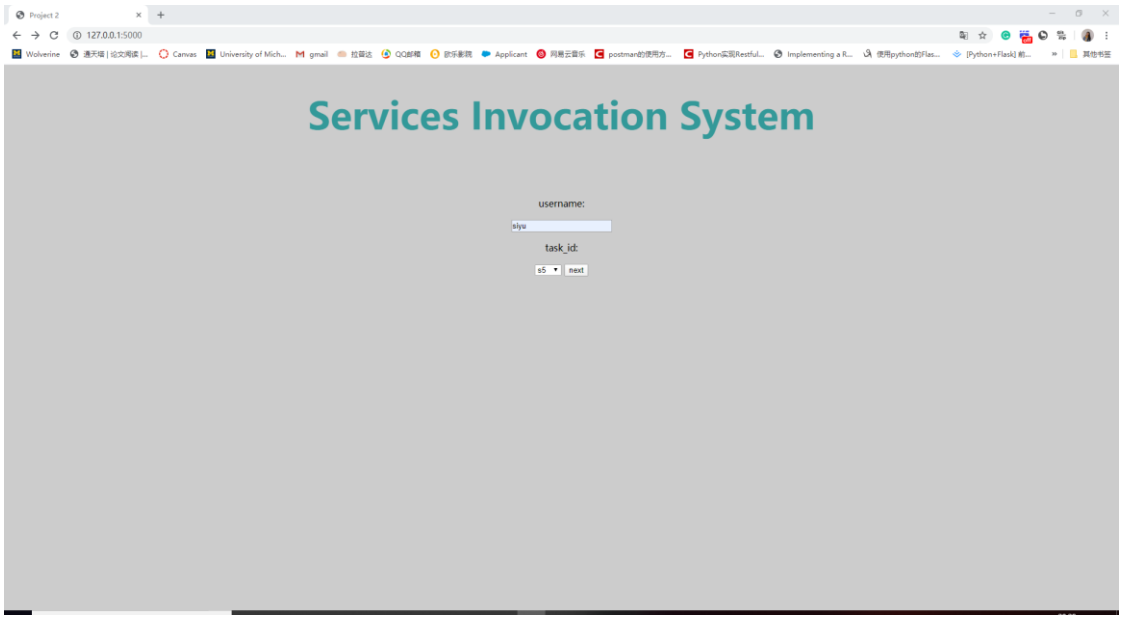
In this piece of code, *nextfunc* is the service_id user selected, *next* is the service_id browser saved.

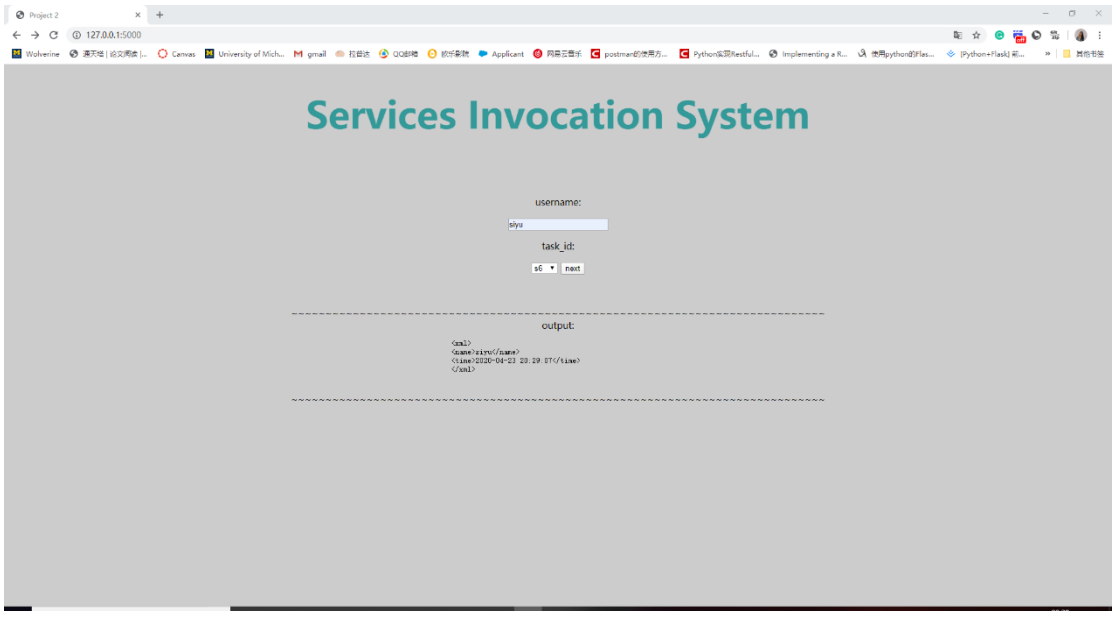
4. UML Sequence Diagram

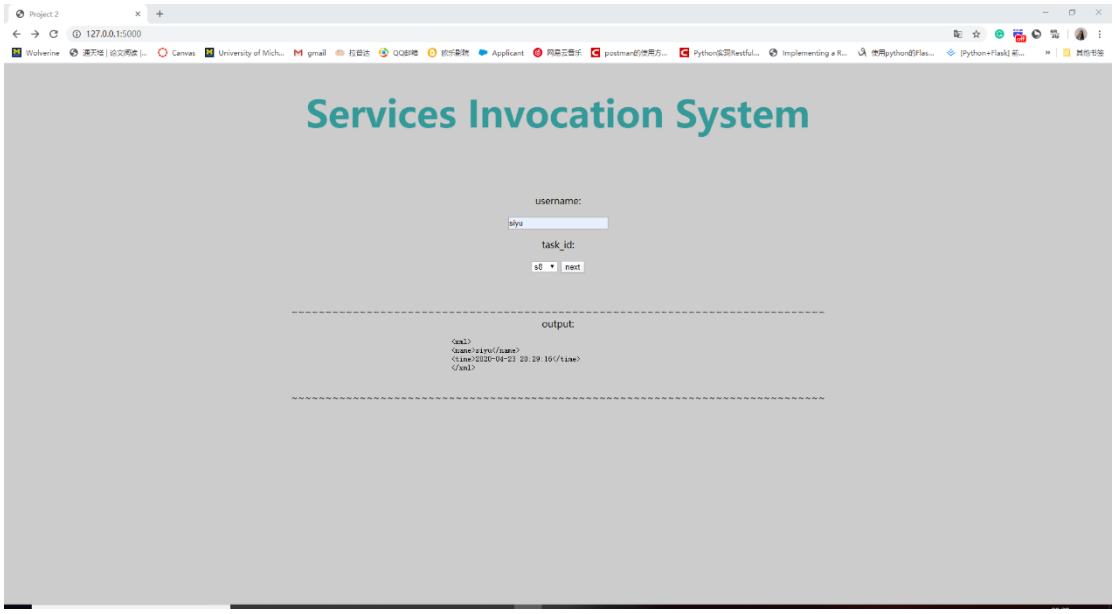


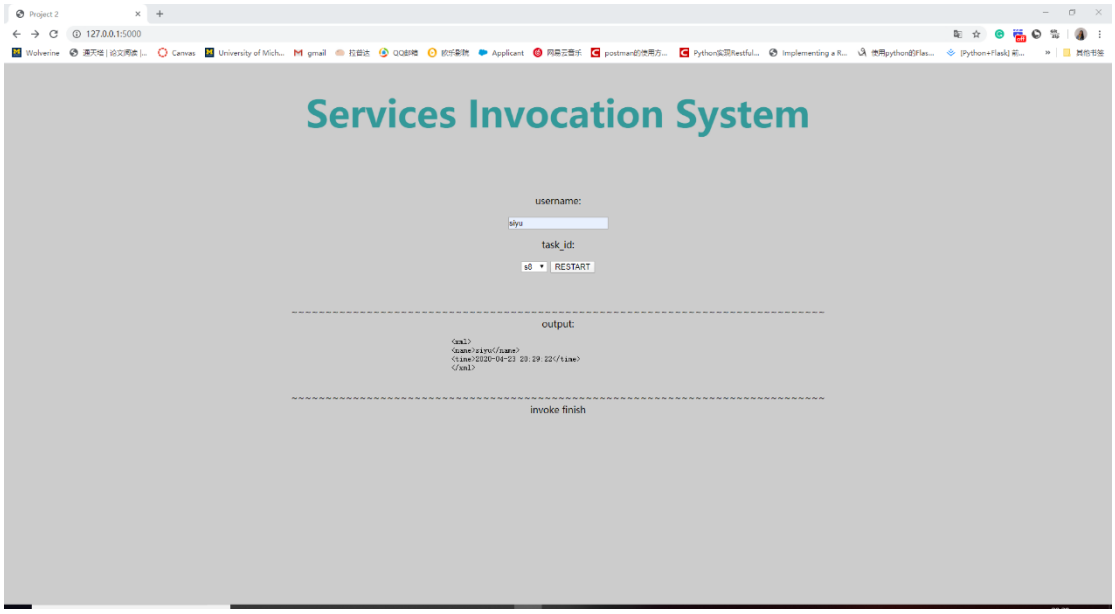
5. Screenshots

Screenshots of that same scenario with UML sequence diagram. The username is *siyu* and the start state is *S5*. The following screenshots are: the initial state, after invoke S5, after invoke S7, after invoke S6, after invoke S8(with error massage).









6. Appendix 1: Source code of the CH

```
def get_map(file_name):
    map_dict = {}
    try:
        with open(file_name,'r')as f:
            content = ''
            while True:
                read_content = f.read(1024)
```

```
                    if read_content:
                        content += read_content
                    else:
                        break
            text_dict = eval(content)
            if type(text_dict == dict):
                for key, value in text_dict.items():
                    map_dict['s' + str(key)] = {}
                    map_dict['s' + str(key)]['this'] = 's' + str(key)
                    map_dict['s' + str(key)]['json'] = 's' + str(value[0])
                    if len(value) == 1:
                        map_dict['s' + str(key)]['xml'] = 's' + str(value[0])
                    else:
                        map_dict['s' + str(key)]['xml'] = 's' + str(value[1])
        except Exception as e:
            print(e)
        else:
            return map_dict

@app.route('/index', methods=["GET", "POST"])
def conversation_handler():
    username = request.args['username']
    nextfunc = request.args.get('nextfunc')    # user select
    next = request.cookies.get("next", '')    # browser saved
    invoke_map = get_map('./coordination_protocol.txt')
    if next == '':
        response = eval(invoke_map.get(nextfunc)['this'])(username,
map_dict=invoke_map.get(nextfunc))
    elif next == 's0':
        response = make_response(jsonify({'errno': 1001, 'errmsg': "invoke
finish"}))
        response.delete_cookie('next')
    else:
        response = eval(invoke_map.get(next)['this'])(username,
map_dict=invoke_map[next])
    return response
```

7. Appendix 2: Source code of the 10 services

```
def s1(username, map_dict):
    now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    return_dict = {
        "name": username,
        "time": now
    }
    if re.match(r'[a-mA-M]', username[0]):
        response = make_response(jsonify(return_dict))
        next_invoke = map_dict.get('json', '')
    else:
        xml = []
        for k in return_dict.keys():
            v = return_dict.get(k)
            if k == 'detail' and not v.startswith('<![CDATA['):
                v = '<![CDATA[{}]]>'.format(v)
            xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
            xml.append('\n')

        response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
        response.headers["Content-Type"] = "text/xml; charset=utf-8"
        next_invoke = map_dict.get('xml', '')
    response.set_cookie(key='next', value=next_invoke)
    return response


def s2(username, map_dict):
    now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    if re.match(r'[a-mA-M]', username[0]):
        return_dict = {
            "name": username,
            "time": now
        }
```

```
            response = make_response(jsonify(return_dict))
            next_invoke = map_dict.get('json', '')
        else:
            return_dict = {
                "name": username,
                "time": now
            }

            xml = []
            for k in return_dict.keys():
                v = return_dict.get(k)
                if k == 'detail' and not v.startswith('<![CDATA['):
                    v = '<![CDATA[{}]]>'.format(v)
                xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
                xml.append('\n')
            response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
            response.headers["Content-Type"] = "text/xml; charset=utf-8"
            next_invoke = map_dict.get('xml', '')
        response.set_cookie(key='next', value=next_invoke)
        return response


def s3(username, map_dict):
        now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        if re.match(r'[a-mA-M]', username[0]):
            return_dict = {
                "name": username,
                "time": now
            }
            response = make_response(jsonify(return_dict))
            next_invoke = map_dict.get('json', '')
        else:
            return_dict = {
                "name": username,
                "time": now
```

```
            }

            xml = []
            for k in return_dict.keys():
                v = return_dict.get(k)
                if k == 'detail' and not v.startswith('<![CDATA['):
                    v = '<![CDATA[{}]]>'.format(v)
                xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
                xml.append('\n')
            response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
            response.headers["Content-Type"] = "text/xml; charset=utf-8"
            next_invoke = map_dict.get('xml', '')
        response.set_cookie(key='next', value=next_invoke)
        return response


def s4(username, map_dict):
        now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        if re.match(r'[a-mA-M]', username[0]):
            return_dict = {
                "name": username,
                "time": now
            }
            response = make_response(jsonify(return_dict))
            next_invoke = map_dict.get('json', '')
        else:
            return_dict = {
                "name": username,
                "time": now
            }

            xml = []
            for k in return_dict.keys():
                v = return_dict.get(k)
                if k == 'detail' and not v.startswith('<![CDATA['):
```

```
                    v = '<![CDATA[{}]]>'.format(v)
                xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
                xml.append('\n')
        response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
        response.headers["Content-Type"] = "text/xml; charset=utf-8"
        next_invoke = map_dict.get('xml', '')
    response.set_cookie(key='next', value=next_invoke)
    return response


def s5(username, map_dict):
    now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    if re.match(r'[a-mA-M]', username[0]):
        return_dict = {
            "name": username,
            "time": now
        }
        response = make_response(jsonify(return_dict))
        next_invoke = map_dict.get('json', '')
    else:
        return_dict = {
            "name": username,
            "time": now
        }

        xml = []
        for k in return_dict.keys():
            v = return_dict.get(k)
            if k == 'detail' and not v.startswith('<![CDATA['):
                v = '<![CDATA[{}]]>'.format(v)
            xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
            xml.append('\n')
        response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
        response.headers["Content-Type"] = "text/xml; charset=utf-8"
```

```
            next_invoke = map_dict.get('xml', '')
        response.set_cookie(key='next', value=next_invoke)
        return response


def s6(username, map_dict):
        now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        if re.match(r'[a-mA-M]', username[0]):
            return_dict = {
                "name": username,
                "time": now
            }
            response = make_response(jsonify(return_dict))
            next_invoke = map_dict.get('json', '')
        else:
            return_dict = {
                "name": username,
                "time": now
            }

            xml = []
            for k in return_dict.keys():
                v = return_dict.get(k)
                if k == 'detail' and not v.startswith('<![CDATA['):
                    v = '<![CDATA[{}]]>'.format(v)
                xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
                xml.append('\n')
            response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
            response.headers["Content-Type"] = "text/xml; charset=utf-8"
            next_invoke = map_dict.get('xml', '')
        response.set_cookie(key='next', value=next_invoke)
        return response


def s7(username, map_dict):
```

```
    now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    if re.match(r'[a-mA-M]', username[0]):
        return_dict = {
            "name": username,
            "time": now
        }
        response = make_response(jsonify(return_dict))
        next_invoke = map_dict.get('json', '')
    else:
        return_dict = {
            "name": username,
            "time": now
        }

        xml = []
        for k in return_dict.keys():
            v = return_dict.get(k)
            if k == 'detail' and not v.startswith('<![CDATA['):
                v = '<![CDATA[{}]]>'.format(v)
            xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
            xml.append('\n')
        response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
        response.headers["Content-Type"] = "text/xml; charset=utf-8"
        next_invoke = map_dict.get('xml', '')
    response.set_cookie(key='next', value=next_invoke)
    return response


def s8(username, map_dict):
    now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    if re.match(r'[a-mA-M]', username[0]):
        return_dict = {
            "name": username,
            "time": now
        }
```

```
            response = make_response(jsonify(return_dict))
            next_invoke = map_dict.get('json', '')
        else:
            return_dict = {
                "name": username,
                "time": now
            }

            xml = []
            for k in return_dict.keys():
                v = return_dict.get(k)
                if k == 'detail' and not v.startswith('<![CDATA['):
                    v = '<![CDATA[{}]]>'.format(v)
                xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
                xml.append('\n')
            response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
            response.headers["Content-Type"] = "text/xml; charset=utf-8"
            next_invoke = map_dict.get('xml', '')
        response.set_cookie(key='next', value=next_invoke)
        return response


def s9(username, map_dict):
        now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        if re.match(r'[a-mA-M]', username[0]):
            return_dict = {
                "name": username,
                "time": now
            }
            response = make_response(jsonify(return_dict))
            next_invoke = map_dict.get('json', '')
        else:
            return_dict = {
                "name": username,
                "time": now
```

```
            }

            xml = []
            for k in return_dict.keys():
                v = return_dict.get(k)
                if k == 'detail' and not v.startswith('<![CDATA['):
                    v = '<![CDATA[{}]]>'.format(v)
                xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
                xml.append('\n')
            response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
            response.headers["Content-Type"] = "text/xml; charset=utf-8"
            next_invoke = map_dict.get('xml', '')
        response.set_cookie(key='next', value=next_invoke)
        return response


def s10(username, map_dict):
        now = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        if re.match(r'[a-mA-M]', username[0]):
            return_dict = {
                "name": username,
                "time": now
            }
            response = make_response(jsonify(return_dict))
            next_invoke = map_dict.get('json', '')
        else:
            return_dict = {
                "name": username,
                "time": now
            }

            xml = []
            for k in return_dict.keys():
                v = return_dict.get(k)
                if k == 'detail' and not v.startswith('<![CDATA['):
```

```
                    v = '<![CDATA[{}]]>'.format(v)
                xml.append('<{key}>{value}</{key}>'.format(key=k, value=v))
                xml.append('\n')
        response = Response(response='<xml>\n{}</xml>'.format(''.join(xml)),
status=200, mimetype="application/xml")
        response.headers["Content-Type"] = "text/xml; charset=utf-8"
        next_invoke = map_dict.get('xml', '')
    response.set_cookie(key='next', value=next_invoke)
    return response
```