Contents

Abstract

OnlineCI is an online compiler, interpreter for C/C++ and Python languages. OnlineCI allows a person to select language from any one of Python, C and C++. Once the program is typed in the input field and submitted, the results are returned in the output field. If output is not received but error has occurred, that will be visible in the remarks field.

Introduction

OnlineCI is a very handy webapp for programmers. It uses WSGI (Web Server Gateway Interface) for hosting the backend which is developed in flask framework.

Flask is a web development framework developed in Python that can be used to create backends of web applications and connect them to front-end HTML or Javascript code. Flask makes it easy to interact with front-end code of javascript and HTML by introducing python variables from Flask backend to front-end code. Thus, the processing can be done in Flask and the results can be returned to the front-end HTML site.

This project is deployed in **AWS** (Amazon Web Services). The environment set up was done using **AWS Elastic Beanstalk**. It is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS. You can simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring. At the same time, you retain full control over the AWS resources powering your application and can access the underlying resources at any time.

A **Github** repository for version control purposes has also been created. Github is an online version control software and a web extension of **git**.

Problem Statement

Programmers deal with many languages and multiple platforms, thus they need to install bulky softwares or meta-softwares to test or deploy their implementation. This is very cumbersome and time consuming. So, a platform is needed where developers and programmers can test their code and implementation quickly with the click of a button.

Requirements

Hardware Requirements (Server):

- Infrastructure managed by AWS Elastic Beanstalk

Software Requirements (Server):

- Compilers/Interpreters for C, C++ and Python.
- Flask Framework

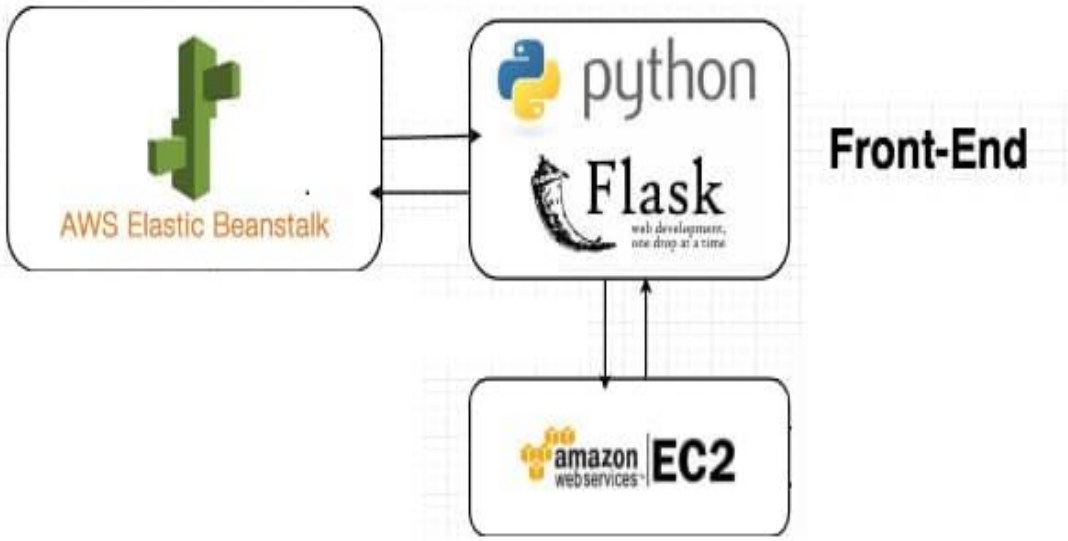
Hardware Requirements (Client):

- 500 MB RAM
- 5 GB Hard Disk space
- Keyboard
- Mouse
- Monitor/ Display

Software Requirements (Client):

- Web Browser and its requirements

Flow Chart


AWS Elastic Beanstalk
python
Flask
web development, one drop at a time
Front-End
amazon web services
EC2

Implementation

Flask Backend (applications.py):

```
from flask import *
import os
application = Flask(__name__)

languages = ['Python', 'C', 'C++']

@application.route('/', methods=['POST','GET'])
def compute():
    if request.method == 'POST':
        result = request.form
        result_dict = dict(result.items())
        lang = result_dict['languages']
        # Validate if input field is empty or not
        # Check when compilation fails
        if lang == "Python":

open("code.py",'w').write(result_dict['prog'])

open("input.file",'w').write(result_dict['input'])
```

```
            os.system("python  code.py  <  input.file  1>
output.file 2> remarks.file")
            remark = open("remarks.file",'r').read()
            outputs = open("output.file",'r').read()
            os.system("rm code.py *.file")
            return
render_template("template.html",languages=languages,curr
ent_lang=lang,prog=result_dict['prog'],input=result_dict
['input'],remarks=remark,output=outputs)
        elif lang == "C":

open("code.c",'w').write(result_dict['prog'])
            os.system("gcc code.c -o code 1> output.file
2> remarks.file")

open("input.file",'w').write(result_dict['input'])
            os.system("./code    <    input.file    1>>
output.file 2>> remarks.file")
            remark = open("remarks.file",'r').read()
            outputs = open("output.file",'r').read()
            os.system("rm code* *.file")
            return
render_template("template.html",languages=languages,curr
ent_lang=lang,prog=result_dict['prog'],input=result_dict
['input'],remarks=remark,output=outputs)
        elif lang == "C++":
```

```
open("code.cpp",'w').write(result_dict['prog'])
            os.system("g++    code.cpp    -o    code    1>
output.file 2> remarks.file")

open("input.file",'w').write(result_dict['input'])
            os.system("./code    <    input.file    1>>
output.file 2>> remarks.file")
            remark = open("remarks.file",'r').read()
            outputs = open("output.file",'r').read()
            os.system("rm code* *.file")
            return
render_template("template.html",languages=languages,curr
ent_lang=lang,prog=result_dict['prog'],input=result_dict
['input'],remarks=remark,output=outputs)
        else:
            print("You are one kind of a tester")
        return
render_template('template.html',languages=languages,curr
ent_lang='',prog='',input='',remarks='',output='')
    elif request.method == 'GET':
        return
render_template('template.html',languages=languages,curr
ent_lang='',prog='',input='',remarks='',output='')
if __name__ == '__main__':
    application.run(debug = False)
```

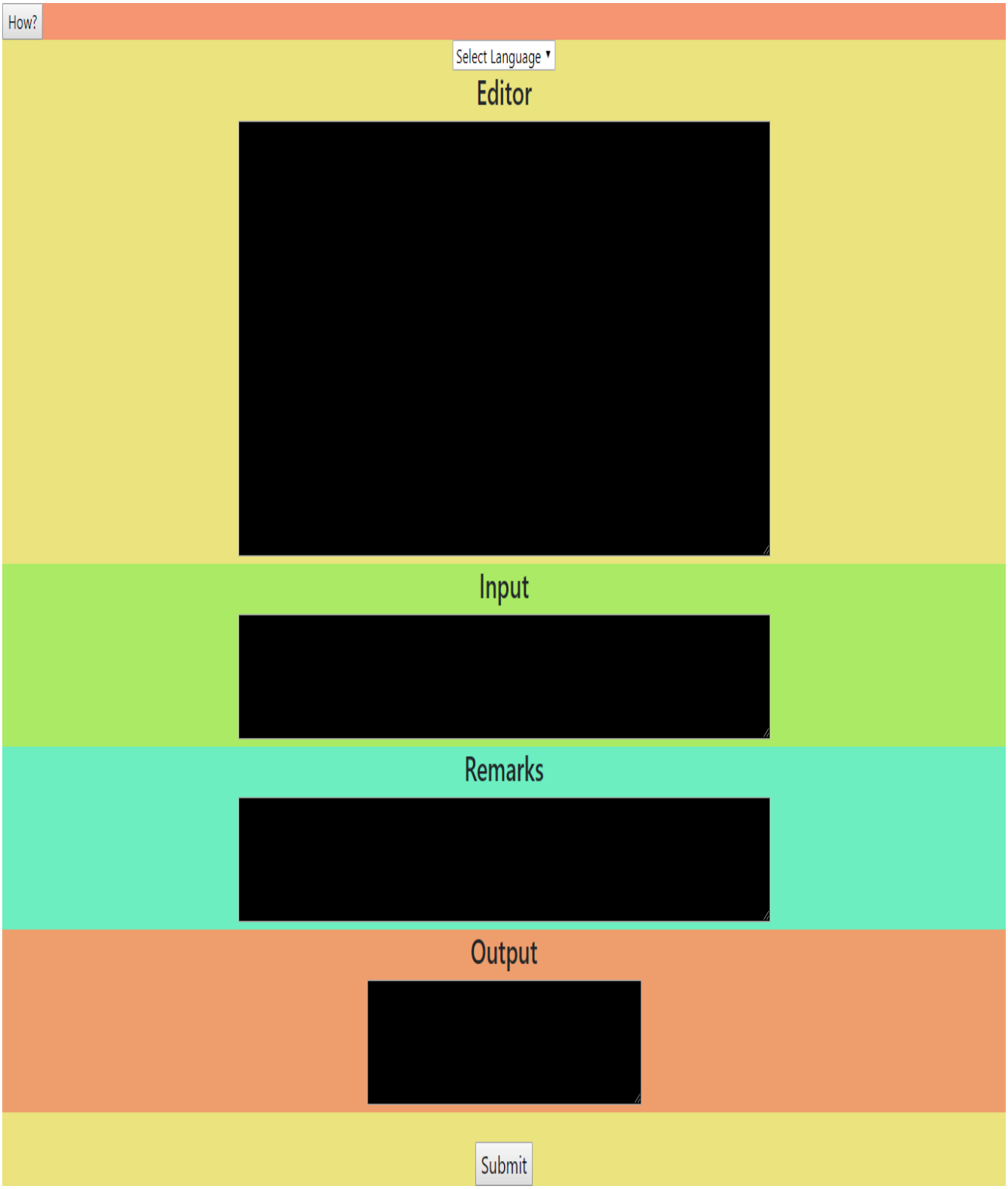
HTML/ Javascript Front-end:

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>TextEditor</title>
    <style>
        textarea{ background-color: black; color: white;
}
    </style>
    <link                                          href="{{
url_for('static',filename='css/bootstrap.min.css')    }}"
rel="stylesheet">
    <script>
        function display() {
            document.getElementById("text").innerHTML   =
"Type Code in the textbox and hit compile! to see the
results";
        }
    </script>
    </head>
    <body style="background-color: #F59571;">
        <div id="tutorial">
```

```
            <input       type="button"       value="How?"
onclick="display()" />
            <div id="text"></div>
        </div>
        <center><form      id="userinput"      action="/"
method="POST" style="background-color: #EAE37E;">
            <select name ="languages">
                        <option      selected="selected"
disabled="disabled">Select Language</option>
        {% for language in languages %}
        <option value="{{language}}">{{language}}
</option>
                {% endfor %}
            </select>
        {% if current_lang != '' %}
        <script>
document.getElementsByName('languages')[0].value       =
"{{current_lang}}"; </script>
        {% endif %}
        <div id="inputcode">
            <h3>Editor</h3>
            <textarea  rows="50"  cols="100"  name="prog"
onkeydown="if(event.keyCode===9){var
v=this.value,s=this.selectionStart,e=this.selectionEnd;t
his.value=v.substring(0,
```

```
s)+'\t'+v.substring(e);this.selectionStart=this.selectio
nEnd=s+1;return false;}">{{ prog }}</textarea>
        </div>
        <div    id="inputs"    style="background-color:
#AAEA64;">
            <h3>Input</h3>
            <textarea        rows="4"        cols="100"
name="input">{{ input }}</textarea><br>
        </div>
        <div    id="remarks"    style="background-color:
#6CEEC0">
            <h3>Remarks</h3>
            <textarea  rows="4"  cols="100"  name="remark"
readonly>{{ remarks }}</textarea><br>
        </div><div  id="output"  style="background-color:
#EE9E6C;">
            <h3>Output</h3>
            <textarea  rows="4"  cols="50"  name="output"
readonly>{{ output }}</textarea>
        <br></div><br>
        <input type="submit" value="Submit" style="font-
size: 20px;">
        </form></center>
    </body>
</html>
```

Output Screenshots



How?

Python

Editor

```
# Function for nth fibonacci number - Space Optimisataion
# Taking 1st two fibonacci numbers as 0 and 1

def fibonacci(n):
    a = 0
    b = 1
    if n < 0:
        print("Incorrect input")
    elif n == 0:
        return a
    elif n == 1:
        return b
    else:
        for i in range(2,n):
            c = a + b
            a = b
            b = c
        return b

# Driver Program

n = int(input(""))
print(fibonacci(n))
```

Input

```
10
```

Remarks

Output

```
34
```

Submit

Conclusion

OnlineCI can be extended in the future to support other languages as well. **AWS Elastic Beanstalk** was used for deployment which manages all the infrastructure and resources for the webapp all by itself. **Flask** framework used was specially designed for making SPA (Single Page Applications). This can be changed to **Django** framework developed again in Python but is more suitable for creating Dynamic web applications.

References

- Flask Framework – www.tutorialspoint.com
- AWS Elastic Beanstalk – docs.aws.amazon.com
- Miscellaneous Issues during dev – www.stackoverflow.com