# A G.U.I. WITH HTML & CSS

## Introduction

Our project intends to use **Java** to program our chatbot logic. Our decision was heavily influenced by its type-safety, object-oriented paradigm, predictability and a collective comfort with using Java. However, this reduces the available options for building a **GUI(graphical user interface)** due to its compatibility with modern options. Indeed, Java is an old and proven language evolving to meet the demands of our technological advances while remaining true to its philosophy.

**HTML and CSS** drive the modern web. HTML serves as the main markup language while CSS is used for styling the HTML. Both markup languages are supported by all major browsers in use right now. Additionally, they are both quite easy and straight-forward for beginners and relatively kind to developers who want to expand their knowledge on the same.

In this document, I will explore various ways to leverage the modern web while using Java which includes:

1. **Java Applets**
2. **Jakarta Server Pages** (JSP)
3. **HTML-JavaScript-PHP-Java** bridge.

I will also explore the advantages and disadvantages of each and the best option to use along with a code snippet as a proof of concept.

## A. JAVA APPLETS

- Arguably, the oldest of the three options available to use on the modern web.
- It predates **DOM(Document Object Model)** that serves as the standard for **HTML-CSS-JavaScript** interface for building applications.
- Before this standard, HTML had no way of interacting with a mouse and the keyboard as we know it.
- Java stepped in with Applets to bridge this gap which was quite a feat.
- HTML included an applet tag where you'd embed your compiled .jar file from Java. This file would be downloaded and executed by browsers in a sandbox.
- Currently, there is no browser support and thus not an option for use or discussion.

## B. JAKARTA SERVER PAGES (JSP)

- This is a server-side technology similar to **PHP** that is dynamic and platform-independent.
- It enables the generation of dynamic web pages based on popular markup languages such as **HTML & XML**.
- **JSPs** serve as the client-facing component whereas the real logic resides in the server run in a Java container called a **servlet.**
- By default, the code is compiled to Java bytecode with the servlet generating a stream of HTML on request based on the **JSP** definition-interop with Java in it.
- It allows you to write Java directly in the HTML within <%= {Java code goes here} %> .

```
<p>Counting to three:</p>
<% for (int i = 1; i < 4; i++) { %>
    <p>This number is <%= i %>.</p>
<% } %>
<p>OK.</p>
```

- The code above is a loop that prints one to three.
- As seen above, the syntax is quite similar to writing **PHP** within HTML.

**Advantages**

1. Purely server-side. The client only interacts with the code rendered on the browser and any business logic remains compiled and stored in the server.
2. Dynamically generates HTML based on user requests and performs any intermediate network requests for our chatbot without revealing it to the client.
3. Easy to learn as it involves embedding part of our Java code in HTML to trigger any functions written in Java.

**Disadvantages**

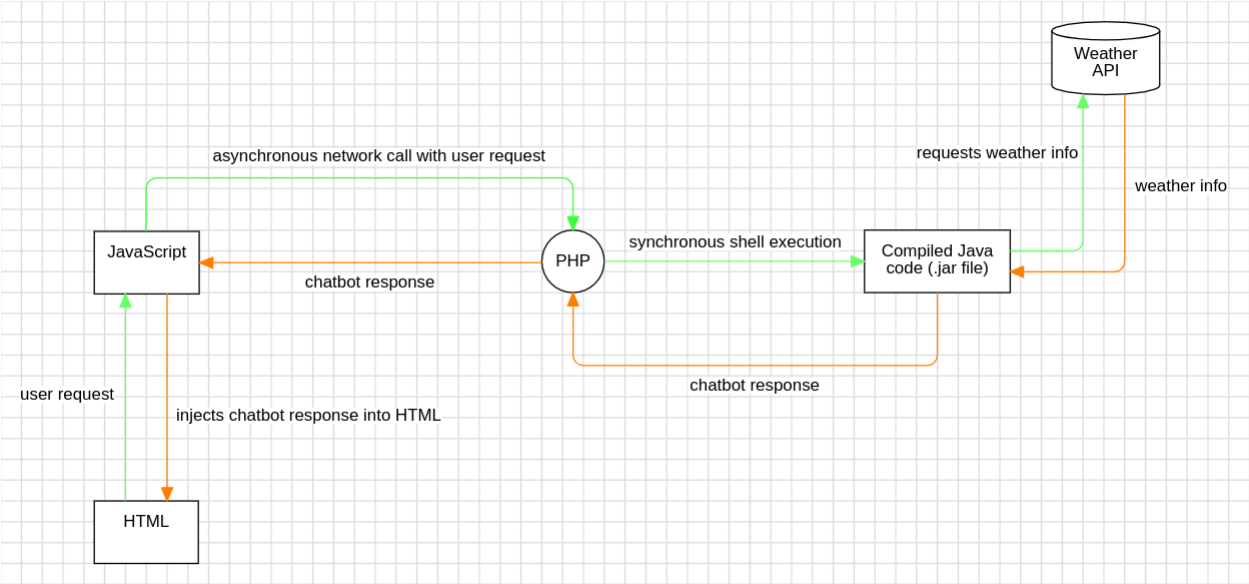
1. Requires additional Java Servlet logic together with the JSP code.
2. Code becomes hard to debug if an issue is encountered only in the Servlet logic.

### C. HTML-JavaScript-PHP-Java bridge

- This is a custom solution inspired by **JSP.**
- In this solution, we employ the separation of concerns strategy.
- We leverage existing client-side technologies and execute our chatbot logic on the server.

**Basic Steps**

1. Compile Java code with chatbot logic to a .jar file using Java Compiler and add it to the server.
2. Add the .jar file to the server with **PHP** file.
3. Make an asynchronous call using **JavaScript** to **PHP** on the server.
4. The compiled .jar file is synchronously executed by **PHP** via java shell command whose output is captured by **PHP.**
5. Once complete, **PHP** sends back the response.
6. The response is injected directly to the static **HTML** via **JavaScript**.




**Green indicates "request". Orange indicates "response"**

- As seen in the above flowchart, **PHP** is nothing but a connector. We leverage its capability to execute shell commands to execute Java via the java command on the server.
- Refer to below proof of concept on the same:

```java
public class Main {
    public static void main(String[] args) {
        // Print request
        System.out.println(args[0]);

        // Simple print line with welcome
        System.out.println("Hello and welcome! Here is a loop from the server!");

        // Loop 1 - 5 and print i
        for (int i = 1; i <= 5; i++) {
            System.out.println("i = " + i);
        }
    }
}
```

**Java Code: Prints the request from arguments passed in, then prints a loop.**

php_poc.php

```php
<?php

$request = file_get_contents( filename: 'php://input'); // Get content of request made to url

// Execute java with our compiled java code path
$results = shell_exec( command: "java -jar ../JavaPoC/out/artifacts/JavaPoC_jar/JavaPoC.jar $request");

echo $results; // Return response as text or json. Any can fit!
```

**PHP Code: Executes file I already compiled via the shell (terminal with no GUI)**

test.js

```javascript
1+ usages
async function testPhp() : Promise<string>  {
    // Make fetch request to PHP
    const result : Response  = await fetch(
        input: "http://localhost:81/localDev/HEPIK_PoC/PHP_PoC/php_poc.php",  init: {
        method: 'POST',
        body: JSON.stringify( value: {"request": "I am from client"}),
    });

    return await result.text(); // PHP response
}

testPhp().then((data : string ) : void  => console.log(data));
```

**JavaScript Code: Makes call to localhost url with php file**

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Request PoC</title>
    </head>

    <body>
        <div class="gimmick">
            This is a gimmick!
        </div>

        <script src="test.js"></script>
    </body>
</html>
```

**HTML code: Contains the JavaScript file**

- No modification of **HTML** is included in this demonstration but can easily be implemented.
- If we run this code in the browser with **PHP** running on **XAMPP**, we obtain this in the browser console:

```
{request:I am from client}                                    test.js:12
Hello and welcome! Here is a loop from the server!
i = 1
i = 2
i = 3
i = 4
i = 5
```

**Output from response**

- Our Java code is executed to perfection. Ultimately, this can be modified to return **whatever we want** in **whichever format we need**.

**Advantages**

1. Clear separation of concerns in regard to the UI and Chatbot logic. Development can be done concurrently and each component can be modified independently.
2. Ability to abstract the chatbot logic in any way we see fit.
3. Excellent developer experience as we are learning Server-Side Web Development with PHP and Java this semester while Client Side Web Development was covered last year.
4. No additional tools/setup required.

**Disadvantages**

1. Unorthodox yet effective use of programming languages.
2. Insecure access to .jar file if access is not prevented in Apache server.

## Conclusion

As exemplified, use of an **HTML-JavaScript-PHP-Java** bridge proves to be the most effective and simplest way to easily display our page on the web. However, **JSP** tries to provide an interesting approach of purely using Java for this but at the cost of learning a new way of working with client side technologies.