📘 EXECUTION REPORT (CONTINUED)

PART 7 — CONNECT CODELAB UI → SAGEMAKER (ASYNC SUBMISSION)

PREREQUISITES

- **AWS account with SageMaker enabled**
- **IAM role with:**
  - **AmazonSageMakerFullAccess**
  - **AmazonS3FullAccess**
  - **CloudWatchLogsFullAccess**
- **Existing CodeLab UI working**
- **Parts 2–6 already completed successfully**
- **code.tar.gz uploaded to S3 with runner.py**

🎯 OBJECTIVE (PART 7)

Enable **CodeLab UI** to:

• Accept Python code from editor
• Validate `#ai` hashtag
• Trigger SageMaker execution via backend
• Return **Job ID immediately**
• **Do NOT wait for output**

🔁 ARCHITECTURE (PART 7)

```
CodeLab UI
   |
   | POST /CodeEditor/run-ai-code
   |
Backend (Jersey)
   |
   | Upload code → S3
   | Start SageMaker job
   |
Return jobName immediately
```

Layer	Responsibility
UI	Submit + Poll
AuthenticationService	API orchestration
AiCodeService	Validation + S3
RunPythonFromS3	SageMaker trigger
runner.py	Output isolation
AuthenticateImpl	Log plumbing

📦 PART 7 — BACKEND CHANGES

🔷 STEP 7.1 — New REST API: `run-ai-code`

📍 File:
`AuthenticationService.java`

📌 Why this exists
This API replaces the legacy `/CodeEditor/output` path **ONLY for AI execution**.

It:

- Accepts raw Python code
- Validates `#ai`
- Uploads code to S3
- Triggers SageMaker
- Returns **jobName** (not output)

✅ CODE — PASTE EXACTLY

```
@POST
@Path("/CodeEditor/run-ai-code")
@Consumes("text/plain")
@Produces("text/plain")
public String runAiCode(String code) {

    try {
        AiCodeService service = new AiCodeService();

        // IMPORTANT: capture jobName
        String jobName = service.uploadAndRun(code);

        // RETURN jobName for async polling
        return jobName;
```

```
    } catch (Exception e) {
        e.printStackTrace();
        return "ERROR";
    }
}
```

🔷 STEP 7.2 — Service Layer: Upload + Trigger SageMaker

📍 File: Added in webservice.rest

`AiCodeService.java`

📌 Why this exists

- Keeps S3 + SageMaker logic **out of REST layer**
- Enforces `#ai` validation
- Returns **jobName**, not output

✅ CODE — FULL FILE

```
package com.thb.webservice.rest;

import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;

import java.io.File;
import java.io.FileWriter;

public class AiCodeService {

    private static final String BUCKET =
            "sagemaker-python-scripts-shivanshi";

    // --------------------------------
    // Validate AI code
    // --------------------------------
    private void validate(String code) {
        if (code == null || !code.contains("#ai")) {
            throw new RuntimeException("Missing #ai hashtag");
        }
    }

    // --------------------------------
    // Write code to temp .py file
    // --------------------------------
    private File writeTempFile(String code) throws Exception {
```

```
        File file = File.createTempFile("ai_code_", ".py");
        try (FileWriter w = new FileWriter(file)) {
            w.write(code);
        }
        return file;
    }

    // --------------------------------
    // Upload code to S3
    // --------------------------------
    public String uploadCodeToS3(String code) throws Exception {

        validate(code);

        File file = writeTempFile(code);
        String key = "ai-code/" + System.currentTimeMillis() + ".py";

        S3Client s3 = S3Client.builder()
                .region(Region.US_EAST_1)
                .build();

        s3.putObject(
                PutObjectRequest.builder()
                        .bucket(BUCKET)
                        .key(key)
                        .build(),
                RequestBody.fromFile(file)
        );

        return "s3://" + BUCKET + "/" + key;
    }

    // --------------------------------
    // Upload + Run via SageMaker
    // --------------------------------
    public String uploadAndRun(String code) throws Exception {

        String scriptS3Path = uploadCodeToS3(code);

        // Trigger SageMaker and get job name
        String jobName = RunPythonFromS3.runWithUserScript(scriptS3Path);

        return jobName;
    }
}
```

🔷 STEP 7.3 — SageMaker Trigger with Job Tracking

📍 File: Added in webservice.rest

`RunPythonFromS3.java`

📌 Why this exists

- Generates **jobName**
- Starts SageMaker training job
- Returns jobName to caller

✅ CODE — FULL FILE

```
package com.thb.webservice.rest;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sagemaker.SageMakerClient;
import software.amazon.awssdk.services.sagemaker.model.*;

import java.util.Map;

public class RunPythonFromS3 {

    public static String runWithUserScript(String scriptS3Path) {

        String jobName = "ai-job-" + System.currentTimeMillis();

        SageMakerClient client = SageMakerClient.builder()
                .region(Region.US_EAST_1)
                .build();

        CreateTrainingJobRequest request =
                CreateTrainingJobRequest.builder()
                        .trainingJobName(jobName)

.roleArn("arn:aws:iam::780167008601:role/AmazonSageMaker-ExecutionRole-
JavaS3")
                        .algorithmSpecification(
                                AlgorithmSpecification.builder()
                                        .trainingImage(
                                          "683313688378.dkr.ecr.us-east-
1.amazonaws.com/sagemaker-scikit-learn:1.2-1-cpu-py3"
                                        )

.trainingInputMode(TrainingInputMode.FILE)
                                        .build()
                        )
                        .hyperParameters(Map.of(
                                "sagemaker_program", "runner.py",
                                "sagemaker_submit_directory",
                                "s3://sagemaker-python-scripts-
shivanshi/code.tar.gz"
                        ))
                        .environment(Map.of(
                                "USER_SCRIPT_S3_PATH", scriptS3Path
                        ))
                        .resourceConfig(
                                ResourceConfig.builder()

.instanceType(TrainingInstanceType.ML_M5_LARGE)
```

```
                                        .instanceCount(1)
                                        .volumeSizeInGB(5)
                                        .build()
                                )
                                .stoppingCondition(
                                        StoppingCondition.builder()
                                                .maxRuntimeInSeconds(600)
                                                .build()
                                )
                                .outputDataConfig(
                                        OutputDataConfig.builder()
                                                .s3OutputPath("s3://sagemaker-python-
scripts-shivanshi/output/")
                                                .build()
                                )
                                .build();

                client.createTrainingJob(request);
                client.close();

                return jobName;
            }
}
```

📦 PART 7 — FRONTEND (CodeLab UI)

📍 File:
`CodeEditor.html`

📌 Where
Inside existing `myFunction()` (Run button handler)

✅ UPDATED SUCCESS BLOCK

```
success: function(jobName) {

  $("#numberOfCodeRuns").empty();
  numberOfCodeRuns++;
  $("#numberOfCodeRuns").append("<p>Points: +20</p>");

  $("#output").append("<h2>Execution Status</h2>");
  $("#output").append("<p>Job submitted</p>");
  $("#output").append("<p>Job ID: " + jobName + "</p>");
  $("#output").append("<p>Waiting for output...</p>");

  pollForAiOutput(jobName);
}
```

Updated:

```
//TO GET OUTPUT OF CODE

    var myFunction = function(){
    myFunctionClear("#output");
      let data = $("#data").val();
      //KhushiChangesStartFeaturePassFileNameToShowInErrorCase
      var Newdata= data;
      //KhushiChangesEndFeaturePassFileNameToShowInErrorCase
      // alert(Newdata);

          console.log(data);

          $.ajax({
            type: 'POST',
            url: SERVER + "/portal/login/CodeEditor/run-ai-code",
            data: Newdata,
            contentType: "text/plain",

            beforeSend: function(){
              $(".loader").show();
            },

            success: function(jobName) {

              $("#numberOfCodeRuns").empty();
              numberOfCodeRuns++;
              $("#numberOfCodeRuns").append("<p>Points: +20</p>");

              $("#output").append("<h2>Execution Status</h2>");
              $("#output").append("<p>Job submitted</p>");
              $("#output").append("<p>Job ID: " + jobName + "</p>");
              $("#output").append("<p>Waiting for output...</p>");

              pollForAiOutput(jobName);
            },

            complete: function(){
              $(".loader").hide();
            }
          });


  }
```

🎯 PART 8 OBJECTIVE — ASYNC POLLING FOR OUTPUT

Goal:
Show **actual Python output** in CodeLab UI **after job completes**, without blocking.

🧱 PART 8 — BACKEND (Log Fetching)

🔷 STEP 8.1 — Authenticate Interface

📍 File:
`Authenticate.java`

```
public String outputSageMakerLogs(String jobName)
        throws IOException, InterruptedException;
```

🔷 STEP 8.2 — AuthenticateImpl (CloudWatch Logs)

📍 File:
`AuthenticateImpl.java`

📌 Why this exists

- Reuses existing Spark log logic
- Only changes log group to SageMaker

✅ CODE — FULL METHOD

```
@Override
public String outputSageMakerLogs(String jobName)
        throws IOException, InterruptedException {

    AWSLogs logsClient = AWSLogsClientBuilder.standard()
            .withRegion(Regions.US_EAST_1)
            .build();

    String logGroup = "/aws/sagemaker/TrainingJobs";

    DescribeLogStreamsRequest streamsRequest =
            new DescribeLogStreamsRequest()
                    .withLogGroupName(logGroup)
                    .withLogStreamNamePrefix(jobName)
                    .withOrderBy("LogStreamName")
                    .withDescending(true);

    DescribeLogStreamsResult streamsResult =
```

```
            logsClient.describeLogStreams(streamsRequest);

    if (streamsResult.getLogStreams().isEmpty()) {
        return "";
    }

    StringBuilder allLogs = new StringBuilder();

    for (LogStream stream : streamsResult.getLogStreams()) {

        GetLogEventsRequest eventsRequest =
                new GetLogEventsRequest()
                        .withLogGroupName(logGroup)
                        .withLogStreamName(stream.getLogStreamName())
                        .withStartFromHead(true);

        GetLogEventsResult eventsResult =
                logsClient.getLogEvents(eventsRequest);

        for (OutputLogEvent event : eventsResult.getEvents()) {
            allLogs.append(event.getMessage()).append("\n");
        }
    }

    return allLogs.toString();
}
```

🔷 STEP 8.3 — Polling API

📍 File:

AuthenticationService.java

```
@GET
@Path("/CodeEditor/ai-job-status/{jobName}")
@Produces("text/plain")
public String getAiJobStatus(@PathParam("jobName") String jobName) {

    try {
        String logs = authenticate.outputSageMakerLogs(jobName);

        if (logs == null || logs.trim().isEmpty()) {
            return "WAITING";
        }

        return logs;

    } catch (Exception e) {
        return "WAITING";
    }
}
```

📦 PART 8 — FRONTEND POLLING

🔷 STEP 8.4 — JavaScript Poller

📍 Paste inside `<script>`

```
function pollForAiOutput(jobName) {

  let interval = setInterval(function () {

    $.ajax({
      type: 'GET',
      url: SERVER + "/portal/login/CodeEditor/ai-job-status/" + jobName,

      success: function (data) {

        if (data === "WAITING") {
          console.log("Still running...");
          return;
        }

        clearInterval(interval);

        $("#output").append("<h2>Output</h2>");

            let lines = data.split("\n");

            for (let i = 0; i < lines.length; i++) {

              if (lines[i].startsWith("__AI_OUTPUT__")) {
                let cleanLine = lines[i].replace("__AI_OUTPUT__", "").trim();
                $("#output").append("<p>" + cleanLine + "</p>");
              }

            }

      },

      error: function () {
        console.log("Polling failed");
      }
    });

  }, 10000); // every 10 sec
}
```

RUNNER SCRIPT (PACKED IN TAR)

📍 File

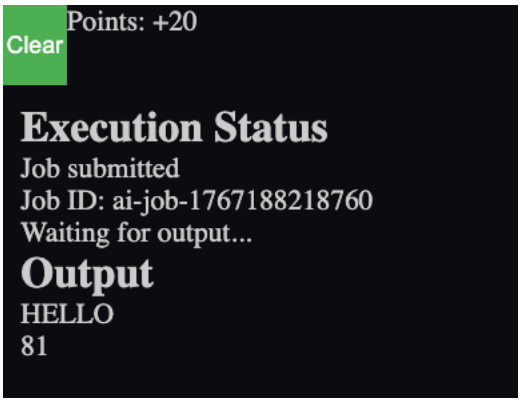
`runner.py` (inside `code.tar.gz`)

✅ CODE

```
import os
import subprocess
import boto3
import sys
from io import StringIO

print("Runner started")

s3_path = os.environ.get("USER_SCRIPT_S3_PATH")
if not s3_path:
    raise Exception("USER_SCRIPT_S3_PATH not set")

print("User script S3 path:", s3_path)

bucket = s3_path.replace("s3://", "").split("/")[0]
key = "/".join(s3_path.replace("s3://", "").split("/")[1:])

local_file = "/opt/ml/code/user_code.py"

s3 = boto3.client("s3")
s3.download_file(bucket, key, local_file)

print("Downloaded user code, executing...")

# ---- CAPTURE STDOUT ----
old_stdout = sys.stdout
sys.stdout = StringIO()

try:
    exec(open(local_file).read(), {})
    output = sys.stdout.getvalue()
finally:
    sys.stdout = old_stdout

# ---- PRINT USER OUTPUT WITH MARKER ----
for line in output.splitlines():
    print("__AI_OUTPUT__", line)

print("Runner finished")
```

FINAL RESULT

✓ CodeLab UI executes Python dynamically ✓ SageMaker jobs are async ✓ UI polls logs ✓ Only user output appears


Clear
Points: +20
Execution Status
Job submitted
Job ID: ai-job-1767188218760
Waiting for output...
Output
HELLO
81