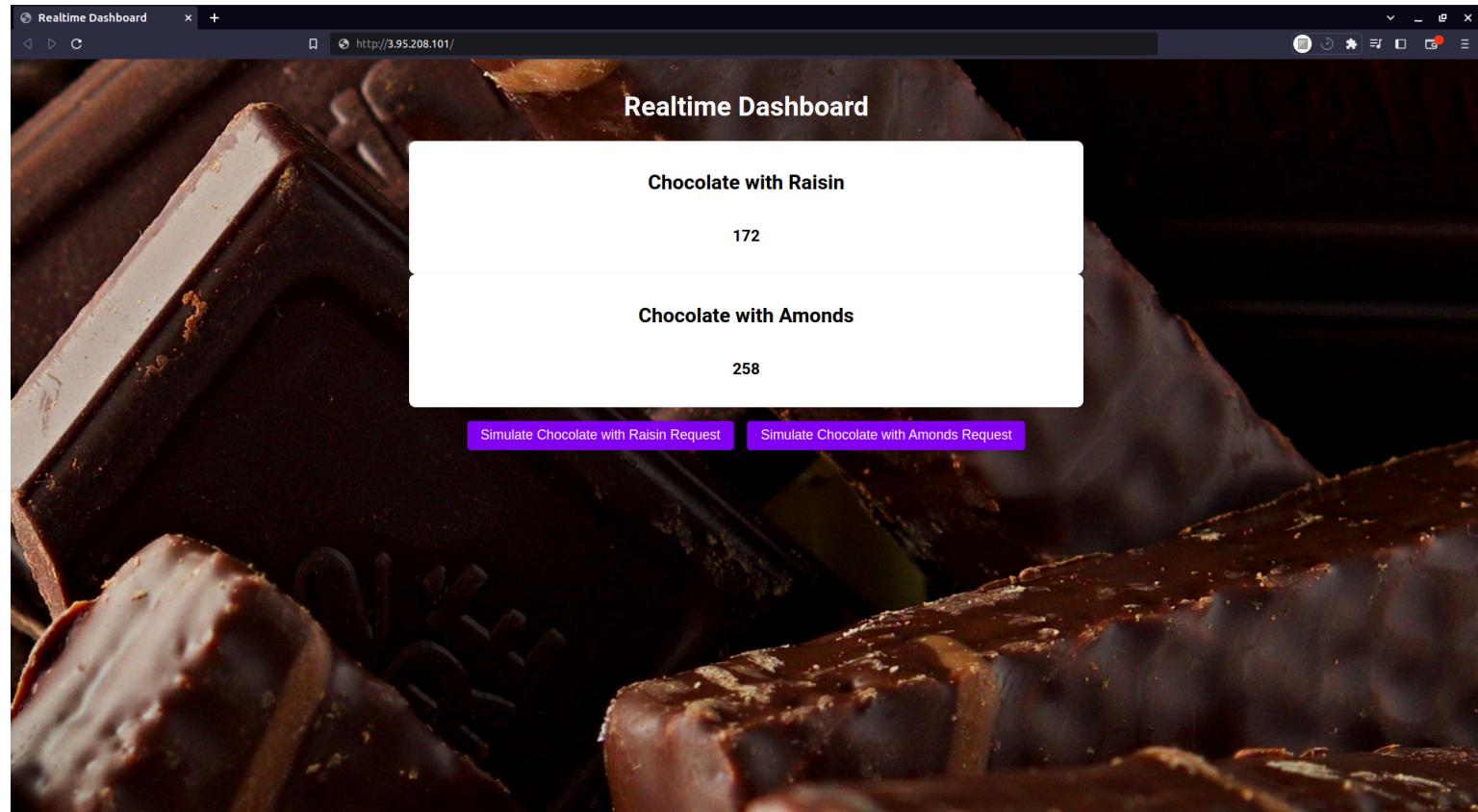


A. Build The App

- build an Express and React app with TypeScript that handles the two GET requests and displays the request counts in real-time



- Deploy The Backend app and the Frontend app to separate Amazon AWS servers

A screenshot of the AWS CloudWatch Metrics dashboard. On the left, there is a sidebar with navigation links for Services, EC2 Dashboard, EC2 Global View, Events, Limits, Instances, Images, AMIs, AMI Catalog, and Elastic Block Store. The main area shows two EC2 instances: "app-server" (i-003b5152c3a60da80) and "realtime-app" (i-0cec9978c17aff8ae), both listed as "Running". Below the instance list, there is a section titled "Instances: i-003b5152c3a60da80 (app-server), i-0cec9978c17aff8ae (realtime-app)". This section contains eight metrics charts arranged in a 2x4 grid. The charts are: "CPU utilization (%)" (Percent vs. Time), "Status check failed (any) (count)" (Count vs. Time), "Status check failed (instance) ... (Count vs. Time)", "Status check failed (system) (Count vs. Time)", "Network in (bytes)" (Bytes vs. Time), "Network out (bytes)" (Bytes vs. Time), "Network packets in (count)" (Count vs. Time), and "Network packets out (count)" (Count vs. Time). All charts show data from 18:40 to 19:40.

- c. Build a Java Class inside the AnyLogic Client, so that after the production of each type of product

The screenshot shows the AnyLogic Personal Learning Edition interface. The top menu bar includes File, Edit, View, Model, Tools, and Help. The left sidebar displays the project structure under TP4, including sub-folders like Chocolat, MailingServiceTrigger, PostRequest, Simulation, Run Configuration, Database, Resources, FluidSelectInput, test, and usine. The main workspace contains a code editor window titled "PostRequest.java" with the following Java code:

```

19     return super.toString();
20 }
21
22 public static void testin() {
23     System.out.println("Loup");
24 }
25
26 public static void send(boolean amonds) {
27     String urlString = amonds ? "http://54.163.200.76/chocolate" : "http://54.163.200.76/white";
28     try {
29         URL url = new URL(urlString);
30         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
31         conn.setRequestMethod("GET");
32         int responseCode = conn.getResponseCode();
33         if (responseCode == HttpURLConnection.HTTP_OK) {
34             System.out.println("yes 200 to" + urlString);
35         } else {
36             System.out.println("no");
37         }
38     } catch (IOException e) {
39         e.printStackTrace();
40     }
41 }
42
43 private static final long serialVersionUID = 1L;
44
45 }

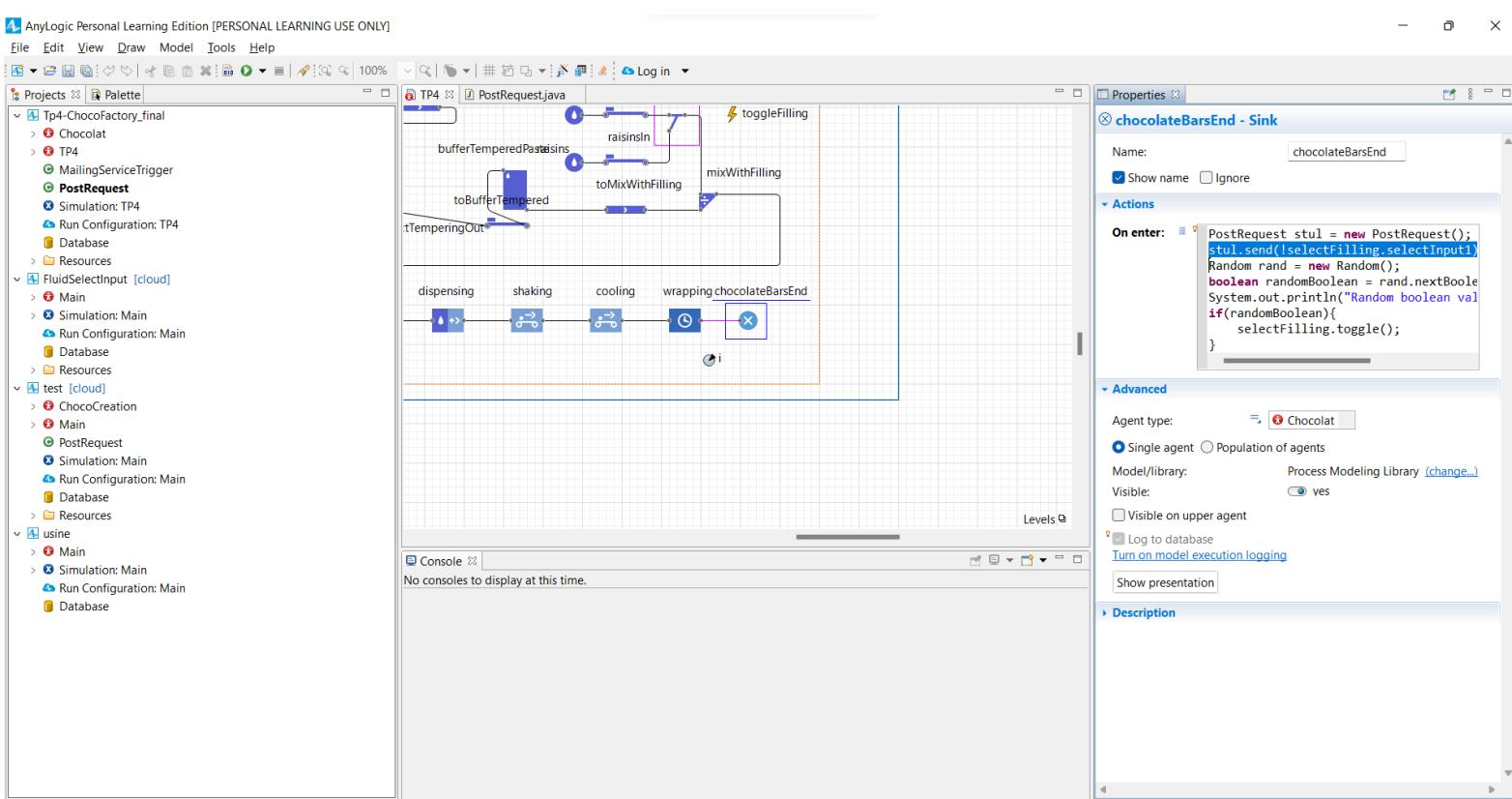
```

To the right of the code editor is a "Properties" panel titled "PostRequest - Java Class". It contains sections for Description, Ignore, and Actions. The Actions section shows the following code:

```

On enter: PostRequest stu1 = new PostRequest();
stu1.send(selectfilling.selectInput());
Random rand = new Random();
boolean randomBoolean = rand.nextBoolean();
System.out.println("Random boolean val");
if(randomBoolean){
    selectfilling.toggle();
}

```



- d. this event will trigger the backend Server to notify it about the final Creation of a new specific type of product, so that the server would count and display in real-time the number of final Products of each type

```

File Edit Selection View Go Run Terminal Help
EXPLORER ... server.ts ●
REALTIME-DASHBOARD
> .cache
> dist
> img
> node_modules
src
  App.css M
  App.tsx M
  index.html
  index.tsx
  server.ts
  .gitignore
  infra.sh u
  package-lock.json
  package.json
  README.md
  tsconfig.json
... OUTLINE
... TIMELINE
Live Share Git Graph tabnine starter Ln 26, Col 1 (24 selected) Spaces: 2 UTF-8 LF {} TypeScript Go Live Prettier
server.ts - Visual Studio Code
● server.ts - Visual Studio Code
src > server.ts > app.get('/chocolate-with-ammonds') callback
16   wss.clients.forEach(client => {
17     if (client.readyState === WebSocket.OPEN) {
18       client.send(JSON.stringify(counters));
19     }
20   });
21   res.send('Chocolate-with-raisin route accessed');
22 });
23
24 app.get('/chocolate-with-ammonds', (req: Request, res: Response) => {
25   counters.chocolate++;
26   wss.clients.forEach(client => {
27     if (client.readyState === WebSocket.OPEN) {
28       client.send(JSON.stringify(counters));
29     }
30   });
31   res.send('Chocolate-with-ammonds route accessed');
32 });
33
34 const PORT = process.env.PORT || 3001;
35 httpServer.listen(PORT, () => {
36   console.log(`Server is listening on port ${PORT}`);
37 });

```

B. The Mailing Service

- a. Create an API endpoint that Triggers a Lambda function, which is going to handle the process of sending emails to Suppliers or Special Customers (For example after passing a command of 50 products)

Lambda > Functions > mailSenderFunction

mailSenderFunction

Function overview [Info](#)

 mailSenderFunction  Layers (0)	Description -
 API Gateway	Last modified 2 hours ago
+ Add trigger	Function ARN <code>arn:aws:lambda:us-east-1:664390427007:function:mailSenderFunction</code>
	Function URL Info -

Code [Test](#) [Monitor](#) [Configuration](#) [Aliases](#) [Versions](#)

Code source [Info](#) [Upload from](#)

File Edit Find View Go Tools Window [Test](#) Deploy

CloudShell Feedback Language © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

File Edit Selection View Go Run Terminal Help

EXPLORER ... sender.py lmandacode.py

MAILSENDER lmandacode.py sender.py

```

14     )
15
16     response = client.send_email(
17         Source=sender,
18         Destination={'ToAddresses': recipients},
19         Message={
20             'Subject': {'Data': subject},
21             'Body': {'Text': {'Data': body}}
22         }
23     )
24     return response
25
26 def lambda_handler(event, context):
27     sender_email = "ayoubhedfi80@gmail.com"
28     teammate_emails = ["ayoub.hedfi@insat.ucar.tn", "lupojericho@gmail.com"]
29     email_subject = "Hello Team From Lambda"
30     email_body = "This is a test email sent using Amazon SES."
31
32     response = send_email(sender_email, teammate_emails, email_subject, email_body)
33
34     return {
35         'statusCode': 200,
36         'messageId': response['MessageId']
37     }

```

> OUTLINE > TIMELINE

0 ▲ 0 Live Share Discovering Python Interpreters tabnine starter See Tabnine Insights LF Python 3.8.10 64-bit Go Live Prettier

- Create the Mailing Client (Amazon Simple Email Service) to Manage Verified emails of the Costumers plus Contact-Sales emails

aws Services Search [Alt+S]

Amazon SES Configuration: Verified identities

Verified identities

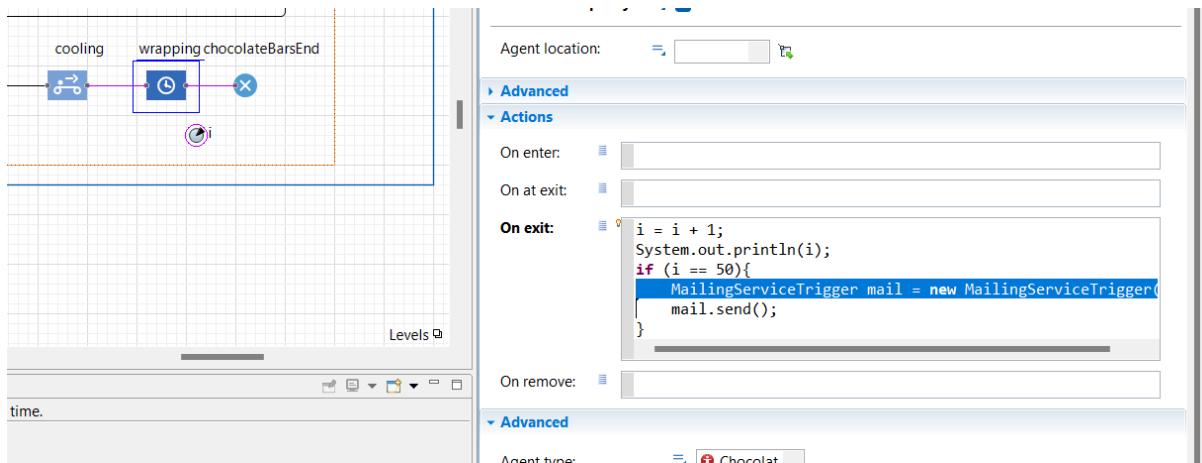
A *verified identity* is a domain, subdomain, or email address you use to send email through Amazon SES. [Learn more](#)

New identity status update
The Identity status now represents the explicit verification of the identity itself. For the domain identities this means verifying ownership through updates in the DNS records, and for the email address identities, this means opening the verification email from no-reply-aws@amazon.com and selecting the link to complete the verification process. [Learn more](#)

Identities (5) Info			
Send test email Delete Create identity			
<input type="checkbox"/> Identity	Identity type	Identity status	
med.amine.karoui@gmail.com	Email address	Verification pending	
ayoubhedfi80@gmail.com	Email address	Verified	
lupojericho@gmail.com	Email address	Verified	
mohamedamine.karoui@insat.ucar.tn	Email address	Verified	
ayoub.hedfi@insat.ucar.tn	Email address	Verified	

CloudShell Feedback Language © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

- d. This would be the Java Script that would trigger the event of reaching 50 Products to send the emails to Customers



The screenshot shows the Java code editor with the file "MailingServiceTrigger.java" open. The code implements the `MailingServiceTrigger` class, which extends `Serializable`. It includes methods for construction, conversion to string, testing, and sending emails via an HTTP connection. The code uses `System.out.println` statements to log messages.

```

1 /**
2 * MailingServiceTrigger
3 */
4 import java.io.BufferedReader;
5 import java.io.IOException;
6 import java.io.InputStreamReader;
7 import java.net.HttpURLConnection;
8 import java.net.URL;
9
10 public class MailingServiceTrigger implements Serializable {
11     ...
12     public MailingServiceTrigger() {
13         System.out.println("Class MailingServiceTrigger Created");
14     }
15     ...
16     @Override
17     public String toString() {
18         return super.toString();
19     }
20     ...
21     public static void testin() {
22         System.out.println("Loup MailingServiceTrigger");
23     }
24     ...
25     public static void send() {
26         String urlString = "https://teq232qkxf.execute-api.us-east-1.amazonaws.com/default/mailSenderFor";
27         try {
28             URL url = new URL(urlString);
29             HttpURLConnection conn = (HttpURLConnection) url.openConnection();
30             conn.setRequestMethod("GET");
31             int responseCode = conn.getResponseCode();
32             if (responseCode == HttpURLConnection.HTTP_OK) {
33                 System.out.println("=> mail sent 200 to" + urlString);
34             } else {
35                 System.out.println("no");
36             }
37         } catch (IOException e) {
38             e.printStackTrace();
39         }
40     }
41     ...
42     private static final long serialVersionUID = 1L;
43 }

```