# Configuration Manual

MSc Research Project
MSc Cyber Security

Prajwal Yadav
Student ID: 22210270

School of Computing
National College of Ireland

Supervisor: Prof. Mark Monaghan

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | | | |
|---|---|---|---|
| **Student Name:** | Prajwal Balasaheb Yadav | | |
| **Student ID:** | 22210270 | | |
| **Programme:** | MSc in Cybersecurity | **Year:** | 2024 |
| **Module:** | Practicum | | |
| **Lecturer:** | Prof. Mark Monaghan | | |
| **Submission Due Date:** | 12/08/2024 | | |
| **Project Title:** | Enhancing Serverless Architecture Security: A Framework for Mitigating Event Injection Attacks | | |
| **Word Count:** | 610 | **Page Count:** | 13 |

**Signature:** Prajwal Yadav

**Date:** 12/08/2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Prajwal Yadav
Student ID: 22210270

## 1 Introduction

This manual documents about all the used tools, technologies as well as some steps for building the framework. The manual is further divided into few parts such as environment setup, tools technology used and some screenshots of code, implementation.

## 2 Project Requirements

This section has the essential requirements needed to successfully implement the proposed serverless security framework. It covers hardware as well as the software specifications, along with the necessary cloud services and development tools:

**1. Hardware requirements:**
Device: MacBook air m2 with 16GB and 512 GB SSD.(with sufficient processing power andstorage for development and testing.

**2. Software requirements:**

- **Operating System**: macOS Ventura.
- **Progaraming Language**: Python 3.9.
- **Framework: Django** : 5.0.7
- **IDE:** PyCharm

**3. Cloud Services:**

- AWS Account: Essential as to develop and deploy a serverless function and manage the API gateways as well along with Lambda.
- AWS Lambda: For deploying the serverless functions.
- AWS CloudWatch: For real time monitoring and logging of events and triggering alerts.
- AWS SNS: To send notifications and alerts in respond for detecting attack.
- AWS cloud9: cloud-based IDE for supporting. Real time collaboration.
- API Gateway: Manage and securing HTTP Request.

**4. Development tools**

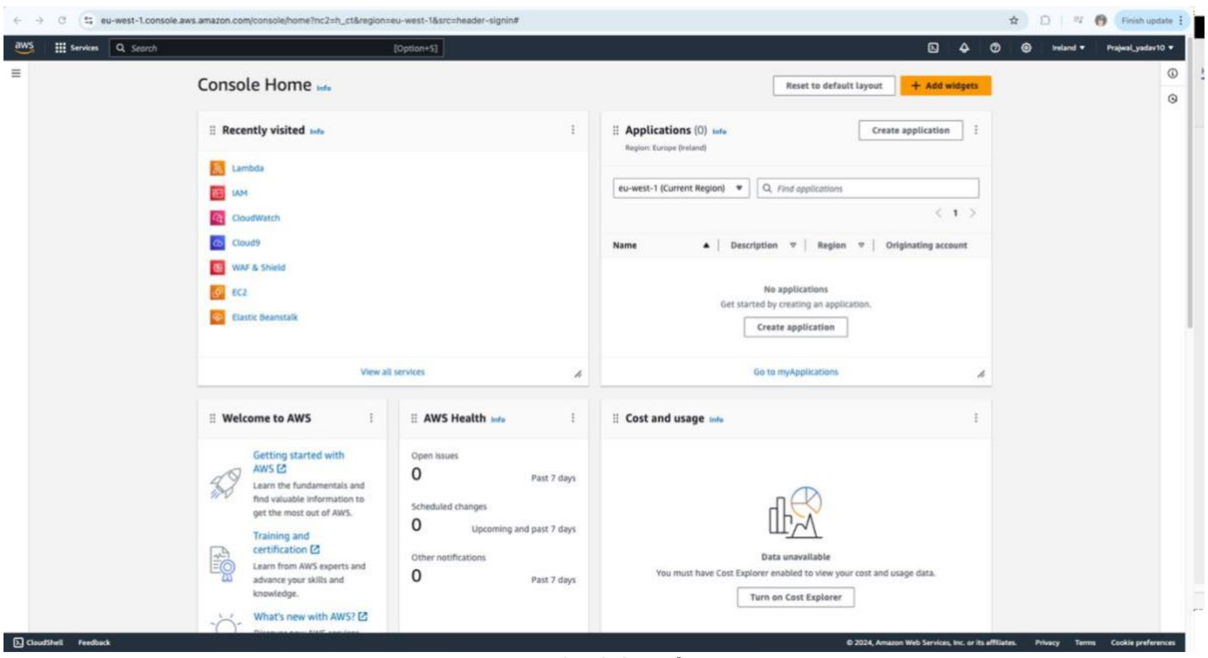
**GitHub:** For version control purpose,

# 3 Building the Framework for Event injection attacks

My project was focused on building a robust framework for detecting and preventing event injection attacks in serverless environments. This framework uses advanced machine learning model specially the Isolation Forest ML model for analyzing data for anomalies that could indicate security threats. The Framework is built on AWS environment such as Lambda, CloudWatch, SNS and API gateway which provides scalable and efficient solution for safeguard he serverless application.

## 3.1 Setup AWS Account:

Created AWS account of personal on free tier. Also, Configured IAM roles and permissions for securing the environment as well.

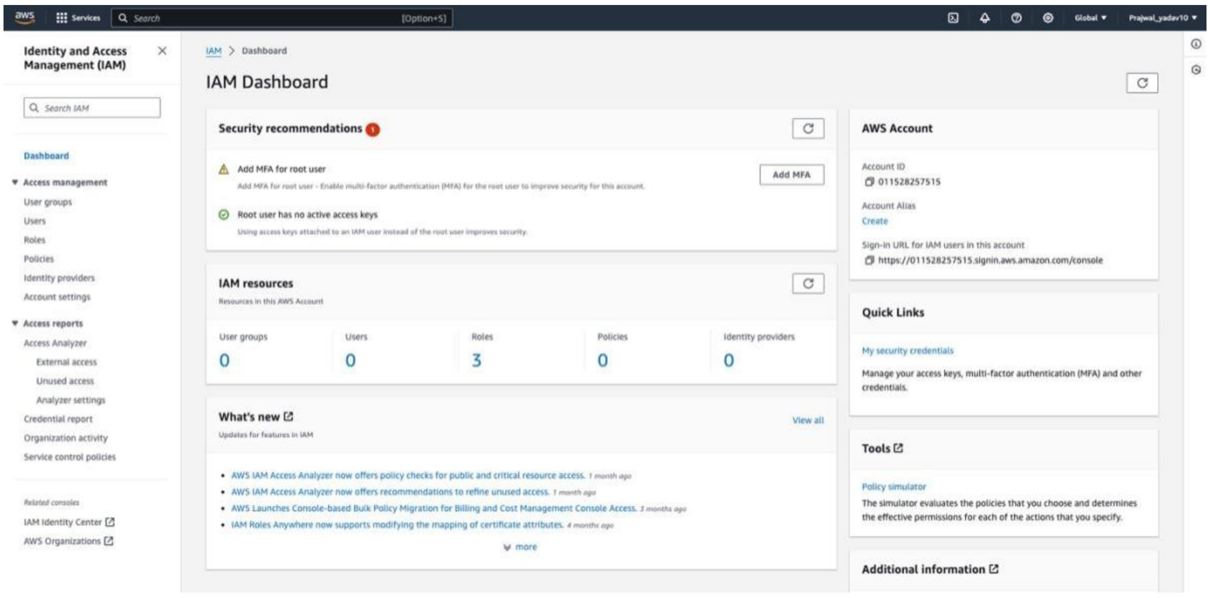


Learn the fundamentals and find valuable information to get the most out of AWS.

Learn from AWS experts and advance your skills and knowledge.

You must have Cost Explorer enabled to view your cost and usage data.

Fig1. AWS Console


aws Services Search [Option+S] Global Prajwal_yadav10

Identity and Access Management (IAM)

Search IAM

Dashboard

- Access management
  - User groups
  - Users
  - Roles
  - Policies
  - Identity providers
  - Account settings
- Access reports
  - Access Analyzer
    - External access
    - Unused access
    - Analyzer settings
  - Credential report
  - Organization activity
  - Service control policies

Related consoles

IAM Identity Center

AWS Organizations

IAM > Dashboard

IAM Dashboard

Security recommendations 1

Add MFA for root user — Add MFA

Add MFA for root user - Enable multi-factor authentication (MFA) for the root user to improve security for this account.

Root user has no active access keys

Using access keys attached to an IAM user instead of the root user improves security.

IAM resources

Resources in this AWS Account

| User groups | Users | Roles | Policies | Identity providers |
|---|---|---|---|---|
| 0 | 0 | 3 | 0 | 0 |

What's new — View all

Updates for features in IAM

- AWS IAM Access Analyzer now offers policy checks for public and critical resource access. 1 month ago
- AWS IAM Access Analyzer now offers recommendations to refine unused access. 1 month ago
- AWS Launches Console-based Bulk Policy Migration for Billing and Cost Management Console Access. 3 months ago
- IAM Roles Anywhere now supports modifying the mapping of certificate attributes. 4 months ago

more

AWS Account

Account ID
011528257515

Account Alias
Create

Sign-in URL for IAM users in this account
https://011528257515.signin.aws.amazon.com/console

Quick Links

My security credentials

Manage your access keys, multi-factor authentication (MFA) and other credentials.

Tools

Policy simulator

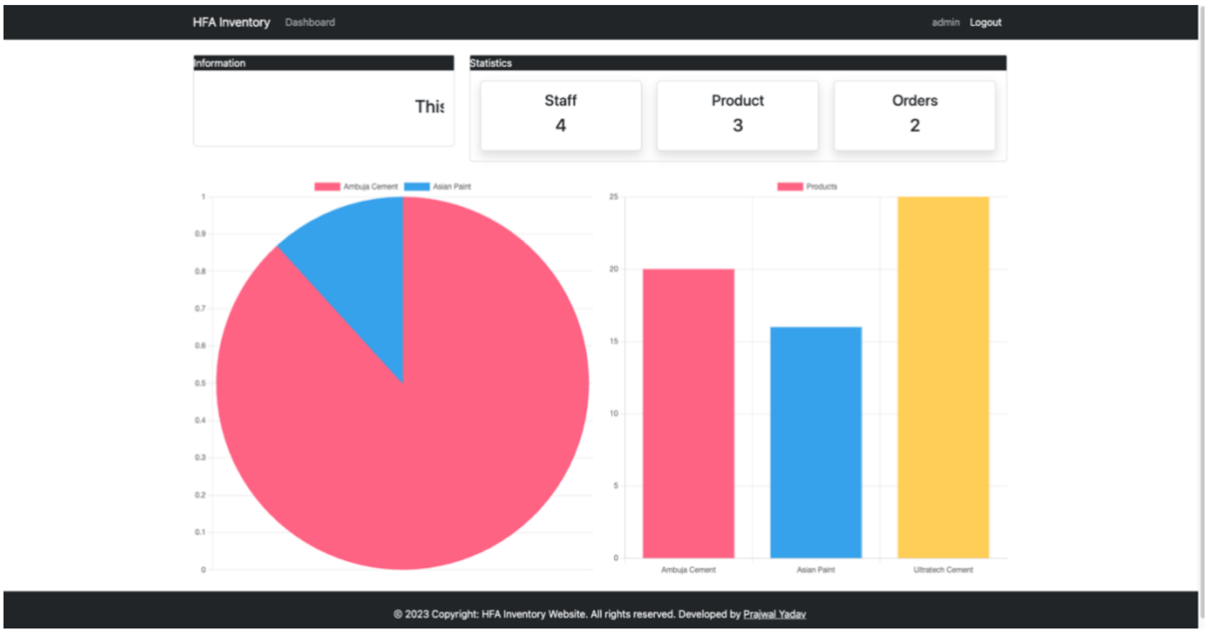
The simulator evaluates the policies that you choose and determines the effective permissions for each of the actions that you specify.

Additional information

Fig2. IAM

## 3.2 Creation of Webapp on deploying serverless environment:

The use of webapp was very critical component of proposed framework which enables real time detection and prevention of event injection attacks. It involves setting up the web application using the Django framework, integrating it with AWS services and ensuring that it is safe and robust.


HFA Inventory Dashboard admin Logout

Information

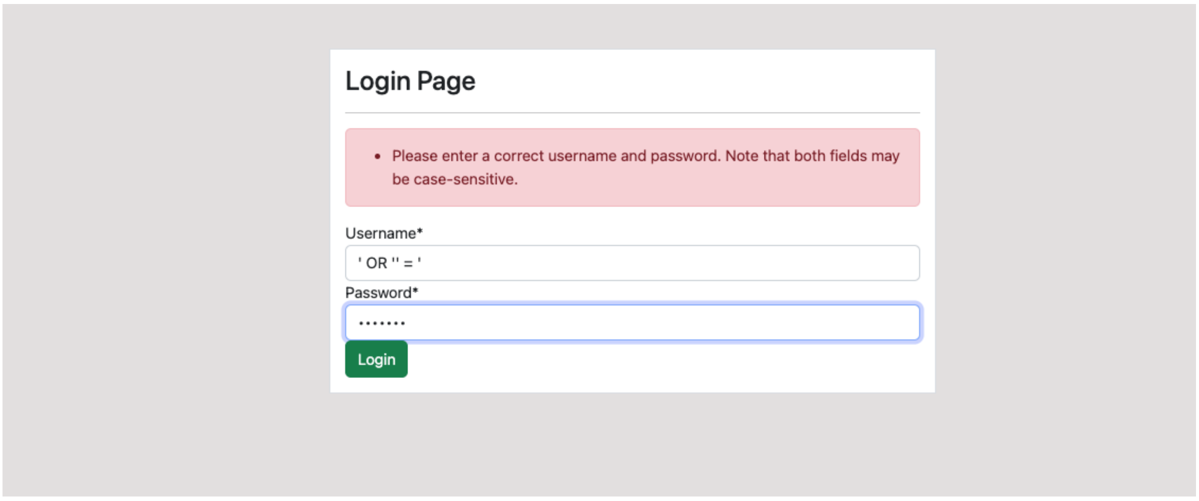
This

Statistics

| Staff | Product | Orders |
|---|---|---|
| 4 | 3 | 2 |

© 2023 Copyright: HFA Inventory Website. All rights reserved. Developed by Prajwal Yadav

Fig3. WEBAPP dashboard



Login Page

- Please enter a correct username and password. Note that both fields may be case-sensitive.

Username*

' OR '' = '

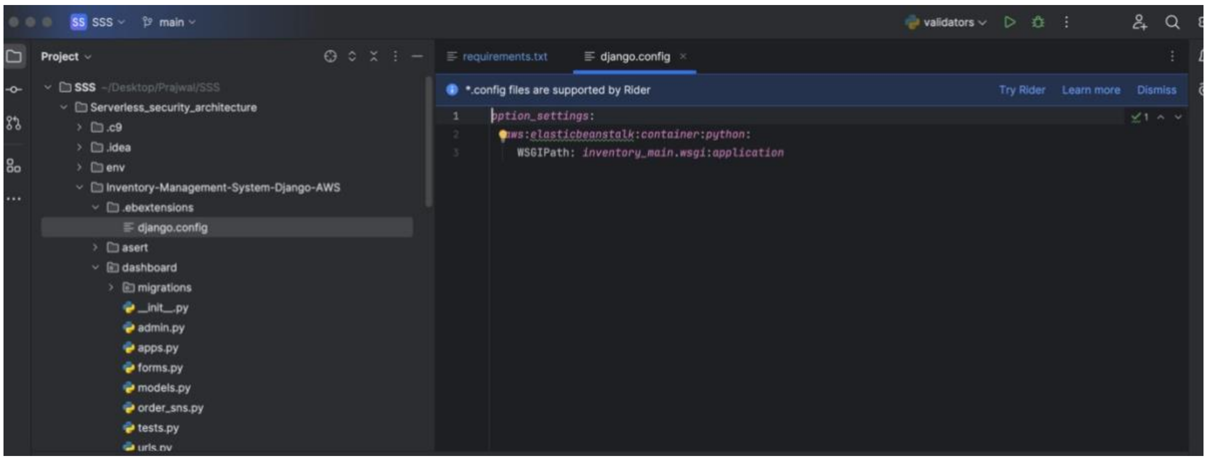
Password*

•••••••

Fig4. WEBAPP LOGIN

### 3.2.1 Setting up with Django framework

Django Installation and Project initiation:

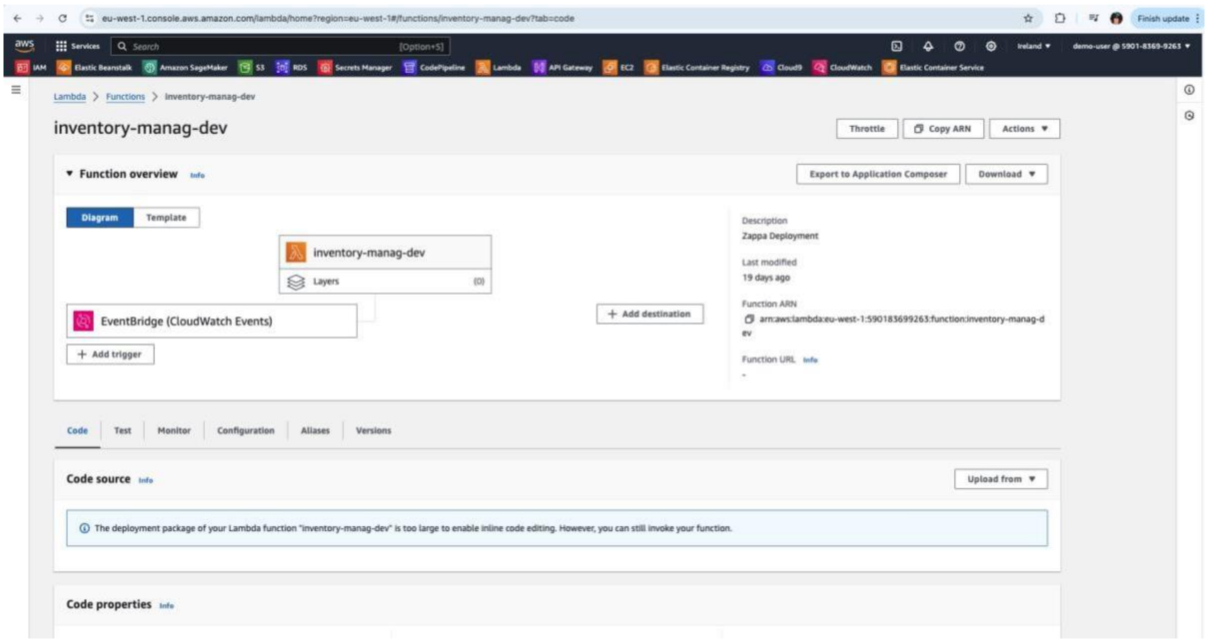


```
option_settings:
 aws:elasticbeanstalk:container:python:
   WSGIPath: inventory_main.wsgi:application
```

Fig5. Django

### 3.2.2 Integration with AWS Lambda:

Developed serverless functions using AWS lambda for handling the backend logic of web application. The lambda function are written in language Python and designed for interact with the Django app. Also processing incoming data events and executing the Isolation ForestML model for anomaly detection.


eu-west-1.console.aws.amazon.com/lambda/home?region=eu-west-1#/functions/inventory-manag-dev?tab=code

Lambda > Functions > inventory-manag-dev

inventory-manag-dev

Function overview

Description
Zappa Deployment

Last modified
19 days ago

Function ARN
arn:aws:lambda:eu-west-1:590183699263:function:inventory-manag-dev

Function URL
-

Code source

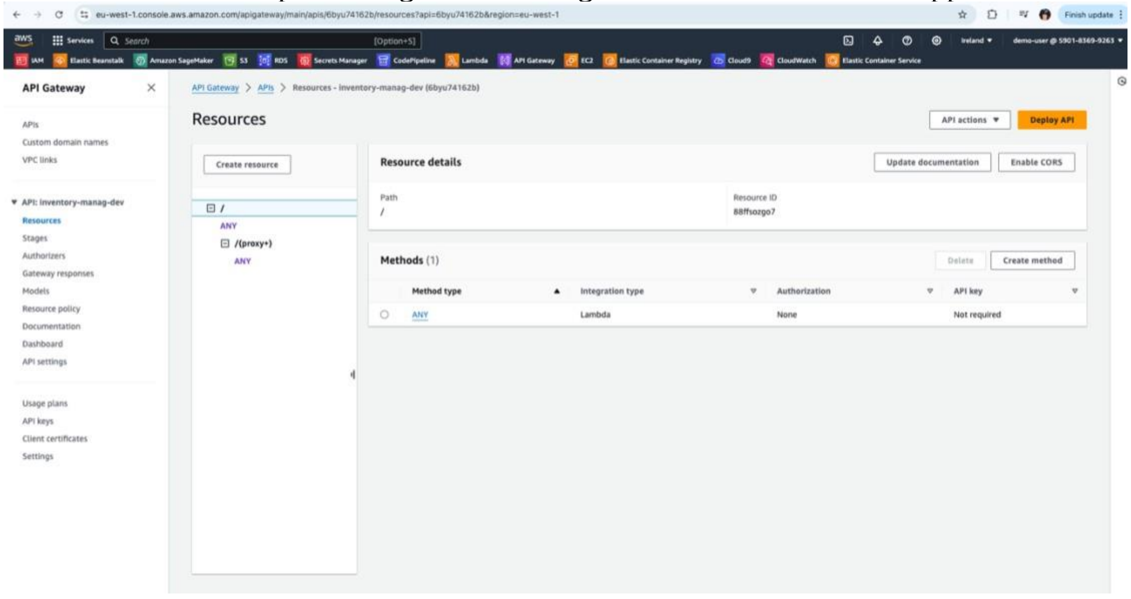
The deployment package of your Lambda function "inventory-manag-dev" is too large to enable inline code editing. However, you can still invoke your function.

Code properties

Fig6. LAMBDA

## 3.2.3 API gateway:

I used AWS API gateway for creating RESTful Api which has interface between web application and Lambda functions. Also, It handles HTTP requests, passing through lambda functions for processing and returning to the results of the web apps.


eu-west-1.console.aws.amazon.com/apigateway/main/apis/6byu74162b/resources?api=6byu74162b&region=eu-west-1

API Gateway > APIs > Resources - inventory-manag-dev (6byu74162b)

Resources

Resource details

Path
/

Resource ID
88ffsozgo7

Methods (1)

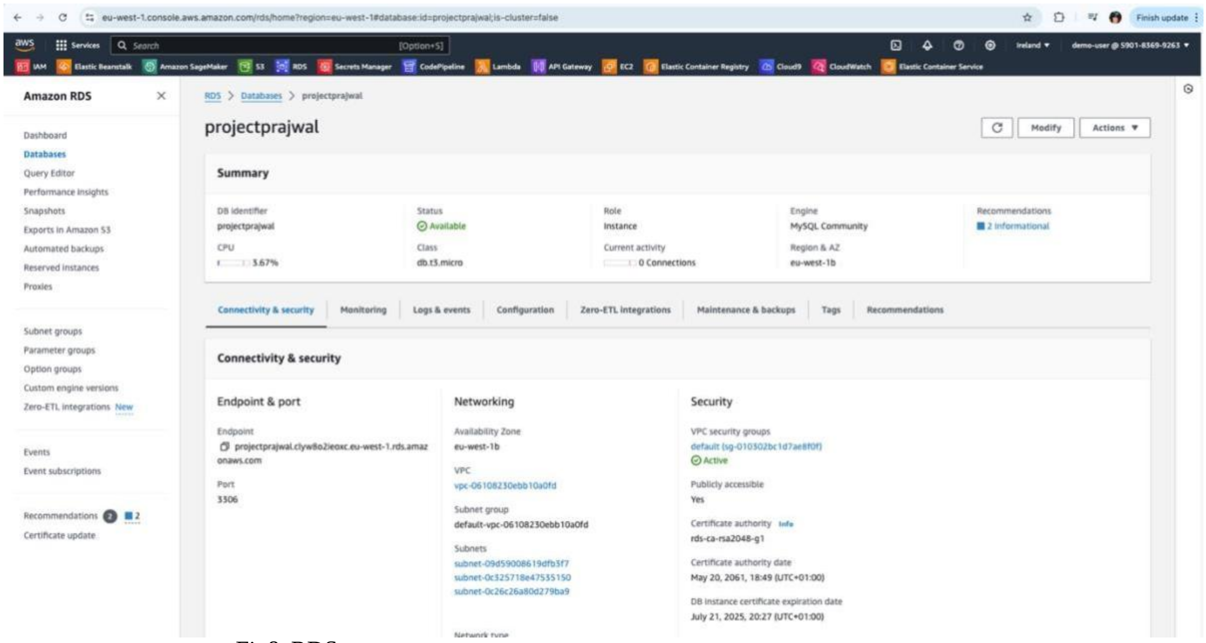
| Method type | Integration type | Authorization | API key |
|---|---|---|---|
| ANY | Lambda | None | Not required |

Fig7. API GATEWAY

## 3.2.4 Database Configuration With RDS:

### *3.2.4.1 RDS Setup:*

I set up an Amazon RDS (Relational Database Service) instance with MYSQL. RDS provides a managed database services which is scalable, secured and also integrates smoothly with Django.



projectprajwal

Summary

DB identifier: projectprajwal
CPU: 3.67%
Status: Available
Class: db.t3.micro
Role: Instance
Current activity: 0 Connections
Engine: MySQL Community
Region & AZ: eu-west-1b
Recommendations: 2 Informational

Connectivity & security

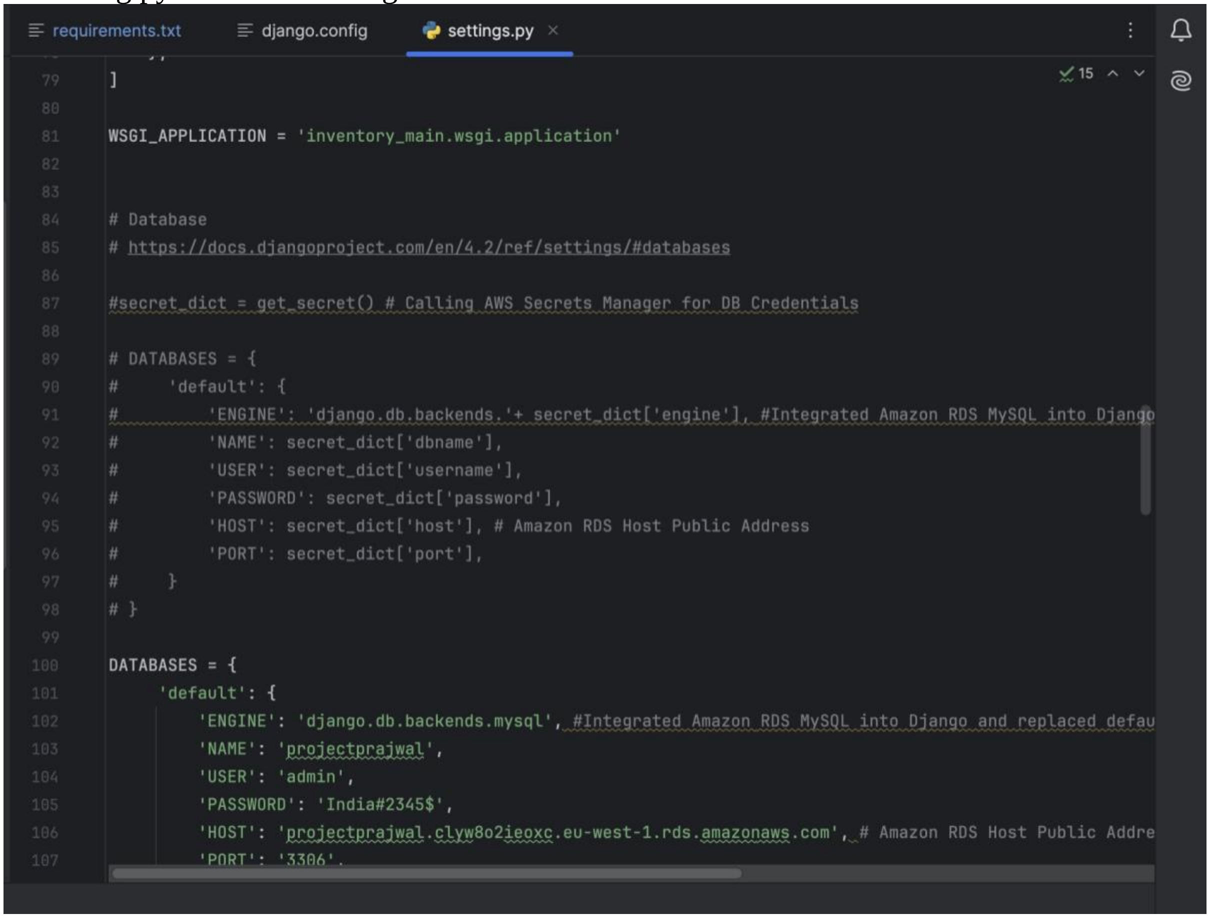
Endpoint & port
Endpoint: projectprajwal.clyw8o2ieoxc.eu-west-1.rds.amazonaws.com
Port: 3306

Networking
Availability Zone: eu-west-1b
VPC: vpc-06108230ebb10a0fd
Subnet group: default-vpc-06108230ebb10a0fd
Subnets: subnet-09d59008619dfb3f7, subnet-0c325718e47535150, subnet-0c26c26a80d279ba9
Network type

Security
VPC security groups: default (sg-010302bc1d7ae8f0f) Active
Publicly accessible: Yes
Certificate authority: rds-ca-rsa2048-g1
Certificate authority date: May 20, 2061, 18:49 (UTC+01:00)
DB instance certificate expiration date: July 21, 2025, 20:27 (UTC+01:00)

Fig8. RDS

In setting.py file I have configured the RDS with credentials.

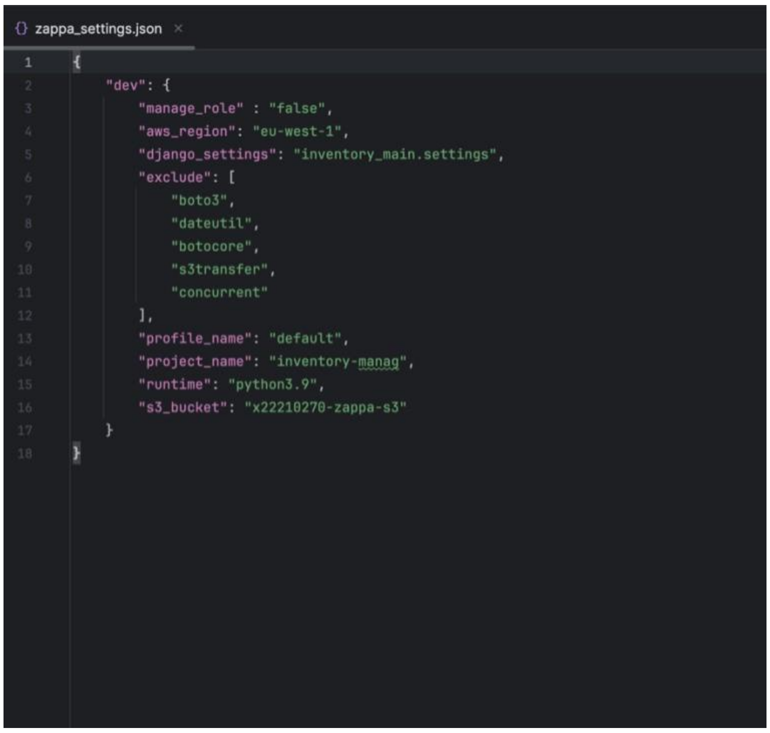


```
]

WSGI_APPLICATION = 'inventory_main.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

#secret_dict = get_secret() # Calling AWS Secrets Manager for DB Credentials

# DATABASES = {
#     'default': {
#         'ENGINE': 'django.db.backends.'+ secret_dict['engine'], #Integrated Amazon RDS MySQL into Django
#         'NAME': secret_dict['dbname'],
#         'USER': secret_dict['username'],
#         'PASSWORD': secret_dict['password'],
#         'HOST': secret_dict['host'], # Amazon RDS Host Public Address
#         'PORT': secret_dict['port'],
#     }
# }

DATABASES = {
     'default': {
         'ENGINE': 'django.db.backends.mysql', #Integrated Amazon RDS MySQL into Django and replaced defau
         'NAME': 'projectprajwal',
         'USER': 'admin',
         'PASSWORD': 'India#2345$',
         'HOST': 'projectprajwal.clyw8o2ieoxc.eu-west-1.rds.amazonaws.com', # Amazon RDS Host Public Addre
         'PORT': '3306',
```

Fig9. SETTINGs

## 3.2.5 Integration with AWS Lambda as well as API Gateway

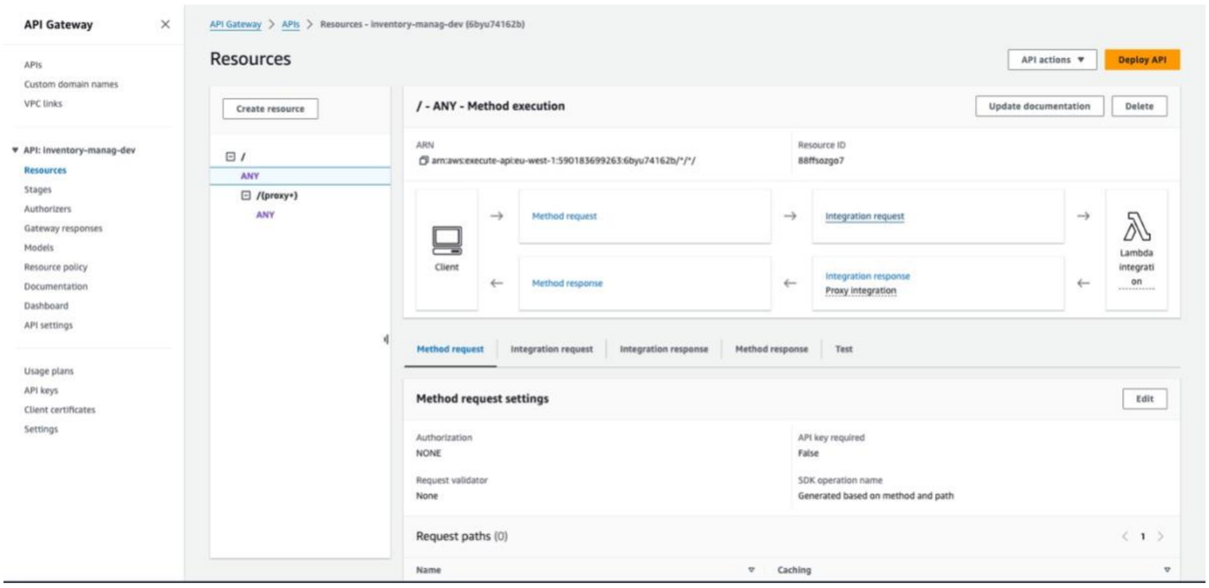
Lambda Function: Developed serverless functions using AWS Lambda for handling backend processing task, these functions are triggered by HTTP which is managed by gateway.
ZipApp: I used ZipApp package for bundling the Django app along with its dependencies inthe AWS Lambda.



zappa_settings.json

```
{
    "dev": {
        "manage_role" : "false",
        "aws_region": "eu-west-1",
        "django_settings": "inventory_main.settings",
        "exclude": [
            "boto3",
            "dateutil",
            "botocore",
            "s3transfer",
            "concurrent"
        ],
        "profile_name": "default",
        "project_name": "inventory-manag",
        "runtime": "python3.9",
        "s3_bucket": "x22210270-zappa-s3"
    }
}
```

Fig10. ZAPPA



/ - ANY - Method execution

ARN
arn:aws:execute-api:eu-west-1:590183699263:6byu74162b/*/*/

Resource ID
88ffsozgo7

Method request settings

Authorization
NONE

API key required
False

Request validator
None

SDK operation name
Generated based on method and path

Request paths (0)

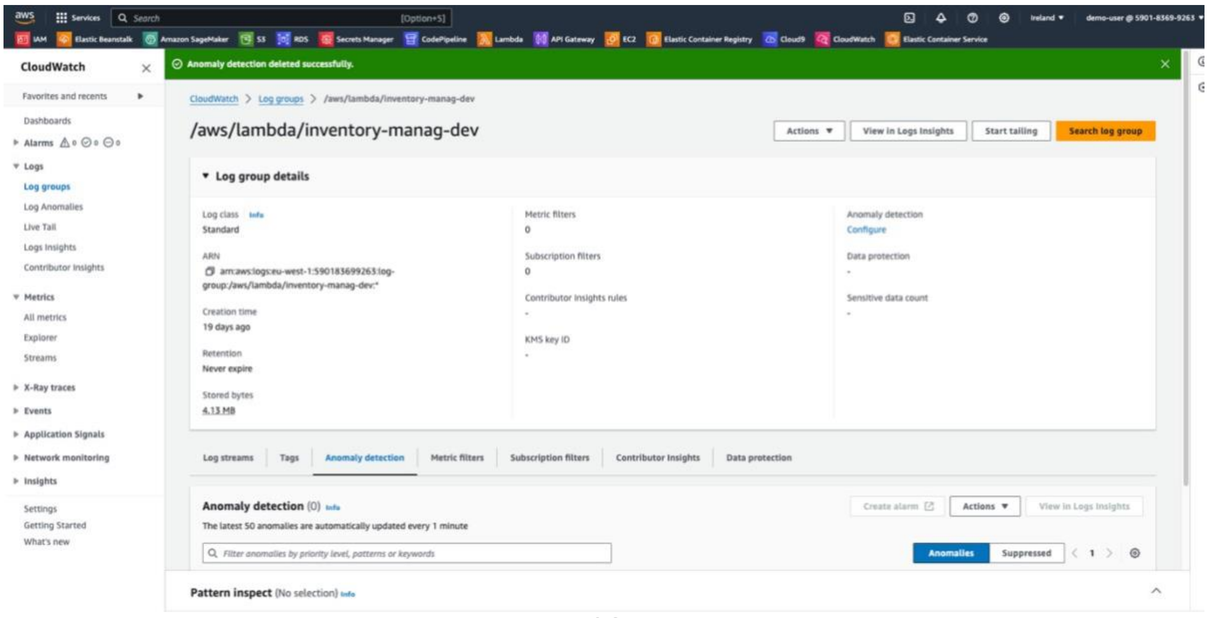
| Name | Caching |
|---|---|

Fig11. API

API Gateway:
Configured AWS API gateway for creating RESTful APIs which is the interface between the frontend Django Application and backend of Lambda functions. It ensures that HTTP request are routed appropriately to Lambda.

## 3.2.6 CloudWatch:

Amazon CloudWatch is a service in which it monitors applications and also responds for performance changes, optimizes resource use, and provides insights into operational health (AWS, 2018)



Anomaly detection deleted successfully.

CloudWatch > Log groups > /aws/lambda/inventory-manag-dev

/aws/lambda/inventory-manag-dev

Log group details

Log class Info
Standard

ARN
arn:aws:logs:eu-west-1:590183699263:log-group:/aws/lambda/inventory-manag-dev:*

Creation time
19 days ago

Retention
Never expire

Stored bytes
4.13 MB

Metric filters
0

Subscription filters
0

Contributor Insights rules
-

KMS key ID
-

Anomaly detection
Configure

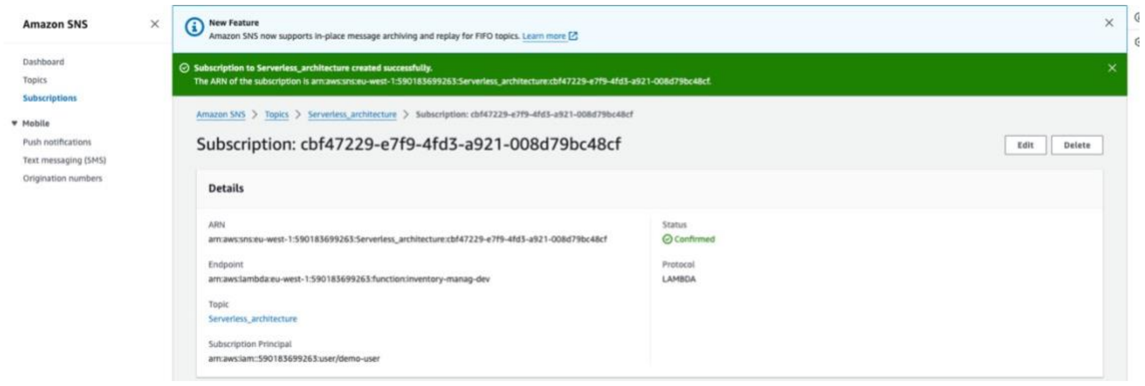
Data protection
-

Sensitive data count
-

Anomaly detection (0) Info
The latest 50 anomalies are automatically updated every 1 minute

Pattern inspect (No selection) Info

Fig12 . LOGs



New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. Learn more

Subscription to Serverless_architecture created successfully.
The ARN of the subscription is arn:aws:sns:eu-west-1:590183699263:Serverless_architecture:cbf47229-e7f9-4fd3-a921-008d79bc48cf.

Amazon SNS > Topics > Serverless_architecture > Subscription: cbf47229-e7f9-4fd3-a921-008d79bc48cf

Subscription: cbf47229-e7f9-4fd3-a921-008d79bc48cf

Details

ARN
arn:aws:sns:eu-west-1:590183699263:Serverless_architecture:cbf47229-e7f9-4fd3-a921-008d79bc48cf

Endpoint
arn:aws:lambda:eu-west-1:590183699263:function:inventory-manag-dev

Topic
Serverless_architecture

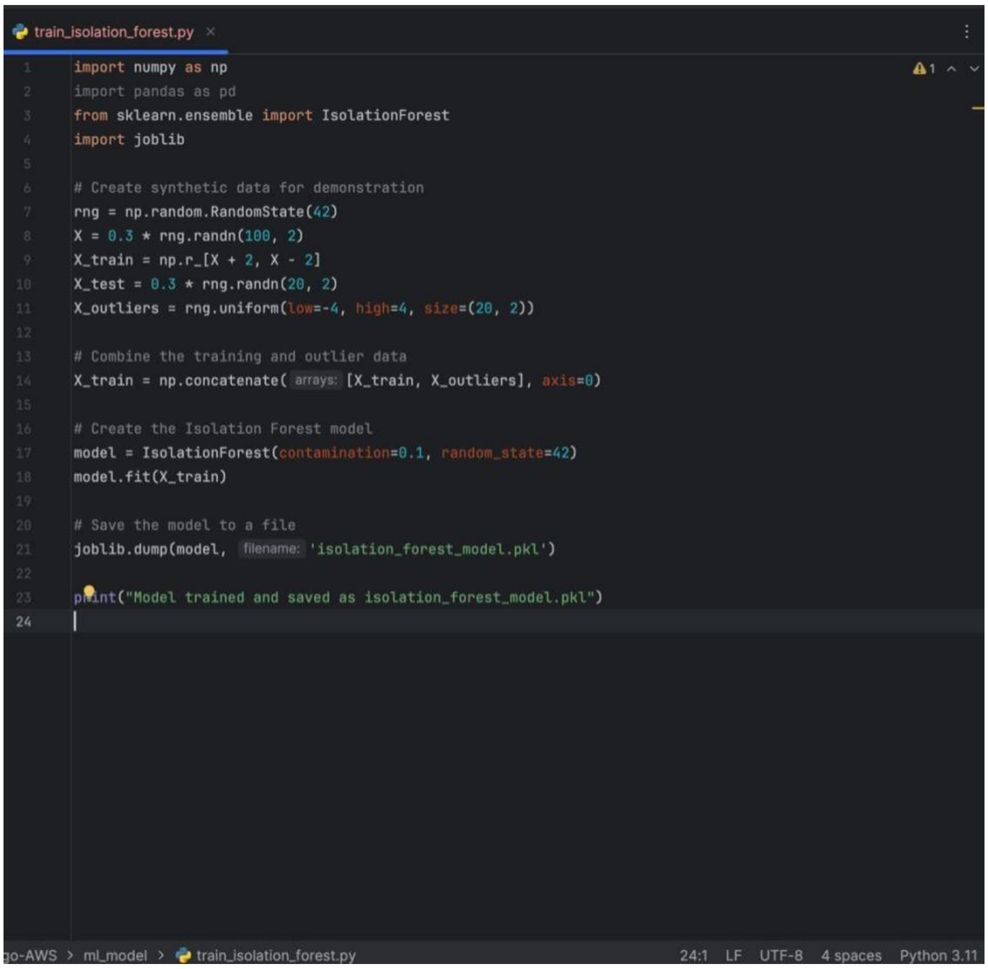
Subscription Principal
arn:aws:iam::590183699263:user/demo-user

Status
Confirmed

Protocol
LAMBDA

Fig12 . SNS

## 3.2.7 Creation of ML model



train_isolation_forest.py

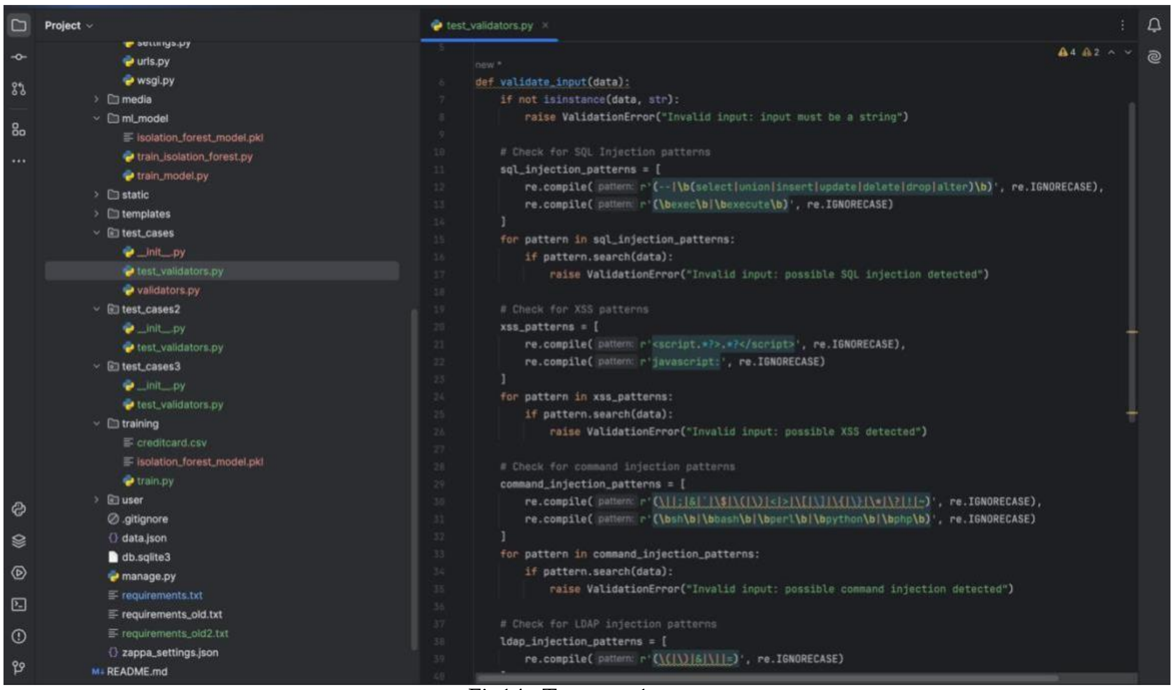
```python
import numpy as np
import pandas as pd
from sklearn.ensemble import IsolationForest
import joblib

# Create synthetic data for demonstration
rng = np.random.RandomState(42)
X = 0.3 * rng.randn(100, 2)
X_train = np.r_[X + 2, X - 2]
X_test = 0.3 * rng.randn(20, 2)
X_outliers = rng.uniform(low=-4, high=4, size=(20, 2))

# Combine the training and outlier data
X_train = np.concatenate([X_train, X_outliers], axis=0)

# Create the Isolation Forest model
model = IsolationForest(contamination=0.1, random_state=42)
model.fit(X_train)

# Save the model to a file
joblib.dump(model, 'isolation_forest_model.pkl')

print("Model trained and saved as isolation_forest_model.pkl")
```

go-AWS > ml_model > train_isolation_forest.py    24:1  LF  UTF-8  4 spaces  Python 3.11

Fig13 . Model

## Test Cases Used:

Test Case 1:



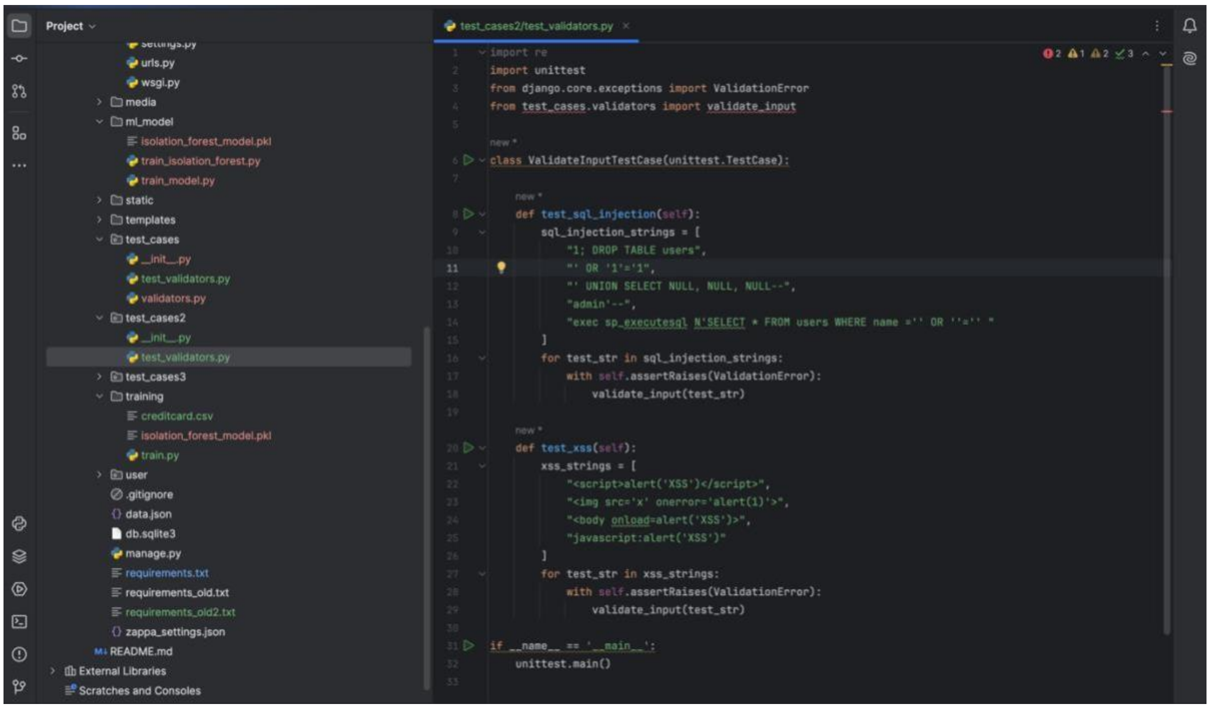
```
Project
    settings.py
    urls.py
    wsgi.py
  media
  ml_model
    isolation_forest_model.pkl
    train_isolation_forest.py
    train_model.py
  static
  templates
  test_cases
    __init__.py
    test_validators.py
    validators.py
  test_cases2
    __init__.py
    test_validators.py
  test_cases3
    __init__.py
    test_validators.py
  training
    creditcard.csv
    isolation_forest_model.pkl
    train.py
  user
  .gitignore
  data.json
  db.sqlite3
  manage.py
  requirements.txt
  requirements_old.txt
  requirements_old2.txt
  zappa_settings.json
  README.md
```

test_validators.py

```python
def validate_input(data):
    if not isinstance(data, str):
        raise ValidationError("Invalid input: input must be a string")

    # Check for SQL Injection patterns
    sql_injection_patterns = [
        re.compile(r'(--|\b(select|union|insert|update|delete|drop|alter)\b)', re.IGNORECASE),
        re.compile(r'(\bexec\b|\bexecute\b)', re.IGNORECASE)
    ]
    for pattern in sql_injection_patterns:
        if pattern.search(data):
            raise ValidationError("Invalid input: possible SQL injection detected")

    # Check for XSS patterns
    xss_patterns = [
        re.compile(r'<script.*?>.*?</script>', re.IGNORECASE),
        re.compile(r'javascript:', re.IGNORECASE)
    ]
    for pattern in xss_patterns:
        if pattern.search(data):
            raise ValidationError("Invalid input: possible XSS detected")

    # Check for command injection patterns
    command_injection_patterns = [
        re.compile(r'(\||;|&|`|\$|\(|\)|<|>|\[|\]|\{|\}|\*|\?|!|~)', re.IGNORECASE),
        re.compile(r'(\bsh\b|\bbash\b|\bperl\b|\bpython\b|\bphp\b)', re.IGNORECASE)
    ]
    for pattern in command_injection_patterns:
        if pattern.search(data):
            raise ValidationError("Invalid input: possible command injection detected")

    # Check for LDAP injection patterns
    ldap_injection_patterns = [
        re.compile(r'(\(|\)|&|\||=)', re.IGNORECASE)
```

Fig14 . Test case 1

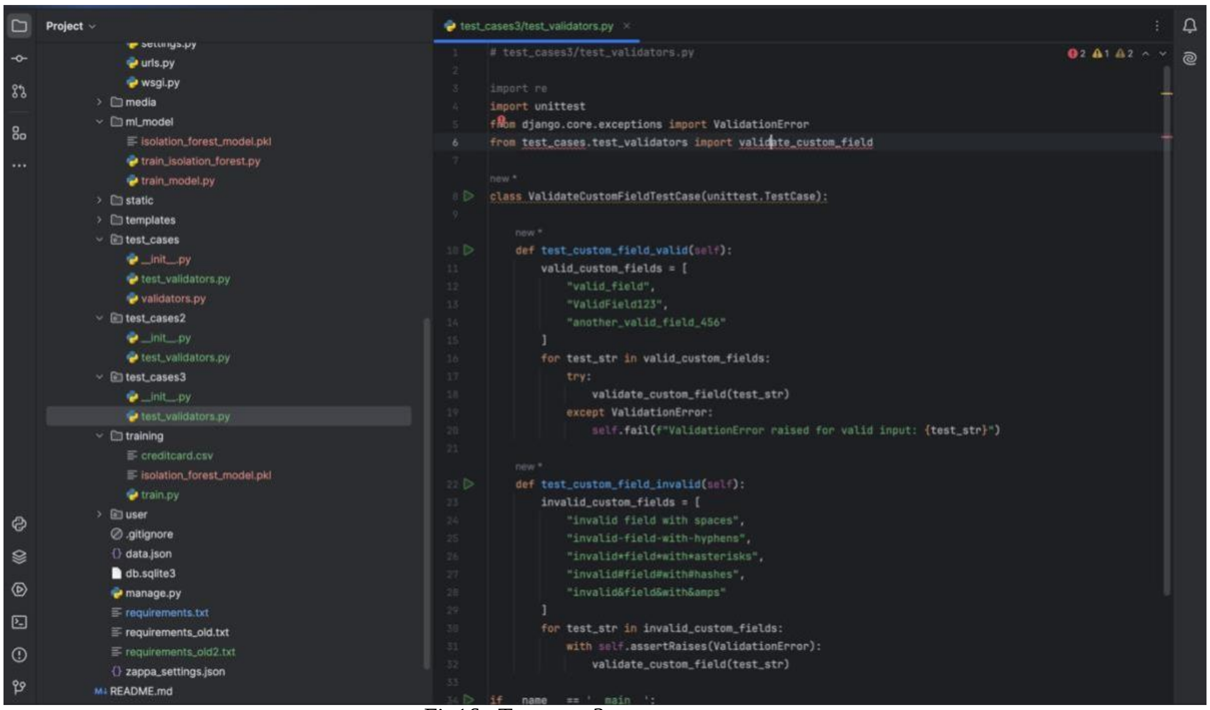
## Test Case 2



```python
import re
import unittest
from django.core.exceptions import ValidationError
from test_cases.validators import validate_input

class ValidateInputTestCase(unittest.TestCase):

    def test_sql_injection(self):
        sql_injection_strings = [
            "1; DROP TABLE users",
            "' OR '1'='1",
            "' UNION SELECT NULL, NULL, NULL--",
            "admin'--",
            "exec sp_executesql N'SELECT * FROM users WHERE name ='' OR ''='' "
        ]
        for test_str in sql_injection_strings:
            with self.assertRaises(ValidationError):
                validate_input(test_str)

    def test_xss(self):
        xss_strings = [
            "<script>alert('XSS')</script>",
            "<img src='x' onerror='alert(1)'>",
            "<body onload=alert('XSS')>",
            "javascript:alert('XSS')"
        ]
        for test_str in xss_strings:
            with self.assertRaises(ValidationError):
                validate_input(test_str)

if __name__ == '__main__':
    unittest.main()
```

Fig15 . Test case 2

## Test Case 3



```python
# test_cases3/test_validators.py

import re
import unittest
from django.core.exceptions import ValidationError
from test_cases.test_validators import validate_custom_field

class ValidateCustomFieldTestCase(unittest.TestCase):

    def test_custom_field_valid(self):
        valid_custom_fields = [
            "valid_field",
            "ValidField123",
            "another_valid_field_456"
        ]
        for test_str in valid_custom_fields:
            try:
                validate_custom_field(test_str)
            except ValidationError:
                self.fail(f"ValidationError raised for valid input: {test_str}")

    def test_custom_field_invalid(self):
        invalid_custom_fields = [
            "invalid field with spaces",
            "invalid-field-with-hyphens",
            "invalid*field*with*asterisks",
            "invalid#field#with#hashes",
            "invalid&field&with&amps"
        ]
        for test_str in invalid_custom_fields:
            with self.assertRaises(ValidationError):
                validate_custom_field(test_str)

if __name__ == '__main__':
```

Fig16 . Test case 3

## References

AWS (2018). *Amazon CloudWatch - Application and Infrastructure Monitoring*. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/cloudwatch/.