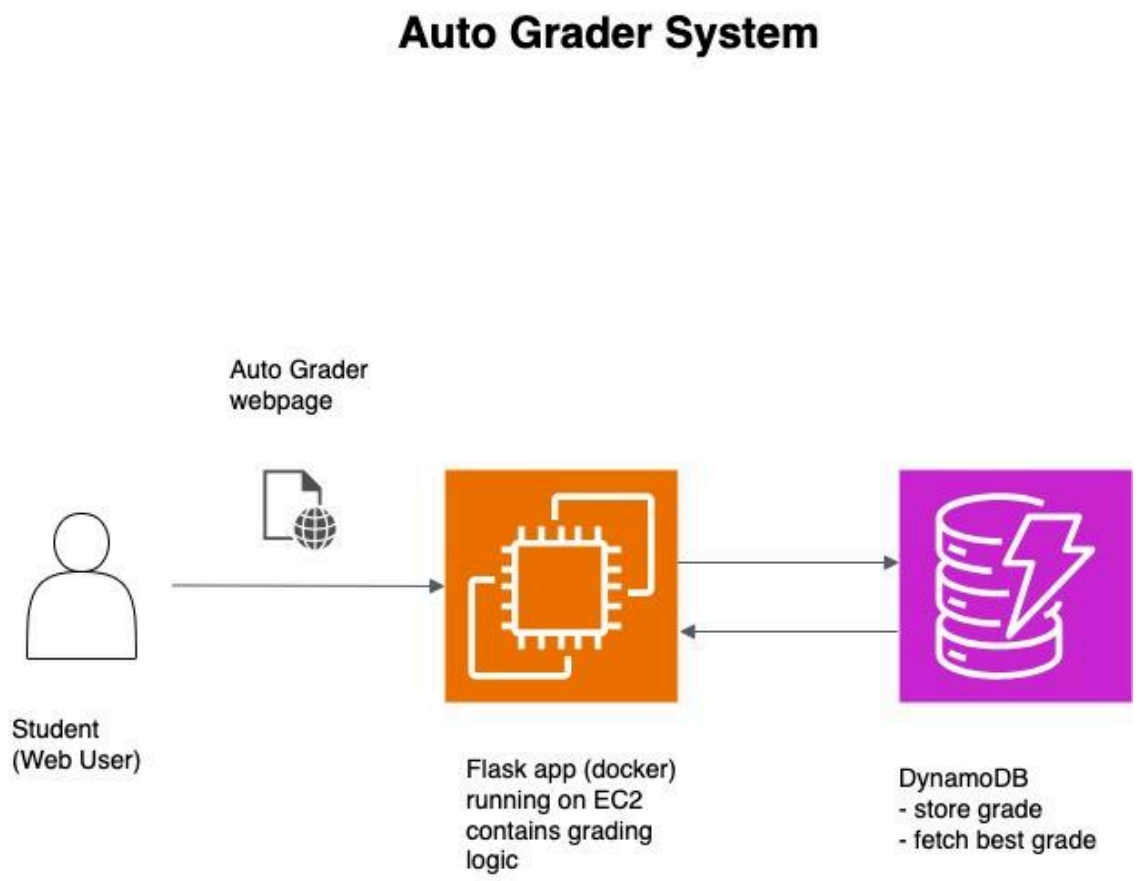
Auto Grader System

Auto grader is a Flask application that provides a coding environment to run coding solutions for given a problem. It presents a webpage to enter ‘Student ID’ and ‘Email’. Validation checks for student id to contain only numbers and email to contain ‘sjsu.edu’ domain. After successful entries of student id and email, ‘Knight Attack’ problem is presented with a code template in the space provided to enter code with syntax highlighting (CodeMirror, CSS). There is also an option to upload a .py file containing the solution. In case of error, the traceback is sent back to the browser.

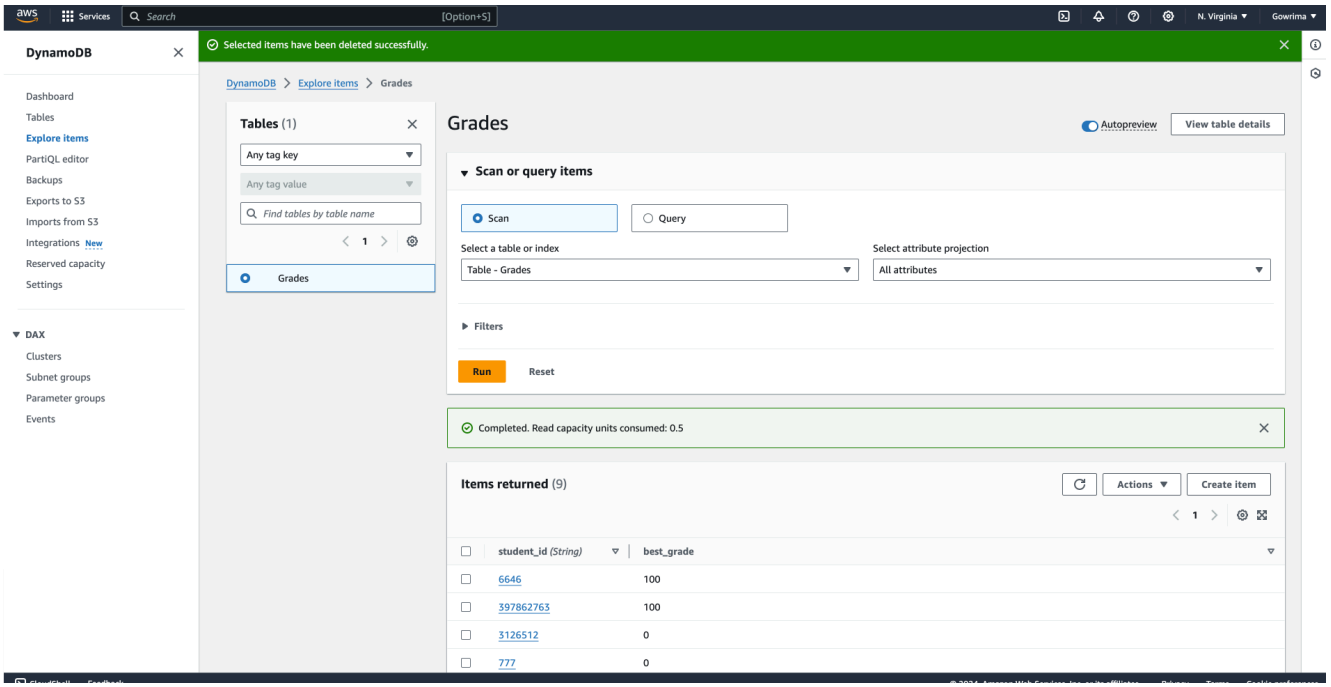


High level architecture

Auto grader URL
Auto grader can be accessed at http://52.73.2.7/

APIs
validate_email()
run_code()
execute_code(student_id, email, code)

DynamoDB:



Docker:

Install docker on EC2 instance

```
sudo yum update -y
sudo yum install -y docker
```

Start docker service

```
sudo service docker start
```

Build auto grader flask app and push it to dockerhub

```
docker build -t auto-grader:1.0 .
docker tag auto-grader:1.0 kjgowrima517/auto-grader:1.0
docker push kjgowrima517/auto-grader:1.0
```

Run

```
sudo docker run -p 80:5000 kjgowrima517/auto-grader:1.0
```

Results:
Student ID: 397862763 Email: gg@sjsu.edu test_01 [FAIL] 0ms test_02 [FAIL] 0ms test_03 [FAIL] 0ms test_04 [FAIL] 0ms test_05 [FAIL] 0ms test_06 [FAIL] 0ms test_07 [FAIL] 0ms test_08 [FAIL] 0ms > 0/8 tests passed Grade: 0 Best grade: 0

Student ID: 397862763 Email: gg@sjsu.edu test_01 [PASS] 0ms test_02 [PASS] 0ms test_03 [PASS] 0ms test_04 [PASS] 0ms test_05 [PASS] 2ms test_06 [PASS] 5ms test_07 [PASS] 0ms test_08 [PASS] 0ms > 8/8 tests passed Grade: 100 Best grade: 100

Student ID: 397862763 Email: gg@sjsu.edu test_01 [FAIL] 0ms test_02 [FAIL] 0ms test_03 [FAIL] 0ms test_04 [FAIL] 0ms test_05 [FAIL] 0ms test_06 [FAIL] 0ms test_07 [FAIL] 0ms test_08 [PASS] 0ms > 1/8 tests passed Grade: 0 Best grade: 100

Code:

```
from flask import Flask, request, render_template_string, jsonify
import requests
import subprocess
import boto3
from time import time
from botocore.exceptions import ClientError
from boto3.dynamodb.conditions import Key
import sys
import os
import traceback

app = Flask(__name__)
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # Limit file size
to 16MB
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
table_name = 'Grades'
table = dynamodb.Table(table_name)

@app.route('/')
def home():
    return render_template_string('''
        <style>
            .container {
                max-width: 600px;
                margin: 50px auto;
                font-family: Arial, sans-serif;
            }
            h1 {
```

```
font-size: 36px;
                text-align: center;
            }
            .form-group {
                margin: 15px 0;
            }
            label {
                display: block;
                margin-bottom: 5px;
            }
            input[type="text"] {
                width: 100%;
                padding: 8px;
                box-sizing: border-box;
            }
```

```
            input[type="file"] {
                width: 100%;
                padding: 8px;
                box-sizing: border-box;
            }
            button {
                display: block;
                width: 100%;
                padding: 10px;
                background-color: #007BFF;
                color: white;
                border: none;
                cursor: pointer;
                font-size: 16px;
            }
            button:hover {
                background-color: #0056b3;
            }
```

```
</style>
        <div class="container">
            <h1>Auto Grader</h1>
            <form action="/validate" method="post">
                <div class="form-group">
                    <label for="student_id">Student ID</label>
                    <input type="text" id="student_id" name="student_id"
required>
                </div>
                <div class="form-group">
                    <label for="email">Email</label>
                    <input type="text" id="email" name="email" required>
                </div>
                <button type="submit">Submit</button>
            </form>
        </div>
    ''')

@app.route('/validate', methods=['POST'])
def validate_email():
    student_id = request.form['student_id']
    email = request.form['email']

    # Validate student ID to contain only numbers
    if not student_id.isdigit():
        return 'Invalid student ID. Please use numbers only.'
```

```
    if email.endswith('@sjsu.edu'):
        return render_template_string('''
            <style>
                .problem-statement {
                    font-size: 16px;
                    margin: 20px;
                    line-height: 1.6;
                }
                .problem-statement h1 {
                    font-size: 24px;
                }
            </style>
            <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.7/codemirro
r.min.css">
            <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.7/codemirror
.min.js"></script>
            <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.7/mode/pytho
n/python.min.js"></script>
```

```
<form action="/run" method="post" enctype="multipart/form-data">
                <div class="problem-statement">
                    <h1>Knight Attack</h1>
                    <p>
                        A knight and a pawn are on a chess board. Can
you figure out the minimum number of moves required for the knight to
travel to the same position of the pawn?
                        On a single move, the knight can move in an "L"
shape; two spaces in any direction, then one space in a perpendicular
direction.
                        This means that on a single move, a knight has
eight possible positions it can move to.
                    </p>
                    <p>
                        Write a function, <code>knight_attack</code>,
that takes in 5 arguments: <code>n, kr, kc, pr, pc</code><br>
                        <code>n</code> = the length of the chess
board<br>
                        <code>kr</code> = the starting row of the
knight<br>
                        <code>kc</code> = the starting column of the
knight<br>
```

```
                        <code>pr</code> = the row of the pawn<br>
                        <code>pc</code> = the column of the pawn<br>
                        The function should return a number representing
the minimum number of moves required for the knight to land on top of
the pawn.
                        The knight cannot move out of bounds of the
board. You can assume that rows and columns are 0-indexed.
                        This means that if <code>n = 8</code>, there are
8 rows and 8 columns numbered 0 to 7.
                        If it is not possible for the knight to attack
the pawn, then return None.
                    </p>
                </div>
                <input type="hidden" name="student_id" value="{{
student_id }}">
                <input type="hidden" name="email" value="{{ email }}">
                <textarea id="code" name="code" rows="20" cols="70">def
knight_attack(n, kr, kc, pr, pc):\n    # Your code here\n
pass</textarea><br>
                <label for="file">or upload a .py file:</label>
                <input type="file" id="file" name="file"><br>
                <button type="submit">Submit</button>
            </form>
            <script>
                var editor =
CodeMirror.fromTextArea(document.getElementById("code"), {
                    lineNumbers: true,
                    mode: "python",
```

```
 theme: "default"
                });
                editor.setSize("100%", "400px");
            </script>
        ''', student_id=student_id, email=email)
    else:
        return 'Invalid email. Please use an @sjsu.edu email.'

@app.route('/run', methods=['POST'])
def run_code():
    student_id = request.form['student_id']
    email = request.form['email']
    code = request.form['code']

    if not student_id.isdigit():
        return 'Invalid student ID. It should only contain numbers.'
```

```
    if '@sjsu.edu' not in email:
        return 'Invalid email. It should be an SJSU email.'

    if 'file' in request.files and request.files['file'].filename != '':
        file = request.files['file']
        if file.filename.endswith('.py'):
            code = file.read().decode('utf-8')
```

```
else:
            return 'Invalid file type. Please upload a .py file.'

    result = execute_code(student_id, email, code)
    return result

def execute_code(student_id, email, code):
    results = []
    abort = False

    test_cases = [
        ('test_01', 'assert knight_attack(8, 1, 1, 2, 2) == 2'),
        ('test_02', 'assert knight_attack(8, 1, 1, 2, 3) == 1'),
        ('test_03', 'assert knight_attack(8, 0, 3, 4, 2) == 3'),
        ('test_04', 'assert knight_attack(8, 0, 3, 5, 2) == 4'),
        ('test_05', 'assert knight_attack(24, 4, 7, 19, 20) == 10'),
        ('test_06', 'assert knight_attack(100, 21, 10, 0, 0) == 11'),
        ('test_07', 'assert knight_attack(3, 0, 0, 1, 2) == 1'),
        ('test_08', 'assert knight_attack(3, 0, 0, 1, 1) is None')
    ]

    exec_globals = {}
    try:
        exec(code, exec_globals)
        if 'knight_attack' not in exec_globals:
            raise ValueError("Function 'knight_attack' is not defined in
the submitted code.")
    except Exception as e:
        return handle_error(student_id, email, 0, f"Code execution
error: {str(e)}", traceback.format_exc())
```

```
for test_name, test_case in test_cases:
        start_time = time()
        try:
            exec(test_case, exec_globals)
```

```
            duration = int((time() - start_time) * 1000)
            if duration > 2000:
                return handle_error(student_id, email, 0, "Test
execution time exceeded 2 seconds", "")
            results.append((test_name, 'PASS', duration))
        except Exception as e:
            duration = int((time() - start_time) * 1000)
            if duration > 2000:
                return handle_error(student_id, email, 0, "Test
execution time exceeded 2 seconds", "")
            results.append((test_name, 'FAIL', duration, str(e)))

    passed_tests = sum(1 for result in results if result[1] == 'PASS')
    total_tests = len(test_cases)
    grade = (passed_tests // total_tests) * 100

    fetch_grade = -1
    try:
        response =
table.query(KeyConditionExpression=Key('student_id').eq(student_id))
        if 'Item' in response:
            fetch_grade =response['Item'].get('best_grade', 0)
    except ClientError as e:
        return {
            'statusCode': 500,
            'error': f"Failed to retrieve previous best grade:
{str(e)}",
            'traceback': traceback.format_exc()
        }

    if grade > fetch_grade and fetch_grade != -1:
        try:
            table.update_item(Key={
                'student_id': student_id},
                UpdateExpression='SET best_grade = :bg',
                ExpressionAttributeValues={
                    ':bg' : grade
                }
            )
        except ClientError as e:
            return {
                'statusCode': 500,
```

```
'error': f"Failed to update results in DynamoDB: {str(e)}",
                'traceback': traceback.format_exc()
```

```
            }
    elif fetch_grade == -1:
        try:
            table.put_item(Item={
                'student_id' : student_id,
                'best_grade' : grade
                })
        except ClientError as e:
            return {
                'statusCode' : 500,
                'error' : f"Failed to store results in DynamoDB:
{str(e)}",
                'traceback':traceback.format_exc()
            }

    response_results = f"Student ID: {student_id}\nEmail: {email}\n"
    response_results += "\n".join([f"{result[0]} [{result[1]}]
{result[2]}ms" for result in results]) + "\n"
    response_summary = f"\n> {passed_tests}/{total_tests} tests
passed\nGrade: {grade}\nBest grade: {max(grade, fetch_grade)}"

    return response_results + "\n" + response_summary

def handle_error(student_id, email, grade, error_message, trace):
    try:
        table.put_item(Item={
            'student_id': student_id,
            'email': email,
            'best_grade': grade,
            'error': error_message,
            'traceback': trace
        })
    except ClientError as e:
        return {
            'statusCode': 500,
```

```
'error': f"Failed to store error results in DynamoDB: {str(e)}",
            'traceback': traceback.format_exc()
        }
    return {
        'statusCode': 400,
        'error': error_message,
        'traceback': trace
    }
```

```
def format_results(results):
    return [{'test_name': result[0], 'status': result[1], 'duration_ms':
result[2]} for result in results]

if __name__ == '__main__':
    publicIP = sys.argv[1]
    publicIP = publicIP.strip()
    print (publicIP)
    app.run(debug=True, host=os.getenv('IP', '0.0.0.0'),
port=(int(os.getenv('PORT', 5000))))
```