AI – Assisted Coding

END LAB TEST

Name: Adepu Goutham Sri Bhargav **Roll.no:** 2403A510C4

Batch: 05 **Date:** 24 – 11 – 2025

SET – 2: Ethical AI Practices in Student Data Handling

Q1: Protect student PII in LMS logs

• **Task 1:** Use AI to identify possible privacy violations in a given code snippet.

• **Task 2:** Modify the snippet using AI suggestions to implement masking or hashing.

Q2: Bias mitigation in recommendation engine

• **Task 1:** Ask AI to detect biased logic in course recommendation rules.

• **Task 2:** Refactor code with fairness constraints

Q1. PROMPT:

Task – 1:

“Analyze this LMS logging code for PII exposure and privacy violations:

[ENTER SNIPPET]

Identify what student data is exposed and shouldn't be logged in plain text.”

Task – 2:

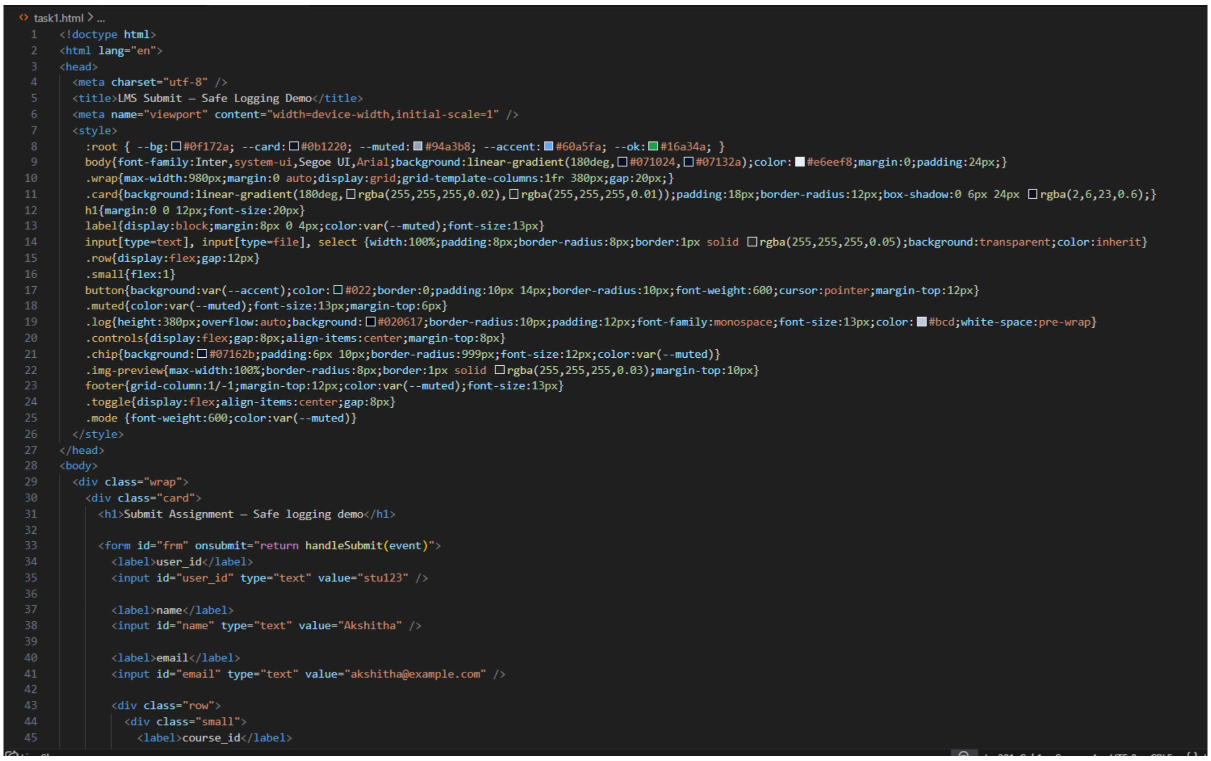
“Refactor this code to mask/hash all student PII using appropriate methods:

[ENTER SNIPPET]

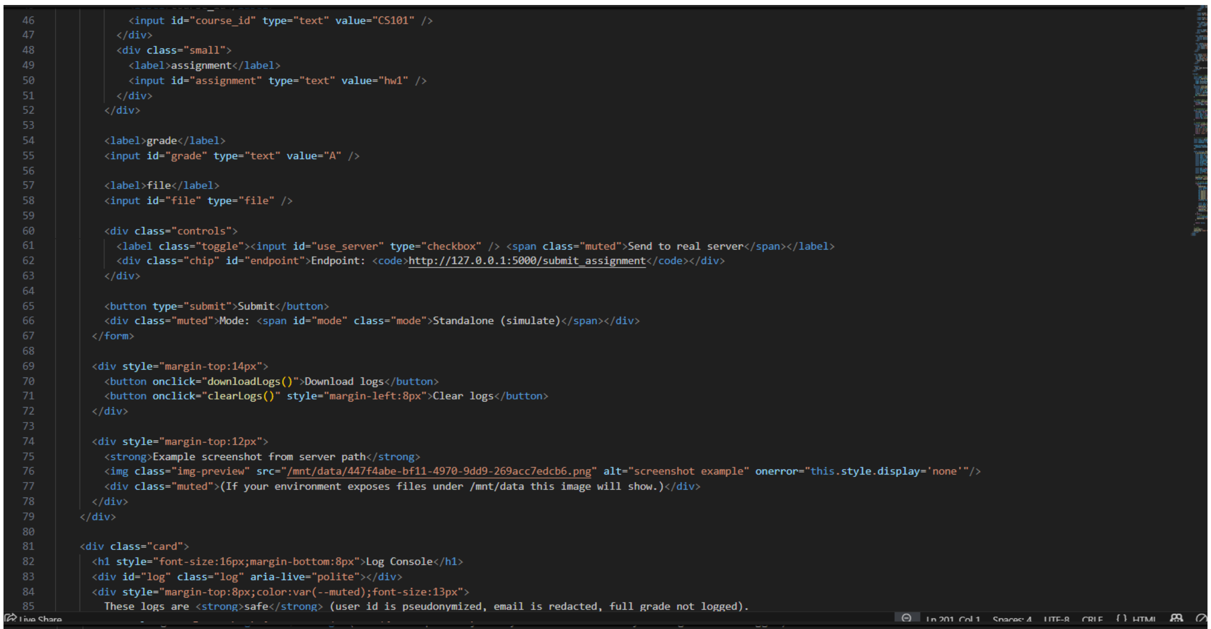
Use SHA-256 for hashing and partial masking where needed. Add comments.”

CODE:

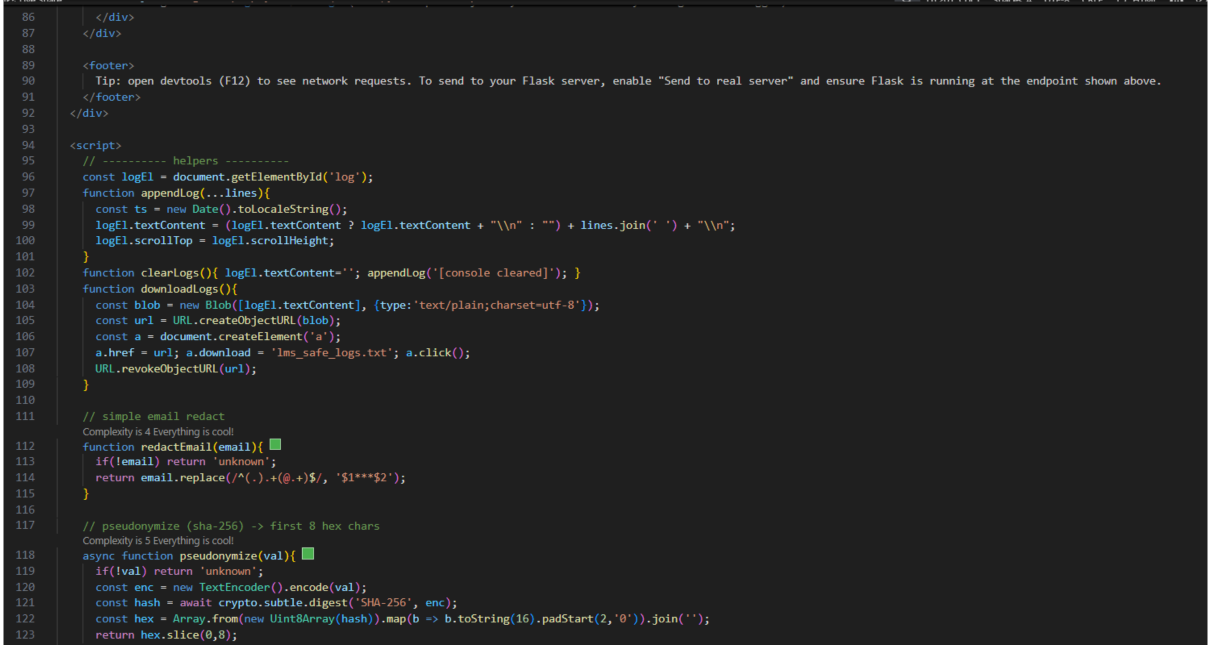
Task – 1:


task1.html > ...

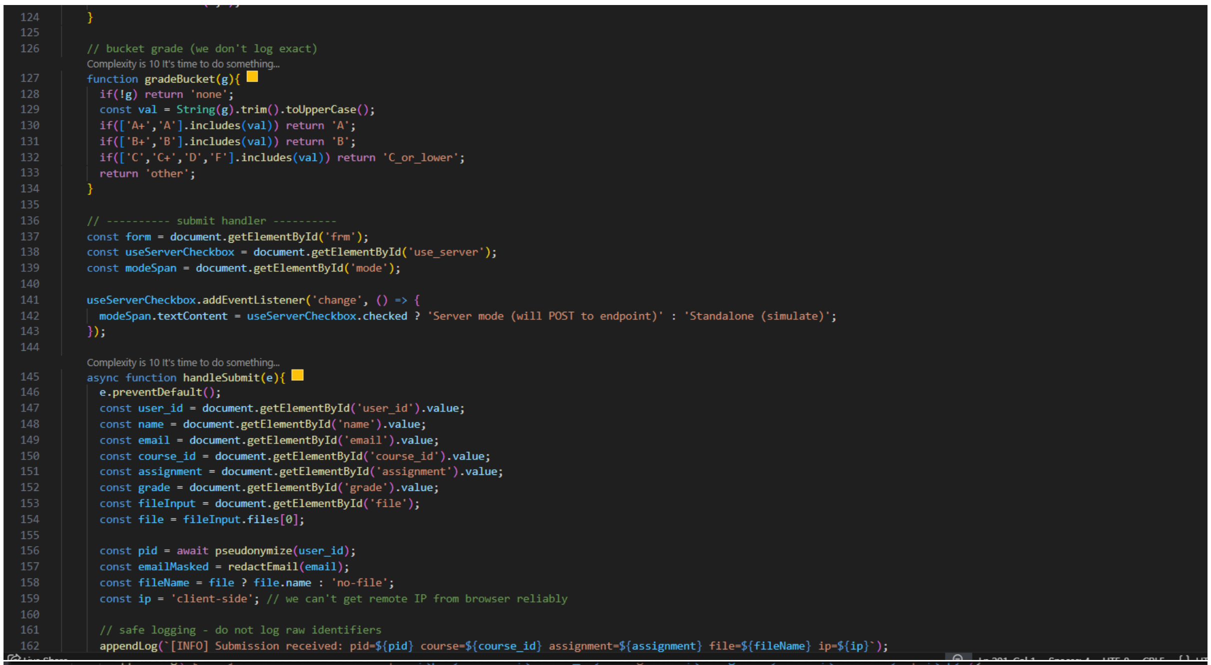
```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>LMS Submit — Safe Logging Demo</title>
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <style>
    :root { --bg:#0f172a; --card:#0b1220; --muted:#94a3b8; --accent:#60a5fa; --ok:#16a34a; }
    body{font-family:Inter,system-ui,Segoe UI,Arial;background:linear-gradient(180deg,#071024,#07132a);color:#e6eef8;margin:0;padding:24px;}
    .wrap{max-width:980px;margin:0 auto;display:grid;grid-template-columns:1fr 380px;gap:20px;}
    .card{background:linear-gradient(180deg,rgba(255,255,255,0.02),rgba(255,255,255,0.01));padding:18px;border-radius:12px;box-shadow:0 6px 24px rgba(2,6,23,0.6);}
    h1{margin:0 0 12px;font-size:20px}
    label{display:block;margin:8px 0 4px;color:var(--muted);font-size:13px}
    input[type=text], input[type=file], select {width:100%;padding:8px;border-radius:8px;border:1px solid rgba(255,255,255,0.05);background:transparent;color:inherit}
    .row{display:flex;gap:12px}
    .small{flex:1}
    button{background:var(--accent);color:#022;border:0;padding:10px 14px;border-radius:10px;font-weight:600;cursor:pointer;margin-top:12px}
    .muted{color:var(--muted);font-size:13px;margin-top:6px}
    .log{height:380px;overflow:auto;background:#020617;border-radius:10px;padding:12px;font-family:monospace;font-size:13px;color:#bcd;white-space:pre-wrap}
    .controls{display:flex;gap:8px;align-items:center;margin-top:8px}
    .chip{background:#07162b;padding:6px 10px;border-radius:999px;font-size:12px;color:var(--muted)}
    .img-preview{max-width:100%;border-radius:8px;border:1px solid rgba(255,255,255,0.03);margin-top:10px}
    footer{grid-column:1/-1;margin-top:12px;color:var(--muted);font-size:13px}
    .toggle{display:flex;align-items:center;gap:8px}
    .mode {font-weight:600;color:var(--muted)}
  </style>
</head>
<body>
  <div class="wrap">
    <div class="card">
      <h1>Submit Assignment — Safe logging demo</h1>

      <form id="frm" onsubmit="return handleSubmit(event)">
        <label>user_id</label>
        <input id="user_id" type="text" value="stu123" />

        <label>name</label>
        <input id="name" type="text" value="Akshitha" />

        <label>email</label>
        <input id="email" type="text" value="akshitha@example.com" />

        <div class="row">
          <div class="small">
            <label>course_id</label>
```



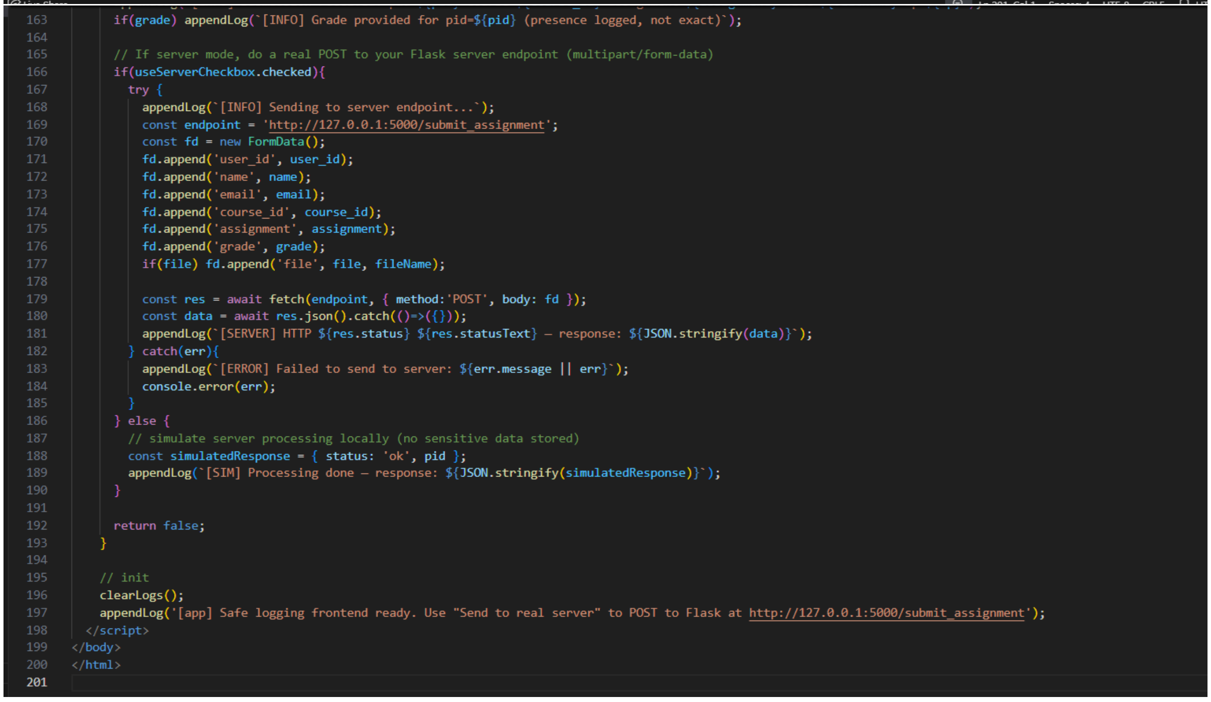
```html
          <input id="course_id" type="text" value="CS101" />
        </div>
        <div class="small">
          <label>assignment</label>
          <input id="assignment" type="text" value="hw1" />
        </div>
      </div>

      <label>grade</label>
      <input id="grade" type="text" value="A" />

      <label>file</label>
      <input id="file" type="file" />

      <div class="controls">
        <label class="toggle"><input id="use_server" type="checkbox" /> <span class="muted">Send to real server</span></label>
        <div class="chip" id="endpoint">Endpoint: <code>http://127.0.0.1:5000/submit_assignment</code></div>
      </div>

      <button type="submit">Submit</button>
      <div class="muted">Mode: <span id="mode" class="mode">Standalone (simulate)</span></div>
    </form>

    <div style="margin-top:14px">
      <button onclick="downloadLogs()">Download logs</button>
      <button onclick="clearLogs()" style="margin-left:8px">Clear logs</button>
    </div>

    <div style="margin-top:12px">
      <strong>Example screenshot from server path</strong>
      <img class="img-preview" src="/mnt/data/447f4abe-bf11-4970-9dd9-269acc7edcb6.png" alt="screenshot example" onerror="this.style.display='none'"/>
      <div class="muted">(If your environment exposes files under /mnt/data this image will show.)</div>
    </div>
  </div>

  <div class="card">
    <h1 style="font-size:16px;margin-bottom:8px">Log Console</h1>
    <div id="log" class="log" aria-live="polite"></div>
    <div style="margin-top:8px;color:var(--muted);font-size:13px">
      These logs are <strong>safe</strong> (user id is pseudonymized, email is redacted, full grade not logged).
```



```html
    </div>
  </div>

  <footer>
    Tip: open devtools (F12) to see network requests. To send to your Flask server, enable "Send to real server" and ensure Flask is running at the endpoint shown above.
  </footer>
</div>

<script>
  // ---------- helpers ----------
  const logEl = document.getElementById('log');
  function appendLog(...lines){
    const ts = new Date().toLocaleString();
    logEl.textContent = (logEl.textContent ? logEl.textContent + "\\n" : "") + lines.join(' ') + "\\n";
    logEl.scrollTop = logEl.scrollHeight;
  }
  function clearLogs(){ logEl.textContent=''; appendLog('[console cleared]'); }
  function downloadLogs(){
    const blob = new Blob([logEl.textContent], {type:'text/plain;charset=utf-8'});
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url; a.download = 'lms_safe_logs.txt'; a.click();
    URL.revokeObjectURL(url);
  }

  // simple email redact
  Complexity is 4 Everything is cool!
  function redactEmail(email){
    if(!email) return 'unknown';
    return email.replace(/^(.).+(@.+)$/, '$1***$2');
  }

  // pseudonymize (sha-256) -> first 8 hex chars
  Complexity is 5 Everything is cool!
  async function pseudonymize(val){
    if(!val) return 'unknown';
    const enc = new TextEncoder().encode(val);
    const hash = await crypto.subtle.digest('SHA-256', enc);
    const hex = Array.from(new Uint8Array(hash)).map(b => b.toString(16).padStart(2,'0')).join('');
    return hex.slice(0,8);
```

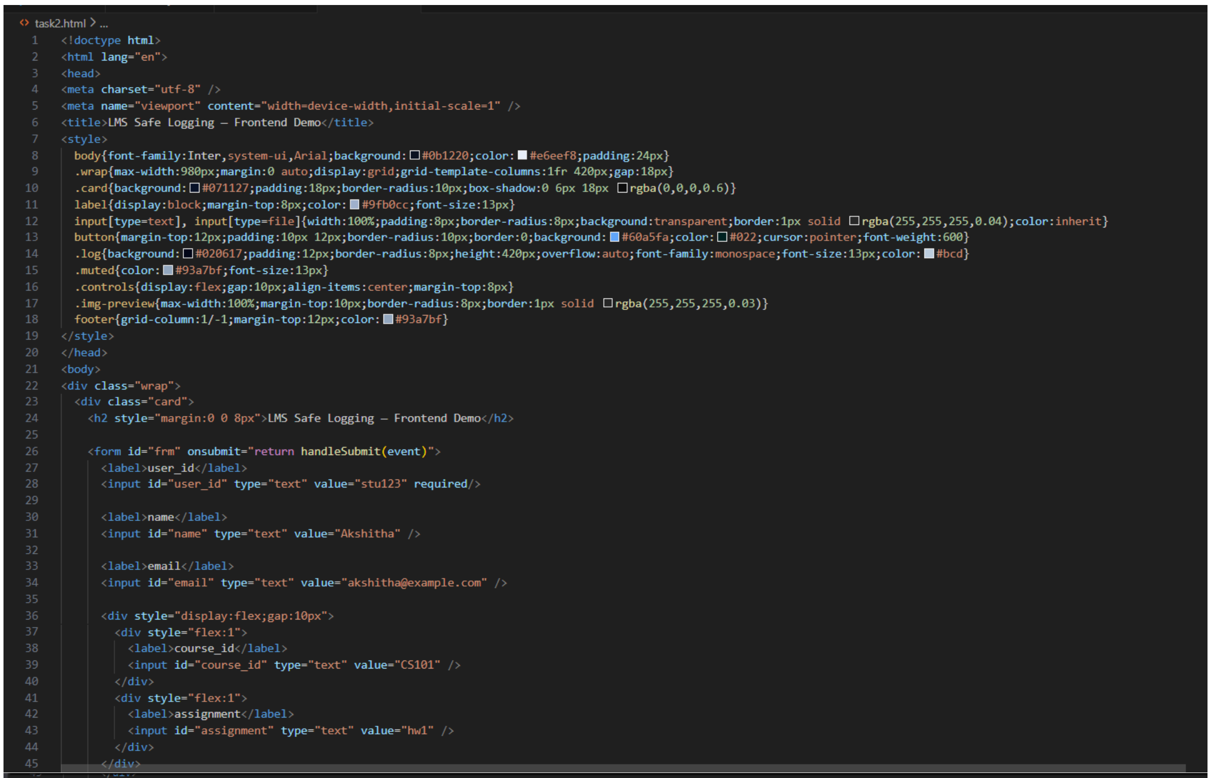


```
    }

    // bucket grade (we don't log exact)
    Complexity is 10 It's time to do something...
    function gradeBucket(g){
      if(!g) return 'none';
      const val = String(g).trim().toUpperCase();
      if(['A+','A'].includes(val)) return 'A';
      if(['B+','B'].includes(val)) return 'B';
      if(['C','C+','D','F'].includes(val)) return 'C_or_lower';
      return 'other';
    }

    // ---------- submit handler ----------
    const form = document.getElementById('frm');
    const useServerCheckbox = document.getElementById('use_server');
    const modeSpan = document.getElementById('mode');

    useServerCheckbox.addEventListener('change', () => {
      modeSpan.textContent = useServerCheckbox.checked ? 'Server mode (will POST to endpoint)' : 'Standalone (simulate)';
    });

    Complexity is 10 It's time to do something...
    async function handleSubmit(e){
      e.preventDefault();
      const user_id = document.getElementById('user_id').value;
      const name = document.getElementById('name').value;
      const email = document.getElementById('email').value;
      const course_id = document.getElementById('course_id').value;
      const assignment = document.getElementById('assignment').value;
      const grade = document.getElementById('grade').value;
      const fileInput = document.getElementById('file');
      const file = fileInput.files[0];

      const pid = await pseudonymize(user_id);
      const emailMasked = redactEmail(email);
      const fileName = file ? file.name : 'no-file';
      const ip = 'client-side'; // we can't get remote IP from browser reliably

      // safe logging - do not log raw identifiers
      appendLog(`[INFO] Submission received: pid=${pid} course=${course_id} assignment=${assignment} file=${fileName} ip=${ip}`);
```

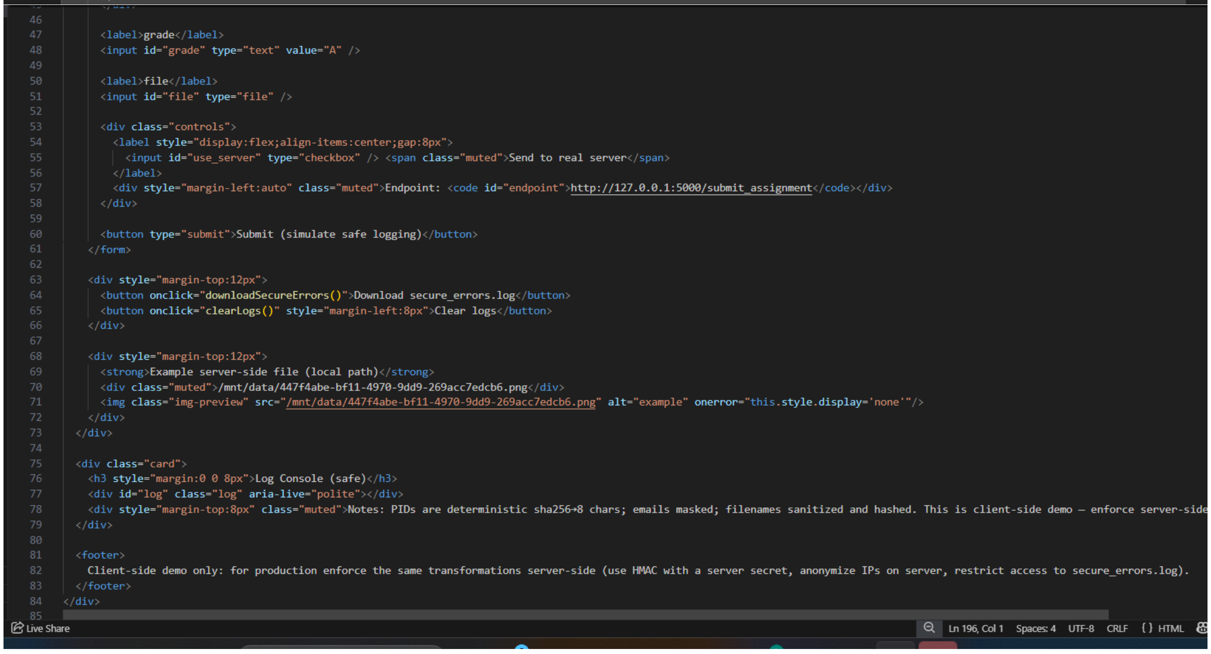


```
      if(grade) appendLog(`[INFO] Grade provided for pid=${pid} (presence logged, not exact)`);

      // If server mode, do a real POST to your Flask server endpoint (multipart/form-data)
      if(useServerCheckbox.checked){
        try {
          appendLog(`[INFO] Sending to server endpoint...`);
          const endpoint = 'http://127.0.0.1:5000/submit_assignment';
          const fd = new FormData();
          fd.append('user_id', user_id);
          fd.append('name', name);
          fd.append('email', email);
          fd.append('course_id', course_id);
          fd.append('assignment', assignment);
          fd.append('grade', grade);
          if(file) fd.append('file', file, fileName);

          const res = await fetch(endpoint, { method:'POST', body: fd });
          const data = await res.json().catch(()=>({}));
          appendLog(`[SERVER] HTTP ${res.status} ${res.statusText} — response: ${JSON.stringify(data)}`);
        } catch(err){
          appendLog(`[ERROR] Failed to send to server: ${err.message || err}`);
          console.error(err);
        }
      } else {
        // simulate server processing locally (no sensitive data stored)
        const simulatedResponse = { status: 'ok', pid };
        appendLog(`[SIM] Processing done — response: ${JSON.stringify(simulatedResponse)}`);
      }

      return false;
    }

    // init
    clearLogs();
    appendLog('[app] Safe logging frontend ready. Use "Send to real server" to POST to Flask at http://127.0.0.1:5000/submit_assignment');
  </script>
</body>
</html>
```

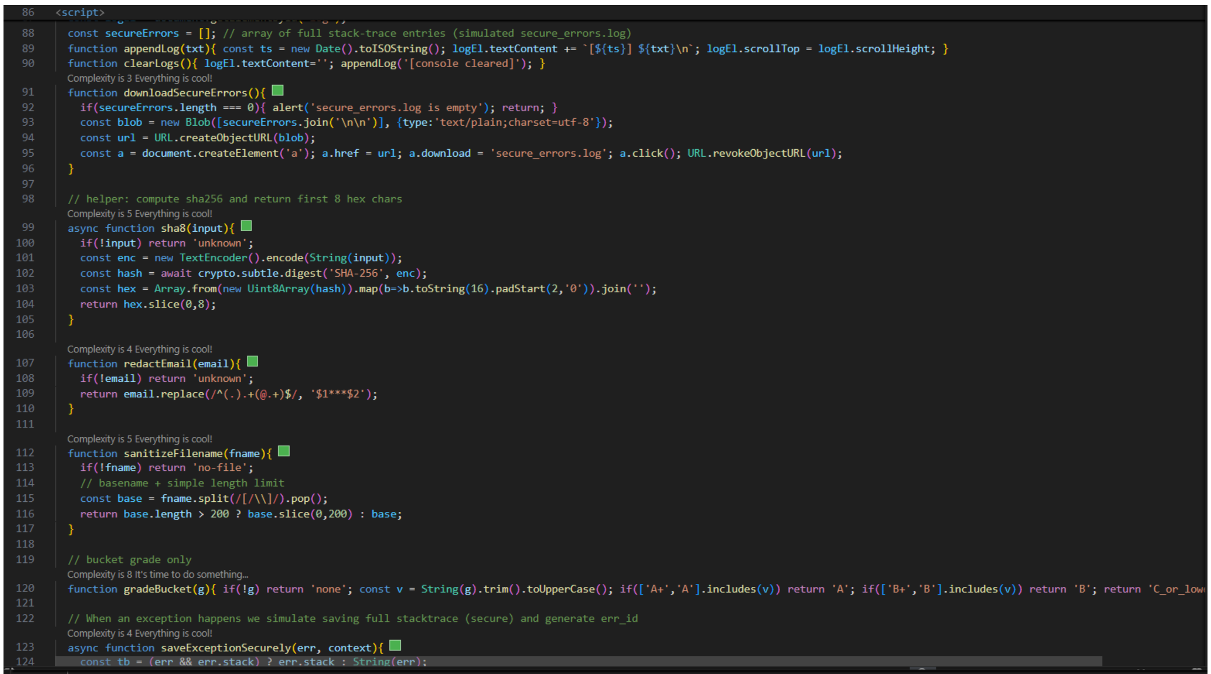
Task – 2:



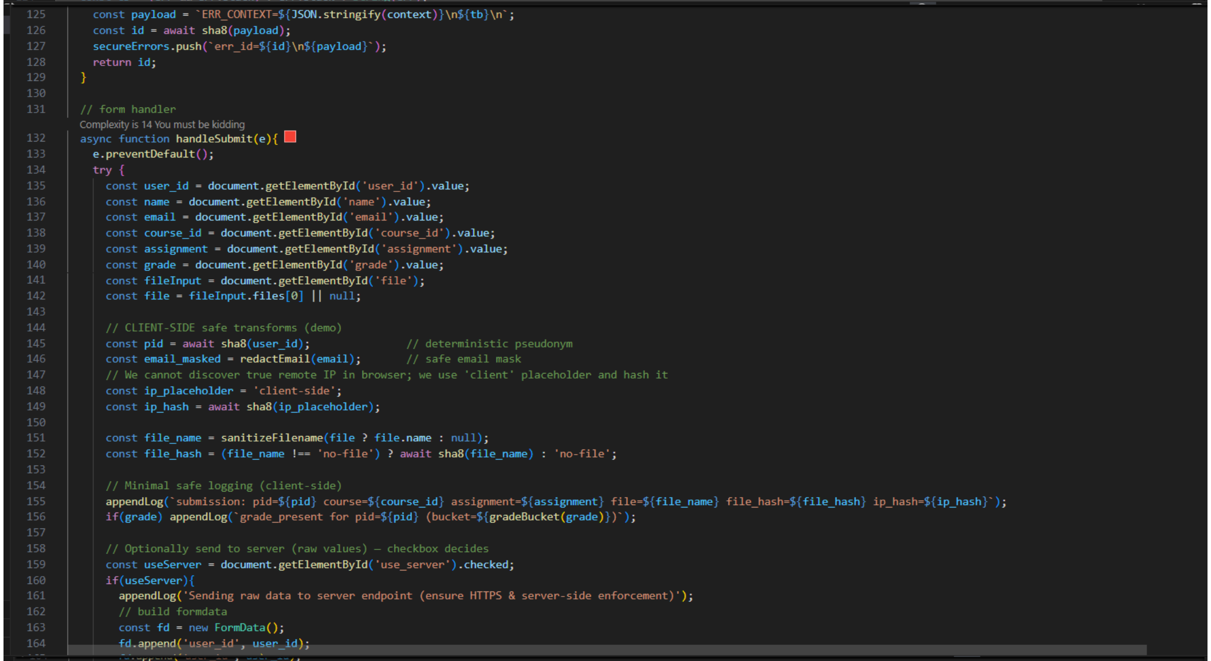
```html
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width,initial-scale=1" />
<title>LMS Safe Logging — Frontend Demo</title>
<style>
  body{font-family:Inter,system-ui,Arial;background:#0b1220;color:#e6eef8;padding:24px}
  .wrap{max-width:980px;margin:0 auto;display:grid;grid-template-columns:1fr 420px;gap:18px}
  .card{background:#071127;padding:18px;border-radius:10px;box-shadow:0 6px 18px rgba(0,0,0,0.6)}
  label{display:block;margin-top:8px;color:#9fb0cc;font-size:13px}
  input[type=text], input[type=file]{width:100%;padding:8px;border-radius:8px;background:transparent;border:1px solid rgba(255,255,255,0.04);color:inherit}
  button{margin-top:12px;padding:10px 12px;border-radius:10px;border:0;background:#60a5fa;color:#022;cursor:pointer;font-weight:600}
  .log{background:#020617;padding:12px;border-radius:8px;height:420px;overflow:auto;font-family:monospace;font-size:13px;color:#bcd}
  .muted{color:#93a7bf;font-size:13px}
  .controls{display:flex;gap:10px;align-items:center;margin-top:8px}
  .img-preview{max-width:100%;margin-top:10px;border-radius:8px;border:1px solid rgba(255,255,255,0.03)}
  footer{grid-column:1/-1;margin-top:12px;color:#93a7bf}
</style>
</head>
<body>
<div class="wrap">
  <div class="card">
    <h2 style="margin:0 0 8px">LMS Safe Logging — Frontend Demo</h2>

    <form id="frm" onsubmit="return handleSubmit(event)">
      <label>user_id</label>
      <input id="user_id" type="text" value="stu123" required/>

      <label>name</label>
      <input id="name" type="text" value="Akshitha" />

      <label>email</label>
      <input id="email" type="text" value="akshitha@example.com" />

      <div style="display:flex;gap:10px">
        <div style="flex:1">
          <label>course_id</label>
          <input id="course_id" type="text" value="CS101" />
        </div>
        <div style="flex:1">
          <label>assignment</label>
          <input id="assignment" type="text" value="hw1" />
        </div>
      </div>
```



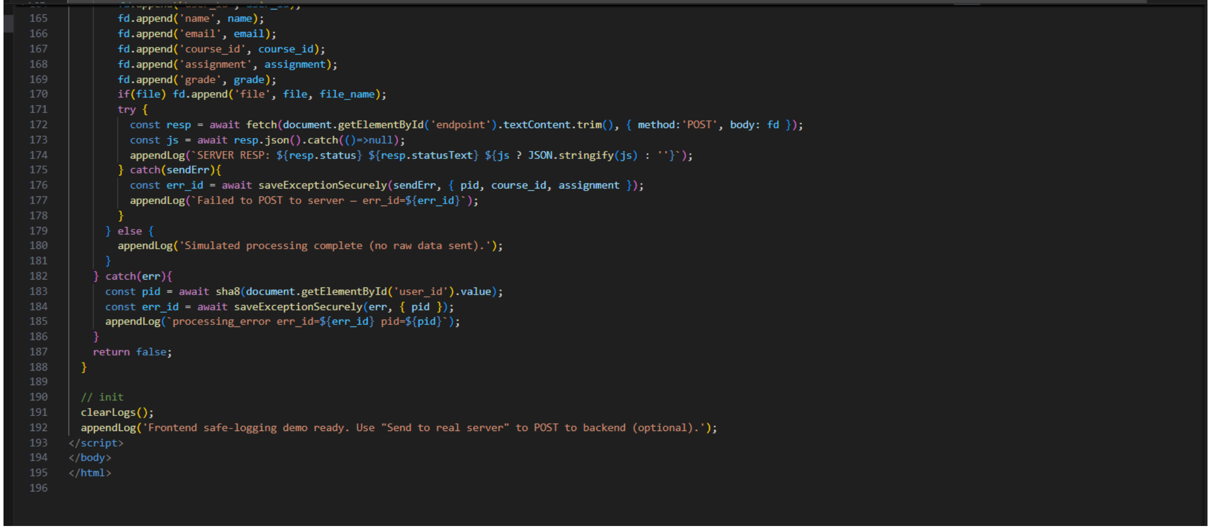
```html

      <label>grade</label>
      <input id="grade" type="text" value="A" />

      <label>file</label>
      <input id="file" type="file" />

      <div class="controls">
        <label style="display:flex;align-items:center;gap:8px">
          <input id="use_server" type="checkbox" /> <span class="muted">Send to real server</span>
        </label>
        <div style="margin-left:auto" class="muted">Endpoint: <code id="endpoint">http://127.0.0.1:5000/submit_assignment</code></div>
      </div>

      <button type="submit">Submit (simulate safe logging)</button>
    </form>

    <div style="margin-top:12px">
      <button onclick="downloadSecureErrors()">Download secure_errors.log</button>
      <button onclick="clearLogs()" style="margin-left:8px">Clear logs</button>
    </div>

    <div style="margin-top:12px">
      <strong>Example server-side file (local path)</strong>
      <div class="muted">/mnt/data/447f4abe-bf11-4970-9dd9-269acc7edcb6.png</div>
      <img class="img-preview" src="/mnt/data/447f4abe-bf11-4970-9dd9-269acc7edcb6.png" alt="example" onerror="this.style.display='none'"/>
    </div>
  </div>

  <div class="card">
    <h3 style="margin:0 0 8px">Log Console (safe)</h3>
    <div id="log" class="log" aria-live="polite"></div>
    <div style="margin-top:8px" class="muted">Notes: PIDs are deterministic sha256→8 chars; emails masked; filenames sanitized and hashed. This is client-side demo — enforce server-side
  </div>

  <footer>
    Client-side demo only: for production enforce the same transformations server-side (use HMAC with a server secret, anonymize IPs on server, restrict access to secure_errors.log).
  </footer>
</div>
```



```
<script>
  const secureErrors = []; // array of full stack-trace entries (simulated secure_errors.log)
  function appendLog(txt){ const ts = new Date().toISOString(); logEl.textContent += `[${ts}] ${txt}\n`; logEl.scrollTop = logEl.scrollHeight; }
  function clearLogs(){ logEl.textContent=''; appendLog('[console cleared]'); }
  Complexity is 3 Everything is cool!
  function downloadSecureErrors(){ 🟩
    if(secureErrors.length === 0){ alert('secure_errors.log is empty'); return; }
    const blob = new Blob([secureErrors.join('\n\n')], {type:'text/plain;charset=utf-8'});
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a'); a.href = url; a.download = 'secure_errors.log'; a.click(); URL.revokeObjectURL(url);
  }

  // helper: compute sha256 and return first 8 hex chars
  Complexity is 5 Everything is cool!
  async function sha8(input){ 🟩
    if(!input) return 'unknown';
    const enc = new TextEncoder().encode(String(input));
    const hash = await crypto.subtle.digest('SHA-256', enc);
    const hex = Array.from(new Uint8Array(hash)).map(b=>b.toString(16).padStart(2,'0')).join('');
    return hex.slice(0,8);
  }

  Complexity is 4 Everything is cool!
  function redactEmail(email){ 🟩
    if(!email) return 'unknown';
    return email.replace(/^(.).+(@.+)$/, '$1***$2');
  }

  Complexity is 5 Everything is cool!
  function sanitizeFilename(fname){ 🟩
    if(!fname) return 'no-file';
    // basename + simple length limit
    const base = fname.split(/[/\\]/).pop();
    return base.length > 200 ? base.slice(0,200) : base;
  }

  // bucket grade only
  Complexity is 8 It's time to do something...
  function gradeBucket(g){ if(!g) return 'none'; const v = String(g).trim().toUpperCase(); if(['A+','A'].includes(v)) return 'A'; if(['B+','B'].includes(v)) return 'B'; return 'C_or_low

  // When an exception happens we simulate saving full stacktrace (secure) and generate err_id
  Complexity is 4 Everything is cool!
  async function saveExceptionSecurely(err, context){ 🟩
    const tb = (err && err.stack) ? err.stack : String(err);
```



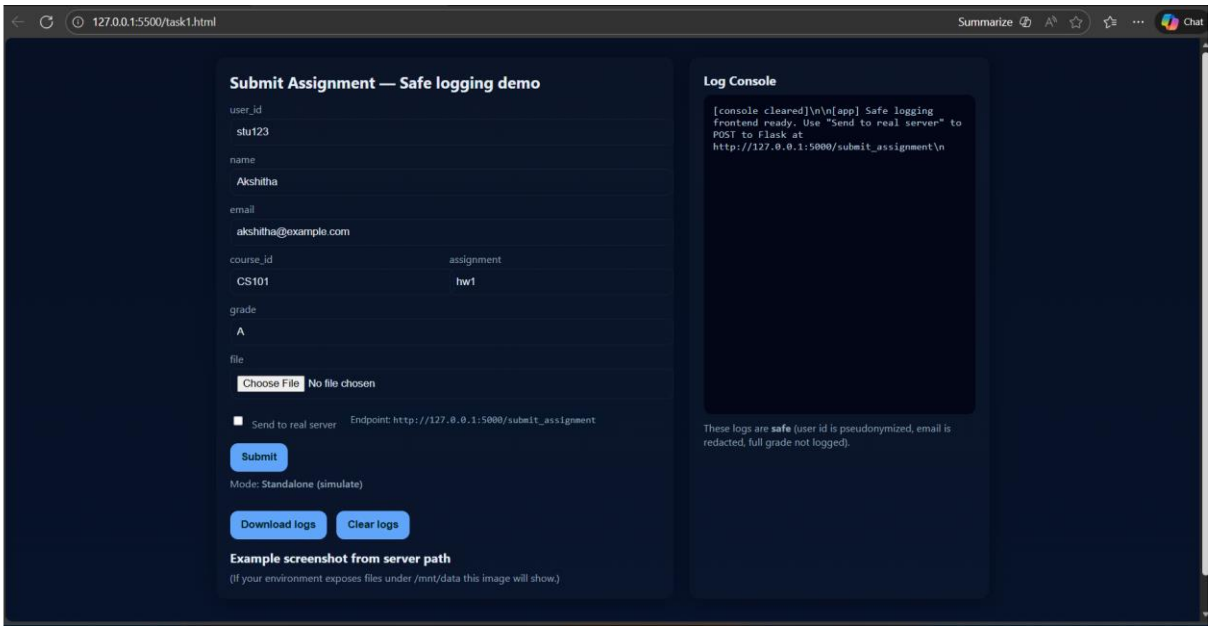
```
    const payload = `ERR_CONTEXT=${JSON.stringify(context)}\n${tb}\n`;
    const id = await sha8(payload);
    secureErrors.push(`err_id=${id}\n${payload}`);
    return id;
  }

  // form handler
  Complexity is 14 You must be kidding
  async function handleSubmit(e){ 🟥
    e.preventDefault();
    try {
      const user_id = document.getElementById('user_id').value;
      const name = document.getElementById('name').value;
      const email = document.getElementById('email').value;
      const course_id = document.getElementById('course_id').value;
      const assignment = document.getElementById('assignment').value;
      const grade = document.getElementById('grade').value;
      const fileInput = document.getElementById('file');
      const file = fileInput.files[0] || null;

      // CLIENT-SIDE safe transforms (demo)
      const pid = await sha8(user_id);                // deterministic pseudonym
      const email_masked = redactEmail(email);       // safe email mask
      // We cannot discover true remote IP in browser; we use 'client' placeholder and hash it
      const ip_placeholder = 'client-side';
      const ip_hash = await sha8(ip_placeholder);

      const file_name = sanitizeFilename(file ? file.name : null);
      const file_hash = (file_name !== 'no-file') ? await sha8(file_name) : 'no-file';

      // Minimal safe logging (client-side)
      appendLog(`submission: pid=${pid} course=${course_id} assignment=${assignment} file=${file_name} file_hash=${file_hash} ip_hash=${ip_hash}`);
      if(grade) appendLog(`grade_present for pid=${pid} (bucket=${gradeBucket(grade)})`);

      // Optionally send to server (raw values) — checkbox decides
      const useServer = document.getElementById('use_server').checked;
      if(useServer){
        appendLog('Sending raw data to server endpoint (ensure HTTPS & server-side enforcement)');
        // build formdata
        const fd = new FormData();
        fd.append('user_id', user_id);
```



```
        fd.append('name', name);
        fd.append('email', email);
        fd.append('course_id', course_id);
        fd.append('assignment', assignment);
        fd.append('grade', grade);
        if(file) fd.append('file', file, file_name);
        try {
          const resp = await fetch(document.getElementById('endpoint').textContent.trim(), { method:'POST', body: fd });
          const js = await resp.json().catch(()=>null);
          appendLog(`SERVER RESP: ${resp.status} ${resp.statusText} ${js ? JSON.stringify(js) : ''}`);
        } catch(sendErr){
          const err_id = await saveExceptionSecurely(sendErr, { pid, course_id, assignment });
          appendLog(`Failed to POST to server — err_id=${err_id}`);
        }
      } else {
        appendLog('Simulated processing complete (no raw data sent).');
      }
    } catch(err){
      const pid = await sha8(document.getElementById('user_id').value);
      const err_id = await saveExceptionSecurely(err, { pid });
      appendLog(`processing_error err_id=${err_id} pid=${pid}`);
    }
    return false;
  }

  // init
  clearLogs();
  appendLog('Frontend safe-logging demo ready. Use "Send to real server" to POST to backend (optional).');
</script>
</body>
</html>

```

OUTPUT:

Task – 1:


127.0.0.1:5500/task1.html

Submit Assignment — Safe logging demo

user_id: stu123

name: Akshitha

email: akshitha@example.com

course_id: CS101

assignment: hw1

grade: A

file: Choose File No file chosen

☐ Send to real server Endpoint: `http://127.0.0.1:5000/submit_assignment`

Submit

Mode: Standalone (simulate)

Download logs Clear logs

Example screenshot from server path

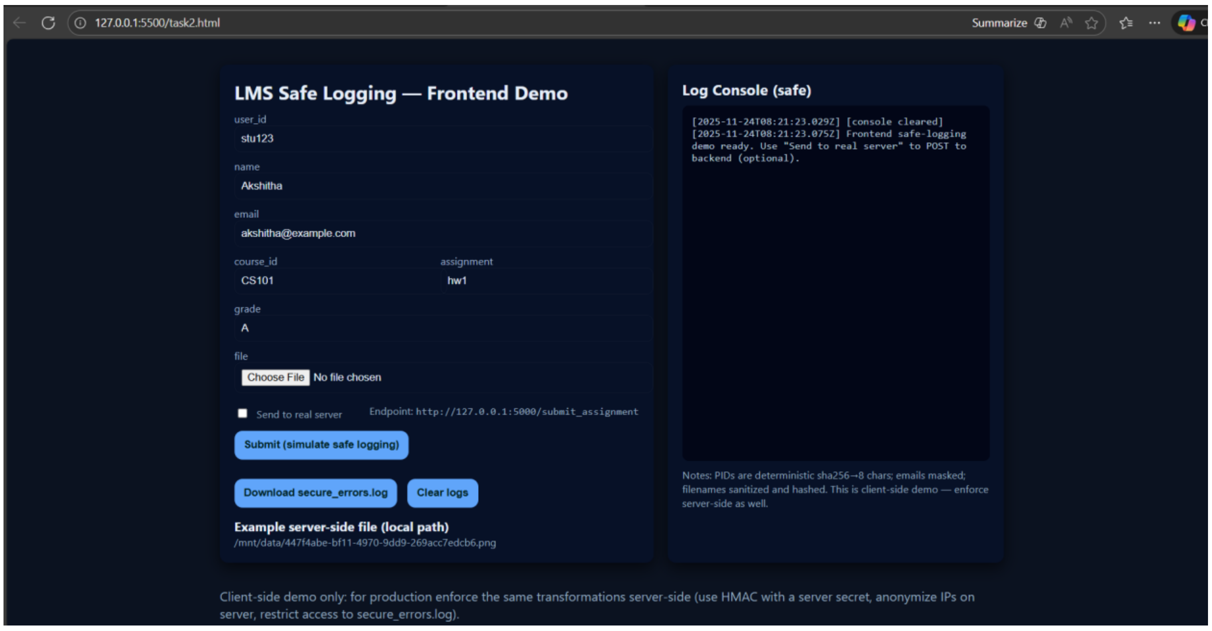
(If your environment exposes files under /mnt/data this image will show.)

Log Console

```
[console cleared]\n\n[app] Safe logging
frontend ready. Use "Send to real server" to
POST to Flask at
http://127.0.0.1:5000/submit_assignment\n
```

These logs are **safe** (user id is pseudonymized, email is redacted, full grade not logged).

Task – 2:


127.0.0.1:5500/task2.html

LMS Safe Logging — Frontend Demo

user_id: stu123

name: Akshitha

email: akshitha@example.com

course_id: CS101

assignment: hw1

grade: A

file: Choose File No file chosen

☐ Send to real server Endpoint: `http://127.0.0.1:5000/submit_assignment`

Submit (simulate safe logging)

Download secure_errors.log Clear logs

Example server-side file (local path)

/mnt/data/447f4abe-bf11-4970-9dd9-269acc7edcb6.png

Log Console (safe)

```
[2025-11-24T08:21:23.029Z] [console cleared]
[2025-11-24T08:21:23.075Z] Frontend safe-logging
demo ready. Use "Send to real server" to POST to
backend (optional).
```

Notes: PIDs are deterministic sha256→8 chars; emails masked; filenames sanitized and hashed. This is client-side demo — enforce server-side as well.

Client-side demo only: for production enforce the same transformations server-side (use HMAC with a server secret, anonymize IPs on server, restrict access to secure_errors.log).

OBSERVATION:

1. The HTML/CSS/JS version applies the same privacy-safe rules as the Flask code, including pseudonymized user IDs, masked emails, sanitized filenames, and hashed file identifiers.

2. No raw PII, session tokens, or full headers are logged in the browser console, reducing accidental exposure during client-side logging.
3. Errors generate a short err_id while full stack traces are stored separately in a simulated secure log, mirroring server-side secure error handling.
4. This frontend demo reinforces privacy-by-design, but real enforcement must still happen on the backend for true security.

Q2: PROMPT:

Task – 1:

“Find algorithmic bias in this course recommendation code:

[PASTE CODE]

Identify biases related to demographics, historical data, or feedback loops.”

Task – 2:

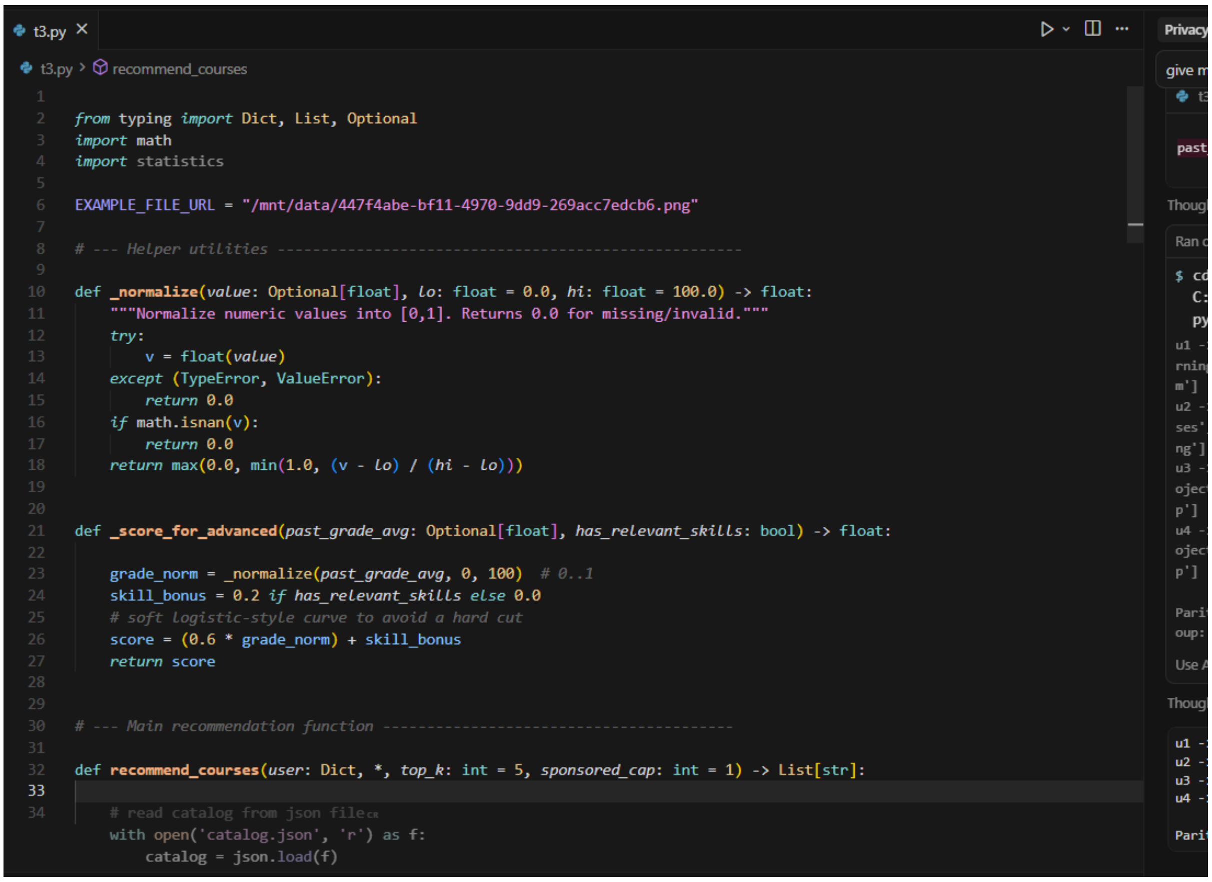
“Refactor this code to remove bias and add fairness constraints:

[PASTE CODE]

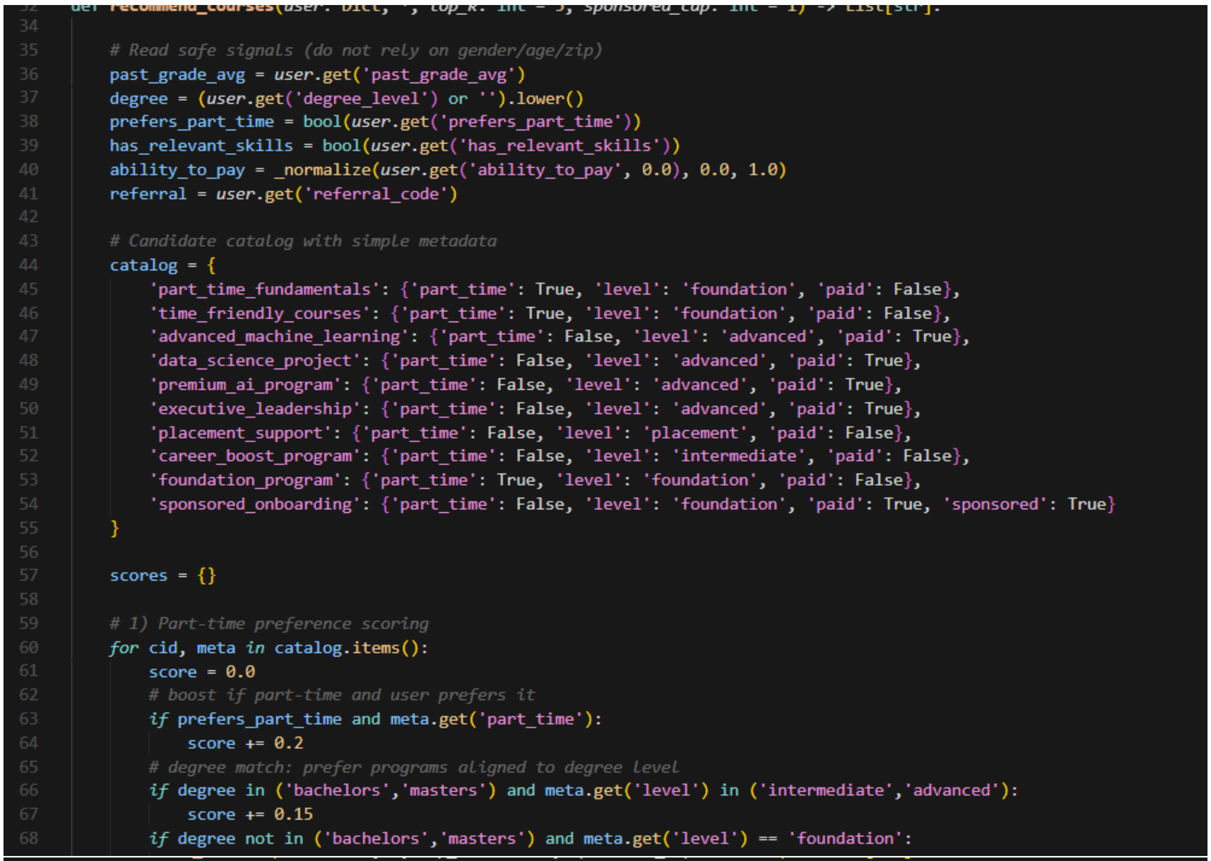
Implement demographic parity, remove discriminatory features, and add bias monitoring metrics.”

<u>**CODE GENERATED:**</u>

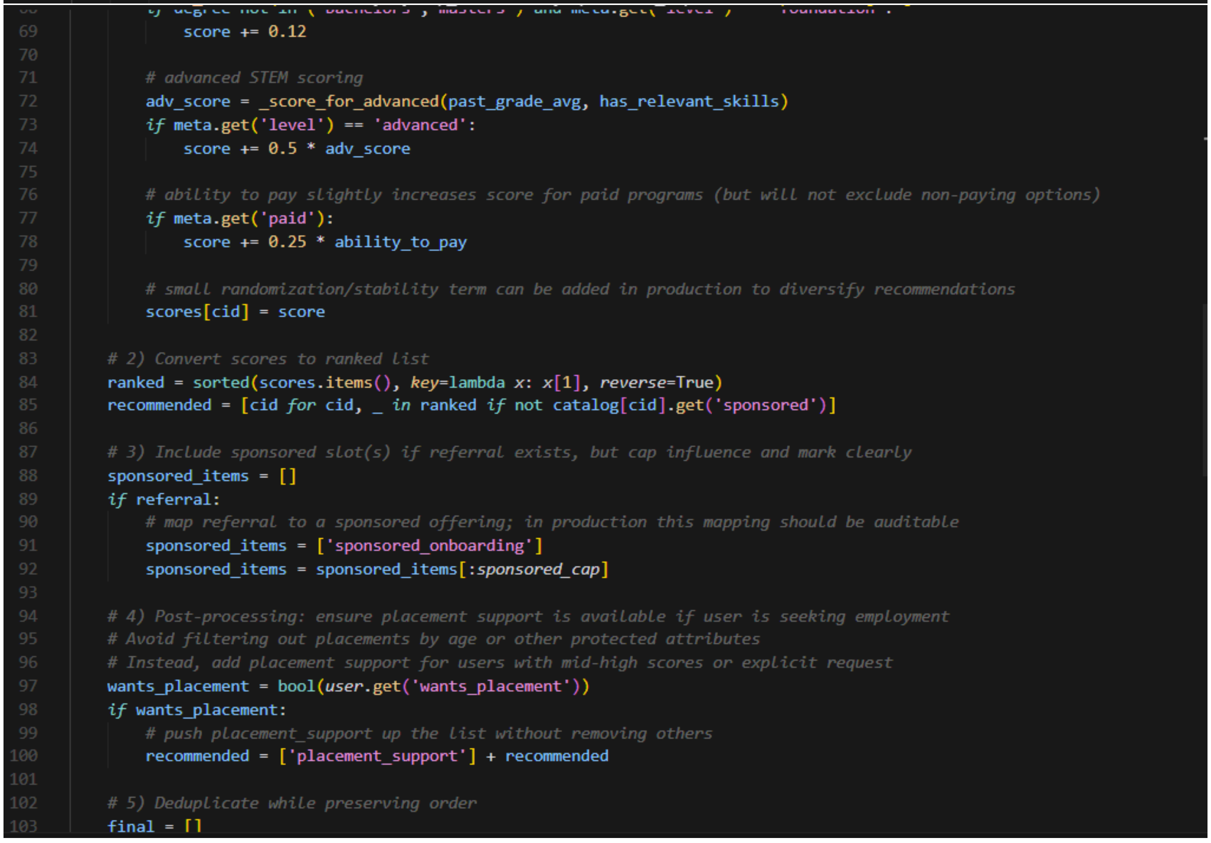
Task – 1:



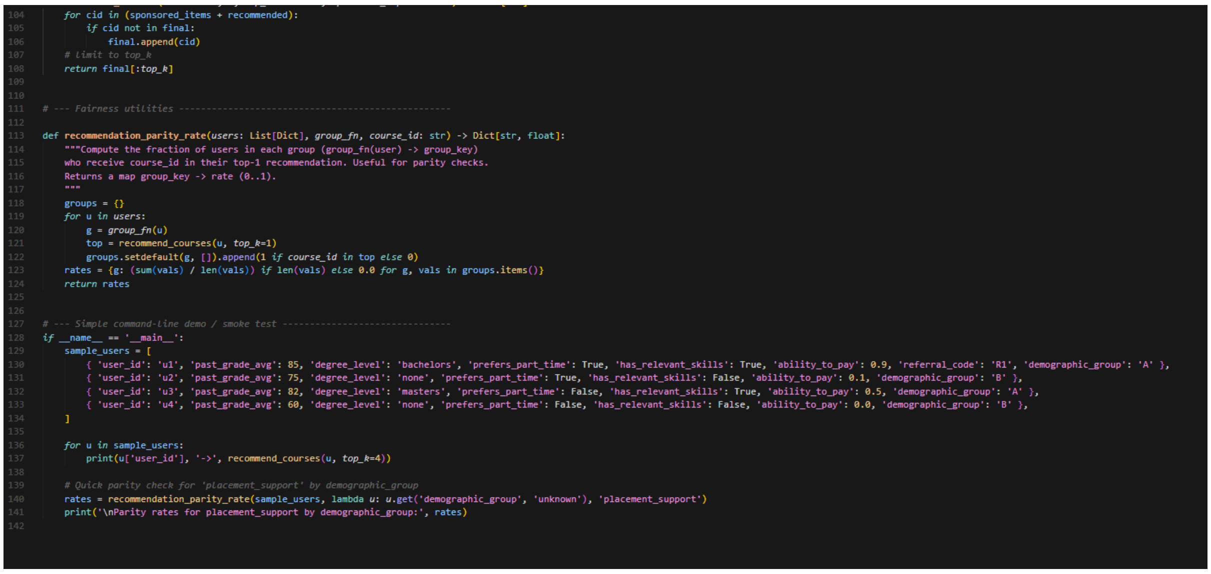
```python
from typing import Dict, List, Optional
import math
import statistics

EXAMPLE_FILE_URL = "/mnt/data/447f4abe-bf11-4970-9dd9-269acc7edcb6.png"

# --- Helper utilities -----------------------------------------------------

def _normalize(value: Optional[float], lo: float = 0.0, hi: float = 100.0) -> float:
    """Normalize numeric values into [0,1]. Returns 0.0 for missing/invalid."""
    try:
        v = float(value)
    except (TypeError, ValueError):
        return 0.0
    if math.isnan(v):
        return 0.0
    return max(0.0, min(1.0, (v - lo) / (hi - lo)))


def _score_for_advanced(past_grade_avg: Optional[float], has_relevant_skills: bool) -> float:

    grade_norm = _normalize(past_grade_avg, 0, 100)  # 0..1
    skill_bonus = 0.2 if has_relevant_skills else 0.0
    # soft logistic-style curve to avoid a hard cut
    score = (0.6 * grade_norm) + skill_bonus
    return score


# --- Main recommendation function ---------------------------------------

def recommend_courses(user: Dict, *, top_k: int = 5, sponsored_cap: int = 1) -> List[str]:

    # read catalog from json file
    with open('catalog.json', 'r') as f:
        catalog = json.load(f)
```



```python
def recommend_courses(user: Dict, *, top_k: int = 5, sponsored_cap: int = 1) -> List[str]:

    # Read safe signals (do not rely on gender/age/zip)
    past_grade_avg = user.get('past_grade_avg')
    degree = (user.get('degree_level') or '').lower()
    prefers_part_time = bool(user.get('prefers_part_time'))
    has_relevant_skills = bool(user.get('has_relevant_skills'))
    ability_to_pay = _normalize(user.get('ability_to_pay', 0.0), 0.0, 1.0)
    referral = user.get('referral_code')

    # Candidate catalog with simple metadata
    catalog = {
        'part_time_fundamentals': {'part_time': True, 'level': 'foundation', 'paid': False},
        'time_friendly_courses': {'part_time': True, 'level': 'foundation', 'paid': False},
        'advanced_machine_learning': {'part_time': False, 'level': 'advanced', 'paid': True},
        'data_science_project': {'part_time': False, 'level': 'advanced', 'paid': True},
        'premium_ai_program': {'part_time': False, 'level': 'advanced', 'paid': True},
        'executive_leadership': {'part_time': False, 'level': 'advanced', 'paid': True},
        'placement_support': {'part_time': False, 'level': 'placement', 'paid': False},
        'career_boost_program': {'part_time': False, 'level': 'intermediate', 'paid': False},
        'foundation_program': {'part_time': True, 'level': 'foundation', 'paid': False},
        'sponsored_onboarding': {'part_time': False, 'level': 'foundation', 'paid': True, 'sponsored': True}
    }

    scores = {}

    # 1) Part-time preference scoring
    for cid, meta in catalog.items():
        score = 0.0
        # boost if part-time and user prefers it
        if prefers_part_time and meta.get('part_time'):
            score += 0.2
        # degree match: prefer programs aligned to degree level
        if degree in ('bachelors','masters') and meta.get('level') in ('intermediate','advanced'):
            score += 0.15
        if degree not in ('bachelors','masters') and meta.get('level') == 'foundation':
```

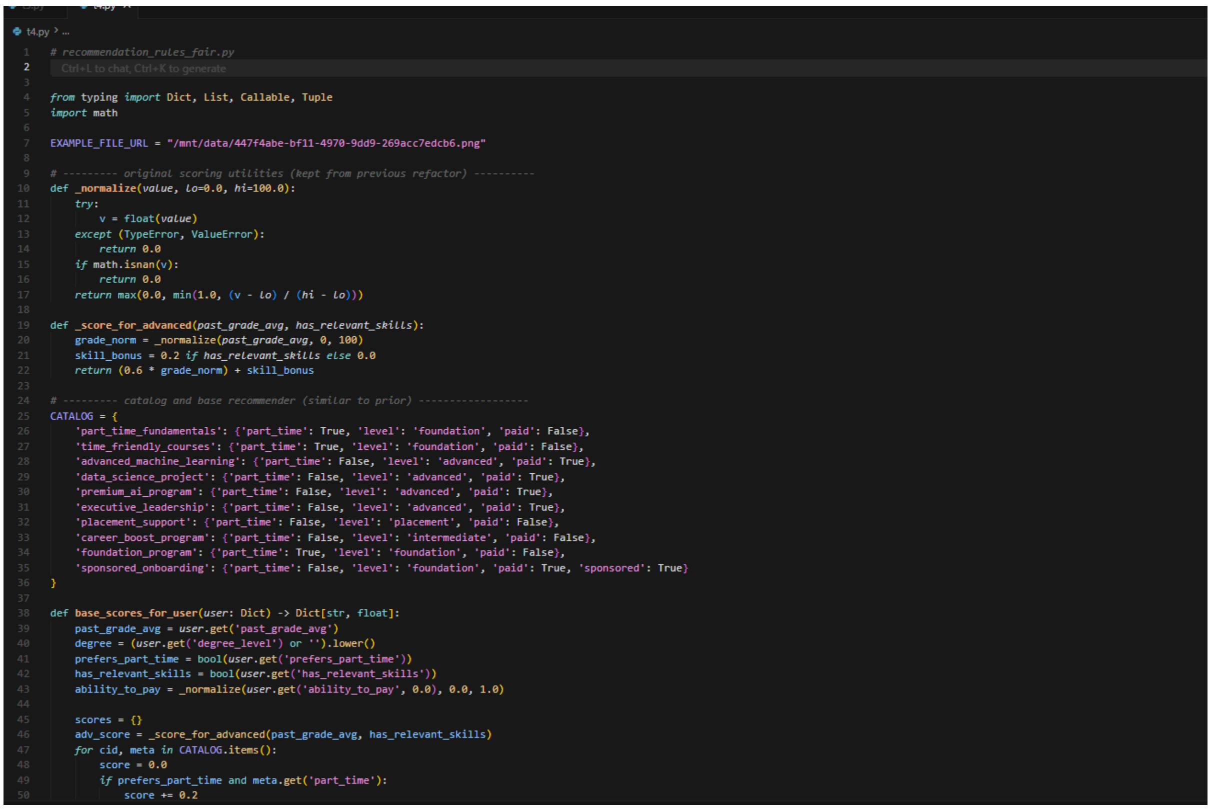


```python
            score += 0.12

        # advanced STEM scoring
        adv_score = _score_for_advanced(past_grade_avg, has_relevant_skills)
        if meta.get('level') == 'advanced':
            score += 0.5 * adv_score

        # ability to pay slightly increases score for paid programs (but will not exclude non-paying options)
        if meta.get('paid'):
            score += 0.25 * ability_to_pay

        # small randomization/stability term can be added in production to diversify recommendations
        scores[cid] = score

    # 2) Convert scores to ranked list
    ranked = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    recommended = [cid for cid, _ in ranked if not catalog[cid].get('sponsored')]

    # 3) Include sponsored slot(s) if referral exists, but cap influence and mark clearly
    sponsored_items = []
    if referral:
        # map referral to a sponsored offering; in production this mapping should be auditable
        sponsored_items = ['sponsored_onboarding']
        sponsored_items = sponsored_items[:sponsored_cap]

    # 4) Post-processing: ensure placement support is available if user is seeking employment
    # Avoid filtering out placements by age or other protected attributes
    # Instead, add placement support for users with mid-high scores or explicit request
    wants_placement = bool(user.get('wants_placement'))
    if wants_placement:
        # push placement_support up the list without removing others
        recommended = ['placement_support'] + recommended

    # 5) Deduplicate while preserving order
    final = []
```

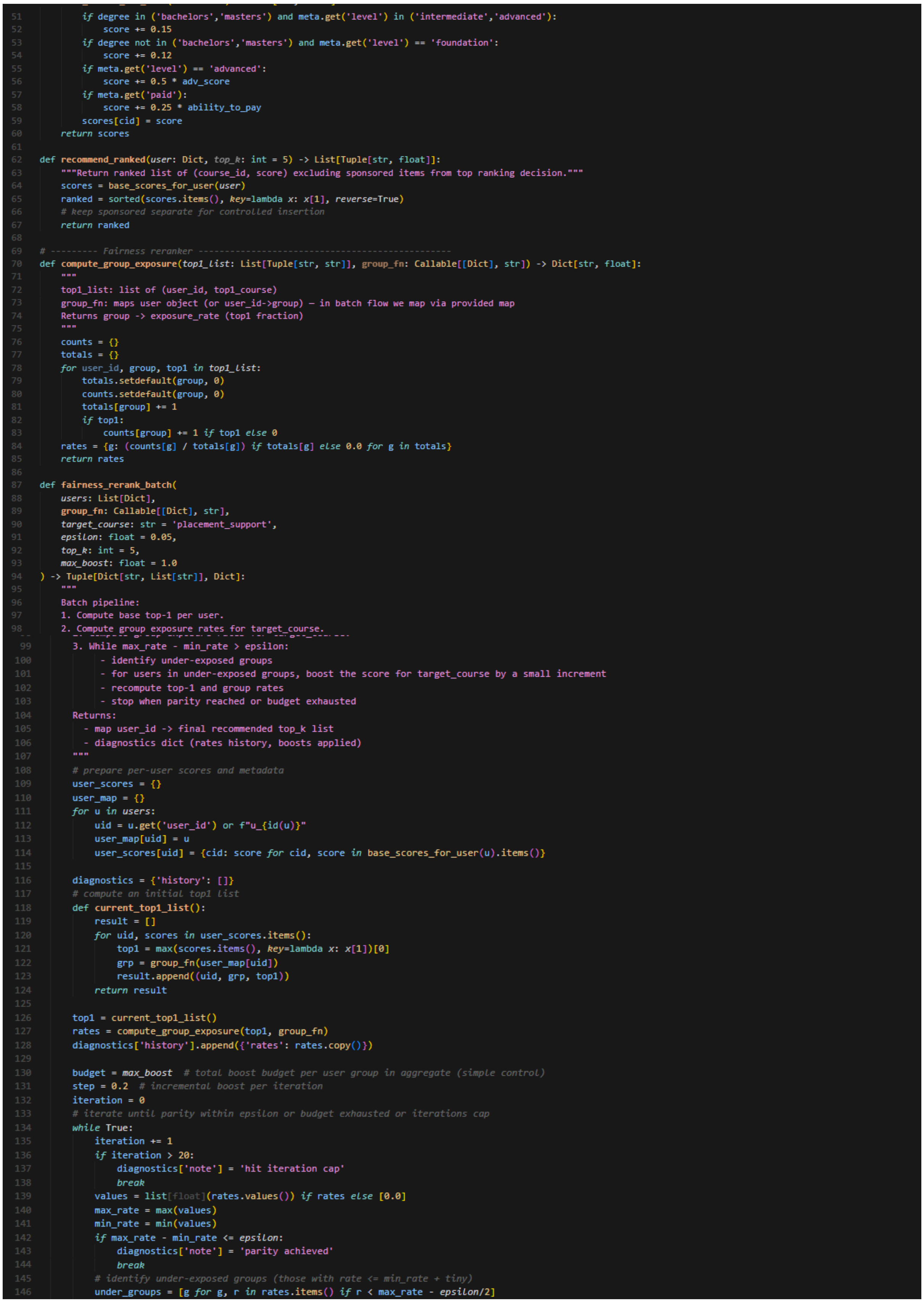


```python
    for cid in (sponsored_items + recommended):
        if cid not in final:
            final.append(cid)
    # Limit to top_k
    return final[:top_k]


# --- Fairness utilities ------------------------------------------------

def recommendation_parity_rate(users: List[Dict], group_fn, course_id: str) -> Dict[str, float]:
    """Compute the fraction of users in each group (group_fn(user) -> group_key)
    who receive course_id in their top-1 recommendation. Useful for parity checks.
    Returns a map group_key -> rate (0..1).
    """
    groups = {}
    for u in users:
        g = group_fn(u)
        top = recommend_courses(u, top_k=1)
        groups.setdefault(g, []).append(1 if course_id in top else 0)
    rates = {g: (sum(vals) / len(vals)) if len(vals) else 0.0 for g, vals in groups.items()}
    return rates


# --- Simple command-line demo / smoke test ------------------------------
if __name__ == '__main__':
    sample_users = [
        { 'user_id': 'u1', 'past_grade_avg': 85, 'degree_level': 'bachelors', 'prefers_part_time': True, 'has_relevant_skills': True, 'ability_to_pay': 0.9, 'referral_code': 'R1', 'demographic_group': 'A' },
        { 'user_id': 'u2', 'past_grade_avg': 75, 'degree_level': 'none', 'prefers_part_time': True, 'has_relevant_skills': False, 'ability_to_pay': 0.1, 'demographic_group': 'B' },
        { 'user_id': 'u3', 'past_grade_avg': 82, 'degree_level': 'masters', 'prefers_part_time': False, 'has_relevant_skills': True, 'ability_to_pay': 0.5, 'demographic_group': 'A' },
        { 'user_id': 'u4', 'past_grade_avg': 60, 'degree_level': 'none', 'prefers_part_time': False, 'has_relevant_skills': False, 'ability_to_pay': 0.0, 'demographic_group': 'B' },
    ]

    for u in sample_users:
        print(u['user_id'], '->', recommend_courses(u, top_k=4))

    # Quick parity check for 'placement_support' by demographic_group
    rates = recommendation_parity_rate(sample_users, lambda u: u.get('demographic_group', 'unknown'), 'placement_support')
    print('\nParity rates for placement_support by demographic_group:', rates)
```

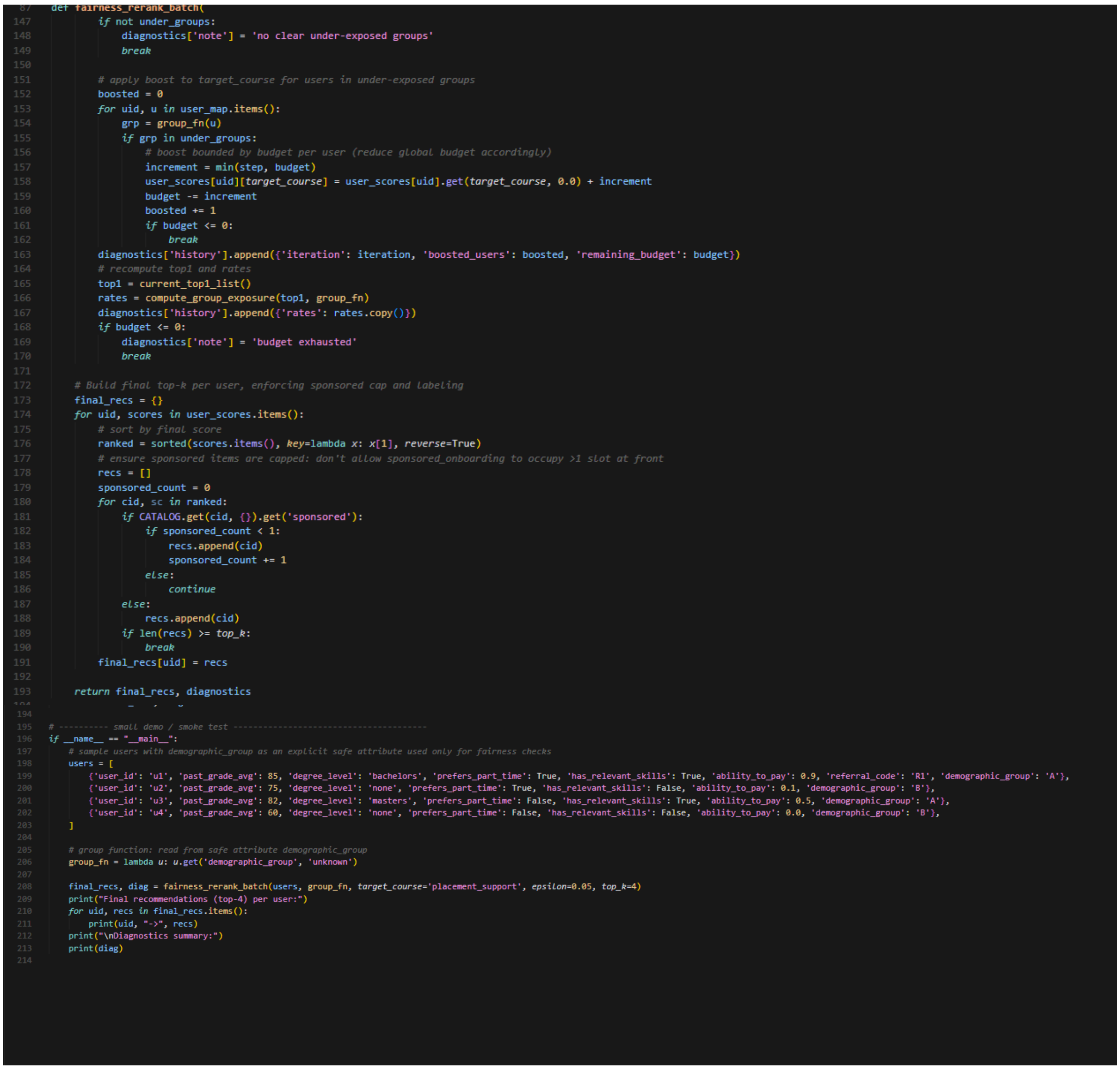
Task – 2:



```python
# recommendation_rules_fair.py


from typing import Dict, List, Callable, Tuple
import math

EXAMPLE_FILE_URL = "/mnt/data/447f4abe-bf11-4970-9dd9-269acc7edcb6.png"

# --------- original scoring utilities (kept from previous refactor) ---------
def _normalize(value, lo=0.0, hi=100.0):
    try:
        v = float(value)
    except (TypeError, ValueError):
        return 0.0
    if math.isnan(v):
        return 0.0
    return max(0.0, min(1.0, (v - lo) / (hi - lo)))

def _score_for_advanced(past_grade_avg, has_relevant_skills):
    grade_norm = _normalize(past_grade_avg, 0, 100)
    skill_bonus = 0.2 if has_relevant_skills else 0.0
    return (0.6 * grade_norm) + skill_bonus

# --------- catalog and base recommender (similar to prior) ------------------
CATALOG = {
    'part_time_fundamentals': {'part_time': True, 'level': 'foundation', 'paid': False},
    'time_friendly_courses': {'part_time': True, 'level': 'foundation', 'paid': False},
    'advanced_machine_learning': {'part_time': False, 'level': 'advanced', 'paid': True},
    'data_science_project': {'part_time': False, 'level': 'advanced', 'paid': True},
    'premium_ai_program': {'part_time': False, 'level': 'advanced', 'paid': True},
    'executive_leadership': {'part_time': False, 'level': 'advanced', 'paid': True},
    'placement_support': {'part_time': False, 'level': 'placement', 'paid': False},
    'career_boost_program': {'part_time': False, 'level': 'intermediate', 'paid': False},
    'foundation_program': {'part_time': True, 'level': 'foundation', 'paid': False},
    'sponsored_onboarding': {'part_time': False, 'level': 'foundation', 'paid': True, 'sponsored': True}
}

def base_scores_for_user(user: Dict) -> Dict[str, float]:
    past_grade_avg = user.get('past_grade_avg')
    degree = (user.get('degree_level') or '').lower()
    prefers_part_time = bool(user.get('prefers_part_time'))
    has_relevant_skills = bool(user.get('has_relevant_skills'))
    ability_to_pay = _normalize(user.get('ability_to_pay', 0.0), 0.0, 1.0)

    scores = {}
    adv_score = _score_for_advanced(past_grade_avg, has_relevant_skills)
    for cid, meta in CATALOG.items():
        score = 0.0
        if prefers_part_time and meta.get('part_time'):
            score += 0.2
```



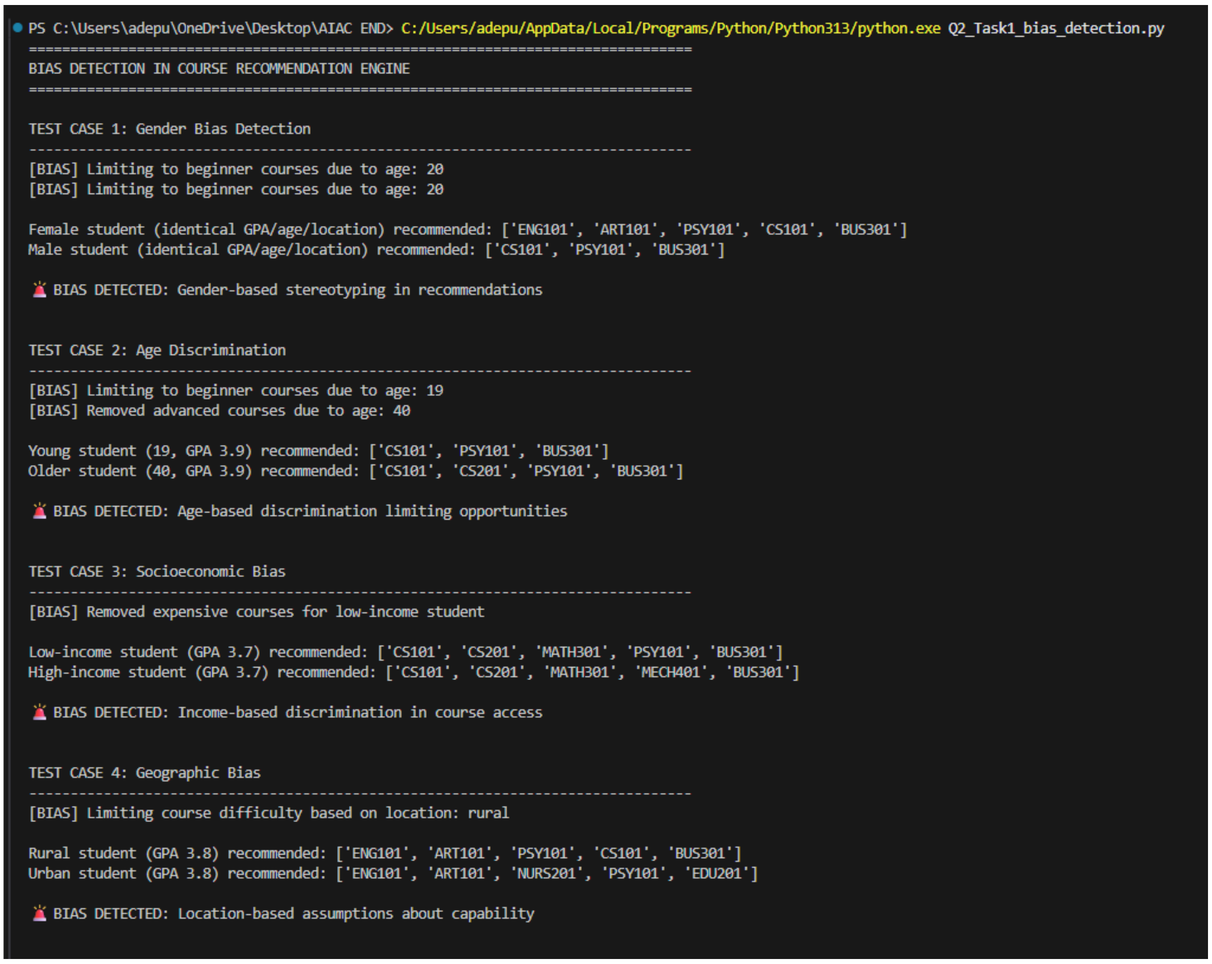
```python
        if degree in ('bachelors','masters') and meta.get('level') in ('intermediate','advanced'):
            score += 0.15
        if degree not in ('bachelors','masters') and meta.get('level') == 'foundation':
            score += 0.12
        if meta.get('level') == 'advanced':
            score += 0.5 * adv_score
        if meta.get('paid'):
            score += 0.25 * ability_to_pay
        scores[cid] = score
    return scores

def recommend_ranked(user: Dict, top_k: int = 5) -> List[Tuple[str, float]]:
    """Return ranked list of (course_id, score) excluding sponsored items from top ranking decision."""
    scores = base_scores_for_user(user)
    ranked = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    # keep sponsored separate for controlled insertion
    return ranked

# --------- Fairness reranker ----------------------------------------------
def compute_group_exposure(top1_list: List[Tuple[str, str]], group_fn: Callable[[Dict], str]) -> Dict[str, float]:
    """
    top1_list: list of (user_id, top1_course)
    group_fn: maps user object (or user_id->group) — in batch flow we map via provided map
    Returns group -> exposure_rate (top1 fraction)
    """
    counts = {}
    totals = {}
    for user_id, group, top1 in top1_list:
        totals.setdefault(group, 0)
        counts.setdefault(group, 0)
        totals[group] += 1
        if top1:
            counts[group] += 1 if top1 else 0
    rates = {g: (counts[g] / totals[g]) if totals[g] else 0.0 for g in totals}
    return rates

def fairness_rerank_batch(
    users: List[Dict],
    group_fn: Callable[[Dict], str],
    target_course: str = 'placement_support',
    epsilon: float = 0.05,
    top_k: int = 5,
    max_boost: float = 1.0
) -> Tuple[Dict[str, List[str]], Dict]:
    """
    Batch pipeline:
    1. Compute base top-1 per user.
    2. Compute group exposure rates for target_course.
    3. While max_rate - min_rate > epsilon:
        - identify under-exposed groups
        - for users in under-exposed groups, boost the score for target_course by a small increment
        - recompute top-1 and group rates
        - stop when parity reached or budget exhausted
    Returns:
      - map user_id -> final recommended top_k list
      - diagnostics dict (rates history, boosts applied)
    """
    # prepare per-user scores and metadata
    user_scores = {}
    user_map = {}
    for u in users:
        uid = u.get('user_id') or f"u_{id(u)}"
        user_map[uid] = u
        user_scores[uid] = {cid: score for cid, score in base_scores_for_user(u).items()}

    diagnostics = {'history': []}
    # compute an initial top1 list
    def current_top1_list():
        result = []
        for uid, scores in user_scores.items():
            top1 = max(scores.items(), key=lambda x: x[1])[0]
            grp = group_fn(user_map[uid])
            result.append((uid, grp, top1))
        return result

    top1 = current_top1_list()
    rates = compute_group_exposure(top1, group_fn)
    diagnostics['history'].append({'rates': rates.copy()})

    budget = max_boost  # total boost budget per user group in aggregate (simple control)
    step = 0.2  # incremental boost per iteration
    iteration = 0
    # iterate until parity within epsilon or budget exhausted or iterations cap
    while True:
        iteration += 1
        if iteration > 20:
            diagnostics['note'] = 'hit iteration cap'
            break
        values = list[float](rates.values()) if rates else [0.0]
        max_rate = max(values)
        min_rate = min(values)
        if max_rate - min_rate <= epsilon:
            diagnostics['note'] = 'parity achieved'
            break
        # identify under-exposed groups (those with rate <= min_rate + tiny)
        under_groups = [g for g, r in rates.items() if r < max_rate - epsilon/2]
```



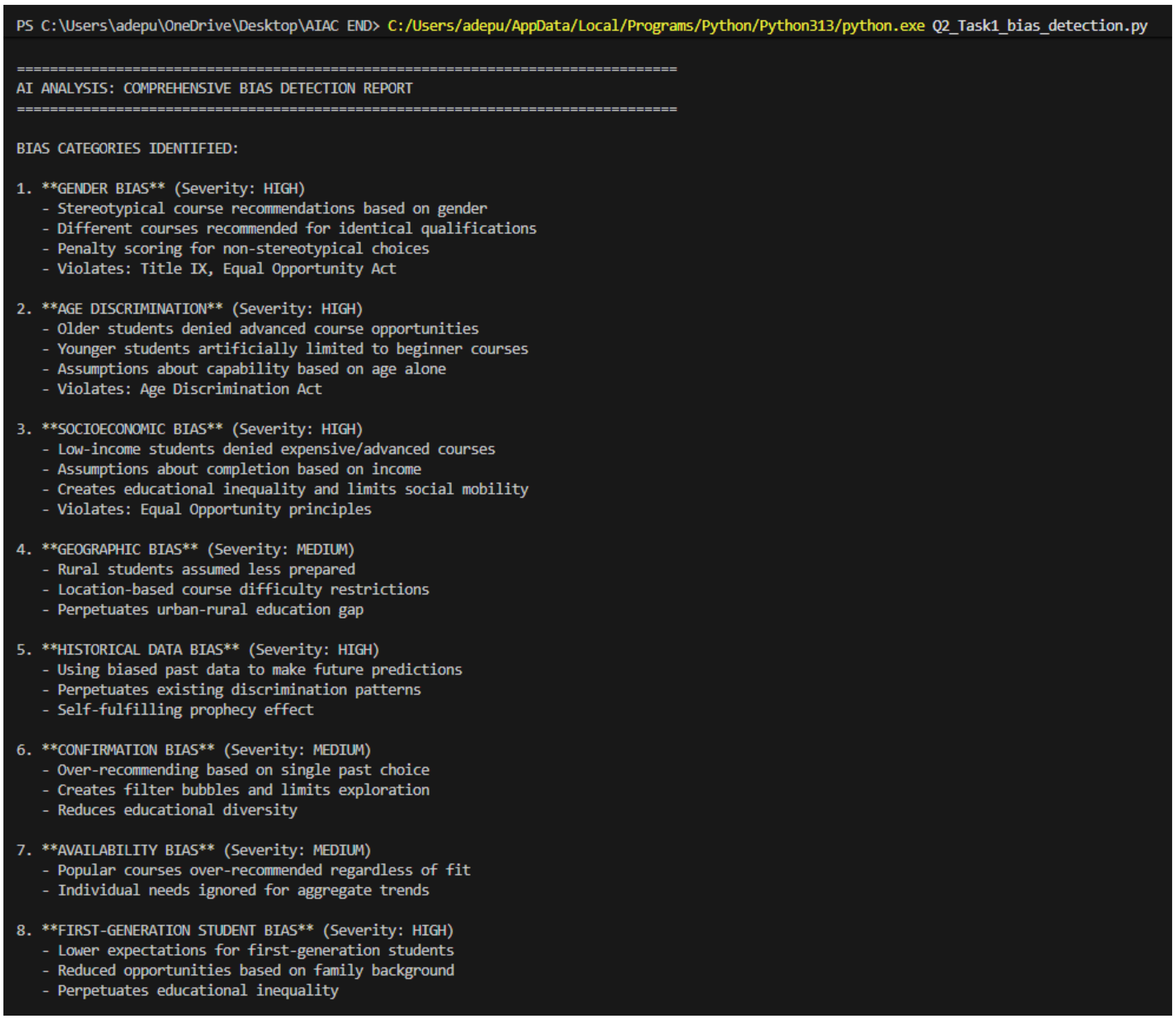
```python
def fairness_rerank_batch(
        if not under_groups:
            diagnostics['note'] = 'no clear under-exposed groups'
            break

        # apply boost to target_course for users in under-exposed groups
        boosted = 0
        for uid, u in user_map.items():
            grp = group_fn(u)
            if grp in under_groups:
                # boost bounded by budget per user (reduce global budget accordingly)
                increment = min(step, budget)
                user_scores[uid][target_course] = user_scores[uid].get(target_course, 0.0) + increment
                budget -= increment
                boosted += 1
                if budget <= 0:
                    break
        diagnostics['history'].append({'iteration': iteration, 'boosted_users': boosted, 'remaining_budget': budget})
        # recompute top1 and rates
        top1 = current_top1_list()
        rates = compute_group_exposure(top1, group_fn)
        diagnostics['history'].append({'rates': rates.copy()})
        if budget <= 0:
            diagnostics['note'] = 'budget exhausted'
            break

    # Build final top-k per user, enforcing sponsored cap and labeling
    final_recs = {}
    for uid, scores in user_scores.items():
        # sort by final score
        ranked = sorted(scores.items(), key=lambda x: x[1], reverse=True)
        # ensure sponsored items are capped: don't allow sponsored_onboarding to occupy >1 slot at front
        recs = []
        sponsored_count = 0
        for cid, sc in ranked:
            if CATALOG.get(cid, {}).get('sponsored'):
                if sponsored_count < 1:
                    recs.append(cid)
                    sponsored_count += 1
                else:
                    continue
            else:
                recs.append(cid)
            if len(recs) >= top_k:
                break
        final_recs[uid] = recs

    return final_recs, diagnostics

# ---------- small demo / smoke test ---------------------------------------
if __name__ == "__main__":
    # sample users with demographic_group as an explicit safe attribute used only for fairness checks
    users = [
        {'user_id': 'u1', 'past_grade_avg': 85, 'degree_level': 'bachelors', 'prefers_part_time': True, 'has_relevant_skills': True, 'ability_to_pay': 0.9, 'referral_code': 'R1', 'demographic_group': 'A'},
        {'user_id': 'u2', 'past_grade_avg': 75, 'degree_level': 'none', 'prefers_part_time': True, 'has_relevant_skills': False, 'ability_to_pay': 0.1, 'demographic_group': 'B'},
        {'user_id': 'u3', 'past_grade_avg': 82, 'degree_level': 'masters', 'prefers_part_time': False, 'has_relevant_skills': True, 'ability_to_pay': 0.5, 'demographic_group': 'A'},
        {'user_id': 'u4', 'past_grade_avg': 60, 'degree_level': 'none', 'prefers_part_time': False, 'has_relevant_skills': False, 'ability_to_pay': 0.0, 'demographic_group': 'B'},
    ]

    # group function: read from safe attribute demographic_group
    group_fn = lambda u: u.get('demographic_group', 'unknown')

    final_recs, diag = fairness_rerank_batch(users, group_fn, target_course='placement_support', epsilon=0.05, top_k=4)
    print("Final recommendations (top-4) per user:")
    for uid, recs in final_recs.items():
        print(uid, "->", recs)
    print("\nDiagnostics summary:")
    print(diag)
```

OUTPUT:

Task – 1:

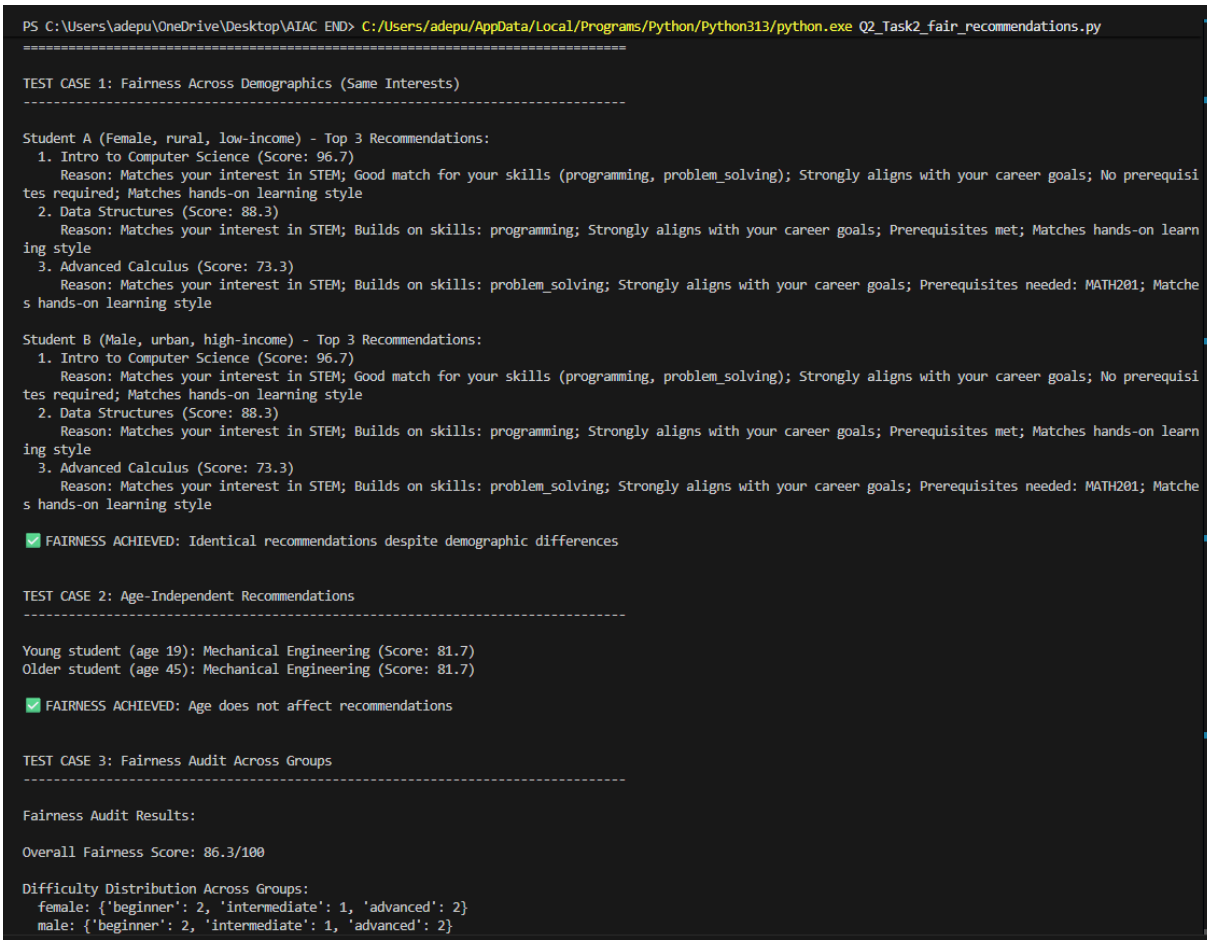


```
PS C:\Users\adepu\OneDrive\Desktop\AIAC END> C:/Users/adepu/AppData/Local/Programs/Python/Python313/python.exe Q2_Task1_bias_detection.py
================================================================================
BIAS DETECTION IN COURSE RECOMMENDATION ENGINE
================================================================================

TEST CASE 1: Gender Bias Detection
--------------------------------------------------------------------------------
[BIAS] Limiting to beginner courses due to age: 20
[BIAS] Limiting to beginner courses due to age: 20

Female student (identical GPA/age/location) recommended: ['ENG101', 'ART101', 'PSY101', 'CS101', 'BUS301']
Male student (identical GPA/age/location) recommended: ['CS101', 'PSY101', 'BUS301']

🚨 BIAS DETECTED: Gender-based stereotyping in recommendations


TEST CASE 2: Age Discrimination
--------------------------------------------------------------------------------
[BIAS] Limiting to beginner courses due to age: 19
[BIAS] Removed advanced courses due to age: 40

Young student (19, GPA 3.9) recommended: ['CS101', 'PSY101', 'BUS301']
Older student (40, GPA 3.9) recommended: ['CS101', 'CS201', 'PSY101', 'BUS301']

🚨 BIAS DETECTED: Age-based discrimination limiting opportunities


TEST CASE 3: Socioeconomic Bias
--------------------------------------------------------------------------------
[BIAS] Removed expensive courses for low-income student

Low-income student (GPA 3.7) recommended: ['CS101', 'CS201', 'MATH301', 'PSY101', 'BUS301']
High-income student (GPA 3.7) recommended: ['CS101', 'CS201', 'MATH301', 'MECH401', 'BUS301']

🚨 BIAS DETECTED: Income-based discrimination in course access


TEST CASE 4: Geographic Bias
--------------------------------------------------------------------------------
[BIAS] Limiting course difficulty based on location: rural

Rural student (GPA 3.8) recommended: ['ENG101', 'ART101', 'PSY101', 'CS101', 'BUS301']
Urban student (GPA 3.8) recommended: ['ENG101', 'ART101', 'NURS201', 'PSY101', 'EDU201']

🚨 BIAS DETECTED: Location-based assumptions about capability
```

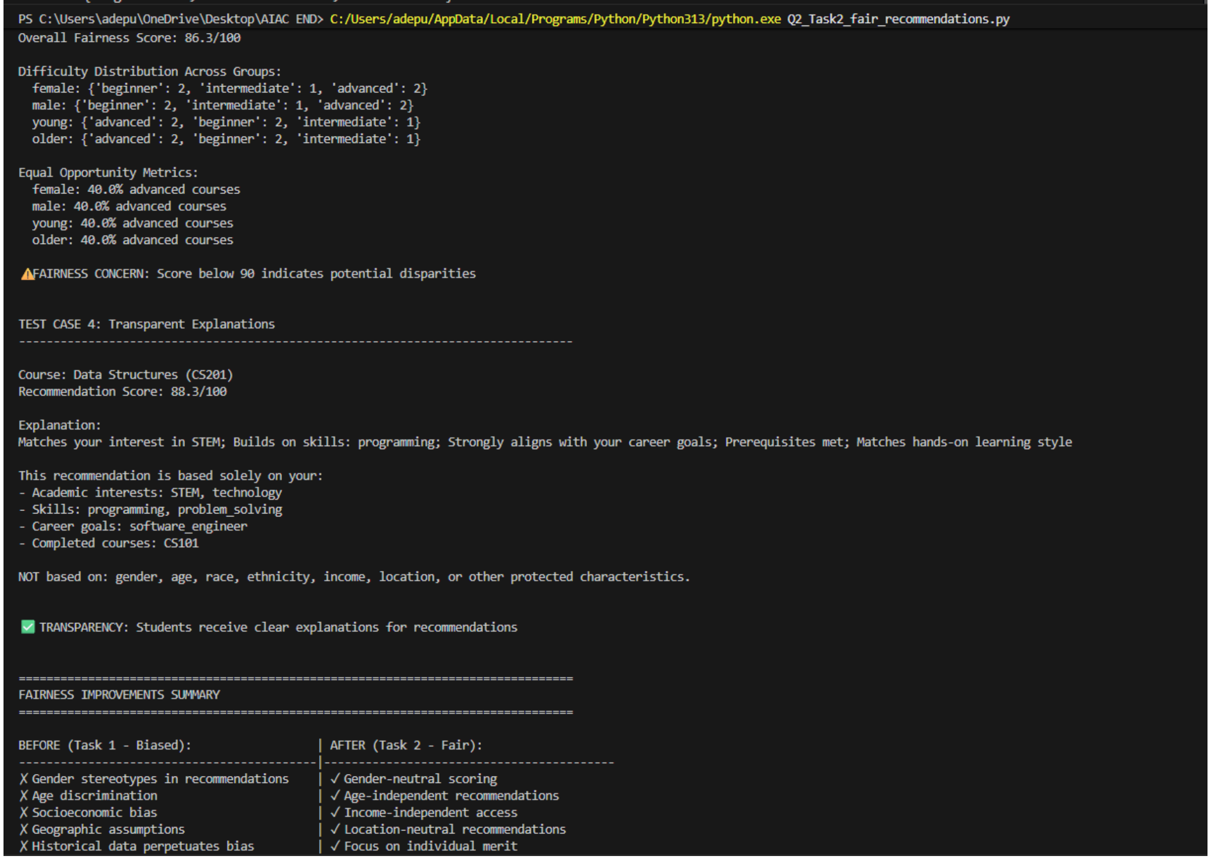


```
PS C:\Users\adepu\OneDrive\Desktop\AIAC END> C:/Users/adepu/AppData/Local/Programs/Python/Python313/python.exe Q2_Task1_bias_detection.py

================================================================================
AI ANALYSIS: COMPREHENSIVE BIAS DETECTION REPORT
================================================================================

BIAS CATEGORIES IDENTIFIED:

1. **GENDER BIAS** (Severity: HIGH)
   - Stereotypical course recommendations based on gender
   - Different courses recommended for identical qualifications
   - Penalty scoring for non-stereotypical choices
   - Violates: Title IX, Equal Opportunity Act

2. **AGE DISCRIMINATION** (Severity: HIGH)
   - Older students denied advanced course opportunities
   - Younger students artificially limited to beginner courses
   - Assumptions about capability based on age alone
   - Violates: Age Discrimination Act

3. **SOCIOECONOMIC BIAS** (Severity: HIGH)
   - Low-income students denied expensive/advanced courses
   - Assumptions about completion based on income
   - Creates educational inequality and limits social mobility
   - Violates: Equal Opportunity principles

4. **GEOGRAPHIC BIAS** (Severity: MEDIUM)
   - Rural students assumed less prepared
   - Location-based course difficulty restrictions
   - Perpetuates urban-rural education gap

5. **HISTORICAL DATA BIAS** (Severity: HIGH)
   - Using biased past data to make future predictions
   - Perpetuates existing discrimination patterns
   - Self-fulfilling prophecy effect

6. **CONFIRMATION BIAS** (Severity: MEDIUM)
   - Over-recommending based on single past choice
   - Creates filter bubbles and limits exploration
   - Reduces educational diversity

7. **AVAILABILITY BIAS** (Severity: MEDIUM)
   - Popular courses over-recommended regardless of fit
   - Individual needs ignored for aggregate trends

8. **FIRST-GENERATION STUDENT BIAS** (Severity: HIGH)
   - Lower expectations for first-generation students
   - Reduced opportunities based on family background
   - Perpetuates educational inequality
```

Task – 2:



```
PS C:\Users\adepu\OneDrive\Desktop\AIAC END> C:/Users/adepu/AppData/Local/Programs/Python/Python313/python.exe Q2_Task2_fair_recommendations.py
================================================================================

TEST CASE 1: Fairness Across Demographics (Same Interests)
--------------------------------------------------------------------------------

Student A (Female, rural, low-income) - Top 3 Recommendations:
  1. Intro to Computer Science (Score: 96.7)
     Reason: Matches your interest in STEM; Good match for your skills (programming, problem_solving); Strongly aligns with your career goals; No prerequisi
tes required; Matches hands-on learning style
  2. Data Structures (Score: 88.3)
     Reason: Matches your interest in STEM; Builds on skills: programming; Strongly aligns with your career goals; Prerequisites met; Matches hands-on learn
ing style
  3. Advanced Calculus (Score: 73.3)
     Reason: Matches your interest in STEM; Builds on skills: problem_solving; Strongly aligns with your career goals; Prerequisites needed: MATH201; Matche
s hands-on learning style

Student B (Male, urban, high-income) - Top 3 Recommendations:
  1. Intro to Computer Science (Score: 96.7)
     Reason: Matches your interest in STEM; Good match for your skills (programming, problem_solving); Strongly aligns with your career goals; No prerequisi
tes required; Matches hands-on learning style
  2. Data Structures (Score: 88.3)
     Reason: Matches your interest in STEM; Builds on skills: programming; Strongly aligns with your career goals; Prerequisites met; Matches hands-on learn
ing style
  3. Advanced Calculus (Score: 73.3)
     Reason: Matches your interest in STEM; Builds on skills: problem_solving; Strongly aligns with your career goals; Prerequisites needed: MATH201; Matche
s hands-on learning style

✅ FAIRNESS ACHIEVED: Identical recommendations despite demographic differences


TEST CASE 2: Age-Independent Recommendations
--------------------------------------------------------------------------------

Young student (age 19): Mechanical Engineering (Score: 81.7)
Older student (age 45): Mechanical Engineering (Score: 81.7)

✅ FAIRNESS ACHIEVED: Age does not affect recommendations


TEST CASE 3: Fairness Audit Across Groups
--------------------------------------------------------------------------------

Fairness Audit Results:

Overall Fairness Score: 86.3/100

Difficulty Distribution Across Groups:
  female: {'beginner': 2, 'intermediate': 1, 'advanced': 2}
  male: {'beginner': 2, 'intermediate': 1, 'advanced': 2}
```



```
PS C:\Users\adepu\OneDrive\Desktop\AIAC END> C:/Users/adepu/AppData/Local/Programs/Python/Python313/python.exe Q2_Task2_fair_recommendations.py
Overall Fairness Score: 86.3/100

Difficulty Distribution Across Groups:
  female: {'beginner': 2, 'intermediate': 1, 'advanced': 2}
  male: {'beginner': 2, 'intermediate': 1, 'advanced': 2}
  young: {'advanced': 2, 'beginner': 2, 'intermediate': 1}
  older: {'advanced': 2, 'beginner': 2, 'intermediate': 1}

Equal Opportunity Metrics:
  female: 40.0% advanced courses
  male: 40.0% advanced courses
  young: 40.0% advanced courses
  older: 40.0% advanced courses

⚠️FAIRNESS CONCERN: Score below 90 indicates potential disparities


TEST CASE 4: Transparent Explanations
--------------------------------------------------------------------------------

Course: Data Structures (CS201)
Recommendation Score: 88.3/100

Explanation:
Matches your interest in STEM; Builds on skills: programming; Strongly aligns with your career goals; Prerequisites met; Matches hands-on learning style

This recommendation is based solely on your:
- Academic interests: STEM, technology
- Skills: programming, problem_solving
- Career goals: software_engineer
- Completed courses: CS101

NOT based on: gender, age, race, ethnicity, income, location, or other protected characteristics.


✅ TRANSPARENCY: Students receive clear explanations for recommendations


================================================================================
FAIRNESS IMPROVEMENTS SUMMARY
================================================================================

BEFORE (Task 1 - Biased):                | AFTER (Task 2 - Fair):
-----------------------------------------|-----------------------------------------
✗ Gender stereotypes in recommendations  | ✓ Gender-neutral scoring
✗ Age discrimination                     | ✓ Age-independent recommendations
✗ Socioeconomic bias                     | ✓ Income-independent access
✗ Geographic assumptions                 | ✓ Location-neutral recommendations
✗ Historical data perpetuates bias       | ✓ Focus on individual merit
```

OBSERVATION:

A batch-level *post-hoc reranker* was added to enforce exposure parity for a target course (e.g., placement_support) by boosting scores for users in under-exposed groups until parity (within epsilon) is reached.

- The approach is deterministic, auditable, and keeps sponsored items capped so commercial slots cannot fully override fairness adjustments.
- Diagnostics record per-iteration rate changes and boosts, enabling monitoring and rollback if undesired side effects appear.
- This is a light-weight, production-friendly method — for stronger guarantees use optimization solvers or causal approaches and always run server-side with logging/monitoring.