HTML(templates/index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Personalized Tutor Bot</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">

  <!-- Chart.js for dynamic charts -->
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

  <!-- Tippy.js for tooltips -->
  <script src="https://unpkg.com/@popperjs/core@2"></script>
  <script src="https://unpkg.com/tippy.js@6"></script>
  <link rel="stylesheet" href="https://unpkg.com/tippy.js@6/animations/scale.css"/>

  <!-- Google Fonts -->
  <link href="https://fonts.googleapis.com/css?family=Roboto:400,500,700&display=swap"
rel="stylesheet">

  <script>
    // Initialize tooltips and register outside-click to close profile dropdown
    document.addEventListener("DOMContentLoaded", () => {
      tippy('[data-tippy-content]', { animation: 'scale' });
      document.addEventListener("click", (event) => {
        const pd = document.getElementById("profile-dropdown");
        const menu = document.getElementById("profile-menu");
        if (!pd.contains(event.target)) {
          menu.classList.remove("active");
        }
      });
    });
    window.addEventListener("load", () => {
      document.getElementById("loading-screen").style.display = "none";
    });

    // Toggle password visibility
    function togglePassword() {
      const passField = document.getElementById("password") ||
document.getElementById("signup-password");
      const toggleIcon = document.getElementById("togglePassIcon") ||
document.getElementById("signup-togglePassIcon");
```

```
  if(passField.type === "password"){
    passField.type = "text";
    toggleIcon.innerText = "🙈";
  } else {
    passField.type = "password";
    toggleIcon.innerText = "👁️";
  }
}

// Show a section
function showSection(id) {
  document.querySelectorAll(".section").forEach(sec => sec.classList.remove("active"));
  document.getElementById(id).classList.add("active");
}

/* ---------- Authentication ---------- */
async function signupUser() {
  const username = document.getElementById("signup-username").value.trim();
  const password = document.getElementById("signup-password").value.trim();
  const res = await fetch("/signup", {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify({ username, password })
  });
  const data = await res.json();
  document.getElementById("signup-msg").innerText = data.message;
  if(data.success) { showSection("profile-section"); }
}

async function login() {
  const username = document.getElementById("username").value.trim();
  const password = document.getElementById("password").value.trim();
  const res = await fetch("/login", {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify({ username, password })
  });
  const data = await res.json();
  if(data.success) {
     document.getElementById("login-msg").innerText = "Login Successful!";
     showSection("dashboard-section");
  } else {
     document.getElementById("login-msg").innerText = "Login failed. Please try again.";
  }
```

```
}

async function logout() {
  await fetch("/logout", { method: "POST" });
  location.reload();
}

/* ---------------- Profile & Personalization ---------------- */
async function updateProfile() {
  const type = document.getElementById("profile-type").value;
  let level = "";
  let grade = "";
  let stream = "";
  let field = "";
  let otherText = "";
  if(type === "student") {
    level = document.getElementById("profile-level-select").value;
    if(level === "school") {
      grade = document.getElementById("profile-grade").value;
    } else if(level === "college") {
      stream = document.getElementById("profile-stream").value;
    } else if(level === "graduated") {
      otherText = document.getElementById("profile-degree").value;
    }
  } else if(type === "professional") {
    field = document.getElementById("profile-field").value;
  } else if(type === "other") {
    otherText = document.getElementById("profile-other").value;
  }
  const profile = { type, level, grade, stream, field, other: otherText };
  const res = await fetch("/update_profile", {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify(profile)
  });
  const data = await res.json();
  alert(data.message);
}

async function loadProfile() {
  const res = await fetch("/get_profile");
  const data = await res.json();
  document.getElementById("profile-info").innerHTML = data.profile;
}
```

```
  /* ---------------- Capacity Test ---------------- */
  async function startCapacityTest() {
    document.getElementById("capacity-test-questions").innerHTML = "<p>Your
personalized quizzes are generating…</p>";
    await loadCapacityTestQuestions();
    document.getElementById("start-capacity-btn").style.display = "none";
    document.getElementById("capacity-test-form").style.display = "block";
  }

  async function loadCapacityTestQuestions() {
    const res = await fetch("/get_capacity_test");
    const data = await res.json();
    let container = document.getElementById("capacity-test-questions");
    container.innerHTML = "";
    data.questions.forEach((q, i) => {
      let questionHTML = `<div class="question-item">
        <label><strong>${i+1}. ${q.question}</strong></label>`;
      if(q.type.toLowerCase().includes("mcq") || q.type.toLowerCase().includes("multiple")){
        let optionsHTML = "";
        let opts = q.options || q.choices;
        if(Array.isArray(opts)){
          optionsHTML = opts.map(opt => {
            return `<label class="quiz-option" data-tippy-content="Click to select">
                    <input type="radio" name="q${i+1}" value="${opt}"
style="display:none;">
                    <span>${opt}</span>
                  </label>`;
          }).join("");
        } else if(typeof opts === "object"){
          optionsHTML = Object.values(opts).map(opt => {
            return `<label class="quiz-option" data-tippy-content="Click to select">
                    <input type="radio" name="q${i+1}" value="${opt}"
style="display:none;">
                    <span>${opt}</span>
                  </label>`;
          }).join("");
        }
        questionHTML += `<div>${optionsHTML}</div>`;
      } else {
        questionHTML += `<input type="text" name="q${i+1}" placeholder="Your answer"
required>`;
      }
      questionHTML += `</div>`;
```

```
      container.innerHTML += questionHTML;
    });
    Array.from(document.querySelectorAll(".quiz-option")).forEach(option => {
      option.addEventListener("click", function(){
        const radio = this.querySelector("input[type='radio']");
        if(!radio) return;
        radio.checked = true;
        const groupName = radio.name;
        Array.from(document.getElementsByName(groupName)).forEach(input => {
          input.parentElement.classList.remove("selected");
        });
        this.classList.add("selected");
      });
    });
  }

  document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("capacity-test-form").addEventListener("submit", async (e)
=> {
      e.preventDefault();
      const formData = new FormData(document.getElementById("capacity-test-form"));
      const answers = {};
      formData.forEach((val, key) => { answers[key] = val; });
      const res = await fetch("/submit_capacity_test", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(answers)
      });
      const data = await res.json();
      let feedbackHTML = `<p>Your score: ${data.score} (${data.category})</p>`;
      if(data.details){
        feedbackHTML += `<table border="1" cellspacing="0" cellpadding="5">
                    <tr>
                      <th>Question</th>
                      <th>Your Answer</th>
                      <th>Correct Answer</th>
                      <th>Result</th>
                    </tr>`;
        data.details.forEach(item => {
          feedbackHTML += `<tr>
                      <td>${item.question}</td>
                      <td>${item.user_answer}</td>
                      <td>${item.correct_answer}</td>
                      <td>${item.result}</td>
```

```
                    </tr>`;
        });
        feedbackHTML += `</table>`;
      }
      document.getElementById("chartContainer").style.display = "block";
      feedbackHTML += `<button onclick="showSection('goal-section')">Continue to Goal Selection</button>
                  <button onclick="retakeCapacityTest()">Retake Capacity Test</button>`;
      document.getElementById("capacity-test-msg").innerHTML = feedbackHTML;

      const corr = data.details.filter(item => item.result.includes("✅")).length;
      const wrong = data.details.filter(item => item.result.includes("❌")).length;
      const unanswered = data.details.filter(item => item.user_answer === "(not answered)").length;
      const ctx = document.getElementById("capacityChart").getContext("2d");
      new Chart(ctx, {
        type: "pie",
        data: {
          labels: ["Correct", "Wrong", "Unanswered"],
          datasets: [{ data: [corr, wrong, unanswered], backgroundColor: ["#4CAF50", "#F44336", "#FFC107"] }]
        },
        options: { responsive: true, maintainAspectRatio: false }
      });
    });

    // Final Exam Submission.
    document.getElementById("final-exam-form").addEventListener("submit", async (e) => {
      e.preventDefault();
      clearInterval(timerInterval);
      const formData = new FormData(document.getElementById("final-exam-form"));
      const answers = {};
      formData.forEach((val, key) => { answers[key] = val; });
      const res = await fetch("/submit_final_exam", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(answers)
      });
      const data = await res.json();
      let feedbackHTML = `<p>Your final exam score: ${data.score} out of ${data.total}</p>`;
      feedbackHTML += `<div class="feedback-section">
                  <h3>Detailed Feedback & Study Recommendations</h3>
```

```
                    ${data.feedback}
                  </div>`;
      if(data.details){
        feedbackHTML += `<table border="1" cellspacing="0" cellpadding="5">
                    <tr>
                      <th>Question</th>
                      <th>Your Answer</th>
                      <th>Correct Answer</th>
                      <th>Result</th>
                    </tr>`;
        data.details.forEach(item => {
          feedbackHTML += `<tr>
                      <td>${item.question}</td>
                      <td>${item.user_answer}</td>
                      <td>${item.correct_answer}</td>
                      <td>${item.result}</td>
                    </tr>`;
        });
        feedbackHTML += `</table>`;
      }
      feedbackHTML += `<button onclick="retakeFinalExam()">Retake Final
Exam</button>`;
      document.getElementById("final-exam-feedback").innerHTML = feedbackHTML;
    });
  });

  async function selectGoal() {
    const goal = document.getElementById("goal").value.trim();
    const res = await fetch("/select_goal", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ goal })
    });
    const data = await res.json();
    document.getElementById("goal-msg").innerText = data.message;
    document.getElementById("learning-path").innerHTML = `<div
class="learning-path-result">Generating personalized path…</div>`;
    const res2 = await fetch("/get_learning_path");
    const pathData = await res2.json();
    setTimeout(() => {
      document.getElementById("learning-path").innerHTML = `<div
class="learning-path-result">${pathData.learning_path}</div>
      <button onclick="showSection('final-exam-section')">Proceed to Final
Exam</button>`;
```

```
  }, 2000);
}

let timerInterval;
function startExamTimer() {
  let secondsElapsed = 0;
  const examTimer = document.getElementById("exam-timer");
  timerInterval = setInterval(() => {
    let hrs = Math.floor(secondsElapsed / 3600);
    let mins = Math.floor((secondsElapsed % 3600) / 60);
    let secs = secondsElapsed % 60;
    let timeStr = (hrs < 10 ? "0" + hrs : hrs) + ":" +
                  (mins < 10 ? "0" + mins : mins) + ":" +
                  (secs < 10 ? "0" + secs : secs);
    examTimer.textContent = "Elapsed: " + timeStr;
    secondsElapsed++;
  }, 1000);
}

function toggleFinalExam() {
  const examDiv = document.getElementById("final-exam-questions");
  if(examDiv.style.display === "none" || examDiv.style.display === ""){
    examDiv.style.display = "block";
    loadFinalExamQuestions();
    startExamTimer();
    document.getElementById("toggle-final-exam-btn").innerText = "Hide Questions";
  } else {
    examDiv.style.display = "none";
    clearInterval(timerInterval);
    document.getElementById("toggle-final-exam-btn").innerText = "Show Questions";
  }
}

async function loadFinalExamQuestions() {
  const res = await fetch("/get_final_exam");
  const data = await res.json();
  let container = document.getElementById("final-exam-questions");
  container.innerHTML = "";
  data.questions.forEach((q, i) => {
    let questionHTML = `<div class="question-item">
      <label><strong>${i+1}. ${q.question}</strong></label><br>`;
    if(q.type.toLowerCase().includes("mcq") || q.type.toLowerCase().includes("multiple")){
      let inputField = "";
      let opts = q.options || q.choices;
```

```
        if(Array.isArray(opts)){
          inputField = opts.map(opt => {
            return `<label class="final-exam-option" data-tippy-content="Click to select">
                    <input type="radio" name="q${i+1}" value="${opt}"
style="display:none;">
                    <span>${opt}</span>
                  </label>`;
          }).join("");
        } else if(typeof opts === "object"){
          inputField = Object.values(opts).map(opt => {
            return `<label class="final-exam-option" data-tippy-content="Click to select">
                    <input type="radio" name="q${i+1}" value="${opt}"
style="display:none;">
                    <span>${opt}</span>
                  </label>`;
          }).join("");
        }
        questionHTML += inputField;
      } else {
        questionHTML += `<input type="text" name="q${i+1}" placeholder="Your answer"
required>`;
      }
      questionHTML += `</div>`;
      container.innerHTML += questionHTML;
    });
    Array.from(document.querySelectorAll(".final-exam-option")).forEach(option => {
      option.addEventListener("click", function(){
        const radio = this.querySelector("input[type='radio']");
        if(!radio) return;
        radio.checked = true;
        const groupName = radio.name;
        Array.from(document.getElementsByName(groupName)).forEach(input => {
          input.parentElement.classList.remove("selected");
        });
        this.classList.add("selected");
      });
    });
  }

  function retakeCapacityTest() {
    document.getElementById("capacity-test-msg").innerHTML = "";
    document.getElementById("capacity-test-form").reset();
    document.getElementById("start-capacity-btn").style.display = "block";
    document.getElementById("capacity-test-form").style.display = "none";
```

```
    loadCapacityTestQuestions();
  }

  function retakeFinalExam() {
    document.getElementById("final-exam-feedback").innerHTML = "";
    document.getElementById("final-exam-form").reset();
    document.getElementById("final-exam-questions").innerHTML = "";
    loadFinalExamQuestions();
  }

  async function sendChat() {
    const inputField = document.getElementById("chat-input");
    const msg = inputField.value.trim();
    if(!msg) return;
    document.getElementById("chat-box").innerHTML += `<div class="chat-message
user-message">You: ${msg}</div>`;
    inputField.value = "";
    const res = await fetch("/ask", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ message: msg })
    });
    const data = await res.json();
    document.getElementById("chat-box").innerHTML += `<div class="chat-message
bot-message">Bot: ${data.reply}</div>`;
    document.getElementById("chat-box").scrollTop =
document.getElementById("chat-box").scrollHeight;
  }

  let recognizing = false;
  let recognition;
  if('webkitSpeechRecognition' in window) {
    recognition = new webkitSpeechRecognition();
    recognition.continuous = false;
    recognition.interimResults = false;
    recognition.lang = 'en-US';
    recognition.onstart = function() {
      document.getElementById("chat-status").innerText = "Voice recognition activated...";
    };
    recognition.onerror = function(event) { console.error(event.error); };
    recognition.onend = function() {
      document.getElementById("chat-status").innerText = "";
      recognizing = false;
    };
```

```
  recognition.onresult = function(event) {
    const transcript = event.results[0][0].transcript;
    document.getElementById("chat-input").value = transcript;
  };
}
function toggleVoice() {
  if(!recognition){
    alert("Voice recognition not supported in this browser.");
    return;
  }
  if(recognizing){
    recognition.stop();
    recognizing = false;
  } else {
    recognition.start();
    recognizing = true;
  }
}

async function loadHistory() {
  const res = await fetch("/get_history");
  const data = await res.json();
  document.getElementById("history-container").innerHTML = data.history;
  showSection("history-section");
}

async function loadLeaderboard() {
  const res = await fetch("/get_leaderboard");
  const data = await res.json();
  document.getElementById("leaderboard-content").innerHTML = data.leaderboard;
  showSection("leaderboard-section");
}

async function loadDashboard() {
  const res = await fetch("/get_dashboard");
  const data = await res.json();
  document.getElementById("dashboard-content").innerHTML = data.dashboard;
  const ctx = document.getElementById("dashboardChart").getContext("2d");
  new Chart(ctx, {
    type: "bar",
    data: {
      labels: ["Quiz 1", "Quiz 2", "Quiz 3", "Final Exam"],
      datasets: [{
        label: "Score",
```

```
            data: [70, 80, 65, 75],
            backgroundColor: "#4CAF50"
          }]
        },
        options: { responsive: true, maintainAspectRatio: false }
      });
      showSection("dashboard-section");
    }

    async function saveNote() {
      const note = document.getElementById("note-input").value.trim();
      if(!note){
        alert("Please enter a note.");
        return;
      }
      const formData = new FormData();
      formData.append("note", note);
      const fileInput = document.getElementById("note-file");
      if(fileInput && fileInput.files.length > 0){
        formData.append("file", fileInput.files[0]);
      }
      const res = await fetch("/save_note", {
        method: "POST",
        body: formData
      });
      const data = await res.json();
      alert(data.message);
      loadNotes();
    }

    async function loadNotes() {
      const res = await fetch("/get_notes");
      const data = await res.json();
      document.getElementById("notes-output").innerHTML = data.notes;
    }
  </script>
</head>
<body>
  <!-- Loading Screen -->
  <div id="loading-screen">
    <img src="{{ url_for('static', filename='robot.gif') }}" alt="Loading...">
  </div>

  <!-- Header with Home Icon, Profile Button, and Navigation -->
```

```
  <header>
    <!-- Home Icon (top-left) -->
    <div id="home-icon" onclick="showSection('dashboard-section')">
      <img src="{{ url_for('static', filename='home_icon.png') }}" alt="Home"
class="home-img">
    </div>
    <h1>Personalized Tutor Bot</h1>
    <!-- Profile Button (icon with overlay label) -->
    <div id="profile-dropdown">
      <div class="profile-btn"
onclick="document.getElementById('profile-menu').classList.toggle('active')">
        <img id="profile-avatar" src="{{ url_for('static', filename='avatar.png') }}" alt="Profile
Avatar">
        <span class="profile-label">Profile</span>
      </div>
      <div id="profile-menu">
        <a href="#" onclick="showSection('profile-section')">Profile</a>
        <a href="#" onclick="showSection('dashboard-section')">Dashboard</a>
        <a href="#" onclick="loadHistory()">History</a>
        <a href="#" onclick="logout()" id="logout-link">Logout</a>
      </div>
    </div>
    <nav>
      <a href="#" onclick="showSection('capacity-test-section')">Capacity Test</a>
      <a href="#" onclick="showSection('goal-section')">Goal Selection</a>
      <a href="#" onclick="showSection('final-exam-section')">Final Exam</a>
      <a href="#" onclick="showSection('chat-section')">Chat Bot</a>
      <a href="#" onclick="showSection('notes-section')">Notes</a>
    </nav>
  </header>

  <!-- Dashboard / Home Page -->
  <div id="dashboard-section" class="section active">
    <h2>Welcome to Personalized Tutor Bot</h2>
    <p>This website is designed to help you assess your learning capacity and chart your
career journey with interactive tools and personalized study plans. Please sign up and log in
to access all features.</p>
    <ul>
      <li>Learning Capacity Test</li>
      <li>Goal Selection &amp;amp; Personal Learning Path</li>
      <li>Final Exam with Detailed Feedback</li>
      <li>Interactive Chat Bot</li>
      <li>Personal Notes &amp;amp; File Uploads</li>
    </ul>
```

```
    <p>Navigate using the menu above or via the profile dropdown.</p>
    <div id="dashboard-content"></div>
    <canvas id="dashboardChart" style="max-width: 100%; height: 300px;"></canvas>
    <!-- Example Progress Bar on Left Center -->
    <div id="progress-bar">
      <div id="progress-fill"></div>
    </div>
  </div>

  <!-- Login Section -->
  <div id="login-section" class="section">
    <h2>Login</h2>
    <input type="text" id="username" placeholder="Enter your username">
    <div class="form-group">
      <input type="password" id="password" placeholder="Enter your password">
      <span id="togglePassIcon" class="toggle-icon" onclick="togglePassword()">👁️</span>
    </div>
    <button onclick="login()">Login</button>
    <p id="login-msg"></p>
    <button onclick="showSection('signup-section')" class="secondary-button">Sign
Up</button>
  </div>

  <!-- Signup Section -->
  <div id="signup-section" class="section">
    <h2>Sign Up</h2>
    <input type="text" id="signup-username" placeholder="Choose a username" required>
    <div class="form-group">
      <input type="password" id="signup-password" placeholder="Choose a password"
required>
      <span id="signup-togglePassIcon" class="toggle-icon"
onclick="togglePassword()">👁️</span>
    </div>
    <button onclick="signupUser()">Sign Up</button>
    <p id="signup-msg"></p>
    <p class="small-note">Already have an account? Click Login above.</p>
  </div>

  <!-- Profile & Personalization Section -->
  <div id="profile-section" class="section">
    <h2>Profile &amp;amp; Personalization</h2>
    <p>Please tell us about yourself:</p>
    <label for="profile-type">I am a:</label>
```

```
    <select id="profile-type"
onchange="if(this.value===='other'){document.getElementById('other-input').style.display='bl
ock'; document.getElementById('student-options').style.display='none';
document.getElementById('professional-input').style.display='none';} else
if(this.value===='professional'){document.getElementById('professional-input').style.display=
'block'; document.getElementById('student-options').style.display='none';
document.getElementById('other-input').style.display='none';} else
{document.getElementById('student-options').style.display='block';
document.getElementById('professional-input').style.display='none';
document.getElementById('other-input').style.display='none';}">
      <option value="student" selected>Student</option>
      <option value="professional">Working Professional</option>
      <option value="other">Other</option>
    </select>
    <!-- Student Options -->
    <div id="student-options">
      <label for="profile-level-select">Select your level:</label>
      <select id="profile-level-select"
onchange="if(this.value===='school'){document.getElementById('grade-input').style.display='
block'; document.getElementById('college-stream').style.display='none';
document.getElementById('degree-input').style.display='none';} else
if(this.value===='college'){document.getElementById('grade-input').style.display='none';
document.getElementById('college-stream').style.display='block';
document.getElementById('degree-input').style.display='none';} else
if(this.value===='graduated'){document.getElementById('grade-input').style.display='none';
document.getElementById('college-stream').style.display='none';
document.getElementById('degree-input').style.display='block';} else
{document.getElementById('grade-input').style.display='none';
document.getElementById('college-stream').style.display='none';
document.getElementById('degree-input').style.display='none';}">
        <option value="school">School</option>
        <option value="college">College</option>
        <option value="graduated">Graduated</option>
      </select>
      <div id="grade-input" style="display: none;">
        <label for="profile-grade">Your Grade:</label>
        <input type="text" id="profile-grade" placeholder="e.g., 10th">
      </div>
      <div id="college-stream" style="display: none;">
        <label for="profile-stream">Your Stream:</label>
        <select id="profile-stream">
          <option value="engineering">Engineering</option>
          <option value="business">Business</option>
          <option value="arts">Arts</option>
```

```
          <option value="science">Science</option>
          <option value="medicine">Medicine</option>
          <option value="law">Law</option>
          <option value="architecture">Architecture</option>
          <option value="computer-science">Computer Science</option>
          <option value="psychology">Psychology</option>
          <option value="economics">Economics</option>
          <option value="education">Education</option>
          <option value="design">Design</option>
          <option value="social-science">Social Science</option>
          <option value="pharmacy">Pharmacy</option>
          <option value="nursing">Nursing</option>
          <option value="accounting">Accounting</option>
          <option value="marketing">Marketing</option>
          <option value="tourism">Tourism</option>
          <option value="journalism">Journalism</option>
          <option value="others">Others</option>
        </select>
      </div>
      <div id="degree-input" style="display: none;">
        <label for="profile-degree">Your Degree:</label>
        <input type="text" id="profile-degree" placeholder="e.g., B.Tech in Computer
Science">
      </div>
    </div>

    <!-- Professional Options -->
    <div id="professional-input" style="display: none;">
      <label for="profile-field">Your Field of Work:</label>
      <input type="text" id="profile-field" placeholder="e.g., IT, Finance, etc.">
    </div>

    <!-- Other Options -->
    <div id="other-input" style="display: none;">
      <label for="profile-other">Please specify:</label>
      <input type="text" id="profile-other" placeholder="Enter your details here">
    </div>
    <button onclick="updateProfile()">Save Profile</button>
    <div id="profile-info"></div>
  </div>

  <!-- Capacity Test Section -->
  <div id="capacity-test-section" class="section">
    <h2>Learning Capacity Test</h2>
```

```
    <button id="start-capacity-btn" onclick="startCapacityTest()">Start Quiz</button>
    <form id="capacity-test-form" style="display: none; overflow-y: auto; max-height:
100vh;">
      <div id="capacity-test-questions"></div>
      <button type="submit">Submit Test</button>
    </form>
    <div id="capacity-test-msg"></div>
    <!-- Centered Pie Chart -->
    <div id="chartContainer">
      <canvas id="capacityChart" width="300" height="300"></canvas>
    </div>
  </div>

  <!-- Goal Selection Section -->
  <div id="goal-section" class="section">
    <h2>Goal Selection</h2>
    <input type="text" id="goal" placeholder="Enter your career goal (e.g., doctor,
engineer)">
    <button onclick="selectGoal()">Set Goal</button>
    <p id="goal-msg"></p>
    <div id="learning-path"></div>
  </div>

  <!-- Final Exam Section -->
  <div id="final-exam-section" class="section">
    <h2>Final Exam</h2>
    <button id="toggle-final-exam-btn" onclick="toggleFinalExam()">Show
Questions</button>
    <span id="exam-timer"></span>
    <form id="final-exam-form">
      <div id="final-exam-questions" style="display: none; overflow-y: auto; max-height:
100vh;"></div>
      <button type="submit">Submit Final Exam</button>
    </form>
    <div id="final-exam-feedback"></div>
  </div>

  <!-- Chat Bot Section -->
  <div id="chat-section" class="section">
    <h2>Chat Bot</h2>
    <div class="bot-panel">
      <div class="chat-box" id="chat-box"></div>
      <div class="chat-input-group">
```

```
      <textarea id="chat-input" class="chat-input" placeholder="Type your message..."
rows="1"></textarea>
      <button class="chat-send-btn" onclick="sendChat()">
        <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="#fff"
viewBox="0 0 24 24">
          <path d="M2 21l21-9L2 3v7l15 2-15 2v7z"/>
        </svg>
      </button>
      <button class="chat-mic-btn" onclick="toggleVoice()">
        <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="#fff"
viewBox="0 0 24 24">
          <path d="M12 14a3 3 0 0 0 3-3V5a3 3 0 0 0-6 0v6a3 3 0 0 0 3 3zm5-3a5 5 0 0 1-10
0H5a7 7 0 0 0 14 0z"/>
        </svg>
      </button>
    </div>
    <p id="chat-status" class="small-note"></p>
  </div>
 </div>

 <!-- History Section -->
 <div id="history-section" class="section">
  <h2>History</h2>
  <div id="history-container"></div>
 </div>

 <!-- Leaderboard Section -->
 <div id="leaderboard-section" class="section">
  <h2>Live Leaderboard</h2>
  <div id="leaderboard-content">
    <p>User: {% if session.get("username") %}{{ session.get("username") }}{% else
%}Anonymous{% endif %}</p>
    <p>Your Score: <strong>{% if session.get("capacity_test_score") %}{{
session.get("capacity_test_score") }}{% else %}0{% endif %}</strong></p>
    <p>No additional leaderboard data available.</p>
  </div>
 </div>

 <!-- Personal Notes Section -->
 <div id="notes-section" class="section">
  <h2>Personal Notes</h2>
  <textarea id="note-input" placeholder="Type your note here..."></textarea>
  <input type="file" id="note-file" accept="image/*,audio/*,video/*">
  <button onclick="saveNote()">Save Note</button>
```

```
    <div id="notes-output">
      <p>Your saved notes will appear here.</p>
    </div>
  </div>

  <!-- Tutor Bot Image (clickable to open Chat Bot) -->
  <img id="tutor-bot" src="{{ url_for('static', filename='tutor_bot.png') }}" alt="Tutor Bot"
onclick="showSection('chat-section')">
</body>
</html>
```

CSS(static/styles.css):

```
/* ---------- Global Styles ---------- */
body {
  margin: 0;
  font-family: 'Roboto', Arial, sans-serif;
  font-size: 18px;
  background-color: #fdfdfd; /* Elegant off-white background */
  color: #333;
  transition: background 0.5s ease;
}

/* ---------- Loader Animation ---------- */
#loading-screen {
  position: fixed;
  width: 100%;
  height: 100%;
  background: #fff;
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 1000;
}
#loading-screen img {
  width: 300px;
  animation: slideUp 2s ease-out forwards;
}
@keyframes slideUp {
```

```
  0% { transform: translateY(100%); opacity: 1; }
  100% { transform: translateY(0); opacity: 0; }
}

/* ---------- Header & Navigation ---------- */
header {
  background: #f8c1c1;
  padding: 15px 20px;
  text-align: center;
  position: relative;
}
header h1 {
  margin: 0;
  font-size: 2.5em;
  color: #dc143c;
}

/* Home Icon on Top-Left */
#home-icon {
  position: fixed;
  top: 10px;
  left: 10px;
  cursor: pointer;
}
.home-img {
  width: 40px;
  height: 40px;
  border-radius: 50%;
}

/* Navigation Menu */
nav {
  display: flex;
  justify-content: center;
  gap: 10px;
  margin-top: 15px;
}
nav a {
  background: #fff;
  color: #dc143c;
  text-decoration: none;
  padding: 10px 15px;
  border: 2px solid #dc143c;
  border-radius: 20px;
```

```
  font-size: 1.2em;
  transition: background 0.3s, transform 0.2s;
}
nav a:hover {
  background: #ffb6c1;
  transform: scale(1.05);
}

/* ---------- Profile Dropdown (Top-Right) ---------- */
#profile-dropdown {
  position: fixed;
  top: 10px;
  right: 10px;
  z-index: 1100;
  display: flex;
  align-items: center;
}
.profile-btn {
  display: flex;
  align-items: center;
  cursor: pointer;
  background: transparent;
  position: relative;
}
#profile-avatar {
  width: 40px;
  height: 40px;
  border-radius: 50%;
  border: 2px solid #dc143c;
  transition: transform 0.3s ease;
}
.profile-btn:hover #profile-avatar {
  transform: scale(1.1);
}
.profile-label {
  position: absolute;
  bottom: -18px;
  width: 100%;
  text-align: center;
  font-size: 0.9em;
  color: #dc143c;
  font-weight: bold;
}
#profile-menu {
```

```
  display: none;
  position: absolute;
  top: 55px;
  right: 0;
  background: #fff;
  border: 1px solid #ccc;
  border-radius: 5px;
  min-width: 150px;
  box-shadow: 0 4px 8px rgba(0,0,0,0.1);
  animation: slideDown 0.3s ease-out;
}
#profile-menu.active {
  display: block;
}
@keyframes slideDown {
  from { opacity: 0; transform: translateY(-10px); }
  to { opacity: 1; transform: translateY(0); }
}
#profile-menu a {
  display: block;
  padding: 10px;
  color: #333;
  text-decoration: none;
  border-bottom: 1px solid #eee;
}
#profile-menu a:hover {
  background: #f2f2f2;
}
#logout-link {
  margin-top: 10px;
  text-align: center;
  color: #dc143c;
  font-weight: bold;
}

/* ---------- Progress Bar (Left Center) ---------- */
#progress-bar {
  position: fixed;
  top: 50%;
  left: 0;
  transform: translateY(-50%);
  width: 8px;
  height: 60%;
  background: #eee;
```

```
  border-radius: 0 5px 5px 0;
  z-index: 100;
}
#progress-fill {
  width: 100%;
  height: 0; /* Update dynamically with JS if desired */
  background: #4CAF50;
  border-radius: 0 5px 5px 0;
  transition: height 0.3s ease;
}

/* ---------- Sections ---------- */
.section {
  display: none;
  max-width: 900px;
  margin: 30px auto;
  background: #fff;
  color: #333;
  padding: 35px;
  border-radius: 10px;
  box-shadow: 0 6px 15px rgba(0,0,0,0.1);
  animation: fadeIn 0.8s ease;
}
.section.active {
  display: block;
}
.section h2 {
  color: #dc143c;
  font-size: 2em;
}
@keyframes fadeIn {
  from { opacity: 0; transform: translateY(20px); }
  to { opacity: 1; transform: translateY(0); }
}

/* ---------- Form Elements ---------- */
input, textarea, button, select {
  width: 100%;
  padding: 10px;
  margin: 10px 0;
  font-size: 1em;
  box-sizing: border-box;
  border: 1px solid #ccc;
  border-radius: 4px;
```

```
}
textarea {
  resize: vertical;
  min-height: 35px;
  min-width: 120px;
}
button {
  background: #dc143c;
  color: #fff;
  border: none;
  cursor: pointer;
  transition: background 0.2s ease, transform 0.2s ease;
}
button:hover {
  background: #c1122b;
  transform: scale(1.05);
}
.secondary-button {
  background: #fff;
  border: 2px solid #dc143c;
  color: #dc143c;
}
.small-note {
  font-size: 0.9em;
  color: #888;
}

/* ---------- Profile Section Specific ---------- */
#profile-info {
  margin-top: 15px;
}

/* ---------- Quiz & Exam Containers ---------- */
.question-item {
  margin-bottom: 15px;
}
.quiz-option, .final-exam-option {
  display: inline-block;
  border: 2px solid #dc143c;
  border-radius: 8px;
  padding: 8px 12px;
  margin: 5px;
  cursor: pointer;
  transition: background 0.3s;
```

```
  text-align: center;
  font-weight: 500;
}
.quiz-option:hover, .final-exam-option:hover {
  background: #ffb6c1;
}
.selected {
  background-color: #ffe4e1;
  border-color: #c1122b;
  font-weight: bold;
}

/* ---------- Learning Path Section ---------- */
.learning-path-result {
  background: #fff;
  padding: 15px;
  border-radius: 8px;
  margin-top: 10px;
  text-align: left;
  line-height: 1.6;
  border: 2px solid #dc143c;
  color: #333;
}

/* ---------- Final Exam Timer ---------- */
#exam-timer {
  font-size: 1.2em;
  font-weight: 700;
  margin-left: 10px;
}

/* ---------- Chat Bot Panel ---------- */
.bot-panel {
  background: #fff;
  color: #333;
  padding: 15px;
  border-radius: 8px;
  margin-top: 15px;
  border: 2px solid #dc143c;
  height: 400px;
  display: flex;
  flex-direction: column;
}
.chat-box {
```

```
  background: #fff;
  color: #333;
  padding: 10px;
  flex: 1;
  overflow-y: auto;
  border-radius: 6px;
  margin-bottom: 10px;
  border: 2px solid #dc143c;
}
.chat-message {
  margin-bottom: 10px;
  line-height: 1.4;
}
.chat-message.user-message {
  background: #ffebef;
  padding: 8px;
  border-radius: 5px;
  border: 1px solid #dc143c;
}
.chat-message.bot-message {
  background: #ffe4e1;
  padding: 8px;
  border-radius: 5px;
  border: 1px solid #dc143c;
}
.chat-input-group {
  display: flex;
  align-items: center;
  gap: 5px;
}
.chat-input {
  flex: 1;
  padding: 15px; /* Increased padding for larger typing area */
  font-size: 1em;
  min-width: 200px; /* Increased width */
  border: 1px solid #ccc;
  border-radius: 4px;
  resize: vertical;
  min-height: 50px; /* Increased height */
}
.chat-send-btn, .chat-mic-btn {
  background: #dc143c;
  border: none;
  border-radius: 4px;
```

```
  width: 50px; /* Square button */
  height: 50px; /* Square button */
  cursor: pointer;
  transition: transform 0.2s ease, background 0.2s;
  display: flex;
  align-items: center;
  justify-content: center;
}
.chat-send-btn:hover, .chat-mic-btn:hover {
  background: #c1122b;
  transform: scale(1.05);
}
.chat-send-btn svg, .chat-mic-btn svg {
  width: 20px; /* Larger icons */
  height: 20px;
}
.chat-buttons {
  display: flex;
  align-items: center;
  gap: 5px;
}

/* ---------- Centered Pie Chart Container (Capacity Test Only) ---------- */
#chartContainer {
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  width: 300px;
  height: 300px;
  background: #fff;
  border: 3px solid #dc143c;
  border-radius: 10px;
  padding: 10px;
  z-index: 1200;
  display: none;
}

/* ---------- History Section ---------- */
#history-container ul {
  list-style-type: disc;
  padding-left: 20px;
  color: #333;
}
```

```
/* ---------- Leaderboard & Personal Notes Sections ---------- */
#leaderboard-section,
#notes-section {
  background: #fff;
  max-width: 900px;
  margin: 30px auto;
  padding: 35px;
  border-radius: 10px;
  box-shadow: 0 6px 15px rgba(0,0,0,0.1);
  color: #333;
}
#leaderboard-section h2,
#notes-section h2 {
  color: #dc143c;
}

/* ---------- Responsive Media Queries ---------- */
@media (max-width: 768px) {
  header h1 {
    font-size: 1.8em;
  }
  nav a {
    font-size: 1em;
    padding: 8px 12px;
  }
  .section {
    padding: 20px;
    margin: 15px;
  }
  #profile-avatar, .home-img {
    width: 35px;
    height: 35px;
  }
  .chat-input {
    min-width: 180px;
  }
}

/* ---------- Tutor Bot Image ---------- */
#tutor-bot {
  position: fixed;
  bottom: 20px;
  left: 20px;
```

```
  width: 100px;
  height: 100px;
  border-radius: 50%;
  animation: pulse 3s infinite;
  cursor: pointer;
  z-index: 100;
}
@keyframes pulse {
  0% { transform: scale(1); opacity: 1; }
  50% { transform: scale(1.1); opacity: 0.8; }
  100% { transform: scale(1); opacity: 1; }
}
```

Python(app.py):

```
import json
from flask import Flask, render_template, request, jsonify, session
import openai
import datetime
import os

app = Flask(__name__)
app.secret_key = "super-secret-key"  # Replace with a robust key in production.

# --------------------------
# Persistence Files & Helper Functions
# --------------------------
USERS_FILE = "users.json"
NOTES_FILE = "notes.json"

def load_json_file(filename):
    if os.path.exists(filename):
        with open(filename, 'r') as f:
            return json.load(f)
    return {}

def save_json_file(filename, data):
    with open(filename, 'w') as f:
        json.dump(data, f)
```

```
# Load persistent user data and notes.
users = load_json_file(USERS_FILE)
notes_data = load_json_file(NOTES_FILE)

# Ensure the uploads folder exists.
if not os.path.exists("uploads"):
    os.makedirs("uploads")

# --------------------------
# Configure OpenRouter API – update with your API key.
# --------------------------
openai.api_base = "https://openrouter.ai/api/v1"
openai.api_key = "Here place your api key from ex. deepseek rover v2"  # Replace with your valid API key.

def query_model(prompt_text):
    try:
        response = openai.ChatCompletion.create(
            model="deepseek/deepseek-prover-v2:free",
            messages=[{"role": "user", "content": prompt_text}]
        )
        return response.choices[0].message.content
    except Exception as e:
        print("Error querying model:", e)
        return ""

def get_json_output(prompt):
    output_text = query_model(prompt)
    if output_text.startswith("```"):
        lines = output_text.splitlines()
        if lines and lines[0].startswith("```"):
            lines = lines[1:]
        if lines and lines[-1].strip() == "```":
            lines = lines[:-1]
        output_text = "\n".join(lines)
    print("Cleaned output:", output_text)
    try:
        return json.loads(output_text)
    except Exception as e:
        print("JSON parse error:", e)
        return {"questions": []}

def clean_html(text):
```

```
    if text.startswith("```") and text.endswith("```"):
        lines = text.splitlines()
        if lines and lines[0].startswith("```"):
            lines = lines[1:]
        if lines and lines[-1].strip() == "```":
            lines = lines[:-1]
        return "\n".join(lines)
    return text

def current_timestamp():
    return datetime.datetime.now().isoformat(timespec='seconds')

# --------------------------
# USER AUTHENTICATION
# --------------------------
@app.route("/signup", methods=["POST"])
def signup():
    data = request.get_json()
    username = data.get("username", "").strip()
    password = data.get("password", "").strip()
    global users
    if username in users:
        return jsonify({"success": False, "message": "Username already exists."})
    users[username] = password
    save_json_file(USERS_FILE, users)
    # Initialize the profile; extra fields for graduates:
    session["profile"] = {"type": "", "level": "", "grade": "", "stream": "", "field": "", "other":
""}
    return jsonify({"success": True, "message": "Signup successful! Please log in."})

@app.route("/login", methods=["POST"])
def login_route():
    data = request.get_json()
    username = data.get("username", "").strip()
    password = data.get("password", "").strip()
    if users.get(username) == password:
        session["username"] = username
        session.setdefault("history", []).append(f"{current_timestamp()} - {username} logged
in.")
        if "profile" not in session:
            session["profile"] = {"type": "", "level": "", "grade": "", "stream": "", "field": "",
"other": ""}
        return jsonify({"success": True})
    return jsonify({"success": False})
```

```
@app.route("/logout", methods=["POST"])
def logout():
    session.setdefault("history", []).append(f"{current_timestamp()} - {session.get('username',
'Unknown')} logged out.")
    session.clear()
    return jsonify({"success": True})

# --------------------------
# LANGUAGE (English Only)
# --------------------------
@app.route("/set_language", methods=["POST"])
def set_language_route():
    data = request.get_json()
    language = data.get("language", "en").strip()
    session["language"] = language
    session.setdefault("history", []).append(f"{current_timestamp()} - Language set to
{language}.")
    return jsonify({"success": True, "language": language})

# --------------------------
# PROFILE & PERSONALIZATION
# --------------------------
@app.route("/update_profile", methods=["POST"])
def update_profile_route():
    data = request.get_json()
    profile = {
        "type": data.get("type", ""),
        "level": data.get("level", ""),
        "grade": data.get("grade", ""),    # For school students.
        "stream": data.get("stream", ""),  # For college students.
        "field": data.get("field", ""),    # For professionals.
        "other": data.get("other", "")
    }
    session["profile"] = profile
    session.setdefault("history", []).append(f"{current_timestamp()} - Profile updated:
{profile}")
    return jsonify({"success": True, "message": "Profile updated successfully!"})

@app.route("/get_profile", methods=["GET"])
def get_profile_route():
    profile = session.get("profile", {"type": "Not Set", "level": "Not Set", "grade": "Not Set",
"stream": "Not Set", "field": "Not Set", "other": ""})
    # Build HTML showing the stored profile info.
```

```
    html = (
        f"<h3>Profile Details</h3>"
        f"<p><strong>Type:</strong> {profile['type']}</p>"
        f"<p><strong>Level:</strong> {profile['level']}</p>"
        f"<p><strong>Grade (if school):</strong> {profile['grade'] or 'N/A'}</p>"
        f"<p><strong>Stream (if college):</strong> {profile['stream'] or 'N/A'}</p>"
        f"<p><strong>Field (if professional):</strong> {profile['field'] or 'N/A'}</p>"
        f"<p><strong>Other:</strong> {profile.get('other', '') or 'N/A'}</p>"
    )
    return jsonify({"profile": html})

# -------------------------
# GOAL SELECTION & LEARNING PATH
# -------------------------
@app.route("/select_goal", methods=["POST"])
def select_goal_route():
    data = request.get_json()
    goal = data.get("goal", "").strip()
    if goal:
        session["goal"] = goal
        session.setdefault("history", []).append(f"{current_timestamp()} - Goal Selected:
{goal}")
        return jsonify({"success": True, "message": f"Your goal '{goal}' has been selected."})
    return jsonify({"error": "No goal provided"}), 400

@app.route("/get_learning_path", methods=["GET"])
def get_learning_path_route():
    goal = session.get("goal", "General")
    prompt = (
        f"Provide an extremely detailed study plan for becoming a {goal} in India tailored for a
user with this profile: {session.get('profile', {})}. "
        "Break down the plan step-by-step with clear recommendations and direct links (video
tutorials, study materials). "
        "Output must be plain HTML (using headings, paragraphs, lists and links that open in a
new window). "
        "First output the text 'Generating personalized path…' and then the complete plan."
    )
    learning_path = query_model(prompt)
    learning_path = clean_html(learning_path)
    session.setdefault("history", []).append(f"{current_timestamp()} - Personalized learning
path generated.")
    return jsonify({"learning_path": learning_path})

# -------------------------
```

```
# CAPACITY TEST ENDPOINTS
# -------------------------
@app.route("/get_capacity_test", methods=["GET"])
def get_capacity_test_endpoint():
    prompt = (
        "Generate a JSON object with key 'questions' that is an array of 15 mixed-format IQ/GK questions tailored to the user's profile. "
        "If the user is a student: If level is 'school', include grade-appropriate IQ questions (including one about age or grade), "
        "if 'college', include advanced questions concerning their stream, and if 'graduated', include questions about their degree. "
        "For working professionals, include questions relating to their field; for Other, generate detailed free-form questions. "
        "Ensure that ~75% of questions are multiple choice with exactly 3 options and ~25% are fill-in-the-blanks. "
        "Each question must include a key 'answer' with the correct answer in lowercase. Output valid JSON."
    )
    result = get_json_output(prompt)
    session["capacity_test_answers"] = {
        f"q{idx+1}": q.get("answer", "").strip().lower()
        for idx, q in enumerate(result.get("questions", []))
    }
    session.setdefault("history", []).append(f"{current_timestamp()} - Capacity Test questions fetched.")
    return jsonify(result)

@app.route("/submit_capacity_test", methods=["POST"])
def submit_capacity_test_route():
    responses = request.get_json()
    stored = session.get("capacity_test_answers", {})
    score, details = 0, []
    for key, correct in stored.items():
        user_ans = responses.get(key, "").strip().lower()
        is_correct = (user_ans == correct) and user_ans != ""
        result_str = "✅ Correct" if is_correct else "❌ Wrong"
        if is_correct: score += 1
        details.append({
            "question": f"Question {key[1:]}",
            "user_answer": user_ans if user_ans else "(not answered)",
            "correct_answer": correct if correct else "(not provided)",
            "result": result_str
        })
    percent = (score / len(stored)) * 100 if stored else 0
```

```
    category = "Fast Learner" if percent >= 70 else "Average Learner" if percent >= 40 else
"Slow Learner"
    session["capacity_test_score"] = score
    session["learner_category"] = category
    session.setdefault("history", []).append(f"{current_timestamp()} - Capacity Test:
{score}/{len(stored)} - {category}")
    return jsonify({"score": score, "category": category, "details": details})

# -------------------------
# FINAL EXAM ENDPOINTS
# -------------------------
@app.route("/get_final_exam", methods=["GET"])
def get_final_exam():
    goal = session.get("goal", "General")
    prompt = (
        f"Generate a JSON object with key 'questions' that is an array of 25 deep, conceptual
final exam questions for a career in {goal}. "
        "Exactly 7 must be multiple choice with 3 options each; the remaining 18 should be
fill-in-the-blanks. "
        "Each question must include a key 'answer' with the correct answer in lowercase. "
        "Output valid JSON including keys 'id', 'question', 'type', and (for MCQs) an 'options'
array."
    )
    result = get_json_output(prompt)
    session["final_exam_answers"] = {
        f"q{idx+1}": q.get("answer", "").strip().lower()
        for idx, q in enumerate(result.get("questions", []))
    }
    session.setdefault("history", []).append(f"{current_timestamp()} - Final Exam questions
fetched.")
    return jsonify(result)

@app.route("/submit_final_exam", methods=["POST"])
def submit_final_exam():
    responses = request.get_json()
    correct_answers = session.get("final_exam_answers", {})
    score, details = 0, []
    for key, correct in correct_answers.items():
        user_ans = responses.get(key, "").strip().lower()
        is_correct = (user_ans == correct) and user_ans != ""
        result_str = "✅ Correct" if is_correct else "❌ Wrong"
        if is_correct: score += 1
        details.append({
            "question": f"Question {key[1:]}",
```

```
            "user_answer": user_ans if user_ans else "(not answered)",
            "correct_answer": correct if correct else "(not provided)",
            "result": result_str
        })
    total = len(correct_answers)
    prompt = (
        f"Act as an enthusiastic mentor and provide detailed, step-by-step feedback for a final exam in the field '{session.get('goal', 'General')}'. "
        f"The student scored {score} out of {total}. Include elaborate study recommendations, clear explanations, and direct links to video tutorials and study materials. "
        "Output as pure HTML with clear headings, paragraphs, lists, and clickable links (with target='_blank')."
    )
    feedback = query_model(prompt)
    feedback = clean_html(feedback)
    if not feedback:
        feedback = "Your final exam has been graded. Please review your weak areas and study diligently!"
    session.setdefault("history", []).append(f"{current_timestamp()} - Final Exam: {score}/{total}")
    return jsonify({"score": score, "total": total, "details": details, "feedback": feedback})

# -------------------------
# AI CHAT BOT ENDPOINT
# -------------------------
@app.route("/ask", methods=["POST"])
def ask_endpoint():
    data = request.get_json()
    message = data.get("message", "").strip()
    greetings = ["hi", "hey", "hello", "howdy"]
    if message.lower() in greetings:
        response = "Hey there! How can I help you today?"
    else:
        prompt = f"Respond as a friendly mentor with detailed answers to: {message}"
        response = query_model(prompt)
        if not response:
            response = "I'm sorry, I could not process your request. Please try again later! 🤖"
        response = response.replace("#", "<br>")
    session.setdefault("history", []).append(f"{current_timestamp()} - Chat: {message} | Bot: {response}")
    return jsonify({"reply": response})

# -------------------------
# HISTORY RETRIEVAL (Per-User)
```

```
# --------------------------
@app.route("/get_history", methods=["GET"])
def get_history_route():
    user_history = session.get("history", [])
    history_html = "<ul>"
    for item in user_history:
        history_html += f"<li>{item}</li>"
    history_html += "</ul>"
    return jsonify({"history": history_html})

# --------------------------
# LEADERBOARD (Per-User)
# --------------------------
@app.route("/get_leaderboard", methods=["GET"])
def get_leaderboard():
    user = session.get("username", "Anonymous")
    score = session.get("capacity_test_score", 0)
    html = f"<h3>Live Leaderboard</h3><p>User: {user}</p><p>Your Score: {score}</p>"
    return jsonify({"leaderboard": html})

# --------------------------
# DASHBOARD (Progress Tracking – Placeholder)
# --------------------------
@app.route("/get_dashboard", methods=["GET"])
def get_dashboard():
    progress = {
        "quizzesTaken": 5,
        "averageScore": 65,
        "lastQuiz": "Capacity Test - 70%"
    }
    html = (
        "<h3>Progress Dashboard</h3>"
        "<ul>"
        f"<li>Quizzes Taken: {progress['quizzesTaken']}</li>"
        f"<li>Average Score: {progress['averageScore']}%</li>"
        f"<li>Last Quiz: {progress['lastQuiz']}</li>"
        "</ul>"
    )
    return jsonify({"dashboard": html})

# --------------------------
# COMMUNITY Q&A (Placeholder)
# --------------------------
@app.route("/get_community", methods=["GET"])
```

```
def get_community():
    html = (
        "<h3>Community Q&amp;amp;A</h3>"
        "<p>Welcome to the community forum! Ask questions and share your insights.</p>"
    )
    return jsonify({"community": html})

# --------------------------
# PERSONAL NOTES (File Uploads Supported)
# --------------------------
@app.route("/save_note", methods=["POST"])
def save_note():
    note_text = request.form.get("note", "").strip()
    file = request.files.get("file")
    filename = ""
    if file:
        filename = file.filename
        file_path = os.path.join("uploads", filename)
        file.save(file_path)
    if not note_text and not filename:
        return jsonify({"success": False, "message": "Empty note."})
    notes = session.get("notes", [])
    note_entry = f"{current_timestamp()} - {note_text}"
    if filename:
        note_entry += f" [File: {filename}]"
    notes.append(note_entry)
    session["notes"] = notes
    save_json_file(NOTES_FILE, notes)
    return jsonify({"success": True, "message": "Note saved!"})

@app.route("/get_notes", methods=["GET"])
def get_notes():
    notes = session.get("notes", [])
    html = "<h3>Your Notes</h3><ul>"
    for note in notes:
        html += f"<li>{note}</li>"
    html += "</ul>"
    return jsonify({"notes": html})

# --------------------------
# HOME ROUTE – Permanent Dashboard
# --------------------------
@app.route("/")
def index():
```

```
    return render_template("index.html")

if __name__ == "__main__":
    app.run(debug=True)
```
