let's remove everything we did to connect or run the actual ai for now. we should focus on the frontend only. let's just make sure this returns something . connecting with ai will be done later, afyer making sure it returns something. this is my codes.// app.js

```
// VIEW SWITCHING
function showView(id) {
  document.querySelectorAll('.view')
    .forEach(sec => sec.classList.toggle('hidden', sec.id !== id));
}

// Reset login form fields
function resetLoginForm() {
  document.getElementById('loginId').value = '';
  document.getElementById('loginPassword').value = '';
}

// “Go Back” buttons always take you home
document.querySelectorAll('.back').forEach(btn =>
  btn.addEventListener('click', () => showView('home'))
);

// Navigation buttons
document.querySelectorAll('[data-view]').forEach(btn =>
  btn.addEventListener('click', () => showView(btn.dataset.view))
);

// Start on home
showView('home');


// DROP-ZONE UTILITY
function setupDropZone(zoneId, inputId, listId) {
  const zone = document.getElementById(zoneId),
        inp  = document.getElementById(inputId),
        lst  = document.getElementById(listId);

  function refresh() {
    lst.innerHTML = '';
    Array.from(inp.files).forEach(f => {
      const li = document.createElement('li');
      li.textContent = f.name;
      lst.appendChild(li);
    });
```

```
  }

  zone.addEventListener('dragover', e => {
    e.preventDefault();
    zone.classList.add('dragover');
  });
  zone.addEventListener('dragleave', () => {
    zone.classList.remove('dragover');
  });
  zone.addEventListener('drop', e => {
    e.preventDefault();
    zone.classList.remove('dragover');
    inp.files = e.dataTransfer.files;
    refresh();
  });
  inp.addEventListener('change', refresh);
}

['materials','template','answerKey','student']
  .forEach(pref => setupDropZone(
    pref + 'Drop',
    pref + 'Input',
    pref + 'List'
  ));


// GENERATE → PREVIEW
document.getElementById('genExamBtn').onclick = () => {
  const counts = {
    truefalse: +document.getElementById('count-truefalse').value,
    multiple:  +document.getElementById('count-multiple').value,
    short:     +document.getElementById('count-short').value,
    numeric:   +document.getElementById('count-numeric').value,
  };
  const total = Object.values(counts).reduce((a,b)=>a+b, 0);
  const cont  = document.getElementById('questionsContainer');
  cont.innerHTML = '';
  for (let i = 1; i <= total; i++) {
    const d = document.createElement('div');
    d.className = 'question-line';
    d.draggable = true;
    d.innerHTML = `
      <span>${i}.</span>
      <span class="blank-line"></span>
```

```
      <button class="remove-btn">Remove</button>
    `;
    cont.appendChild(d);
  }
  showView('preview');
};

// REORDER & REMOVE QUESTIONS
const cont = document.getElementById('questionsContainer');
let dragSrc = null;

cont.addEventListener('dragstart', e => {
  if (!e.target.classList.contains('question-line')) return;
  dragSrc = e.target;
  e.target.classList.add('dragging');
});
cont.addEventListener('dragend', e => {
  e.target.classList.remove('dragging');
});
cont.addEventListener('dragover', e => {
  e.preventDefault();
  const tgt = e.target.closest('.question-line');
  if (tgt && tgt !== dragSrc) {
    const { top, height } = tgt.getBoundingClientRect();
    const mid = top + height / 2;
    cont.insertBefore(dragSrc, e.clientY < mid ? tgt : tgt.nextSibling);
    renumber();
  }
});
cont.addEventListener('click', e => {
  if (e.target.classList.contains('remove-btn')) {
    e.target.closest('.question-line').remove();
    renumber();
  }
});
function renumber() {
  Array.from(cont.children).forEach((ln, i) => {
    ln.querySelector('span').textContent = `${i+1}.`;
  });
}

// INSERT DIRECTLY & AI STUB
document.getElementById('insertDirect').onclick = () => {
  const q = document.getElementById('newQuestion').value.trim();
```

```
  if (!q) return alert('Enter a question.');
  const idx = cont.children.length + 1;
  const d = document.createElement('div');
  d.className = 'question-line';
  d.draggable = true;
  d.innerHTML = `<span>${idx}.</span><span>${q}</span><button
class="remove-btn">Remove</button>`;
  cont.appendChild(d);
  document.getElementById('newQuestion').value = '';
  renumber();
};
document.getElementById('insertAI').onclick =
  () => document.getElementById('insertDirect').click();

// DOWNLOAD EXAM TEMPLATE AS PDF
document.getElementById('downloadExam').onclick = () => {
  const { jsPDF } = window.jspdf;
  const doc = new jsPDF();
  cont.querySelectorAll('.question-line').forEach((ln, i) => {
    const t = ln.querySelector('span:nth-child(2)').textContent || '';
    doc.text(`${i+1}. ${t}`, 10, 10 + i * 10);
  });
  doc.save('exam_template.pdf');
};

// STUB ANSWER KEY DOWNLOAD
document.getElementById('downloadKey').onclick =
  () => alert('Answer key download not implemented');


// HAMBURGER MENU TOGGLE
const ham  = document.getElementById('hamburgerBtn'),
      side = document.getElementById('sideMenu');
ham.addEventListener('click', () => side.classList.toggle('open'));
side.querySelectorAll('a').forEach(a =>
  a.addEventListener('click', () => side.classList.remove('open'))
);


// AUTH & MENU STATE
async function updateMenu(user) {
  document.getElementById('menuUser').textContent    = user ? user.name : 'Guest';
  document.getElementById('menuLogin') .classList.toggle('hidden', !!user);
  document.getElementById('menuSignup').classList.toggle('hidden', !!user);
```

```
  document.getElementById('menuLogout').classList.toggle('hidden', !user);
  document.getElementById('logoutLink').classList.toggle('hidden', !user);
  // only show personal & gradebook when logged in
  document.querySelectorAll('.menu-link').forEach(el =>
    el.classList.toggle('hidden', !user)
  );
}

async function checkAuth() {
  const res = await fetch('/auth-status'),
        { user } = await res.json();
  updateMenu(user);
}
window.addEventListener('load', checkAuth);


// LOGIN & SIGNUP HANDLERS
document.getElementById('menuLogin').onclick = e => {
  e.preventDefault();
  resetLoginForm();
  showView('login');
};
document.getElementById('toSignup').onclick = e => {
  e.preventDefault();
  resetLoginForm();
  showView('signup');
};
document.getElementById('menuSignup').onclick = e => {
  e.preventDefault();
  resetLoginForm();
  showView('signup');
};

document.getElementById('loginBtn').onclick = async () => {
  const id = document.getElementById('loginId').value.trim(),
        pw = document.getElementById('loginPassword').value;
  const res = await fetch('/login', {
    method: 'POST',
    headers: {'Content-Type':'application/json'},
    body: JSON.stringify({ id, password: pw })
  });
  if (res.ok) {
    await checkAuth();
    showView('home');
```

```
  } else {
    alert('Login failed: ' + (await res.json()).error);
  }
};

document.getElementById('signupBtn').onclick = async () => {
  const payload = {
    id:      document.getElementById('signupId').value.trim(),
    name:    document.getElementById('signupName').value.trim(),
    email:   document.getElementById('signupEmail').value.trim(),
    password:document.getElementById('signupPassword').value,
    confirm: document.getElementById('signupConfirm').value
  };
  const res = await fetch('/signup', {
    method: 'POST',
    headers:{'Content-Type':'application/json'},
    body: JSON.stringify(payload)
  });
  if (res.ok) {
    alert('Account created! Please log in.');
    resetLoginForm();
    showView('login');
  } else {
    alert('Sign up error: ' + (await res.json()).error);
  }
};

// LOGOUT HANDLER
document.getElementById('logoutLink').onclick =
document.getElementById('menuLogout').onclick = async () => {
  await fetch('/logout');
  await checkAuth();
  resetLoginForm();
  showView('login');
};


// PERSONAL INFO VIEW
document.querySelector('[data-view="personal"]').addEventListener('click', async () => {
  const res = await fetch('/auth-status'),
        { user } = await res.json();
  if (!user) return;
  document.getElementById('profileId').textContent           = user.id;
  document.getElementById('profileNameDisplay').textContent  = user.name;
```

```
  document.getElementById('profileEmailDisplay').textContent = user.email || '—';
});


// GRADEBOOK SETUP
const gradeList = new List('previous', {
  valueNames: ['date','exam','studentid','studentname','score']
});

async function loadGradebook() {
  const res = await fetch('/scores');
  if (!res.ok) return alert('Please log in first');
  const grades = await res.json();
  gradeList.clear();
  grades.forEach(g => gradeList.add({
    date:        g.date.split('T')[0],
    exam:        g.examName || 'Exam',
    studentid:   g.studentId,
    studentname: g.studentName,
    score:       g.score
  }));
}

// EXPORT TO EXCEL
document.getElementById('exportExcelBtn').onclick = () => {
  const data = gradeList.items.map(i => i.values());
  const ws   = XLSX.utils.json_to_sheet(data);
  const wb   = XLSX.utils.book_new();
  XLSX.utils.book_append_sheet(wb, ws, 'Gradebook');
  XLSX.writeFile(wb, 'gradebook.xlsx');
};

// Hook gradebook submenu
document.querySelectorAll('.menu-link').forEach(btn =>
  btn.addEventListener('click', () => {
    const v = btn.dataset.view;
    showView(v);
    if (v === 'previous') loadGradebook();
  })
);


// BATCH GRADING (AI)
document.getElementById('gradeBatchBtn').onclick = async () => {
```

```
// 1) gather inputs
const examName = document.getElementById('examCodeInput').value.trim();
const tpl      = document.getElementById('templateInput').files[0];
const key      = document.getElementById('answerKeyInput').files[0];
const subs     = Array.from(document.getElementById('studentInput').files);

// 2) validate
if (!examName) {
  return alert('Please enter the exam name/code.');
}
if (!tpl || !key || !subs.length) {
  return alert('Please upload template, answer key and student papers.');
}

// 3) show spinner
document.getElementById('spinner').classList.remove('hidden');

// 4) build FormData
const form = new FormData();
form.append('examName',   examName);
form.append('template',   tpl);
form.append('answerKey',  key);
subs.forEach(f => form.append('studentPapers', f));

try {
  // 5) send it—with credentials
  const res = await fetch('/grade-batch', {
    method:      'POST',
    credentials: 'same-origin',
    body:        form
  });

  // 6) hide spinner
  document.getElementById('spinner').classList.add('hidden');

  if (!res.ok) {
    const err = await res.json().catch(()=>({}));
    throw new Error(err.error || res.statusText);
  }

  // 7) render results
  const results = await res.json();
  const tbody   = document.querySelector('#batchResults tbody');
  tbody.innerHTML = results.map(r => `
```

```
      <tr>
        <td>${r.studentId}</td>
        <td>${r.studentName}</td>
        <td>${r.score}</td>
        <td><a href="${r.paperUrl}" download>Download</a></td>
      </tr>
    `).join('');
    document.getElementById('resultsSection').classList.remove('hidden');

  } catch (e) {
    console.error('Batch grading failed:', e);
    alert('Batch grading failed: ' + e.message);
    document.getElementById('spinner').classList.add('hidden');
  }
};, // server.js
const express    = require('express');
const session    = require('express-session');
const bodyParser = require('body-parser');
const bcrypt     = require('bcrypt');
const path       = require('path');
const fs         = require('fs');
const sqlite3    = require('sqlite3').verbose();
const multer     = require('multer');

const app = express();
const PORT = process.env.PORT || 3000;
const SALT_ROUNDS = 10;

// Ensure upload directories exist
['uploads/lectures','uploads/exams','uploads/submissions']
  .forEach(dir => fs.existsSync(dir) || fs.mkdirSync(dir, { recursive: true }));

// Multer storage
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    let dest = 'uploads/lectures';
    if (file.fieldname === 'examTemplate') dest = 'uploads/exams';
    if (file.fieldname === 'submission')   dest = 'uploads/submissions';
    cb(null, dest);
  },
  filename: (req, file, cb) =>
    cb(null, `${Date.now()}-${file.originalname}`)
});
const upload = multer({ storage });
```

```
// SQLite setup
const dbFile = path.join(__dirname, 'data.db');
const db = new sqlite3.Database(dbFile, err => {
  if (err) throw err;
  console.log('Connected to SQLite:', dbFile);
});
db.serialize(() => {
  db.run(`CREATE TABLE IF NOT EXISTS users (
    id TEXT PRIMARY KEY,
    name TEXT NOT NULL,
    passwordHash TEXT NOT NULL
  )`);
  db.run(`CREATE TABLE IF NOT EXISTS grades (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    graderId TEXT NOT NULL,
    examName TEXT NOT NULL,
    studentId TEXT NOT NULL,
    studentName TEXT NOT NULL,
    score REAL NOT NULL,
    date TEXT NOT NULL,
    FOREIGN KEY(graderId) REFERENCES users(id)
  )`);
  db.run(`CREATE TABLE IF NOT EXISTS lectures (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    uploaderId TEXT NOT NULL,
    filename TEXT NOT NULL,
    path TEXT NOT NULL,
    uploadDate TEXT NOT NULL,
    FOREIGN KEY(uploaderId) REFERENCES users(id)
  )`);
  db.run(`CREATE TABLE IF NOT EXISTS exams (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    creatorId TEXT NOT NULL,
    filename TEXT NOT NULL,
    path TEXT NOT NULL,
    createDate TEXT NOT NULL,
    FOREIGN KEY(creatorId) REFERENCES users(id)
  )`);
  db.run(`CREATE TABLE IF NOT EXISTS submissions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    examId INTEGER NOT NULL,
    studentId TEXT NOT NULL,
    studentName TEXT NOT NULL,
```

```
    filename TEXT NOT NULL,
    path TEXT NOT NULL,
    submitDate TEXT NOT NULL,
    FOREIGN KEY(examId) REFERENCES exams(id)
  )`);
});

// Middleware
app.use(bodyParser.json());
app.use(session({
  secret: 'replace_with_a_secure_secret',
  resave: false,
  saveUninitialized: true,
}));

// Serve uploads
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));

// Serve frontend from "public/"
const FRONTEND = path.join(__dirname, '..', 'public');
app.use(express.static(FRONTEND));

// Auth guard
function requireLogin(req, res, next) {
  if (!req.session.user) return res.status(401).json({ error: 'Not logged in' });
  next();
}

// --- Sign up ---
app.post('/signup', (req, res) => {
  const { id, name, password, confirm } = req.body;
  if (!id||!name||!password||!confirm)
    return res.status(400).json({ error:'All fields required' });
  if (password !== confirm)
    return res.status(400).json({ error:'Passwords must match' });

  db.get(`SELECT id FROM users WHERE id=?`, [id], (err,row) => {
    if (err) return res.status(500).json({ error:err.message });
    if (row) return res.status(400).json({ error:'ID taken' });
    bcrypt.hash(password, SALT_ROUNDS, (err,hash) => {
      if (err) return res.status(500).json({ error:err.message });
      db.run(
        `INSERT INTO users (id,name,passwordHash) VALUES (?,?,?)`,
        [id,name,hash],
```

```
        err => err
          ? res.status(500).json({ error:err.message })
          : res.json({ ok:true })
      );
    });
  });
});

// --- Log in ---
app.post('/login', (req, res) => {
  const { id, password } = req.body;
  if (!id||!password) return res.status(400).json({ error:'ID+password required' });

  db.get(`SELECT id,name,passwordHash FROM users WHERE id=?`, [id], (err,user) => {
    if (err) return res.status(500).json({ error:err.message });
    if (!user) return res.status(401).json({ error:'Invalid credentials' });
    bcrypt.compare(password, user.passwordHash, (err,match) => {
      if (err) return res.status(500).json({ error:err.message });
      if (!match) return res.status(401).json({ error:'Invalid credentials' });
      req.session.user = { id:user.id, name:user.name };
      res.json({ ok:true, user:req.session.user });
    });
  });
});

// --- Log out & auth status ---
app.get('/logout', (req, res) => {
  req.session.destroy(err =>
    err
      ? res.status(500).json({ error:'Logout failed' })
      : res.json({ ok:true })
  );
});
app.get('/auth-status', (req, res) =>
  res.json({ user: req.session.user || null })
);

// --- Lecture uploads ---
app.post(
  '/upload-lectures',
  requireLogin,
  upload.array('lectureFiles',10),
  (req,res) => {
    const now = new Date().toISOString();
```

```
    const stmt = db.prepare(`
      INSERT INTO lectures (uploaderId,filename,path,uploadDate)
      VALUES (?,?,?,?)
    `);
    req.files.forEach(f =>
      stmt.run(req.session.user.id, f.originalname, f.path, now)
    );
    stmt.finalize(err =>
      err
        ? res.status(500).json({ error:err.message })
        : res.json({ ok:true })
    );
  }
);
app.get('/lectures', (req,res) =>
  db.all(`SELECT * FROM lectures`, [], (err,rows) =>
    err ? res.status(500).json({ error:err.message }) : res.json(rows)
  )
);

// --- Exam templates ---
app.post(
  '/upload-exam',
  requireLogin,
  upload.single('examTemplate'),
  (req,res) => {
    const now = new Date().toISOString();
    db.run(
      `INSERT INTO exams (creatorId,filename,path,createDate) VALUES (?,?,?,?)`,
      [req.session.user.id, req.file.originalname, req.file.path, now],
      err => err
        ? res.status(500).json({ error:err.message })
        : res.json({ ok:true })
    );
  }
);
app.get('/exams', (req,res) =>
  db.all(`SELECT * FROM exams`, [], (err,rows) =>
    err ? res.status(500).json({ error:err.message }) : res.json(rows)
  )
);

// --- Submissions & manual grading ---
app.post(
```

```
  '/upload-submission',
  requireLogin,
  upload.single('submission'),
  (req,res) => {
    const { examId, studentId, studentName } = req.body;
    const now = new Date().toISOString();
    db.run(
      `INSERT INTO submissions
         (examId,studentId,studentName,filename,path,submitDate)
       VALUES (?,?,?,?,?,?)`,
      [examId, studentId, studentName,
       req.file.originalname, req.file.path, now],
      err => err
        ? res.status(500).json({ error:err.message })
        : res.json({ ok:true })
    );
  }
);
app.get('/submissions/:examId', (req,res) =>
  db.all(`SELECT * FROM submissions WHERE examId = ?`,
         [req.params.examId],
         (err,rows) => err
           ? res.status(500).json({ error:err.message })
           : res.json(rows)
  )
);

// --- Single-grade submission endpoint ---
app.post('/submit-score', requireLogin, (req,res) => {
  const { examName, studentId, studentName, score } = req.body;
  if (!examName||!studentId||!studentName||score==null)
    return res.status(400).json({ error:'Missing fields' });
  const now = new Date().toISOString();
  db.run(
    `INSERT INTO grades
       (graderId,examName,studentId,studentName,score,date)
     VALUES (?,?,?,?,?,?)`,
    [req.session.user.id, examName, studentId, studentName, score, now],
    err => err
      ? res.status(500).json({ error:err.message })
      : res.json({ ok:true })
  );
});
app.get('/scores', requireLogin, (req,res) =>
```

```
  db.all(
    `SELECT * FROM grades WHERE graderId = ? ORDER BY date DESC`,
    [req.session.user.id],
    (err,rows) => err
      ? res.status(500).json({ error:err.message })
      : res.json(rows)
  )
);

// ——————————————————————————————————————
// Batch AI grading: forward to Python service
// ——————————————————————————————————————
app.post(
  '/grade-batch',
  requireLogin,
  upload.fields([
    { name: 'template',      maxCount: 1 },
    { name: 'answerKey',     maxCount: 1 },
    { name: 'studentPapers', maxCount: 100 }
  ]),
  async (req, res) => {
    try {
      // 1) Gather inputs
      const { examName }      = req.body;
      const now               = new Date().toISOString();
      const templatePath      = req.files.template[0].path;
      const answerKeyPath     = req.files.answerKey[0].path;
      const studentPaths      = (req.files.studentPapers || []).map(f => f.path);

      // 2) Call Python grading API on port 7860
      const aiRes = await fetch('http://localhost:7860/grade', {
        method:  'POST',
        headers: { 'Content-Type': 'application/json' },
        body:    JSON.stringify({ examName, templatePath, answerKeyPath, studentPaths })
      });
      if (!aiRes.ok) {
        const errText = await aiRes.text().catch(() => aiRes.statusText);
        throw new Error(err || aiRes.statusText);
      }
      const results = await aiRes.json();

      // 3) Persist those grades in SQLite
      const stmt = db.prepare(`
        INSERT INTO grades
```

```
        (graderId, examName, studentId, studentName, score, date)
       VALUES (?,?,?,?,?,?)
      `);
      results.forEach(r => {
       stmt.run(
        req.session.user.id,
        examName,
        r.studentId,
        r.studentName,
        r.score,
        now
       );
      });
      stmt.finalize();

      // 4) Return AI results to front-end
      return res.json(results);

    } catch (err) {
      console.error('AI grading error:', err);
      return res.status(500).json({ error: err.message });
    }
  }
);

// ——————————————————————————————————————
// Fallback: serve index.html
// ——————————————————————————————————————
app.use((req, res) => {
  res.sendFile(path.join(FRONTEND, 'index.html'));
});

// ——————————————————————————————————————
// Start server
// ——————————————————————————————————————
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
 {
  "name": "daic-database",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "init-db": "node init-db.js",
```

```
    "start": "node server.js"
  },
  "dependencies": {
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.3",
    "express": "^4.21.2",
    "express-session": "^1.18.1",
    "multer": "^1.4.4",
    "node-fetch": "^3.3.2",
    "sqlite3": "^5.1.7"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
} at least let us just return these on the pics when it is missing file even if it is not actually getting graded because we are missing ai part ('Please enter the exam name/code.'/     'Please upload template, answer key and student papers.');
)
```