/\*\*

 \* Google Apps Script backend for Exam Attendance System

 \* StudentSeating columns (0-based):

 \* Date(0), Department(1), Batch(2), Hall No(3), Student Roll No(4), Name(5),

 \* Seating No(6), Subject Code(7), Subject(8), Year(9), Section(10),

 \* Status(11), Reason(12)

 \*/

const USERS\_SHEET = "Users";

const SEATING\_SHEET = "StudentSeating";

const LOCK\_SHEET = "Settings";

function doPost(e) {

  try {

    const req = JSON.parse(e.postData.contents);

    const action = req.action;

    if (action === "login") return jsonResponse(handleLogin(req));

    if (action === "uploadSeating") return jsonResponse(handleUploadSeating(req));

    if (action === "getHalls") return jsonResponse(handleGetHalls(req));

    if (action === "getHall") return jsonResponse(handleGetHall(req));

    if (action === "saveAttendance") return jsonResponse(handleSaveAttendance(req));

    if (action === "getAbsentees") return jsonResponse(handleGetAbsentees(req));

    if (action === "saveReason") return jsonResponse(handleSaveReason(req));

    if (action === "report") return jsonResponse(handleReport(req));

    if (action === "getLockStatus") return jsonResponse({ success: true, lock: getLockStatus(req.date) });

    if (action === "setLockStatus") return jsonResponse(setLockStatus(req.date, req.value));

    if (action === 'checkLock') return jsonResponse({ success: true, locked: isDateLocked(req.date) });

    if (action === "getStudentSeating") return jsonResponse(handleGetStudentSeating(req));

    if (action === "uploadUsers") return jsonResponse(handleUploadUsers(req.users));

    if (action === "uploadUsersOverwrite") return jsonResponse(handleUploadUsersOverwrite(req.users));

    return jsonResponse({ success: false, message: "Unknown action" });

  } catch (err) {

    return jsonResponse({ success: false, message: err && err.message ? err.message : String(err) });

  }

}

function jsonResponse(obj) {

  return ContentService.createTextOutput(JSON.stringify(obj)).setMimeType(ContentService.MimeType.JSON);

}

// ---------------- Helpers ----------------

function parseSheetDateCell(cell) {

  const tz = Session.getScriptTimeZone();

  if (cell === "" || cell === null || cell === undefined) return "";

  if (Object.prototype.toString.call(cell) === "[object Date]" && !isNaN(cell.getTime())) {

    return Utilities.formatDate(cell, tz, "yyyy-MM-dd");

  }

  if (typeof cell === "number") {

    const jsDate = new Date(Math.round((cell - 25569) \* 86400 \* 1000));

    return Utilities.formatDate(jsDate, tz, "yyyy-MM-dd");

  }

  const s = String(cell).trim();

  if (/^\d{4}-\d{2}-\d{2}$/.test(s)) return s;

  const d = new Date(s);

  if (!isNaN(d.getTime())) return Utilities.formatDate(d, tz, "yyyy-MM-dd");

  return s;

}

function normalizeStatus(raw) {

  if (raw === null || raw === undefined) return "";

  let s = String(raw).trim();

  if (!s) return "";

  const up = s.toUpperCase();

  if (up === "OD" || up === "O D" || up === "ON DUTY" || up === "ON-DUTY") return "OD";

  if (up === "A" || up === "ABS" || up === "ABSENT") return "Absent";

  if (up === "P" || up === "PRESENT") return "Present";

  return s;

}

function normalizeRowToObject(r) {

  return {

    Date: parseSheetDateCell(r[0]),

    Department: r[1],

    Batch: r[2],

    HallNo: r[3],

    RollNo: r[4],

    Name: r[5],

    SeatNo: r[6],

    SubjectCode: r[7],

    Subject: r[8],

    Year: r[9],

    Section: r[10],

    Status: normalizeStatus(r[11]),

    Reason: r[12]

  };

}

// ---------------- Login ----------------

function handleLogin(req) {

  const { email, password } = req;

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(USERS\_SHEET);

  if (!sheet) return { success: false, message: "Users sheet not found" };

  const data = sheet.getDataRange().getValues();

  if (!data || data.length < 2) return { success: false, message: "No users" };

  data.shift();

  for (let row of data) {

    if (row[0] === email && row[1] === password) {

      return { success: true, user: { email: row[0], name: row[2], role: row[3], department: row[4] } };

    }

  }

  return { success: false, message: "Invalid credentials" };

}

// ---------------- Upload Seating ----------------

function handleUploadSeating(req) {

  const rows = req.rows;

  if (!rows || !Array.isArray(rows) || rows.length === 0) return { success: false, message: "No rows received" };

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const sheet = ss.getSheetByName(SEATING\_SHEET) || ss.insertSheet(SEATING\_SHEET);

  if (sheet.getLastRow() === 0) {

    sheet.appendRow([

      "Date","Department","Batch","Hall No","Student Roll No","Name",

      "Seating No","Subject Code","Subject","Year","Section","Status","Reason"

    ]);

  }

  const toAppend = [];

  rows.forEach(r => {

    let dateCell = r.Date;

    if (typeof dateCell === "string" && /^\d{4}-\d{2}-\d{2}$/.test(dateCell)) {

      dateCell = new Date(dateCell + "T00:00:00");

    } else if (typeof dateCell === "number") {

      dateCell = new Date(Math.round((dateCell - 25569) \* 86400 \* 1000));

    }

    toAppend.push([

      dateCell || "",

      r.Department || "",

      r.Batch || "", // new field FN/AN

      r["Hall No"] || r.HallNo || "",

      r["Student Roll No"] || r.RollNo || "",

      r["Name"] || r.Name || "",

      r["Seating No"] || r.SeatNo || "",

      r["Subject Code"] || r.SubjectCode || "",

      r.Subject || "",

      r.Year || "",

      r.Section || "",

      "Present",

      ""

    ]);

  });

  sheet.getRange(sheet.getLastRow() + 1, 1, toAppend.length, toAppend[0].length).setValues(toAppend);

  return { success: true, message: "Uploaded successfully", count: toAppend.length };

}

// ---------------- Get All Halls for a Department ----------------

function handleGetHalls(req) {

  const { department } = req;

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(SEATING\_SHEET);

  if (!sheet) return { success: false, message: "Seating sheet not found" };

  const data = sheet.getDataRange().getValues();

  if (!data || data.length < 2) return { success: true, halls: [] };

  data.shift();

  const halls = Array.from(

    new Set(

      data

        .map(r => normalizeRowToObject(r))

        .filter(r => r.Department === department && r.HallNo)

        .map(r => r.HallNo)

    )

  ).sort();

  return { success: true, halls };

}

// ---------------- Get Hall ----------------

function handleGetHall(req) {

  const { hallNo, department, date, batch } = req;

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(SEATING\_SHEET);

  if (!sheet) return { success: false, message: "Seating sheet not found" };

  const data = sheet.getDataRange().getValues();

  if (!data || data.length < 2) return { success: true, hallData: [] };

  data.shift();

  const targetDate = date ? String(date).trim() : "";

  let rows = data

    .map(r => normalizeRowToObject(r))

    .filter(r => r.Department === department && String(r.HallNo) == String(hallNo));

  if (targetDate) rows = rows.filter(r => r.Date === targetDate);

  if (batch) rows = rows.filter(r => r.Batch === batch);

  return { success: true, hallData: rows };

}

// ---------------- Save Attendance ----------------

function handleSaveAttendance(req) {

  const { hallNo, date, updates, batch } = req;

  if (!Array.isArray(updates)) return { success: false, message: "Invalid updates" };

  if (isDateLocked(date)) {

    return { success: false, message: "Editing locked by CoE for this date." };

  }

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(SEATING\_SHEET);

  if (!sheet) return { success: false, message: "Seating sheet not found" };

  const values = sheet.getDataRange().getValues();

  if (!values || values.length < 2) return { success: false, message: "No data" };

  const headers = values.shift();

  const targetDate = date ? String(date).trim() : "";

  let updated = 0;

  for (let i = 0; i < values.length; i++) {

    const r = values[i];

    const rowDate = parseSheetDateCell(r[0]);

    const rowHall = r[3];

    const rowRoll = r[4];

    const u = updates.find(x => String(x.RollNo) == String(rowRoll));

    if (u && String(rowHall) == String(hallNo) && (!targetDate || rowDate === targetDate)) {

      sheet.getRange(i + 2, 12).setValue(u.Status); // column 12 = Status

      updated++;

    }

  }

  return { success: true, message: "Attendance updated", updatedCount: updated };

}

// ---------------- Get Absentees ----------------

function handleGetAbsentees(req) {

  const { department, date, batch } = req;

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(SEATING\_SHEET);

  if (!sheet) return { success: false, message: "Seating sheet not found" };

  const data = sheet.getDataRange().getValues();

  if (!data || data.length < 2) return { success: true, absentees: [], totals: {}, yearSummary: {} };

  data.shift();

  const targetDate = date ? String(date).trim() : "";

  const deptRows = data

    .map(r => normalizeRowToObject(r))

    .filter(r =>

      r.Department === department &&

      (!targetDate || r.Date === targetDate) &&

      (!batch || r.Batch === batch)

    );

  const absentees = deptRows.filter(r => {

    const st = (r.Status || "").trim().toUpperCase();

    return st === "ABSENT" || st === "OD";

  });

  const total = deptRows.length;

  const present = deptRows.filter(r => (r.Status || "").toUpperCase() === "PRESENT").length;

  const absent = deptRows.filter(r => (r.Status || "").toUpperCase() === "ABSENT").length;

  const od = deptRows.filter(r => (r.Status || "").toUpperCase() === "OD").length;

  const attendancePercent = total === 0 ? 0 : Math.round((present / total) \* 10000) / 100;

  const yearSummary = {};

  deptRows.forEach(r => {

    const yr = r.Year || "NA";

    const sec = r.Section || "NA";

    if (!yearSummary[yr]) yearSummary[yr] = { sections: {} };

    if (!yearSummary[yr].sections[sec]) yearSummary[yr].sections[sec] = { total: 0, present: 0, absent: 0, od: 0 };

    const s = yearSummary[yr].sections[sec];

    s.total++;

    if ((r.Status || "").toUpperCase() === "PRESENT") s.present++;

    if ((r.Status || "").toUpperCase() === "ABSENT") s.absent++;

    if ((r.Status || "").toUpperCase() === "OD") s.od++;

  });

  return { success: true, absentees, totals: { total, present, absent, od, attendancePercent }, yearSummary };

}

// ---------------- Save Reason ----------------

function handleSaveReason(req) {

  const { rollNo, reason, date } = req;

  if (!rollNo) return { success: false, message: "Missing rollNo" };

  if (isDateLocked(date)) {

    return { success: false, message: "Editing not allowed. This date is locked by CoE." };

  }

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(SEATING\_SHEET);

  if (!sheet) return { success: false, message: "Seating sheet not found" };

  const data = sheet.getDataRange().getValues();

  if (!data || data.length < 2) return { success: false, message: "No data" };

  data.shift();

  const targetDate = date ? String(date).trim() : "";

  let saved = 0;

  for (let i = 0; i < data.length; i++) {

    const row = data[i];

    const rowDate = parseSheetDateCell(row[0]);

    const rowRoll = String(row[4]).trim();

    const rowStatus = normalizeStatus(row[11]);

    if (

      rowRoll === String(rollNo).trim() &&

      (rowStatus === "Absent" || rowStatus === "OD") &&

      (!targetDate || rowDate === targetDate)

    ) {

      sheet.getRange(i + 2, 13).setValue(reason || ""); // col 13 = Reason

      saved++;

    }

  }

  return { success: true, saved };

}

// ---------------- Report Summary for CoE ----------------

function handleReport(req) {

  const { date, department, batch } = req;

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(SEATING\_SHEET);

  if (!sheet) return { success: false, message: "Seating sheet not found" };

  const data = sheet.getDataRange().getValues();

  if (!data || data.length < 2) return { success: true, rows: [], deptSummary: {} };

  data.shift(); // Remove header row

  // Convert each row to object

  const allRows = data.map(r => normalizeRowToObject(r));

  const targetDate = date ? String(date).trim() : "";

  const targetBatch = batch ? String(batch).trim() : "";

  // Filter rows by date and batch if provided

  let rows = allRows.filter(r =>

    (!targetDate || r.Date === targetDate) &&

    (!targetBatch || (r.Batch || "").toUpperCase() === targetBatch.toUpperCase())

  );

  // Filter by department if provided

  if (department) rows = rows.filter(r => r.Department === department);

  // Generate department summary

  const deptSummary = {};

  rows.forEach(r => {

    const dept = r.Department || "NA";

    const year = r.Year || "NA";

    const sec = r.Section || "NA";

    const st = normalizeStatus(r.Status);

    if (!deptSummary[dept]) deptSummary[dept] = { total: 0, present: 0, absent: 0, od: 0, years: {} };

    const D = deptSummary[dept];

    D.total++;

    if (st === "Present") D.present++;

    if (st === "Absent") D.absent++;

    if (st === "OD") D.od++;

    if (!D.years[year]) D.years[year] = { total: 0, present: 0, absent: 0, od: 0, sections: {} };

    const Y = D.years[year];

    Y.total++;

    if (st === "Present") Y.present++;

    if (st === "Absent") Y.absent++;

    if (st === "OD") Y.od++;

    if (!Y.sections[sec]) Y.sections[sec] = { total: 0, present: 0, absent: 0, od: 0 };

    const S = Y.sections[sec];

    S.total++;

    if (st === "Present") S.present++;

    if (st === "Absent") S.absent++;

    if (st === "OD") S.od++;

  });

  // Return both all rows and summary

  return { success: true, rows: allRows, deptSummary };

}

// ---------------- DATE-WISE LOCK CONTROL ----------------

// ---------------- LOCK SYSTEM ----------------

function getLockStatus(date) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  let sheet = ss.getSheetByName(LOCK\_SHEET);

  if (!sheet) {

    sheet = ss.insertSheet(LOCK\_SHEET);

    sheet.appendRow(["Date", "Status"]);

  }

  const data = sheet.getDataRange().getValues();

  const formattedDate = Utilities.formatDate(new Date(date), Session.getScriptTimeZone(), "yyyy-MM-dd");

  for (let i = 1; i < data.length; i++) {

    const d = Utilities.formatDate(new Date(data[i][0]), Session.getScriptTimeZone(), "yyyy-MM-dd");

    if (d === formattedDate) return data[i][1] === "Locked" ? "Locked" : "Unlocked";

  }

  return "Unlocked";

}

function setLockStatus(date, value) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  let sheet = ss.getSheetByName(LOCK\_SHEET);

  if (!sheet) {

    sheet = ss.insertSheet(LOCK\_SHEET);

    sheet.appendRow(["Date", "Status"]);

  }

  const data = sheet.getDataRange().getValues();

  const formattedDate = Utilities.formatDate(new Date(date), Session.getScriptTimeZone(), "yyyy-MM-dd");

  for (let i = 1; i < data.length; i++) {

    const d = Utilities.formatDate(new Date(data[i][0]), Session.getScriptTimeZone(), "yyyy-MM-dd");

    if (d === formattedDate) {

      sheet.getRange(i + 1, 2).setValue(value);

      return { success: true, lock: value };

    }

  }

  sheet.appendRow([formattedDate, value]);

  return { success: true, lock: value };

}

function isDateLocked(date) {

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(LOCK\_SHEET);

  if (!sheet) return false;

  const data = sheet.getDataRange().getValues();

  const formattedDate = Utilities.formatDate(new Date(date), Session.getScriptTimeZone(), "yyyy-MM-dd");

  for (let i = 1; i < data.length; i++) {

    const d = Utilities.formatDate(new Date(data[i][0]), Session.getScriptTimeZone(), "yyyy-MM-dd");

    if (d === formattedDate && String(data[i][1]).toLowerCase() === "locked") return true;

  }

  return false;

}

// ---------------- CoE Report ----------------

function handleReport(req) {

  const { department, hallNo, date } = req;

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(SEATING\_SHEET);

  if (!sheet) return { success: false, message: "Seating sheet not found" };

  const data = sheet.getDataRange().getValues();

  if (!data || data.length < 2) return { success: true, total: 0, present: 0, absent: 0, od: 0, rows: [], deptSummary: {} };

  data.shift(); // remove headers

  let rows = data.map(r => normalizeRowToObject(r));

  if (department) rows = rows.filter(r => r.Department === department);

  if (hallNo) rows = rows.filter(r => String(r.HallNo) == String(hallNo));

  if (date) rows = rows.filter(r => r.Date === String(date));

  const total = rows.length;

  const present = rows.filter(r => r.Status === "Present").length;

  const absent = rows.filter(r => r.Status === "Absent").length;

  const od = rows.filter(r => r.Status === "OD").length;

  const attendancePercent = total === 0 ? 0 : Math.round((present / total) \* 10000) / 100;

  // build dept->year->section map

  const deptMap = {};

  rows.forEach(r => {

    const dept = r.Department || "Unknown";

    const yr = r.Year || "Unknown";

    const sec = r.Section || "Unknown";

    if (!deptMap[dept]) deptMap[dept] = { total: 0, present: 0, absent: 0, od: 0, years: {} };

    const D = deptMap[dept];

    D.total++; if (r.Status === "Present") D.present++; if (r.Status === "Absent") D.absent++; if (r.Status === "OD") D.od++;

    if (!D.years[yr]) D.years[yr] = { total: 0, present: 0, absent: 0, od: 0, sections: {} };

    const Y = D.years[yr];

    Y.total++; if (r.Status === "Present") Y.present++; if (r.Status === "Absent") Y.absent++; if (r.Status === "OD") Y.od++;

    if (!Y.sections[sec]) Y.sections[sec] = { total: 0, present: 0, absent: 0, od: 0 };

    const S = Y.sections[sec];

    S.total++; if (r.Status === "Present") S.present++; if (r.Status === "Absent") S.absent++; if (r.Status === "OD") S.od++;

  });

  return {

    success: true,

    total, present, absent, od, attendancePercent,

    rows,

    deptSummary: deptMap

  };

}

// ---------------- Get Student Seating (student view) ----------------

function handleGetStudentSeating(req) {

  const { email, date } = req;

  const userSheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(USERS\_SHEET);

  if (!userSheet) return { success: false, message: "Users sheet not found" };

  const users = userSheet.getDataRange().getValues();

  if (!users || users.length < 2) return { success: false, message: "No users found" };

  users.shift(); // remove header

  const user = users.find(u => u[0] === email && u[3] === "Student");

  if (!user) return { success: false, message: "Student not found" };

  const studentRoll = user[5];

  const dept = user[4];

  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(SEATING\_SHEET);

  if (!sheet) return { success: false, message: "Seating sheet not found" };

  const data = sheet.getDataRange().getValues();

  if (!data || data.length < 2) return { success: true, seating: [] };

  data.shift();

  const today = date ? String(date) : Utilities.formatDate(new Date(), Session.getScriptTimeZone(), "yyyy-MM-dd");

  const now = new Date();

  const batchAccessTime = { "FN": new Date(), "AN": new Date() };

  batchAccessTime["FN"].setHours(8, 30, 0, 0);  // 8:30 AM

  batchAccessTime["AN"].setHours(13, 0, 0, 0);   // 1:00 PM

  const rows = data

    .map(r => normalizeRowToObject(r))

    .filter(r => r.RollNo === studentRoll && r.Department === dept && r.Date === today)

    .map(r => {

      const batch = r.Batch.toUpperCase();

      const canView = now >= (batchAccessTime[batch] || now); // Only show after batch time

      return canView ? r : { ...r, HallNo: "Locked", SeatNo: "Locked" };

    });

  return { success: true, seating: rows };

}

// ---------------- Upload Users ----------------

function handleUploadUsers(users) {

  try {

    if (!Array.isArray(users) || users.length === 0)

      return { success: false, message: "No users provided" };

    const ss = SpreadsheetApp.getActiveSpreadsheet();

    const sheet = ss.getSheetByName(USERS\_SHEET) || ss.insertSheet(USERS\_SHEET);

    // Add headers if sheet is empty

    if (sheet.getLastRow() === 0) {

      sheet.appendRow(["email", "password", "Name", "Role", "Department", "Student Roll No"]);

    }

    const existingData = sheet.getDataRange().getValues();

    existingData.shift(); // remove header

    const existingEmails = existingData.map(r => (r[0] || "").toLowerCase());

    const toAdd = [];

    users.forEach(u => {

      const email = (u.email || "").trim();

      const password = (u.password || "").trim();

      const name = (u.name || "").trim();

      const role = (u.role || "").trim();

      const dept = (u.department || "").trim();

      const roll = (u.studentRollNo || "").trim();

      if (!email || existingEmails.includes(email.toLowerCase())) return;

      toAdd.push([email, password, name, role, dept, roll]);

      existingEmails.push(email.toLowerCase());

    });

    if (toAdd.length > 0) {

      sheet.getRange(sheet.getLastRow() + 1, 1, toAdd.length, toAdd[0].length).setValues(toAdd);

    }

    return { success: true, count: toAdd.length };

  } catch (err) {

    return { success: false, message: err.message };

  }

}

function handleUploadUsersOverwrite(users) {

  try {

    if (!Array.isArray(users) || users.length === 0)

      return { success: false, message: "No users provided" };

    const ss = SpreadsheetApp.getActiveSpreadsheet();

    const sheet = ss.getSheetByName(USERS\_SHEET) || ss.insertSheet(USERS\_SHEET);

    // Ensure headers exist

    if (sheet.getLastRow() === 0) {

      sheet.appendRow(["email", "password", "Name", "Role", "Department", "Student Roll No"]);

    }

    const data = sheet.getDataRange().getValues();

    const headers = data.shift();

    const emailCol = headers.indexOf("email");

    const passCol = headers.indexOf("password");

    const nameCol = headers.indexOf("Name");

    const roleCol = headers.indexOf("Role");

    const deptCol = headers.indexOf("Department");

    const rollCol = headers.indexOf("Student Roll No");

    let updated = 0, added = 0;

    users.forEach(u => {

      const email = (u.email || "").trim().toLowerCase();

      if (!email) return;

      const rowIndex = data.findIndex(r => (r[emailCol] || "").toLowerCase() === email);

      const newRow = [

        u.email || "",

        u.password || "",

        u.name || "",

        u.role || "",

        u.department || "",

        u.studentRollNo || ""

      ];

      if (rowIndex !== -1) {

        // Overwrite existing row

        sheet.getRange(rowIndex + 2, 1, 1, newRow.length).setValues([newRow]);

        updated++;

      } else {

        // Add new user

        sheet.appendRow(newRow);

        added++;

      }

    });

    return { success: true, count: updated + added, updated, added };

  } catch (err) {

    return { success: false, message: err.message };

  }

}