/\*\*

 \* Google Apps Script backend for Attendance System

 \* Sheets: Users, Students, Attendance

 \*/

const USERS\_SHEET = "Users";

const STUDENTS\_SHEET = "Students";

const ATTENDANCE\_SHEET = "Absentees\_OD";

/\* === Main Handler === \*/

function doPost(e) {

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

  const action = data.action;

  let result = { success: false, message: "Unknown action" };

  try {

    switch (action) {

      case "login":

        result = handleLogin(data.email, data.password);

        break;

      case "getStudents":

        result = getStudents(data.department, data.year, data.section, data.date, data.batch);

        break;

      case "saveAttendance":

        result = saveAttendance(data.date, data.batch, data.updates);

        break;

      case "getAbsenteesForFaculty":

        result = getAbsenteesForFaculty(data.email, data.date, data.batch);

        break;

      case "saveFacultyUpdates":

        result = saveFacultyUpdates(data.date, data.batch, data.updates);

        break;

      case "getUsersByDept":

    result = getUsersByDept(data.department);

    break;

  case "uploadUsers":

    result = uploadUsers(data.users);

    break;

  case "deleteUser":

    result = deleteUser(data.email);

    break;

  case "uploadStudents":

    result = uploadStudents(data.students);

    break;

  case "getStudentHistory":

    result = getStudentHistory(data.rollNo);

    break;

    case "getCollegeAttendanceSummary":

      result =  getCollegeAttendanceSummary(data.date, data.batch);

      break;

    case "getDeptClassSummary":

      result =  getDeptClassSummary(data.date, data.batch, data.department);

      break;

    case "getUsersByRole":

      result = getUsersByRole(data.role);

      break;

    case "getCollegeAbsenteesForOffice":

      result = getCollegeAbsenteesForOffice(data.date, data.batch);

      break;

    case "saveAttendanceStatus":

  result = saveAttendanceStatus(data.date, data.batch, data.year, data.section, data.department, data.status);

    break;

    case "getAttendanceStatusSummary":

  result = getAttendanceStatusSummary(data);

    break;

    case "lockAttendance":

  result = lockAttendance(data.date, data.batch, data.email, data.role);

  break;

case "unlockAttendance":

  result = unlockAttendance(data.date, data.batch, data.email, data.role);

  break;

    case "checkLockStatus":

  result =  checkLockStatus(data.date, data.batch);

    break;

}

  } catch (err) {

    result = { success: false, message: err.message };

  }

  return ContentService.createTextOutput(JSON.stringify(result))

    .setMimeType(ContentService.MimeType.JSON);

}

/\* === LOGIN === \*/

function handleLogin(email, password) {

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

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

  const headers = rows.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");

  for (let row of rows) {

    if (row[emailCol] == email && row[passCol] == password) {

      return {

        success: true,

        name: row[nameCol],

        role: row[roleCol],

        department: row[deptCol],

        email

      };

    }

  }

  return { success: false, message: "Invalid Email or Password" };

}

/\*\*

 \* Robust getStudents() - maps headers case-insensitively and applies saved attendance

 \* Expects constant STUDENTS\_SHEET and ATTENDANCE\_SHEET to be defined.

 \*/

function getStudents(department, year, section, date, batch) {

  try {

    const ss = SpreadsheetApp.getActiveSpreadsheet();

    const sheet = ss.getSheetByName(STUDENTS\_SHEET);

    if (!sheet) return { success: false, message: `Sheet "${STUDENTS\_SHEET}" not found.` };

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

    if (!all || all.length <= 1) return { success: true, students: [] };

    const header = all[0].map(h => String(h || "").trim());

    // build header map: normalized -> index

    const hmap = {};

    header.forEach((h, i) => { hmap[h.toLowerCase().replace(/\s+/g,'')] = i; });

    const rollKey = hmap["rollno"] !== undefined ? "rollno" : (hmap["rollno"]===undefined && Object.keys(hmap).find(k=>k.includes("roll")) || "rollno");

    const nameKey = Object.keys(hmap).find(k => k.includes("name")) || "name";

    const parentNameKey = Object.keys(hmap).find(k => k.includes("parent")) || "parentname";

    const parentPhoneKey = Object.keys(hmap).find(k => k.includes("phone")) || "parentphone";

    const deptKey = Object.keys(hmap).find(k => k.includes("department")) || "department";

    const yearKey = Object.keys(hmap).find(k => k.includes("year")) || "year";

    const sectionKey = Object.keys(hmap).find(k => k.includes("section")) || "section";

    // build base student list filtered by dept/year/section

    const rows = all.slice(1);

    const results = rows

      .filter(r => {

        const deptVal = String(r[hmap[deptKey]] || "").trim();

        const yearVal = String(r[hmap[yearKey]] || "").trim();

        const secVal = String(r[hmap[sectionKey]] || "").trim();

        return deptVal === department && yearVal === year && secVal === section;

      })

      .map(r => ({

        RollNo: String(r[hmap[rollKey]] || "").trim(),

        Name: String(r[hmap[nameKey]] || "").trim(),

        ParentName: String(r[hmap[parentNameKey]] || "").trim(),

        ParentPhone: String(r[hmap[parentPhoneKey]] || "").trim(),

        Department: department,

        Year: year,

        Section: section,

        Status: "Present",

        Reason: ""

      }));

    // if attendance sheet exists, apply saved statuses for the same date+batch

    const attSheet = ss.getSheetByName(ATTENDANCE\_SHEET);

    if (attSheet && results.length) {

      const attAll = attSheet.getDataRange().getValues();

      if (attAll && attAll.length > 1) {

        const attHeader = attAll[0].map(h => String(h || "").trim());

        const ahmap = {};

        attHeader.forEach((h,i) => { ahmap[h.toLowerCase().replace(/\s+/g,'')] = i; });

        const aDateKey = ahmap["date"] !== undefined ? "date" : Object.keys(ahmap).find(k=>k.includes("date"));

        const aBatchKey = Object.keys(ahmap).find(k=>k.includes("batch")) || "batch";

        const aRollKey = Object.keys(ahmap).find(k=>k.includes("roll")) || "rollno";

        const aStatusKey = Object.keys(ahmap).find(k=>k.includes("status")) || "status";

        const aReasonKey = Object.keys(ahmap).find(k=>k.includes("reason")) || "reason";

        // build map of existing attendance for the given date & batch

        const attRows = attAll.slice(1);

const attMap = {};

attRows.forEach(r => {

  let sheetDate = r[ahmap[aDateKey]];

  if (sheetDate instanceof Date) {

    sheetDate = Utilities.formatDate(sheetDate, Session.getScriptTimeZone(), "yyyy-MM-dd");

  } else {

    sheetDate = String(sheetDate).trim();

  }

  const rBatch = String(r[ahmap[aBatchKey]] || "").trim();

  const rRoll = String(r[ahmap[aRollKey]] || "").trim();

  if (sheetDate === date && rBatch === batch) {

    attMap[rRoll] = {

      status: String(r[ahmap[aStatusKey]] || "").trim(),

      reason: String(ahmap[aReasonKey] !== undefined ? (r[ahmap[aReasonKey]]||"").trim() : "")

    };

  }

});

        // apply to results

        results.forEach(s => {

          if (attMap[s.RollNo]) {

            s.Status = attMap[s.RollNo].status || s.Status;

            if (attMap[s.RollNo].reason) s.Reason = attMap[s.RollNo].reason;

          }

        });

      }

    }

    return { success: true, students: results };

  } catch (err) {

    return { success: false, message: "Error in getStudents: " + err.message };

  }

}

/\* === SAVE ATTENDANCE (Update/Delete logic added) === \*/

function saveAttendance(date, batch, updates) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

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

  if (!sheet) sheet = ss.insertSheet(ATTENDANCE\_SHEET);

  const header = [

    "Date","Batch","RollNo","Name","ParentName","ParentPhone",

    "Status","Reason","Department","Year","Section"

  ];

  if (sheet.getLastRow() === 0) sheet.appendRow(header);

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

  const headers = allRows.shift();

  const dateCol = headers.indexOf("Date");

  const batchCol = headers.indexOf("Batch");

  const rollCol = headers.indexOf("RollNo");

  // Build a map: normalized key = yyyy-MM-dd|batch|roll

  const existingMap = {};

  allRows.forEach((r, i) => {

    let sheetDate = r[dateCol];

    if (sheetDate instanceof Date) {

      sheetDate = Utilities.formatDate(sheetDate, Session.getScriptTimeZone(), "yyyy-MM-dd");

    } else {

      sheetDate = String(sheetDate).trim();

    }

    const key = `${sheetDate}|${r[batchCol]}|${r[rollCol]}`;

    existingMap[key] = i + 2;

  });

  const toInsert = [];

  updates.forEach(s => {

    const key = `${date}|${batch}|${s.RollNo}`;

    const rowData = [

      date, batch, s.RollNo, s.Name, s.ParentName, s.ParentPhone,

      s.Status, s.Reason || "", s.Department, s.Year, s.Section

    ];

    if (s.Status === "Absent" || s.Status === "OD") {

      if (existingMap[key]) {

        sheet.getRange(existingMap[key], 1, 1, rowData.length).setValues([rowData]);

      } else {

        toInsert.push(rowData);

      }

    } else if (s.Status === "Present" && existingMap[key]) {

      // Normalize all date comparisons before delete

      sheet.deleteRow(existingMap[key]);

    }

  });

  if (toInsert.length > 0) {

    sheet.getRange(sheet.getLastRow() + 1, 1, toInsert.length, header.length).setValues(toInsert);

  }

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

}

/\* === Get Absentees/OD for Faculty === \*/

function getAbsenteesForFaculty(email, date, batch) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const usersSheet = ss.getSheetByName("Users");

  const studentsSheet = ss.getSheetByName("Students");

  const attendanceSheet = ss.getSheetByName("Absentees\_OD");

  // ---- Step 1: Find faculty info ----

  const usersData = usersSheet.getDataRange().getValues();

  const uHeader = usersData[0];

  const emailIdx = uHeader.indexOf("Email");

  const deptIdx = uHeader.indexOf("Department");

  const yearIdx = uHeader.indexOf("Year");

  const secIdx = uHeader.indexOf("Section");

  let facultyInfo = null;

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

    if (String(usersData[i][emailIdx]).trim().toLowerCase() === String(email).trim().toLowerCase()) {

      facultyInfo = {

        department: usersData[i][deptIdx],

        year: usersData[i][yearIdx],

        section: usersData[i][secIdx],

      };

      break;

    }

  }

  if (!facultyInfo)

    return { success: false, message: "Faculty not found or section not assigned." };

  // ---- Step 2: Get all students for that section ----

  const stuData = studentsSheet.getDataRange().getValues();

  const sHeader = stuData[0];

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

  const yearCol = sHeader.indexOf("Year");

  const secCol = sHeader.indexOf("Section");

  const rollCol = sHeader.indexOf("RollNo");

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

  const parentNameCol = sHeader.indexOf("ParentName");

  const parentPhoneCol = sHeader.indexOf("ParentPhone");

  const students = stuData

    .filter((r, i) =>

      i > 0 &&

      r[deptCol] === facultyInfo.department &&

      r[yearCol] === facultyInfo.year &&

      r[secCol] === facultyInfo.section

    )

    .map(r => ({

      RollNo: r[rollCol],

      Name: r[nameCol],

      Year: r[yearCol],

      Section: r[secCol],

      ParentName: r[parentNameCol] || "",

      ParentPhone: r[parentPhoneCol] || "",

      Status: "Present",

      Reason: ""

    }));

  // ---- Step 3: Merge existing attendance ----

  const attData = attendanceSheet.getDataRange().getValues();

  const aHeader = attData[0];

  const dateCol = aHeader.indexOf("Date");

  const batchCol = aHeader.indexOf("Batch");

  const rollCol2 = aHeader.indexOf("RollNo");

  const statusCol = aHeader.indexOf("Status");

  const reasonCol = aHeader.indexOf("Reason");

  const phoneCol = aHeader.indexOf("ParentPhone");

  const attendanceMap = {};

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

    const row = attData[i];

    const rowDate = row[dateCol] instanceof Date

      ? Utilities.formatDate(row[dateCol], Session.getScriptTimeZone(), "yyyy-MM-dd")

      : String(row[dateCol]);

    if (rowDate === date && String(row[batchCol]) === String(batch)) {

      attendanceMap[row[rollCol2]] = {

        Status: row[statusCol],

        Reason: row[reasonCol],

        ParentPhone: row[phoneCol]

      };

    }

  }

  // ---- Step 4: Merge status & reason into student list ----

  const finalList = students.map(s => {

    const a = attendanceMap[s.RollNo];

    if (a) {

      s.Status = a.Status || "Present";

      s.Reason = a.Reason || "";

      s.ParentPhone = a.ParentPhone || s.ParentPhone;

    }

    return s;

  });

  return {

    success: true,

    message: "Absentees data fetched successfully.",

    records: finalList,

    department: facultyInfo.department,

    year: facultyInfo.year,

    section: facultyInfo.section

  };

}

/\* === Faculty Update Handler === \*/

function saveFacultyUpdates(date, batch, updates) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const sheet = ss.getSheetByName(ATTENDANCE\_SHEET);

  if (!sheet) return { success: false, message: `"${ATTENDANCE\_SHEET}" not found.` };

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

  const headers = all[0];

  const dateCol = headers.indexOf("Date");

  const batchCol = headers.indexOf("Batch");

  const rollCol = headers.indexOf("RollNo");

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

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

  const yearCol = headers.indexOf("Year");

  const secCol = headers.indexOf("Section");

  const phoneCol = headers.indexOf("ParentPhone");

  const reasonCol = headers.indexOf("Reason");

  const statusCol = headers.indexOf("Status");

  const existing = all.slice(1).map((r, i) => ({

    row: i + 2,

    date: r[dateCol] instanceof Date

      ? Utilities.formatDate(r[dateCol], Session.getScriptTimeZone(), "yyyy-MM-dd")

      : String(r[dateCol]),

    batch: String(r[batchCol]),

    roll: String(r[rollCol])

  }));

  const toAppend = [];

  updates.forEach(s => {

    const found = existing.find(r => r.date === date && r.batch === batch && r.roll === s.RollNo);

    if (found) {

      if (phoneCol >= 0) sheet.getRange(found.row, phoneCol + 1).setValue(s.ParentPhone);

      if (reasonCol >= 0) sheet.getRange(found.row, reasonCol + 1).setValue(s.Reason || "");

      if (statusCol >= 0) sheet.getRange(found.row, statusCol + 1).setValue(s.Status);

    } else {

      // Append new row if not existing

      toAppend.push([

        date,

        batch,

        s.RollNo,

        s.Name,

        s.Department || "",

        s.Year,

        s.Section,

        s.ParentPhone || "",

        s.Reason || "",

        s.Status || ""

      ]);

    }

  });

  if (toAppend.length > 0)

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

  return { success: true, message: "Updates saved successfully." };

}

/\* === Upload Users === \*/

function uploadUsers(users) {

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

    return { success:false, message:"No users to upload." };

  const ss = SpreadsheetApp.getActiveSpreadsheet();

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

  if (!sheet) sheet = ss.insertSheet(USERS\_SHEET);

  const header = ["Email","Password","Name","Role","Department"];

  if (sheet.getLastRow() === 0) sheet.appendRow(header);

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

  const existingEmails = allRows.slice(1).map(r => String(r[0]).toLowerCase());

  const toInsert = [];

  users.forEach(u => {

    if (!u.Email || existingEmails.includes(u.Email.toLowerCase())) return;

    toInsert.push([u.Email,u.Password,u.Name,u.Role,u.Department]);

  });

  if (toInsert.length > 0) {

    sheet.getRange(sheet.getLastRow()+1, 1, toInsert.length, header.length).setValues(toInsert);

  }

  return { success:true, message:`${toInsert.length} users uploaded successfully.` };

}

/\* === Delete User by Email === \*/

function deleteUser(email) {

  if (!email) return { success:false, message:"Email required." };

  const ss = SpreadsheetApp.getActiveSpreadsheet();

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

  if (!sheet) return { success:false, message:`Sheet "${USERS\_SHEET}" not found.` };

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

  const headers = all.shift();

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

  let rowToDelete = -1;

  for (let i=0;i<all.length;i++){

    if (String(all[i][emailCol]).toLowerCase() === email.toLowerCase()){

      rowToDelete = i + 2; // +2 because headers + 1-based indexing

      break;

    }

  }

  if (rowToDelete > 0){

    sheet.deleteRow(rowToDelete);

    return { success:true, message:`User ${email} deleted successfully.` };

  } else {

    return { success:false, message:`User ${email} not found.` };

  }

}

/\* === Upload Students === \*/

function uploadStudents(students) {

  if (!Array.isArray(students) || !students.length)

    return { success:false, message:"No students to upload." };

  const ss = SpreadsheetApp.getActiveSpreadsheet();

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

  if (!sheet) sheet = ss.insertSheet(STUDENTS\_SHEET);

  const header = ["RollNo","Name","ParentName","ParentPhone","Department","Year","Section"];

  if (sheet.getLastRow() === 0) sheet.appendRow(header);

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

  const existingRolls = allRows.slice(1).map(r => String(r[0]).toLowerCase());

  const toInsert = [];

  students.forEach(s => {

    if (!s.RollNo || existingRolls.includes(String(s.RollNo).toLowerCase())) return;

    toInsert.push([

      s.RollNo, s.Name, s.ParentName, s.ParentPhone, s.Department, s.Year, s.Section

    ]);

  });

  if (toInsert.length > 0) {

    sheet.getRange(sheet.getLastRow()+1, 1, toInsert.length, header.length).setValues(toInsert);

  }

  return { success:true, message:`${toInsert.length} students uploaded successfully.` };

}

/\* === Get Users by Department & Role === \*/

function getUsersByDept(department, role="Faculty") {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

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

  if (!sheet) return { success:false, message:`Sheet "${USERS\_SHEET}" not found.` };

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

  if (!all || all.length <= 1) return { success:true, records:[] };

  const headers = all[0].map(h => String(h||"").trim());

  const hmap = {};

  headers.forEach((h,i)=>hmap[h.toLowerCase().replace(/\s+/g,'')] = i);

  const emailCol = hmap["email"];

  const nameCol = hmap["name"];

  const roleCol = hmap["role"];

  const deptCol = hmap["department"];

  const rows = all.slice(1)

    .filter(r =>

      String(r[deptCol]).trim().toUpperCase() === department.trim().toUpperCase() &&

      String(r[roleCol]).trim().toUpperCase() === role.trim().toUpperCase()

    )

    .map(r => ({

      Email: r[emailCol],

      Name: r[nameCol],

      Role: r[roleCol],

      Department: r[deptCol]

    }));

  return { success:true, users: rows };

}

/\* === Upload Users === \*/

function uploadUsers(users) {

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

    return { success:false, message:"No users to upload." };

  const ss = SpreadsheetApp.getActiveSpreadsheet();

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

  if (!sheet) sheet = ss.insertSheet(USERS\_SHEET);

  const header = ["Email","Password","Name","Role","Department","Year","Section"];

  if (sheet.getLastRow() === 0) sheet.appendRow(header);

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

  const existingEmails = allRows.slice(1).map(r => String(r[0]).toLowerCase());

  const toInsert = [];

  users.forEach(u => {

    if (!u.Email || existingEmails.includes(u.Email.toLowerCase())) return;

    toInsert.push([u.Email,u.Password,u.Name,u.Role,u.Department,u.Year,u.Section]);

  });

  if (toInsert.length > 0) {

    sheet.getRange(sheet.getLastRow()+1, 1, toInsert.length, header.length).setValues(toInsert);

  }

  return { success:true, message:`${toInsert.length} users uploaded successfully.` };

}

/\* === Delete User by Email === \*/

function deleteUser(email) {

  if (!email) return { success:false, message:"Email required." };

  const ss = SpreadsheetApp.getActiveSpreadsheet();

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

  if (!sheet) return { success:false, message:`Sheet "${USERS\_SHEET}" not found.` };

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

  const headers = all.shift();

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

  let rowToDelete = -1;

  for (let i=0;i<all.length;i++){

    if (String(all[i][emailCol]).toLowerCase() === email.toLowerCase()){

      rowToDelete = i + 2; // +2 because headers + 1-based indexing

      break;

    }

  }

  if (rowToDelete > 0){

    sheet.deleteRow(rowToDelete);

    return { success:true, message:`User ${email} deleted successfully.` };

  } else {

    return { success:false, message:`User ${email} not found.` };

  }

}

/\* === Upload Students === \*/

function uploadStudents(students) {

  if (!Array.isArray(students) || !students.length)

    return { success:false, message:"No students to upload." };

  const ss = SpreadsheetApp.getActiveSpreadsheet();

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

  if (!sheet) sheet = ss.insertSheet(STUDENTS\_SHEET);

  const header = ["RollNo","Name","ParentName","ParentPhone","Department","Year","Section"];

  if (sheet.getLastRow() === 0) sheet.appendRow(header);

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

  const existingRolls = allRows.slice(1).map(r => String(r[0]).toLowerCase());

  const toInsert = [];

  students.forEach(s => {

    if (!s.RollNo || existingRolls.includes(String(s.RollNo).toLowerCase())) return;

    toInsert.push([

      s.RollNo, s.Name, s.ParentName, s.ParentPhone, s.Department, s.Year, s.Section

    ]);

  });

  if (toInsert.length > 0) {

    sheet.getRange(sheet.getLastRow()+1, 1, toInsert.length, header.length).setValues(toInsert);

  }

  return { success:true, message:`${toInsert.length} students uploaded successfully.` };

}

function getStudentHistory(rollNo) {

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

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const sheet = ss.getSheetByName(ATTENDANCE\_SHEET);

  if (!sheet) return { success: false, message: `"${ATTENDANCE\_SHEET}" not found.` };

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

  if (!all || all.length <= 1) return { success: true, records: [] };

  const headers = all[0].map(h => String(h || "").trim());

  const hmap = {};

  headers.forEach((h, i) => { hmap[h.toLowerCase().replace(/\s+/g,'')] = i; });

  const dateCol = hmap["date"];

  const batchCol = hmap["batch"];

  const rollCol = hmap["rollno"] !== undefined ? hmap["rollno"] : (hmap["roll"] !== undefined ? hmap["roll"] : null);

  const statusCol = hmap["status"];

  const reasonCol = hmap["reason"];

  const rows = all.slice(1)

    .filter(r => String(r[rollCol]).trim() === String(rollNo).trim())

    .map(r => {

      let d = r[dateCol];

      if (d instanceof Date) d = Utilities.formatDate(d, Session.getScriptTimeZone(), "yyyy-MM-dd");

      else d = String(d).trim();

      return {

        Date: d,

        Batch: String(r[batchCol]),

        Status: statusCol !== undefined ? String(r[statusCol]) : "",

        Reason: (reasonCol !== undefined ? String(r[reasonCol]) : "")

      };

    })

    .sort((a,b) => (a.Date < b.Date ? 1 : -1)); // latest first

  return { success: true, records: rows };

}

// ========== FUNCTION: Get All HoDs (for Principal to manage) ==========

function getUsersByRole(role = "HoD") {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

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

  if (!sheet) return { success: false, message: `Sheet "${USERS\_SHEET}" not found.` };

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

  if (all.length <= 1) return { success: true, users: [] };

  const headers = all[0].map(h => String(h).trim().toLowerCase());

  const hmap = {};

  headers.forEach((h, i) => hmap[h] = i);

  const rows = all.slice(1)

    .filter(r => String(r[hmap["role"]]).toLowerCase() === role.toLowerCase())

    .map(r => ({

      Email: r[hmap["email"]],

      Name: r[hmap["name"]],

      Role: r[hmap["role"]],

      Department: r[hmap["department"]]

    }));

  return { success: true, users: rows };

}

// ========== FUNCTION: College Attendance Summary ==========

// ========== FUNCTION: College Attendance Summary (extended with class-wise records) ==========

function getCollegeAttendanceSummary(date, batch) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const studentsSheet = ss.getSheetByName(STUDENTS\_SHEET);

  const attendanceSheet = ss.getSheetByName(ATTENDANCE\_SHEET);

  if (!studentsSheet || !attendanceSheet)

      return { success: false, message: "Required sheets not found." };

  const allStudents = studentsSheet.getDataRange().getValues();

  if (allStudents.length <= 1) return { success: true, summary: [], absent: [], od: [], classRecords: [] };

  // header map for Students sheet

  const headers = allStudents[0].map(h => String(h || "").toLowerCase().trim());

  const hmap = {};

  headers.forEach((h, i) => hmap[h] = i);

  // Load attendance

  const allAttendance = attendanceSheet.getDataRange().getValues();

  const attHeaders = allAttendance[0].map(h => String(h || "").toLowerCase().trim());

  const attMap = {};

  attHeaders.forEach((h, i) => attMap[h] = i);

  // Pre-build attendance map: roll|date|batch -> status & reason

  const attLookup = {};

  allAttendance.slice(1).forEach(r => {

    let d = r[attMap["date"]];

    if (d instanceof Date) {

      d = Utilities.formatDate(d, Session.getScriptTimeZone(), "yyyy-MM-dd");

    } else {

      d = String(d).trim();

    }

    const roll = String(r[attMap["rollno"]] || r[attMap["roll"]] || "").trim();

    const batchVal = String(r[attMap["batch"]] || "").trim();

    const key = `${roll}|${d}|${batchVal}`;

    attLookup[key] = {

      status: String(r[attMap["status"]] || "").trim().toUpperCase(),

      reason: r[attMap["reason"]] || ""

    };

  });

  // Dept-level summary (existing)

  const deptSummary = {};

  const absentList = [];

  const odList = [];

  // New: classMap keyed by dept|year|section

  const classMap = {};

  allStudents.slice(1).forEach(s => {

    const dept = String(s[hmap["department"]] || "").trim();

    const roll = String(s[hmap["rollno"]] || s[hmap["roll"]] || "").trim();

    const year = String(s[hmap["year"]] || "").trim();

    const section = String(s[hmap["section"]] || "").trim();

    // Dept summary bookkeeping

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

    deptSummary[dept].total++;

    // Class summary bookkeeping

    const ckey = `${dept}|${year}|${section}`;

    if (!classMap[ckey]) {

      classMap[ckey] = {

        Department: dept,

        Year: year,

        Section: section,

        total: 0,

        present: 0,

        absent: 0,

        od: 0

      };

    }

    classMap[ckey].total++;

    // Determine attendance status for this student on given date+batch

    const lookupKey = `${roll}|${date}|${batch}`;

    const att = attLookup[lookupKey];

    const status = att ? (String(att.status || "").trim().toUpperCase()) : "PRESENT";

    if (status === "ABSENT") {

      deptSummary[dept].absent++;

      classMap[ckey].absent++;

      absentList.push({ RollNo: roll, Name: s[hmap["name"]], Department: dept, Year: year, Section: section, Reason: att ? att.reason : "" });

    } else if (status === "OD") {

      deptSummary[dept].od++;

      classMap[ckey].od++;

      odList.push({ RollNo: roll, Name: s[hmap["name"]], Department: dept, Year: year, Section: section, Reason: att ? att.reason : "" });

    } else {

      deptSummary[dept].present++;

      classMap[ckey].present++;

    }

  });

  const classRecords = Object.values(classMap);

  return {

    success: true,

    summary: Object.values(deptSummary),    // dept-level

    absent: absentList,

    od: odList,

    classRecords: classRecords              // class-level (dept-year-section)

  };

}

// ========== FUNCTION: Department Class-wise Summary ==========

function getDeptClassSummary(date, batch, department) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const studentsSheet = ss.getSheetByName(STUDENTS\_SHEET);

  const attendanceSheet = ss.getSheetByName(ATTENDANCE\_SHEET);

  if (!studentsSheet || !attendanceSheet)

    return { success: false, message: "Required sheets not found." };

  const students = studentsSheet.getDataRange().getValues();

  const headers = students[0].map(h => h.toLowerCase().trim());

  const hmap = {};

  headers.forEach((h, i) => hmap[h] = i);

  const attendance = attendanceSheet.getDataRange().getValues();

  const attHeaders = attendance[0].map(h => h.toLowerCase().trim());

  const attMap = {};

  attHeaders.forEach((h, i) => attMap[h] = i);

  const classMap = {}; // key = year|section

  students.slice(1)

    .filter(s => s[hmap["department"]].trim() === department)

    .forEach(s => {

      const year = s[hmap["year"]].trim();

      const section = s[hmap["section"]].trim();

      const key = `${year}|${section}`;

      if (!classMap[key]) classMap[key] = { year, section, total:0, present:0, absent:0, od:0 };

      classMap[key].total++;

      // check attendance

      const roll = s[hmap["rollno"]].trim();

      const att = attendance.slice(1).find(a =>

        String(a[attMap["rollno"]]).trim() === roll &&

        String(a[attMap["date"]]).trim() === date &&

        String(a[attMap["batch"]]).trim() === batch

      );

      let status = "PRESENT";

if (att) status = String(att[attMap["status"]]).trim().toUpperCase();

if (status === "ABSENT") {

  classMap[key].absent++;

} else if (status === "OD") {

  classMap[key].od++;

} else {

  classMap[key].present++;

}

    });

  return { success: true, records: Object.values(classMap) };

}

function getCollegeAbsenteesForOffice(date, batch) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const sheet = ss.getSheetByName("Absentees\_OD");

  if (!sheet)

    return { success: false, message: "Absentees\_OD sheet not found" };

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

  if (data.length <= 1)

    return { success: true, absentees: [], ods: [] };

  const headers = data[0].map(h => String(h).trim());

  const map = {};

  headers.forEach((h, i) => map[h.toLowerCase().replace(/\s+/g, '')] = i);

  // ✅ Proper date matching

  const filtered = data.slice(1).filter(r => {

    const cellDate = r[map["date"]];

    const formattedDate = cellDate instanceof Date

      ? Utilities.formatDate(cellDate, Session.getScriptTimeZone(), "yyyy-MM-dd")

      : String(cellDate);

    return formattedDate === date && String(r[map["batch"]]) === batch;

  });

  const absentees = [];

  const ods = [];

  filtered.forEach(r => {

    const record = {

      RollNo: r[map["rollno"]] || "",

      Name: r[map["name"]] || "",

      Department: r[map["department"]] || "",

      Year: r[map["year"]] || "",

      Section: r[map["section"]] || "",

      Reason: r[map["reason"]] || "",

      ParentName: r[map["parentname"]] || "",

      ParentPhone: r[map["parentphone"]] || ""

    };

    const status = String(r[map["status"]]).trim().toUpperCase();

    if (status === "OD") ods.push(record);

    else absentees.push(record);

  });

  return {

    success: true,

    absentees,

    ods

  };

}

function saveAttendanceStatus(date, batch, year, section, department, status) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const sheet = ss.getSheetByName("Attendance\_Status") || ss.insertSheet("Attendance\_Status");

  const headers = ["Date", "Batch", "Year", "Section", "Department", "Status"];

  if (sheet.getLastRow() === 0) sheet.appendRow(headers);

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

  // Check if record exists for same date+batch+year+section+dept

  let found = false;

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

    const [d, b, y, s, dept] = data[i];

    const formattedDate = d instanceof Date

      ? Utilities.formatDate(d, Session.getScriptTimeZone(), "yyyy-MM-dd")

      : d;

    if (formattedDate === date && b === batch && y === year && s === section && dept === department) {

      sheet.getRange(i + 1, 6).setValue(status); // Update Status

      found = true;

      break;

    }

  }

  // If not found, append new entry

  if (!found) sheet.appendRow([date, batch, year, section, department, status]);

  return { success: true, message: "Attendance status recorded successfully" };

}

function getAttendanceStatusSummary(e) {

  const { date, batch } = e;

  if (!date) return { success: false, message: "Date not provided." };

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const attSheet = ss.getSheetByName("Attendance\_Status");

  const stuSheet = ss.getSheetByName("Students");

  if (!attSheet || !stuSheet)

    return { success: false, message: "Required sheets not found." };

  const attData = attSheet.getDataRange().getValues();

  const stuData = stuSheet.getDataRange().getValues();

  if (attData.length < 2 || stuData.length < 2)

    return { success: false, message: "No data available." };

  const aHead = attData[0];

  const sHead = stuData[0];

  const idxDeptA = aHead.indexOf("Department");

  const idxYearA = aHead.indexOf("Year");

  const idxSectionA = aHead.indexOf("Section");

  const idxDateA = aHead.indexOf("Date");

  const idxBatchA = aHead.indexOf("Batch");

  const idxStatusA = aHead.indexOf("Status");

  const idxDeptS = sHead.indexOf("Department");

  const idxYearS = sHead.indexOf("Year");

  const idxSectionS = sHead.indexOf("Section");

  if (idxDeptA < 0 || idxYearA < 0 || idxSectionA < 0 || idxDateA < 0 || idxBatchA < 0)

    return { success: false, message: "Attendance\_status headers missing." };

  if (idxDeptS < 0 || idxYearS < 0 || idxSectionS < 0)

    return { success: false, message: "Students headers missing." };

  const targetBatch = batch.toUpperCase();

  // Normalize input date to yyyy-mm-dd

  const targetDateObj = new Date(date);

  if (isNaN(targetDateObj)) return { success: false, message: "Invalid date format." };

  const targetDateStr = Utilities.formatDate(targetDateObj, Session.getScriptTimeZone(), "yyyy-MM-dd");

  // Collect unique Dept-Year-Section combos from Students

  const allCombos = {};

  stuData.slice(1).forEach(r => {

    const dept = String(r[idxDeptS]).trim();

    const year = String(r[idxYearS]).trim();

    const section = String(r[idxSectionS]).trim();

    if (dept && year && section) {

      const key = `${dept}|${year}|${section}`;

      allCombos[key] = { Department: dept, Year: year, Section: section, Status: "Not Taken" };

    }

  });

  // Go through Attendance\_status and mark “Taken” where matching date & batch

  attData.slice(1).forEach(r => {

    let rowDate = r[idxDateA];

    if (!rowDate) return;

    // Convert sheet date to yyyy-mm-dd

    if (rowDate instanceof Date) {

      rowDate = Utilities.formatDate(rowDate, Session.getScriptTimeZone(), "yyyy-MM-dd");

    } else {

      // If string, try to convert reliably

      const d = new Date(rowDate);

      if (!isNaN(d)) rowDate = Utilities.formatDate(d, Session.getScriptTimeZone(), "yyyy-MM-dd");

    }

    const rowBatch = String(r[idxBatchA]).trim().toUpperCase();

    if (rowDate === targetDateStr && rowBatch === targetBatch) {

      const dept = String(r[idxDeptA]).trim();

      const year = String(r[idxYearA]).trim();

      const section = String(r[idxSectionA]).trim();

      const key = `${dept}|${year}|${section}`;

      if (allCombos[key]) allCombos[key].Status = "Taken";

    }

  });

  // Group by Department

  const summaryByDept = {};

  Object.values(allCombos).forEach(r => {

    if (!summaryByDept[r.Department]) summaryByDept[r.Department] = [];

    summaryByDept[r.Department].push({

      Year: r.Year,

      Section: r.Section,

      Status: r.Status

    });

  });

  // Final sorted summary

  const summary = Object.keys(summaryByDept)

    .sort()

    .map(dept => ({

      department: dept,

      records: summaryByDept[dept].sort(

        (a, b) => a.Year.localeCompare(b.Year) || a.Section.localeCompare(b.Section)

      )

    }));

  return { success: true, summary };

}

// --- Lock/Unlock Attendance (fixed date handling) ---

function lockAttendance(date, batch, lockedBy, role) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const sheet = ss.getSheetByName("AttendanceLock");

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

  // Convert to consistent date string (yyyy-mm-dd)

  const dateStr = formatDateString(date);

  let existingRow = null;

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

    const rowDate = formatDateString(data[i][0]);

    const rowBatch = data[i][1];

    if (rowDate === dateStr && rowBatch === batch) {

      existingRow = i + 1;

      break;

    }

  }

  if (existingRow) {

    sheet.getRange(existingRow, 3, 1, 3).setValues([[lockedBy, role, new Date()]]);

  } else {

    sheet.appendRow([dateStr, batch, lockedBy, role, new Date()]);

  }

  logLockUnlock(dateStr, batch, lockedBy, role, "Lock");

  return { success: true, message: `Attendance for ${dateStr} (${batch}) locked successfully.` };

}

function unlockAttendance(date, batch, unlockedBy, role) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const sheet = ss.getSheetByName("AttendanceLock");

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

  const dateStr = formatDateString(date);

  let deleted = false;

  for (let i = data.length - 1; i > 0; i--) {

    const rowDate = formatDateString(data[i][0]);

    const rowBatch = data[i][1];

    if (rowDate === dateStr && rowBatch === batch) {

      sheet.deleteRow(i + 1);

      deleted = true;

      break;

    }

  }

  if (deleted) {

    logLockUnlock(dateStr, batch, unlockedBy, role, "Unlock");

    return { success: true, message: `Attendance for ${dateStr} (${batch}) unlocked successfully.` };

  } else {

    return { success: false, message: `No existing lock found for ${dateStr} (${batch}).` };

  }

}

function checkLockStatus(date, batch) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const sheet = ss.getSheetByName("AttendanceLock");

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

  const dateStr = formatDateString(date);

  const found = data.find(r => formatDateString(r[0]) === dateStr && r[1] === batch);

  if (found) {

    return {

      success: true,

      locked: true,

      lockedBy: found[2],

      role: found[3],

      message: `Locked by ${found[3]} (${found[2]})`

    };

  }

  return { success: true, locked: false, message: "Unlocked" };

}

/\* Helper function \*/

function formatDateString(d) {

  if (!d) return "";

  if (Object.prototype.toString.call(d) === "[object Date]") {

    return Utilities.formatDate(d, Session.getScriptTimeZone(), "yyyy-MM-dd");

  }

  if (typeof d === "string" && d.includes("/")) {

    // Handle MM/DD/YYYY or DD/MM/YYYY

    const parts = d.split(/[/-]/);

    if (parts[0].length === 4) return d; // already yyyy-mm-dd

    if (Number(parts[0]) > 12) return `${parts[2]}-${parts[1]}-${parts[0]}`;

    return `${parts[2]}-${parts[0].padStart(2, "0")}-${parts[1].padStart(2, "0")}`;

  }

  return d;

}

// --- Log Lock/Unlock History ---

function logLockUnlock(date, batch, user, role, action) {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  let logSheet = ss.getSheetByName("LockHistory");

  // Create if not exists

  if (!logSheet) {

    logSheet = ss.insertSheet("LockHistory");

    logSheet.appendRow(["Timestamp", "Date", "Batch", "Action", "User", "Role"]);

  }

  logSheet.appendRow([

    new Date(),

    date,

    batch,

    action,

    user,

    role

  ]);

}