assets/images/README.md

```
# Company Assets

## Logo Instructions

1. Save the Traincape Technology logo as `traincape-logo.png` in this directory
2. Recommended dimensions: 300x180 pixels (PNG format)
3. The logo will appear at the top of all salary slips

## Current Logo Requirements:
- Format: PNG
- Filename: traincape-logo.png
- Max width: 100px (will be auto-scaled)
- Max height: 60px (will be auto-scaled)

The logo is positioned at the top center of the salary slip PDF.
```

assign_leads.js

```
const mongoose = require('mongoose');
const readline = require('readline');
require('dotenv').config();

mongoose.connect(process.env.MONGO_URI);

const Lead = require('./models/Lead');
const User = require('./models/User');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

async function assignLeads() {
  try {
    console.log('Ø=Ý' LEAD ASSIGNMENT TOOL Ø=Ý'\n');

    // Show available Lead Persons
    const leadPersons = await User.find({ role: 'Lead Person' });
    console.log('=== AVAILABLE LEAD PERSONS ===');
    leadPersons.forEach((lp, index) => {
      console.log(`${index + 1}. ${lp.fullName}`);
    });

    // Show unassigned leads by month
    const unassignedLeads = await Lead.find({
      $or: [
        { leadPerson: null },
        { leadPerson: { $exists: false } },
        { assignedTo: null },
        { assignedTo: { $exists: false } }
      ]
    }).sort({ createdAt: -1 });

    console.log(`\n=== UNASSIGNED LEADS ===`);
    console.log(`Total unassigned leads: ${unassignedLeads.length}`);
```

```
    // Group by month
    const leadsByMonth = {};
    unassignedLeads.forEach(lead => {
      const date = new Date(lead.createdAt);
      const monthKey = `${date.getFullYear()}-${String(date.getMonth() +
1).padStart(2, '0')}`;
      const monthName = date.toLocaleDateString('en-US', { year: 'numeric',
month: 'long' });

      if (!leadsByMonth[monthKey]) {
        leadsByMonth[monthKey] = { name: monthName, count: 0, leads: [] };
      }
      leadsByMonth[monthKey].count++;
      leadsByMonth[monthKey].leads.push(lead);
    });

    console.log('\nUnassigned leads by month:');
    Object.entries(leadsByMonth).forEach(([key, data]) => {
      console.log(`- ${data.name}: ${data.count} leads`);
    });

    if (unassignedLeads.length === 0) {
      console.log('\n'  No unassigned leads found!');
      process.exit(0);
    }

    // Ask for Lead Person
    const leadPersonAnswer = await new Promise((resolve) => {
      rl.question('\nØ=Üd Enter Lead Person number (1-' + leadPersons.length + '): ',
resolve);
    });

    const selectedLeadPerson = leadPersons[parseInt(leadPersonAnswer) - 1];
    if (!selectedLeadPerson) {
      console.log(''L Invalid Lead Person selection');
      process.exit(1);
    }

    console.log(`'  Selected: ${selectedLeadPerson.fullName}`);

    // Ask for month
    const monthKeys = Object.keys(leadsByMonth).sort();
    console.log('\nØ=ÜÅ Available months:');
    monthKeys.forEach((key, index) => {
      console.log(`${index + 1}. ${leadsByMonth[key].name}
(${leadsByMonth[key].count} leads)`);
    });

    const monthAnswer = await new Promise((resolve) => {
      rl.question('\nØ=ÜÅ Enter month number (1-' + monthKeys.length + ') or "all"
for all months: ', resolve);
    });

    let leadsToAssign = [];
    if (monthAnswer.toLowerCase() === 'all') {
      leadsToAssign = unassignedLeads;
      console.log(`'  Selected: All months (${unassignedLeads.length} leads)`);
    } else {
```

```
      const selectedMonthKey = monthKeys[parseInt(monthAnswer) - 1];
      if (!selectedMonthKey) {
        console.log(''L Invalid month selection');
        process.exit(1);
      }
      leadsToAssign = leadsByMonth[selectedMonthKey].leads;
      console.log(`'  Selected: ${leadsByMonth[selectedMonthKey].name}
(${leadsToAssign.length} leads)`);
    }

    // Confirm assignment
    const confirmAnswer = await new Promise((resolve) => {
      rl.question(`\n& þ   Assign ${leadsToAssign.length} leads to
${selectedLeadPerson.fullName}? (yes/no): `, resolve);
    });

    if (confirmAnswer.toLowerCase() !== 'yes') {
      console.log(''L Assignment cancelled');
      process.exit(0);
    }

    // Perform the assignment
    console.log('\nØ=Ý  Assigning leads...');

    const leadIds = leadsToAssign.map(lead => lead._id);
    const updateResult = await Lead.updateMany(
      { _id: { $in: leadIds } },
      {
        $set: {
          leadPerson: selectedLeadPerson._id,
          assignedTo: selectedLeadPerson._id,
          updatedAt: new Date()
        }
      }
    );

    console.log(`'  Successfully assigned ${updateResult.modifiedCount} leads to
${selectedLeadPerson.fullName}`);

    // Verify the assignment
    const verifyLeads = await Lead.find({
      _id: { $in: leadIds },
      leadPerson: selectedLeadPerson._id
    }).populate('leadPerson', 'fullName');

    console.log(`\nØ=ÜË Verification: ${verifyLeads.length} leads now assigned to
${selectedLeadPerson.fullName}`);

    if (verifyLeads.length > 0) {
      console.log('\nSample of assigned leads:');
      verifyLeads.slice(0, 5).forEach((lead, index) => {
        console.log(`${index + 1}. ${lead.name} (${new
Date(lead.createdAt).toLocaleDateString()})`);
      });
    }

    console.log(`\nØ<ß‰ Assignment complete! ${selectedLeadPerson.fullName} can now
see these leads in their Lead Person dashboard.`);
```

```
    process.exit(0);
  } catch (error) {
    console.error(''L Error:', error);
    process.exit(1);
  }
}

assignLeads();
```

check-db.js

```
const mongoose = require('mongoose');
const Sale = require('./models/Sale');

async function checkDatabase() {
  try {
    await mongoose.connect(process.env.MONGO_URI || 'mongodb://localhost:27017/
crm');
    console.log('Connected to MongoDB');

    const totalSales = await Sale.countDocuments();
    console.log('Total sales in database:', totalSales);

    const sales = await Sale.find().limit(3);
    console.log('Sample sales count:', sales.length);
    if (sales.length > 0) {
      console.log('First sale:', JSON.stringify(sales[0], null, 2));
    }

    const collections = await mongoose.connection.db.listCollections().toArray();
    console.log('Collections:', collections.map(c => c.name));

    process.exit(0);
  } catch (error) {
    console.error('Error:', error.message);
    process.exit(1);
  }
}

checkDatabase();
```

check-env.js

```
require('dotenv').config();

console.log('Checking environment variables...');
console.log('EMAIL_USER:', process.env.EMAIL_USER ? 'Set' : 'Not set');
console.log('EMAIL_PASS:', process.env.EMAIL_PASS ? 'Set' : 'Not set');
console.log('JWT_SECRET:', process.env.JWT_SECRET ? 'Set' : 'Not set');
console.log('MONGODB_URI:', process.env.MONGODB_URI ? 'Set' : 'Not set');
console.log('NODE_ENV:', process.env.NODE_ENV || 'development');
```

config/db.js

```
const mongoose = require('mongoose');

const connectDB = async () => {
```

```
  try {
    // Use environment variable for MongoDB connection
    const mongoUri = process.env.MONGO_URI;

    if (!mongoUri) {
      console.error('MongoDB connection string is not defined in environment
variables');
      process.exit(1);
    }

    const conn = await mongoose.connect(mongoUri, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });

    console.log(`MongoDB Connected: ${conn.connection.host}`);
  } catch (error) {
    console.error(`MongoDB Connection Error: ${error.message}`);
    process.exit(1);
  }
};

module.exports = connectDB;
```

config/storage.js

```
const multer = require('multer');
const path = require('path');
const fs = require('fs');

// Detect if running on Render.com
const isRender = process.env.RENDER === 'true';

// Base upload paths based on environment
const getBasePath = () => {
  if (isRender) {
    return '/tmp/crm-uploads'; // Use /tmp on Render
  }
  return process.env.NODE_ENV === 'production'
    ? '/var/www/crm/uploads'
    : path.join(__dirname, '..', 'uploads');
};

// Get base path
const basePath = getBasePath();

// Define all required upload directories
const UPLOAD_PATHS = {
  DOCUMENTS: path.join(basePath, 'documents'),
  EMPLOYEES: path.join(basePath, 'employees'),
  INCENTIVES: path.join(basePath, 'incentives'),
  PROFILE_PICTURES: path.join(basePath, 'profile-pictures')
};

// Ensure all upload directories exist
Object.values(UPLOAD_PATHS).forEach(dir => {
  try {
    fs.mkdirSync(dir, { recursive: true });
```

```
    console.log(`Created directory: ${dir}`);
  } catch (err) {
    if (err.code !== 'EEXIST') {
      console.error(`Failed to create directory ${dir}:`, err.message);
    }
  }
});

// Local storage configuration
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    // Choose appropriate directory based on upload type
    let uploadPath = UPLOAD_PATHS.DOCUMENTS; // default

    if (file.fieldname.includes('employee') ||
        ['photograph', 'tenthMarksheet', 'twelfthMarksheet', 'bachelorDegree',
         'postgraduateDegree', 'aadharCard', 'panCard', 'pcc', 'resume',
'offerLetter']
         .includes(file.fieldname)) {
      uploadPath = UPLOAD_PATHS.EMPLOYEES;
    } else if (file.fieldname.includes('incentive')) {
      uploadPath = UPLOAD_PATHS.INCENTIVES;
    } else if (file.fieldname.includes('profile')) {
      uploadPath = UPLOAD_PATHS.PROFILE_PICTURES;
    }

    // Create directory if it doesn't exist
    fs.mkdirSync(uploadPath, { recursive: true });
    cb(null, uploadPath);
  },
  filename: (req, file, cb) => {
    const timestamp = Date.now();
    const random = Math.round(Math.random() * 1E9);
    const ext = path.extname(file.originalname);
    cb(null, `${file.fieldname}-${timestamp}-${random}${ext}`);
  }
});

// File filter for security
const fileFilter = (req, file, cb) => {
  const allowedTypes = [
    'image/jpeg',
    'image/png',
    'image/gif',
    'application/pdf',
    'application/msword',
    'application/vnd.openxmlformats-officedocument.wordprocessingml.document'
  ];

  if (allowedTypes.includes(file.mimetype)) {
    cb(null, true);
  } else {
    cb(new Error(`Invalid file type: ${file.mimetype}. Allowed types:
${allowedTypes.join(', ')}`), false);
  }
};

// Configure multer
const upload = multer({
```

```
  storage: storage,
  fileFilter: fileFilter,
  limits: {
    fileSize: process.env.NODE_ENV === 'production' ? 5 * 1024 * 1024 : 10 * 1024
* 1024 // 5MB in prod, 10MB in dev
  }
});

module.exports = {
  upload,
  UPLOAD_PATHS,
  getBasePath
};
```

controllers/attendance.js

```
const Attendance = require('../models/Attendance');
const Employee = require('../models/Employee');
const User = require('../models/User');

// @desc    Check-in employee
// @route   POST /api/attendance/checkin
// @access  Private
exports.checkIn = async (req, res) => {
  try {
    const { notes } = req.body;

    // Find employee record
    const employee = await Employee.findOne({ userId: req.user.id });
    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee record not found'
      });
    }

    // Get today's date
    const today = new Date();
    today.setHours(0, 0, 0, 0);

    // Check if already checked in today
    const existingAttendance = await Attendance.findOne({
      employeeId: employee._id,
      date: today
    });

    if (existingAttendance) {
      return res.status(400).json({
        success: false,
        message: 'Already checked in for today'
      });
    }

    // Create new attendance record
    const attendance = await Attendance.create({
      employeeId: employee._id,
      userId: req.user.id,
      date: today,
```

```
      checkIn: new Date(),
      notes: notes || ''
    });

    res.status(201).json({
      success: true,
      data: attendance,
      message: 'Check-in successful'
    });
  } catch (error) {
    console.error('Check-in error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error during check-in'
    });
  }
};

// @desc    Check-out employee
// @route   PUT /api/attendance/checkout
// @access  Private
exports.checkOut = async (req, res) => {
  try {
    const { notes } = req.body;

    // Find employee record
    const employee = await Employee.findOne({ userId: req.user.id });
    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee record not found'
      });
    }

    // Get today's date
    const today = new Date();
    today.setHours(0, 0, 0, 0);

    // Find today's attendance record
    const attendance = await Attendance.findOne({
      employeeId: employee._id,
      date: today
    });

    if (!attendance) {
      return res.status(404).json({
        success: false,
        message: 'No check-in found for today'
      });
    }

    if (attendance.checkOut) {
      return res.status(400).json({
        success: false,
        message: 'Already checked out for today'
      });
    }

    // Update attendance with check-out time
```

```
    attendance.checkOut = new Date();
    if (notes) attendance.notes = notes;

    await attendance.save();

    res.status(200).json({
      success: true,
      data: attendance,
      message: 'Check-out successful'
    });
  } catch (error) {
    console.error('Check-out error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error during check-out'
    });
  }
};

// @desc    Get attendance status for today
// @route   GET /api/attendance/today
// @access  Private
exports.getTodayAttendance = async (req, res) => {
  try {
    // Find employee record
    const employee = await Employee.findOne({ userId: req.user.id });
    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee record not found'
      });
    }

    // Get today's date
    const today = new Date();
    today.setHours(0, 0, 0, 0);

    // Find today's attendance
    const attendance = await Attendance.findOne({
      employeeId: employee._id,
      date: today
    });

    res.status(200).json({
      success: true,
      data: attendance,
      hasCheckedIn: !!attendance,
      hasCheckedOut: !!(attendance && attendance.checkOut)
    });
  } catch (error) {
    console.error('Get today attendance error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Get attendance history
```

```
// @route   GET /api/attendance/history
// @access  Private
exports.getAttendanceHistory = async (req, res) => {
  try {
    const { month, year, page = 1, limit = 30 } = req.query;

    // Find employee record
    const employee = await Employee.findOne({ userId: req.user.id });
    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee record not found'
      });
    }

    // Build query
    let query = { employeeId: employee._id };

    if (month && year) {
      const startDate = new Date(year, month - 1, 1);
      const endDate = new Date(year, month, 0);
      query.date = { $gte: startDate, $lte: endDate };
    }

    // Get attendance records with pagination
    const skip = (page - 1) * limit;
    const attendance = await Attendance.find(query)
      .sort({ date: -1 })
      .skip(skip)
      .limit(parseInt(limit));

    // Get total count
    const total = await Attendance.countDocuments(query);

    // Calculate statistics
    const stats = {
      totalDays: attendance.length,
      presentDays: attendance.filter(a => a.status === 'PRESENT').length,
      halfDays: attendance.filter(a => a.status === 'HALF_DAY').length,
      lateDays: attendance.filter(a => a.status === 'LATE').length,
      totalHours: attendance.reduce((sum, a) => sum + (a.totalHours || 0), 0)
    };

    res.status(200).json({
      success: true,
      data: attendance,
      stats: stats,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total: total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    console.error('Get attendance history error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
```

```
    });
  }
};

// @desc    Get all employees attendance (Admin/HR only)
// @route   GET /api/attendance/all
// @access  Private (Admin/HR/Manager)
exports.getAllAttendance = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to view all attendance'
      });
    }

    const { date, employeeId, department, page = 1, limit = 50 } = req.query;

    // Build query
    let query = {};

    if (date) {
      const queryDate = new Date(date);
      queryDate.setHours(0, 0, 0, 0);
      query.date = queryDate;
    }

    if (employeeId) {
      query.employeeId = employeeId;
    }

    // Get attendance records
    const skip = (page - 1) * limit;
    const attendance = await Attendance.find(query)
      .populate('employeeId', 'fullName email department')
      .populate('userId', 'fullName email')
      .sort({ date: -1 })
      .skip(skip)
      .limit(parseInt(limit));

    // Filter by department if specified
    let filteredAttendance = attendance;
    if (department) {
      filteredAttendance = attendance.filter(a =>
        a.employeeId && a.employeeId.department &&
        a.employeeId.department.toString() === department
      );
    }

    // Get total count
    const total = await Attendance.countDocuments(query);

    res.status(200).json({
      success: true,
      data: filteredAttendance,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
```

```
        total: total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    console.error('Get all attendance error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Update attendance (Admin/HR only)
// @route   PUT /api/attendance/:id
// @access  Private (Admin/HR/Manager)
exports.updateAttendance = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to update attendance'
      });
    }

    const { status, notes, totalHours } = req.body;

    const attendance = await Attendance.findById(req.params.id);
    if (!attendance) {
      return res.status(404).json({
        success: false,
        message: 'Attendance record not found'
      });
    }

    // Update fields
    if (status) attendance.status = status;
    if (notes) attendance.notes = notes;
    if (totalHours) attendance.totalHours = totalHours;

    attendance.approvedBy = req.user.id;

    await attendance.save();

    res.status(200).json({
      success: true,
      data: attendance,
      message: 'Attendance updated successfully'
    });
  } catch (error) {
    console.error('Update attendance error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};
```

```
// @desc    Get monthly attendance summary
// @route   GET /api/attendance/summary/:month/:year
// @access  Private
exports.getMonthlyAttendanceSummary = async (req, res) => {
  try {
    const { month, year } = req.params;

    // Find employee record
    const employee = await Employee.findOne({ userId: req.user.id });
    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee record not found'
      });
    }

    // Get date range for the month
    const startDate = new Date(year, month - 1, 1);
    const endDate = new Date(year, month, 0);

    // Get attendance records for the month
    const attendance = await Attendance.find({
      employeeId: employee._id,
      date: { $gte: startDate, $lte: endDate }
    }).sort({ date: 1 });

    // Calculate summary
    const workingDays = endDate.getDate();
    const presentDays = attendance.filter(a => a.status === 'PRESENT').length;
    const halfDays = attendance.filter(a => a.status === 'HALF_DAY').length;
    const lateDays = attendance.filter(a => a.status === 'LATE').length;
    const absentDays = workingDays - attendance.length;
    const totalHours = attendance.reduce((sum, a) => sum + (a.totalHours || 0),
0);
    const overtimeHours = attendance.reduce((sum, a) => sum + (a.overtimeHours ||
0), 0);

    const summary = {
      month: parseInt(month),
      year: parseInt(year),
      workingDays,
      presentDays,
      halfDays,
      lateDays,
      absentDays,
      totalHours: Math.round(totalHours * 100) / 100,
      overtimeHours: Math.round(overtimeHours * 100) / 100,
      attendancePercentage: Math.round((presentDays / workingDays) * 100 * 100) /
100,
      dailyAttendance: attendance
    };

    res.status(200).json({
      success: true,
      data: summary
    });
  } catch (error) {
    console.error('Get monthly attendance summary error:', error);
    res.status(500).json({
```

```
      success: false,
      message: 'Server error'
    });
  }
};
```

controllers/auth.js

```
const User = require("../models/User");
const bcrypt = require("bcrypt");
const fs = require('fs'); // Added for file cleanup
const path = require('path'); // Added for path.join
const { UPLOAD_PATHS } = require('../config/storage');
const asyncHandler = require('../middleware/async'); // Added for asyncHandler

// @desc    Register user
// @route   POST /api/auth/register
// @access  Public
exports.register = async (req, res) => {
  try {
    console.log("Register attempt:", {
      email: req.body.email,
      fullName: req.body.fullName,
      role: req.body.role
    });

    const { fullName, email, password, role } = req.body;

    // Basic validation
    if (!fullName || !email || !password) {
      console.log("Missing required registration fields");
      return res.status(400).json({
        success: false,
        message: "Please provide name, email and password"
      });
    }

    // Check if user already exists
    const existingUser = await User.findOne({ email });

    if (existingUser) {
      console.log(`User with email ${email} already exists`);
      return res.status(400).json({
        success: false,
        message: "Email already registered",
      });
    }

    console.log("Creating new user...");
    // Create user
    const user = await User.create({
      fullName,
      email,
      password,
      role: role || 'Sales Person', // Default role if not specified
    });

    console.log(`User created successfully with ID: ${user._id}`);
```

```
    sendTokenResponse(user, 201, res);
  } catch (err) {
    console.error("Registration error details:", {
      name: err.name,
      message: err.message,
      stack: err.stack,
      code: err.code
    });

    // Provide more specific error messages for common issues
    if (err.name === 'ValidationError') {
      const messages = Object.values(err.errors).map(val => val.message);
      return res.status(400).json({
        success: false,
        message: messages.join(', ')
      });
    }

    if (err.code === 11000) {
      return res.status(400).json({
        success: false,
        message: 'Email already registered'
      });
    }

    res.status(500).json({
      success: false,
      message: err.message || 'Internal server error during registration'
    });
  }
};

// @desc    Login user
// @route   POST /api/auth/login
// @access  Public
exports.login = asyncHandler(async (req, res) => {
  const { email, password } = req.body;

  // Validate email & password
  if (!email || !password) {
    return res.status(400).json({
      success: false,
      message: 'Please provide email and password'
    });
  }

  // Check for user
  const user = await User.findOne({ email }).select('+password');

  if (!user) {
    return res.status(401).json({
      success: false,
      message: 'Invalid credentials'
    });
  }

  // Check if password matches
  const isMatch = await user.matchPassword(password);
```

```
  if (!isMatch) {
    return res.status(401).json({
      success: false,
      message: 'Invalid credentials'
    });
  }

  // Create token
  const token = user.getSignedJwtToken();

  // Remove password from output
  user.password = undefined;

  res.status(200).json({
    success: true,
    token,
    data: user
  });
});

// @desc    Get current logged in user
// @route   GET /api/auth/me
// @access  Private
exports.getMe = async (req, res) => {
  const user = await User.findById(req.user.id);

  res.status(200).json({
    success: true,
    data: user,
  });
};

// @desc    Get all users for assignment
// @route   GET /api/auth/users
// @access  Private (only for Admin and Manager)
exports.getAllUsers = async (req, res) => {
  try {
    // Get the role filter from query params (if provided)
    const roleFilter = req.query.role || "";

    // Build the filter object
    const filter = {};
    if (roleFilter) {
      filter.role = roleFilter;
    }

    const users = await User.find(filter, "fullName email role");

    res.status(200).json({
      success: true,
      count: users.length,
      data: users,
    });
  } catch (err) {
    res.status(400).json({
      success: false,
      message: err.message,
    });
  }
```

```
};

// @desc    Update user
// @route   PUT /api/auth/users/:id
// @access  Private (Admin and Manager, but Manager cannot modify Admin accounts)
exports.updateUser = async (req, res) => {
  try {
    console.log(`Attempting to update user with ID: ${req.params.id}`);
    const { fullName, email, role } = req.body;

    // Check if user exists
    let user = await User.findById(req.params.id);

    if (!user) {
      console.log(`User not found with ID: ${req.params.id}`);
      return res.status(404).json({
        success: false,
        message: "User not found",
      });
    }

    // Prevent Managers from modifying Admin accounts
    if (req.user.role === 'Manager' && user.role === 'Admin') {
      return res.status(403).json({
        success: false,
        message: "Managers cannot modify Admin accounts",
      });
    }

    // Prevent Managers from creating new Admin accounts
    if (req.user.role === 'Manager' && role === 'Admin') {
      return res.status(403).json({
        success: false,
        message: "Managers cannot create Admin accounts",
      });
    }

    // Update basic user data
    user.fullName = fullName || user.fullName;
    user.email = email || user.email;
    user.role = role || user.role;

    // If password is provided, update it
    if (req.body.password && req.body.password.trim() !== "") {
      user.password = req.body.password;
      // The password will be hashed via the pre-save middleware
    }

    // Save the user - this will trigger the pre-save hook for password hashing
    await user.save();

    // Make sure we don't return the password
    user = await User.findById(user._id);

    console.log(`User updated successfully: ${user._id}`);
    res.status(200).json({
      success: true,
      data: user,
    });
```

```
  } catch (err) {
    console.error(`Error updating user: ${err.message}`);
    res.status(400).json({
      success: false,
      message: err.message,
    });
  }
};

// @desc    Delete user
// @route   DELETE /api/auth/users/:id
// @access  Private (Admin and Manager, but Manager cannot delete Admin accounts)
exports.deleteUser = async (req, res) => {
  try {
    console.log(`Attempting to delete user with ID: ${req.params.id}`);

    // Check if user exists
    const user = await User.findById(req.params.id);

    if (!user) {
      console.log(`User not found with ID: ${req.params.id}`);
      return res.status(404).json({
        success: false,
        message: "User not found",
      });
    }

    // Prevent user from deleting themselves
    if (user._id.toString() === req.user.id) {
      console.log("User attempted to delete their own account");
      return res.status(400).json({
        success: false,
        message: "You cannot delete your own account",
      });
    }

    // Prevent Managers from deleting Admin accounts
    if (req.user.role === 'Manager' && user.role === 'Admin') {
      return res.status(403).json({
        success: false,
        message: "Managers cannot delete Admin accounts",
      });
    }

    // Delete user with the findByIdAndDelete method
    const result = await User.findByIdAndDelete(req.params.id);

    if (!result) {
      return res.status(404).json({
        success: false,
        message: "Failed to delete user",
      });
    }

    console.log(`User deleted successfully: ${req.params.id}`);
    res.status(200).json({
      success: true,
      data: {},
    });
```

```
  } catch (err) {
    console.error(`Error deleting user: ${err.message}`);
    res.status(400).json({
      success: false,
      message: err.message,
    });
  }
};

// @desc    Update profile picture
// @route   PUT /api/auth/profile-picture
// @access  Private
exports.updateProfilePicture = async (req, res) => {
  try {
    console.log('Profile picture update requested by user:', req.user.id);
    console.log('Request files:', req.files);
    console.log('Request body:', req.body);

    // Check if a file was uploaded
    if (!req.files || !req.files.profilePicture) {
      console.log('No profile picture file provided in request');
      return res.status(400).json({
        success: false,
        message: 'Please provide a profile picture file'
      });
    }

    const profilePictureFile = req.files.profilePicture[0];
    console.log('Received profile picture file:', profilePictureFile.filename);

    // Store the file path in the database using the new UPLOAD_PATHS
    const profilePicturePath = path.join(UPLOAD_PATHS.PROFILE_PICTURES,
profilePictureFile.filename);

    // Update the user's profile picture
    console.log('Updating user profile picture in database');
    const user = await User.findByIdAndUpdate(
      req.user.id,
      { profilePicture: profilePicturePath },
      { new: true, runValidators: true }
    );

    if (!user) {
      console.log('User not found with ID:', req.user.id);
      return res.status(404).json({
        success: false,
        message: 'User not found'
      });
    }

    console.log('Profile picture updated successfully for user:', user._id);
    res.status(200).json({
      success: true,
      data: user,
      profilePicture: profilePicturePath
    });
  } catch (err) {
    console.error(`Error updating profile picture: ${err.message}`);
    console.error('Stack trace:', err.stack);
```

```
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};

// @desc    Create new user (for Admin and Manager)
// @route   POST /api/auth/users
// @access  Private (Admin and Manager, but Manager cannot create Admin accounts)
exports.createUser = async (req, res) => {
  try {
    console.log("Create user attempt by:", req.user.role, "for:", req.body);
    const { fullName, email, password, role } = req.body;

    // Basic validation
    if (!fullName || !email || !password) {
      console.log("Missing required fields for user creation");
      return res.status(400).json({
        success: false,
        message: "Please provide name, email and password"
      });
    }

    // Prevent Managers from creating Admin accounts
    if (req.user.role === 'Manager' && role === 'Admin') {
      return res.status(403).json({
        success: false,
        message: "Managers cannot create Admin accounts",
      });
    }

    // Check if user already exists
    const existingUser = await User.findOne({ email });

    if (existingUser) {
      console.log(`User with email ${email} already exists`);
      return res.status(400).json({
        success: false,
        message: "Email already registered",
      });
    }

    console.log("Creating new user...");
    // Create user
    const user = await User.create({
      fullName,
      email,
      password,
      role: role || 'Sales Person', // Default role if not specified
    });

    console.log(`User created successfully with ID: ${user._id}`);

    // Return user data without password
    const userData = await User.findById(user._id);

    res.status(201).json({
      success: true,
```

```
      data: userData,
    });
  } catch (err) {
    console.error("User creation error details:", {
      name: err.name,
      message: err.message,
      stack: err.stack,
      code: err.code
    });

    // Provide more specific error messages for common issues
    if (err.name === 'ValidationError') {
      const messages = Object.values(err.errors).map(val => val.message);
      return res.status(400).json({
        success: false,
        message: messages.join(', ')
      });
    }

    if (err.code === 11000) {
      return res.status(400).json({
        success: false,
        message: 'Email already registered'
      });
    }

    res.status(500).json({
      success: false,
      message: err.message || 'Internal server error during user creation'
    });
  }
};

// @desc    Create new user with documents (for Admin and Manager)
// @route   POST /api/auth/users/with-documents
// @access  Private (Admin and Manager, but Manager cannot create Admin accounts)
exports.createUserWithDocuments = async (req, res) => {
  try {
    console.log("Create user with documents attempt by:", req.user.role);
    console.log("Request body:", req.body);
    console.log("Request files:", req.files ? Object.keys(req.files) : 'No
files');

    const { fullName, email, password, role } = req.body;

    // Basic validation
    if (!fullName || !email || !password) {
      console.log("Missing required fields for user creation");
      return res.status(400).json({
        success: false,
        message: "Please provide name, email and password"
      });
    }

    // Prevent Managers from creating Admin accounts
    if (req.user.role === 'Manager' && role === 'Admin') {
      return res.status(403).json({
        success: false,
        message: "Managers cannot create Admin accounts",
```

```
  });
}

// Check if user already exists
const existingUser = await User.findOne({ email });

if (existingUser) {
  console.log(`User with email ${email} already exists`);
  return res.status(400).json({
    success: false,
    message: "Email already registered",
  });
}

// Create user
let user;
try {
  console.log("Creating new user...");
  user = await User.create({
    fullName,
    email,
    password,
    role: role || 'Sales Person', // Default role if not specified
  });
  console.log(`User created successfully with ID: ${user._id}`);
} catch (userError) {
  console.error("Error creating user:", userError);
  throw userError;
}

// Handle employee creation for non-admin/non-hr roles
if (['Sales Person', 'Lead Person', 'Manager', 'Employee'].includes(role)) {
  try {
    const Employee = require('../models/Employee');
    const Department = require('../models/Department');
    const Role = require('../models/EmployeeRole');

    // Find or create department
    let department = await Department.findOne({ name: 'General' });
    if (!department) {
      department = await Department.create({
        name: 'General',
        description: 'General Department for all employees'
      });
    }

    // Find or create role
    let employeeRole = await Role.findOne({ name: role });
    if (!employeeRole) {
      employeeRole = await Role.create({
        name: role,
        description: `Role for ${role}`
      });
    }

    // Create employee record
    const employeeData = {
      fullName,
      email,
```

```
          userId: user._id,
          role: employeeRole._id,
          department: department._id,

          // Add all the new fields from the admin form
          phoneNumber: req.body.phoneNumber || '',
          whatsappNumber: req.body.whatsappNumber || '',
          linkedInUrl: req.body.linkedInUrl || '',
          currentAddress: req.body.currentAddress || '',
          permanentAddress: req.body.permanentAddress || '',
          dateOfBirth: req.body.dateOfBirth || null,
          joiningDate: req.body.joiningDate || new Date(),
          salary: req.body.salary ? parseFloat(req.body.salary) : 0,
          status: req.body.status || 'ACTIVE',
          collegeName: req.body.collegeName || '',
          internshipDuration: req.body.internshipDuration ?
parseInt(req.body.internshipDuration) : null,

          // Initialize documents object
          documents: {}
        };

        // Process uploaded documents
        if (req.files) {
          const documentTypes = ['photograph', 'tenthMarksheet',
'twelfthMarksheet', 'bachelorDegree', 'postgraduateDegree', 'aadharCard',
'panCard', 'pcc', 'resume', 'offerLetter'];

          for (const docType of documentTypes) {
            if (req.files[docType] && req.files[docType][0]) {
              const file = req.files[docType][0];
              try {
                // Ensure the file was saved successfully
                if (!fs.existsSync(file.path)) {
                  console.error(`File not saved: ${file.path}`);
                  continue;
                }

                employeeData.documents[docType] = {
                  filename: file.filename,
                  originalName: file.originalname,
                  path: file.path,
                  mimetype: file.mimetype,
                  size: file.size,
                  uploadedAt: new Date()
                };
              } catch (fileError) {
                console.error(`Error processing file ${docType}:`, fileError);
                // Continue with other files if one fails
              }
            }
          }
        }

        const employee = await Employee.create(employeeData);
        console.log(`Employee created successfully with ID: ${employee._id}`);
      } catch (employeeError) {
        console.error("Error creating employee record:", employeeError);
        // If employee creation fails, delete the user and throw error
```

```
        await User.findByIdAndDelete(user._id);
        throw new Error(`Failed to create employee record:
${employeeError.message}`);
      }
    }

    // Return user data without password
    const userData = await User.findById(user._id);

    res.status(201).json({
      success: true,
      data: userData,
    });
  } catch (err) {
    console.error("User creation with documents error details:", {
      name: err.name,
      message: err.message,
      stack: err.stack,
      code: err.code
    });

    // Clean up any uploaded files if there was an error
    if (req.files) {
      Object.values(req.files).forEach(fileArray => {
        fileArray.forEach(file => {
          try {
            if (fs.existsSync(file.path)) {
              fs.unlinkSync(file.path);
              console.log(`Cleaned up file: ${file.path}`);
            }
          } catch (cleanupError) {
            console.error(`Error cleaning up file ${file.path}:`, cleanupError);
          }
        });
      });
    }

    // Provide more specific error messages for common issues
    if (err.name === 'ValidationError') {
      const messages = Object.values(err.errors).map(val => val.message);
      return res.status(400).json({
        success: false,
        message: messages.join(', ')
      });
    }

    if (err.code === 11000) {
      return res.status(400).json({
        success: false,
        message: 'Email already registered'
      });
    }

    res.status(500).json({
      success: false,
      message: err.message || 'Internal server error during user creation'
    });
  }
};
```

```
// @desc    Update user with documents
// @route   PUT /api/auth/users/:id/with-documents
// @access  Private (Admin and Manager, but Manager cannot modify Admin accounts)
exports.updateUserWithDocuments = async (req, res) => {
  try {
    console.log(`Attempting to update user with documents, ID: ${req.params.id}`);
    console.log('Request body:', req.body);
    console.log('Request files:', req.files ? Object.keys(req.files) : 'No
files');

    const { fullName, email, role } = req.body;

    // Check if user exists
    let user = await User.findById(req.params.id);

    if (!user) {
      console.log(`User not found with ID: ${req.params.id}`);
      return res.status(404).json({
        success: false,
        message: "User not found",
      });
    }

    // Prevent Managers from modifying Admin accounts
    if (req.user.role === 'Manager' && user.role === 'Admin') {
      return res.status(403).json({
        success: false,
        message: "Managers cannot modify Admin accounts",
      });
    }

    // Prevent Managers from making users Admin
    if (req.user.role === 'Manager' && role === 'Admin') {
      return res.status(403).json({
        success: false,
        message: "Managers cannot create Admin accounts",
      });
    }

    // Update user fields - only update provided fields
    const updateData = {};
    if (fullName && fullName.trim() !== '') updateData.fullName = fullName;
    if (email && email.trim() !== '') updateData.email = email;
    if (role && role.trim() !== '') updateData.role = role;

    // Handle password update if provided
    if (req.body.password && req.body.password.trim() !== '') {
      updateData.password = req.body.password;
    }

    // Ensure we have at least some data to update
    if (Object.keys(updateData).length === 0 && (!req.files ||
Object.keys(req.files).length === 0)) {
      return res.status(400).json({
        success: false,
        message: "No data provided for update"
      });
    }
```

```
    // Update user only if there's data to update
    if (Object.keys(updateData).length > 0) {
      user = await User.findByIdAndUpdate(req.params.id, updateData, {
        new: true,
        runValidators: true,
      });
    }

    console.log(`User updated successfully: ${user._id}`);

    // Handle employee update for non-admin/non-hr roles
    const userRole = updateData.role || user.role;
    if (['Sales Person', 'Lead Person', 'Manager',
'Employee'].includes(userRole)) {
      const Employee = require('../models/Employee');
      const Department = require('../models/Department');
      const Role = require('../models/EmployeeRole');

      // Find or create employee record
      let employee = await Employee.findOne({ userId: user._id });

      if (!employee) {
        // Find or create department
        let department = await Department.findOne({ name: 'General' });
        if (!department) {
          department = await Department.create({
            name: 'General',
            description: 'General Department for all employees'
          });
        }

        // Find or create role for the current user role
        let employeeRole = await Role.findOne({ name: userRole });
        if (!employeeRole) {
          employeeRole = await Role.create({
            name: userRole,
            description: `Role for ${userRole}`
          });
        }

        // Create new employee if doesn't exist
        employee = await Employee.create({
          fullName: user.fullName,
          email: user.email,
          userId: user._id,
          role: employeeRole._id,
          department: department._id,
          documents: {}
        });
      }

      // Update employee basic info with new values
      if (updateData.fullName) employee.fullName = updateData.fullName;
      if (updateData.email) employee.email = updateData.email;

      // Update employee role if user role changed
      if (updateData.role) {
        // Find or create role for the new role
```

```
        let newEmployeeRole = await Role.findOne({ name: updateData.role });
        if (!newEmployeeRole) {
          newEmployeeRole = await Role.create({
            name: updateData.role,
            description: `Role for ${updateData.role}`
          });
        }
        employee.role = newEmployeeRole._id;
      }

      // Update all employee fields from form
      if (req.body.phoneNumber !== undefined) employee.phoneNumber =
req.body.phoneNumber;
      if (req.body.whatsappNumber !== undefined) employee.whatsappNumber =
req.body.whatsappNumber;
      if (req.body.linkedInUrl !== undefined) employee.linkedInUrl =
req.body.linkedInUrl;
      if (req.body.currentAddress !== undefined) employee.currentAddress =
req.body.currentAddress;
      if (req.body.permanentAddress !== undefined) employee.permanentAddress =
req.body.permanentAddress;
      if (req.body.dateOfBirth !== undefined) employee.dateOfBirth =
req.body.dateOfBirth ? new Date(req.body.dateOfBirth) : null;
      if (req.body.joiningDate !== undefined) employee.joiningDate =
req.body.joiningDate ? new Date(req.body.joiningDate) : null;
      if (req.body.salary !== undefined) employee.salary = req.body.salary ?
parseFloat(req.body.salary) : 0;
      if (req.body.status !== undefined) employee.status = req.body.status;
      if (req.body.collegeName !== undefined) employee.collegeName =
req.body.collegeName;
      if (req.body.internshipDuration !== undefined) employee.internshipDuration
= req.body.internshipDuration ? parseInt(req.body.internshipDuration) : null;

      // Update department if provided
      if (req.body.department !== undefined && req.body.department !== '') {
        const department = await Department.findById(req.body.department);
        if (department) {
          employee.department = department._id;
        }
      }

      // Update employee role if provided
      if (req.body.employeeRole !== undefined && req.body.employeeRole !== '') {
        const employeeRole = await Role.findById(req.body.employeeRole);
        if (employeeRole) {
          employee.role = employeeRole._id;
        }
      }

      // Process uploaded documents if any
      if (req.files) {
        if (!employee.documents) {
          employee.documents = {};
        }

        const documentTypes = ['photograph', 'tenthMarksheet',
'twelfthMarksheet', 'bachelorDegree', 'postgraduateDegree', 'aadharCard',
'panCard', 'pcc', 'resume', 'offerLetter'];
```

```
        for (const docType of documentTypes) {
          if (req.files[docType]) {
            const file = req.files[docType][0];
            employee.documents[docType] = {
              filename: file.filename, // Use the generated filename, not
originalname
              originalName: file.originalname, // Store original name separately
              path: file.path,
              mimetype: file.mimetype,
              size: file.size,
              uploadedAt: new Date()
            };
          }
        }
      }

      await employee.save();
      console.log(`Employee updated successfully with ID: ${employee._id}`);
    } else {
      console.log(`User role ${userRole} does not require employee record`);
    }

    res.status(200).json({
      success: true,
      data: user,
    });
  } catch (err) {
    console.error(`Error updating user with documents: ${err.message}`);
    res.status(400).json({
      success: false,
      message: err.message,
    });
  }
};

// Get token from model, create cookie and send response
const sendTokenResponse = (user, statusCode, res) => {
  // Create token
  const token = user.getSignedJwtToken();

  const options = {
    expires: new Date(
      Date.now() + (process.env.JWT_COOKIE_EXPIRE || 30) * 24 * 60 * 60 * 1000
    ),
    httpOnly: true,
  };

  if (process.env.NODE_ENV === "production") {
    options.secure = true;
  }

  res.status(statusCode).json({
    success: true,
    token,
    user: {
      id: user._id,
      name: user.fullName,
      email: user.email,
      role: user.role,
```

```
      profilePicture: user.profilePicture
    },
  });
};
```

controllers/chatController.js

```
const ChatService = require('../services/chatService');
const User = require('../models/User');

// @desc    Send a message
// @route   POST /api/chat/messages
// @access  Private
const sendMessage = async (req, res) => {
  try {
    const { recipientId, content, messageType } = req.body;
    const senderId = req.user._id;

    if (!recipientId || !content) {
      return res.status(400).json({
        success: false,
        message: 'Recipient ID and content are required'
      });
    }

    const message = await ChatService.saveMessage({
      senderId,
      recipientId,
      content,
      messageType
    });

    // Emit the message via Socket.IO
    const io = req.app.get('io');
    if (io) {
      // Send to recipient
      io.to(`user-${recipientId}`).emit('newMessage', {
        _id: message._id,
        chatId: message.chatId,
        senderId: message.senderId,
        recipientId: message.recipientId,
        content: message.content,
        messageType: message.messageType,
        timestamp: message.timestamp,
        isRead: message.isRead
      });

      // Send notification to recipient
      io.to(`user-${recipientId}`).emit('messageNotification', {
        senderId: message.senderId,
        senderName: message.senderId.fullName,
        content: message.content,
        timestamp: message.timestamp
      });
    }

    res.status(201).json({
      success: true,
```

```
      data: message
    });
  } catch (error) {
    console.error('Error sending message:', error);
    res.status(500).json({
      success: false,
      message: error.message
    });
  }
};

// @desc    Get chat messages between two users
// @route   GET /api/chat/messages/:recipientId
// @access  Private
const getChatMessages = async (req, res) => {
  try {
    const { recipientId } = req.params;
    const senderId = req.user._id;
    const { page = 1, limit = 50 } = req.query;

    const messages = await ChatService.getChatMessages(
      senderId,
      recipientId,
      parseInt(page),
      parseInt(limit)
    );

    res.status(200).json({
      success: true,
      data: messages,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total: messages.length
      }
    });
  } catch (error) {
    console.error('Error getting chat messages:', error);
    res.status(500).json({
      success: false,
      message: error.message
    });
  }
};

// @desc    Get user's chat rooms
// @route   GET /api/chat/rooms
// @access  Private
const getChatRooms = async (req, res) => {
  try {
    const userId = req.user._id;
    const chatRooms = await ChatService.getUserChatRooms(userId);

    res.status(200).json({
      success: true,
      data: chatRooms
    });
  } catch (error) {
    console.error('Error getting chat rooms:', error);
```

```
    res.status(500).json({
      success: false,
      message: error.message
    });
  }
};

// @desc    Get online users
// @route   GET /api/chat/users/online
// @access  Private
const getOnlineUsers = async (req, res) => {
  try {
    const userId = req.user._id;
    const users = await ChatService.getOnlineUsers(userId);

    res.status(200).json({
      success: true,
      data: users
    });
  } catch (error) {
    console.error('Error getting online users:', error);
    res.status(500).json({
      success: false,
      message: error.message
    });
  }
};

// @desc    Get all users for chat
// @route   GET /api/chat/users
// @access  Private
const getAllUsersForChat = async (req, res) => {
  try {
    const currentUserId = req.user.id;

    // Get all users except current user, including customers
    const users = await User.find({
      _id: { $ne: currentUserId }
    }).select('fullName email role chatStatus lastSeen profilePicture');

    res.status(200).json({
      success: true,
      data: users
    });
  } catch (error) {
    console.error('Error fetching users for chat:', error);
    res.status(500).json({
      success: false,
      message: 'Error fetching users for chat'
    });
  }
};

// @desc    Update user chat status
// @route   PUT /api/chat/status
// @access  Private
const updateChatStatus = async (req, res) => {
  try {
    const { status } = req.body;
```

```
    const userId = req.user._id;

    if (!['ONLINE', 'OFFLINE', 'AWAY'].includes(status)) {
      return res.status(400).json({
        success: false,
        message: 'Invalid status. Must be ONLINE, OFFLINE, or AWAY'
      });
    }

    await ChatService.updateUserStatus(userId, status);

    // Emit status update via Socket.IO
    const io = req.app.get('io');
    if (io) {
      io.emit('userStatusUpdate', {
        userId,
        status,
        lastSeen: new Date()
      });
    }

    res.status(200).json({
      success: true,
      message: 'Status updated successfully'
    });
  } catch (error) {
    console.error('Error updating chat status:', error);
    res.status(500).json({
      success: false,
      message: error.message
    });
  }
};

// @desc    Mark messages as read
// @route   PUT /api/chat/messages/read/:senderId
// @access  Private
const markMessagesAsRead = async (req, res) => {
  try {
    const { senderId } = req.params;
    const recipientId = req.user._id;

    // This is handled automatically in getChatMessages, but we can also provide
a separate endpoint
    await ChatService.getChatMessages(recipientId, senderId, 1, 1);

    res.status(200).json({
      success: true,
      message: 'Messages marked as read'
    });
  } catch (error) {
    console.error('Error marking messages as read:', error);
    res.status(500).json({
      success: false,
      message: error.message
    });
  }
};
```

```
module.exports = {
  sendMessage,
  getChatMessages,
  getChatRooms,
  getOnlineUsers,
  getAllUsersForChat,
  updateChatStatus,
  markMessagesAsRead
};
```

controllers/currency.js

```
const axios = require('axios');

let cachedRates = null;
let lastFetched = 0;
const CACHE_DURATION = 60 * 60 * 1000; // 1 hour cache

// Try multiple APIs until one works
const APIs = [
  { url: 'https://api.exchangerate-api.com/v4/latest/USD', path: 'rates' },
  { url: 'https://open.er-api.com/v6/latest/USD', path: 'rates' },
  { url: 'https://api.exchangerate.host/latest?base=USD', path: 'rates' }
];

// Expanded list of supported currencies
const SUPPORTED_CURRENCIES = [
  'USD', 'INR', 'EUR', 'GBP', 'JPY', 'CAD', 'AUD', 'CNY',
  'SGD', 'CHF', 'AED', 'ZAR', 'BRL', 'MXN', 'HKD', 'SEK',
  'NZD', 'THB', 'IDR', 'MYR', 'PHP', 'SAR', 'KRW', 'VND'
];

// Fallback rates in case all APIs fail
const fallbackRates = {
  USD: 1,
  INR: 84.62,
  EUR: 0.92,
  GBP: 0.79,
  JPY: 151.15,
  CAD: 1.37,
  AUD: 1.52,
  CNY: 7.24,
  SGD: 1.35,
  CHF: 0.90,
  AED: 3.67,
  ZAR: 18.39,
  BRL: 5.14,
  MXN: 17.04,
  HKD: 7.81,
  SEK: 10.58,
  NZD: 1.64,
  THB: 36.25,
  IDR: 15928.30,
  MYR: 4.72,
  PHP: 57.14,
  SAR: 3.75,
  KRW: 1362.26,
  VND: 25162.50
```

```
};

// Get exchange rates from API or cache
exports.getRates = async (req, res) => {
  const now = Date.now();

  // Return cached rates if fresh enough
  if (cachedRates && now - lastFetched < CACHE_DURATION) {
    return res.json(cachedRates);
  }

  // Try APIs in sequence until one succeeds
  for (const api of APIs) {
    try {
      console.log(`Attempting to fetch rates from: ${api.url}`);
      const response = await axios.get(api.url);

      if (response.data && response.data[api.path]) {
        // We have a valid response
        const rates = response.data[api.path];

        // Create a standardized response format with all supported currencies
        const standardizedRates = { USD: 1 }; // USD is always 1 as base

        // Add all available currencies from the API response
        for (const currency of SUPPORTED_CURRENCIES) {
          if (rates[currency]) {
            standardizedRates[currency] = rates[currency];
          }
        }

        cachedRates = {
          base: 'USD',
          date: new Date().toISOString(),
          rates: standardizedRates
        };

        lastFetched = now;
        console.log(`Exchange rates updated successfully with
${Object.keys(standardizedRates).length} currencies`);
        return res.json(cachedRates);
      }
    } catch (err) {
      console.error(`API ${api.url} failed: ${err.message}`);
      // Continue to next API
    }
  }

  // All APIs failed, use fallback rates
  console.log('All currency APIs failed, using fallback rates');
  if (!cachedRates) {
    // First-time failure, create cache with fallback rates
    cachedRates = {
      base: 'USD',
      date: new Date().toISOString(),
      rates: fallbackRates,
      source: 'fallback'
    };
  } else {
```

```
    // Update existing cache timestamp but keep the rates
    cachedRates.date = new Date().toISOString();
    cachedRates.source = 'fallback';
  }

  lastFetched = now;
  res.json(cachedRates);
};
```

controllers/documentation.js

```
const PDFDocument = require('pdfkit');
const fs = require('fs');
const path = require('path');

// Helper function to draw a system component box
const drawComponentBox = (doc, x, y, width, height, title, description) => {
  doc.rect(x, y, width, height).stroke();
  doc.fontSize(8).font('Helvetica-Bold').text(title, x + 5, y + 5);
  doc.fontSize(7).font('Helvetica').text(description, x + 5, y + 20, {
    width: width - 10,
    align: 'left'
  });
};

// Helper function to draw a connection arrow
const drawArrow = (doc, startX, startY, endX, endY, label) => {
  doc.moveTo(startX, startY)
     .lineTo(endX, endY)
     .stroke();

  // Arrow head
  const angle = Math.atan2(endY - startY, endX - startX);
  const arrowLength = 10;
  doc.moveTo(endX, endY)
     .lineTo(
       endX - arrowLength * Math.cos(angle - Math.PI / 6),
       endY - arrowLength * Math.sin(angle - Math.PI / 6)
     )
     .moveTo(endX, endY)
     .lineTo(
       endX - arrowLength * Math.cos(angle + Math.PI / 6),
       endY - arrowLength * Math.sin(angle + Math.PI / 6)
     )
     .stroke();

  // Label
  if (label) {
    doc.fontSize(6).text(label,
      (startX + endX) / 2 - 20,
      (startY + endY) / 2 - 10,
      { width: 40, align: 'center' }
    );
  }
};

exports.generateProjectDocumentation = async (req, res) => {
  try {
```

```
    const chunks = [];
    const doc = new PDFDocument();

    // Track PDF size
    let totalSize = 0;
    doc.on('data', chunk => {
      totalSize += chunk.length;
      chunks.push(chunk);

      if (totalSize > 20480) {
        doc.end();
        return res.status(400).json({
          success: false,
          message: 'PDF size would exceed 20KB limit.'
        });
      }
    });

    const filename = `CRM_Project_Documentation.pdf`;
    const filepath = path.join(__dirname, '../documentation/', filename);

    // Ensure directory exists
    const dir = path.dirname(filepath);
    if (!fs.existsSync(dir)) {
      fs.mkdirSync(dir, { recursive: true });
    }

    // Collect PDF data in memory
    const pdfChunks = [];
    doc.on('data', chunk => pdfChunks.push(chunk));
    doc.on('end', () => {
      const pdfBuffer = Buffer.concat(pdfChunks);
      const fileSize = pdfBuffer.length;

      if (fileSize < 10240 || fileSize > 20480) {
        return res.status(400).json({
          success: false,
          message: `PDF size (${Math.round(fileSize/1024)}KB) must be between
10KB and 20KB`
        });
      }

      fs.writeFileSync(filepath, pdfBuffer);
      res.setHeader('Content-Type', 'application/pdf');
      res.setHeader('Content-Disposition', `attachment; filename="${filename}"`);
      res.send(pdfBuffer);
    });

    // Title Page
    doc.font('Helvetica-Bold')
       .fontSize(20)
       .text('CRM SYSTEM', { align: 'center' })
       .fontSize(16)
       .moveDown(0.5)
       .text('Technical Documentation', { align: 'center' })
       .moveDown(2)
       .fontSize(10)
       .text('Traincape Technology', { align: 'center' })
       .moveDown(0.5)
```

```
       .text(new Date().toLocaleDateString(), { align: 'center' });

    doc.addPage();

    // Table of Contents
    doc.fontSize(14)
       .text('Table of Contents', { align: 'left', underline: true })
       .moveDown(1);

    const sections = [
      '1. System Architecture',
      '2. Authentication Flow',
      '3. Employee Management',
      '4. Payroll System',
      '5. Chat System',
      '6. Code Examples',
      '7. API Reference',
      '8. Future Enhancements'
    ];

    sections.forEach((section, i) => {
      doc.fontSize(10)
         .text(section, { link: i + 1 })
         .moveDown(0.5);
    });

    doc.addPage();

    // 1. System Architecture Diagram
    doc.fontSize(14)
       .text('1. System Architecture', { align: 'left', underline: true })
       .moveDown(1);

    // Draw system architecture diagram
    const startY = doc.y;

    // Frontend Box
    drawComponentBox(doc, 50, startY, 150, 60, 'Frontend (React)',
      'Components, Context API, Socket.IO Client, Web Audio');

    // Backend Box
    drawComponentBox(doc, 300, startY, 150, 60, 'Backend (Node.js)',
      'Express, Socket.IO Server, JWT Auth');

    // Database Box
    drawComponentBox(doc, 300, startY + 100, 150, 60, 'MongoDB',
      'Free Tier, Collections: Users, Payroll, Chat');

    // Database Schema
    doc.moveDown(10)
       .fontSize(10)
       .text('Database Schema:', { underline: true })
       .moveDown(1);

    // Users Collection
    drawComponentBox(doc, 50, doc.y, 120, 70, 'Users Collection',
      'email: String\npassword: String (hashed)\nrole: String\nstatus:
String\nprofile: Object\ncreatedAt: Date');
```

```
    // Payroll Collection
    drawComponentBox(doc, 200, doc.y, 120, 70, 'Payroll Collection',
      'employeeId: ObjectId\nmonth: Date\nbaseSalary: Number\ndeductions:
Object\nstatus: String\napprovedBy: ObjectId');

    // Chat Collection
    drawComponentBox(doc, 350, doc.y, 120, 70, 'Chat Collection',
      'sender: ObjectId\nrecipient: ObjectId\nmessage: String\nreadAt:
Date\nattachments: Array\ncreatedAt: Date');

    // Draw connections
    drawArrow(doc, 200, startY + 30, 300, startY + 30, 'API Calls');
    drawArrow(doc, 300, startY + 90, 200, startY + 30, 'Responses');
    drawArrow(doc, 375, startY + 60, 375, startY + 100, 'Queries');

    doc.moveDown(7)
       .fontSize(8)
       .text('System Architecture Diagram showing main components and their
interactions',
         { align: 'center' });

    doc.addPage();

    // 2. Authentication Flow
    doc.fontSize(14)
       .text('2. Authentication Flow', { align: 'left', underline: true })
       .moveDown(1);

    // Draw authentication flow diagram
    const authStartY = doc.y;

    // Login Box
    drawComponentBox(doc, 50, authStartY, 100, 40, 'Login',
      'Email + Password');

    // JWT Generation
    drawComponentBox(doc, 200, authStartY, 100, 40, 'JWT Token',
      'Generation + Validation');

    // Protected Routes
    drawComponentBox(doc, 350, authStartY, 100, 40, 'Protected Routes',
      'Role-based Access');

    // Draw flow arrows
    drawArrow(doc, 150, authStartY + 20, 200, authStartY + 20, 'Verify');
    drawArrow(doc, 300, authStartY + 20, 350, authStartY + 20, 'Allow');

    doc.moveDown(5)
       .fontSize(8)
       .text('Authentication Flow Diagram', { align: 'center' })
       .moveDown(1);

    // Code example for authentication
    doc.fontSize(10)
       .text('Authentication Code Example:', { underline: true })
       .moveDown(0.5)
       .fontSize(7)
       .font('Courier')
       .text(`
```

```
// Frontend Authentication with Error Handling
const login = async (email, password) => {
  try {
    const response = await api.post('/auth/login', {
      email,
      password
    });

    if (response.data.token) {
      localStorage.setItem('token', response.data.token);
      return { success: true };
    }
    return {
      success: false,
      error: 'Invalid credentials'
    };
  } catch (error) {
    console.error('Login error:', error);
    return {
      success: false,
      error: error.response?.data?.message || 'Network error'
    };
  }
};

// Backend JWT Verification
const verifyToken = (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'No token provided' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid token' });
  }
};`, { lineGap: 1 });

    doc.addPage();

    // 3. Employee Management
    doc.fontSize(14)
       .text('3. Employee Management', { align: 'left', underline: true })
       .moveDown(1);

    // Draw employee management diagram
    const empStartY = doc.y;

    // Employee Profile Box
    drawComponentBox(doc, 50, empStartY, 120, 50, 'Employee Profile',
      'Personal Details\nDepartment\nRole');

    // Document Management
    drawComponentBox(doc, 200, empStartY, 120, 50, 'Documents',
      'Upload\nVerification\nStorage');
```

```
    // Attendance System
    drawComponentBox(doc, 350, empStartY, 120, 50, 'Attendance',
      'Daily Status\nReports\nAnalytics');

    // Draw connections
    drawArrow(doc, 170, empStartY + 25, 200, empStartY + 25, 'Upload');
    drawArrow(doc, 320, empStartY + 25, 350, empStartY + 25, 'Track');

    doc.moveDown(5)
       .fontSize(8)
       .text('Employee Management System Components', { align: 'center' });

    doc.addPage();

    // 4. Payroll System
    doc.fontSize(14)
       .text('4. Payroll System', { align: 'left', underline: true })
       .moveDown(1);

    // Draw payroll system diagram
    const payStartY = doc.y;

    // Salary Calculation Box
    drawComponentBox(doc, 50, payStartY, 120, 50, 'Salary Calculation',
      '30 Days Base\nProration\nDeductions');

    // Approval Workflow
    drawComponentBox(doc, 200, payStartY, 120, 50, 'Approval Workflow',
      'Draft !’ Review\n!’ Approved !’ Paid');

    // PDF Generation
    drawComponentBox(doc, 350, payStartY, 120, 50, 'Salary Slip',
      'PDF Generation\n10KB-20KB Size');

    // Draw workflow
    drawArrow(doc, 170, payStartY + 25, 200, payStartY + 25, 'Submit');
    drawArrow(doc, 320, payStartY + 25, 350, payStartY + 25, 'Generate');

    doc.moveDown(5)
       .fontSize(8)
       .text('Payroll System Workflow', { align: 'center' })
       .moveDown(1);

    // Payroll calculation code example
    doc.fontSize(10)
       .text('Salary Calculation Example:', { underline: true })
       .moveDown(0.5)
       .fontSize(7)
       .font('Courier')
       .text(`
const calculateSalary = (baseSalary, daysWorked) => {
  const dailyRate = baseSalary / 30; // 30 days base
  const workedAmount = dailyRate * daysWorked;

  // Apply prorations
  const finalAmount = Math.round(workedAmount * 100) / 100;
  return finalAmount;
};`, { lineGap: 1 });
```

```
    doc.addPage();

    // 5. Chat System
    doc.fontSize(14)
       .text('5. Chat System', { align: 'left', underline: true })
       .moveDown(1);

    // Draw chat system diagram
    const chatStartY = doc.y;

    // Client Socket Box
    drawComponentBox(doc, 50, chatStartY, 120, 50, 'Socket.IO Client',
      'Real-time Events\nNotifications\nSound Effects');

    // Server Socket
    drawComponentBox(doc, 200, chatStartY, 120, 50, 'Socket.IO Server',
      'Event Handling\nMessage Routing\nStatus Updates');

    // Message Storage
    drawComponentBox(doc, 350, chatStartY, 120, 50, 'MongoDB Storage',
      'Chat History\nUser Status\nPreferences');

    // Draw real-time flow
    drawArrow(doc, 170, chatStartY + 25, 200, chatStartY + 25, 'Send');
    drawArrow(doc, 320, chatStartY + 25, 350, chatStartY + 25, 'Store');

    doc.moveDown(5)
       .fontSize(8)
       .text('Real-time Chat System Architecture', { align: 'center' })
       .moveDown(1);

    // Chat system code example
    doc.fontSize(10)
       .text('Chat Implementation Example:', { underline: true })
       .moveDown(0.5)
       .fontSize(7)
       .font('Courier')
       .text(`
// Frontend Socket Setup
const socket = io(SOCKET_URL);

socket.on('connect', () => {
  console.log('Connected to chat server');
});

socket.on('new_message', (message) => {
  if (soundEnabled) {
    playNotificationSound();
  }
  addMessageToChat(message);
});

// Backend Socket Handler
io.on('connection', (socket) => {
  socket.on('send_message', async (data) => {
    const message = await saveMessage(data);
    io.to(data.roomId).emit('new_message', message);
  });
```

```
});`, { lineGap: 1 });

    doc.addPage();

    // 6. API Reference
    doc.fontSize(14)
       .text('6. API Reference', { align: 'left', underline: true })
       .moveDown(1);

    // API documentation with examples
    const apis = [
      {
        endpoint: '/api/auth/login',
        method: 'POST',
        description: 'User authentication with email/password',
        request: `{
  "email": "user@example.com",
  "password": "securepass123"
}`,
        response: `{
  "success": true,
  "token": "eyJhbGciOiJ...",
  "user": {
    "id": "123",
    "email": "user@example.com",
    "role": "Employee"
  }
}`,
        returns: 'JWT token with user details'
      },
      {
        endpoint: '/api/employees',
        method: 'GET',
        description: 'List all employees with pagination',
        request: `?page=1&limit=10&department=IT`,
        response: `{
  "success": true,
  "data": [{
    "id": "123",
    "name": "John Doe",
    "department": "IT",
    "role": "Developer"
  }],
  "pagination": {
    "total": 50,
    "pages": 5,
    "current": 1
  }
}`,
        returns: 'Employee[] with pagination metadata'
      },
      {
        endpoint: '/api/payroll/generate',
        method: 'POST',
        description: 'Generate monthly payroll',
        request: `{
  "employeeId": "123",
  "month": "2024-03",
  "baseSalary": 5000,
```

```
  "deductions": {
    "tax": 500,
    "insurance": 200
  }
}`,
        response: `{
  "success": true,
  "payroll": {
    "id": "456",
    "netSalary": 4300,
    "status": "Draft"
  }
}`,
        returns: 'Payroll details with calculation'
      },
      {
        endpoint: '/api/chat/history',
        method: 'GET',
        description: 'Fetch chat history with pagination',
        request: `?userId=123&page=1&limit=20`,
        response: `{
  "success": true,
  "messages": [{
    "id": "789",
    "sender": "123",
    "message": "Hello!",
    "createdAt": "2024-03-20T10:30:00Z"
  }],
  "pagination": {
    "total": 100,
    "pages": 5
  }
}`,
        returns: 'Message[] with metadata'
      }
    ];

    apis.forEach(api => {
      doc.fontSize(10)
         .font('Helvetica-Bold')
         .text(`${api.method} ${api.endpoint}`)
         .moveDown(0.2)
         .fontSize(8)
         .font('Helvetica')
         .text(`Description: ${api.description}`)
         .text(`Returns: ${api.returns}`)
         .moveDown(0.5);
    });

    // 7. Future Enhancements
    doc.addPage()
       .fontSize(14)
       .text('7. Future Enhancements', { align: 'left', underline: true })
       .moveDown(1)
       .fontSize(10);

    const enhancements = [
      {
        feature: 'Advanced Reporting',
```

```
        description: 'Interactive dashboards with data visualization'
      },
      {
        feature: 'Leave Management',
        description: 'Automated leave request and approval system'
      },
      {
        feature: 'Performance Reviews',
        description: 'KPI tracking and evaluation system'
      },
      {
        feature: 'Mobile Application',
        description: 'Native mobile apps for iOS and Android'
      }
    ];

    enhancements.forEach(item => {
      doc.font('Helvetica-Bold')
         .text(item.feature)
         .font('Helvetica')
         .fontSize(8)
         .text(item.description)
         .moveDown(0.5);
    });

    // Footer
    doc.fontSize(7)
       .text('Generated: ' + new Date().toLocaleString(), { align: 'center' })
       .text('Traincape Technology CRM System', { align: 'center' });

    // Finish PDF
    doc.end();

  } catch (error) {
    console.error('Documentation generation error:', error);
    res.status(500).json({
      success: false,
      message: 'Error generating documentation'
    });
  }
};
```

controllers/employees.js

```
const Employee = require('../models/Employee');
const Department = require('../models/Department');
const Role = require('../models/EmployeeRole');
const User = require('../models/User');
const multer = require('multer');
const path = require('path');
const fs = require('fs');
const { UPLOAD_PATHS } = require('../config/storage');

// Configure multer for file uploads
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, UPLOAD_PATHS.EMPLOYEES);
  },
```

```
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    cb(null, file.fieldname + '-' + uniqueSuffix +
path.extname(file.originalname));
  }
});

const upload = multer({
  storage: storage,
  limits: {
    fileSize: 20 * 1024, // 20KB max
    files: 10 // Maximum 10 files per request
  },
  fileFilter: function (req, file, cb) {
    // Check minimum file size (10KB)
    if (parseInt(req.headers['content-length']) < 10 * 1024) {
      cb(new Error('File size too small. Minimum size is 10KB'), false);
      return;
    }

    // Allow images and PDFs
    if (file.mimetype.startsWith('image/') || file.mimetype === 'application/
pdf') {
      cb(null, true);
    } else {
      cb(new Error('Only images and PDF files are allowed!'), false);
    }
  }
});

// @desc    Get all employees
// @route   GET /api/employees
// @access  Private
exports.getEmployees = async (req, res) => {
  try {
    let query;

    // Copy req.query
    const reqQuery = { ...req.query };

    // Fields to exclude
    const removeFields = ['select', 'sort', 'page', 'limit'];

    // Loop over removeFields and delete them from reqQuery
    removeFields.forEach(param => delete reqQuery[param]);

    // Create query string
    let queryStr = JSON.stringify(reqQuery);

    // Create operators ($gt, $gte, etc)
    queryStr = queryStr.replace(/\b(gt|gte|lt|lte|in)\b/g, match => `$${match}`);

    // Role-based filtering
    if (req.user.role === 'HR') {
      // HR can see employees they manage or all if no hrId restriction
      query = Employee.find(JSON.parse(queryStr));
    } else if (req.user.role === 'Admin' || req.user.role === 'Manager') {
      // Admin and Manager can see all employees
      query = Employee.find(JSON.parse(queryStr));
```

```
  } else {
    // Other users can see all employees (for profile viewing)
    query = Employee.find(JSON.parse(queryStr));
  }

  // Select Fields
  if (req.query.select) {
    const fields = req.query.select.split(',').join(' ');
    query = query.select(fields);
  }

  // Sort
  if (req.query.sort) {
    const sortBy = req.query.sort.split(',').join(' ');
    query = query.sort(sortBy);
  } else {
    query = query.sort('-createdAt');
  }

  // Pagination
  const page = parseInt(req.query.page, 10) || 1;
  const limit = parseInt(req.query.limit, 10) || 25;
  const startIndex = (page - 1) * limit;
  const endIndex = page * limit;
  const total = await Employee.countDocuments();

  query = query.skip(startIndex).limit(limit);

  // Executing query
  const employees = await query;

  // Pagination result
  const pagination = {};

  if (endIndex < total) {
    pagination.next = {
      page: page + 1,
      limit
    };
  }

  if (startIndex > 0) {
    pagination.prev = {
      page: page - 1,
      limit
    };
  }

  res.status(200).json({
    success: true,
    count: employees.length,
    pagination,
    data: employees
  });
} catch (err) {
  console.error('Error fetching employees:', err);
  res.status(500).json({
    success: false,
    message: 'Server Error'
```

```
    });
  }
};

// @desc    Get single employee
// @route   GET /api/employees/:id
// @access  Private
exports.getEmployee = async (req, res) => {
  try {
    const employee = await Employee.findById(req.params.id);

    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee not found'
      });
    }

    // Check authorization - Allow HR, Admin, Manager, and users viewing their
own profile
    if (req.user.role === 'HR' || req.user.role === 'Admin' || req.user.role ===
'Manager' ||
        employee.userId?.toString() === req.user.id) {
      // Authorized
    } else {
      // Allow all users to view employee data for profile purposes
      // This enables the profile page to show employee information
    }

    res.status(200).json({
      success: true,
      data: employee
    });
  } catch (err) {
    console.error('Error fetching employee:', err);
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Create new employee
// @route   POST /api/employees
// @access  Private
exports.createEmployee = async (req, res) => {
  try {
    // Parse employee data from form
    const employeeData = typeof req.body.employee === 'string' ?
JSON.parse(req.body.employee) : req.body;

    // Check if user is trying to create their own profile
    const isCreatingOwnProfile = employeeData.userId === req.user.id ||
                                 !employeeData.userId ||
                                 employeeData.email === req.user.email;

    // Allow users to create their own profiles, but restrict admin functions
    if (!isCreatingOwnProfile && !['HR', 'Admin',
'Manager'].includes(req.user.role)) {
```

```
      return res.status(403).json({
        success: false,
        message: 'Not authorized to create employee profiles for other users'
      });
    }

    // Set the user ID for the employee
    if (isCreatingOwnProfile) {
      employeeData.userId = req.user.id;
      employeeData.fullName = employeeData.fullName || req.user.fullName;
      employeeData.email = employeeData.email || req.user.email;
    }

    // Add HR ID if user is HR
    if (req.user.role === 'HR') {
      employeeData.hrId = req.user.id;
    }

    // Handle file uploads
    if (req.files) {
      Object.keys(req.files).forEach(fieldName => {
        if (req.files[fieldName] && req.files[fieldName][0]) {
          employeeData[fieldName] = req.files[fieldName][0].path;
        }
      });
    }

    // Create employee
    const employee = await Employee.create(employeeData);

    // Create user account if username and password provided (admin function only)
    if (req.body.username && req.body.password && ['HR', 'Admin',
'Manager'].includes(req.user.role)) {
      const userData = {
        fullName: employeeData.fullName,
        email: employeeData.email,
        password: req.body.password,
        role: 'Employee',
        employeeId: employee._id
      };

      const user = await User.create(userData);
      employee.userId = user._id;
      await employee.save();
    }

    res.status(201).json({
      success: true,
      data: employee
    });
  } catch (err) {
    console.error('Error creating employee:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};
```

```
// @desc    Update employee
// @route   PUT /api/employees/:id
// @access  Private
exports.updateEmployee = async (req, res) => {
  try {
    let employee = await Employee.findById(req.params.id);

    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee not found'
      });
    }

    // Check authorization - Allow only HR, Admin, and Manager to update employees
    if (!['HR', 'Admin', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to update employee profiles'
      });
    }

    // Parse employee data from form
    const employeeData = req.body.employee ? JSON.parse(req.body.employee) :
req.body;

    // Handle file uploads
    if (req.files) {
      Object.keys(req.files).forEach(fieldName => {
        if (req.files[fieldName] && req.files[fieldName][0]) {
          // Delete old file if exists
          if (employee[fieldName] && fs.existsSync(employee[fieldName])) {
            fs.unlinkSync(employee[fieldName]);
          }
          employeeData[fieldName] = req.files[fieldName][0].path;
        }
      });
    }

    employee = await Employee.findByIdAndUpdate(req.params.id, employeeData, {
      new: true,
      runValidators: true
    });

    res.status(200).json({
      success: true,
      data: employee
    });
  } catch (err) {
    console.error('Error updating employee:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};

// @desc    Delete employee
// @route   DELETE /api/employees/:id
```

```
// @access  Private
exports.deleteEmployee = async (req, res) => {
  try {
    const employee = await Employee.findById(req.params.id);

    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee not found'
      });
    }

    // Check authorization - Allow HR, Admin, and Manager to delete employees
    if (!['HR', 'Admin', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to delete employees'
      });
    }

    // Delete associated files
    const fileFields = ['photograph', 'tenthMarksheet', 'twelfthMarksheet',
'bachelorDegree',
                        'postgraduateDegree', 'aadharCard', 'panCard', 'pcc',
'resume', 'offerLetter'];

    fileFields.forEach(field => {
      if (employee[field] && fs.existsSync(employee[field])) {
        fs.unlinkSync(employee[field]);
      }
    });

    // Delete associated user account
    if (employee.userId) {
      await User.findByIdAndDelete(employee.userId);
    }

    await Employee.findByIdAndDelete(req.params.id);

    res.status(200).json({
      success: true,
      data: {}
    });
  } catch (err) {
    console.error('Error deleting employee:', err);
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Get all departments
// @route   GET /api/employees/departments
// @access  Private
exports.getDepartments = async (req, res) => {
  try {
    // Find all active departments
    const departments = await Department.find({ isActive: true });
```

```
    // If no departments exist, create a default one
    if (departments.length === 0) {
      const defaultDepartment = await Department.create({
        name: 'General',
        description: 'General Department',
        isActive: true
      });
      departments.push(defaultDepartment);
    }

    res.status(200).json({
      success: true,
      data: departments
    });
  } catch (err) {
    console.error('Error fetching departments:', err);
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Create department
// @route   POST /api/employees/departments
// @access  Private (Admin/Manager only)
exports.createDepartment = async (req, res) => {
  try {
    if (req.user.role !== 'Admin' && req.user.role !== 'Manager') {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to create departments'
      });
    }

    const department = await Department.create(req.body);

    res.status(201).json({
      success: true,
      data: department
    });
  } catch (err) {
    console.error('Error creating department:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};

// @desc    Get all roles
// @route   GET /api/employees/roles
// @access  Private
exports.getRoles = async (req, res) => {
  try {
    // Find all active roles
    const roles = await Role.find({ isActive: true });
```

```
    // If no roles exist, create default ones
    if (roles.length === 0) {
      const defaultRoles = await Role.insertMany([
        {
          name: 'Sales Person',
          description: 'Sales Person Role',
          isActive: true
        },
        {
          name: 'Lead Person',
          description: 'Lead Person Role',
          isActive: true
        },
        {
          name: 'Manager',
          description: 'Manager Role',
          isActive: true
        },
        {
          name: 'Employee',
          description: 'General Employee Role',
          isActive: true
        }
      ]);
      roles.push(...defaultRoles);
    }

    res.status(200).json({
      success: true,
      data: roles
    });
  } catch (err) {
    console.error('Error fetching roles:', err);
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Create role
// @route   POST /api/employees/roles
// @access  Private (Admin/Manager only)
exports.createRole = async (req, res) => {
  try {
    if (req.user.role !== 'Admin' && req.user.role !== 'Manager') {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to create roles'
      });
    }

    const role = await Role.create(req.body);

    res.status(201).json({
      success: true,
      data: role
    });
  } catch (err) {
```

```
    console.error('Error creating role:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};

// Export multer upload middleware
exports.uploadEmployeeFiles = upload.fields([
  { name: 'photograph', maxCount: 1 },
  { name: 'tenthMarksheet', maxCount: 1 },
  { name: 'twelfthMarksheet', maxCount: 1 },
  { name: 'bachelorDegree', maxCount: 1 },
  { name: 'postgraduateDegree', maxCount: 1 },
  { name: 'aadharCard', maxCount: 1 },
  { name: 'panCard', maxCount: 1 },
  { name: 'pcc', maxCount: 1 },
  { name: 'resume', maxCount: 1 },
  { name: 'offerLetter', maxCount: 1 }
]);

// @desc    Get employee document
// @route   GET /api/employees/documents/:filename
// @access  Private
exports.getDocument = async (req, res) => {
  try {
    const { filename } = req.params;

    // Validate filename
    if (!filename) {
      return res.status(400).json({
        success: false,
        message: 'Filename is required'
      });
    }

    // Construct possible file paths (check both direct path and nested path)
    const possiblePaths = [
      path.join(UPLOAD_PATHS.EMPLOYEES, filename),
      path.join(UPLOAD_PATHS.DOCUMENTS, filename),
      path.join(UPLOAD_PATHS.DOCUMENTS, 'employees', filename)
    ];

    // Find the first path that exists
    let filePath = null;
    for (const path of possiblePaths) {
      if (fs.existsSync(path)) {
        filePath = path;
        break;
      }
    }

    // If no file found in any location
    if (!filePath) {
      console.error('Document not found in paths:', possiblePaths);
      return res.status(404).json({
        success: false,
        message: 'Document not found'
```

```
      });
    }

    // Get file stats
    const stats = fs.statSync(filePath);
    if (!stats.isFile()) {
      return res.status(404).json({
        success: false,
        message: 'Document not found'
      });
    }

    // Set appropriate headers
    const ext = path.extname(filename).toLowerCase();
    const contentType = ext === '.pdf' ? 'application/pdf' :
                        ext === '.jpg' || ext === '.jpeg' ? 'image/jpeg' :
                        ext === '.png' ? 'image/png' :
                        'application/octet-stream';

    res.setHeader('Content-Type', contentType);
    res.setHeader('Content-Length', stats.size);
    res.setHeader('Content-Disposition', `inline; filename="${filename}"`);

    // Stream the file
    const fileStream = fs.createReadStream(filePath);
    fileStream.pipe(res);

    // Handle stream errors
    fileStream.on('error', (err) => {
      console.error('Error streaming file:', err);
      if (!res.headersSent) {
        res.status(500).json({
          success: false,
          message: 'Error streaming document'
        });
      }
    });
  } catch (err) {
    console.error('Error serving document:', err);
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};
```

controllers/incentives.js

```
const Incentive = require('../models/Incentive');
const Employee = require('../models/Employee');
const User = require('../models/User');
const multer = require('multer');
const path = require('path');
const fs = require('fs');
const { UPLOAD_PATHS } = require('../config/storage');

// Configure multer for file uploads
const storage = multer.diskStorage({
```

```
  destination: function (req, file, cb) {
    cb(null, UPLOAD_PATHS.INCENTIVES);
  },
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    cb(null, 'incentive-' + uniqueSuffix + path.extname(file.originalname));
  }
});

const upload = multer({
  storage: storage,
  limits: {
    fileSize: 5 * 1024 * 1024 // 5MB limit
  },
  fileFilter: function (req, file, cb) {
    // Allow documents and images
    const allowedMimes = ['application/pdf', 'image/jpeg', 'image/jpg', 'image/
png', 'application/msword', 'application/vnd.openxmlformats-
officedocument.wordprocessingml.document'];
    if (allowedMimes.includes(file.mimetype)) {
      cb(null, true);
    } else {
      cb(new Error('Only PDF, DOC, DOCX, and image files are allowed!'), false);
    }
  }
});

// @desc    Create new incentive
// @route   POST /api/incentives
// @access  Private (Admin/HR/Manager)
exports.createIncentive = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to create incentives'
      });
    }

    const {
      employeeId, type, title, description, amount,
      performanceRating, performancePeriod, projectName,
      projectCompletionDate, attendancePercentage, attendancePeriod,
      festivalType, validFrom, validTo, isRecurring, recurringType
    } = req.body;

    // Validate required fields
    if (!employeeId || !type || !title || !description || !amount) {
      return res.status(400).json({
        success: false,
        message: 'Employee ID, type, title, description, and amount are required'
      });
    }

    // Get employee details
    const employee = await Employee.findById(employeeId);
    if (!employee) {
      return res.status(404).json({
```

```
    success: false,
    message: 'Employee not found'
  });
}

// Prepare incentive data
const incentiveData = {
  employeeId,
  userId: employee.userId,
  type,
  title,
  description,
  amount: parseFloat(amount),
  requestedBy: req.user.id
};

// Add type-specific fields
if (type === 'PERFORMANCE') {
  incentiveData.performanceRating = performanceRating;
  if (performancePeriod) {
    incentiveData.performancePeriod = JSON.parse(performancePeriod);
  }
}

if (type === 'PROJECT') {
  incentiveData.projectName = projectName;
  if (projectCompletionDate) {
    incentiveData.projectCompletionDate = new Date(projectCompletionDate);
  }
}

if (type === 'ATTENDANCE') {
  incentiveData.attendancePercentage = attendancePercentage;
  if (attendancePeriod) {
    incentiveData.attendancePeriod = JSON.parse(attendancePeriod);
  }
}

if (type === 'FESTIVAL') {
  incentiveData.festivalType = festivalType;
}

// Add validity period
if (validFrom) incentiveData.validFrom = new Date(validFrom);
if (validTo) incentiveData.validTo = new Date(validTo);

// Add recurring info
if (isRecurring === 'true') {
  incentiveData.isRecurring = true;
  incentiveData.recurringType = recurringType;
}

// Handle file attachments
if (req.files && req.files.length > 0) {
  incentiveData.attachments = req.files.map(file => ({
    filename: file.filename,
    originalName: file.originalname,
    path: file.path,
    size: file.size,
```

```
        uploadedBy: req.user.id
      }));
    }

    // Create incentive
    const incentive = await Incentive.create(incentiveData);

    // Populate the created incentive
    const populatedIncentive = await Incentive.findById(incentive._id)
      .populate('employeeId', 'fullName email department')
      .populate('userId', 'fullName email')
      .populate('requestedBy', 'fullName email');

    res.status(201).json({
      success: true,
      data: populatedIncentive,
      message: 'Incentive created successfully'
    });
  } catch (error) {
    console.error('Create incentive error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error during incentive creation'
    });
  }
};

// @desc    Get incentives
// @route   GET /api/incentives
// @access  Private
exports.getIncentives = async (req, res) => {
  try {
    const { employeeId, type, status, page = 1, limit = 20 } = req.query;

    // Build query based on user role
    let query = {};

    if (req.user.role === 'Employee') {
      // Employees can only see their own incentives
      const employee = await Employee.findOne({ userId: req.user.id });
      if (!employee) {
        return res.status(404).json({
          success: false,
          message: 'Employee record not found'
        });
      }
      query.employeeId = employee._id;
    } else if (['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      // Admin/HR/Manager can see all or filter by employee
      if (employeeId) {
        query.employeeId = employeeId;
      }
    } else {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to view incentives'
      });
    }
```

```
    // Add filters
    if (type) query.type = type;
    if (status) query.status = status;

    // Get incentives with pagination
    const skip = (page - 1) * limit;
    const incentives = await Incentive.find(query)
      .populate('employeeId', 'fullName email department')
      .populate('userId', 'fullName email')
      .populate('requestedBy', 'fullName email')
      .populate('approvedBy', 'fullName email')
      .sort({ createdAt: -1 })
      .skip(skip)
      .limit(parseInt(limit));

    // Get total count
    const total = await Incentive.countDocuments(query);

    res.status(200).json({
      success: true,
      data: incentives,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total: total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    console.error('Get incentives error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Get single incentive
// @route   GET /api/incentives/:id
// @access  Private
exports.getIncentive = async (req, res) => {
  try {
    const incentive = await Incentive.findById(req.params.id)
      .populate('employeeId', 'fullName email department')
      .populate('userId', 'fullName email')
      .populate('requestedBy', 'fullName email')
      .populate('approvedBy', 'fullName email')
      .populate('comments.commentBy', 'fullName email');

    if (!incentive) {
      return res.status(404).json({
        success: false,
        message: 'Incentive not found'
      });
    }

    // Check authorization
    if (req.user.role === 'Employee') {
      const employee = await Employee.findOne({ userId: req.user.id });
```

```
      if (!employee || employee._id.toString() !==
incentive.employeeId._id.toString()) {
        return res.status(403).json({
          success: false,
          message: 'Not authorized to view this incentive'
        });
      }
    } else if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to view incentive'
      });
    }

    res.status(200).json({
      success: true,
      data: incentive
    });
  } catch (error) {
    console.error('Get incentive error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Update incentive
// @route   PUT /api/incentives/:id
// @access  Private (Admin/HR/Manager)
exports.updateIncentive = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to update incentives'
      });
    }

    const incentive = await Incentive.findById(req.params.id);
    if (!incentive) {
      return res.status(404).json({
        success: false,
        message: 'Incentive not found'
      });
    }

    // Update allowed fields
    const allowedFields = [
      'title', 'description', 'amount', 'performanceRating',
      'performancePeriod', 'projectName', 'projectCompletionDate',
      'attendancePercentage', 'attendancePeriod', 'festivalType',
      'validFrom', 'validTo', 'isRecurring', 'recurringType'
    ];

    allowedFields.forEach(field => {
      if (req.body[field] !== undefined) {
        if (field === 'amount') {
```

```
          incentive[field] = parseFloat(req.body[field]);
        } else if (field === 'performancePeriod' || field === 'attendancePeriod')
{
          incentive[field] = JSON.parse(req.body[field]);
        } else if (field === 'validFrom' || field === 'validTo' || field ===
'projectCompletionDate') {
          incentive[field] = new Date(req.body[field]);
        } else if (field === 'isRecurring') {
          incentive[field] = req.body[field] === 'true';
        } else {
          incentive[field] = req.body[field];
        }
      }
    });

    await incentive.save();

    const updatedIncentive = await Incentive.findById(incentive._id)
      .populate('employeeId', 'fullName email department')
      .populate('userId', 'fullName email')
      .populate('requestedBy', 'fullName email')
      .populate('approvedBy', 'fullName email');

    res.status(200).json({
      success: true,
      data: updatedIncentive,
      message: 'Incentive updated successfully'
    });
  } catch (error) {
    console.error('Update incentive error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Approve incentive
// @route   PUT /api/incentives/:id/approve
// @access  Private (Admin/HR/Manager)
exports.approveIncentive = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to approve incentives'
      });
    }

    const incentive = await Incentive.findById(req.params.id);
    if (!incentive) {
      return res.status(404).json({
        success: false,
        message: 'Incentive not found'
      });
    }

    incentive.status = 'APPROVED';
```

```
    incentive.approvedBy = req.user.id;
    incentive.approvedDate = new Date();

    await incentive.save();

    const updatedIncentive = await Incentive.findById(incentive._id)
      .populate('employeeId', 'fullName email department')
      .populate('userId', 'fullName email')
      .populate('requestedBy', 'fullName email')
      .populate('approvedBy', 'fullName email');

    res.status(200).json({
      success: true,
      data: updatedIncentive,
      message: 'Incentive approved successfully'
    });
  } catch (error) {
    console.error('Approve incentive error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Reject incentive
// @route   PUT /api/incentives/:id/reject
// @access  Private (Admin/HR/Manager)
exports.rejectIncentive = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to reject incentives'
      });
    }

    const { reason } = req.body;

    const incentive = await Incentive.findById(req.params.id);
    if (!incentive) {
      return res.status(404).json({
        success: false,
        message: 'Incentive not found'
      });
    }

    incentive.status = 'REJECTED';
    incentive.rejectedReason = reason;
    incentive.approvedBy = req.user.id;
    incentive.approvedDate = new Date();

    await incentive.save();

    const updatedIncentive = await Incentive.findById(incentive._id)
      .populate('employeeId', 'fullName email department')
      .populate('userId', 'fullName email')
      .populate('requestedBy', 'fullName email')
```

```
      .populate('approvedBy', 'fullName email');

    res.status(200).json({
      success: true,
      data: updatedIncentive,
      message: 'Incentive rejected successfully'
    });
  } catch (error) {
    console.error('Reject incentive error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Add comment to incentive
// @route   POST /api/incentives/:id/comments
// @access  Private
exports.addComment = async (req, res) => {
  try {
    const { comment } = req.body;

    if (!comment) {
      return res.status(400).json({
        success: false,
        message: 'Comment is required'
      });
    }

    const incentive = await Incentive.findById(req.params.id);
    if (!incentive) {
      return res.status(404).json({
        success: false,
        message: 'Incentive not found'
      });
    }

    // Check authorization
    if (req.user.role === 'Employee') {
      const employee = await Employee.findOne({ userId: req.user.id });
      if (!employee || employee._id.toString() !==
incentive.employeeId.toString()) {
        return res.status(403).json({
          success: false,
          message: 'Not authorized to comment on this incentive'
        });
      }
    } else if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to add comments'
      });
    }

    incentive.addComment(req.user.id, comment);
    await incentive.save();

    const updatedIncentive = await Incentive.findById(incentive._id)
```

```
      .populate('employeeId', 'fullName email department')
      .populate('userId', 'fullName email')
      .populate('requestedBy', 'fullName email')
      .populate('approvedBy', 'fullName email')
      .populate('comments.commentBy', 'fullName email');

    res.status(200).json({
      success: true,
      data: updatedIncentive,
      message: 'Comment added successfully'
    });
  } catch (error) {
    console.error('Add comment error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Delete incentive
// @route   DELETE /api/incentives/:id
// @access  Private (Admin/HR/Manager)
exports.deleteIncentive = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to delete incentives'
      });
    }

    const incentive = await Incentive.findById(req.params.id);
    if (!incentive) {
      return res.status(404).json({
        success: false,
        message: 'Incentive not found'
      });
    }

    // Delete associated files
    if (incentive.attachments && incentive.attachments.length > 0) {
      incentive.attachments.forEach(attachment => {
        if (fs.existsSync(attachment.path)) {
          fs.unlinkSync(attachment.path);
        }
      });
    }

    await Incentive.findByIdAndDelete(req.params.id);

    res.status(200).json({
      success: true,
      message: 'Incentive deleted successfully'
    });
  } catch (error) {
    console.error('Delete incentive error:', error);
    res.status(500).json({
```

```
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Get incentive statistics
// @route   GET /api/incentives/stats
// @access  Private (Admin/HR/Manager)
exports.getIncentiveStats = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to view incentive statistics'
      });
    }

    const { year } = req.query;
    const currentYear = year || new Date().getFullYear();

    // Get statistics
    const stats = await Incentive.aggregate([
      {
        $match: {
          createdAt: {
            $gte: new Date(`${currentYear}-01-01`),
            $lte: new Date(`${currentYear}-12-31`)
          }
        }
      },
      {
        $group: {
          _id: '$type',
          count: { $sum: 1 },
          totalAmount: { $sum: '$amount' },
          approved: {
            $sum: {
              $cond: [{ $eq: ['$status', 'APPROVED'] }, 1, 0]
            }
          },
          pending: {
            $sum: {
              $cond: [{ $eq: ['$status', 'PENDING'] }, 1, 0]
            }
          },
          rejected: {
            $sum: {
              $cond: [{ $eq: ['$status', 'REJECTED'] }, 1, 0]
            }
          }
        }
      }
    ]);

    // Get monthly breakdown
    const monthlyStats = await Incentive.aggregate([
      {
```

```
        $match: {
          createdAt: {
            $gte: new Date(`${currentYear}-01-01`),
            $lte: new Date(`${currentYear}-12-31`)
          }
        }
      },
      {
        $group: {
          _id: { $month: '$createdAt' },
          count: { $sum: 1 },
          totalAmount: { $sum: '$amount' }
        }
      },
      {
        $sort: { '_id': 1 }
      }
    ]);

    res.status(200).json({
      success: true,
      data: {
        byType: stats,
        byMonth: monthlyStats,
        year: currentYear
      }
    });
  } catch (error) {
    console.error('Get incentive stats error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// Export multer upload middleware
exports.uploadIncentiveFiles = upload.array('attachments', 5);
```

controllers/leadPersonSales.js

```
const LeadPersonSale = require('../models/LeadPersonSale');

// @desc    Get all lead person sales
// @route   GET /api/lead-person-sales
// @access  Private (Lead Person, Manager, Admin)
exports.getLeadPersonSales = async (req, res) => {
  try {
    let query;

    // Copy req.query
    const reqQuery = { ...req.query };

    // Fields to exclude
    const removeFields = ['select', 'sort', 'page', 'limit'];

    // Loop over removeFields and delete them from reqQuery
    removeFields.forEach(param => delete reqQuery[param]);
```

```
    // Create query string
    let queryStr = JSON.stringify(reqQuery);

    // Create operators ($gt, $gte, etc)
    queryStr = queryStr.replace(/\b(gt|gte|lt|lte|in)\b/g, match => `$${match}`);

    // If user is a lead person, only show their sales
    if (req.user.role === 'Lead Person') {
      query = LeadPersonSale.find({ leadPerson:
req.user.id, ...JSON.parse(queryStr) });
    }
    // Admin and Manager can see all lead person sales
    else if (['Admin', 'Manager'].includes(req.user.role)) {
      query = LeadPersonSale.find(JSON.parse(queryStr));
    } else {
      // Other roles cannot access these sales
      return res.status(403).json({
        success: false,
        message: 'Not authorized to access lead person sales'
      });
    }

    // Select Fields
    if (req.query.select) {
      const fields = req.query.select.split(',').join(' ');
      query = query.select(fields);
    }

    // Sort
    if (req.query.sort) {
      const sortBy = req.query.sort.split(',').join(' ');
      query = query.sort(sortBy);
    } else {
      query = query.sort('-date');
    }

    // Populate
    query = query.populate('salesPerson leadPerson', 'fullName email');

    // Pagination
    const page = parseInt(req.query.page, 10) || 1;
    const limit = parseInt(req.query.limit, 10) || 25;
    const startIndex = (page - 1) * limit;
    const endIndex = page * limit;
    const total = await LeadPersonSale.countDocuments();

    query = query.skip(startIndex).limit(limit);

    // Executing query
    const sales = await query;

    // Pagination result
    const pagination = {};

    if (endIndex < total) {
      pagination.next = {
        page: page + 1,
        limit
```

```
      };
    }

    if (startIndex > 0) {
      pagination.prev = {
        page: page - 1,
        limit
      };
    }

    res.status(200).json({
      success: true,
      count: sales.length,
      pagination,
      data: sales
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Get single lead person sale
// @route   GET /api/lead-person-sales/:id
// @access  Private (Lead Person, Manager, Admin)
exports.getLeadPersonSale = async (req, res) => {
  try {
    const sale = await LeadPersonSale.findById(req.params.id)
      .populate({
        path: 'salesPerson leadPerson',
        select: 'fullName email'
      });

    if (!sale) {
      return res.status(404).json({
        success: false,
        message: `No lead person sale found with id of ${req.params.id}`
      });
    }

    // Make sure user can access this sale
    if (req.user.role === 'Lead Person' && sale.leadPerson._id.toString() !==
req.user.id) {
      return res.status(403).json({
        success: false,
        message: `User ${req.user.id} is not authorized to access this sale`
      });
    }

    res.status(200).json({
      success: true,
      data: sale
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({
```

```
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Create new lead person sale
// @route   POST /api/lead-person-sales
// @access  Private (Lead Person, Manager, Admin)
exports.createLeadPersonSale = async (req, res) => {
  try {
    // Add user to req.body as creator
    req.body.createdBy = req.user.id;

    console.log('Create lead person sale request from user:', {
      id: req.user.id,
      role: req.user.role,
      body: req.body
    });

    // If user is lead person, set them as the lead person
    if (req.user.role === 'Lead Person') {
      req.body.leadPerson = req.user.id;
    }

    // Make sure leadPerson is set
    if (!req.body.leadPerson) {
      return res.status(400).json({
        success: false,
        message: 'Lead person is required'
      });
    }

    // Make sure salesPerson is set
    if (!req.body.salesPerson) {
      return res.status(400).json({
        success: false,
        message: 'Sales person is required'
      });
    }

    console.log('Creating lead person sale with data:', req.body);

    // Create lead person sale
    const sale = await LeadPersonSale.create(req.body);

    console.log('Lead person sale created successfully:', sale._id);

    res.status(201).json({
      success: true,
      data: sale
    });
  } catch (err) {
    console.error('Error creating lead person sale:', err);

    if (err.name === 'ValidationError') {
      const messages = Object.values(err.errors).map(val => val.message);
      return res.status(400).json({
        success: false,
```

```
        message: messages.join(', ')
      });
    }

    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Update lead person sale
// @route   PUT /api/lead-person-sales/:id
// @access  Private (Lead Person, Manager, Admin)
exports.updateLeadPersonSale = async (req, res) => {
  try {
    let sale = await LeadPersonSale.findById(req.params.id);

    if (!sale) {
      return res.status(404).json({
        success: false,
        message: `No lead person sale found with id of ${req.params.id}`
      });
    }

    // Make sure user is authorized to update this sale
    if (req.user.role === 'Lead Person' && sale.leadPerson.toString() !==
req.user.id) {
      return res.status(403).json({
        success: false,
        message: `User ${req.user.id} is not authorized to update this sale`
      });
    }

    // Add user to req.body as updater
    req.body.updatedBy = req.user.id;

    // Update sale
    sale = await LeadPersonSale.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
      runValidators: true
    });

    res.status(200).json({
      success: true,
      data: sale
    });
  } catch (err) {
    console.error(err);

    if (err.name === 'ValidationError') {
      const messages = Object.values(err.errors).map(val => val.message);
      return res.status(400).json({
        success: false,
        message: messages.join(', ')
      });
    }

    res.status(500).json({
```

```
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Delete lead person sale
// @route   DELETE /api/lead-person-sales/:id
// @access  Private (Lead Person, Manager, Admin)
exports.deleteLeadPersonSale = async (req, res) => {
  try {
    const sale = await LeadPersonSale.findById(req.params.id);

    if (!sale) {
      return res.status(404).json({
        success: false,
        message: `No lead person sale found with id of ${req.params.id}`
      });
    }

    // Make sure user is authorized to delete this sale
    if (req.user.role === 'Lead Person' && sale.leadPerson.toString() !==
req.user.id) {
      return res.status(403).json({
        success: false,
        message: `User ${req.user.id} is not authorized to delete this sale`
      });
    }

    await LeadPersonSale.findByIdAndDelete(req.params.id);

    res.status(200).json({
      success: true,
      data: {}
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};
```

controllers/leads.js

```
const Lead = require('../models/Lead');

// @desc    Get all leads
// @route   GET /api/leads
// @access  Private
exports.getLeads = async (req, res) => {
  try {
    console.log('============= GET LEADS REQUEST =============');
    console.log('User making request:', {
      id: req.user._id,
      idString: req.user._id.toString(),
      role: req.user.role,
```

```
      name: req.user.fullName,
      email: req.user.email
    });
    console.log('Query parameters:', req.query);

    // Extract date filtering parameters from query
    const { month, year, startDate, endDate } = req.query;

    // Instead of using complex filtering, let's use direct MongoDB queries
    let query;

    // Different queries based on role
    if (req.user.role === 'Admin' || req.user.role === 'Manager') {
      // Admin and Manager see all leads
      console.log('Admin/Manager role - fetching ALL leads');
      query = Lead.find({});
    }
    else if (req.user.role === 'Lead Person') {
      // Lead Person sees leads they created or leads assigned to them
      console.log('Lead Person role - fetching created or assigned leads');
      const userId = req.user._id;

      // Query with direct ID comparison
      query = Lead.find({
        $or: [
          { leadPerson: userId },
          { assignedTo: userId }
        ]
      });
    }
    else {
      // Sales Person sees only leads assigned to them
      console.log('Sales Person role - fetching assigned leads');
      const userId = req.user._id;

      // Query for assignedTo exact match
      query = Lead.find({ assignedTo: userId });
    }

    // Add date filtering if provided
    if (month && year) {
      console.log(`Filtering by month: ${month}, year: ${year}`);
      const startOfMonth = new Date(parseInt(year), parseInt(month) - 1, 1);
      const endOfMonth = new Date(parseInt(year), parseInt(month), 0, 23, 59, 59,
999);

      query = query.where('createdAt').gte(startOfMonth).lte(endOfMonth);
      console.log(`Date range: ${startOfMonth.toISOString()} to
${endOfMonth.toISOString()}`);
    } else if (startDate && endDate) {
      console.log(`Filtering by date range: ${startDate} to ${endDate}`);
      const start = new Date(startDate);
      const end = new Date(endDate);
      end.setHours(23, 59, 59, 999); // Include the entire end date

      query = query.where('createdAt').gte(start).lte(end);
      console.log(`Date range: ${start.toISOString()} to ${end.toISOString()}`);
    } else if (startDate) {
      console.log(`Filtering from date: ${startDate}`);
```

```
      const start = new Date(startDate);
      query = query.where('createdAt').gte(start);
    } else if (endDate) {
      console.log(`Filtering until date: ${endDate}`);
      const end = new Date(endDate);
      end.setHours(23, 59, 59, 999);
      query = query.where('createdAt').lte(end);
    }

    // Sort by created date, newest first
    query = query.sort({ createdAt: -1 });

    // Populate relevant fields
    query = query.populate('assignedTo', 'fullName email role')
                 .populate('leadPerson', 'fullName email role');

    // Execute the query
    const leads = await query;

    console.log(`Found ${leads.length} leads for this user`);

    // Log lead details for debugging (limit to first 5 for readability)
    if (leads.length > 0) {
      console.log('Lead details (first 5):');
      leads.slice(0, 5).forEach(lead => {
        console.log(`- Lead ID: ${lead._id}, Name: ${lead.name}, Date:
${lead.createdAt}`);
        console.log(`  Assigned to: ${lead.assignedTo ? lead.assignedTo.fullName
+ ' (ID: ' + lead.assignedTo._id + ')' : 'None'}`);
      });

      if (leads.length > 5) {
        console.log(`... and ${leads.length - 5} more leads`);
      }
    } else {
      console.log('No leads found for this user with the given filters');
    }

    console.log('============================================');

    res.status(200).json({
      success: true,
      count: leads.length,
      data: leads,
      filters: {
        month: month || null,
        year: year || null,
        startDate: startDate || null,
        endDate: endDate || null
      }
    });
  } catch (err) {
    console.error('Error in getLeads:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};
```

```
// @desc    Get single lead
// @route   GET /api/leads/:id
// @access  Private
exports.getLead = async (req, res) => {
  try {
    const lead = await Lead.findById(req.params.id)
      .populate('assignedTo', 'fullName email')
      .populate('leadPerson', 'fullName email');

    console.log('Lead found:', lead ? lead._id : 'None');

    if (!lead) {
      return res.status(404).json({
        success: false,
        message: `No lead found with id of ${req.params.id}`
      });
    }

    // Show detailed info for debugging
    console.log('Lead details:');
    console.log(`  - ID: ${lead._id}`);
    console.log(`  - Name: ${lead.name}`);
    console.log(`  - AssignedTo ID: ${lead.assignedTo ? lead.assignedTo._id :
'None'}`);
    console.log(`  - AssignedTo Name: ${lead.assignedTo ?
lead.assignedTo.fullName : 'None'}`);
    console.log(`  - Current User ID: ${req.user._id}`);

    const leadAssignedId = lead.assignedTo ? lead.assignedTo._id.toString() :
null;
    const userId = req.user._id.toString();
    console.log(`  - String comparison: ${leadAssignedId === userId}`);

    // Check if user is authorized to view this lead
    if (
      req.user.role !== 'Admin' &&
      req.user.role !== 'Manager' &&
      lead.assignedTo.toString() !== req.user._id.toString() &&
      !(req.user.role === 'Lead Person' &&
        lead.leadPerson &&
        lead.leadPerson._id.toString() === req.user._id.toString())
    ) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to access this lead'
      });
    }

    res.status(200).json({
      success: true,
      data: lead
    });
  } catch (err) {
    console.error('Error in getLead:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
```

```
  }
};

// @desc    Create new lead
// @route   POST /api/leads
// @access  Private
exports.createLead = async (req, res) => {
  try {
    console.log('============= CREATE LEAD REQUEST =============');
    console.log('Lead data submitted:', req.body);
    console.log('User creating lead:', {
      id: req.user._id,
      role: req.user.role,
      name: req.user.fullName
    });

    // Validate required fields before processing
    const requiredFields = ['NAME', 'COURSE', 'CODE', 'NUMBER', 'COUNTRY', 'SALE
PERSON'];
    const missingFields = requiredFields.filter(field => !req.body[field]);

    if (missingFields.length > 0) {
      console.error('Missing required fields:', missingFields);
      return res.status(400).json({
        success: false,
        message: `Missing required fields: ${missingFields.join(', ')}`,
        missingFields: missingFields
      });
    }

    // Map the new field names to the database model field names
    const leadData = {
      name: req.body.NAME,
      email: req.body['E-MAIL'] || '',
      course: req.body.COURSE,
      countryCode: req.body.CODE,
      phone: req.body.NUMBER,
      country: req.body.COUNTRY,
      pseudoId: req.body['PSUDO ID'],
      client: req.body['CLIENT REMARK'],
      status: req.body.status || 'Introduction',
      source: req.body.SOURSE,
      sourceLink: req.body['SOURCE LINK'],
      assignedTo: req.body['SALE PERSON'],
      leadPerson: req.body['LEAD PERSON'],
      feedback: req.body.FEEDBACK,
      createdAt: req.body.DATE ? new Date(req.body.DATE) : Date.now(),
      isRepeatCustomer: false, // Default value, will be updated if needed
      previousCourses: [],
      relatedLeads: []
    };

    // Set createdBy to the current user
    leadData.createdBy = req.user._id;

    // If the user is a Lead Person, set them as the leadPerson
    if (req.user.role === 'Lead Person') {
      leadData.leadPerson = req.user._id;
      console.log('Setting leadPerson to current user', req.user._id);
```

```
    }

    // Critical: Make sure assignedTo is properly set
    if (!leadData.assignedTo || leadData.assignedTo === '') {
      console.log('No assignedTo provided, using current user', req.user._id);
      leadData.assignedTo = req.user._id;
    } else {
      console.log('Using provided assignedTo:', leadData.assignedTo);
      // ObjectId is handled properly by Mongoose, no need to convert
    }

    // Check if this is a repeat customer by phone number or email
    let previousLeads = [];
    let isRepeatCustomer = false;

    // Only check if either phone or email is provided
    if (leadData.phone || (leadData.email && leadData.email.trim() !== '')) {
      // Build the query to find potential matches
      const matchQuery = { $or: [] };

      // Add phone number condition if provided
      if (leadData.phone) {
        matchQuery.$or.push({ phone: leadData.phone });
      }

      // Add email condition if provided and not empty
      if (leadData.email && leadData.email.trim() !== '') {
        matchQuery.$or.push({ email: leadData.email });
      }

      // Only run the query if we have conditions
      if (matchQuery.$or.length > 0) {
        console.log('Checking for repeat customer with query:', matchQuery);
        previousLeads = await Lead.find(matchQuery).select('_id name course
createdAt');

        // If we found previous leads with the same email or phone
        if (previousLeads.length > 0) {
          isRepeatCustomer = true;

          // Extract previous courses from the found leads
          const previousCourses = previousLeads
            .map(lead => lead.course)
            .filter(course => course !== leadData.course); // Exclude current
course

          // Extract the IDs of related leads
          const relatedLeadIds = previousLeads.map(lead => lead._id);

          // Update the lead data
          leadData.isRepeatCustomer = true;
          leadData.previousCourses = [...new Set(previousCourses)]; // Remove
duplicates
          leadData.relatedLeads = relatedLeadIds;

          console.log(`This is a repeat customer! Found ${previousLeads.length}
previous leads`);
          console.log('Previous courses:', leadData.previousCourses);
        }
```

```
      }
    }

    // Make sure creation timestamp is set
    leadData.updatedAt = Date.now();

    console.log('Final lead data before creation:', leadData);

    const lead = await Lead.create(leadData);

    // Verify the created lead
    const createdLead = await
Lead.findById(lead._id).populate('assignedTo').populate('createdBy');
    console.log('Created lead successfully:', {
      id: createdLead._id,
      name: createdLead.name,
      assignedTo: createdLead.assignedTo ? {
        id: createdLead.assignedTo._id,
        name: createdLead.assignedTo.fullName
      } : 'None',
      createdBy: createdLead.createdBy ? {
        id: createdLead.createdBy._id,
        name: createdLead.createdBy.fullName
      } : 'None'
    });
    console.log('============================================');

    res.status(201).json({
      success: true,
      data: lead
    });
  } catch (err) {
    console.error('Error creating lead:', err);

    // Handle duplicate key errors (commonly for email)
    if (err.code === 11000 && err.keyPattern) {
      // Get the field name causing the duplicate
      const field = Object.keys(err.keyPattern)[0];
      const value = err.keyValue[field];

      console.error(`Duplicate key error for field: ${field}, value: ${value}`);

      // Instead of returning an error, we'll allow duplicates for repeat
customers
      console.log('Allowing duplicate value for repeat customer');

      // Try creating the lead again without the unique constraint
      try {
        // Modify the data to work around the duplicate key issue
        if (field === 'email') {
          // For email duplicates, proceed with creating the lead
          // The email validation in the model already allows duplicates
          const leadData = {
            name: req.body.NAME,
            email: '',  // Set empty email to avoid duplicate
            course: req.body.COURSE,
            countryCode: req.body.CODE,
            phone: req.body.NUMBER,
            country: req.body.COUNTRY,
```

```
            pseudoId: req.body['PSUDO ID'],
            client: req.body['CLIENT REMARK'],
            status: req.body.status || 'Introduction',
            source: req.body.SOURSE,
            sourceLink: req.body['SOURCE LINK'],
            assignedTo: req.body['SALE PERSON'],
            leadPerson: req.body['LEAD PERSON'],
            feedback: req.body.FEEDBACK,
            createdAt: req.body.DATE ? new Date(req.body.DATE) : Date.now(),
            isRepeatCustomer: true,
            previousCourses: [],
            relatedLeads: [],
            createdBy: req.user._id,
            updatedAt: Date.now()
          };

          // Create the lead with the modified data
          const lead = await Lead.create(leadData);

          return res.status(201).json({
            success: true,
            data: lead
          });
        }
      } catch (retryErr) {
        console.error('Error on retry after duplicate key:', retryErr);
        return res.status(400).json({
          success: false,
          message: 'Failed to create lead even after handling duplicate key'
        });
      }
    }

    // Provide more detailed error messages for common validation errors
    if (err.name === 'ValidationError') {
      const validationErrors = Object.keys(err.errors).map(field => ({
        field: field,
        message: err.errors[field].message
      }));

      return res.status(400).json({
        success: false,
        message: 'Validation failed: ' + validationErrors.map(e =>
e.message).join(', '),
        errors: validationErrors
      });
    }

    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};

// @desc    Update lead
// @route   PUT /api/leads/:id
// @access  Private
exports.updateLead = async (req, res) => {
```

```
try {
  console.log('============= UPDATE LEAD REQUEST =============');
  console.log('User updating lead:', {
    id: req.user._id,
    role: req.user.role,
    name: req.user.fullName
  });
  console.log('Update data:', req.body);

  let lead = await Lead.findById(req.params.id);

  if (!lead) {
    return res.status(404).json({
      success: false,
      message: `No lead found with id of ${req.params.id}`
    });
  }

  console.log('Found lead:', {
    id: lead._id,
    name: lead.name,
    assignedTo: lead.assignedTo
  });

  // Check if user is authorized to update this lead
  if (
    req.user.role !== 'Admin' &&
    req.user.role !== 'Manager' &&
    req.user.role !== 'Lead Person' &&
    lead.assignedTo.toString() !== req.user._id.toString()
  ) {
    return res.status(403).json({
      success: false,
      message: 'Not authorized to update this lead'
    });
  }

  // For Sales Persons, only allow updating the status field
  if (req.user.role === 'Sales Person') {
    console.log('Sales Person is updating lead status to:', req.body.status);

    // Only update the status and updatedAt fields
    const updateData = {
      status: req.body.status,
      updatedAt: Date.now()
    };

    lead = await Lead.findByIdAndUpdate(req.params.id, updateData, {
      new: true,
      runValidators: true
    });

    return res.status(200).json({
      success: true,
      data: lead
    });
  }

  // For Lead Person, Manager and Admin, allow full updates
```

```
    // Map the new field names to the database model field names
    const updatedData = {
      name: req.body.NAME,
      email: req.body['E-MAIL'] || '',
      course: req.body.COURSE,
      countryCode: req.body.CODE,
      phone: req.body.NUMBER,
      country: req.body.COUNTRY,
      pseudoId: req.body['PSUDO ID'],
      client: req.body['CLIENT REMARK'],
      status: req.body.status || lead.status,
      source: req.body.SOURSE,
      sourceLink: req.body['SOURCE LINK'],
      assignedTo: req.body['SALE PERSON'],
      leadPerson: req.body['LEAD PERSON'],
      feedback: req.body.FEEDBACK,
      updatedAt: Date.now()
    };

    // Only include fields that are actually provided in the request
    const finalUpdateData = {};
    for (const [key, value] of Object.entries(updatedData)) {
      if (value !== undefined) {
        finalUpdateData[key] = value;
      }
    }

    lead = await Lead.findByIdAndUpdate(req.params.id, finalUpdateData, {
      new: true,
      runValidators: true
    });

    res.status(200).json({
      success: true,
      data: lead
    });
  } catch (err) {
    console.error('Error updating lead:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};

// @desc    Delete lead
// @route   DELETE /api/leads/:id
// @access  Private
exports.deleteLead = async (req, res) => {
  try {
    const lead = await Lead.findById(req.params.id);

    if (!lead) {
      return res.status(404).json({
        success: false,
        message: `No lead found with id of ${req.params.id}`
      });
    }
```

```
    // Check if user is authorized to delete this lead
    if (req.user.role !== 'Admin' && req.user.role !== 'Manager') {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to delete leads'
      });
    }

    await lead.deleteOne();

    res.status(200).json({
      success: true,
      data: {}
    });
  } catch (err) {
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};

// @desc    Get leads assigned to sales person
// @route   GET /api/leads/assigned
// @access  Private (Sales Person only)
exports.getAssignedLeads = async (req, res) => {
  try {
    // Verify the user is a Sales Person
    if (req.user.role !== 'Sales Person') {
      return res.status(403).json({
        success: false,
        message: 'Only Sales Persons can access their assigned leads'
      });
    }

    console.log('============= GET ASSIGNED LEADS REQUEST =============');
    console.log('Sales Person:', req.user.fullName);
    console.log('Query parameters:', req.query);

    // Extract date filtering parameters from query
    const { month, year, startDate, endDate } = req.query;

    let query = Lead.find({
      assignedTo: req.user._id
    });

    // Add date filtering if provided
    if (month && year) {
      console.log(`Filtering by month: ${month}, year: ${year}`);
      const startOfMonth = new Date(parseInt(year), parseInt(month) - 1, 1);
      const endOfMonth = new Date(parseInt(year), parseInt(month), 0, 23, 59, 59,
999);

      query = query.where('createdAt').gte(startOfMonth).lte(endOfMonth);
      console.log(`Date range: ${startOfMonth.toISOString()} to
${endOfMonth.toISOString()}`);
    } else if (startDate && endDate) {
      console.log(`Filtering by date range: ${startDate} to ${endDate}`);
      const start = new Date(startDate);
```

```
      const end = new Date(endDate);
      end.setHours(23, 59, 59, 999); // Include the entire end date

      query = query.where('createdAt').gte(start).lte(end);
      console.log(`Date range: ${start.toISOString()} to ${end.toISOString()}`);
    } else if (startDate) {
      console.log(`Filtering from date: ${startDate}`);
      const start = new Date(startDate);
      query = query.where('createdAt').gte(start);
    } else if (endDate) {
      console.log(`Filtering until date: ${endDate}`);
      const end = new Date(endDate);
      end.setHours(23, 59, 59, 999);
      query = query.where('createdAt').lte(end);
    }

    const leads = await query
      .populate('leadPerson', 'fullName email')
      .populate('createdBy', 'fullName email')
      .populate('assignedTo', 'fullName email')
      .sort({ createdAt: -1 });

    console.log(`Found ${leads.length} assigned leads for ${req.user.fullName}`);

    // Group leads by month/year for better organization
    const leadsByMonth = {};
    leads.forEach(lead => {
      const date = new Date(lead.createdAt);
      const monthKey = `${date.getFullYear()}-${String(date.getMonth() +
1).padStart(2, '0')}`;

      if (!leadsByMonth[monthKey]) {
        leadsByMonth[monthKey] = [];
      }
      leadsByMonth[monthKey].push(lead);
    });

    console.log('Leads grouped by month:');
    Object.keys(leadsByMonth).sort().forEach(month => {
      console.log(`- ${month}: ${leadsByMonth[month].length} leads`);
    });

    console.log('============================================');

    res.status(200).json({
      success: true,
      count: leads.length,
      data: leads,
      groupedByMonth: leadsByMonth,
      filters: {
        month: month || null,
        year: year || null,
        startDate: startDate || null,
        endDate: endDate || null
      }
    });
  } catch (err) {
    console.error('Error fetching assigned leads:', err);
    res.status(500).json({
```

```
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Update lead feedback
// @route   PUT /api/leads/:id/feedback
// @access  Private (Sales Person, Lead Person, Manager, Admin)
exports.updateFeedback = async (req, res) => {
  try {
    const { feedback } = req.body;

    if (!feedback) {
      return res.status(400).json({
        success: false,
        message: 'Feedback field is required'
      });
    }

    let lead = await Lead.findById(req.params.id);

    if (!lead) {
      return res.status(404).json({
        success: false,
        message: `No lead found with id of ${req.params.id}`
      });
    }

    // Check if user is authorized to update feedback for this lead
    if (
      req.user.role !== 'Admin' &&
      req.user.role !== 'Manager' &&
      lead.assignedTo.toString() !== req.user._id.toString() &&
      !(req.user.role === 'Lead Person' &&
        lead.leadPerson &&
        lead.leadPerson.toString() === req.user._id.toString())
    ) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to update feedback for this lead'
      });
    }

    // Update only the feedback field and updatedAt
    lead = await Lead.findByIdAndUpdate(
      req.params.id,
      {
        feedback,
        updatedAt: Date.now()
      },
      {
        new: true,
        runValidators: true
      }
    );

    res.status(200).json({
      success: true,
```

```
      data: lead
    });
  } catch (err) {
    console.error('Error updating feedback:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};

// @desc    Import leads from CSV (Google Sheets)
// @route   POST /api/leads/import
// @access  Private (Admin, Manager, Lead Person)
exports.importLeads = async (req, res) => {
  try {
    console.log('=== IMPORT LEADS REQUEST ===');
    console.log('Request body:', JSON.stringify(req.body, null, 2));
    console.log('User:', req.user.fullName, req.user.role);

    // Handle both direct leads array and nested data structure
    let leads = req.body.leads;
    if (!leads && req.body.data && req.body.data.leads) {
      leads = req.body.data.leads;
      console.log('Found leads in nested data structure');
    }

    if (!leads) {
      console.log('No leads field found in request body');
      return res.status(400).json({
        success: false,
        message: 'No leads field provided in request body'
      });
    }

    if (!Array.isArray(leads)) {
      console.log('Leads is not an array:', typeof leads);
      return res.status(400).json({
        success: false,
        message: 'Leads must be an array'
      });
    }

    if (leads.length === 0) {
      console.log('Leads array is empty');
      return res.status(400).json({
        success: false,
        message: 'Leads array is empty'
      });
    }

    console.log(`Importing ${leads.length} leads from CSV by ${req.user.fullName}
(${req.user.role})...`);
    console.log('First lead sample:', JSON.stringify(leads[0], null, 2));

    // Map Google Sheets column names to our database fields
    const mappedLeads = leads.map((lead, index) => {
      console.log(`Processing lead ${index + 1}:`, lead);
```

```
      const leadData = {
        name: lead.Name || lead.name || lead.NAME || '',
        email: lead.Email || lead.email || lead.EMAIL || '',
        course: lead.Course || lead.course || lead.COURSE || '',
        countryCode: lead.CountryCode || lead['Country Code'] || lead.countryCode
|| lead.COUNTRYCODE || '+1',
        phone: lead.Phone || lead.phone || lead.PHONE || lead.Number ||
lead.number || lead.NUMBER || '',
        country: lead.Country || lead.country || lead.COUNTRY || '',
        pseudoId: lead.PseudoId || lead.pseudoId || lead.ID || lead.id ||
lead.PSEUDOID || '',
        company: lead.Company || lead.company || lead.COMPANY || '',
        client: lead.Client || lead.client || lead.CLIENT || '',
        status: lead.Status || lead.status || lead.STATUS || 'Introduction', //
Default to Introduction stage
        source: lead.Source || lead.source || lead.SOURCE || '',
        sourceLink: lead.SourceLink || lead['Source Link'] || lead.sourceLink ||
lead.SOURCELINK || '',
        remarks: lead.Remarks || lead.remarks || lead.REMARKS || '',
        feedback: lead.Feedback || lead.feedback || lead.FEEDBACK || '',
        createdBy: req.user.id,
        // Set the createdAt date from CSV DATE field if available
        createdAt: (() => {
          if (lead.DATE || lead.Date || lead.date) {
            const dateStr = lead.DATE || lead.Date || lead.date;
            console.log(`Processing date: "${dateStr}"`);

            // Function to parse DD-MM-YYYY format
            const parseDDMMYYYY = (str) => {
              const ddmmyyyyPattern = /^(\d{1,2})[\/\-](\d{1,2})[\/\-](\d{4})$/;
              const match = str.match(ddmmyyyyPattern);
              if (match) {
                const [, day, month, year] = match;
                // Create date in YYYY-MM-DD format for proper parsing
                const isoFormat = `${year}-${month.padStart(2, '0')}-
${day.padStart(2, '0')}`;
                console.log(`Detected DD-MM-YYYY format: "${str}" ->
"${isoFormat}"`);
                return new Date(isoFormat);
              }
              return null;
            };

            // Try DD-MM-YYYY format first
            let parsedDate = parseDDMMYYYY(dateStr);

            if (parsedDate && !isNaN(parsedDate.getTime())) {
              console.log(`Successfully parsed DD-MM-YYYY format:
${parsedDate.toISOString()}`);
              return parsedDate;
            }

            // Try to parse the date directly (for YYYY-MM-DD and other standard
formats)
            parsedDate = new Date(dateStr);

            // If the date is invalid, try different formats
            if (isNaN(parsedDate.getTime())) {
              console.log(`Invalid date format: "${dateStr}", trying alternative
formats`);
```

```
              // Try common date formats
              const formats = [
                dateStr.replace(/\//g, '-'), // Convert slashes to dashes
                dateStr.replace(/-/g, '/'), // Convert dashes to slashes
              ];

              for (const format of formats) {
                parsedDate = new Date(format);
                if (!isNaN(parsedDate.getTime())) {
                  console.log(`Successfully parsed with format: ${format}`);
                  break;
                }
              }

              // If still invalid, use current date
              if (isNaN(parsedDate.getTime())) {
                console.log(`All date formats failed for "${dateStr}", using
current date`);
                parsedDate = new Date();
              }
            }

            console.log(`Final parsed date: ${parsedDate.toISOString()}
(${parsedDate.toString()})`);
            return parsedDate;
          } else {
            console.log('No date field found, using current date');
            return new Date();
          }
        })(),
        updatedAt: new Date()
      };

      // FIXED: Look for Lead Person in CSV data FIRST, regardless of who is
importing
      if (lead.LeadPerson || lead['Lead Person'] || lead.leadPerson ||
lead.LEADPERSON || lead['LEAD PERSON']) {
        const leadPersonName = lead.LeadPerson || lead['Lead Person'] ||
lead.leadPerson || lead.LEADPERSON || lead['LEAD PERSON'];
        leadData.leadPersonName = leadPersonName;
        console.log(`Found Lead Person in CSV: "${leadPersonName}"`);
      } else {
        // If no Lead Person specified in CSV, default to the importing user
(only if they are a Lead Person)
        if (req.user.role === 'Lead Person') {
          leadData.leadPerson = req.user._id;
          leadData.assignedTo = req.user._id; // Default assignment to Lead Person
          console.log(`No Lead Person in CSV, defaulting to importing user:
${req.user.fullName}`);
        }
      }

      // Look for Sales Person assignment
      if (lead.SalesPerson || lead['Sales Person'] || lead.salesPerson ||
lead.assignedTo || lead.SALESPERSON || lead['SALES PERSON']) {
        const salesPersonName = lead.SalesPerson || lead['Sales Person'] ||
lead.salesPerson || lead.assignedTo || lead.SALESPERSON || lead['SALES PERSON'];
        leadData.assignedToName = salesPersonName;
```

```
        console.log(`Found Sales Person in CSV: "${salesPersonName}"`);
      }

      console.log(`Mapped lead data:`, leadData);
      return leadData;
    });

    console.log(`Mapped ${mappedLeads.length} leads`);

    // FIXED: Look up and assign Lead Persons and Sales Persons by name for ALL
imports
    console.log('Looking up users for automatic assignment...');

    // Get all users who could be Lead Persons or Sales Persons
    const User = require('../models/User');
    const allUsers = await User.find({
      role: { $in: ['Lead Person', 'Sales Person', 'Admin', 'Manager'] }
    }).select('_id fullName email role');

    console.log(`Found ${allUsers.length} potential assignees:`, allUsers.map(u
=> `${u.fullName} (${u.role})`));

    // Process each mapped lead to assign Lead Persons and Sales Persons
    for (let leadData of mappedLeads) {
      // Assign Lead Person if specified in CSV
      if (leadData.leadPersonName) {
        console.log(`\nLooking for Lead Person: "${leadData.leadPersonName}"`);

        // Try to find a matching Lead Person by name
        const matchingLeadPerson = allUsers.find(user => {
          if (user.role !== 'Lead Person') return false;

          const userName = user.fullName.toLowerCase();
          const searchName = leadData.leadPersonName.toLowerCase();

          // Try exact match first
          if (userName === searchName) return true;

          // Try partial match (contains)
          if (userName.includes(searchName) || searchName.includes(userName))
return true;

          // Try first name match
          const userFirstName = userName.split(' ')[0];
          const searchFirstName = searchName.split(' ')[0];
          if (userFirstName === searchFirstName) return true;

          return false;
        });

        if (matchingLeadPerson) {
          leadData.leadPerson = matchingLeadPerson._id;
          console.log(`'  Lead Person assigned to: ${matchingLeadPerson.fullName}`);

          // If no sales person specified, default assign to the Lead Person
          if (!leadData.assignedToName) {
            leadData.assignedTo = matchingLeadPerson._id;
            console.log(`'  Also assigned to Lead Person as default assignee`);
          }
```

```
        } else {
          console.log(`'L No matching Lead Person found for
"${leadData.leadPersonName}"`);
          console.log('Available Lead Persons:', allUsers.filter(u => u.role ===
'Lead Person').map(u => u.fullName).join(', '));

          // Fallback to importing user if they are a Lead Person
          if (req.user.role === 'Lead Person') {
            leadData.leadPerson = req.user._id;
            leadData.assignedTo = req.user._id;
            console.log(`Fallback: Assigned to importing Lead Person:
${req.user.fullName}`);
          }
        }

        // Remove the temporary name field
        delete leadData.leadPersonName;
      }

      // Assign Sales Person if specified in CSV
      if (leadData.assignedToName) {
        console.log(`\nLooking for Sales Person: "${leadData.assignedToName}"`);

        // Try to find a matching user by name (Sales Person, Admin, or Manager)
        const matchingSalesPerson = allUsers.find(user => {
          if (!['Sales Person', 'Admin', 'Manager'].includes(user.role)) return
false;

          const userName = user.fullName.toLowerCase();
          const searchName = leadData.assignedToName.toLowerCase();

          // Try exact match first
          if (userName === searchName) return true;

          // Try partial match (contains)
          if (userName.includes(searchName) || searchName.includes(userName))
return true;

          // Try first name match
          const userFirstName = userName.split(' ')[0];
          const searchFirstName = searchName.split(' ')[0];
          if (userFirstName === searchFirstName) return true;

          return false;
        });

        if (matchingSalesPerson) {
          leadData.assignedTo = matchingSalesPerson._id;
          console.log(`'  Sales Person assigned to: ${matchingSalesPerson.fullName}
(${matchingSalesPerson.role})`);
        } else {
          console.log(`'L No matching Sales Person found for
"${leadData.assignedToName}"`);
          console.log('Available Sales Persons:', allUsers.filter(u => ['Sales
Person', 'Admin', 'Manager'].includes(u.role)).map(u => u.fullName).join(', '));

          // Keep existing assignment (Lead Person or importing user)
          console.log(`Keeping existing assignment`);
        }
```

```
        // Remove the temporary name field
        delete leadData.assignedToName;
      }
    }

    // Validate the mapped data - make validation more flexible
    const validLeads = mappedLeads.filter((lead, index) => {
      const isValid = lead.name && (lead.phone || lead.email) && lead.course;
      if (!isValid) {
        console.log(`Lead ${index + 1} is invalid:`, {
          name: lead.name,
          phone: lead.phone,
          email: lead.email,
          course: lead.course,
          hasName: !!lead.name,
          hasContact: !!(lead.phone || lead.email),
          hasCourse: !!lead.course
        });
      }
      return isValid;
    });

    console.log(`Found ${validLeads.length} valid leads out of
${mappedLeads.length}`);

    if (validLeads.length === 0) {
      console.log('No valid leads found');
      return res.status(400).json({
        success: false,
        message: 'No valid leads found in the imported data. Required fields:
Name, Phone/Email, Course',
        details: 'Make sure your CSV has columns for Name, Phone (or Email), and
Course'
      });
    }

    // Insert the leads into the database
    console.log('Attempting to insert leads into database...');
    const results = await Lead.insertMany(validLeads, {
      ordered: false // Continue processing even if some documents have errors
    });

    console.log(`Successfully imported ${results.length} leads`);

    // Log assignment info for debugging
    if (req.user.role === 'Lead Person') {
      console.log(`All imported leads assigned to Lead Person:
${req.user.fullName}`);

      // Count how many were auto-assigned to sales persons
      const autoAssigned = results.filter(lead =>
        lead.assignedTo && lead.assignedTo.toString() !== req.user._id.toString()
      ).length;

      if (autoAssigned > 0) {
        console.log(`${autoAssigned} leads were automatically assigned to sales
persons from CSV data`);
      }
```

```
    }

    res.status(201).json({
      success: true,
      count: results.length,
      data: results,
      message: `Successfully imported ${results.length} leads. ${
        req.user.role === 'Lead Person'
          ? 'Leads with sales person names in CSV were automatically assigned.
Others assigned to you as Lead Person.'
          : ''
      }`,
      skipped: mappedLeads.length - results.length
    });
  } catch (err) {
    console.error('Lead import error:', err);
    console.error('Error stack:', err.stack);
    res.status(400).json({
      success: false,
      message: err.message || 'Import failed',
      error: process.env.NODE_ENV === 'development' ? err.stack : undefined
    });
  }
};

// @desc    Get all customer leads (including reference sales)
// @route   GET /api/leads/customers
// @access  Private (Sales Person only)
exports.getAllCustomers = async (req, res) => {
  try {
    console.log('Getting all customers (leads + reference sales) for:',
req.user.fullName);

    // 1. Get assigned leads first
    const leads = await Lead.find({
      assignedTo: req.user._id
    })
    .populate('leadPerson', 'fullName email')
    .populate('createdBy', 'fullName email')
    .populate('assignedTo', 'fullName email')
    .sort({ createdAt: -1 });

    console.log(`Found ${leads.length} assigned leads`);

    // 2. Get reference sales for this sales person
    const Sale = require('../models/Sale');
    const referenceSales = await Sale.find({
      salesPerson: req.user._id,
      isReference: true
    })
    .sort({ date: -1 });

    console.log(`Found ${referenceSales.length} reference sales`);

    // 3. Convert reference sales to lead-like format
    const referenceSalesAsLeads = referenceSales.map(sale => {
      return {
        _id: sale._id, // Use the sale ID
        name: sale.customerName,
```

```
        email: sale.email,
        phone: sale.contactNumber,
        countryCode: sale.countryCode,
        country: sale.country,
        course: sale.course,
        source: 'Reference Sale',
        status: sale.status,
        assignedTo: sale.salesPerson,
        leadPerson: sale.leadPerson,
        createdBy: sale.createdBy,
        createdAt: sale.date || sale.createdAt,
        updatedAt: sale.updatedAt,
        isReferenceSale: true // Flag to indicate this is actually a reference
sale
      };
    });

    // 4. Combine both sets of data
    const allCustomers = [...leads, ...referenceSalesAsLeads];

    console.log(`Returning ${allCustomers.length} total customers
(${leads.length} leads + ${referenceSalesAsLeads.length} reference sales)`);

    res.status(200).json({
      success: true,
      count: allCustomers.length,
      data: allCustomers
    });
  } catch (err) {
    console.error('Error fetching all customers:', err);
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Get repeated customer information
// @route   GET /api/leads/repeat-customers
// @access  Private (Admin, Manager)
exports.getRepeatCustomers = async (req, res) => {
  try {
    console.log('============= GET REPEAT CUSTOMERS =============');
    console.log('User making request:', {
      id: req.user._id,
      role: req.user.role,
      name: req.user.fullName
    });

    // Only allow admin and manager to access this data
    if (req.user.role !== 'Admin' && req.user.role !== 'Manager') {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to access repeat customer data'
      });
    }

    // Find all leads marked as repeat customers
    const repeatCustomers = await Lead.find({ isRepeatCustomer: true })
```

```
      .populate('assignedTo', 'fullName email')
      .populate('leadPerson', 'fullName email')
      .populate('relatedLeads', 'name course createdAt')
      .sort({ createdAt: -1 });

    // Group customers by contact info to find all unique customers
    const uniqueCustomerMap = new Map();

    repeatCustomers.forEach(lead => {
      // Create a unique key based on contact info
      const key = `${lead.phone}|${lead.email}`;

      if (!uniqueCustomerMap.has(key)) {
        uniqueCustomerMap.set(key, {
          customerInfo: {
            name: lead.name,
            email: lead.email,
            phone: lead.phone,
            countryCode: lead.countryCode,
            country: lead.country
          },
          leads: []
        });
      }

      // Add this lead and related leads to the customer's records
      const customerData = uniqueCustomerMap.get(key);

      // Add the current lead
      customerData.leads.push({
        _id: lead._id,
        course: lead.course,
        createdAt: lead.createdAt,
        salesPerson: lead.assignedTo ? lead.assignedTo.fullName : 'Unassigned',
        previousCourses: lead.previousCourses
      });

      // Add related leads if they're not already included
      if (lead.relatedLeads && lead.relatedLeads.length > 0) {
        lead.relatedLeads.forEach(relatedLead => {
          // Check if this related lead is already in the list
          const alreadyIncluded = customerData.leads.some(l =>
            l._id.toString() === relatedLead._id.toString()
          );

          if (!alreadyIncluded) {
            customerData.leads.push({
              _id: relatedLead._id,
              course: relatedLead.course,
              createdAt: relatedLead.createdAt,
              salesPerson: 'Unknown' // We don't have this info from the
populated data
            });
          }
        });
      }

      // Sort leads by date
      customerData.leads.sort((a, b) => new Date(b.createdAt) - new
Date(a.createdAt));
```

```
    });

    // Convert the map to an array
    const uniqueCustomers = Array.from(uniqueCustomerMap.values());

    // Calculate some statistics
    const stats = {
      totalRepeatCustomers: uniqueCustomers.length,
      totalLeads: repeatCustomers.length,
      averageCoursesPerCustomer: uniqueCustomers.length > 0
        ? (repeatCustomers.length / uniqueCustomers.length).toFixed(2)
        : 0
    };

    console.log(`Found ${stats.totalRepeatCustomers} unique repeat customers with
${stats.totalLeads} total leads`);
    console.log('============================================');

    res.status(200).json({
      success: true,
      stats,
      data: uniqueCustomers
    });
  } catch (err) {
    console.error('Error getting repeat customers:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};
```

controllers/leaves.js

```
const Leave = require('../models/Leave');
const Employee = require('../models/Employee');
const User = require('../models/User');
const mongoose = require('mongoose');

// @desc    Apply for leave
// @route   POST /api/leaves
// @access  Private
exports.applyLeave = async (req, res) => {
  try {
    console.log('=== LEAVE APPLICATION REQUEST ===');
    console.log('User:', { id: req.user.id, role: req.user.role, fullName:
req.user.fullName });
    console.log('Request body:', req.body);

    const { leaveType, startDate, endDate, reason, isHalfDay, halfDaySession } =
req.body;

    // Validate required fields
    if (!leaveType || !startDate || !endDate || !reason) {
      console.log('Missing required fields:', { leaveType, startDate, endDate,
reason });
      return res.status(400).json({
        success: false,
```

```
    message: 'Please provide all required fields'
  });
}

// Find employee record
console.log('Looking for employee record with userId:', req.user.id);
let employee = await Employee.findOne({ userId: req.user.id });
console.log('Employee found:', employee ? 'Yes' : 'No');

if (!employee) {
  console.log('No employee record found, attempting to create one...');

  // Try to create a basic employee record for the user
  try {
    const Department = require('../models/Department');
    const Role = require('../models/EmployeeRole');

    // Find or create a default department
    let department = await Department.findOne({ name: 'General' });
    if (!department) {
      department = await Department.create({
        name: 'General',
        description: 'General Department'
      });
    }

    // Find or create a default role
    let role = await Role.findOne({ name: req.user.role });
    if (!role) {
      role = await Role.create({
        name: req.user.role || 'Employee',
        description: `Role for ${req.user.role || 'Employee'}`
      });
    }

    // Create employee record
    const newEmployee = await Employee.create({
      userId: req.user.id,
      fullName: req.user.fullName,
      email: req.user.email,
      department: department._id,
      role: role._id,
      status: 'ACTIVE'
    });

    console.log('Created new employee record:', newEmployee._id);
    employee = newEmployee;

  } catch (createError) {
    console.error('Error creating employee record:', createError);
    return res.status(400).json({
      success: false,
      message: 'Unable to create employee record. Please contact HR.'
    });
  }
}

// Validate dates
const start = new Date(startDate);
```

```
const end = new Date(endDate);
const today = new Date();
today.setHours(0, 0, 0, 0);

console.log('Date validation:', { start, end, today });

if (start < today) {
  return res.status(400).json({
    success: false,
    message: 'Start date cannot be in the past'
  });
}

if (start > end) {
  return res.status(400).json({
    success: false,
    message: 'End date must be after start date'
  });
}

// Check for overlapping leaves
console.log('Checking for overlapping leaves...');
const overlappingLeave = await Leave.findOne({
  employeeId: employee._id,
  status: { $in: ['pending', 'approved'] },
  $or: [
    { startDate: { $lte: end }, endDate: { $gte: start } }
  ]
});

if (overlappingLeave) {
  console.log('Found overlapping leave:', overlappingLeave._id);
  return res.status(400).json({
    success: false,
    message: 'You already have a leave application for overlapping dates'
  });
}

// Create leave application
console.log('Creating leave application...');

// Calculate total days manually
let totalDays;
if (isHalfDay) {
  totalDays = 0.5;
} else {
  const timeDiff = end.getTime() - start.getTime();
  totalDays = Math.ceil(timeDiff / (1000 * 3600 * 24)) + 1;
}

const leaveData = {
  employeeId: employee._id,
  userId: req.user.id,
  leaveType,
  startDate: start,
  endDate: end,
  totalDays: totalDays,
  reason,
  isHalfDay: isHalfDay || false,
```

```
      halfDaySession: (isHalfDay && halfDaySession) ? halfDaySession : undefined
    };

    console.log('Leave data to create:', leaveData);
    const leave = await Leave.create(leaveData);
    console.log('Leave created successfully:', leave._id);

    // Populate employee and user details
    const populatedLeave = await Leave.findById(leave._id)
      .populate('employeeId', 'fullName email department role')
      .populate('userId', 'fullName email');

    console.log('=== LEAVE APPLICATION SUCCESS ===');
    res.status(201).json({
      success: true,
      data: populatedLeave,
      message: 'Leave application submitted successfully'
    });

  } catch (error) {
    console.error('=== LEAVE APPLICATION ERROR ===');
    console.error('Error details:', {
      name: error.name,
      message: error.message,
      stack: error.stack
    });
    res.status(500).json({
      success: false,
      message: 'Server error while applying leave',
      error: error.message
    });
  }
};

// @desc    Get all leaves (for managers/admins)
// @route   GET /api/leaves
// @access  Private (Manager/Admin)
exports.getAllLeaves = async (req, res) => {
  try {
    const { status, leaveType, startDate, endDate, page = 1, limit = 10 } =
req.query;

    // Build query
    const query = {};

    if (status) query.status = status;
    if (leaveType) query.leaveType = leaveType;
    if (startDate || endDate) {
      query.startDate = {};
      if (startDate) query.startDate.$gte = new Date(startDate);
      if (endDate) query.startDate.$lte = new Date(endDate);
    }

    const skip = (page - 1) * limit;

    const leaves = await Leave.find(query)
      .populate('employeeId', 'fullName email department role')
      .populate('userId', 'fullName email')
      .populate('approvedBy', 'fullName email')
```

```
      .sort({ appliedDate: -1 })
      .skip(skip)
      .limit(parseInt(limit));

    const total = await Leave.countDocuments(query);

    res.status(200).json({
      success: true,
      data: leaves,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });

  } catch (error) {
    console.error('Error getting leaves:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while fetching leaves'
    });
  }
};

// @desc    Get my leaves
// @route   GET /api/leaves/my-leaves
// @access  Private
exports.getMyLeaves = async (req, res) => {
  try {
    const { status, year, page = 1, limit = 10 } = req.query;

    // Find employee record
    const employee = await Employee.findOne({ userId: req.user.id });
    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee record not found'
      });
    }

    // Build query
    const query = { employeeId: employee._id };

    if (status) query.status = status;
    if (year) {
      const startOfYear = new Date(year, 0, 1);
      const endOfYear = new Date(year, 11, 31);
      query.startDate = { $gte: startOfYear, $lte: endOfYear };
    }

    const skip = (page - 1) * limit;

    const leaves = await Leave.find(query)
      .populate('approvedBy', 'fullName email')
      .sort({ appliedDate: -1 })
      .skip(skip)
      .limit(parseInt(limit));
```

```
    const total = await Leave.countDocuments(query);

    res.status(200).json({
      success: true,
      data: leaves,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });

  } catch (error) {
    console.error('Error getting my leaves:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while fetching your leaves'
    });
  }
};

// @desc    Approve/Reject leave
// @route   PUT /api/leaves/:id/status
// @access  Private (Manager/Admin)
exports.updateLeaveStatus = async (req, res) => {
  try {
    const { status, rejectionReason } = req.body;

    if (!['approved', 'rejected'].includes(status)) {
      return res.status(400).json({
        success: false,
        message: 'Invalid status. Must be approved or rejected'
      });
    }

    if (status === 'rejected' && !rejectionReason) {
      return res.status(400).json({
        success: false,
        message: 'Rejection reason is required when rejecting leave'
      });
    }

    const leave = await Leave.findById(req.params.id);
    if (!leave) {
      return res.status(404).json({
        success: false,
        message: 'Leave application not found'
      });
    }

    if (leave.status !== 'pending') {
      return res.status(400).json({
        success: false,
        message: 'Leave application has already been processed'
      });
    }
```

```
    // Update leave status
    leave.status = status;
    leave.approvedBy = req.user.id;
    leave.approvedDate = new Date();

    if (status === 'rejected') {
      leave.rejectionReason = rejectionReason;
    }

    await leave.save();

    // Populate for response
    const populatedLeave = await Leave.findById(leave._id)
      .populate('employeeId', 'fullName email department role')
      .populate('userId', 'fullName email')
      .populate('approvedBy', 'fullName email');

    res.status(200).json({
      success: true,
      data: populatedLeave,
      message: `Leave ${status} successfully`
    });

  } catch (error) {
    console.error('Error updating leave status:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while updating leave status'
    });
  }
};

// @desc    Cancel leave
// @route   PUT /api/leaves/:id/cancel
// @access  Private
exports.cancelLeave = async (req, res) => {
  try {
    const leave = await Leave.findById(req.params.id);
    if (!leave) {
      return res.status(404).json({
        success: false,
        message: 'Leave application not found'
      });
    }

    // Check if user owns this leave
    if (leave.userId.toString() !== req.user.id) {
      return res.status(403).json({
        success: false,
        message: 'You can only cancel your own leaves'
      });
    }

    if (leave.status !== 'pending') {
      return res.status(400).json({
        success: false,
        message: 'You can only cancel pending leave applications'
      });
    }
```

```
    leave.status = 'cancelled';
    await leave.save();

    res.status(200).json({
      success: true,
      data: leave,
      message: 'Leave cancelled successfully'
    });

  } catch (error) {
    console.error('Error cancelling leave:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while cancelling leave'
    });
  }
};

// @desc    Get leave statistics
// @route   GET /api/leaves/stats
// @access  Private
exports.getLeaveStats = async (req, res) => {
  try {
    const { year = new Date().getFullYear() } = req.query;

    let matchQuery = {};

    // If not admin/manager, only show own stats
    if (!['Admin', 'Manager'].includes(req.user.role)) {
      const employee = await Employee.findOne({ userId: req.user.id });
      if (employee) {
        matchQuery.employeeId = employee._id;
      }
    }

    // Add year filter
    const startOfYear = new Date(year, 0, 1);
    const endOfYear = new Date(year, 11, 31);
    matchQuery.startDate = { $gte: startOfYear, $lte: endOfYear };

    const stats = await Leave.aggregate([
      { $match: matchQuery },
      {
        $group: {
          _id: '$status',
          count: { $sum: 1 },
          totalDays: { $sum: '$totalDays' }
        }
      }
    ]);

    const leaveTypeStats = await Leave.aggregate([
      { $match: matchQuery },
      {
        $group: {
          _id: '$leaveType',
          count: { $sum: 1 },
          totalDays: { $sum: '$totalDays' }
```

```
        }
      }
    ]);

    res.status(200).json({
      success: true,
      data: {
        statusStats: stats,
        leaveTypeStats: leaveTypeStats
      }
    });

  } catch (error) {
    console.error('Error getting leave stats:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while fetching leave statistics'
    });
  }
};

// @desc    Get leave balance
// @route   GET /api/leaves/balance
// @access  Private
exports.getLeaveBalance = async (req, res) => {
  try {
    const { year = new Date().getFullYear() } = req.query;

    const employee = await Employee.findOne({ userId: req.user.id });
    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee record not found'
      });
    }

    const startOfYear = new Date(year, 0, 1);
    const endOfYear = new Date(year, 11, 31);

    // Get used leaves
    const usedLeaves = await Leave.aggregate([
      {
        $match: {
          employeeId: employee._id,
          status: 'approved',
          startDate: { $gte: startOfYear, $lte: endOfYear }
        }
      },
      {
        $group: {
          _id: '$leaveType',
          totalDays: { $sum: '$totalDays' }
        }
      }
    ]);

    // Default leave balances (these should be configurable)
    const defaultBalances = {
      annual: 21,
```

```
      sick: 10,
      casual: 12,
      emergency: 5,
      personal: 3
    };

    const balances = {};
    Object.keys(defaultBalances).forEach(type => {
      const used = usedLeaves.find(leave => leave._id === type);
      balances[type] = {
        total: defaultBalances[type],
        used: used ? used.totalDays : 0,
        remaining: defaultBalances[type] - (used ? used.totalDays : 0)
      };
    });

    res.status(200).json({
      success: true,
      data: balances
    });

  } catch (error) {
    console.error('Error getting leave balance:', error);
    res.status(500).json({
      success: false,
      message: 'Server error while fetching leave balance'
    });
  }
};
```

controllers/logController.js

```
const Log = require('../models/Log');
const asyncHandler = require('../middleware/async');

// @desc    Create a new log entry
// @route   POST /api/logs
// @access  Private
exports.createLog = asyncHandler(async (req, res) => {
  const log = await Log.create({
    ...req.body,
    performedBy: req.user._id,
    ipAddress: req.ip,
    userAgent: req.headers['user-agent']
  });

  res.status(201).json({
    success: true,
    data: log
  });
});

// @desc    Get all logs with pagination and filters
// @route   GET /api/logs
// @access  Private/Admin
exports.getLogs = asyncHandler(async (req, res) => {
  const page = parseInt(req.query.page, 10) || 1;
  const limit = parseInt(req.query.limit, 10) || 50;
```

```
  const startIndex = (page - 1) * limit;

  let query = {};

  // Add filters if they exist
  if (req.query.action) {
    query.action = req.query.action;
  }
  if (req.query.performedBy) {
    query.performedBy = req.query.performedBy;
  }
  if (req.query.status) {
    query.status = req.query.status;
  }
  if (req.query.affectedResource) {
    query.affectedResource = req.query.affectedResource;
  }
  if (req.query.startDate && req.query.endDate) {
    query.timestamp = {
      $gte: new Date(req.query.startDate),
      $lte: new Date(req.query.endDate)
    };
  }

  // Execute query with pagination
  const logs = await Log.find(query)
    .populate('performedBy', 'fullName email role')
    .sort({ timestamp: -1 })
    .skip(startIndex)
    .limit(limit);

  // Get total count for pagination
  const total = await Log.countDocuments(query);

  res.status(200).json({
    success: true,
    count: logs.length,
    total,
    pagination: {
      page,
      limit,
      totalPages: Math.ceil(total / limit)
    },
    data: logs
  });
});

// @desc    Get log statistics
// @route   GET /api/logs/stats
// @access  Private/Admin
exports.getLogStats = asyncHandler(async (req, res) => {
  const stats = await Log.aggregate([
    {
      $group: {
        _id: '$action',
        count: { $sum: 1 },
        successCount: {
          $sum: {
            $cond: [{ $eq: ['$status', 'SUCCESS'] }, 1, 0]
```

```
          }
        },
        failureCount: {
          $sum: {
            $cond: [{ $eq: ['$status', 'FAILURE'] }, 1, 0]
          }
        }
      }
    }
  ]);

  // Get user activity stats
  const userStats = await Log.aggregate([
    {
      $group: {
        _id: '$performedBy',
        actionCount: { $sum: 1 }
      }
    },
    {
      $sort: { actionCount: -1 }
    },
    {
      $limit: 10
    }
  ]);

  // Get today's logs count
  const today = new Date();
  today.setHours(0, 0, 0, 0);
  const todayCount = await Log.countDocuments({
    timestamp: { $gte: today }
  });

  res.status(200).json({
    success: true,
    data: {
      actionStats: stats,
      topUsers: userStats,
      todayCount
    }
  });
});

// @desc    Get logs by resource ID
// @route   GET /api/logs/resource/:resourceId
// @access  Private/Admin
exports.getLogsByResource = asyncHandler(async (req, res) => {
  const logs = await Log.find({ resourceId: req.params.resourceId })
    .populate('performedBy', 'fullName email role')
    .sort({ timestamp: -1 });

  res.status(200).json({
    success: true,
    count: logs.length,
    data: logs
  });
});
```

```
// @desc    Delete old logs (older than 30 days)
// @route   DELETE /api/logs/cleanup
// @access  Private/Admin
exports.cleanupOldLogs = asyncHandler(async (req, res) => {
  const thirtyDaysAgo = new Date();
  thirtyDaysAgo.setDate(thirtyDaysAgo.getDate() - 30);

  const result = await Log.deleteMany({
    timestamp: { $lt: thirtyDaysAgo }
  });

  res.status(200).json({
    success: true,
    data: {
      deletedCount: result.deletedCount
    }
  });
});
```

controllers/payroll.js

```
const Payroll = require('../models/Payroll');
const Employee = require('../models/Employee');
const Attendance = require('../models/Attendance');
const Incentive = require('../models/Incentive');
const User = require('../models/User');
const PDFDocument = require('pdfkit');
const fs = require('fs');
const path = require('path');

// @desc    Generate payroll for a specific month
// @route   POST /api/payroll/generate
// @access  Private (Admin/HR/Manager)
exports.generatePayroll = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to generate payroll'
      });
    }

    const { employeeId, month, year } = req.body;

    // Validate input
    if (!employeeId || !month || !year) {
      return res.status(400).json({
        success: false,
        message: 'Employee ID, month, and year are required'
      });
    }

    // Check if payroll already exists
    const existingPayroll = await Payroll.findOne({
      employeeId,
      month,
      year
```

```
    });

    if (existingPayroll) {
      return res.status(400).json({
        success: false,
        message: 'Payroll already exists for this month'
      });
    }

    // Get employee details
    const employee = await Employee.findById(employeeId);
    if (!employee) {
      return res.status(404).json({
        success: false,
        message: 'Employee not found'
      });
    }

    // Get attendance data for the month
    const startDate = new Date(year, month - 1, 1);
    const endDate = new Date(year, month, 0);

    const attendance = await Attendance.find({
      employeeId: employeeId,
      date: { $gte: startDate, $lte: endDate }
    });

    // Calculate attendance summary
    const workingDays = endDate.getDate(); // Total days in the month
    const presentDays = attendance.filter(a => a.status === 'PRESENT').length;
    const halfDays = attendance.filter(a => a.status === 'HALF_DAY').length;
    const absentDays = workingDays - presentDays - halfDays; // Correct
calculation
    const overtimeHours = attendance.reduce((sum, a) => sum + (a.overtimeHours ||
0), 0);

    console.log('Ø=ÜÅ Attendance Summary Calculation:', {
      workingDays,
      presentDays,
      halfDays,
      absentDays,
      overtimeHours
    });

    // Get approved incentives for the month (optional - can be overridden
manually)
    const incentives = await Incentive.getIncentivesForPayroll(employeeId, month,
year);

    // Calculate incentive amounts (as fallback values)
    const autoPerformanceBonus = incentives
      .filter(i => i.type === 'PERFORMANCE')
      .reduce((sum, i) => sum + i.amount, 0);

    const autoProjectBonus = incentives
      .filter(i => i.type === 'PROJECT')
      .reduce((sum, i) => sum + i.amount, 0);

    const autoFestivalBonus = incentives
```

```
      .filter(i => i.type === 'FESTIVAL')
      .reduce((sum, i) => sum + i.amount, 0);

    // Create payroll record - use manual values if provided, otherwise use
calculated values
    const payrollData = {
      employeeId,
      userId: employee.userId,
      month,
      year,
      baseSalary: req.body.baseSalary || employee.salary,
      daysPresent: req.body.daysPresent || presentDays,
      calculatedSalary: req.body.calculatedSalary || ((req.body.baseSalary ||
employee.salary) / 30) * (req.body.daysPresent || presentDays),
      // Use manual values for attendance if provided, otherwise use calculated
values
      workingDays: req.body.workingDays || workingDays,
      presentDays: req.body.daysPresent || presentDays,
      absentDays: req.body.workingDays && req.body.daysPresent ? (30 -
req.body.daysPresent) : (30 - presentDays),
      halfDays: req.body.halfDays || halfDays,
      overtimeHours: req.body.overtimeHours || overtimeHours,
      // Manual Allowances
      hra: req.body.hra || 0,
      da: req.body.da || 0,
      conveyanceAllowance: req.body.conveyanceAllowance || 0,
      medicalAllowance: req.body.medicalAllowance || 0,
      specialAllowance: req.body.specialAllowance || 0,
      overtimeAmount: req.body.overtimeAmount || 0,
      // Bonuses
      performanceBonus: req.body.performanceBonus || autoPerformanceBonus,
      projectBonus: req.body.projectBonus || autoProjectBonus,
      attendanceBonus: req.body.attendanceBonus || 0,
      festivalBonus: req.body.festivalBonus || autoFestivalBonus,
      // Manual Deductions
      pf: req.body.pf || 0,
      esi: req.body.esi || 0,
      tax: req.body.tax || 0,
      loan: req.body.loan || 0,
      other: req.body.other || 0,
      notes: req.body.notes || ''
    };

    console.log('Ø=Ü¾ Final Payroll Data:', {
      workingDays: payrollData.workingDays,
      presentDays: payrollData.presentDays,
      absentDays: payrollData.absentDays,
      halfDays: payrollData.halfDays
    });

    const payroll = await Payroll.create(payrollData);

    // Trigger salary calculation
    await payroll.save();

    // Update incentives with payroll reference
    if (incentives.length > 0) {
      await Incentive.updateMany(
        { _id: { $in: incentives.map(i => i._id) } },
```

```
        { payrollId: payroll._id }
      );
    }

    res.status(201).json({
      success: true,
      data: payroll,
      message: 'Payroll generated successfully'
    });
  } catch (error) {
    console.error('Generate payroll error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error during payroll generation'
    });
  }
};

// @desc    Get payroll records
// @route   GET /api/payroll
// @access  Private
exports.getPayroll = async (req, res) => {
  try {
    const { month, year, employeeId } = req.query;

    // Build query based on user role
    let query = {};

    if (req.user.role === 'Employee') {
      // Employees can only see their own payroll
      const employee = await Employee.findOne({ userId: req.user.id });
      if (!employee) {
        return res.status(404).json({
          success: false,
          message: 'Employee record not found'
        });
      }
      query.employeeId = employee._id;
    } else if (['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      // Admin/HR/Manager can see all or filter by employee
      if (employeeId) {
        query.employeeId = employeeId;
      }
    } else {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to view payroll'
      });
    }

    // Add month/year filters
    if (month) query.month = parseInt(month);
    if (year) query.year = parseInt(year);

    const payroll = await Payroll.find(query)
      .populate('employeeId', 'fullName email department')
      .populate('userId', 'fullName email')
      .sort({ year: -1, month: -1 });
```

```
    res.status(200).json({
      success: true,
      data: payroll
    });
  } catch (error) {
    console.error('Get payroll error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Update payroll
// @route   PUT /api/payroll/:id
// @access  Private (Admin/HR/Manager)
exports.updatePayroll = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to update payroll'
      });
    }

    const payroll = await Payroll.findById(req.params.id);
    if (!payroll) {
      return res.status(404).json({
        success: false,
        message: 'Payroll record not found'
      });
    }

    // Update allowed fields
    const allowedFields = [
      'baseSalary', 'daysPresent', 'calculatedSalary', 'workingDays',
      // Manual Allowances
      'hra', 'da', 'conveyanceAllowance', 'medicalAllowance', 'specialAllowance',
'overtimeAmount',
      // Bonuses
      'performanceBonus', 'projectBonus', 'attendanceBonus', 'festivalBonus',
      // Manual Deductions
      'pf', 'esi', 'tax', 'loan', 'other',
      // Status and notes
      'notes', 'status'
    ];

    allowedFields.forEach(field => {
      if (req.body[field] !== undefined) {
        payroll[field] = req.body[field];
      }
    });

    // Auto-calculate salary if base salary or days present are updated
    if (req.body.baseSalary !== undefined || req.body.daysPresent !== undefined) {
      const baseSalary = req.body.baseSalary || payroll.baseSalary;
      const daysPresent = req.body.daysPresent || payroll.daysPresent;
      payroll.calculatedSalary = (baseSalary / 30) * daysPresent;
```

```
    }

    // Auto-calculate absent days if working days or present days are updated
    if (req.body.workingDays !== undefined || req.body.daysPresent !== undefined)
{
      const presentDays = req.body.daysPresent || payroll.presentDays;
      payroll.absentDays = 30 - presentDays;

      console.log('Ø=ÜÅ Updated attendance calculation:', {
        standardWorkingDays: 30,
        presentDays: payroll.presentDays,
        absentDays: payroll.absentDays
      });
    }

    // If status is being approved, set approval details
    if (req.body.status === 'APPROVED') {
      payroll.approvedBy = req.user.id;
      payroll.approvedDate = new Date();
    }

    await payroll.save();

    // Populate employee details for response
    await payroll.populate('employeeId', 'fullName email department');
    await payroll.populate('userId', 'fullName email');

    res.status(200).json({
      success: true,
      data: payroll,
      message: 'Payroll updated successfully'
    });
  } catch (error) {
    console.error('Update payroll error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Generate salary slip PDF
// @route   GET /api/payroll/:id/salary-slip
// @access  Private
exports.generateSalarySlip = async (req, res) => {
  try {
    const payroll = await Payroll.findById(req.params.id)
      .populate('employeeId', 'fullName email phoneNumber department')
      .populate('userId', 'fullName email');

    if (!payroll) {
      return res.status(404).json({
        success: false,
        message: 'Payroll record not found'
      });
    }

    // Check authorization
    if (req.user.role === 'Employee') {
```

```
      const employee = await Employee.findOne({ userId: req.user.id });
      if (!employee || employee._id.toString() !==
payroll.employeeId._id.toString()) {
        return res.status(403).json({
          success: false,
          message: 'Not authorized to view this salary slip'
        });
      }
    } else if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to generate salary slip'
      });
    }

    // Create PDF and pipe directly to response
    const doc = new PDFDocument();

    // Set response headers
    res.setHeader('Content-Type', 'application/pdf');
    res.setHeader('Content-Disposition', `attachment; filename="salary-slip-
${payroll.employeeId.fullName}-${payroll.month}-${payroll.year}.pdf"`);

    // Pipe the PDF directly to the response
    doc.pipe(res);

    // Company Logo
    try {
      const logoPath = path.join(__dirname, '../assets/images/traincape-
logo.jpg');
      if (fs.existsSync(logoPath)) {
        doc.image(logoPath, 250, 30, { width: 100, height: 60 });
        doc.moveDown(2);
      }
    } catch (error) {
      console.log('Logo not found, continuing without logo');
      doc.moveDown();
    }

    // Header
    doc.fontSize(16).text('SALARY SLIP', { align: 'center' });
    doc.moveDown();
    doc.fontSize(12).text('Traincape Technology', { align: 'center' });
    doc.fontSize(10).text('Khandolia Plaza, 118C, Dabri - Palam Rd, Delhi
110045', { align: 'center' });
    doc.moveDown();

    // Employee Details
    doc.fontSize(10);
    doc.text(`Employee Name: ${payroll.employeeId.fullName}`);
    doc.text(`Employee ID: ${payroll.employeeId._id}`);
    doc.text(`Department: ${payroll.employeeId.department || 'N/A'}`);
    doc.text(`Email: ${payroll.employeeId.email}`);
    doc.text(`Phone: ${payroll.employeeId.phoneNumber || 'N/A'}`);
    doc.moveDown();

    // Pay Period
    const months = ['January', 'February', 'March', 'April', 'May', 'June',
'July', 'August', 'September', 'October', 'November', 'December'];
```

```
    doc.text(`Pay Period: ${months[payroll.month - 1]} ${payroll.year}`);
    doc.text(`Working Days: ${payroll.workingDays}`);
    doc.text(`Days Present: ${payroll.daysPresent}`);
    doc.moveDown();

    // Earnings
    doc.fontSize(12).text('Earnings', { underline: true });
    doc.fontSize(10);
    doc.text(`Base Salary:  ¹${payroll.baseSalary.toFixed(2)}`);
    doc.text(`House Rent Allowance (HRA):  ¹${payroll.hra.toFixed(2)}`);
    doc.text(`Dearness Allowance (DA):  ¹${payroll.da.toFixed(2)}`);
    doc.text(`Conveyance Allowance:  ¹${payroll.conveyanceAllowance.toFixed(2)}`);
    doc.text(`Medical Allowance:  ¹${payroll.medicalAllowance.toFixed(2)}`);
    doc.text(`Special Allowance:  ¹${payroll.specialAllowance.toFixed(2)}`);
    doc.text(`Overtime Amount:  ¹${payroll.overtimeAmount.toFixed(2)}`);
    doc.moveDown();

    // Bonuses
    doc.fontSize(12).text('Bonuses', { underline: true });
    doc.fontSize(10);
    doc.text(`Performance Bonus:  ¹${payroll.performanceBonus.toFixed(2)}`);
    doc.text(`Project Bonus:  ¹${payroll.projectBonus.toFixed(2)}`);
    doc.text(`Attendance Bonus:  ¹${payroll.attendanceBonus.toFixed(2)}`);
    doc.text(`Festival Bonus:  ¹${payroll.festivalBonus.toFixed(2)}`);
    doc.moveDown();

    // Deductions
    doc.fontSize(12).text('Deductions', { underline: true });
    doc.fontSize(10);
    doc.text(`Provident Fund (PF):  ¹${payroll.pf.toFixed(2)}`);
    doc.text(`ESI:  ¹${payroll.esi.toFixed(2)}`);
    doc.text(`Professional Tax:  ¹${payroll.tax.toFixed(2)}`);
    doc.text(`Loan Recovery:  ¹${payroll.loan.toFixed(2)}`);
    doc.text(`Other Deductions:  ¹${payroll.other.toFixed(2)}`);
    doc.moveDown();

    // Total Calculations
    const totalEarnings = payroll.baseSalary + payroll.hra + payroll.da +
                          payroll.conveyanceAllowance + payroll.medicalAllowance +
                          payroll.specialAllowance + payroll.overtimeAmount +
                          payroll.performanceBonus + payroll.projectBonus +
                          payroll.attendanceBonus + payroll.festivalBonus;

    const totalDeductions = payroll.pf + payroll.esi + payroll.tax +
                           payroll.loan + payroll.other;

    doc.fontSize(12).text('Summary', { underline: true });
    doc.fontSize(10);
    doc.text(`Total Earnings:  ¹${totalEarnings.toFixed(2)}`);
    doc.text(`Total Deductions:  ¹${totalDeductions.toFixed(2)}`);
    doc.moveDown();
    doc.fontSize(12).text(`Net Salary:  ¹${payroll.netSalary.toFixed(2)}`, { bold:
true });

    // Footer
    doc.moveDown(2);
    doc.fontSize(8);
    doc.text('This is a computer-generated document. No signature is required.',
{ align: 'center' });
```

```
    doc.text(`Generated on: ${new Date().toLocaleDateString()}`, { align:
'center' });

    // Finalize PDF
    doc.end();

  } catch (error) {
    console.error('Generate salary slip error:', error);
    // Only send error response if headers haven't been sent
    if (!res.headersSent) {
      res.status(500).json({
        success: false,
        message: 'Server error while generating salary slip'
      });
    }
  }
};

// @desc    Get salary slip download link
// @route   GET /api/payroll/:id/download
// @access  Private
exports.downloadSalarySlip = async (req, res) => {
  try {
    const payroll = await Payroll.findById(req.params.id);
    if (!payroll) {
      return res.status(404).json({
        success: false,
        message: 'Payroll record not found'
      });
    }

    // Check authorization
    if (req.user.role === 'Employee') {
      const employee = await Employee.findOne({ userId: req.user.id });
      if (!employee || employee._id.toString() !== payroll.employeeId.toString())
{
        return res.status(403).json({
          success: false,
          message: 'Not authorized to download this salary slip'
        });
      }
    } else if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to download salary slip'
      });
    }

    if (!payroll.salarySlipPath || !fs.existsSync(payroll.salarySlipPath)) {
      return res.status(404).json({
        success: false,
        message: 'Salary slip file not found. Please generate it first.'
      });
    }

    // Send file
    res.download(payroll.salarySlipPath);
  } catch (error) {
    console.error('Download salary slip error:', error);
```

```
    res.status(500).json({
      success: false,
      message: 'Server error during salary slip download'
    });
  }
};

// @desc    Approve payroll
// @route   PUT /api/payroll/:id/approve
// @access  Private (Admin/HR/Manager)
exports.approvePayroll = async (req, res) => {
  try {
    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Not authorized to approve payroll'
      });
    }

    const payroll = await Payroll.findById(req.params.id);
    if (!payroll) {
      return res.status(404).json({
        success: false,
        message: 'Payroll record not found'
      });
    }

    payroll.status = 'APPROVED';
    payroll.approvedBy = req.user.id;
    payroll.approvedDate = new Date();

    await payroll.save();

    res.status(200).json({
      success: true,
      data: payroll,
      message: 'Payroll approved successfully'
    });
  } catch (error) {
    console.error('Approve payroll error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};

// @desc    Delete payroll
// @route   DELETE /api/payroll/:id
// @access  Private (Admin/HR/Manager)
exports.deletePayroll = async (req, res) => {
  try {
    console.log('Delete payroll request received for ID:', req.params.id);
    console.log('User role:', req.user.role);

    // Check authorization
    if (!['Admin', 'HR', 'Manager'].includes(req.user.role)) {
      console.log('Authorization failed - user role not allowed');
```

```
      return res.status(403).json({
        success: false,
        message: 'Not authorized to delete payroll'
      });
    }

    const payroll = await Payroll.findById(req.params.id);
    if (!payroll) {
      console.log('Payroll not found with ID:', req.params.id);
      return res.status(404).json({
        success: false,
        message: 'Payroll record not found'
      });
    }

    console.log('Payroll found with status:', payroll.status);

    console.log('Attempting to delete payroll...');

    // Delete associated salary slip file if exists
    if (payroll.salarySlipPath && fs.existsSync(payroll.salarySlipPath)) {
      fs.unlinkSync(payroll.salarySlipPath);
      console.log('Deleted salary slip file');
    }

    // Reset associated incentives if any
    const Incentive = require('../models/Incentive');
    await Incentive.updateMany(
      { payrollId: payroll._id },
      { $unset: { payrollId: 1 } }
    );
    console.log('Reset associated incentives');

    await Payroll.findByIdAndDelete(req.params.id);
    console.log('Payroll deleted successfully');

    res.status(200).json({
      success: true,
      data: {},
      message: 'Payroll deleted successfully'
    });
  } catch (error) {
    console.error('Delete payroll error:', error);
    res.status(500).json({
      success: false,
      message: 'Server error'
    });
  }
};
```

controllers/prospectController.js

```
const Prospect = require('../models/Prospect');
const Lead = require('../models/Lead');

// Get all prospects with filtering and pagination
const getProspects = async (req, res) => {
  try {
```

```
    const {
      page = 1,
      limit = 10,
      status,
      source,
      priority,
      assignedTo,
      search,
      sortBy = 'createdAt',
      sortOrder = 'desc'
    } = req.query;

    // Build filter object
    const filter = {};

    if (status) filter.status = status;
    if (source) filter.source = source;
    if (priority) filter.priority = priority;
    if (assignedTo) filter.assignedTo = assignedTo;

    // Search functionality
    if (search) {
      filter.$or = [
        { name: { $regex: search, $options: 'i' } },
        { email: { $regex: search, $options: 'i' } },
        { phone: { $regex: search, $options: 'i' } },
        { company: { $regex: search, $options: 'i' } }
      ];
    }

    // Role-based filtering
    if (req.user.role === 'Sales Person') {
      filter.assignedTo = req.user._id;
    }

    const skip = (page - 1) * limit;
    const sortOptions = {};
    sortOptions[sortBy] = sortOrder === 'desc' ? -1 : 1;

    const prospects = await Prospect.find(filter)
      .populate('assignedTo', 'fullName email')
      .populate('createdBy', 'fullName email')
      .sort(sortOptions)
      .skip(skip)
      .limit(parseInt(limit));

    const total = await Prospect.countDocuments(filter);

    res.json({
      success: true,
      data: prospects,
      pagination: {
        current: parseInt(page),
        pages: Math.ceil(total / limit),
        total,
        limit: parseInt(limit)
      }
    });
  } catch (error) {
```

```
    console.error('Get prospects error:', error);
    res.status(500).json({
      success: false,
      message: 'Error fetching prospects',
      error: error.message
    });
  }
};

// Get single prospect by ID
const getProspectById = async (req, res) => {
  try {
    const prospect = await Prospect.findById(req.params.id)
      .populate('assignedTo', 'fullName email role')
      .populate('createdBy', 'fullName email role')
      .populate('leadId', 'name status');

    if (!prospect) {
      return res.status(404).json({
        success: false,
        message: 'Prospect not found'
      });
    }

    // Role-based access check
    if (req.user.role === 'Sales Person' &&
        prospect.assignedTo &&
        prospect.assignedTo._id.toString() !== req.user._id.toString()) {
      return res.status(403).json({
        success: false,
        message: 'Access denied'
      });
    }

    res.json({
      success: true,
      data: prospect
    });
  } catch (error) {
    console.error('Get prospect error:', error);
    res.status(500).json({
      success: false,
      message: 'Error fetching prospect',
      error: error.message
    });
  }
};

// Create new prospect
const createProspect = async (req, res) => {
  try {
    const prospectData = {
      ...req.body,
      createdBy: req.user._id
    };

    // If no assignedTo specified and user is Sales Person, assign to self
    if (!prospectData.assignedTo && req.user.role === 'Sales Person') {
      prospectData.assignedTo = req.user._id;
```

```
    }

    const prospect = new Prospect(prospectData);
    await prospect.save();

    const populatedProspect = await Prospect.findById(prospect._id)
      .populate('assignedTo', 'fullName email')
      .populate('createdBy', 'fullName email');

    res.status(201).json({
      success: true,
      message: 'Prospect created successfully',
      data: populatedProspect
    });
  } catch (error) {
    console.error('Create prospect error:', error);
    res.status(400).json({
      success: false,
      message: 'Error creating prospect',
      error: error.message
    });
  }
};

// Update prospect
const updateProspect = async (req, res) => {
  try {
    const prospect = await Prospect.findById(req.params.id);

    if (!prospect) {
      return res.status(404).json({
        success: false,
        message: 'Prospect not found'
      });
    }

    // Role-based access check
    if (req.user.role === 'Sales Person' &&
        prospect.assignedTo &&
        prospect.assignedTo.toString() !== req.user._id.toString()) {
      return res.status(403).json({
        success: false,
        message: 'Access denied'
      });
    }

    const updatedProspect = await Prospect.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true, runValidators: true }
    ).populate('assignedTo', 'fullName email')
     .populate('createdBy', 'fullName email');

    res.json({
      success: true,
      message: 'Prospect updated successfully',
      data: updatedProspect
    });
  } catch (error) {
```

```
    console.error('Update prospect error:', error);
    res.status(400).json({
      success: false,
      message: 'Error updating prospect',
      error: error.message
    });
  }
};

// Delete prospect
const deleteProspect = async (req, res) => {
  try {
    const prospect = await Prospect.findById(req.params.id);

    if (!prospect) {
      return res.status(404).json({
        success: false,
        message: 'Prospect not found'
      });
    }

    // Only Admin and Manager can delete prospects
    if (!['Admin', 'Manager'].includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: 'Access denied. Only Admin and Manager can delete prospects.'
      });
    }

    await Prospect.findByIdAndDelete(req.params.id);

    res.json({
      success: true,
      message: 'Prospect deleted successfully'
    });
  } catch (error) {
    console.error('Delete prospect error:', error);
    res.status(500).json({
      success: false,
      message: 'Error deleting prospect',
      error: error.message
    });
  }
};

// Convert prospect to lead
const convertToLead = async (req, res) => {
  try {
    const prospect = await Prospect.findById(req.params.id);

    if (!prospect) {
      return res.status(404).json({
        success: false,
        message: 'Prospect not found'
      });
    }

    if (prospect.convertedToLead) {
      return res.status(400).json({
```

```
        success: false,
        message: 'Prospect already converted to lead'
      });
    }

    // Role-based access check
    if (req.user.role === 'Sales Person' &&
        prospect.assignedTo &&
        prospect.assignedTo.toString() !== req.user._id.toString()) {
      return res.status(403).json({
        success: false,
        message: 'Access denied'
      });
    }

    // Create lead from prospect data
    const leadData = prospect.convertToLead();
    const lead = new Lead(leadData);
    await lead.save();

    // Update prospect to mark as converted
    prospect.convertedToLead = true;
    prospect.leadId = lead._id;
    prospect.conversionDate = new Date();
    prospect.status = 'Converted to Lead';
    await prospect.save();

    const populatedLead = await Lead.findById(lead._id)
      .populate('assignedTo', 'fullName email')
      .populate('createdBy', 'fullName email');

    res.json({
      success: true,
      message: 'Prospect converted to lead successfully',
      data: {
        prospect: prospect,
        lead: populatedLead
      }
    });
  } catch (error) {
    console.error('Convert prospect error:', error);
    res.status(500).json({
      success: false,
      message: 'Error converting prospect to lead',
      error: error.message
    });
  }
};

// Get prospect statistics
const getProspectStats = async (req, res) => {
  try {
    const filter = {};

    // Role-based filtering
    if (req.user.role === 'Sales Person') {
      filter.assignedTo = req.user._id;
    }
```

```
    const stats = await Prospect.aggregate([
      { $match: filter },
      {
        $group: {
          _id: null,
          total: { $sum: 1 },
          new: { $sum: { $cond: [{ $eq: ['$status', 'New'] }, 1, 0] } },
          contacted: { $sum: { $cond: [{ $eq: ['$status', 'Contacted'] }, 1,
0] } },
          interested: { $sum: { $cond: [{ $eq: ['$status', 'Interested'] }, 1,
0] } },
          qualified: { $sum: { $cond: [{ $eq: ['$status', 'Qualified'] }, 1,
0] } },
          converted: { $sum: { $cond: [{ $eq: ['$status', 'Converted to Lead'] },
1, 0] } },
          lost: { $sum: { $cond: [{ $eq: ['$status', 'Lost'] }, 1, 0] } }
        }
      }
    ]);

    const sourceStats = await Prospect.aggregate([
      { $match: filter },
      {
        $group: {
          _id: '$source',
          count: { $sum: 1 }
        }
      },
      { $sort: { count: -1 } }
    ]);

    res.json({
      success: true,
      data: {
        overview: stats[0] || {
          total: 0, new: 0, contacted: 0, interested: 0,
          qualified: 0, converted: 0, lost: 0
        },
        sources: sourceStats
      }
    });
  } catch (error) {
    console.error('Get prospect stats error:', error);
    res.status(500).json({
      success: false,
      message: 'Error fetching prospect statistics',
      error: error.message
    });
  }
};

module.exports = {
  getProspects,
  getProspectById,
  createProspect,
  updateProspect,
  deleteProspect,
  convertToLead,
  getProspectStats
```

```
};
```

controllers/sales.js

```
const Sale = require('../models/Sale');
const User = require('../models/User');
const { sendPaymentConfirmationEmail, sendServiceDeliveryEmail } = require('../
services/emailService');

// @desc    Get all sales
// @route   GET /api/sales
// @access  Private
exports.getSales = async (req, res) => {
  try {
    let query;

    // Copy req.query
    const reqQuery = { ...req.query };

    // Fields to exclude
    const removeFields = ['select', 'sort', 'page', 'limit', 'full', 'nocache'];

    // Loop over removeFields and delete them from reqQuery
    removeFields.forEach(param => delete reqQuery[param]);

    // If full=true is requested, ignore any date filters to ensure all sales are
returned
    if (req.query.full === 'true') {
      // Remove any potential date filters
      delete reqQuery.date;
      delete reqQuery.createdAt;
      delete reqQuery.updatedAt;
      // Remove any date range operators
      Object.keys(reqQuery).forEach(key => {
        if (key.includes('date') || key.includes('Date') ||
key.includes('created') || key.includes('updated')) {
          delete reqQuery[key];
        }
      });
    }

    // Create query string
    let queryStr = JSON.stringify(reqQuery);

    // Create operators ($gt, $gte, etc)
    queryStr = queryStr.replace(/\b(gt|gte|lt|lte|in)\b/g, match => `$${match}`);

    const parsedQuery = JSON.parse(queryStr);

    // If user is a sales person, only show their sales
    if (req.user.role === 'Sales Person') {
      query = Sale.find({
        salesPerson: req.user.id,
        ...parsedQuery
      });
    }
    // If user is a lead person, only show sales with them as lead
    else if (req.user.role === 'Lead Person') {
```

```
  query = Sale.find({ leadPerson: req.user.id, ...parsedQuery });
}
// Admin and Manager can see all
else {
  query = Sale.find(parsedQuery);
}

// Select Fields
if (req.query.select) {
  const fields = req.query.select.split(',').join(' ');
  query = query.select(fields);
}

// Sort
if (req.query.sort) {
  const sortBy = req.query.sort.split(',').join(' ');
  query = query.sort(sortBy);
} else {
  query = query.sort('-date');
}

// Populate
query = query.populate('salesPerson leadPerson', 'fullName email');

// Pagination
const page = parseInt(req.query.page, 10) || 1;
const limit = parseInt(req.query.limit, 10) || 25;
const startIndex = (page - 1) * limit;
const endIndex = page * limit;
const total = await Sale.countDocuments();

query = query.skip(startIndex).limit(limit);

// Executing query
const sales = await query;

// Ensure currency fields are consistent for paginated sales
const processedSales = sales.map(sale => {
  const saleObj = sale.toObject();

  // Ensure currency fields are properly set
  if (!saleObj.totalCostCurrency) {
    saleObj.totalCostCurrency = saleObj.currency || 'USD';
  }
  if (!saleObj.tokenAmountCurrency) {
    saleObj.tokenAmountCurrency = saleObj.currency || 'USD';
  }
  if (!saleObj.currency) {
    saleObj.currency = saleObj.totalCostCurrency || 'USD';
  }

  return saleObj;
});

// Pagination result
const pagination = {};

if (endIndex < total) {
  pagination.next = {
```

```
        page: page + 1,
        limit
      };
    }

    if (startIndex > 0) {
      pagination.prev = {
        page: page - 1,
        limit
      };
    }

    // Check if this is a request for all sales without pagination
    if (req.query.full === 'true') {
      // Apply the same role-based filtering for full results, but ignore all
query parameters
      let fullQuery;

      if (req.user.role === 'Sales Person') {
        fullQuery = Sale.find({
          salesPerson: req.user.id
        });
      }
      else if (req.user.role === 'Lead Person') {
        fullQuery = Sale.find({ leadPerson: req.user.id });
      }
      // Only Admin and Manager can see all sales
      else if (req.user.role === 'Admin' || req.user.role === 'Manager') {
        fullQuery = Sale.find({});
      }
      else {
        // For any other role, return empty results
        return res.status(403).json({
          success: false,
          message: 'Not authorized to access sales data'
        });
      }

      const allSales = await fullQuery
        .populate('salesPerson leadPerson', 'fullName email')
        .sort('-date');

      // Ensure currency fields are consistent for all sales
      const processedSales = allSales.map(sale => {
        const saleObj = sale.toObject();

        // Ensure currency fields are properly set
        if (!saleObj.totalCostCurrency) {
          saleObj.totalCostCurrency = saleObj.currency || 'USD';
        }
        if (!saleObj.tokenAmountCurrency) {
          saleObj.tokenAmountCurrency = saleObj.currency || 'USD';
        }
        if (!saleObj.currency) {
          saleObj.currency = saleObj.totalCostCurrency || 'USD';
        }

        return saleObj;
      });
```

```
      return res.status(200).json({
        success: true,
        count: processedSales.length,
        data: processedSales
      });
    }

    res.status(200).json({
      success: true,
      count: processedSales.length,
      pagination,
      data: processedSales
    });
  } catch (err) {
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Get single sale
// @route   GET /api/sales/:id
// @access  Private
exports.getSale = async (req, res) => {
  try {
    const sale = await Sale.findById(req.params.id).populate('salesPerson
leadPerson', 'fullName email');

    if (!sale) {
      return res.status(404).json({
        success: false,
        message: 'Sale not found'
      });
    }

    // Check if user can access this sale
    if (req.user.role === 'Sales Person') {
      const salesPersonId = sale.salesPerson?._id?.toString() ||
sale.salesPerson?.toString();
      const userId = req.user._id?.toString() || req.user.id?.toString();

      if (salesPersonId !== userId) {
        return res.status(403).json({
          success: false,
          message: 'Not authorized to access this sale'
        });
      }
    }

    if (req.user.role === 'Lead Person') {
      const leadPersonId = sale.leadPerson?._id?.toString() ||
sale.leadPerson?.toString();
      const userId = req.user._id?.toString() || req.user.id?.toString();

      if (leadPersonId !== userId) {
        return res.status(403).json({
          success: false,
```

```
          message: 'Not authorized to access this sale'
        });
      }
    }

    res.status(200).json({
      success: true,
      data: sale
    });
  } catch (err) {
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Create new sale
// @route   POST /api/sales
// @access  Private
exports.createSale = async (req, res) => {
  try {
    // Add user to req.body
    req.body.createdBy = req.user.id;

    // If no salesPerson is specified, use the current user
    if (!req.body.salesPerson) {
      req.body.salesPerson = req.user.id;
    }

    // Handle reference sales for Sales Person role
    if (req.user.role === 'Sales Person' && req.body.isReference) {
      // For reference sales, if no lead person is specified, find a default one
      if (!req.body.leadPerson) {
        const leadPerson = await User.findOne({ role: 'Lead Person' });
        if (leadPerson) {
          req.body.leadPerson = leadPerson._id;
        }
      }
    }

    // Create sale
    const sale = await Sale.create(req.body);

    res.status(201).json({
      success: true,
      data: sale
    });
  } catch (err) {
    if (err.name === 'ValidationError') {
      const message = Object.values(err.errors).map(val => val.message);
      return res.status(400).json({
        success: false,
        message: message
      });
    }

    res.status(500).json({
      success: false,
```

```
      message: 'Server Error'
    });
  }
};

// @desc    Update sale
// @route   PUT /api/sales/:id
// @access  Private
exports.updateSale = async (req, res) => {
  try {
    let sale = await Sale.findById(req.params.id).populate('salesPerson
leadPerson', 'fullName email');

    if (!sale) {
      return res.status(404).json({
        success: false,
        message: 'Sale not found'
      });
    }

    // Store original values for comparison
    const originalStatus = sale.status;
    const originalTokenAmount = sale.tokenAmount;
    const originalTotalCost = sale.totalCost;

    // Check permissions
    if (req.user.role === 'Sales Person') {
      // Sales person can only update their own sales
      const salesPersonId = sale.salesPerson?._id?.toString() ||
sale.salesPerson?.toString();
      const userId = req.user._id?.toString() || req.user.id?.toString();

      if (salesPersonId !== userId) {
        return res.status(403).json({
          success: false,
          message: 'Not authorized to update this sale'
        });
      }
    } else if (req.user.role === 'Lead Person') {
      // Lead person can only update sales where they are the lead person
      const leadPersonId = sale.leadPerson?._id?.toString() ||
sale.leadPerson?.toString();
      const userId = req.user._id?.toString() || req.user.id?.toString();

      if (leadPersonId !== userId) {
        return res.status(403).json({
          success: false,
          message: 'Not authorized to update this sale'
        });
      }
    }

    // Add updatedBy field and timestamp
    req.body.updatedBy = req.user.id;
    req.body.updatedAt = new Date();

    // Ensure remarks is preserved if not provided in update
    if (req.body.remarks === undefined) {
      req.body.remarks = sale.remarks || '';
```

```
    }

    // Update the sale with all fields including remarks
    sale = await Sale.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
      runValidators: true
    }).populate('salesPerson leadPerson', 'fullName email');

    // Email logic - send emails when certain conditions are met
    let emailResults = [];

    // 1. Send payment confirmation email when token amount is updated (and
customer has email)
    if (req.body.tokenAmount && req.body.tokenAmount !== originalTokenAmount &&
req.body.tokenAmount > 0) {
      if (sale.email) {
        try {
          const paymentResult = await sendPaymentConfirmationEmail(sale,
sale.salesPerson?.email);
          emailResults.push({
            type: 'payment_confirmation',
            success: paymentResult.success,
            message: paymentResult.success ? 'Payment confirmation email sent' :
paymentResult.message
          });
        } catch (emailError) {
          emailResults.push({
            type: 'payment_confirmation',
            success: false,
            message: 'Failed to send payment confirmation email'
          });
        }
      } else {
        emailResults.push({
          type: 'payment_confirmation',
          success: false,
          message: 'Email unavailable - cannot send payment confirmation'
        });
      }
    }

    // 2. Send service delivery email when status changes to "Completed"
    if (req.body.status === 'Completed' && originalStatus !== 'Completed') {
      if (sale.email) {
        try {
          const deliveryResult = await sendServiceDeliveryEmail(sale,
sale.salesPerson?.email);
          emailResults.push({
            type: 'service_delivery',
            success: deliveryResult.success,
            message: deliveryResult.success ? 'Service delivery email sent' :
deliveryResult.message
          });
        } catch (emailError) {
          emailResults.push({
            type: 'service_delivery',
            success: false,
            message: 'Failed to send service delivery email'
          });
```

```
        }
      } else {
        emailResults.push({
          type: 'service_delivery',
          success: false,
          message: 'Email unavailable - cannot send service delivery confirmation'
        });
      }
    }

    res.status(200).json({
      success: true,
      data: sale,
      emailNotifications: emailResults.length > 0 ? emailResults : undefined
    });
  } catch (err) {
    if (err.name === 'ValidationError') {
      const message = Object.values(err.errors).map(val => val.message);
      return res.status(400).json({
        success: false,
        message: message
      });
    }

    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Delete sale
// @route   DELETE /api/sales/:id
// @access  Private
exports.deleteSale = async (req, res) => {
  try {
    const sale = await Sale.findById(req.params.id);

    if (!sale) {
      return res.status(404).json({
        success: false,
        message: 'Sale not found'
      });
    }

    // Check permissions - only sales person who created it, manager, or admin
can delete
    if (req.user.role === 'Sales Person') {
      const salesPersonId = sale.salesPerson?._id?.toString() ||
sale.salesPerson?.toString();
      const userId = req.user._id?.toString() || req.user.id?.toString();

      if (salesPersonId !== userId) {
        return res.status(403).json({
          success: false,
          message: 'Not authorized to delete this sale'
        });
      }
    } else if (req.user.role === 'Lead Person') {
```

```
      // Lead persons cannot delete sales
      return res.status(403).json({
        success: false,
        message: 'Lead persons cannot delete sales'
      });
    }

    await sale.deleteOne();

    res.status(200).json({
      success: true,
      data: {}
    });
  } catch (err) {
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Get sales count
// @route   GET /api/sales/count
// @access  Private
exports.getSalesCount = async (req, res) => {
  try {
    let count;

    // If user is a sales person, only count their sales
    if (req.user.role === 'Sales Person') {
      count = await Sale.countDocuments({
        salesPerson: req.user.id,
        isLeadPersonSale: { $ne: true } // Exclude lead person sales
      });
    }
    // If user is a lead person, only count sales with them as lead
    else if (req.user.role === 'Lead Person') {
      count = await Sale.countDocuments({ leadPerson: req.user.id });
    }
    // Admin and Manager can see all
    else {
      count = await Sale.countDocuments();
    }

    res.status(200).json({
      success: true,
      count
    });
  } catch (err) {
    res.status(500).json({
      success: false,
      message: 'Server Error'
    });
  }
};

// @desc    Import sales from CSV
// @route   POST /api/sales/import
// @access  Private (Admin only)
```

```
exports.importSales = async (req, res) => {
  try {
    console.log('=== IMPORT SALES REQUEST ===');
    console.log('Request body:', JSON.stringify(req.body, null, 2));
    console.log('User:', req.user.fullName, req.user.role);

    // Handle both direct sales array and nested data structure
    let sales = req.body.sales;
    if (!sales && req.body.data && req.body.data.sales) {
      sales = req.body.data.sales;
      console.log('Found sales in nested data structure');
    }

    if (!sales || !Array.isArray(sales) || sales.length === 0) {
      return res.status(400).json({
        success: false,
        message: 'No sales data provided or invalid format'
      });
    }

    console.log(`Importing ${sales.length} sales from CSV...`);

    // Map CSV column names to our database fields
    const mappedSales = sales.map(sale => {
      return {
        customerName: sale.CustomerName || sale['Customer Name'] ||
sale.customerName || '',
        email: sale.Email || sale.email || '',
        contactNumber: sale.ContactNumber || sale['Contact Number'] ||
sale.contactNumber || sale.Phone || sale.phone || '',
        countryCode: sale.CountryCode || sale['Country Code'] || sale.countryCode
|| '+1',
        country: sale.Country || sale.country || '',
        course: sale.Course || sale.course || sale.Product || sale.product || '',
        amount: parseFloat(sale.Amount || sale.amount || sale.Price || sale.price
|| 0),
        currency: sale.Currency || sale.currency || 'USD',
        status: sale.Status || sale.status || 'Pending',
        paymentMethod: sale.PaymentMethod || sale['Payment Method'] ||
sale.paymentMethod || 'Unknown',
        date: sale.Date || sale.date ? new Date(sale.Date || sale.date) : new
Date(),
        remarks: sale.Remarks || sale.remarks || '',
        // Set created by to the current user (admin)
        createdBy: req.user.id,
        salesPerson: req.user.id // Default to current user, can be updated later
      };
    });

    // Validate the mapped data
    const validSales = mappedSales.filter(sale =>
      sale.customerName && sale.contactNumber && sale.course && sale.amount > 0
    );

    if (validSales.length === 0) {
      return res.status(400).json({
        success: false,
        message: 'No valid sales found in the imported data'
      });
```

```
    }

    console.log(`Found ${validSales.length} valid sales out of ${sales.length}`);

    // Insert the sales into the database
    const results = await Sale.insertMany(validSales, {
      ordered: false // Continue processing even if some documents have errors
    });

    console.log(`Successfully imported ${results.length} sales`);

    res.status(201).json({
      success: true,
      count: results.length,
      data: results,
      errorCount: sales.length - results.length
    });
  } catch (err) {
    console.error('Sales import error:', err);
    res.status(400).json({
      success: false,
      message: err.message
    });
  }
};
```

controllers/taskController.js

```
const Task = require('../models/Task');
const User = require('../models/User');
const Lead = require('../models/Lead');
const ErrorResponse = require('../utils/errorResponse');
const asyncHandler = require('../middleware/async');

// Common customer field selection for populating lead data
const CUSTOMER_SELECT_FIELDS = 'name NAME email E-MAIL contactNumber phone MOBILE
NUMBER country customerName fullName course COURSE';

// @desc    Create new task
// @route   POST /api/tasks
// @access  Private
exports.createTask = asyncHandler(async (req, res, next) => {
  // Add creator as sales person
  req.body.salesPerson = req.user.id;

  // Validate that either customer or manualCustomer is provided
  if (!req.body.customer && !req.body.manualCustomer) {
    return next(new ErrorResponse('Either customer ID or manual customer details
must be provided', 400));
  }

  // If manualCustomer is provided, validate required fields
  if (req.body.manualCustomer) {
    const { name, contactNumber, course } = req.body.manualCustomer;
    if (!name || !contactNumber || !course) {
      return next(new ErrorResponse('Manual customer requires name, contact
number, and course', 400));
    }
```

```
  }

  const task = await Task.create(req.body);

  res.status(201).json({
    success: true,
    data: task
  });
});

// @desc    Get all tasks
// @route   GET /api/tasks
// @access  Private
exports.getTasks = asyncHandler(async (req, res, next) => {
  // For non-admin users, only show their own tasks
  let query;

  // If user is not admin or manager, only show their tasks
  if (req.user.role !== 'Admin' && req.user.role !== 'Manager') {
    query = Task.find({ salesPerson: req.user.id });
  } else {
    query = Task.find();
  }

  // Execute query with populated fields
  const tasks = await query
    .populate({
      path: 'salesPerson',
      select: 'fullName email'
    })
    .populate({
      path: 'customer',
      select: CUSTOMER_SELECT_FIELDS
    });

  // Process the tasks to handle the customer display consistently
  const processedTasks = tasks.map(task => {
    const taskObj = task.toObject();

    // If this is a manual customer entry (no customer ID but has manualCustomer
data)
    if (!taskObj.customer && taskObj.manualCustomer) {
      // Create a consistent customer field from manualCustomer for the frontend
      taskObj.customer = {
        _id: 'manual',
        name: taskObj.manualCustomer.name,
        email: taskObj.manualCustomer.email,
        contactNumber: taskObj.manualCustomer.contactNumber,
        course: taskObj.manualCustomer.course,
        isManualEntry: true
      };
    }

    return taskObj;
  });

  res.status(200).json({
    success: true,
    count: processedTasks.length,
```

```
    data: processedTasks
  });
});

// @desc    Get single task
// @route   GET /api/tasks/:id
// @access  Private
exports.getTask = asyncHandler(async (req, res, next) => {
  const task = await Task.findById(req.params.id)
    .populate({
      path: 'salesPerson',
      select: 'fullName email'
    })
    .populate({
      path: 'customer',
      select: CUSTOMER_SELECT_FIELDS
    });

  if (!task) {
    return next(new ErrorResponse(`Task not found with id of ${req.params.id}`,
404));
  }

  // Allow any user to view tasks (removing permission restrictions)
  // This enables everyone to see all scheduled exams

  // Process the task to handle manual customer consistently
  const taskObj = task.toObject();

  // If this is a manual customer entry (no customer ID but has manualCustomer
data)
  if (!taskObj.customer && taskObj.manualCustomer) {
    // Create a consistent customer field from manualCustomer for the frontend
    taskObj.customer = {
      _id: 'manual',
      name: taskObj.manualCustomer.name,
      email: taskObj.manualCustomer.email,
      contactNumber: taskObj.manualCustomer.contactNumber,
      course: taskObj.manualCustomer.course,
      isManualEntry: true
    };
  }

  res.status(200).json({
    success: true,
    data: taskObj
  });
});

// @desc    Update task
// @route   PUT /api/tasks/:id
// @access  Private
exports.updateTask = asyncHandler(async (req, res, next) => {
  let task = await Task.findById(req.params.id);

  if (!task) {
    return next(new ErrorResponse(`Task not found with id of ${req.params.id}`,
404));
  }
```

```
  // Allow any user to update tasks (removing permission restrictions)
  // This enables team collaboration on exam scheduling

  // Validate that either customer or manualCustomer is provided
  if (req.body.customer === '' && !req.body.manualCustomer) {
    return next(new ErrorResponse('Either customer ID or manual customer details
must be provided', 400));
  }

  // If manualCustomer is provided, validate required fields
  if (req.body.manualCustomer) {
    const { name, contactNumber, course } = req.body.manualCustomer;
    if (!name || !contactNumber || !course) {
      return next(new ErrorResponse('Manual customer requires name, contact
number, and course', 400));
    }
  }

  req.body.updatedAt = Date.now();

  task = await Task.findByIdAndUpdate(req.params.id, req.body, {
    new: true,
    runValidators: true
  });

  // Process the task to handle manual customer consistently
  const taskObj = task.toObject();

  // If this is a manual customer entry (no customer ID but has manualCustomer
data)
  if (!taskObj.customer && taskObj.manualCustomer) {
    // Create a consistent customer field from manualCustomer for the frontend
    taskObj.customer = {
      _id: 'manual',
      name: taskObj.manualCustomer.name,
      email: taskObj.manualCustomer.email,
      contactNumber: taskObj.manualCustomer.contactNumber,
      course: taskObj.manualCustomer.course,
      isManualEntry: true
    };
  }

  res.status(200).json({
    success: true,
    data: taskObj
  });
});

// @desc    Delete task
// @route   DELETE /api/tasks/:id
// @access  Private
exports.deleteTask = asyncHandler(async (req, res, next) => {
  const task = await Task.findById(req.params.id);

  if (!task) {
    return next(new ErrorResponse(`Task not found with id of ${req.params.id}`,
404));
  }
```

```
  // Only allow task owner, admin, or manager to delete tasks
  if (task.salesPerson.toString() !== req.user.id &&
      req.user.role !== 'Admin' &&
      req.user.role !== 'Manager') {
    return next(new ErrorResponse(`User ${req.user.id} is not authorized to
delete this task`, 401));
  }

  await task.deleteOne();

  res.status(200).json({
    success: true,
    data: {}
  });
});
```

debug-sales.js

```
const mongoose = require('mongoose');
const Sale = require('./models/Sale');

async function debugSales() {
  try {
    await mongoose.connect(process.env.MONGO_URI || 'mongodb://localhost:27017/
crm');
    console.log('Connected to MongoDB');

    // Get all sales without any filters
    const allSales = await Sale.find({});
    console.log('Total sales in database (no filter):', allSales.length);

    // Group by different criteria to understand the data
    const salesByCreatedBy = {};
    const salesByLeadPerson = {};
    const salesByIsLeadPersonSale = { true: 0, false: 0, undefined: 0 };

    allSales.forEach(sale => {
      // Group by createdBy
      const createdBy = sale.createdBy ? sale.createdBy.toString() : 'null';
      salesByCreatedBy[createdBy] = (salesByCreatedBy[createdBy] || 0) + 1;

      // Group by leadPerson
      const leadPerson = sale.leadPerson ? sale.leadPerson.toString() : 'null';
      salesByLeadPerson[leadPerson] = (salesByLeadPerson[leadPerson] || 0) + 1;

      // Group by isLeadPersonSale
      const isLeadPersonSale = sale.isLeadPersonSale;
      if (isLeadPersonSale === true) salesByIsLeadPersonSale.true++;
      else if (isLeadPersonSale === false) salesByIsLeadPersonSale.false++;
      else salesByIsLeadPersonSale.undefined++;
    });

    console.log('\nSales by createdBy:');
    Object.entries(salesByCreatedBy).forEach(([key, count]) => {
      console.log(`  ${key}: ${count}`);
    });
```

```
    console.log('\nSales by leadPerson:');
    Object.entries(salesByLeadPerson).forEach(([key, count]) => {
      console.log(`  ${key}: ${count}`);
    });

    console.log('\nSales by isLeadPersonSale:');
    console.log(`  true: ${salesByIsLeadPersonSale.true}`);
    console.log(`  false: ${salesByIsLeadPersonSale.false}`);
    console.log(`  undefined: ${salesByIsLeadPersonSale.undefined}`);

    // Check what query would return for admin user
    const adminUserId = '6835608d5d2b8c84dcbc9666';
    console.log(`\nChecking what admin user ${adminUserId} would see:`);

    // This simulates the query from the controller for Admin role
    const adminQuery = Sale.find({});
    const adminSales = await adminQuery;
    console.log('Admin query result count:', adminSales.length);

    // Check recent sales
    const recentSales = await Sale.find({}).sort({ date: -1 }).limit(5);
    console.log('\nMost recent 5 sales:');
    recentSales.forEach((sale, index) => {
      console.log(`  ${index + 1}. ${sale.customerName} - ${sale.course} -
${sale.date} - isLeadPersonSale: ${sale.isLeadPersonSale}`);
    });

    process.exit(0);
  } catch (error) {
    console.error('Error:', error.message);
    process.exit(1);
  }
}

debugSales();
```

DEPLOYMENT_CHECKLIST.md

```
# Ø=Þ€ Production Deployment Checklist

## Pre-Deployment Setup

### 1. Choose Storage Strategy & þ
- [ ] **Option A: AWS S3** (Recommended for scale)
  - [ ] Create AWS account
  - [ ] Create S3 bucket: `your-crm-documents-prod`
  - [ ] Configure IAM user with S3 permissions
  - [ ] Note down: Access Key ID & Secret Access Key

- [ ] **Option B: VPS Storage** (Cost-effective)
  - [ ] Provision VPS (minimum 20GB storage)
  - [ ] Create directory: `/var/www/crm/uploads/`
  - [ ] Set proper permissions: `chmod 755`
  - [ ] Setup backup strategy

### 2. Environment Configuration & þ
- [ ] Create `.env.production` file
- [ ] Set `NODE_ENV=production`
```

- [ ] Configure production database URL
- [ ] Set strong JWT secret (minimum 32 characters)
- [ ] Configure storage type and credentials

3. Security Setup & þ
- [ ] Update CORS origins for production domain
- [ ] Configure rate limiting
- [ ] Set up HTTPS/SSL certificate
- [ ] Review file upload limits
- [ ] Test authentication endpoints

Deployment Steps

4. Code Preparation & þ
- [ ] Update storage configuration in `config/storage.js`
- [ ] Ensure `.gitignore` excludes sensitive files
- [ ] Run security audit: `npm audit`
- [ ] Build production bundle
- [ ] Test in staging environment

5. Server Setup & þ
- [ ] Install Node.js 18+ on production server
- [ ] Install PM2 for process management: `npm install -g pm2`
- [ ] Clone repository to server
- [ ] Install dependencies: `npm ci --production`
- [ ] Create uploads directory structure

6. Database Migration & þ
- [ ] Setup production MongoDB instance
- [ ] Import development data (if needed)
- [ ] Update database connection strings
- [ ] Test database connectivity

7. File Migration & þ
If migrating from development:
- [ ] Archive development uploads: `tar -czf dev-uploads.tar.gz uploads/`
- [ ] Transfer to production storage
- [ ] Update database file paths
- [ ] Verify file access through application

Post-Deployment Verification

8. Functionality Testing & þ
- [ ] Test user registration/login
- [ ] Test file upload functionality
- [ ] Test file download/view
- [ ] Test document management features
- [ ] Verify employee data access

9. Performance & Monitoring & þ
- [ ] Setup application monitoring (PM2 logs)
- [ ] Configure storage monitoring
- [ ] Test backup procedures
- [ ] Monitor memory and CPU usage
- [ ] Test application under load

10. Security Verification & þ
- [ ] Verify no sensitive files in Git history
- [ ] Test file access permissions

- [ ] Verify HTTPS enforcement
- [ ] Test rate limiting
- [ ] Check for exposed sensitive endpoints

Ongoing Maintenance

11. Backup Strategy & þ
- [ ] **For S3**: Enable versioning and cross-region replication
- [ ] **For Local**: Setup automated daily backups
- [ ] Test restore procedures
- [ ] Document backup locations

12. Monitoring & Alerts & þ
- [ ] Setup uptime monitoring
- [ ] Configure storage usage alerts
- [ ] Monitor application errors
- [ ] Setup email notifications for critical issues

Production Environment Variables Template

```bash
# Basic Configuration
NODE_ENV=production
PORT=8080
BASE_URL=https://yourcrm.com

# Database
DB_URI=mongodb://username:password@cluster.mongodb.net/crm_production

# JWT
JWT_SECRET=your-super-secure-32-character-secret
JWT_EXPIRE=30d

# Storage (choose one option)

# Option 1: AWS S3
STORAGE_TYPE=s3
AWS_S3_BUCKET=your-crm-documents-prod
AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=AKIA...
AWS_SECRET_ACCESS_KEY=...

# Option 2: Local Storage
STORAGE_TYPE=local
UPLOAD_PATH=/var/www/crm/uploads/documents
PUBLIC_PATH=/uploads/documents

# Security
CORS_ORIGIN=https://yourcrm.com
RATE_LIMIT_MAX=100
```

Common Issues & Solutions

Issue: Files not accessible after deployment
Solution:
- Check file permissions (755 for directories, 644 for files)
- Verify storage configuration
- Check BASE_URL setting

```
### Issue: Upload fails in production
**Solution**:
- Verify upload directory exists and is writable
- Check file size limits
- Ensure adequate disk space

### Issue: S3 access denied
**Solution**:
- Verify AWS credentials
- Check S3 bucket permissions
- Ensure IAM user has proper policies

## Emergency Rollback Plan

### If deployment fails:
1. [ ] Revert to previous code version
2. [ ] Restore database backup
3. [ ] Restore file storage backup
4. [ ] Update DNS if needed
5. [ ] Notify users of any issues

---

**Deployment Status**: #ó In Progress / '  Complete / 'L Failed

**Notes**:
_Document any specific configurations or issues encountered during deployment_
```

EMAIL_SETUP.md

```
# Email Notification Setup Guide

## Overview
The CRM now automatically sends email notifications when sales are updated:

1. **Payment Confirmation Email** - Sent when token amount is updated
2. **Service Delivery Email** - Sent when status changes to "Completed"

## Multi-Provider Email Support

The system automatically detects the email provider based on the sales person's
email domain and uses the appropriate SMTP configuration:

### Supported Providers

#### 1. **Hostinger Email** (traincapetech.in)
```env
HOSTINGER_EMAIL_PASS=your-hostinger-email-password
```

2. **Gmail** (gmail.com)
```env
GMAIL_APP_PASS=your-gmail-app-password
```

3. **Outlook/Hotmail** (outlook.com, hotmail.com, live.com)
```env
```
```

````
OUTLOOK_EMAIL_PASS=your-outlook-password
```

#### 4. **Yahoo** (yahoo.com, yahoo.in)
```env
YAHOO_EMAIL_PASS=your-yahoo-password
```

#### 5. **Generic SMTP** (fallback for other providers)
```env
SMTP_HOST=your-smtp-server.com
SMTP_PORT=587
EMAIL_PASS=your-email-password
```

## Environment Variables Setup

### For Hostinger (Primary - traincapetech.in emails)
```env
HOSTINGER_EMAIL_PASS=your_hostinger_email_password
```

### For Multiple Providers (Optional)
```env
GMAIL_APP_PASS=gmail_app_password
OUTLOOK_EMAIL_PASS=outlook_password
YAHOO_EMAIL_PASS=yahoo_password
EMAIL_PASS=fallback_password
```

## How It Works

1. **Sales person updates sale** (e.g., Saurav updates token amount)
2. **System detects email domain** (saurav@traincapetech.in !' Hostinger)
3. **Uses appropriate SMTP config** (Hostinger SMTP settings)
4. **Sends email FROM sales person's email** (saurav@traincapetech.in)
5. **Customer receives email** from the actual sales person
6. **Sales person gets CC copy**

## Email Provider Configurations

### Hostinger Setup
- **SMTP Server**: smtp.hostinger.com
- **Port**: 587
- **Security**: STARTTLS
- **Authentication**: Email + Password

### Gmail Setup
1. Enable 2-Factor Authentication
2. Generate App Password:
   - Google Account !' Security !' 2-Step Verification !' App passwords
   - Generate password for "Mail"
   - Use as GMAIL_APP_PASS

### Outlook Setup
- Use regular email password
- Or generate app password if 2FA enabled

### Yahoo Setup
````
````

- Generate app password:
  - Yahoo Account !’ Security !’ Generate app password
  - Use as YAHOO_EMAIL_PASS

Email Templates

Payment Confirmation
- **Trigger**: When `tokenAmount` is updated in a sale
- **From**: Sales person's email (e.g., saurav@traincapetech.in)
- **To**: Customer email
- **CC**: Sales person email
- **Content**: Dynamic payment details with pending amount

Service Delivery
- **Trigger**: When sale `status` changes to "Completed"
- **From**: Sales person's email
- **To**: Customer email
- **CC**: Sales person email
- **Content**: Service completion confirmation

Frontend Notifications

Users see real-time toast messages:
- ' "Payment confirmation email sent"
- ' "Service delivery email sent"
- & þ "Customer email not available"
- & þ "Sales person email not available"
- 'L "Email authentication failed - check email credentials"

Testing Scenarios

Test 1: Hostinger Email (Primary)
1. Sales person: saurav@traincapetech.in
2. Update sale token amount
3. Should use Hostinger SMTP
4. Customer receives email from saurav@traincapetech.in

Test 2: Gmail Sales Person
1. Sales person: someone@gmail.com
2. Update sale status to "Completed"
3. Should use Gmail SMTP
4. Customer receives email from someone@gmail.com

Test 3: Mixed Providers
1. Different sales persons with different email providers
2. Each should use their respective SMTP settings
3. All emails should work correctly

Troubleshooting

Common Issues

"Email authentication failed"
- Check the correct environment variable for the email provider
- Verify password/app password is correct
- Ensure 2FA app passwords are used where required

"Email server connection failed"
- Check internet connectivity

```
- Verify SMTP server settings
- Check if email provider blocks SMTP access

**"Sales person email not available"**
- Ensure sales person has email field filled in user profile
- Check user data in database

**"Customer email not available"**
- Ensure customer email field is filled in sale record
- Verify email format is valid

### Provider-Specific Issues

**Hostinger**
- Ensure email account exists in Hostinger panel
- Check if SMTP is enabled for the email account
- Verify password is correct

**Gmail**
- Must use App Password, not regular password
- 2FA must be enabled
- Check if "Less secure app access" is disabled (should be)

**Outlook**
- May need app password if 2FA enabled
- Check if account is not locked

## Security Best Practices

1. **Use App Passwords** where available (Gmail, Yahoo)
2. **Store passwords securely** in environment variables
3. **Enable 2FA** on email accounts
4. **Rotate passwords** regularly
5. **Monitor email logs** for suspicious activity

## Production Deployment

### Environment Variables Checklist
- [ ] HOSTINGER_EMAIL_PASS (primary)
- [ ] GMAIL_APP_PASS (if using Gmail)
- [ ] OUTLOOK_EMAIL_PASS (if using Outlook)
- [ ] YAHOO_EMAIL_PASS (if using Yahoo)
- [ ] EMAIL_PASS (fallback)

### Testing Checklist
- [ ] Test with Hostinger email (traincapetech.in)
- [ ] Test with different customer email providers
- [ ] Verify dynamic content is correct
- [ ] Check CC functionality
- [ ] Test error handling (invalid emails, etc.)
```

fix-currency-data.js

```
const mongoose = require('mongoose');
const Sale = require('./models/Sale');
require('dotenv').config();

const fixCurrencyData = async () => {
```

```
try {
  // Connect to MongoDB
  await mongoose.connect(process.env.MONGO_URI);
  console.log('Connected to MongoDB');

  // Find the sale for Pradeep with the specific amounts
  const sale = await Sale.findOne({
    customerName: 'Pradeep',
    totalCost: 15000,
    tokenAmount: 5000,
    totalCostCurrency: 'USD'
  });

  if (sale) {
    console.log('Found sale:', {
      id: sale._id,
      customer: sale.customerName,
      totalCost: sale.totalCost,
      totalCostCurrency: sale.totalCostCurrency,
      tokenAmount: sale.tokenAmount,
      tokenAmountCurrency: sale.tokenAmountCurrency
    });

    // Update the currency fields to INR
    sale.totalCostCurrency = 'INR';
    sale.tokenAmountCurrency = 'INR';
    sale.currency = 'INR';

    await sale.save();
    console.log('' Successfully updated currency to INR');

    // Verify the update
    const updatedSale = await Sale.findById(sale._id);
    console.log('Updated sale:', {
      id: updatedSale._id,
      customer: updatedSale.customerName,
      totalCost: updatedSale.totalCost,
      totalCostCurrency: updatedSale.totalCostCurrency,
      tokenAmount: updatedSale.tokenAmount,
      tokenAmountCurrency: updatedSale.tokenAmountCurrency,
      currency: updatedSale.currency
    });
  } else {
    console.log(''L Sale not found with the specified criteria');

    // Let's search more broadly
    const allPradeepSales = await Sale.find({ customerName: 'Pradeep' });
    console.log(`Found ${allPradeepSales.length} sales for Pradeep:`);
    allPradeepSales.forEach(sale => {
      console.log({
        id: sale._id,
        customer: sale.customerName,
        totalCost: sale.totalCost,
        totalCostCurrency: sale.totalCostCurrency,
        tokenAmount: sale.tokenAmount,
        tokenAmountCurrency: sale.tokenAmountCurrency,
        date: sale.date
      });
    });
```

```
    }

  } catch (error) {
    console.error('Error fixing currency data:', error);
  } finally {
    await mongoose.disconnect();
    console.log('Disconnected from MongoDB');
  }
};

// Run the fix
fixCurrencyData();
```

middleware/async.js

```
const asyncHandler = fn => (req, res, next) =>
  Promise.resolve(fn(req, res, next)).catch(next);

module.exports = asyncHandler;
```

middleware/auth.js

```
const jwt = require('jsonwebtoken');
const User = require('../models/User');

// Protect routes
const protect = async (req, res, next) => {
  let token;

  if (req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')) {
    token = req.headers.authorization.split(' ')[1];
  }

  // Make sure token exists
  if (!token) {
    return res.status(401).json({
      success: false,
      message: 'Not authorized to access this route'
    });
  }

  try {
    // Verify token
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Get user from token
    const user = await User.findById(decoded.id);

    if (!user) {
      return res.status(401).json({
        success: false,
        message: 'No user found with this token'
      });
    }

    req.user = user;
    next();
```

```
  } catch (err) {
    return res.status(401).json({
      success: false,
      message: 'Not authorized to access this route'
    });
  }
};

// Grant access to specific roles
const authorize = (...roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({
        success: false,
        message: `User role ${req.user.role} is not authorized to access this
route`
      });
    }
    next();
  };
};

module.exports = { protect, authorize };
```

middleware/cors.js

```
const cors = require('cors');

const allowedOrigins = [
  'http://localhost:5173',
  'http://127.0.0.1:5173',
  'https://traincapecrm.traincapetech.in',
  'http://traincapecrm.traincapetech.in',
  'https://crm-backend-o36v.onrender.com',
  // Add any additional origins here
];

// CORS middleware with detailed logging for debugging
const corsMiddleware = cors({
  origin: function (origin, callback) {
    // Log the origin for debugging
    console.log('Request origin:', origin);

    // Allow requests with no origin (like mobile apps or curl requests)
    if (!origin) {
      console.log('Request has no origin, allowing');
      return callback(null, true);
    }

    // Check if origin is allowed
    if (allowedOrigins.includes(origin)) {
      console.log('Origin allowed:', origin);
      return callback(null, true);
    }

    // For development, allow all origins
    if (process.env.NODE_ENV === 'development') {
      console.log('Development mode - allowing all origins');
```

```
      return callback(null, true);
    }

    // For production but in debugging mode, allow all origins temporarily
    const debugCORS = process.env.DEBUG_CORS === 'true';
    if (debugCORS) {
      console.log('Debug CORS enabled - allowing origin:', origin);
      return callback(null, true);
    }

    // Otherwise, block the request
    console.log('CORS blocked request from:', origin);
    return callback(new Error('Not allowed by CORS'));
  },
  methods: ['GET', 'HEAD', 'PUT', 'PATCH', 'POST', 'DELETE', 'OPTIONS'],
  credentials: true,
  optionsSuccessStatus: 204,
  allowedHeaders: [
    'Content-Type',
    'Authorization',
    'Origin',
    'X-Requested-With',
    'Accept',
    'Access-Control-Allow-Origin',
    'Access-Control-Allow-Headers',
    'Access-Control-Allow-Methods'
  ],
  exposedHeaders: ['Content-Length', 'X-Content-Type-Options']
});

// Secondary middleware to ensure headers are always set
const ensureCorsHeaders = (req, res, next) => {
  const origin = req.headers.origin;

  // Always set these headers for allowed origins or in development
  if (!origin || allowedOrigins.includes(origin) || process.env.NODE_ENV ===
'development' || process.env.DEBUG_CORS === 'true') {
    res.header('Access-Control-Allow-Origin', origin || '*');
    res.header('Access-Control-Allow-Methods',
'GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS');
    res.header('Access-Control-Allow-Headers', 'Content-Type, Authorization,
Origin, X-Requested-With, Accept');
    res.header('Access-Control-Allow-Credentials', 'true');
  }

  // Handle preflight requests
  if (req.method === 'OPTIONS') {
    console.log('Handling OPTIONS preflight request for:', req.url);
    return res.status(204).send();
  }

  next();
};

module.exports = {
  corsMiddleware,
  ensureCorsHeaders,
  handleOptions: (req, res) => {
    console.log('Explicit OPTIONS handler called for:', req.url);
```

```
    const origin = req.headers.origin;

    if (!origin || allowedOrigins.includes(origin) || process.env.NODE_ENV ===
'development' || process.env.DEBUG_CORS === 'true') {
      res.header('Access-Control-Allow-Origin', origin || '*');
      res.header('Access-Control-Allow-Methods',
'GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS');
      res.header('Access-Control-Allow-Headers', 'Content-Type, Authorization,
Origin, X-Requested-With, Accept');
      res.header('Access-Control-Allow-Credentials', 'true');
    }

    res.status(204).send();
  }
};
```

middleware/ipFilter.js

```
const allowedIPs = ['122.176.88.105']; // Replace with your office's IP

const ipFilter = (req, res, next) => {
  const clientIP = req.headers['x-forwarded-for'] || req.socket.remoteAddress;

  if (allowedIPs.includes(clientIP)) {
    next();
  } else {
    return res.status(403).json({ message: 'Access denied. Office Wi-Fi only.' });
  }
};

module.exports = ipFilter;
```

models/Attendance.js

```
const mongoose = require('mongoose');

const attendanceSchema = new mongoose.Schema({
  employeeId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Employee',
    required: true
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  date: {
    type: Date,
    required: true
  },
  checkIn: {
    type: Date,
    required: true
  },
  checkOut: {
    type: Date
  },
```

```
  totalHours: {
    type: Number,
    default: 0
  },
  status: {
    type: String,
    enum: ['PRESENT', 'ABSENT', 'HALF_DAY', 'LATE', 'EARLY_LEAVE'],
    default: 'PRESENT'
  },
  isOvertime: {
    type: Boolean,
    default: false
  },
  overtimeHours: {
    type: Number,
    default: 0
  },
  notes: {
    type: String,
    maxlength: 500
  },
  approvedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }
}, {
  timestamps: true
});

// Create compound index for employee and date
attendanceSchema.index({ employeeId: 1, date: 1 }, { unique: true });

// Virtual for formatted date
attendanceSchema.virtual('formattedDate').get(function() {
  return this.date.toDateString();
});

// Method to calculate total hours
attendanceSchema.methods.calculateTotalHours = function() {
  if (this.checkIn && this.checkOut) {
    const diff = this.checkOut - this.checkIn;
    this.totalHours = diff / (1000 * 60 * 60); // Convert to hours

    // Standard working hours (8 hours)
    const standardHours = 8;
    if (this.totalHours > standardHours) {
      this.isOvertime = true;
      this.overtimeHours = this.totalHours - standardHours;
    }

    // Determine status based on hours
    if (this.totalHours < 4) {
      this.status = 'HALF_DAY';
    } else if (this.totalHours < 7) {
      this.status = 'EARLY_LEAVE';
    } else {
      this.status = 'PRESENT';
    }
  }
```

```
};

// Pre-save middleware to calculate hours
attendanceSchema.pre('save', function(next) {
  if (this.checkIn && this.checkOut) {
    this.calculateTotalHours();
  }
  next();
});

module.exports = mongoose.model('Attendance', attendanceSchema);
```

models/ChatMessage.js

```
const mongoose = require('mongoose');

const chatMessageSchema = new mongoose.Schema({
  chatId: {
    type: String,
    required: true,
    index: true
  },
  senderId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  recipientId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  content: {
    type: String,
    required: true,
    trim: true
  },
  messageType: {
    type: String,
    enum: ['text', 'image', 'file'],
    default: 'text'
  },
  isRead: {
    type: Boolean,
    default: false
  },
  timestamp: {
    type: Date,
    default: Date.now
  }
}, {
  timestamps: true
});

// Index for efficient querying
chatMessageSchema.index({ chatId: 1, timestamp: 1 });
chatMessageSchema.index({ senderId: 1, recipientId: 1 });
```

```
module.exports = mongoose.model('ChatMessage', chatMessageSchema);
```

models/ChatRoom.js

```
const mongoose = require('mongoose');

const chatRoomSchema = new mongoose.Schema({
  chatId: {
    type: String,
    required: true,
    unique: true,
    index: true
  },
  senderId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  recipientId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  lastMessage: {
    type: String,
    default: ''
  },
  lastMessageTime: {
    type: Date,
    default: Date.now
  },
  unreadCount: {
    senderId: {
      type: Number,
      default: 0
    },
    recipientId: {
      type: Number,
      default: 0
    }
  }
}, {
  timestamps: true
});

// Ensure unique chat rooms between two users
chatRoomSchema.index({ senderId: 1, recipientId: 1 }, { unique: true });

module.exports = mongoose.model('ChatRoom', chatRoomSchema);
```

models/Department.js

```
const mongoose = require('mongoose');

const departmentSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please add a department name'],
```

```
    unique: true,
    trim: true
  },
  description: {
    type: String,
    trim: true
  },
  isActive: {
    type: Boolean,
    default: true
  }
}, {
  timestamps: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});

// Virtual for employee count
departmentSchema.virtual('employeeCount', {
  ref: 'Employee',
  localField: '_id',
  foreignField: 'department',
  count: true
});

module.exports = mongoose.model('Department', departmentSchema);
```

models/Employee.js

```
const mongoose = require('mongoose');

const employeeSchema = new mongoose.Schema({
  fullName: {
    type: String,
    required: [true, 'Please add a full name'],
    trim: true
  },
  email: {
    type: String,
    required: [true, 'Please add an email'],
    unique: true,
    lowercase: true,
    trim: true
  },
  phoneNumber: {
    type: String,
    trim: true
  },
  whatsappNumber: {
    type: String,
    trim: true
  },
  linkedInUrl: {
    type: String,
    trim: true
  },
  currentAddress: {
    type: String,
```

```
  trim: true
},
permanentAddress: {
  type: String,
  trim: true
},
dateOfBirth: {
  type: Date
},
joiningDate: {
  type: Date,
  default: Date.now
},
salary: {
  type: Number,
  min: 0
},
status: {
  type: String,
  enum: ['ACTIVE', 'INACTIVE', 'TERMINATED'],
  default: 'ACTIVE'
},
department: {
  type: mongoose.Schema.ObjectId,
  ref: 'Department',
  required: [true, 'Please assign a department']
},
role: {
  type: mongoose.Schema.ObjectId,
  ref: 'Role',
  required: [true, 'Please assign a role']
},
hrId: {
  type: mongoose.Schema.ObjectId,
  ref: 'User'
},
// Educational Information
collegeName: {
  type: String,
  trim: true
},
internshipDuration: {
  type: Number // in months
},
// Document Storage (supporting both simple strings and detailed objects)
photograph: {
  type: mongoose.Schema.Types.Mixed
},
tenthMarksheet: {
  type: mongoose.Schema.Types.Mixed
},
twelfthMarksheet: {
  type: mongoose.Schema.Types.Mixed
},
bachelorDegree: {
  type: mongoose.Schema.Types.Mixed
},
postgraduateDegree: {
  type: mongoose.Schema.Types.Mixed
```

```
  },
  aadharCard: {
    type: mongoose.Schema.Types.Mixed
  },
  panCard: {
    type: mongoose.Schema.Types.Mixed
  },
  pcc: {
    type: mongoose.Schema.Types.Mixed
  },
  resume: {
    type: mongoose.Schema.Types.Mixed
  },
  offerLetter: {
    type: mongoose.Schema.Types.Mixed
  },
  // General documents object for additional flexibility
  documents: {
    type: mongoose.Schema.Types.Mixed,
    default: {}
  },
  // User account reference
  userId: {
    type: mongoose.Schema.ObjectId,
    ref: 'User'
  }
}, {
  timestamps: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});

// Populate department and role on find
employeeSchema.pre(/^find/, function(next) {
  this.populate({
    path: 'department',
    select: 'name description'
  }).populate({
    path: 'role',
    select: 'name description'
  }).populate({
    path: 'hrId',
    select: 'fullName email'
  });
  next();
});

module.exports = mongoose.model('Employee', employeeSchema);
```

models/EmployeeRole.js

```
const mongoose = require('mongoose');

const roleSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please add a role name'],
    unique: true,
```

```
    trim: true
  },
  description: {
    type: String,
    trim: true
  },
  isActive: {
    type: Boolean,
    default: true
  }
}, {
  timestamps: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});

// Virtual for employee count
roleSchema.virtual('employeeCount', {
  ref: 'Employee',
  localField: '_id',
  foreignField: 'role',
  count: true
});

module.exports = mongoose.model('Role', roleSchema);
```

models/ExchangeRate.js

```
const mongoose = require('mongoose');

const exchangeRateSchema = new mongoose.Schema({
  rates: {
    type: Map,
    of: Number,
    default: {
      'USD': 1,
      'EUR': 0.85,
      'GBP': 0.73,
      'INR': 83.12,
      'CAD': 1.36,
      'AUD': 1.52,
      'JPY': 149.50,
      'CNY': 7.24
    }
  },
  baseCurrency: {
    type: String,
    default: 'USD'
  },
  lastUpdated: {
    type: Date,
    default: Date.now
  }
}, {
  timestamps: true
});

// Create a default exchange rate document if none exists
```

```
exchangeRateSchema.statics.getOrCreateDefault = async function() {
  let exchangeRate = await this.findOne();
  if (!exchangeRate) {
    exchangeRate = await this.create({});
  }
  return exchangeRate;
};

module.exports = mongoose.model('ExchangeRate', exchangeRateSchema);
```

models/GroupChat.js

```
const mongoose = require('mongoose');

const groupChatSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true,
    maxlength: 100
  },
  description: {
    type: String,
    trim: true,
    maxlength: 500
  },
  avatar: {
    type: String,
    default: ''
  },
  createdBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  admins: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }],
  members: [{
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      required: true
    },
    joinedAt: {
      type: Date,
      default: Date.now
    },
    role: {
      type: String,
      enum: ['admin', 'member'],
      default: 'member'
    }
  }],
  lastMessage: {
    content: String,
    sender: {
```

```
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    },
    timestamp: Date
  },
  isActive: {
    type: Boolean,
    default: true
  },
  settings: {
    allowMemberInvite: {
      type: Boolean,
      default: false
    },
    muteNotifications: {
      type: Boolean,
      default: false
    }
  }
}, {
  timestamps: true
});

// Index for efficient querying
groupChatSchema.index({ 'members.user': 1 });
groupChatSchema.index({ createdBy: 1 });
groupChatSchema.index({ isActive: 1 });

module.exports = mongoose.model('GroupChat', groupChatSchema);
```

models/Incentive.js

```
const mongoose = require('mongoose');

const incentiveSchema = new mongoose.Schema({
  employeeId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Employee',
    required: true
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  type: {
    type: String,
    enum: ['PERFORMANCE', 'PROJECT', 'ATTENDANCE', 'FESTIVAL', 'ANNUAL',
'SPOT_AWARD', 'REFERRAL', 'RETENTION'],
    required: true
  },
  title: {
    type: String,
    required: true,
    trim: true
  },
  description: {
    type: String,
```

```
  required: true,
  trim: true
},
amount: {
  type: Number,
  required: true,
  min: 0
},
currency: {
  type: String,
  default: 'INR'
},

// Performance related fields
performanceRating: {
  type: Number,
  min: 1,
  max: 5
},
performancePeriod: {
  from: Date,
  to: Date
},

// Project related fields
projectName: {
  type: String,
  trim: true
},
projectCompletionDate: {
  type: Date
},

// Attendance related fields
attendancePercentage: {
  type: Number,
  min: 0,
  max: 100
},
attendancePeriod: {
  month: Number,
  year: Number
},

// Festival/Annual bonus fields
festivalType: {
  type: String,
  enum: ['DIWALI', 'CHRISTMAS', 'EID', 'HOLI', 'DUSSEHRA', 'NEW_YEAR', 'OTHER']
},

// Status and Approval
status: {
  type: String,
  enum: ['PENDING', 'APPROVED', 'PAID', 'REJECTED', 'CANCELLED'],
  default: 'PENDING'
},

// Approval workflow
requestedBy: {
```

```
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: true
},
approvedBy: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User'
},
approvedDate: {
  type: Date
},
rejectedReason: {
  type: String,
  trim: true
},

// Payment details
paymentDate: {
  type: Date
},
paymentMethod: {
  type: String,
  enum: ['SALARY_INCLUDED', 'SEPARATE_PAYMENT', 'CASH', 'BANK_TRANSFER'],
  default: 'SALARY_INCLUDED'
},

// Include in which payroll
payrollId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'Payroll'
},

// Validity period
validFrom: {
  type: Date,
  default: Date.now
},
validTo: {
  type: Date
},

// Additional metadata
isRecurring: {
  type: Boolean,
  default: false
},
recurringType: {
  type: String,
  enum: ['MONTHLY', 'QUARTERLY', 'YEARLY'],
  required: function() {
    return this.isRecurring;
  }
},

// Comments and notes
comments: [{
  commentBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
```

```
    },
    comment: String,
    timestamp: {
      type: Date,
      default: Date.now
    }
  }],

  // Attachments (for supporting documents)
  attachments: [{
    filename: String,
    originalName: String,
    path: String,
    size: Number,
    uploadedBy: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    },
    uploadedAt: {
      type: Date,
      default: Date.now
    }
  }]
}, {
  timestamps: true
});

// Indexes for better performance
incentiveSchema.index({ employeeId: 1, type: 1 });
incentiveSchema.index({ status: 1 });
incentiveSchema.index({ validFrom: 1, validTo: 1 });

// Virtual for formatted amount
incentiveSchema.virtual('formattedAmount').get(function() {
  return new Intl.NumberFormat('en-IN', {
    style: 'currency',
    currency: this.currency
  }).format(this.amount);
});

// Method to check if incentive is valid
incentiveSchema.methods.isValid = function() {
  const now = new Date();
  return (!this.validFrom || this.validFrom <= now) &&
         (!this.validTo || this.validTo >= now);
};

// Method to add comment
incentiveSchema.methods.addComment = function(userId, comment) {
  this.comments.push({
    commentBy: userId,
    comment: comment,
    timestamp: new Date()
  });
};

// Static method to get incentives for payroll
incentiveSchema.statics.getIncentivesForPayroll = function(employeeId, month,
year) {
```

```
  const startDate = new Date(year, month - 1, 1);
  const endDate = new Date(year, month, 0);

  return this.find({
    employeeId: employeeId,
    status: 'APPROVED',
    validFrom: { $lte: endDate },
    $or: [
      { validTo: { $gte: startDate } },
      { validTo: null }
    ]
  });
};

// Pre-save middleware
incentiveSchema.pre('save', function(next) {
  // Note: Auto-approval removed - all incentives require manual approval based
on sales performance
  next();
});

module.exports = mongoose.model('Incentive', incentiveSchema);
```

models/Lead.js

```
const mongoose = require('mongoose');

const LeadSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please add a lead name'],
    trim: true,
    maxlength: [100, 'Name cannot be more than 100 characters']
  },
  email: {
    type: String,
    // Accept any string with @ as a valid email - more permissive to allow
international formats
    validate: {
      validator: function(value) {
        // If email is empty, it's valid (email is optional)
        if (!value || value.trim() === '') return true;
        // Otherwise just check for @ symbol
        return value.includes('@');
      },
      message: props => `${props.value} is not a valid email format. Must contain
@ symbol.`
    },
    // Don't enforce unique index on email - allow duplicates
    index: false
  },
  course: {
    type: String,
    trim: true,
    required: [true, 'Please specify the course']
  },
  countryCode: {
    type: String,
```

```
    trim: true,
    required: [true, 'Please add country code']
  },
  phone: {
    type: String,
    required: [true, 'Please add a phone number'],
    maxlength: [20, 'Phone number cannot be longer than 20 characters'],
    // Don't enforce unique index on phone - allow duplicates
    index: false
  },
  country: {
    type: String,
    trim: true,
    required: [true, 'Please add the country']
  },
  pseudoId: {
    type: String,
    trim: true
  },
  company: {
    type: String,
    trim: true
  },
  client: {
    type: String,
    trim: true
  },
  status: {
    type: String,
    enum: ['New', 'Contacted', 'Qualified', 'Lost', 'Converted', 'Introduction',
'Acknowledgement', 'Question', 'Future Promise', 'Payment', 'Analysis'],
    default: 'Introduction'
  },
  source: {
    type: String,
    default: ''
  },
  sourceLink: {
    type: String,
    trim: true
  },
  assignedTo: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  leadPerson: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  createdBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  remarks: {
    type: String
  },
  feedback: {
    type: String
  },
```

```
  // Fields to track repeat customers
  isRepeatCustomer: {
    type: Boolean,
    default: false
  },
  previousCourses: [{
    type: String,
    trim: true
  }],
  relatedLeads: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Lead'
  }],
  createdAt: {
    type: Date,
    default: Date.now
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
});

module.exports = mongoose.model('Lead', LeadSchema);
```

models/LeadPersonSale.js

```
const mongoose = require('mongoose');

const LeadPersonSaleSchema = new mongoose.Schema({
  date: {
    type: Date,
    default: Date.now,
    required: true
  },
  customerName: {
    type: String,
    required: [true, 'Please add a customer name'],
    trim: true,
    maxlength: [100, 'Name cannot be more than 100 characters']
  },
  country: {
    type: String,
    required: [true, 'Please add a country'],
    trim: true
  },
  course: {
    type: String,
    required: [true, 'Please add a course name'],
    trim: true
  },
  countryCode: {
    type: String,
    trim: true
  },
  contactNumber: {
    type: String,
    required: [true, 'Please add a contact number'],
```

```
  trim: true
},
email: {
  type: String,
  trim: true,
  match: [
    /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/,
    'Please add a valid email'
  ]
},
pseudoId: {
  type: String,
  trim: true
},
salesPerson: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: [true, 'Please assign a sales person']
},
leadPerson: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: [true, 'Please assign a lead person']
},
source: {
  type: String,
  trim: true
},
clientRemark: {
  type: String,
  trim: true
},
feedback: {
  type: String,
  trim: true
},
totalCost: {
  type: Number,
  default: 0
},
totalCostCurrency: {
  type: String,
  default: 'USD',
  trim: true
},
tokenAmount: {
  type: Number,
  default: 0
},
tokenAmountCurrency: {
  type: String,
  default: 'USD',
  trim: true
},
status: {
  type: String,
  enum: ['Completed', 'Pending', 'Cancelled'],
  default: 'Pending'
},
```

```
  notes: {
    type: String,
    trim: true
  },
  createdBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  updatedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }
}, {
  timestamps: true
});

module.exports = mongoose.model('LeadPersonSale', LeadPersonSaleSchema);
```

models/Leave.js

```
const mongoose = require('mongoose');

const leaveSchema = new mongoose.Schema({
  employeeId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Employee',
    required: true
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  leaveType: {
    type: String,
    enum: ['sick', 'casual', 'annual', 'emergency', 'maternity', 'paternity',
'bereavement', 'personal'],
    required: true
  },
  startDate: {
    type: Date,
    required: true
  },
  endDate: {
    type: Date,
    required: true
  },
  totalDays: {
    type: Number,
    required: true
  },
  reason: {
    type: String,
    required: true,
    minlength: 10,
    maxlength: 500
  },
  status: {
```

```
    type: String,
    enum: ['pending', 'approved', 'rejected', 'cancelled'],
    default: 'pending'
  },
  appliedDate: {
    type: Date,
    default: Date.now
  },
  approvedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  approvedDate: {
    type: Date
  },
  rejectionReason: {
    type: String,
    maxlength: 300
  },
  attachments: [{
    filename: String,
    path: String,
    uploadedAt: {
      type: Date,
      default: Date.now
    }
  }],
  comments: [{
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    },
    comment: String,
    createdAt: {
      type: Date,
      default: Date.now
    }
  }],
  isHalfDay: {
    type: Boolean,
    default: false
  },
  halfDaySession: {
    type: String,
    enum: ['morning', 'afternoon'],
    required: function() {
      return this.isHalfDay;
    }
  }
}, {
  timestamps: true
});

// Calculate total days automatically
leaveSchema.pre('save', function(next) {
  if (this.isNew || this.isModified('startDate') || this.isModified('endDate') ||
this.isModified('isHalfDay')) {
    if (this.isHalfDay) {
      this.totalDays = 0.5;
```

```
    } else {
      const timeDiff = this.endDate.getTime() - this.startDate.getTime();
      const daysDiff = Math.ceil(timeDiff / (1000 * 3600 * 24)) + 1;
      this.totalDays = daysDiff;
    }
  }
  next();
});

// Index for better query performance
leaveSchema.index({ employeeId: 1, status: 1 });
leaveSchema.index({ startDate: 1, endDate: 1 });
leaveSchema.index({ appliedDate: -1 });

module.exports = mongoose.model('Leave', leaveSchema);
```

models/Log.js

```
const mongoose = require('mongoose');

const logSchema = new mongoose.Schema({
  action: {
    type: String,
    required: true,
    enum: [
      'LOGIN',
      'LOGOUT',
      'LEAD_CREATE',
      'LEAD_UPDATE',
      'LEAD_DELETE',
      'LEAD_ASSIGN',
      'SALE_CREATE',
      'SALE_UPDATE',
      'SALE_DELETE',
      'USER_CREATE',
      'USER_UPDATE',
      'USER_DELETE',
      'EMPLOYEE_CREATE',
      'EMPLOYEE_UPDATE',
      'EMPLOYEE_DELETE',
      'ATTENDANCE_MARK',
      'LEAVE_REQUEST',
      'LEAVE_UPDATE',
      'PAYROLL_UPDATE',
      'SETTINGS_UPDATE'
    ]
  },
  performedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  timestamp: {
    type: Date,
    default: Date.now
  },
  details: {
    type: mongoose.Schema.Types.Mixed,
```

```
    required: true
  },
  ipAddress: String,
  userAgent: String,
  affectedResource: {
    type: String,
    required: true
  },
  resourceId: {
    type: mongoose.Schema.Types.ObjectId,
    required: false
  },
  previousState: mongoose.Schema.Types.Mixed,
  newState: mongoose.Schema.Types.Mixed,
  status: {
    type: String,
    enum: ['SUCCESS', 'FAILURE', 'WARNING'],
    default: 'SUCCESS'
  },
  additionalInfo: mongoose.Schema.Types.Mixed
});

// Add indexes for better query performance
logSchema.index({ timestamp: -1 });
logSchema.index({ action: 1 });
logSchema.index({ performedBy: 1 });
logSchema.index({ affectedResource: 1 });
logSchema.index({ status: 1 });

const Log = mongoose.model('Log', logSchema);

module.exports = Log;
```

models/Payroll.js

```
const mongoose = require('mongoose');

const payrollSchema = new mongoose.Schema({
  employeeId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Employee',
    required: true
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  month: {
    type: Number,
    required: true,
    min: 1,
    max: 12
  },
  year: {
    type: Number,
    required: true
  },
```

```
baseSalary: {
  type: Number,
  required: true
},
daysPresent: {
  type: Number,
  required: true,
  default: 0
},
calculatedSalary: {
  type: Number,
  required: true,
  default: 0
},
workingDays: {
  type: Number,
  required: true,
  default: 30
},
presentDays: {
  type: Number,
  required: true,
  default: 0
},
absentDays: {
  type: Number,
  required: true,
  default: 0
},
halfDays: {
  type: Number,
  default: 0
},
overtimeHours: {
  type: Number,
  default: 0
},

// Salary Components
basicAmount: {
  type: Number,
  default: 0
},
hra: {
  type: Number,
  default: 0
},
da: {
  type: Number,
  default: 0
},
conveyanceAllowance: {
  type: Number,
  default: 0
},
medicalAllowance: {
  type: Number,
  default: 0
},
```

```
specialAllowance: {
  type: Number,
  default: 0
},
overtimeAmount: {
  type: Number,
  default: 0
},

// Incentives
performanceBonus: {
  type: Number,
  default: 0
},
projectBonus: {
  type: Number,
  default: 0
},
attendanceBonus: {
  type: Number,
  default: 0
},
festivalBonus: {
  type: Number,
  default: 0
},

// Deductions
pf: {
  type: Number,
  default: 0
},
esi: {
  type: Number,
  default: 0
},
tax: {
  type: Number,
  default: 0
},
loan: {
  type: Number,
  default: 0
},
other: {
  type: Number,
  default: 0
},

// Calculated Fields
grossSalary: {
  type: Number,
  default: 0
},
totalDeductions: {
  type: Number,
  default: 0
},
netSalary: {
```

```
    type: Number,
    default: 0
  },

  // Status
  status: {
    type: String,
    enum: ['DRAFT', 'APPROVED', 'PAID', 'CANCELLED'],
    default: 'DRAFT'
  },

  // Approval
  approvedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  approvedDate: {
    type: Date
  },

  // Payment
  paymentDate: {
    type: Date
  },
  paymentMethod: {
    type: String,
    enum: ['BANK_TRANSFER', 'CASH', 'CHEQUE'],
    default: 'BANK_TRANSFER'
  },

  // Salary Slip
  salarySlipPath: {
    type: String
  },

  notes: {
    type: String,
    maxlength: 1000
  }
}, {
  timestamps: true
});

// Create compound index for employee, month, and year
payrollSchema.index({ employeeId: 1, month: 1, year: 1 }, { unique: true });

// Virtual for month name
payrollSchema.virtual('monthName').get(function() {
  const months = ['January', 'February', 'March', 'April', 'May', 'June',
                  'July', 'August', 'September', 'October', 'November',
'December'];
  return months[this.month - 1];
});

// Method to calculate salary based on manual input
payrollSchema.methods.calculateSalary = function() {
  console.log('Ø>Ýî Starting fully manual salary calculation for payroll:', this._id);
  console.log('Ø=ÜÊ All input values:', {
    baseSalary: this.baseSalary,
```

```
    daysPresent: this.daysPresent,
    calculatedSalary: this.calculatedSalary,
    // Manual Allowances
    hra: this.hra,
    da: this.da,
    conveyanceAllowance: this.conveyanceAllowance,
    medicalAllowance: this.medicalAllowance,
    specialAllowance: this.specialAllowance,
    overtimeAmount: this.overtimeAmount,
    // Manual Bonuses
    performanceBonus: this.performanceBonus,
    projectBonus: this.projectBonus,
    attendanceBonus: this.attendanceBonus,
    festivalBonus: this.festivalBonus,
    // Manual Deductions
    pf: this.pf,
    esi: this.esi,
    tax: this.tax,
    loan: this.loan,
    other: this.other
  });

  // 1. Basic amount is the manually calculated salary
  this.basicAmount = this.calculatedSalary || 0;
  console.log('Ø=Ü° Basic amount (calculated salary):', this.basicAmount);

  // 2. Calculate gross salary = basic + ALL manual allowances + ALL manual
bonuses
  this.grossSalary = this.basicAmount +
                     (this.hra || 0) +
                     (this.da || 0) +
                     (this.conveyanceAllowance || 0) +
                     (this.medicalAllowance || 0) +
                     (this.specialAllowance || 0) +
                     (this.overtimeAmount || 0) +
                     (this.performanceBonus || 0) +
                     (this.projectBonus || 0) +
                     (this.attendanceBonus || 0) +
                     (this.festivalBonus || 0);

  console.log('Ø=Üµ Gross salary calculated:', {
    basicAmount: this.basicAmount,
    totalAllowances: (this.hra || 0) + (this.da || 0) + (this.conveyanceAllowance
|| 0) + (this.medicalAllowance || 0) + (this.specialAllowance || 0) +
(this.overtimeAmount || 0),
    totalBonuses: (this.performanceBonus || 0) + (this.projectBonus || 0) +
(this.attendanceBonus || 0) + (this.festivalBonus || 0),
    grossSalary: this.grossSalary
  });

  // 3. Calculate total deductions = ALL manual deductions
  this.totalDeductions = (this.pf || 0) +
                         (this.esi || 0) +
                         (this.tax || 0) +
                         (this.loan || 0) +
                         (this.other || 0);

  console.log('Ø=ÜÉ Deductions calculated:', {
    pf: this.pf || 0,
```

```
    esi: this.esi || 0,
    tax: this.tax || 0,
    loan: this.loan || 0,
    other: this.other || 0,
    totalDeductions: this.totalDeductions
  });

  // 4. Calculate net salary = gross - total deductions
  this.netSalary = this.grossSalary - this.totalDeductions;

  console.log('Ø<ß¯ Final manual calculation:', {
    grossSalary: this.grossSalary,
    totalDeductions: this.totalDeductions,
    netSalary: this.netSalary
  });

  console.log('' Formula: Basic + All Allowances + All Bonuses - All Deductions =
Net Salary');
  console.log(`' ${this.basicAmount} + ${(this.hra || 0) + (this.da || 0) +
(this.conveyanceAllowance || 0) + (this.medicalAllowance || 0) +
(this.specialAllowance || 0) + (this.overtimeAmount || 0)} +
${(this.performanceBonus || 0) + (this.projectBonus || 0) + (this.attendanceBonus
|| 0) + (this.festivalBonus || 0)} - ${this.totalDeductions} = ${this.netSalary}
`);

  return this.netSalary;
};

// Pre-save middleware to calculate salary
payrollSchema.pre('save', function(next) {
  // Always recalculate when saving
  this.calculateSalary();
  next();
});

module.exports = mongoose.model('Payroll', payrollSchema);
```

models/Prospect.js

```
const mongoose = require('mongoose');

const prospectSchema = new mongoose.Schema({
  // Basic Information (all optional)
  name: {
    type: String,
    trim: true
  },
  email: {
    type: String,
    trim: true,
    lowercase: true
  },
  phone: {
    type: String,
    trim: true
  },
  company: {
    type: String,
```

```
    trim: true
  },
  designation: {
    type: String,
    trim: true
  },

  // Source Information
  source: {
    type: String,
    enum: ['LinkedIn', 'Website', 'Referral', 'Cold Call', 'Email Campaign',
'Social Media', 'Event', 'Other'],
    default: 'Other'
  },
  sourceDetails: {
    type: String,
    trim: true
  },

  // Business Information
  industry: {
    type: String,
    trim: true
  },
  companySize: {
    type: String,
    enum: ['1-10', '11-50', '51-200', '201-500', '501-1000', '1000+', 'Unknown'],
    default: 'Unknown'
  },
  budget: {
    type: Number,
    min: 0
  },
  budgetCurrency: {
    type: String,
    default: 'USD'
  },

  // Interest & Requirements
  serviceInterest: {
    type: String,
    trim: true
  },
  requirements: {
    type: String,
    trim: true
  },
  timeline: {
    type: String,
    enum: ['Immediate', 'Within 1 month', '1-3 months', '3-6 months', '6+
months', 'Not specified'],
    default: 'Not specified'
  },

  // Status & Priority
  status: {
    type: String,
    enum: ['New', 'Contacted', 'Interested', 'Not Interested', 'Follow Up',
'Qualified', 'Converted to Lead', 'Lost'],
```

```
  default: 'New'
},
priority: {
  type: String,
  enum: ['High', 'Medium', 'Low'],
  default: 'Medium'
},

// Assignment & Tracking
assignedTo: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User'
},
createdBy: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: true
},

// Follow-up Information
lastContactDate: {
  type: Date
},
nextFollowUpDate: {
  type: Date
},
contactMethod: {
  type: String,
  enum: ['Email', 'Phone', 'LinkedIn', 'WhatsApp', 'Meeting', 'Other']
},

// Notes & Communication
notes: {
  type: String,
  trim: true
},
tags: [{
  type: String,
  trim: true
}],

// Conversion tracking
convertedToLead: {
  type: Boolean,
  default: false
},
leadId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'Lead'
},
conversionDate: {
  type: Date
},

// Social Media Links
linkedinProfile: {
  type: String,
  trim: true
},
```

```
  websiteUrl: {
    type: String,
    trim: true
  }
}, {
  timestamps: true
});

// Indexes for better performance
prospectSchema.index({ email: 1 });
prospectSchema.index({ phone: 1 });
prospectSchema.index({ assignedTo: 1 });
prospectSchema.index({ createdBy: 1 });
prospectSchema.index({ status: 1 });
prospectSchema.index({ source: 1 });
prospectSchema.index({ createdAt: -1 });

// Virtual for full contact info
prospectSchema.virtual('fullContactInfo').get(function() {
  const contact = [];
  if (this.email) contact.push(this.email);
  if (this.phone) contact.push(this.phone);
  return contact.join(' | ');
});

// Method to convert prospect to lead
prospectSchema.methods.convertToLead = function() {
  return {
    name: this.name,
    email: this.email,
    phone: this.phone,
    company: this.company,
    source: this.source,
    budget: this.budget,
    requirements: this.requirements,
    assignedTo: this.assignedTo,
    createdBy: this.createdBy,
    notes: `Converted from Prospect. Original notes: ${this.notes || 'None'}`
  };
};

module.exports = mongoose.model('Prospect', prospectSchema);
```

models/Sale.js

```
const mongoose = require('mongoose');

const SaleSchema = new mongoose.Schema({
  date: {
    type: Date,
    default: Date.now,
    required: true
  },
  customerName: {
    type: String,
    required: [true, 'Please add a customer name'],
    trim: true,
    maxlength: [100, 'Name cannot be more than 100 characters']
```

```
},
country: {
  type: String,
  required: [true, 'Please add a country'],
  trim: true
},
course: {
  type: String,
  required: [true, 'Please add a course name'],
  trim: true
},
countryCode: {
  type: String,
  trim: true
},
contactNumber: {
  type: String,
  required: [true, 'Please add a contact number'],
  trim: true
},
email: {
  type: String,
  trim: true,
  match: [
    /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/,
    'Please add a valid email'
  ]
},
pseudoId: {
  type: String,
  trim: true
},
salesPerson: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: [true, 'Please assign a sales person']
},
leadPerson: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: [true, 'Please assign a lead person']
},
leadBy: {
  type: String,
  trim: true
},
loginId: {
  type: String,
  trim: true
},
password: {
  type: String,
  trim: true
},
source: {
  type: String,
  trim: true
},
isReference: {
```

```
  type: Boolean,
  default: false
},
isLeadPersonSale: {
  type: Boolean,
  default: false
},
clientRemark: {
  type: String,
  trim: true
},
feedback: {
  type: String,
  trim: true
},
totalCost: {
  type: Number,
  default: 0
},
totalCostCurrency: {
  type: String,
  default: 'USD',
  trim: true
},
tokenAmount: {
  type: Number,
  default: 0
},
tokenAmountCurrency: {
  type: String,
  default: 'USD',
  trim: true
},
currency: {
  type: String,
  default: 'USD',
  enum: ['USD', 'EUR', 'GBP', 'INR', 'CAD', 'AUD', 'JPY', 'CNY'],
  trim: true
},
pending: {
  type: Boolean,
  default: true
},
status: {
  type: String,
  enum: ['Completed', 'Pending', 'Cancelled'],
  default: 'Pending'
},
notes: {
  type: String,
  trim: true
},
remarks: {
  type: String,
  trim: true,
  default: '' // Set default value to empty string
},
createdBy: {
  type: mongoose.Schema.Types.ObjectId,
```

```
    ref: 'User'
  },
  updatedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }
}, {
  timestamps: true
});

module.exports = mongoose.model('Sale', SaleSchema);
```

models/Task.js

```
const mongoose = require('mongoose');

const TaskSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, 'Please add a task title'],
    trim: true
  },
  description: {
    type: String,
    trim: true
  },
  taskType: {
    type: String,
    enum: ['Exam', 'Follow-up', 'Other'],
    default: 'Exam'
  },
  course: {
    type: String,
    trim: true,
    required: function() {
      return this.taskType === 'Exam';
    }
  },
  location: {
    type: String,
    trim: true,
    default: 'Online'
  },
  examLink: {
    type: String,
    trim: true
  },
  customer: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Lead',
    required: function() {
      return !this.manualCustomer; // Required only if manualCustomer is not
provided
    }
  },
  manualCustomer: {
    name: {
      type: String,
```

```
      trim: true
    },
    email: {
      type: String,
      trim: true,
      match: [
        /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/,
        'Please add a valid email'
      ]
    },
    contactNumber: {
      type: String,
      trim: true
    },
    course: {
      type: String,
      trim: true
    }
  },
  examDate: {
    type: Date,
    required: [true, 'Please specify the exam date and time']
  },
  examDateTime: {
    type: Date,
    required: [true, 'Please specify the exam date and time']
  },
  assignedTo: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: [true, 'Please assign a user to this task']
  },
  reminderSent: {
    type: Boolean,
    default: false
  },
  remindersSent: [{
    sentAt: {
      type: Date,
      required: true
    },
    reminderType: {
      type: String,
      enum: ['30-minute-before', '10-minute-before', 'exam-time', '10-minute-
after', 'other'],
      default: 'other'
    }
  }],
  completed: {
    type: Boolean,
    default: false
  },
  salesPerson: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: [true, 'Please assign a sales person']
  },
  createdAt: {
    type: Date,
```

```
    default: Date.now
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
}, {
  timestamps: true
});

// Pre-save middleware to sync examDate and examDateTime
TaskSchema.pre('save', function(next) {
  if (this.examDate && !this.examDateTime) {
    this.examDateTime = this.examDate;
  } else if (this.examDateTime && !this.examDate) {
    this.examDate = this.examDateTime;
  }
  next();
});

module.exports = mongoose.model('Task', TaskSchema);
```

models/User.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

const UserSchema = new mongoose.Schema({
  fullName: {
    type: String,
    required: [true, 'Please add a name'],
    trim: true,
    maxlength: [50, 'Name cannot be more than 50 characters']
  },
  email: {
    type: String,
    required: [true, 'Please add an email'],
    unique: true,
    match: [
      /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,})+$/,
      'Please add a valid email'
    ]
  },
  password: {
    type: String,
    required: [true, 'Please add a password'],
    minlength: [6, 'Password must be at least 6 characters'],
    select: false
  },
  role: {
    type: String,
    enum: ['Sales Person', 'Lead Person', 'Manager', 'Admin', 'Customer', 'HR',
'Employee'],
    default: 'Sales Person'
  },
  // Employee reference for Employee role users
  employeeId: {
```

```
    type: mongoose.Schema.ObjectId,
    ref: 'Employee'
  },
  profilePicture: {
    type: String,
    default: ''
  },
  // Chat-related fields
  chatStatus: {
    type: String,
    enum: ['ONLINE', 'OFFLINE', 'AWAY'],
    default: 'OFFLINE'
  },
  lastSeen: {
    type: Date,
    default: Date.now
  },
  createdAt: {
    type: Date,
    default: Date.now
  },
  verifyOtp: { type: String, default: "123456" },
  verifyOtpExpireAt: { type: Number, default: 0 },
  resetOtp: { type: String, default: "" },
  resetOtpExpireAt: { type: Number, default: 0 },
}, {
  // Use the existing collection
  collection: 'users'
});

// Encrypt password using bcrypt
UserSchema.pre('save', async function(next) {
  if (!this.isModified('password')) {
    return next();
  }

  try {
    const salt = await bcrypt.genSalt(10);
    this.password = await bcrypt.hash(this.password, salt);
    next();
  } catch (error) {
    next(error);
  }
});

// Sign JWT and return
UserSchema.methods.getSignedJwtToken = function() {
  return jwt.sign({ id: this._id }, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRE || '30d'
  });
};

// Match user entered password to hashed password in database
UserSchema.methods.matchPassword = async function(enteredPassword) {
  try {
    return await bcrypt.compare(enteredPassword, this.password);
  } catch (error) {
    console.error('Error comparing passwords:', error);
    throw error;
```

```
  }
};

module.exports = mongoose.model('User', UserSchema);
```

models/UserActivity.js

```
const mongoose = require('mongoose');

const UserActivitySchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  date: {
    type: String, // Format: YYYY-MM-DD
    required: true
  },
  sessions: [{
    startTime: {
      type: Date,
      required: true
    },
    endTime: {
      type: Date,
      default: null
    },
    duration: {
      type: Number, // Duration in seconds
      default: 0
    },
    isActive: {
      type: Boolean,
      default: true
    }
  }],
  totalActiveTime: {
    type: Number, // Total time for the day in seconds
    default: 0
  },
  lastActivity: {
    type: Date,
    default: Date.now
  }
}, {
  timestamps: true
});

// Create compound index for efficient queries
UserActivitySchema.index({ userId: 1, date: 1 }, { unique: true });

// Method to add a new session
UserActivitySchema.methods.startSession = function() {
  this.sessions.push({
    startTime: new Date(),
    isActive: true
  });
```

```
  this.lastActivity = new Date();
  return this.save();
};

// Method to end the current active session
UserActivitySchema.methods.endCurrentSession = function() {
  const activeSession = this.sessions.find(session => session.isActive);
  if (activeSession) {
    activeSession.endTime = new Date();
    activeSession.duration = Math.floor((activeSession.endTime -
activeSession.startTime) / 1000);
    activeSession.isActive = false;

    // Update total active time
    this.totalActiveTime += activeSession.duration;
    this.lastActivity = new Date();
  }
  return this.save();
};

// Static method to get or create today's activity record
UserActivitySchema.statics.getTodaysActivity = async function(userId) {
  const today = new Date().toISOString().split('T')[0];

  let activity = await this.findOne({ userId, date: today });

  if (!activity) {
    activity = new this({
      userId,
      date: today,
      sessions: [],
      totalActiveTime: 0
    });
    await activity.save();
  }

  return activity;
};

// Static method to get activity for a specific date range
UserActivitySchema.statics.getActivityByDateRange = async function(userId,
startDate, endDate) {
  return this.find({
    userId,
    date: {
      $gte: startDate,
      $lte: endDate
    }
  }).sort({ date: -1 });
};

module.exports = mongoose.model('UserActivity', UserActivitySchema);
```

package-lock.json

```
{
  "name": "crm-server",
  "version": "1.0.0",
```

```
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "crm-server",
      "version": "1.0.0",
      "hasInstallScript": true,
      "dependencies": {
        "@google/generative-ai": "^0.24.1",
        "agenda": "^5.0.0",
        "axios": "^1.9.0",
        "bcrypt": "^5.1.1",
        "bcryptjs": "^2.4.3",
        "cors": "^2.8.5",
        "dotenv": "^16.0.3",
        "express": "^4.18.2",
        "express-rate-limit": "^6.7.0",
        "jsonwebtoken": "^9.0.0",
        "lottie-react": "^2.4.1",
        "mongoose": "^7.0.3",
        "morgan": "^1.10.0",
        "multer": "^1.4.5-lts.1",
        "node-cron": "^3.0.3",
        "nodemailer": "^6.9.1",
        "pdfkit": "^0.17.1",
        "socket.io": "^4.8.1"
      },
      "devDependencies": {
        "nodemon": "^3.0.1"
      }
    },
    "node_modules/@aws-crypto/sha256-browser": {
      "version": "5.2.0",
      "resolved": "https://registry.npmjs.org/@aws-crypto/sha256-browser/-/sha256-
browser-5.2.0.tgz",
      "integrity": "sha512-AXfN/lGotSQwu6HNcEsIASo7kWXZ5HYWvfOmSNKDsEqC4OashTp8alT
maz+F7TC2L083SFv5RdB+qU3Vs1kZqw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-crypto/sha256-js": "^5.2.0",
        "@aws-crypto/supports-web-crypto": "^5.2.0",
        "@aws-crypto/util": "^5.2.0",
        "@aws-sdk/types": "^3.222.0",
        "@aws-sdk/util-locate-window": "^3.0.0",
        "@smithy/util-utf8": "^2.0.0",
        "tslib": "^2.6.2"
      }
    },
    "node_modules/@aws-crypto/sha256-browser/node_modules/@smithy/is-array-
buffer": {
      "version": "2.2.0",
      "resolved": "https://registry.npmjs.org/@smithy/is-array-buffer/-/is-array-
buffer-2.2.0.tgz",
      "integrity": "sha512-GGP3O9QFD24uGeAXYUjwSTXARoqpZykHadOmA8G5vfJPK0/
DC67qa//0qvqrJzL1xc8WQWX7/yc7fwudjPHPhA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
```

```
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=14.0.0"
      }
    },
    "node_modules/@aws-crypto/sha256-browser/node_modules/@smithy/util-buffer-
from": {
      "version": "2.2.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-buffer-from/-/util-
buffer-from-2.2.0.tgz",
      "integrity": "sha512-IJdWBbTcMQ6DA0gdNhh/
BwrLkDR+ADW5Kr1aZmd4k3DIF6ezMV4R2NIAmT08wQJ3yUK82thHWmC/TnK/wpMMIA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/is-array-buffer": "^2.2.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=14.0.0"
      }
    },
    "node_modules/@aws-crypto/sha256-browser/node_modules/@smithy/util-utf8": {
      "version": "2.3.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-utf8/-/util-
utf8-2.3.0.tgz",
      "integrity": "sha512-R8Rdn8Hy72KKcebgLiv8jQcQkXoLMOGGv5uI1/
k0l+snqkOzQ1R0ChUBCxWMlBsFMekWjq0wRudIweFs7sKT5A==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/util-buffer-from": "^2.2.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=14.0.0"
      }
    },
    "node_modules/@aws-crypto/sha256-js": {
      "version": "5.2.0",
      "resolved": "https://registry.npmjs.org/@aws-crypto/sha256-js/-/sha256-
js-5.2.0.tgz",
      "integrity": "sha512-FFQQyu7edu4ufvIZ+OadFpHHOt+eSTBaYaki44c+akjg7qZg9oOQeLl
k77F6tSYqjDAFClrHJk9tMf0HdVyOvA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-crypto/util": "^5.2.0",
        "@aws-sdk/types": "^3.222.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=16.0.0"
      }
    },
    "node_modules/@aws-crypto/supports-web-crypto": {
      "version": "5.2.0",
      "resolved": "https://registry.npmjs.org/@aws-crypto/supports-web-crypto/-/
```

```
supports-web-crypto-5.2.0.tgz",
      "integrity": "sha512-iAvUotm021kM33eCdNfwIN//F77/
IADDSs58i+MDaOqFrVjZo9bAal0NK7HurRuWLLpF1iLX7gbWrjHjeo+YFg==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      }
    },
    "node_modules/@aws-crypto/util": {
      "version": "5.2.0",
      "resolved": "https://registry.npmjs.org/@aws-crypto/util/-/util-5.2.0.tgz",
      "integrity": "sha512-4RkU9EsI6ZpBve5fseQlGNUWKMa1RLPQ1dnjnQoe07ldfIzcsGb5hC5
W0Dm7u423KWzawlrpbjXBrXCEv9zazQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/types": "^3.222.0",
        "@smithy/util-utf8": "^2.0.0",
        "tslib": "^2.6.2"
      }
    },
    "node_modules/@aws-crypto/util/node_modules/@smithy/is-array-buffer": {
      "version": "2.2.0",
      "resolved": "https://registry.npmjs.org/@smithy/is-array-buffer/-/is-array-
buffer-2.2.0.tgz",
      "integrity": "sha512-GGP3O9QFD24uGeAXYUjwSTXARoqpZykHadOmA8G5vfJPK0/
DC67qa//0qvqrJzL1xc8WQWX7/yc7fwudjPHPhA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=14.0.0"
      }
    },
    "node_modules/@aws-crypto/util/node_modules/@smithy/util-buffer-from": {
      "version": "2.2.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-buffer-from/-/util-
buffer-from-2.2.0.tgz",
      "integrity": "sha512-IJdWBbTcMQ6DA0gdNhh/
BwrLkDR+ADW5Kr1aZmd4k3DIF6ezMV4R2NIAmT08wQJ3yUK82thHWmC/TnK/wpMMIA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/is-array-buffer": "^2.2.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=14.0.0"
      }
    },
    "node_modules/@aws-crypto/util/node_modules/@smithy/util-utf8": {
      "version": "2.3.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-utf8/-/util-
utf8-2.3.0.tgz",
      "integrity": "sha512-R8Rdn8Hy72KKcebgLiv8jQcQkXoLMOGGv5uI1/
k0l+snqkOzQ1R0ChUBCxWMlBsFMekWjq0wRudIweFs7sKT5A==",
```

```
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/util-buffer-from": "^2.2.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=14.0.0"
      }
    },
    "node_modules/@aws-sdk/client-cognito-identity": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/client-cognito-identity/-/
client-cognito-identity-3.821.0.tgz",
      "integrity": "sha512-c6TpvrRAb4hVcbGMCPjTWU2IRNBzfEz2qZ1v6DGViW0i8vN4+zXY/
DcVOL2P3ZA9MDXjFRiiA8RdIy1/zsi3YQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-crypto/sha256-browser": "5.2.0",
        "@aws-crypto/sha256-js": "5.2.0",
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/credential-provider-node": "3.821.0",
        "@aws-sdk/middleware-host-header": "3.821.0",
        "@aws-sdk/middleware-logger": "3.821.0",
        "@aws-sdk/middleware-recursion-detection": "3.821.0",
        "@aws-sdk/middleware-user-agent": "3.821.0",
        "@aws-sdk/region-config-resolver": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@aws-sdk/util-endpoints": "3.821.0",
        "@aws-sdk/util-user-agent-browser": "3.821.0",
        "@aws-sdk/util-user-agent-node": "3.821.0",
        "@smithy/config-resolver": "^4.1.4",
        "@smithy/core": "^3.5.1",
        "@smithy/fetch-http-handler": "^5.0.4",
        "@smithy/hash-node": "^4.0.4",
        "@smithy/invalid-dependency": "^4.0.4",
        "@smithy/middleware-content-length": "^4.0.4",
        "@smithy/middleware-endpoint": "^4.1.9",
        "@smithy/middleware-retry": "^4.1.10",
        "@smithy/middleware-serde": "^4.0.8",
        "@smithy/middleware-stack": "^4.0.4",
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/node-http-handler": "^4.0.6",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/smithy-client": "^4.4.1",
        "@smithy/types": "^4.3.1",
        "@smithy/url-parser": "^4.0.4",
        "@smithy/util-base64": "^4.0.0",
        "@smithy/util-body-length-browser": "^4.0.0",
        "@smithy/util-body-length-node": "^4.0.0",
        "@smithy/util-defaults-mode-browser": "^4.0.17",
        "@smithy/util-defaults-mode-node": "^4.0.17",
        "@smithy/util-endpoints": "^3.0.6",
        "@smithy/util-middleware": "^4.0.4",
        "@smithy/util-retry": "^4.0.5",
        "@smithy/util-utf8": "^4.0.0",
        "tslib": "^2.6.2"
      },
```

```
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/client-sso": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/client-sso/-/client-
sso-3.821.0.tgz",
      "integrity": "sha512-aDEBZUKUd/
+Tvudi0d9KQlqt2OW2P27LATZX0jkNC8yVk4145bAPS04EYoqdKLuyUn/U33DibEOgKUpxZB12jQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-crypto/sha256-browser": "5.2.0",
        "@aws-crypto/sha256-js": "5.2.0",
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/middleware-host-header": "3.821.0",
        "@aws-sdk/middleware-logger": "3.821.0",
        "@aws-sdk/middleware-recursion-detection": "3.821.0",
        "@aws-sdk/middleware-user-agent": "3.821.0",
        "@aws-sdk/region-config-resolver": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@aws-sdk/util-endpoints": "3.821.0",
        "@aws-sdk/util-user-agent-browser": "3.821.0",
        "@aws-sdk/util-user-agent-node": "3.821.0",
        "@smithy/config-resolver": "^4.1.4",
        "@smithy/core": "^3.5.1",
        "@smithy/fetch-http-handler": "^5.0.4",
        "@smithy/hash-node": "^4.0.4",
        "@smithy/invalid-dependency": "^4.0.4",
        "@smithy/middleware-content-length": "^4.0.4",
        "@smithy/middleware-endpoint": "^4.1.9",
        "@smithy/middleware-retry": "^4.1.10",
        "@smithy/middleware-serde": "^4.0.8",
        "@smithy/middleware-stack": "^4.0.4",
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/node-http-handler": "^4.0.6",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/smithy-client": "^4.4.1",
        "@smithy/types": "^4.3.1",
        "@smithy/url-parser": "^4.0.4",
        "@smithy/util-base64": "^4.0.0",
        "@smithy/util-body-length-browser": "^4.0.0",
        "@smithy/util-body-length-node": "^4.0.0",
        "@smithy/util-defaults-mode-browser": "^4.0.17",
        "@smithy/util-defaults-mode-node": "^4.0.17",
        "@smithy/util-endpoints": "^3.0.6",
        "@smithy/util-middleware": "^4.0.4",
        "@smithy/util-retry": "^4.0.5",
        "@smithy/util-utf8": "^4.0.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/core": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/core/-/core-3.821.0.tgz",
```

```
      "integrity":
"sha512-8eB3wKbmfciQFmxFq7hAjy7mXdUs7vBOR5SwT0ZtQBg0Txc18Lc9tMViqqdO6/
KU7OukA6ib2IAVSjIJJEN7FQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/types": "3.821.0",
        "@smithy/core": "^3.5.1",
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/signature-v4": "^5.1.2",
        "@smithy/smithy-client": "^4.4.1",
        "@smithy/types": "^4.3.1",
        "@smithy/util-middleware": "^4.0.4",
        "fast-xml-parser": "4.4.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/credential-provider-cognito-identity": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/credential-provider-
cognito-identity/-/credential-provider-cognito-identity-3.821.0.tgz",
      "integrity":
"sha512-8ZdFwmSxvQv8QindA0DJ3YUT9FD8T9sA5hQWp3B9+Znkze29IiIadnsXY0Heo2/
FOFygxh8jRXiCWEie7/YpzA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/client-cognito-identity": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/credential-provider-env": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/credential-provider-env/-/
credential-provider-env-3.821.0.tgz",
      "integrity": "sha512-C+s/A72pd7CXwEsJj9+Uq9T726iIfIF18hGRY8o82xcIEfOyakiPnli
sku8zZOaAu+jm0CihbbYN4NyYNQ+HZQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
```

```
      }
    },
    "node_modules/@aws-sdk/credential-provider-http": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/credential-provider-http/-/
credential-provider-http-3.821.0.tgz",
      "integrity": "sha512-gIRzTLnAsRfRSNarCag7G7rhcHagz4x5nNTWRihQs5cwTOghEExDy7T
j5m4TEkv3dcTAsNn+l4tnR4nZXo6R+Q==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/fetch-http-handler": "^5.0.4",
        "@smithy/node-http-handler": "^4.0.6",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/smithy-client": "^4.4.1",
        "@smithy/types": "^4.3.1",
        "@smithy/util-stream": "^4.2.2",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/credential-provider-ini": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/credential-provider-ini/-/
credential-provider-ini-3.821.0.tgz",
      "integrity": "sha512-VRTrmsca8kBHtY1tTek1ce+XkK/H0fzodBKcilM/
qXjTyumMHPAzVAxKZfSvGC+28/pXyQzhOEyxZfw7giCiWA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/credential-provider-env": "3.821.0",
        "@aws-sdk/credential-provider-http": "3.821.0",
        "@aws-sdk/credential-provider-process": "3.821.0",
        "@aws-sdk/credential-provider-sso": "3.821.0",
        "@aws-sdk/credential-provider-web-identity": "3.821.0",
        "@aws-sdk/nested-clients": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/credential-provider-imds": "^4.0.6",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/shared-ini-file-loader": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/credential-provider-node": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/credential-provider-node/-/
credential-provider-node-3.821.0.tgz",
      "integrity": "sha512-oBgbcgOXWMgknAfhIdTeHSSVIv+k2LXN9oTbxu1r+
+o4WWBWrEQ8mHU0Zo9dfr7Uaoqi3pezYZznsBkXnMLEOg==",
```

```
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/credential-provider-env": "3.821.0",
        "@aws-sdk/credential-provider-http": "3.821.0",
        "@aws-sdk/credential-provider-ini": "3.821.0",
        "@aws-sdk/credential-provider-process": "3.821.0",
        "@aws-sdk/credential-provider-sso": "3.821.0",
        "@aws-sdk/credential-provider-web-identity": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/credential-provider-imds": "^4.0.6",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/shared-ini-file-loader": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/credential-provider-process": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/credential-provider-
process/-/credential-provider-process-3.821.0.tgz",
      "integrity": "sha512-e18ucfqKB3ICNj5RP/
FEdvUfhVK6E9MALOsl8pKP13mwegug46p/1BsZWACD5n+Zf9ViiiHxIO7td03zQixfwA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/shared-ini-file-loader": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/credential-provider-sso": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/credential-provider-sso/-/
credential-provider-sso-3.821.0.tgz",
      "integrity": "sha512-Dt+pheBLom4O/
egO4L75/72k9C1qtUOLl0F0h6lmqZe4Mvhz+wDtjoO/MdGC/P1q0kcIX/bBKr0NQ3cIvAH8pA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/client-sso": "3.821.0",
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/token-providers": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/shared-ini-file-loader": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
```

```
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/credential-provider-web-identity": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/credential-provider-web-
identity/-/credential-provider-web-identity-3.821.0.tgz",
      "integrity": "sha512-FF5wnRJkxSQaCVVvWNv53K1MhTMgH8d+O+MHTbkv51gVIgVATrtfFQM
KBLcEAxzXrgAliIO3LiNv+1TqqBZ+BA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/nested-clients": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/credential-providers": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/credential-providers/-/
credential-providers-3.821.0.tgz",
      "integrity": "sha512-ZkV7KlKD+rSW/
AP5zjSgMi+0xJ5TL5J6XVaP3IG5qyqBYTREJ8DbB/9YVUpYt2qtzpWUh/K43nmDEyfLd2YJog==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/client-cognito-identity": "3.821.0",
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/credential-provider-cognito-identity": "3.821.0",
        "@aws-sdk/credential-provider-env": "3.821.0",
        "@aws-sdk/credential-provider-http": "3.821.0",
        "@aws-sdk/credential-provider-ini": "3.821.0",
        "@aws-sdk/credential-provider-node": "3.821.0",
        "@aws-sdk/credential-provider-process": "3.821.0",
        "@aws-sdk/credential-provider-sso": "3.821.0",
        "@aws-sdk/credential-provider-web-identity": "3.821.0",
        "@aws-sdk/nested-clients": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/config-resolver": "^4.1.4",
        "@smithy/core": "^3.5.1",
        "@smithy/credential-provider-imds": "^4.0.6",
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/middleware-host-header": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/middleware-host-header/-/
```

```
middleware-host-header-3.821.0.tgz",
      "integrity": "sha512-xSMR+sopSeWGx5/4pAGhhfMvGBHioVBbqGvDs6pG64xfNwM5vq5s5v6
D04e2i+uSTj4qGa71dLUs5I0UzAK3sw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/types": "3.821.0",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/middleware-logger": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/middleware-logger/-/
middleware-logger-3.821.0.tgz",
      "integrity": "sha512-0cvI0ipf2tGx7fXYEEN5fBeZDz2RnHyb9xftSgUsEq7NBxjV0yTZfLJ
w6Za5rjE6snC80dRN8+bTNR1tuG89zA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/types": "3.821.0",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/middleware-recursion-detection": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/middleware-recursion-
detection/-/middleware-recursion-detection-3.821.0.tgz",
      "integrity": "sha512-efmaifbhBoqKG3bAoEfDdcM8hn1psF+4qa7ykWuYmfmah59JBeqHLfz
5W9m9JoTwoKPkFcVLWZxnyZzAnVBOIg==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/types": "3.821.0",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/middleware-user-agent": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/middleware-user-agent/-/
middleware-user-agent-3.821.0.tgz",
      "integrity": "sha512-rw8q3TxygMg3VrofN04QyWVCCyGwz3bVthYmBZZseENPWG3Krz1OCKc
yqjkTcAxMQlEywOske+GIiOasGKnJ3w==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
```

```
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@aws-sdk/util-endpoints": "3.821.0",
        "@smithy/core": "^3.5.1",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/nested-clients": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/nested-clients/-/nested-
clients-3.821.0.tgz",
      "integrity": "sha512-2IuHcUsWw44ftSEDYU4dvktTEqgyDvkOcfpoGC/
UmT4Qo6TVCP3U5tWEGpNK9nN+7nLvekruxxG/jaMt5/oWVw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-crypto/sha256-browser": "5.2.0",
        "@aws-crypto/sha256-js": "5.2.0",
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/middleware-host-header": "3.821.0",
        "@aws-sdk/middleware-logger": "3.821.0",
        "@aws-sdk/middleware-recursion-detection": "3.821.0",
        "@aws-sdk/middleware-user-agent": "3.821.0",
        "@aws-sdk/region-config-resolver": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@aws-sdk/util-endpoints": "3.821.0",
        "@aws-sdk/util-user-agent-browser": "3.821.0",
        "@aws-sdk/util-user-agent-node": "3.821.0",
        "@smithy/config-resolver": "^4.1.4",
        "@smithy/core": "^3.5.1",
        "@smithy/fetch-http-handler": "^5.0.4",
        "@smithy/hash-node": "^4.0.4",
        "@smithy/invalid-dependency": "^4.0.4",
        "@smithy/middleware-content-length": "^4.0.4",
        "@smithy/middleware-endpoint": "^4.1.9",
        "@smithy/middleware-retry": "^4.1.10",
        "@smithy/middleware-serde": "^4.0.8",
        "@smithy/middleware-stack": "^4.0.4",
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/node-http-handler": "^4.0.6",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/smithy-client": "^4.4.1",
        "@smithy/types": "^4.3.1",
        "@smithy/url-parser": "^4.0.4",
        "@smithy/util-base64": "^4.0.0",
        "@smithy/util-body-length-browser": "^4.0.0",
        "@smithy/util-body-length-node": "^4.0.0",
        "@smithy/util-defaults-mode-browser": "^4.0.17",
        "@smithy/util-defaults-mode-node": "^4.0.17",
        "@smithy/util-endpoints": "^3.0.6",
        "@smithy/util-middleware": "^4.0.4",
        "@smithy/util-retry": "^4.0.5",
        "@smithy/util-utf8": "^4.0.0",
        "tslib": "^2.6.2"
```

```
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/region-config-resolver": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/region-config-resolver/-/
region-config-resolver-3.821.0.tgz",
      "integrity": "sha512-t8og+lRCIIy5nlId0bScNpCkif8sc0LhmtaKsbm0ZPm3sCa/
WhCbSZibjbZ28FNjVCV+p0D9RYZx0VDDbtWyjw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/types": "3.821.0",
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/types": "^4.3.1",
        "@smithy/util-config-provider": "^4.0.0",
        "@smithy/util-middleware": "^4.0.4",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/token-providers": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/token-providers/-/token-
providers-3.821.0.tgz",
      "integrity": "sha512-
qJ7wgKhdxGbPg718zWXbCYKDuSWZNU3TSw64hPRW6FtbZrIyZxObpiTKC6DKwfsVoZZhHEoP/
imGykN1OdOTJA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/core": "3.821.0",
        "@aws-sdk/nested-clients": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/shared-ini-file-loader": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/types": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/types/-/types-3.821.0.tgz",
      "integrity": "sha512-Znroqdai1a90TlxGaJ+FK1lwC0fHpo97Xjsp5UKGR5JODYm7f9+/
fF17ebO1KdoBr/Rm0UIFiF5VmI8ts9F1eA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
```

```
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/util-endpoints": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/util-endpoints/-/util-
endpoints-3.821.0.tgz",
      "integrity": "sha512-Uknt/zUZnLE76zaAAPEayOeF5/4IZ2puTFXvcSCWHsi9m3tqbb9Uozl
nlVqvCZLCRWfQryZQoG2W4XSS3qgk5A==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/types": "3.821.0",
        "@smithy/types": "^4.3.1",
        "@smithy/util-endpoints": "^3.0.6",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/util-locate-window": {
      "version": "3.804.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/util-locate-window/-/util-
locate-window-3.804.0.tgz",
      "integrity": "sha512-zVoRfpmBVPodYlnMjgVjfGoEZagyRF5IPn3Uo6ZvOZp24chnW/
FRstH7ESDHDDRga4z3V+ElUQHKpFDXWyBW5A==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@aws-sdk/util-user-agent-browser": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/util-user-agent-browser/-/
util-user-agent-browser-3.821.0.tgz",
      "integrity": "sha512-
irWZHyM0Jr1xhC+38OuZ7JB6OXMLPZlj48thElpsO1ZSLRkLZx5+I7VV6k3sp2yZ7BYbKz/
G2ojSv4wdm7XTLw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/types": "3.821.0",
        "@smithy/types": "^4.3.1",
        "bowser": "^2.11.0",
        "tslib": "^2.6.2"
      }
    },
    "node_modules/@aws-sdk/util-user-agent-node": {
      "version": "3.821.0",
      "resolved": "https://registry.npmjs.org/@aws-sdk/util-user-agent-node/-/
util-user-agent-node-3.821.0.tgz",
      "integrity": "sha512-
YwMXc9EvuzJgnLBTyiQly2juPujXwDgcMHB0iSN92tHe7Dd1jJ1feBmTgdClaaqCeHFUaFpw+3JU/
ZUJ6LjR+A==",
```

```
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@aws-sdk/middleware-user-agent": "3.821.0",
        "@aws-sdk/types": "3.821.0",
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      },
      "peerDependencies": {
        "aws-crt": ">=1.0.0"
      },
      "peerDependenciesMeta": {
        "aws-crt": {
          "optional": true
        }
      }
    },
    "node_modules/@google/generative-ai": {
      "version": "0.24.1",
      "resolved": "https://registry.npmjs.org/@google/generative-ai/-/generative-
ai-0.24.1.tgz",
      "integrity": "sha512-MqO+MLfM6kjxcKoy0p1wRzG3b4ZZXtPI+z2IE26UogS2Cm/
XHO+7gGRBh6gcJsOiIVoH93UwKvW4HdgiOZCy9Q==",
      "license": "Apache-2.0",
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@mapbox/node-pre-gyp": {
      "version": "1.0.11",
      "resolved": "https://registry.npmjs.org/@mapbox/node-pre-gyp/-/node-pre-
gyp-1.0.11.tgz",
      "integrity": "sha512-Yhlar6v9WQgUp/
He7BdgzOz8lqMQ8sU+jkCq7Wx8Myc5YFJLbEe7lgui/V7G1qB1DJykHSGwreceSaD60Y0PUQ==",
      "license": "BSD-3-Clause",
      "dependencies": {
        "detect-libc": "^2.0.0",
        "https-proxy-agent": "^5.0.0",
        "make-dir": "^3.1.0",
        "node-fetch": "^2.6.7",
        "nopt": "^5.0.0",
        "npmlog": "^5.0.1",
        "rimraf": "^3.0.2",
        "semver": "^7.3.5",
        "tar": "^6.1.11"
      },
      "bin": {
        "node-pre-gyp": "bin/node-pre-gyp"
      }
    },
    "node_modules/@mongodb-js/saslprep": {
      "version": "1.2.2",
      "resolved": "https://registry.npmjs.org/@mongodb-js/saslprep/-/
saslprep-1.2.2.tgz",
      "integrity": "sha512-
```

```
EB0O3SCSNRUFk66iRCpI+cXzIjdswfCs7F6nOC3RAGJ7xr5YhaicvsRwJ9eyzYvYRlCSDUO/
c7g4yNulxKC1WA==",
      "license": "MIT",
      "optional": true,
      "dependencies": {
        "sparse-bitfield": "^3.0.3"
      }
    },
    "node_modules/@smithy/abort-controller": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/abort-controller/-/abort-
controller-4.0.4.tgz",
      "integrity": "sha512-gJnEjZMvigPDQWHrW3oPrFhQtkrgqBkyjj3pCIdF3A5M6vsZODG93KN
lfJprv6bp4245bdT32fsHK4kkH3KYDA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/config-resolver": {
      "version": "4.1.4",
      "resolved": "https://registry.npmjs.org/@smithy/config-resolver/-/config-
resolver-4.1.4.tgz",
      "integrity": "sha512-prmU+rDddxHOH0oNcwemL+SwnzcG65sBF2yXRO7aeXIn/
xTlq2pX7JLVbkBnVLowHLg4/OL4+jBmv9hVrVGS+w==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/types": "^4.3.1",
        "@smithy/util-config-provider": "^4.0.0",
        "@smithy/util-middleware": "^4.0.4",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/core": {
      "version": "3.5.1",
      "resolved": "https://registry.npmjs.org/@smithy/core/-/core-3.5.1.tgz",
      "integrity": "sha512-xSw7bZEFKwOKrm/
iv8e2BLt2ur98YZdrRD6nII8ditQeUsY2Q1JmIQ0rpILOhaLKYxxG2ivnoOpokzr9qLyDWA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/middleware-serde": "^4.0.8",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/types": "^4.3.1",
        "@smithy/util-base64": "^4.0.0",
        "@smithy/util-body-length-browser": "^4.0.0",
        "@smithy/util-middleware": "^4.0.4",
        "@smithy/util-stream": "^4.2.2",
        "@smithy/util-utf8": "^4.0.0",
```

```
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/credential-provider-imds": {
      "version": "4.0.6",
      "resolved": "https://registry.npmjs.org/@smithy/credential-provider-imds/-/
credential-provider-imds-4.0.6.tgz",
      "integrity": "sha512-hKMWcANhUiNbCJouYkZ9V3+/
Qf9pteR1dnwgdyzR09R4ODEYx8BbUysHwRSyex4rZ9zapddZhLFTnT4ZijR4pw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "@smithy/url-parser": "^4.0.4",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/fetch-http-handler": {
      "version": "5.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/fetch-http-handler/-/fetch-
http-handler-5.0.4.tgz",
      "integrity": "sha512-AMtBR5pHppYMVD7z7G+OlHHAcgAN7v0kVKEpHuTO4Gb199Gowh0taYi
9oDStFeUhetkeP55JLSVlTW1n9rFtUw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/querystring-builder": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "@smithy/util-base64": "^4.0.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/hash-node": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/hash-node/-/hash-
node-4.0.4.tgz",
      "integrity": "sha512-qnbTPUhCVnCgBp4z4BUJUhOEkVwxiEi1cyFM+Zj6o+aY8OFGxUQleKW
q8ltgp3dujuhXojIvJWdoqpm6dVO3lQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "@smithy/util-buffer-from": "^4.0.0",
        "@smithy/util-utf8": "^4.0.0",
        "tslib": "^2.6.2"
      },
      "engines": {
```

```
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/invalid-dependency": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/invalid-dependency/-/
invalid-dependency-4.0.4.tgz",
      "integrity": "sha512-bNYMi7WKTJHu0gn26wg8OscncTt1t2b8KcsZxvOv56XA6cyXtOAAAaN
P7+m45xfppXfOatXF3Sb1MNsLUgVLTw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/is-array-buffer": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/@smithy/is-array-buffer/-/is-array-
buffer-4.0.0.tgz",
      "integrity": "sha512-
saYhF8ZZNoJDTvJBEWgeBccCg+yvp1CX+ed12yORU3NilJScfc6gfch2oVb4QgxZrGUx3/ZJlb+c/
dJbyupxlw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/middleware-content-length": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/middleware-content-length/-/
middleware-content-length-4.0.4.tgz",
      "integrity": "sha512-
F7gDyfI2BB1Kc+4M6rpuOLne5LOcEknH1n6UQB69qv+HucXBR1rkzXBnQTB2q46sFy1PM/
zuSJOB532yc8bg3w==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/middleware-endpoint": {
      "version": "4.1.9",
      "resolved": "https://registry.npmjs.org/@smithy/middleware-endpoint/-/
middleware-endpoint-4.1.9.tgz",
      "integrity": "sha512-AjDgX4UjORLltD/LZCBQTwjQqEfyrx/
GeDTHcYLzIgf87pIT70tMWnN87NQpJru1K4ITirY2htSOxNECZJCBOg==",
```

```
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/core": "^3.5.1",
        "@smithy/middleware-serde": "^4.0.8",
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/shared-ini-file-loader": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "@smithy/url-parser": "^4.0.4",
        "@smithy/util-middleware": "^4.0.4",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/middleware-retry": {
      "version": "4.1.10",
      "resolved": "https://registry.npmjs.org/@smithy/middleware-retry/-/
middleware-retry-4.1.10.tgz",
      "integrity": "sha512-
RyhcA3sZIIvAo6r48b2Nx2qfg0OnyohlaV0fw415xrQyx5HQ2bvHl9vs/
WBiDXIP49mCfws5wX4308c9Pi/isw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/service-error-classification": "^4.0.5",
        "@smithy/smithy-client": "^4.4.1",
        "@smithy/types": "^4.3.1",
        "@smithy/util-middleware": "^4.0.4",
        "@smithy/util-retry": "^4.0.5",
        "tslib": "^2.6.2",
        "uuid": "^9.0.1"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/middleware-retry/node_modules/uuid": {
      "version": "9.0.1",
      "resolved": "https://registry.npmjs.org/uuid/-/uuid-9.0.1.tgz",
      "integrity": "sha512-b+1eJOlsR9K8HJpow9Ok3fiWOWSIcIzXodvv0rQjVoOVNpWMpxf1wZN
pt4y9h10odCNrqnYp1OBzRktckBe3sA==",
      "funding": [
        "https://github.com/sponsors/broofa",
        "https://github.com/sponsors/ctavan"
      ],
      "license": "MIT",
      "optional": true,
      "bin": {
        "uuid": "dist/bin/uuid"
      }
    },
    "node_modules/@smithy/middleware-serde": {
      "version": "4.0.8",
      "resolved": "https://registry.npmjs.org/@smithy/middleware-serde/-/
middleware-serde-4.0.8.tgz",
```

```
      "integrity": "sha512-
iSSl7HJoJaGyMIoNn2B7czghOVwJ9nD7TMvLhMWeSB5vt0TnEYyRRqPJu/
TqW76WScaNvYYB8nRoiBHR9S1Ddw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/middleware-stack": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/middleware-stack/-/
middleware-stack-4.0.4.tgz",
      "integrity": "sha512-kagK5ggDrBUCCzI93ft6DjteNSfY8Ulr83UtySog/h09lTIOAJ/
xUSObutanlPT0nhoHAkpmW9V5K8oPyLh+QA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/node-config-provider": {
      "version": "4.1.3",
      "resolved": "https://registry.npmjs.org/@smithy/node-config-provider/-/node-
config-provider-4.1.3.tgz",
      "integrity": "sha512-
HGHQr2s59qaU1lrVH6MbLlmOBxadtzTsoO4c+bF5asdgVik3I8o7JIOzoeqWc5MjVa+vD36/
LWE0iXKpNqooRw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/property-provider": "^4.0.4",
        "@smithy/shared-ini-file-loader": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/node-http-handler": {
      "version": "4.0.6",
      "resolved": "https://registry.npmjs.org/@smithy/node-http-handler/-/node-
http-handler-4.0.6.tgz",
      "integrity": "sha512-NqbmSz7AW2rvw4kXhKGrYTiJVDHnMsFnX4i+/
FzcZAfbOBauPYs2ekuECkSbtqaxETLLTu9Rl/ex6+I2BKErPA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/abort-controller": "^4.0.4",
```

```
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/querystring-builder": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/property-provider": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/property-provider/-/
property-provider-4.0.4.tgz",
      "integrity": "sha512-qHJ2sSgu4FqF4U/5UUp4DhXNmdTrgmoAai6oQiM+c5RZ/
sbDwJ12qxB1M6FnP+Tn/ggkPZf9ccn4jqKSINaquw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/protocol-http": {
      "version": "5.1.2",
      "resolved": "https://registry.npmjs.org/@smithy/protocol-http/-/protocol-
http-5.1.2.tgz",
      "integrity": "sha512-rOG5cNLBXovxIrICSBm95dLqzfvxjEmuZx4KK3hWwPFHGdW3lxY0fZN
Xfv2zebfRO7sJZ5pKJYHScsqopeIWtQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/querystring-builder": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/querystring-builder/-/
querystring-builder-4.0.4.tgz",
      "integrity": "sha512-SwREZcDnEYoh9tLNgMbpop+UTGq44Hl9tdj3rf+yeLcfH7+J8OXEBaM
c2kDxtyRHu8BhSg9ADEx0gFHvpJgU8w==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "@smithy/util-uri-escape": "^4.0.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/querystring-parser": {
```

```
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/querystring-parser/-/
querystring-parser-4.0.4.tgz",
      "integrity": "sha512-6yZf53i/qB8gRHH/l2ZwUG5xgkPgQF15/
KxH0DdXMDHjesA9MeZje/853ifkSY0x4m5S+dfDZ+c4x439PF0M2w==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/service-error-classification": {
      "version": "4.0.5",
      "resolved": "https://registry.npmjs.org/@smithy/service-error-
classification/-/service-error-classification-4.0.5.tgz",
      "integrity": "sha512-LvcfhrnCBvCmTee81pRlh1F39yTS/+kYleVeLCwNtkY8wtGg8V/
ca9rbZZvYIl8OjlMtL6KIjaiL/lgVqHD2nA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/shared-ini-file-loader": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/shared-ini-file-loader/-/
shared-ini-file-loader-4.0.4.tgz",
      "integrity": "sha512-63X0260LoFBjrHifPDs+nM9tV0VMkOTl4JRMYNuKh/
f5PauSjowTfvF3LogfkWdcPoxsA9UjqEOgjeYIbhb7Nw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/signature-v4": {
      "version": "5.1.2",
      "resolved": "https://registry.npmjs.org/@smithy/signature-v4/-/signature-
v4-5.1.2.tgz",
      "integrity": "sha512-d3+U/VpX7a60seHziWnVZOHuEgJlclufjkS6zhXvxcJgkJq4UWdH5eO
BLzHRMx6gXjsdT9h6lfpmLzbrdupHgQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/is-array-buffer": "^4.0.0",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/types": "^4.3.1",
        "@smithy/util-hex-encoding": "^4.0.0",
```

```
        "@smithy/util-middleware": "^4.0.4",
        "@smithy/util-uri-escape": "^4.0.0",
        "@smithy/util-utf8": "^4.0.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/smithy-client": {
      "version": "4.4.1",
      "resolved": "https://registry.npmjs.org/@smithy/smithy-client/-/smithy-
client-4.4.1.tgz",
      "integrity": "sha512-XPbcHRfd0iwx8dY5XCBCGyI7uweMW0oezYezxXcG8ANgvZ5YPuC6Ylh
+n0bTHpdU3SCMZOnhzgVklYz+p3fIhw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/core": "^3.5.1",
        "@smithy/middleware-endpoint": "^4.1.9",
        "@smithy/middleware-stack": "^4.0.4",
        "@smithy/protocol-http": "^5.1.2",
        "@smithy/types": "^4.3.1",
        "@smithy/util-stream": "^4.2.2",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/types": {
      "version": "4.3.1",
      "resolved": "https://registry.npmjs.org/@smithy/types/-/types-4.3.1.tgz",
      "integrity": "sha512-UqKOQBL2x6+HWl3P+3QqFD4ncKq0I8Nuz9QItGv5WuKuMHuuwlhvqcZ
CoXGfc+P1QmfJE7VieykoYYmrOoFJxA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/url-parser": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/url-parser/-/url-
parser-4.0.4.tgz",
      "integrity": "sha512-eMkc144MuN7B0TDA4U2fKs+BqczVbk3W+qIvcoCY6D1JY3hnAdCuhCZ
ODC+GAeaxj0p6Jroz4+XMUn3PCxQQeQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/querystring-parser": "^4.0.4",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
```

```
      }
    },
    "node_modules/@smithy/util-base64": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-base64/-/util-
base64-4.0.0.tgz",
      "integrity": "sha512-CvHfCmO2mchox9kjrtzoHkWHxjHZzaFojLc8quxXY7WAAMAg43nuxwv
95tATVgQFNDwd4M9S1qFzj40Ul41Kmg==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/util-buffer-from": "^4.0.0",
        "@smithy/util-utf8": "^4.0.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-body-length-browser": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-body-length-browser/-/
util-body-length-browser-4.0.0.tgz",
      "integrity": "sha512-sNi3DL0/k64/
LO3A256M+m3CDdG6V7WKWHdAiBBMUN8S3hK3aMPhwnPik2A/a2ONN+9doY9UxaLfgqsIRg69QA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-body-length-node": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-body-length-node/-/
util-body-length-node-4.0.0.tgz",
      "integrity": "sha512-q0iDP3VsZzqJyje8xJWEJCNIu3lktUGVoSy1KB0UWym2CL1siV3artm
+u1DFYTLejpsrdGyCSWBdGNjJzfDPjg==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-buffer-from": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-buffer-from/-/util-
buffer-from-4.0.0.tgz",
      "integrity": "sha512-9TOQ7781sZvddgO8nxueKi3+yGvkY35kotA0Y6BWRajAv8jjmigQ1sB
wz0UX47pQMYXJPahSKEKYFgt+rXdcug==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/is-array-buffer": "^4.0.0",
```

```
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-config-provider": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-config-provider/-/util-
config-provider-4.0.0.tgz",
      "integrity": "sha512-L1RBVzLyfE8OXH+1hsJ8p+acNUSirQnWQ6/
EgpchV88G6zGBTDPdXiiExei6Z1wR2RxYvxY/XLw6AMNCCt8H3w==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-defaults-mode-browser": {
      "version": "4.0.17",
      "resolved": "https://registry.npmjs.org/@smithy/util-defaults-mode-
browser/-/util-defaults-mode-browser-4.0.17.tgz",
      "integrity": "sha512-HXq5181qnXmIwB7VrwqwP8rsJybHMoYuJnNoXy4PROs2pfSI4sWDMAS
F2i+7Lo+u64Y6xowhegcdxczowgJtZg==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/property-provider": "^4.0.4",
        "@smithy/smithy-client": "^4.4.1",
        "@smithy/types": "^4.3.1",
        "bowser": "^2.11.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-defaults-mode-node": {
      "version": "4.0.17",
      "resolved": "https://registry.npmjs.org/@smithy/util-defaults-mode-node/-/
util-defaults-mode-node-4.0.17.tgz",
      "integrity": "sha512-RfU2A5LjFhEHw4Nwl1GZNitK4AUWu5jGtigAUDoQtfDUvYHpQxcuLw2
QGAdKDtKRflIiHSZ8wXBDR36H9R2Ang==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/config-resolver": "^4.1.4",
        "@smithy/credential-provider-imds": "^4.0.6",
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/property-provider": "^4.0.4",
        "@smithy/smithy-client": "^4.4.1",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
```

```
      }
    },
    "node_modules/@smithy/util-endpoints": {
      "version": "3.0.6",
      "resolved": "https://registry.npmjs.org/@smithy/util-endpoints/-/util-
endpoints-3.0.6.tgz",
      "integrity": "sha512-YARl3tFL3WgPuLzljRUnrS2ngLiUtkwhQtj8PAL13XZSyUiNLQxwG3f
BBq3QXFqGFUXepIN73pINp3y8c2nBmA==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/node-config-provider": "^4.1.3",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-hex-encoding": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-hex-encoding/-/util-
hex-encoding-4.0.0.tgz",
      "integrity": "sha512-Yk5mLhHtfIgW2W2WQZWSg5kuMZCVbvhFmC7rV4IO2QqnZdbEFPmQnCc
GMAX2z/8Qj3B9hYYNjZOhWym+RwhePw==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-middleware": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/@smithy/util-middleware/-/util-
middleware-4.0.4.tgz",
      "integrity": "sha512-9MLKmkBmf4PRb0ONJikCbCwORACcil6gUWojwARCClT7RmLzF04hUR4
WdRprIXal7XVyrddadYNfp2eF3nrvtQ==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-retry": {
      "version": "4.0.5",
      "resolved": "https://registry.npmjs.org/@smithy/util-retry/-/util-
retry-4.0.5.tgz",
      "integrity": "sha512-V7MSjVDTlEt/
plmOFBn1762Dyu5uqMrV2Pl2X0dYk4XvWfdWJNe9Bs5Bzb56wkCuiWjSfClVMGcsuKrGj7S/yg==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
```

```
        "@smithy/service-error-classification": "^4.0.5",
        "@smithy/types": "^4.3.1",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-stream": {
      "version": "4.2.2",
      "resolved": "https://registry.npmjs.org/@smithy/util-stream/-/util-
stream-4.2.2.tgz",
      "integrity": "sha512-aI+GLi7MJoVxg24/3J1ipwLoYzgkB4kUfogZfnslcYlynj3xsQ0e7vk
4TnTro9hhsS5PvX1mwmkRqqHQjwcU7w==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/fetch-http-handler": "^5.0.4",
        "@smithy/node-http-handler": "^4.0.6",
        "@smithy/types": "^4.3.1",
        "@smithy/util-base64": "^4.0.0",
        "@smithy/util-buffer-from": "^4.0.0",
        "@smithy/util-hex-encoding": "^4.0.0",
        "@smithy/util-utf8": "^4.0.0",
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-uri-escape": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-uri-escape/-/util-uri-
escape-4.0.0.tgz",
      "integrity":
"sha512-77yfbCbQMtgtTylO9itEAdpPXSog3ZxMe09AEhm0dU0NLTalV70ghDZFR+Nfi1C60jnJoh/
Re4090/DuZh2Omg==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "tslib": "^2.6.2"
      },
      "engines": {
        "node": ">=18.0.0"
      }
    },
    "node_modules/@smithy/util-utf8": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/@smithy/util-utf8/-/util-
utf8-4.0.0.tgz",
      "integrity": "sha512-b+zebfKCfRdgNJDknHCob3O7FpeYQN6ZG6YLExMcasDHsCXlsXCEuiP
ZeLnJLpwa5dvPetGlnGCiMHuLwGvFow==",
      "license": "Apache-2.0",
      "optional": true,
      "dependencies": {
        "@smithy/util-buffer-from": "^4.0.0",
        "tslib": "^2.6.2"
      },
      "engines": {
```

```
        "node": ">=18.0.0"
      }
    },
    "node_modules/@socket.io/component-emitter": {
      "version": "3.1.2",
      "resolved": "https://registry.npmjs.org/@socket.io/component-emitter/-/
component-emitter-3.1.2.tgz",
      "integrity": "sha512-9BCxFwvbGg/
RsZK9tjXd8s4UcwR0MWeFQ1XEKIQVVvAGJyINdrqKMcTRyLoK8Rse1GjzLV9cwjWV1olXRWEXVA==",
      "license": "MIT"
    },
    "node_modules/@swc/helpers": {
      "version": "0.5.17",
      "resolved": "https://registry.npmjs.org/@swc/helpers/-/helpers-0.5.17.tgz",
      "integrity": "sha512-5IKx/Y13RsYd+sauPb2x+U/xZikHjolzfuDgTAl/
Tdf3Q8rslRvC19NKDLgAJQ6wsqADk10ntlv08nPFw/gO/A==",
      "license": "Apache-2.0",
      "dependencies": {
        "tslib": "^2.8.0"
      }
    },
    "node_modules/@types/cors": {
      "version": "2.8.18",
      "resolved": "https://registry.npmjs.org/@types/cors/-/cors-2.8.18.tgz",
      "integrity": "sha512-nX3d0sxJW41CqQvfOzVG1NCTXfFDrDWIghCZncpHeWlVFd81zxB/
DLhg7avFg6eHLCRX7ckBmoIIcqa++upvJA==",
      "license": "MIT",
      "dependencies": {
        "@types/node": "*"
      }
    },
    "node_modules/@types/node": {
      "version": "22.15.3",
      "resolved": "https://registry.npmjs.org/@types/node/-/node-22.15.3.tgz",
      "integrity": "sha512-lX7HFZeHf4QG/J7tBZqrCAXwz9J5RD56Y6MpP0eJkka8p+K0RY/
yBTW7CYFJ4VGCclxqOLKmiGP5juQc6MKgcw==",
      "license": "MIT",
      "dependencies": {
        "undici-types": "~6.21.0"
      }
    },
    "node_modules/@types/webidl-conversions": {
      "version": "7.0.3",
      "resolved": "https://registry.npmjs.org/@types/webidl-conversions/-/webidl-
conversions-7.0.3.tgz",
      "integrity": "sha512-
CiJJvcRtIgzadHCYXw7dqEnMNRjhGZlYK05Mj9OyktqV8uVT8fD2BFOB7S1uwBE3Kj2Z+4UyPmFw/Ixgw/
LAlA==",
      "license": "MIT"
    },
    "node_modules/@types/whatwg-url": {
      "version": "8.2.2",
      "resolved": "https://registry.npmjs.org/@types/whatwg-url/-/whatwg-
url-8.2.2.tgz",
      "integrity": "sha512-
FtQu10RWgn3D9U4aazdwIE2yzphmTJREDqNdODHrbrZmmMqI0vMheC/6NE/
J1Yveaj8H+ela+YwWTjq5PGmuhA==",
      "license": "MIT",
      "dependencies": {
```

```
        "@types/node": "*",
        "@types/webidl-conversions": "*"
      }
    },
    "node_modules/abbrev": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/abbrev/-/abbrev-1.1.1.tgz",
      "integrity": "sha512-nne9/IiQ/hzIhY6pdDnbBtz7DjPTKrY00P/
zvPSm5pOFkl6xuGrGnXn/VtTNNfNtAfZ9/1RtehkszU9qcTii0Q==",
      "license": "ISC"
    },
    "node_modules/accepts": {
      "version": "1.3.8",
      "resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.8.tgz",
      "integrity": "sha512-PYAthTa2m2VKxuvSD3DPC/
Gy+U+sOA1LAuT8mkmRuvw+NACSaeXEQ+NHcVF7rONl6qcaxV3Uuemwawk+7+SJLw==",
      "license": "MIT",
      "dependencies": {
        "mime-types": "~2.1.34",
        "negotiator": "0.6.3"
      },
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/agenda": {
      "version": "5.0.0",
      "resolved": "https://registry.npmjs.org/agenda/-/agenda-5.0.0.tgz",
      "integrity": "sha512-jOoa7PvARpst/y2PI8h0wph4NmcjYJ/4wzFhQcHUbNgN+Hte/9h/
MzKE0ZmHfIwdsSlnv3rhbBQ3Zd/gwFkThg==",
      "license": "MIT",
      "dependencies": {
        "cron-parser": "^3.5.0",
        "date.js": "~0.3.3",
        "debug": "~4.3.4",
        "human-interval": "~2.0.1",
        "moment-timezone": "~0.5.37",
        "mongodb": "^4.11.0"
      },
      "engines": {
        "node": ">=12.9.0"
      }
    },
    "node_modules/agenda/node_modules/bson": {
      "version": "4.7.2",
      "resolved": "https://registry.npmjs.org/bson/-/bson-4.7.2.tgz",
      "integrity": "sha512-Ry9wCtIZ5kGqkJoi6aD8KjxFZEx78guTQDnpXWiNthsxzrxAK/
i8E6pCHAIZTbaEFWcOCvbecMukfK7XUvyLpQ==",
      "license": "Apache-2.0",
      "dependencies": {
        "buffer": "^5.6.0"
      },
      "engines": {
        "node": ">=6.9.0"
      }
    },
    "node_modules/agenda/node_modules/debug": {
      "version": "4.3.7",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.3.7.tgz",
```

```
      "integrity": "sha512-Er2nc/
H7RrMXZBFCEim6TCmMk02Z8vLC2Rbi1KEBggpo0fS6l0S1nnapwmIi3yW/
+GOJap1Krg4w0Hg80oCqgQ==",
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
    },
    "node_modules/agenda/node_modules/mongodb": {
      "version": "4.17.2",
      "resolved": "https://registry.npmjs.org/mongodb/-/mongodb-4.17.2.tgz",
      "integrity": "sha512-
mLV7SEiov2LHleRJPMPrK2PMyhXFZt2UQLC4VD4pnth3jMjYKHhtqfwwkkvS/NXuo/
Fp3vbhaNcXrIDaLRb9Tg==",
      "license": "Apache-2.0",
      "dependencies": {
        "bson": "^4.7.2",
        "mongodb-connection-string-url": "^2.6.0",
        "socks": "^2.7.1"
      },
      "engines": {
        "node": ">=12.9.0"
      },
      "optionalDependencies": {
        "@aws-sdk/credential-providers": "^3.186.0",
        "@mongodb-js/saslprep": "^1.1.0"
      }
    },
    "node_modules/agenda/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/agent-base": {
      "version": "6.0.2",
      "resolved": "https://registry.npmjs.org/agent-base/-/agent-base-6.0.2.tgz",
      "integrity": "sha512-RZNwNclF7+MS/8bDg70amg32dyeZGZxiDuQmZxKLAlQjr3jGyLx+4Kk
k58UO7D2QdgFIQCovuSuZESne6RG6XQ==",
      "license": "MIT",
      "dependencies": {
        "debug": "4"
      },
      "engines": {
        "node": ">= 6.0.0"
      }
    },
    "node_modules/agent-base/node_modules/debug": {
      "version": "4.4.0",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.4.0.tgz",
```

```
      "integrity": "sha512-6WTZ/IxCY/T6BALoZHaE4ctp9xm+Z5kY/
pzYaCHRFeyVhojxlrm+46y68HA6hr0TcwEssoxNiDEUJQjfPZ/RYA==",
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
    },
    "node_modules/agent-base/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/ansi-regex": {
      "version": "5.0.1",
      "resolved": "https://registry.npmjs.org/ansi-regex/-/ansi-regex-5.0.1.tgz",
      "integrity": "sha512-
quJQXlTSUGL2LH9SUXo8VwsY4soanhgo6LNSm84E1LBcE8s3O0wpdiRzyR9z/
ZZJMlMWv37qOOb9pdJlMUEKFQ==",
      "license": "MIT",
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/anymatch": {
      "version": "3.1.3",
      "resolved": "https://registry.npmjs.org/anymatch/-/anymatch-3.1.3.tgz",
      "integrity": "sha512-
KMReFUr0B4t+D+OBkjR3KYqvocp2XaSzO55UcB6mgQMd3KbcE+mWTyvVV7D/
zsdEbNnV6acZUutkiHQXvTr1Rw==",
      "dev": true,
      "license": "ISC",
      "dependencies": {
        "normalize-path": "^3.0.0",
        "picomatch": "^2.0.4"
      },
      "engines": {
        "node": ">= 8"
      }
    },
    "node_modules/append-field": {
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/append-field/-/append-
field-1.0.0.tgz",
      "integrity": "sha512-klpgFSWLW1ZEs8svjfb7g4qWY0YS5imI82dTg+QahUvJ8YqAY0P10Uk
8tTyh9ZGuYEZEMaeJYCF5BFuX552hsw==",
      "license": "MIT"
    },
    "node_modules/aproba": {
      "version": "2.0.0",
```

```
      "resolved": "https://registry.npmjs.org/aproba/-/aproba-2.0.0.tgz",
      "integrity": "sha512-lYe4Gx7QT+MKGbDsA+Z+he/Wtef0BiwDOlK/XkBrdfsh9J/
jPPXbX0tE9x9cl27Tmu5gg3QUbUrQYa/y+KOHPQ==",
      "license": "ISC"
    },
    "node_modules/are-we-there-yet": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/are-we-there-yet/-/are-we-there-
yet-2.0.0.tgz",
      "integrity": "sha512-Ci/qENmwHnsYo9xKIcUJN5LeDKdJ6R1Z1j9V/J5wyq8nh/
mYPEpIKJbBZXtZjG04HiK7zV/p6Vs9952MrMeUIw==",
      "deprecated": "This package is no longer supported.",
      "license": "ISC",
      "dependencies": {
        "delegates": "^1.0.0",
        "readable-stream": "^3.6.0"
      },
      "engines": {
        "node": ">=10"
      }
    },
    "node_modules/array-flatten": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/array-flatten/-/array-
flatten-1.1.1.tgz",
      "integrity": "sha512-PCVAQswWemu6UdxsDFFX/
+gVeYqKAod3D3UVm91jHwynguOwAvYPhx8nNlM++NqRcK6CxxpUafjmhIdKiHibqg==",
      "license": "MIT"
    },
    "node_modules/asynckit": {
      "version": "0.4.0",
      "resolved": "https://registry.npmjs.org/asynckit/-/asynckit-0.4.0.tgz",
      "integrity": "sha512-Oei9OH4tRh0YqU3GxhX79dM/
mwVgvbZJaSNaRk+bshkj0S5cfHcgYakreBjrHwatXKbz+IoIdYLxrKim2MjW0Q==",
      "license": "MIT"
    },
    "node_modules/axios": {
      "version": "1.9.0",
      "resolved": "https://registry.npmjs.org/axios/-/axios-1.9.0.tgz",
      "integrity": "sha512-re4CqKTJaURpzbLHtIi6XpDv20/CnpXOtjRY5/
CU32L8gU8ek9UIivcfvSWvmKEngmVbrUtPpdDwWDWL7DNHvg==",
      "license": "MIT",
      "dependencies": {
        "follow-redirects": "^1.15.6",
        "form-data": "^4.0.0",
        "proxy-from-env": "^1.1.0"
      }
    },
    "node_modules/balanced-match": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/balanced-match/-/balanced-
match-1.0.2.tgz",
      "integrity": "sha512-3oSeUO0TMV67hN1AmbXsK4yaqU7tjiHlbxRDZOpH0KW9+CeX4bRAaX0
Anxt0tx2MrpRpWwQaPwIlISEJhYU5Pw==",
      "license": "MIT"
    },
    "node_modules/base64-js": {
      "version": "1.5.1",
      "resolved": "https://registry.npmjs.org/base64-js/-/base64-js-1.5.1.tgz",
```

```
      "integrity": "sha512-AKpaYlHn8t4SVbOHCy+b5+KKgvR4vrsD8vbvrbiQJps7fKDTkjkDry6
ji0rUJjC0kzbNePLwzxq8iypo41qeWA==",
      "funding": [
        {
          "type": "github",
          "url": "https://github.com/sponsors/feross"
        },
        {
          "type": "patreon",
          "url": "https://www.patreon.com/feross"
        },
        {
          "type": "consulting",
          "url": "https://feross.org/support"
        }
      ],
      "license": "MIT"
    },
    "node_modules/base64id": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/base64id/-/base64id-2.0.0.tgz",
      "integrity": "sha512-lGe34o6EHj9y3Kts9R4ZYs/Gr+6N7MCaMlIFA3F1R2O5/
m7K06AxfSeO5530PEERE6/WyEg3lsuyw4GHlPZHog==",
      "license": "MIT",
      "engines": {
        "node": "^4.5.0 || >= 5.9"
      }
    },
    "node_modules/basic-auth": {
      "version": "2.0.1",
      "resolved": "https://registry.npmjs.org/basic-auth/-/basic-auth-2.0.1.tgz",
      "integrity": "sha512-NF+epuEdnUYVlGuhaxbbq+dvJttwLnGY+YixlXlME5KpQ5W3CnXA5cV
TneY3SPbPDRkcjMbifrwmFYcClgOZeg==",
      "license": "MIT",
      "dependencies": {
        "safe-buffer": "5.1.2"
      },
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/basic-auth/node_modules/safe-buffer": {
      "version": "5.1.2",
      "resolved": "https://registry.npmjs.org/safe-buffer/-/safe-
buffer-5.1.2.tgz",
      "integrity": "sha512-Gd2UZBJDkXlY7GbJxfsE8/
nvKkUEU1G38c1siN6QP6a9PT9MmHB8GnpscSmMJSoF8LOIrt8ud/wPtojys4G6+g==",
      "license": "MIT"
    },
    "node_modules/bcrypt": {
      "version": "5.1.1",
      "resolved": "https://registry.npmjs.org/bcrypt/-/bcrypt-5.1.1.tgz",
      "integrity": "sha512-AGBHOG5hPYZ5Xl9KXzU5iKq9516yEmvCKDg3ecP5kX2aB6UqTeXZxk2
ELnDgDm6BQSMlLt9rDB4LoSMx0rYwww==",
      "hasInstallScript": true,
      "license": "MIT",
      "dependencies": {
        "@mapbox/node-pre-gyp": "^1.0.11",
        "node-addon-api": "^5.0.0"
```

```
      },
      "engines": {
        "node": ">= 10.0.0"
      }
    },
    "node_modules/bcryptjs": {
      "version": "2.4.3",
      "resolved": "https://registry.npmjs.org/bcryptjs/-/bcryptjs-2.4.3.tgz",
      "integrity": "sha512-V/Hy/X9Vt7f3BbPJEi8BdVFMByHi+jNXrYkW3huaybV/
kQ0KJg0Y6PkEMbn+zeT+i+SiKZ/HMqJGIIt4LZDqNQ==",
      "license": "MIT"
    },
    "node_modules/binary-extensions": {
      "version": "2.3.0",
      "resolved": "https://registry.npmjs.org/binary-extensions/-/binary-
extensions-2.3.0.tgz",
      "integrity": "sha512-Ceh+7ox5qe7LJuLHoY0feh3pHuUDHAcRUeyL2VYghZwfpkNIy/
+8Ocg0a3UuSoYzavmylwuLWQOf3hl0jjMMIw==",
      "dev": true,
      "license": "MIT",
      "engines": {
        "node": ">=8"
      },
      "funding": {
        "url": "https://github.com/sponsors/sindresorhus"
      }
    },
    "node_modules/body-parser": {
      "version": "1.20.3",
      "resolved": "https://registry.npmjs.org/body-parser/-/body-
parser-1.20.3.tgz",
      "integrity": "sha512-7rAxByjUMqQ3/bHJy7D6OGXvx/MMc4IqBn/
X0fcM1QUcAItpZrBEYhWGem+tzXH90c+G01ypMcYJBO9Y30203g==",
      "license": "MIT",
      "dependencies": {
        "bytes": "3.1.2",
        "content-type": "~1.0.5",
        "debug": "2.6.9",
        "depd": "2.0.0",
        "destroy": "1.2.0",
        "http-errors": "2.0.0",
        "iconv-lite": "0.4.24",
        "on-finished": "2.4.1",
        "qs": "6.13.0",
        "raw-body": "2.5.2",
        "type-is": "~1.6.18",
        "unpipe": "1.0.0"
      },
      "engines": {
        "node": ">= 0.8",
        "npm": "1.2.8000 || >= 1.4.16"
      }
    },
    "node_modules/bowser": {
      "version": "2.11.0",
      "resolved": "https://registry.npmjs.org/bowser/-/bowser-2.11.0.tgz",
      "integrity": "sha512-AlcaJBi/
pqqJBIQ8U9Mcpc9i8Aqxn88Skv5d+xBX006BY5u8N3mGLHa5Lgppa7L/HfwgwLgZ6NYs+Ag6uUmJRA==",
      "license": "MIT",
```

```
      "optional": true
    },
    "node_modules/brace-expansion": {
      "version": "1.1.11",
      "resolved": "https://registry.npmjs.org/brace-expansion/-/brace-
expansion-1.1.11.tgz",
      "integrity": "sha512-iCuPHDFgrHX7H2vEI/5xpz07zSHB00TpugqhmYtVmMO6518mCuRMoOY
FldEBl0g187ufozdaHgWKcYFb61qGiA==",
      "license": "MIT",
      "dependencies": {
        "balanced-match": "^1.0.0",
        "concat-map": "0.0.1"
      }
    },
    "node_modules/braces": {
      "version": "3.0.3",
      "resolved": "https://registry.npmjs.org/braces/-/braces-3.0.3.tgz",
      "integrity": "sha512-yQbXgO/
OSZVD2IsiLlro+7Hf6Q18EJrKSEsdoMzKePKXct3gvD8oLcOQdIzGupr5Fj+EDe8gO/
lxc1BzfMpxvA==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "fill-range": "^7.1.1"
      },
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/brotli": {
      "version": "1.3.3",
      "resolved": "https://registry.npmjs.org/brotli/-/brotli-1.3.3.tgz",
      "integrity": "sha512-oTKjJdShmDuGW94SyyaoQvAjf30dZaHnjJ8uAF+u2/
vGJkJbJPJAT1gDiOJP5v1Zb6f9KEyW/1HpuaWIXtGHPg==",
      "license": "MIT",
      "dependencies": {
        "base64-js": "^1.1.2"
      }
    },
    "node_modules/bson": {
      "version": "5.5.1",
      "resolved": "https://registry.npmjs.org/bson/-/bson-5.5.1.tgz",
      "integrity": "sha512-ix0EwukN2EpC0SRWIj/7B5+A6uQMQy6KMREI9qQqvgpkV2frH63T0UD
Vd1SYedL6dNCmDBYB3QtXi4ISk9YT+g==",
      "license": "Apache-2.0",
      "engines": {
        "node": ">=14.20.1"
      }
    },
    "node_modules/buffer": {
      "version": "5.7.1",
      "resolved": "https://registry.npmjs.org/buffer/-/buffer-5.7.1.tgz",
      "integrity": "sha512-EHcyIPBQ4BSGlvjB16k5KgAJ27CIsHY/2JBmCRReo48y9rQ3MaUzWX3
KVlBa4U7MyX02HdVj0K7C3WaB3ju7FQ==",
      "funding": [
        {
          "type": "github",
          "url": "https://github.com/sponsors/feross"
        },
```

```
        {
          "type": "patreon",
          "url": "https://www.patreon.com/feross"
        },
        {
          "type": "consulting",
          "url": "https://feross.org/support"
        }
      ],
      "license": "MIT",
      "dependencies": {
        "base64-js": "^1.3.1",
        "ieee754": "^1.1.13"
      }
    },
    "node_modules/buffer-equal-constant-time": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/buffer-equal-constant-time/-/buffer-
equal-constant-time-1.0.1.tgz",
      "integrity": "sha512-zRpUiDwd/
xk6ADqPMATG8vc9VPrkck7T07OIx0gnjmJAnHnTVXNQG3vfvWNuiZIkwu9KrKdA1iJKfsfTVxE6NA==",
      "license": "BSD-3-Clause"
    },
    "node_modules/buffer-from": {
      "version": "1.1.2",
      "resolved": "https://registry.npmjs.org/buffer-from/-/buffer-
from-1.1.2.tgz",
      "integrity": "sha512-E+XQCRwSbaaiChtv6k6Dwgc+bx+Bs6vuKJHHl5kox/
BaKbhiXzqQOwK4cO22yElGp2OCmjwVhT3HmxgyPGnJfQ==",
      "license": "MIT"
    },
    "node_modules/busboy": {
      "version": "1.6.0",
      "resolved": "https://registry.npmjs.org/busboy/-/busboy-1.6.0.tgz",
      "integrity": "sha512-8SFQbg/0hQ9xy3UNTB0YEnsNBbWfhf7RtnzpL7TkBiTBRfrQ9Fxcnz7
VJsleJpyp6rVLvXiuORqjlHi5q+PYuA==",
      "dependencies": {
        "streamsearch": "^1.1.0"
      },
      "engines": {
        "node": ">=10.16.0"
      }
    },
    "node_modules/bytes": {
      "version": "3.1.2",
      "resolved": "https://registry.npmjs.org/bytes/-/bytes-3.1.2.tgz",
      "integrity": "sha512-/Nf7TyzTx6S3yRJObOAV7956r8cr2+Oj8AC5dt8wSP3BQAoeX58NoHy
CU8P8zGkNXStjTSi6fzO6F0pBdcYbEg==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/call-bind": {
      "version": "1.0.8",
      "resolved": "https://registry.npmjs.org/call-bind/-/call-bind-1.0.8.tgz",
      "integrity": "sha512-oKlSFMcMwpUg2ednkhQ454wfWiU/ul3CkJe/
PEHcTKuiX6RpbehUiFMXu13HalGZxfUwCQzZG747YXBn1im9ww==",
      "license": "MIT",
```

```
      "dependencies": {
        "call-bind-apply-helpers": "^1.0.0",
        "es-define-property": "^1.0.0",
        "get-intrinsic": "^1.2.4",
        "set-function-length": "^1.2.2"
      },
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/call-bind-apply-helpers": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/call-bind-apply-helpers/-/call-bind-
apply-helpers-1.0.2.tgz",
      "integrity": "sha512-Sp1ablJ0ivDkSzjcaJdxEunN5/XvksFJ2sMBFfq6x0ryhQV/2b/
KwFe21cMpmHtPOSij8K99/wSfoEuTObmuMQ==",
      "license": "MIT",
      "dependencies": {
        "es-errors": "^1.3.0",
        "function-bind": "^1.1.2"
      },
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/call-bound": {
      "version": "1.0.4",
      "resolved": "https://registry.npmjs.org/call-bound/-/call-bound-1.0.4.tgz",
      "integrity": "sha512-+ys997U96po4Kx/
ABpBCqhA9EuxJaQWDQg7295H4hBphv3IZg0boBKuwYpt4YXp6MZ5AmZQnU/tyMTlRpaSejg==",
      "license": "MIT",
      "dependencies": {
        "call-bind-apply-helpers": "^1.0.2",
        "get-intrinsic": "^1.3.0"
      },
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/chokidar": {
      "version": "3.6.0",
      "resolved": "https://registry.npmjs.org/chokidar/-/chokidar-3.6.0.tgz",
      "integrity": "sha512-7VT13fmjotKpGipCW9JEQAusEPE+Ei8nl6/
g4FBAmIm0GOOLMua9NDDo/DWp0ZAxCr3cPq5ZpBqmPAQgDda2Pw==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "anymatch": "~3.1.2",
        "braces": "~3.0.2",
        "glob-parent": "~5.1.2",
        "is-binary-path": "~2.1.0",
        "is-glob": "~4.0.1",
        "normalize-path": "~3.0.0",
```

```
        "readdirp": "~3.6.0"
      },
      "engines": {
        "node": ">= 8.10.0"
      },
      "funding": {
        "url": "https://paulmillr.com/funding/"
      },
      "optionalDependencies": {
        "fsevents": "~2.3.2"
      }
    },
    "node_modules/chownr": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/chownr/-/chownr-2.0.0.tgz",
      "integrity": "sha512-bIomtDF5KGpdogkLd9VspvFzk9KfpyyGlS8YFVZl7TGPBHL5snIOnxe
shwVgPteQ9b4Eydl+pVbIyE1DcvCWgQ==",
      "license": "ISC",
      "engines": {
        "node": ">=10"
      }
    },
    "node_modules/clone": {
      "version": "2.1.2",
      "resolved": "https://registry.npmjs.org/clone/-/clone-2.1.2.tgz",
      "integrity": "sha512-3Pe/CF1Nn94hyhIYpjtiLhdCoEoz0DqQ+988E9gmeEdQZlojxnOb74w
ctFyuwWQHzqyf9X7C7MG8juUpqBJT8w==",
      "license": "MIT",
      "engines": {
        "node": ">=0.8"
      }
    },
    "node_modules/color-support": {
      "version": "1.1.3",
      "resolved": "https://registry.npmjs.org/color-support/-/color-
support-1.1.3.tgz",
      "integrity": "sha512-qiBjkpbMLO/HL68y+lh4q0/O1MZFj2RX6X/
KmMa3+gJD3z+WwI1ZzDHysvqHGS3mP6mznPckpXmw1nI9cJjyRg==",
      "license": "ISC",
      "bin": {
        "color-support": "bin.js"
      }
    },
    "node_modules/combined-stream": {
      "version": "1.0.8",
      "resolved": "https://registry.npmjs.org/combined-stream/-/combined-
stream-1.0.8.tgz",
      "integrity": "sha512-
FQN4MRfuJeHf7cBbBMJFXhKSDq+2kAArBlmRBvcvFE5BB1HZKXtSFASDhdlz9zOYwxh8lDdnvmMOe/
+5cdoEdg==",
      "license": "MIT",
      "dependencies": {
        "delayed-stream": "~1.0.0"
      },
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/concat-map": {
```

```
      "version": "0.0.1",
      "resolved": "https://registry.npmjs.org/concat-map/-/concat-map-0.0.1.tgz",
      "integrity": "sha512-/Srv4dswyQNBfohGpz9o6Yb3Gz3SrUDqBH5rTuhGR7ahtlbYKnVxw2b
CFMRljaA7EXHaXZ8wsHdodFvbkhKmqg==",
      "license": "MIT"
    },
    "node_modules/concat-stream": {
      "version": "1.6.2",
      "resolved": "https://registry.npmjs.org/concat-stream/-/concat-
stream-1.6.2.tgz",
      "integrity": "sha512-27HBghJxjiZtIk3Ycvn/4kbJk/1uZuJFfuPEns6LaEvpvG1f0hTea8l
ilrouyo9mVc2GWdcEZ8OLoGmSADlrCw==",
      "engines": [
        "node >= 0.8"
      ],
      "license": "MIT",
      "dependencies": {
        "buffer-from": "^1.0.0",
        "inherits": "^2.0.3",
        "readable-stream": "^2.2.2",
        "typedarray": "^0.0.6"
      }
    },
    "node_modules/concat-stream/node_modules/readable-stream": {
      "version": "2.3.8",
      "resolved": "https://registry.npmjs.org/readable-stream/-/readable-
stream-2.3.8.tgz",
      "integrity": "sha512-8p0AUk4XODgIewSi0l8Epjs+EVnWiK7NoDIEGU0HhE7+ZyY8D1IMY7o
du5lRrFXGg71L15KG8QrPmum45RTtdA==",
      "license": "MIT",
      "dependencies": {
        "core-util-is": "~1.0.0",
        "inherits": "~2.0.3",
        "isarray": "~1.0.0",
        "process-nextick-args": "~2.0.0",
        "safe-buffer": "~5.1.1",
        "string_decoder": "~1.1.1",
        "util-deprecate": "~1.0.1"
      }
    },
    "node_modules/concat-stream/node_modules/safe-buffer": {
      "version": "5.1.2",
      "resolved": "https://registry.npmjs.org/safe-buffer/-/safe-
buffer-5.1.2.tgz",
      "integrity": "sha512-Gd2UZBJDkXlY7GbJxfsE8/
nvKkUEU1G38c1siN6QP6a9PT9MmHB8GnpscSmMJSoF8LOIrt8ud/wPtojys4G6+g==",
      "license": "MIT"
    },
    "node_modules/concat-stream/node_modules/string_decoder": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/string_decoder/-/
string_decoder-1.1.1.tgz",
      "integrity": "sha512-n/
ShnvDi6FHbbVfviro+WojiFzv+s8MPMHBczVePfUpDJLwoLT0ht1l4YwBCbi8pJAveEEdnkHyPyTP/
mzRfwg==",
      "license": "MIT",
      "dependencies": {
        "safe-buffer": "~5.1.0"
      }
```

```
    },
    "node_modules/console-control-strings": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/console-control-strings/-/console-
control-strings-1.1.0.tgz",
      "integrity": "sha512-ty/fTekppD2fIwRvnZAVdeOiGd1c7YXEixbgJTNzqcxJWKQnjJ/
V1bNEEE6hygpM3WjwHFUVK6HTjWSzV4a8sQ==",
      "license": "ISC"
    },
    "node_modules/content-disposition": {
      "version": "0.5.4",
      "resolved": "https://registry.npmjs.org/content-disposition/-/content-
disposition-0.5.4.tgz",
      "integrity": "sha512-FveZTNuGw04cxlAiWbzi6zTAL/lhehaWbTtgluJh4/
E95DqMwTmha3KZN1aAWA8cFIhHzMZUvLevkw5Rqk+tSQ==",
      "license": "MIT",
      "dependencies": {
        "safe-buffer": "5.2.1"
      },
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/content-type": {
      "version": "1.0.5",
      "resolved": "https://registry.npmjs.org/content-type/-/content-
type-1.0.5.tgz",
      "integrity": "sha512-nTjqfcBFEipKdXCv4YDQWCfmcLZKm81ldF0pAopTvyrFGVbcR6P/
VAAd5G7N+0tTr8QqiU0tFadD6FK4NtJwOA==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/cookie": {
      "version": "0.7.1",
      "resolved": "https://registry.npmjs.org/cookie/-/cookie-0.7.1.tgz",
      "integrity": "sha512-6DnInpx7SJ2AK3+CTUE/
ZM0vWTUboZCegxhC2xiIydHR9jNuTAASBrfEpHhiGOZw/nX51bHt6YQl8jsGo4y/0w==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/cookie-signature": {
      "version": "1.0.6",
      "resolved": "https://registry.npmjs.org/cookie-signature/-/cookie-
signature-1.0.6.tgz",
      "integrity": "sha512-QADzlaHc8icV8I7vbaJXJwod9HWYp8uCqf1xa4OfNu1T7JVxQIrUgOW
tHdNDtPiywmFbiS12VjotIXLrKM3orQ==",
      "license": "MIT"
    },
    "node_modules/core-util-is": {
      "version": "1.0.3",
      "resolved": "https://registry.npmjs.org/core-util-is/-/core-util-
is-1.0.3.tgz",
      "integrity": "sha512-ZQBvi1DcpJ4GDqanjucZ2Hj3wEO5pZDS89BWbkcrvdxksJorwUDDZam
X9ldFkp9aw2lmBDLgkObEA4DWNJ9FYQ==",
      "license": "MIT"
```

```
    },
    "node_modules/cors": {
      "version": "2.8.5",
      "resolved": "https://registry.npmjs.org/cors/-/cors-2.8.5.tgz",
      "integrity": "sha512-KIHbLJqu73RGr/hnbrO9uBeixNGuvSQjul/jdFvS/
KFSIH1hWVd1ng7zOHx+YrEfInLG7q4n6GHQ9cDtxv/P6g==",
      "license": "MIT",
      "dependencies": {
        "object-assign": "^4",
        "vary": "^1"
      },
      "engines": {
        "node": ">= 0.10"
      }
    },
    "node_modules/cron-parser": {
      "version": "3.5.0",
      "resolved": "https://registry.npmjs.org/cron-parser/-/cron-
parser-3.5.0.tgz",
      "integrity": "sha512-wyVZtbRs6qDfFd8ap457w3XVntdvqcwBGxBoTvJQH9KGVKL/
fB+h2k3C8AqiVxvUQKN1Ps/Ns46CNViOpVDhfQ==",
      "license": "MIT",
      "dependencies": {
        "is-nan": "^1.3.2",
        "luxon": "^1.26.0"
      },
      "engines": {
        "node": ">=0.8"
      }
    },
    "node_modules/crypto-js": {
      "version": "4.2.0",
      "resolved": "https://registry.npmjs.org/crypto-js/-/crypto-js-4.2.0.tgz",
      "integrity": "sha512-KALDyEYgpY+Rlob/
iriUtjV6d5Eq+Y191A5g4UqLAi8CyGP9N1+FdVbkc1SxKc2r4YAYqG8JzO2KGL+AizD70Q==",
      "license": "MIT"
    },
    "node_modules/date.js": {
      "version": "0.3.3",
      "resolved": "https://registry.npmjs.org/date.js/-/date.js-0.3.3.tgz",
      "integrity": "sha512-HgigOS3h3k6HnW011nAb43c5xx5rBXk8P2v/
WIT9Zv4koIaVXiH2BURguI78VVp+5Qc076T7OR378JViCnZtBw==",
      "license": "MIT",
      "dependencies": {
        "debug": "~3.1.0"
      }
    },
    "node_modules/date.js/node_modules/debug": {
      "version": "3.1.0",
      "resolved": "https://registry.npmjs.org/debug/-/debug-3.1.0.tgz",
      "integrity": "sha512-OX8XqP7/1a9cqkxYw2yXss15f26NKWBpDXQd0/uK/
KPqdQhxbPa994hnzjcE2VqQpDslf55723cKPUOGSmMY3g==",
      "license": "MIT",
      "dependencies": {
        "ms": "2.0.0"
      }
    },
    "node_modules/debug": {
      "version": "2.6.9",
```

```
      "resolved": "https://registry.npmjs.org/debug/-/debug-2.6.9.tgz",
      "integrity": "sha512-
bC7ElrdJaJnPbAP+1EotYvqZsb3ecl5wi6Bfi6BJTUcNowp6cvspg0jXznRTKDjm/
E7AdgFBVeAPVMNcKGsHMA==",
      "license": "MIT",
      "dependencies": {
        "ms": "2.0.0"
      }
    },
    "node_modules/define-data-property": {
      "version": "1.1.4",
      "resolved": "https://registry.npmjs.org/define-data-property/-/define-data-
property-1.1.4.tgz",
      "integrity": "sha512-rBMvIzlpA8v6E+SJZoo+
+HAYqsLrkg7MSfIinMPFhmkorw7X+dOXVJQs+QT69zGkzMyfDnIMN2Wid1+NbL3T+A==",
      "license": "MIT",
      "dependencies": {
        "es-define-property": "^1.0.0",
        "es-errors": "^1.3.0",
        "gopd": "^1.0.1"
      },
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/define-properties": {
      "version": "1.2.1",
      "resolved": "https://registry.npmjs.org/define-properties/-/define-
properties-1.2.1.tgz",
      "integrity": "sha512-8QmQKqEASLd5nx0U1B1okLElbUuuttJ/
AnYmRXbbbGDWh6uS208EjD4Xqq/I9wK7u0v6O08XhTWnt5XtEbR6Dg==",
      "license": "MIT",
      "dependencies": {
        "define-data-property": "^1.0.1",
        "has-property-descriptors": "^1.0.0",
        "object-keys": "^1.1.1"
      },
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/delayed-stream": {
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/delayed-stream/-/delayed-
stream-1.0.0.tgz",
      "integrity": "sha512-ZySD7Nf91aLB0RxL4KGrKHBXl7Eds1DAmEdcoVawXnLD7SDhpNgtuII
2aAkg7a7QS41jxPSZ17p4VdGnMHk3MQ==",
      "license": "MIT",
      "engines": {
        "node": ">=0.4.0"
      }
    },
    "node_modules/delegates": {
```

```
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/delegates/-/delegates-1.0.0.tgz",
      "integrity": "sha512-bd2L678uiWATM6m5Z1VzNCErI3jiGzt6HGY8OVICs40JQq/
HALfbyNJmp0UDakEY4pMMaN0Ly5om/B1VI/+xfQ==",
      "license": "MIT"
    },
    "node_modules/depd": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/depd/-/depd-2.0.0.tgz",
      "integrity": "sha512-g7nH6P6dyDioJogAAGprGpCtVImJhpPk/roCzdb3fIh61/s/
nPsfR6onyMwkCAR/OlC3yBC0lESvUoQEAssIrw==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/destroy": {
      "version": "1.2.0",
      "resolved": "https://registry.npmjs.org/destroy/-/destroy-1.2.0.tgz",
      "integrity": "sha512-2sJGJTaXIIaR1w4iJSNoN0hnMY7Gpc/
n8D4qSCJw8QqFWXf7cuAgnEHxBpweaVcPevC2l3KpjYCx3NypQQgaJg==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8",
        "npm": "1.2.8000 || >= 1.4.16"
      }
    },
    "node_modules/detect-libc": {
      "version": "2.0.4",
      "resolved": "https://registry.npmjs.org/detect-libc/-/detect-
libc-2.0.4.tgz",
      "integrity": "sha512-3UDv+G9CsCKO1WKMGw9fwq/
SWJYbI0c5Y7LU1AXYoDdbhE2AHQ6N6Nb34sG8Fj7T5APy8qXDCKuuIHd1BR0tVA==",
      "license": "Apache-2.0",
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/dfa": {
      "version": "1.2.0",
      "resolved": "https://registry.npmjs.org/dfa/-/dfa-1.2.0.tgz",
      "integrity": "sha512-
ED3jP8saaweFTjeGX8HQPjeC1YYyZs98jGNZx6IiBvxW7JG5v492kamAQB3m2wop07CvU/
RQmzcKr6bgcC5D/Q==",
      "license": "MIT"
    },
    "node_modules/dotenv": {
      "version": "16.5.0",
      "resolved": "https://registry.npmjs.org/dotenv/-/dotenv-16.5.0.tgz",
      "integrity": "sha512-m/C+AwOAr9/
W1UOIZUo232ejMNnJAJtYQjUbHoNTBNTJSvqzzDh7vnrei3o3r3m9blf6ZoDkvcw0VmozNRFJxg==",
      "license": "BSD-2-Clause",
      "engines": {
        "node": ">=12"
      },
      "funding": {
        "url": "https://dotenvx.com"
      }
    },
```

```
    "node_modules/dunder-proto": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/dunder-proto/-/dunder-
proto-1.0.1.tgz",
      "integrity": "sha512-KIN/nDJBQRcXw0MLVhZE9iQHmG68qAVIBg9CqmUYjmQIhgij9U5MFvr
qkUL5FbtyyzZuOeOt0zdeRe4UY7ct+A==",
      "license": "MIT",
      "dependencies": {
        "call-bind-apply-helpers": "^1.0.1",
        "es-errors": "^1.3.0",
        "gopd": "^1.2.0"
      },
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/ecdsa-sig-formatter": {
      "version": "1.0.11",
      "resolved": "https://registry.npmjs.org/ecdsa-sig-formatter/-/ecdsa-sig-
formatter-1.0.11.tgz",
      "integrity": "sha512-nagl3RYrbNv6kQkeJIpt6NJZy8twLB/2vtz6yN9Z4vRKHN4/
QZJIEbqohALSgwKdnksuY3k5Addp5lg8sVoVcQ==",
      "license": "Apache-2.0",
      "dependencies": {
        "safe-buffer": "^5.0.1"
      }
    },
    "node_modules/ee-first": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/ee-first/-/ee-first-1.1.1.tgz",
      "integrity": "sha512-WMwm9LhRUo+WUaRN+vRuETqG89IgZphVSNkdFgeb6sS/
E4OrDIN7t48CAewSHXc6C8lefD8KKfr5vY61brQlow==",
      "license": "MIT"
    },
    "node_modules/emoji-regex": {
      "version": "8.0.0",
      "resolved": "https://registry.npmjs.org/emoji-regex/-/emoji-
regex-8.0.0.tgz",
      "integrity": "sha512-MSjYzcWNOA0ewAHpz0MxpYFvwg6yjy1NG3xteoqz644VCo/RPgnr1/
GGt+ic3iJTzQ8Eu3TdM14SawnVUmGE6A==",
      "license": "MIT"
    },
    "node_modules/encodeurl": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/encodeurl/-/encodeurl-2.0.0.tgz",
      "integrity": "sha512-Q0n9HRi4m6JuGIV1eFlmvJB7ZEVxu93IrMyiMsGC0lrMJMWzRgx6WGq
uyfQgZVb31vhGgXnfmPNNXmxnOkRBrg==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/engine.io": {
      "version": "6.6.4",
      "resolved": "https://registry.npmjs.org/engine.io/-/engine.io-6.6.4.tgz",
      "integrity": "sha512-ZCkIjSYNDyGn0R6ewHDtXgns/Zre/NT6Agvq1/
WobF7JXgFff4SeDroKiCO3fNJreU9YG429Sc81o4w5ok/W5g==",
      "license": "MIT",
      "dependencies": {
```

```
        "@types/cors": "^2.8.12",
        "@types/node": ">=10.0.0",
        "accepts": "~1.3.4",
        "base64id": "2.0.0",
        "cookie": "~0.7.2",
        "cors": "~2.8.5",
        "debug": "~4.3.1",
        "engine.io-parser": "~5.2.1",
        "ws": "~8.17.1"
      },
      "engines": {
        "node": ">=10.2.0"
      }
    },
    "node_modules/engine.io-parser": {
      "version": "5.2.3",
      "resolved": "https://registry.npmjs.org/engine.io-parser/-/engine.io-
parser-5.2.3.tgz",
      "integrity": "sha512-
HqD3yTBfnBxIrbnM1DoD6Pcq8NECnh8d4As1Qgh0z5Gg3jRRIqijury0CL3ghu/
edArpUYiYqQiDUQBIs4np3Q==",
      "license": "MIT",
      "engines": {
        "node": ">=10.0.0"
      }
    },
    "node_modules/engine.io/node_modules/cookie": {
      "version": "0.7.2",
      "resolved": "https://registry.npmjs.org/cookie/-/cookie-0.7.2.tgz",
      "integrity": "sha512-yki5XnKuf750l50uGTllt6kKILY4nQ1eNIQatoXEByZ5dWgnKqbnqmT
rBE5B4N7lrMJKQ2ytWMiTO2o0v6Ew/w==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/engine.io/node_modules/debug": {
      "version": "4.3.7",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.3.7.tgz",
      "integrity": "sha512-Er2nc/
H7RrMXZBFCEim6TCmMk02Z8vLC2Rbi1KEBggpo0fS6l0S1nnapwmIi3yW/
+GOJap1Krg4w0Hg80oCqgQ==",
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
    },
    "node_modules/engine.io/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
```

```
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/es-define-property": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/es-define-property/-/es-define-
property-1.0.1.tgz",
      "integrity": "sha512-e3nRfgfUZ4rNGL232gUgX06QNyyez04KdjFrF+LTRoOXmrOgFKDg4BC
dsjW8EnT69eqdYGmRpJwiPVYNrCaW3g==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/es-errors": {
      "version": "1.3.0",
      "resolved": "https://registry.npmjs.org/es-errors/-/es-errors-1.3.0.tgz",
      "integrity": "sha512-Zf5H2Kxt2xjTvbJvP2ZWLEICxA6j+hAmMzIlypy4xcBg1vKVnx89Wy0
GbS+kf5cwCVFFzdCFh2XSCFNULS6csw==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/es-object-atoms": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/es-object-atoms/-/es-object-
atoms-1.1.1.tgz",
      "integrity": "sha512-FGgH2h8zKNim9ljj7dankFPcICIK9Cp5bm+c2gQSYePhpaG5+esrLOD
ihIorn+Pe6FGJzWhXQotPv73jTaldXA==",
      "license": "MIT",
      "dependencies": {
        "es-errors": "^1.3.0"
      },
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/es-set-tostringtag": {
      "version": "2.1.0",
      "resolved": "https://registry.npmjs.org/es-set-tostringtag/-/es-set-
tostringtag-2.1.0.tgz",
      "integrity": "sha512-j6vWzfrGVfyXxge+O0x5sh6cvxAog0a/4Rdd2K36zCMV5eJ+/
+tOAngRO8cODMNWbVRdVlmGZQL2YS3yR8bIUA==",
      "license": "MIT",
      "dependencies": {
        "es-errors": "^1.3.0",
        "get-intrinsic": "^1.2.6",
        "has-tostringtag": "^1.0.2",
        "hasown": "^2.0.2"
      },
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/escape-html": {
      "version": "1.0.3",
      "resolved": "https://registry.npmjs.org/escape-html/-/escape-
html-1.0.3.tgz",
```

```
      "integrity": "sha512-NiSupZ4OeuGwr68lGIeym/ksIZMJodUGOSCZ/
FSnTxcrekbvqrgdUxlJOMpijaKZVjAJrWrGs/6Jy8OMuyj9ow==",
      "license": "MIT"
    },
    "node_modules/etag": {
      "version": "1.8.1",
      "resolved": "https://registry.npmjs.org/etag/-/etag-1.8.1.tgz",
      "integrity": "sha512-aIL5Fx7mawVa300al2BnEE4iNvo1qETxLrPI/
o05L7z6go7fCw1J6EQmbK4FmJ2AS7kgVF/KEZWufBfdClMcPg==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/express": {
      "version": "4.21.2",
      "resolved": "https://registry.npmjs.org/express/-/express-4.21.2.tgz",
      "integrity":
"sha512-28HqgMZAmih1Czt9ny7qr6ek2qddF4FclbMzwhCREB6OFfH+rXAnuNCwo1/
wFvrtbgsQDb4kSbX9de9lFbrXnA==",
      "license": "MIT",
      "dependencies": {
        "accepts": "~1.3.8",
        "array-flatten": "1.1.1",
        "body-parser": "1.20.3",
        "content-disposition": "0.5.4",
        "content-type": "~1.0.4",
        "cookie": "0.7.1",
        "cookie-signature": "1.0.6",
        "debug": "2.6.9",
        "depd": "2.0.0",
        "encodeurl": "~2.0.0",
        "escape-html": "~1.0.3",
        "etag": "~1.8.1",
        "finalhandler": "1.3.1",
        "fresh": "0.5.2",
        "http-errors": "2.0.0",
        "merge-descriptors": "1.0.3",
        "methods": "~1.1.2",
        "on-finished": "2.4.1",
        "parseurl": "~1.3.3",
        "path-to-regexp": "0.1.12",
        "proxy-addr": "~2.0.7",
        "qs": "6.13.0",
        "range-parser": "~1.2.1",
        "safe-buffer": "5.2.1",
        "send": "0.19.0",
        "serve-static": "1.16.2",
        "setprototypeof": "1.2.0",
        "statuses": "2.0.1",
        "type-is": "~1.6.18",
        "utils-merge": "1.0.1",
        "vary": "~1.1.2"
      },
      "engines": {
        "node": ">= 0.10.0"
      },
      "funding": {
        "type": "opencollective",
```

```
        "url": "https://opencollective.com/express"
      }
    },
    "node_modules/express-rate-limit": {
      "version": "6.11.2",
      "resolved": "https://registry.npmjs.org/express-rate-limit/-/express-rate-
limit-6.11.2.tgz",
      "integrity": "sha512-
a7uwwfNTh1U60ssiIkuLFWHt4hAC5yxlLGU2VP0X4YNlyEDZAqF4tK3GD3NSitVBrCQmQ0+
+0uOyFOgC2y4DDw==",
      "license": "MIT",
      "engines": {
        "node": ">= 14"
      },
      "peerDependencies": {
        "express": "^4 || ^5"
      }
    },
    "node_modules/fast-deep-equal": {
      "version": "3.1.3",
      "resolved": "https://registry.npmjs.org/fast-deep-equal/-/fast-deep-
equal-3.1.3.tgz",
      "integrity": "sha512-f3qQ9oQy9j2AhBe/H9VC91wLmKBCCU/
gDOnKNAYG5hswO7BLKj09Hc5HYNz9cGI++xlpDCIgDaitVs03ATR84Q==",
      "license": "MIT"
    },
    "node_modules/fast-xml-parser": {
      "version": "4.4.1",
      "resolved": "https://registry.npmjs.org/fast-xml-parser/-/fast-xml-
parser-4.4.1.tgz",
      "integrity": "sha512-xkjOecfnKGkSsOwtZ5Pz7Us/
T6mrbPQrq0nh+aCO5V9nk5NLWmasAHumTKjiPJPWANe+kAZ84Jc8ooJkzZ88Sw==",
      "funding": [
        {
          "type": "github",
          "url": "https://github.com/sponsors/NaturalIntelligence"
        },
        {
          "type": "paypal",
          "url": "https://paypal.me/naturalintelligence"
        }
      ],
      "license": "MIT",
      "optional": true,
      "dependencies": {
        "strnum": "^1.0.5"
      },
      "bin": {
        "fxparser": "src/cli/cli.js"
      }
    },
    "node_modules/fill-range": {
      "version": "7.1.1",
      "resolved": "https://registry.npmjs.org/fill-range/-/fill-range-7.1.1.tgz",
      "integrity": "sha512-YsGpe3WHLK8ZYi4tWDg2Jy3ebRz2rXowDxnld4bkQB00cc/1Zw9AWnC
0i9ztDJitivtQvaI9KaLyKrc+hBW0yg==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
```

```
        "to-regex-range": "^5.0.1"
      },
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/finalhandler": {
      "version": "1.3.1",
      "resolved": "https://registry.npmjs.org/finalhandler/-/
finalhandler-1.3.1.tgz",
      "integrity": "sha512-6BN9trH7bp3qvnrRyzsBz+g3lZxTNZTbVO2EV1CS0WIcDbawYVdYvGf
lME/9QP0h0pYlCDBCTjYa9nZzMDpyxQ==",
      "license": "MIT",
      "dependencies": {
        "debug": "2.6.9",
        "encodeurl": "~2.0.0",
        "escape-html": "~1.0.3",
        "on-finished": "2.4.1",
        "parseurl": "~1.3.3",
        "statuses": "2.0.1",
        "unpipe": "~1.0.0"
      },
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/follow-redirects": {
      "version": "1.15.9",
      "resolved": "https://registry.npmjs.org/follow-redirects/-/follow-
redirects-1.15.9.tgz",
      "integrity": "sha512-
gew4GsXizNgdoRyqmyfMHyAmXsZDk6mHkSxZFCzW9gwlbtOW44CDtYavM+y+72qD/
Vq2l550kMF52DT8fOLJqQ==",
      "funding": [
        {
          "type": "individual",
          "url": "https://github.com/sponsors/RubenVerborgh"
        }
      ],
      "license": "MIT",
      "engines": {
        "node": ">=4.0"
      },
      "peerDependenciesMeta": {
        "debug": {
          "optional": true
        }
      }
    },
    "node_modules/fontkit": {
      "version": "2.0.4",
      "resolved": "https://registry.npmjs.org/fontkit/-/fontkit-2.0.4.tgz",
      "integrity": "sha512-syetQadaUEDNdxdugga9CpEYVaQIxOwk7GlwZWWZ19//
qW4zE5bknOKeMBDYAASwnpaSHKJITRLMF9m1fp3s6g==",
      "license": "MIT",
      "dependencies": {
        "@swc/helpers": "^0.5.12",
        "brotli": "^1.3.2",
        "clone": "^2.1.2",
```

```
        "dfa": "^1.2.0",
        "fast-deep-equal": "^3.1.3",
        "restructure": "^3.0.0",
        "tiny-inflate": "^1.0.3",
        "unicode-properties": "^1.4.0",
        "unicode-trie": "^2.0.0"
      }
    },
    "node_modules/form-data": {
      "version": "4.0.2",
      "resolved": "https://registry.npmjs.org/form-data/-/form-data-4.0.2.tgz",
      "integrity": "sha512-hGfm/slu0ZabnNt4oaRZ6uREyfCj6P4fT/
n6A1rGV+Z0VdGXjfOhVUpkn6qVQONHGIFwmveGXyDs75+nr6FM8w==",
      "license": "MIT",
      "dependencies": {
        "asynckit": "^0.4.0",
        "combined-stream": "^1.0.8",
        "es-set-tostringtag": "^2.1.0",
        "mime-types": "^2.1.12"
      },
      "engines": {
        "node": ">= 6"
      }
    },
    "node_modules/forwarded": {
      "version": "0.2.0",
      "resolved": "https://registry.npmjs.org/forwarded/-/forwarded-0.2.0.tgz",
      "integrity": "sha512-
buRG0fpBtRHSTCOASe6hD258tEubFoRLb4ZNA6NxMVHNw2gOcwHo9wyablzMzOA5z9xA9L1KNjk/
Nt6MT9aYow==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/fresh": {
      "version": "0.5.2",
      "resolved": "https://registry.npmjs.org/fresh/-/fresh-0.5.2.tgz",
      "integrity": "sha512-zJ2mQYM18rEFOudeV4GShTGIQ7RbzA7ozbU9I/
XBpm7kqgMywgmylMwXHxZJmkVoYkna9d2pVXVXPdYTP9ej8Q==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/fs-minipass": {
      "version": "2.1.0",
      "resolved": "https://registry.npmjs.org/fs-minipass/-/fs-
minipass-2.1.0.tgz",
      "integrity": "sha512-V/JgOLFCS+R6Vcq0slCuaeWEdNC3ouDlJMNIsacH2VtALiu9mV4LPrH
c5cDl8k5aw6J8jwgWWpiTo5RYhmIzvg==",
      "license": "ISC",
      "dependencies": {
        "minipass": "^3.0.0"
      },
      "engines": {
        "node": ">= 8"
      }
    },
```

```
    "node_modules/fs-minipass/node_modules/minipass": {
      "version": "3.3.6",
      "resolved": "https://registry.npmjs.org/minipass/-/minipass-3.3.6.tgz",
      "integrity": "sha512-DxiNidxSEK+tHG6zOIklvNOwm3hvCrbUrdtzY74U6HKTJxvIDfOUL5W
5P2Ghd3DTkhhKPYGqeNUIh5qcM4YBfw==",
      "license": "ISC",
      "dependencies": {
        "yallist": "^4.0.0"
      },
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/fs.realpath": {
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/fs.realpath/-/
fs.realpath-1.0.0.tgz",
      "integrity": "sha512-OO0pH2lK6a0hZnAdau5ItzHPI6pUlvI7jMVnxUQRtw4owF2wk8lOSab
tGDCTP4Ggrg2MbGnWO9X8K1t4+fGMDw==",
      "license": "ISC"
    },
    "node_modules/fsevents": {
      "version": "2.3.3",
      "resolved": "https://registry.npmjs.org/fsevents/-/fsevents-2.3.3.tgz",
      "integrity": "sha512-5xoDfX+fL7faATnagmWPpbFtwh/
R77WmMMqqHGS65C3vvB0YHrgF+B1YmZ3441tMj5n63k0212XNoJwzlhffQw==",
      "dev": true,
      "hasInstallScript": true,
      "license": "MIT",
      "optional": true,
      "os": [
        "darwin"
      ],
      "engines": {
        "node": "^8.16.0 || ^10.6.0 || >=11.0.0"
      }
    },
    "node_modules/function-bind": {
      "version": "1.1.2",
      "resolved": "https://registry.npmjs.org/function-bind/-/function-
bind-1.1.2.tgz",
      "integrity": "sha512-7XHNxH7qX9xG5mIwxkhumTox/
MIRNcOgDrxWsMt2pAr23WHp6MrRlN7FBSFpCpr+oVO0F744iUgR82nJMfG2SA==",
      "license": "MIT",
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/gauge": {
      "version": "3.0.2",
      "resolved": "https://registry.npmjs.org/gauge/-/gauge-3.0.2.tgz",
      "integrity": "sha512-+5J6MS/5XksCuXq+
+uFRsnUd7Ovu1XenbeuIuNRJxYWjgQbPuFhT14lAvsWfqfAmnwluf1OwMjz39HjfLPci0Q==",
      "deprecated": "This package is no longer supported.",
      "license": "ISC",
      "dependencies": {
        "aproba": "^1.0.3 || ^2.0.0",
        "color-support": "^1.1.2",
        "console-control-strings": "^1.0.0",
```

```
        "has-unicode": "^2.0.1",
        "object-assign": "^4.1.1",
        "signal-exit": "^3.0.0",
        "string-width": "^4.2.3",
        "strip-ansi": "^6.0.1",
        "wide-align": "^1.1.2"
      },
      "engines": {
        "node": ">=10"
      }
    },
    "node_modules/get-intrinsic": {
      "version": "1.3.0",
      "resolved": "https://registry.npmjs.org/get-intrinsic/-/get-
intrinsic-1.3.0.tgz",
      "integrity": "sha512-9fSjSaos/
fRIVIp+xSJlE6lfwhES7LNtKaCBIamHsjr2na1BiABJPo0mOjjz8GJDURarmCPGqaiVg5mfjb98CQ==",
      "license": "MIT",
      "dependencies": {
        "call-bind-apply-helpers": "^1.0.2",
        "es-define-property": "^1.0.1",
        "es-errors": "^1.3.0",
        "es-object-atoms": "^1.1.1",
        "function-bind": "^1.1.2",
        "get-proto": "^1.0.1",
        "gopd": "^1.2.0",
        "has-symbols": "^1.1.0",
        "hasown": "^2.0.2",
        "math-intrinsics": "^1.1.0"
      },
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/get-proto": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/get-proto/-/get-proto-1.0.1.tgz",
      "integrity": "sha512-sTSfBjoXBp89JvIKIefqw7U2CCebsc74kiY6awiGogKtoSGbgjYE/G/
+l9sF3MWFPNc9IcoOC4ODfKHfxFmp0g==",
      "license": "MIT",
      "dependencies": {
        "dunder-proto": "^1.0.1",
        "es-object-atoms": "^1.0.0"
      },
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/glob": {
      "version": "7.2.3",
      "resolved": "https://registry.npmjs.org/glob/-/glob-7.2.3.tgz",
      "integrity": "sha512-nFR0zLpU2YCaRxwoCJvL6UvCH2JFyFVIvwTLsIf21AuHlMskA1hhTdk
+LlYJtOlYt9v6dvszD2BGRqBL+iQK9Q==",
      "deprecated": "Glob versions prior to v9 are no longer supported",
      "license": "ISC",
      "dependencies": {
```

```
        "fs.realpath": "^1.0.0",
        "inflight": "^1.0.4",
        "inherits": "2",
        "minimatch": "^3.1.1",
        "once": "^1.3.0",
        "path-is-absolute": "^1.0.0"
      },
      "engines": {
        "node": "*"
      },
      "funding": {
        "url": "https://github.com/sponsors/isaacs"
      }
    },
    "node_modules/glob-parent": {
      "version": "5.1.2",
      "resolved": "https://registry.npmjs.org/glob-parent/-/glob-
parent-5.1.2.tgz",
      "integrity": "sha512-
AOIgSQCepiJYwP3ARnGx+5VnTu2HBYdzbGP45eLw1vr3zB3vZLeyed1sC9hnbcOc9/
SrMyM5RPQrkGz4aS9Zow==",
      "dev": true,
      "license": "ISC",
      "dependencies": {
        "is-glob": "^4.0.1"
      },
      "engines": {
        "node": ">= 6"
      }
    },
    "node_modules/gopd": {
      "version": "1.2.0",
      "resolved": "https://registry.npmjs.org/gopd/-/gopd-1.2.0.tgz",
      "integrity": "sha512-ZUKRh6/kUFoAiTAtTYPZJ3hw9wNxx+BIBOijnlG9PnrJsCcSjs1wyyD
6vJpaYtgnzDrKYRSqf3OO6Rfa93xsRg==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/has-flag": {
      "version": "3.0.0",
      "resolved": "https://registry.npmjs.org/has-flag/-/has-flag-3.0.0.tgz",
      "integrity": "sha512-
sKJf1+ceQBr4SMkvQnBDNDtf4TXpVhVGateu0t918bl30FnbE2m4vNLX+VWe/
dpjlb+HugGYzW7uQXH98HPEYw==",
      "dev": true,
      "license": "MIT",
      "engines": {
        "node": ">=4"
      }
    },
    "node_modules/has-property-descriptors": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/has-property-descriptors/-/has-
property-descriptors-1.0.2.tgz",
```

```
      "integrity": "sha512-55JNKuIW+vq4Ke1BjOTjM2YctQIvCT7GFzHwmfZPGo5wnrgkid0YQtn
AleFSqumZm4az3n2BS+erby5ipJdgrg==",
      "license": "MIT",
      "dependencies": {
        "es-define-property": "^1.0.0"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/has-symbols": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/has-symbols/-/has-
symbols-1.1.0.tgz",
      "integrity": "sha512-1cDNdwJ2Jaohmb3sg4OmKaMBwuC48sYni5HUw2DvsC8LjGTLK9h+eb1
X6RyuOHe4hT0ULCW68iomhjUoKUqlPQ==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/has-tostringtag": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/has-tostringtag/-/has-
tostringtag-1.0.2.tgz",
      "integrity": "sha512-
NqADB8VjPFLM2V0VvHUewwwsw0ZWBaIdgo+ieHtK3hasLz4qeCRjYcqfB6AQrBggRKppKF8L52/
VqdVsO47Dlw==",
      "license": "MIT",
      "dependencies": {
        "has-symbols": "^1.0.3"
      },
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/has-unicode": {
      "version": "2.0.1",
      "resolved": "https://registry.npmjs.org/has-unicode/-/has-
unicode-2.0.1.tgz",
      "integrity": "sha512-8Rf9Y83NBReMnx0gFzA8JImQACstCYWUplepDa9xprwwtmgEZUF0h/
i5xSA625zB/I37EtrswSST6OXxwaaIJQ==",
      "license": "ISC"
    },
    "node_modules/hasown": {
      "version": "2.0.2",
      "resolved": "https://registry.npmjs.org/hasown/-/hasown-2.0.2.tgz",
      "integrity": "sha512-0hJU9SCPvmMzIBdZFqNPXWa6dqh7WdH0cII9y+CyS8rG3nL48Bclra9
HmKhVVUHyPWNH5Y7xDwAB7bfgSjkUMQ==",
      "license": "MIT",
      "dependencies": {
        "function-bind": "^1.1.2"
      },
```

```
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/http-errors": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/http-errors/-/http-
errors-2.0.0.tgz",
      "integrity": "sha512-FtwrG/euBzaEjYeRqOgly7G0qviiXoJWnvEH2Z1plBdXgbyjv34pHTS
b9zoeHMyDy33+DWy5Wt9Wo+TURtOYSQ==",
      "license": "MIT",
      "dependencies": {
        "depd": "2.0.0",
        "inherits": "2.0.4",
        "setprototypeof": "1.2.0",
        "statuses": "2.0.1",
        "toidentifier": "1.0.1"
      },
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/https-proxy-agent": {
      "version": "5.0.1",
      "resolved": "https://registry.npmjs.org/https-proxy-agent/-/https-proxy-
agent-5.0.1.tgz",
      "integrity": "sha512-dFcAjpTQFgoLMzC2VwU+C/
CbS7uRL0lWmxDITmqm7C+7F0Odmj6s9l6alZc6AELXhrnggM2CeWSXHGOdX2YtwA==",
      "license": "MIT",
      "dependencies": {
        "agent-base": "6",
        "debug": "4"
      },
      "engines": {
        "node": ">= 6"
      }
    },
    "node_modules/https-proxy-agent/node_modules/debug": {
      "version": "4.4.0",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.4.0.tgz",
      "integrity": "sha512-6WTZ/IxCY/T6BALoZHaE4ctp9xm+Z5kY/
pzYaCHRFeyVhojxlrm+46y68HA6hr0TcwEssoxNiDEUJQjfPZ/RYA==",
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
    },
    "node_modules/https-proxy-agent/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
```

```
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/human-interval": {
      "version": "2.0.1",
      "resolved": "https://registry.npmjs.org/human-interval/-/human-
interval-2.0.1.tgz",
      "integrity": "sha512-r4Aotzf+OtKIGQCB3odUowy4GfUDTy3aTWTfLd7ZF2gBCy3XW3v/
dJLRefZnOFFnjqs5B1TypvS8WarpBkYUNQ==",
      "license": "MIT",
      "dependencies": {
        "numbered": "^1.1.0"
      }
    },
    "node_modules/iconv-lite": {
      "version": "0.4.24",
      "resolved": "https://registry.npmjs.org/iconv-lite/-/iconv-lite-0.4.24.tgz",
      "integrity": "sha512-v3MXnZAcvnywkTUEZomIActle7RXXeedOR31wwl7VlyoXO4Qi9arvSe
nNQWne1TcRwhCL1HwLI21bEqdpj8/rA==",
      "license": "MIT",
      "dependencies": {
        "safer-buffer": ">= 2.1.2 < 3"
      },
      "engines": {
        "node": ">=0.10.0"
      }
    },
    "node_modules/ieee754": {
      "version": "1.2.1",
      "resolved": "https://registry.npmjs.org/ieee754/-/ieee754-1.2.1.tgz",
      "integrity": "sha512-dcyqhDvX1C46lXZcVqCpK+FtMRQVdIMN6/
Df5js2zouUsqG7I6sFxitIC+7KYK29KdXOLHdu9zL4sFnoVQnqaA==",
      "funding": [
        {
          "type": "github",
          "url": "https://github.com/sponsors/feross"
        },
        {
          "type": "patreon",
          "url": "https://www.patreon.com/feross"
        },
        {
          "type": "consulting",
          "url": "https://feross.org/support"
        }
      ],
      "license": "BSD-3-Clause"
    },
    "node_modules/ignore-by-default": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/ignore-by-default/-/ignore-by-
default-1.0.1.tgz",
      "integrity": "sha512-
Ius2VYcGNk7T90CppJqcIkS5ooHUZyIQK+ClZfMfMNFEF9VSE73Fq+906u/
CWu92x4gzZMWOwfFYckPObzdEbA==",
      "dev": true,
      "license": "ISC"
    },
    "node_modules/inflight": {
```

```
      "version": "1.0.6",
      "resolved": "https://registry.npmjs.org/inflight/-/inflight-1.0.6.tgz",
      "integrity": "sha512-k92I/
b08q4wvFscXCLvqfsHCrjrF7yiXsQuIVvVE7N82W3+aqpzuUdBbfhWcy/
FZR3/4IgflMgKLOsvPDrGCJA==",
      "deprecated": "This module is not supported, and leaks memory. Do not use
it. Check out lru-cache if you want a good and tested way to coalesce async
requests by a key value, which is much more comprehensive and powerful.",
      "license": "ISC",
      "dependencies": {
        "once": "^1.3.0",
        "wrappy": "1"
      }
    },
    "node_modules/inherits": {
      "version": "2.0.4",
      "resolved": "https://registry.npmjs.org/inherits/-/inherits-2.0.4.tgz",
      "integrity": "sha512-k/vGaX4/
Yla3WzyMCvTQOXYeIHvqOKtnqBduzTHpzpQZzAskKMhZ2K+EnBiSM9zGSoIFeMpXKxa4dYeZIQqewQ==",
      "license": "ISC"
    },
    "node_modules/ip-address": {
      "version": "9.0.5",
      "resolved": "https://registry.npmjs.org/ip-address/-/ip-address-9.0.5.tgz",
      "integrity": "sha512-zHtQzGojZXTwZTHQqra+ETKd4Sn3vgi7uBmlPoXVWZqYvuKmtI0l/
VZTjqGmJY9x88GGOaZ9+G9ES8hC4T4X8g==",
      "license": "MIT",
      "dependencies": {
        "jsbn": "1.1.0",
        "sprintf-js": "^1.1.3"
      },
      "engines": {
        "node": ">= 12"
      }
    },
    "node_modules/ipaddr.js": {
      "version": "1.9.1",
      "resolved": "https://registry.npmjs.org/ipaddr.js/-/ipaddr.js-1.9.1.tgz",
      "integrity": "sha512-0KI/607xoxSToH7GjN1FfSbLoU0+btTicjsQSWQlh/
hZykN8KpmMf7uYwPW3R+akZ6R/w18ZlXSHBYXiYUPO3g==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.10"
      }
    },
    "node_modules/is-binary-path": {
      "version": "2.1.0",
      "resolved": "https://registry.npmjs.org/is-binary-path/-/is-binary-
path-2.1.0.tgz",
      "integrity": "sha512-ZMERYes6pDydyuGidse7OsHxtbI7WVeUEozgR/g7rd0xUimYNlvZRE/
K2MgZTjWy725IfelLeVcEM97mmtRGXw==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "binary-extensions": "^2.0.0"
      },
      "engines": {
        "node": ">=8"
      }
```

```
    },
    "node_modules/is-extglob": {
      "version": "2.1.1",
      "resolved": "https://registry.npmjs.org/is-extglob/-/is-extglob-2.1.1.tgz",
      "integrity": "sha512-SbKbANkN603Vi4jEZv49LeVJMn4yGwsbzZworEoyEiutsN3nJYdbO36
zfhGJ6QEDpOZIFkDtnq5JRxmvl3jsoQ==",
      "dev": true,
      "license": "MIT",
      "engines": {
        "node": ">=0.10.0"
      }
    },
    "node_modules/is-fullwidth-code-point": {
      "version": "3.0.0",
      "resolved": "https://registry.npmjs.org/is-fullwidth-code-point/-/is-
fullwidth-code-point-3.0.0.tgz",
      "integrity": "sha512-zymm5+u+sCsSWyD9qNaejV3DFvhCKclKdizYaJUuHA83RLjb7nSuGnd
dCHGv0hk+KY7BMAlsWeK4Ueg6EV6XQg==",
      "license": "MIT",
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/is-glob": {
      "version": "4.0.3",
      "resolved": "https://registry.npmjs.org/is-glob/-/is-glob-4.0.3.tgz",
      "integrity": "sha512-xelSayHH36ZgE7ZWhli7pW34hNbNl8Ojv5KVmkJD4hBdD3th8Tfk9vY
asLM+mXWOZhFkgZfxhLSnrwRr4elSSg==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "is-extglob": "^2.1.1"
      },
      "engines": {
        "node": ">=0.10.0"
      }
    },
    "node_modules/is-nan": {
      "version": "1.3.2",
      "resolved": "https://registry.npmjs.org/is-nan/-/is-nan-1.3.2.tgz",
      "integrity": "sha512-E+zBKpQ2t6MEo1VsonYmluk9NxGrbzpeeLC2xIViuO2EjU2xsXsBPwT
r3Ykv9l08UYEVEdWeRZNouaZqF6RN0w==",
      "license": "MIT",
      "dependencies": {
        "call-bind": "^1.0.0",
        "define-properties": "^1.1.3"
      },
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/is-number": {
      "version": "7.0.0",
      "resolved": "https://registry.npmjs.org/is-number/-/is-number-7.0.0.tgz",
      "integrity": "sha512-41Cifkg6e8TylSpdtTpeLVMqvSBEVzTttHvERD741+pnZ8ANv0004MR
L43QKPDlK9cGvNp6NZWZUBlbGXYxxng==",
```

```
      "dev": true,
      "license": "MIT",
      "engines": {
        "node": ">=0.12.0"
      }
    },
    "node_modules/isarray": {
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/isarray/-/isarray-1.0.0.tgz",
      "integrity": "sha512-
VLghIWNM6ELQzo7zwmcg0NmTVyWKYjvIeM83yjp0wRDTmUnrM678fQbcKBo6n2CJEF0szoG//
ytg+TKla89ALQ==",
      "license": "MIT"
    },
    "node_modules/jpeg-exif": {
      "version": "1.1.4",
      "resolved": "https://registry.npmjs.org/jpeg-exif/-/jpeg-exif-1.1.4.tgz",
      "integrity": "sha512-a+bKEcCjtuW5WTdgeXFzswSrdqi0jk4XlEtZlx5A94wCoBpFjfFTbo/
Tra5SpNCl/YFZPvcV1dJc+TAYeg6ROQ==",
      "license": "MIT"
    },
    "node_modules/jsbn": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/jsbn/-/jsbn-1.1.0.tgz",
      "integrity": "sha512-4bYVV3aAMtDTTu4+xsDYa6sy9GyJ69/
amsu9sYF2zqjiEoZA5xJi3BrfX3uY+/IekIu7MwdObdbDWpoZdBv3/A==",
      "license": "MIT"
    },
    "node_modules/jsonwebtoken": {
      "version": "9.0.2",
      "resolved": "https://registry.npmjs.org/jsonwebtoken/-/
jsonwebtoken-9.0.2.tgz",
      "integrity": "sha512-PRp66vJ865SSqOlgqS8hujT5U4AOgMfhrwYIuIhfKaoSCZcirrmASQr
8CX7cUg+RMih+hgznrjp99o+W4pJLHQ==",
      "license": "MIT",
      "dependencies": {
        "jws": "^3.2.2",
        "lodash.includes": "^4.3.0",
        "lodash.isboolean": "^3.0.3",
        "lodash.isinteger": "^4.0.4",
        "lodash.isnumber": "^3.0.3",
        "lodash.isplainobject": "^4.0.6",
        "lodash.isstring": "^4.0.1",
        "lodash.once": "^4.0.0",
        "ms": "^2.1.1",
        "semver": "^7.5.4"
      },
      "engines": {
        "node": ">=12",
        "npm": ">=6"
      }
    },
    "node_modules/jsonwebtoken/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
```

```
    "node_modules/jwa": {
      "version": "1.4.1",
      "resolved": "https://registry.npmjs.org/jwa/-/jwa-1.4.1.tgz",
      "integrity": "sha512-qiLX/xhEEFKUAJ6FiBMbes3w9ATzyk5W7Hvzpa/
SLYdxNtng+gcurvrI7TbACjIXlsJyr05/S1oUhZrc63evQA==",
      "license": "MIT",
      "dependencies": {
        "buffer-equal-constant-time": "1.0.1",
        "ecdsa-sig-formatter": "1.0.11",
        "safe-buffer": "^5.0.1"
      }
    },
    "node_modules/jws": {
      "version": "3.2.2",
      "resolved": "https://registry.npmjs.org/jws/-/jws-3.2.2.tgz",
      "integrity": "sha512-YHlZCB6lMTllWDtSPHz/
ZXTsi8S00usEV6v1tjq8tOUZzw7DpSDWVXjXDre6ed1w/pd495ODpHZYSdkRTsa0HA==",
      "license": "MIT",
      "dependencies": {
        "jwa": "^1.4.1",
        "safe-buffer": "^5.0.1"
      }
    },
    "node_modules/kareem": {
      "version": "2.5.1",
      "resolved": "https://registry.npmjs.org/kareem/-/kareem-2.5.1.tgz",
      "integrity": "sha512-7jFxRVm+jD+rkq3kY0iZDJfsO2/
t4BBPeEb2qKn2lR/9KhuksYk5hxzfRYWMPV8P/x2d0kHD306YyWLzjjH+uA==",
      "license": "Apache-2.0",
      "engines": {
        "node": ">=12.0.0"
      }
    },
    "node_modules/linebreak": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/linebreak/-/linebreak-1.1.0.tgz",
      "integrity": "sha512-MHp03UImeVhB7XZtjd0E4n6+3xr5Dq/9xI/5FptGk5FrbDR3zagPa2D
S6U8ks/3HjbKWG9Q1M2ufOzxV2qLYSQ==",
      "license": "MIT",
      "dependencies": {
        "base64-js": "0.0.8",
        "unicode-trie": "^2.0.0"
      }
    },
    "node_modules/linebreak/node_modules/base64-js": {
      "version": "0.0.8",
      "resolved": "https://registry.npmjs.org/base64-js/-/base64-js-0.0.8.tgz",
      "integrity": "sha512-3XSA2cR/
h/73EzlXXdU6YNycmYI7+kicTxks4eJg2g39biHR84slg2+des+p7iHYhbRg/udIS4TD53WabcOUkw==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/lodash.includes": {
      "version": "4.3.0",
      "resolved": "https://registry.npmjs.org/lodash.includes/-/
lodash.includes-4.3.0.tgz",
      "integrity": "sha512-W3Bx6mdkRTGtlJISOvVD/
lbqjTlPPUDTMnlXZFnVwi9NKJ6tiAk6LVdlhZMm17VZisqhKcgzpO5Wz91PCt5b0w==",
```

```
      "license": "MIT"
    },
    "node_modules/lodash.isboolean": {
      "version": "3.0.3",
      "resolved": "https://registry.npmjs.org/lodash.isboolean/-/
lodash.isboolean-3.0.3.tgz",
      "integrity": "sha512-
Bz5mupy2SVbPHURB98VAcw+aHh4vRV5IPNhILUCsOzRmsTmSQ17jIuqopAentWoehktxGd9e/
hbIXq980/1QJg==",
      "license": "MIT"
    },
    "node_modules/lodash.isinteger": {
      "version": "4.0.4",
      "resolved": "https://registry.npmjs.org/lodash.isinteger/-/
lodash.isinteger-4.0.4.tgz",
      "integrity": "sha512-DBwtEWN2caHQ9/
imiNeEA5ys1JoRtRfY3d7V9wkqtbycnAmTvRRmbHKDV4a0EYc678/dia0jrte4tjYwVBaZUA==",
      "license": "MIT"
    },
    "node_modules/lodash.isnumber": {
      "version": "3.0.3",
      "resolved": "https://registry.npmjs.org/lodash.isnumber/-/
lodash.isnumber-3.0.3.tgz",
      "integrity": "sha512-QYqzpfwO3/CWf3XP+Z+tkQsfaLL/
EnUlXWVkIk5FUPc4sBdTehEqZONuyRt2P67PXAk+NXmTBcc97zw9t1FQrw==",
      "license": "MIT"
    },
    "node_modules/lodash.isplainobject": {
      "version": "4.0.6",
      "resolved": "https://registry.npmjs.org/lodash.isplainobject/-/
lodash.isplainobject-4.0.6.tgz",
      "integrity": "sha512-
oSXzaWypCMHkPC3NvBEaPHf0KsA5mvPrOPgQWDsbg8n7orZ290M0BmC/jgRZ4vcJ6DTAhjrsSYgdsW/
F+MFOBA==",
      "license": "MIT"
    },
    "node_modules/lodash.isstring": {
      "version": "4.0.1",
      "resolved": "https://registry.npmjs.org/lodash.isstring/-/
lodash.isstring-4.0.1.tgz",
      "integrity": "sha512-0wJxfxH1wgO3GrbuP+dTTk7op+6L41QCXbGINEmD+ny/G/
eCqGzxyCsh7159S+mgDDcoarnBw6PC1PS5+wUGgw==",
      "license": "MIT"
    },
    "node_modules/lodash.once": {
      "version": "4.1.1",
      "resolved": "https://registry.npmjs.org/lodash.once/-/
lodash.once-4.1.1.tgz",
      "integrity": "sha512-Sb487aTOCr9drQVL8pIxOzVhafOjZN9UU54hiN8PU3uAiSV7lx1yYNp
bNmex2PK6dSJoNTSJUUswT651yww3Mg==",
      "license": "MIT"
    },
    "node_modules/lottie-react": {
      "version": "2.4.1",
      "resolved": "https://registry.npmjs.org/lottie-react/-/lottie-
react-2.4.1.tgz",
      "integrity": "sha512-LQrH7jlkigIIv+
+wIyrOYFLHSKQpEY4zehPicL9bQsrt1rnoKRYCYgpCUe5maqylNtacy58/sQDZTkwMcTRxZw==",
      "license": "MIT",
```

```
      "dependencies": {
        "lottie-web": "^5.10.2"
      },
      "peerDependencies": {
        "react": "^16.8.0 || ^17.0.0 || ^18.0.0 || ^19.0.0",
        "react-dom": "^16.8.0 || ^17.0.0 || ^18.0.0 || ^19.0.0"
      }
    },
    "node_modules/lottie-web": {
      "version": "5.12.2",
      "resolved": "https://registry.npmjs.org/lottie-web/-/lottie-web-5.12.2.tgz",
      "integrity": "sha512-uvhvYPC8kGPjXT3MyKMrL3JitEAmDMp30lVkuq/590Mw9ok6pWcFCwX
Jveo0t5uqYw1UREQHofD+jVpdjBv8wg==",
      "license": "MIT"
    },
    "node_modules/luxon": {
      "version": "1.28.1",
      "resolved": "https://registry.npmjs.org/luxon/-/luxon-1.28.1.tgz",
      "integrity": "sha512-
gYHAa180mKrNIUJCbwpmD0aTu9kV0dREDrwNnuyFAsO1Wt0EVYSZelPnJlbj9HplzXX/
YWXHFTL45kvZ53M0pw==",
      "license": "MIT",
      "engines": {
        "node": "*"
      }
    },
    "node_modules/make-dir": {
      "version": "3.1.0",
      "resolved": "https://registry.npmjs.org/make-dir/-/make-dir-3.1.0.tgz",
      "integrity": "sha512-
g3FeP20LNwhALb/6Cz6Dd4F2ngze0jz7tbzrD2wAV+o9FeNHe4rL+yK2md0J/
fiSf1sa1ADhXqi5+oVwOM/eGw==",
      "license": "MIT",
      "dependencies": {
        "semver": "^6.0.0"
      },
      "engines": {
        "node": ">=8"
      },
      "funding": {
        "url": "https://github.com/sponsors/sindresorhus"
      }
    },
    "node_modules/make-dir/node_modules/semver": {
      "version": "6.3.1",
      "resolved": "https://registry.npmjs.org/semver/-/semver-6.3.1.tgz",
      "integrity": "sha512-
BR7VvDCVHO+q2xBEWskxS6DJE1qRnb7DxzUrogb71CWoSficBxYsiAGd+Kl0mmq/
MprG9yArRkyrQxTO6XjMzA==",
      "license": "ISC",
      "bin": {
        "semver": "bin/semver.js"
      }
    },
    "node_modules/math-intrinsics": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/math-intrinsics/-/math-
intrinsics-1.1.0.tgz",
      "integrity": "sha512-/IXtbwEk5HTPyEwyKX6hGkYXxM9nbj64B+ilVJnC/
R6B0pH5G4V3b0pVbL7DBj4tkhBAppbQUlf6F6Xl9LHu1g==",
```

```
      "license": "MIT",
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/media-typer": {
      "version": "0.3.0",
      "resolved": "https://registry.npmjs.org/media-typer/-/media-
typer-0.3.0.tgz",
      "integrity": "sha512-dq+qelQ9akHpcOl/
gUVRTxVIOkAJ1wR3QAvb4RsVjS8oVoFjDGTc679wJYmUmknUF5HwMLOgb5O+a3KxfWapPQ==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/memory-pager": {
      "version": "1.5.0",
      "resolved": "https://registry.npmjs.org/memory-pager/-/memory-
pager-1.5.0.tgz",
      "integrity": "sha512-ZS4Bp4r/
Zoeq6+NLJpP+0Zzm0pR8whtGPf1XExKLJBAczGMnSi3It14OiNCStjQjM6NU1okjQGSxgEZN8eBYKg==",
      "license": "MIT",
      "optional": true
    },
    "node_modules/merge-descriptors": {
      "version": "1.0.3",
      "resolved": "https://registry.npmjs.org/merge-descriptors/-/merge-
descriptors-1.0.3.tgz",
      "integrity": "sha512-gaNvAS7TZ897/rVaZ0nMtAyxNyi/
pdbjbAwUpFQpN70GqnVfOiXpeUUMKRBmzXaSQ8DdTX4/0ms62r2K+hE6mQ==",
      "license": "MIT",
      "funding": {
        "url": "https://github.com/sponsors/sindresorhus"
      }
    },
    "node_modules/methods": {
      "version": "1.1.2",
      "resolved": "https://registry.npmjs.org/methods/-/methods-1.1.2.tgz",
      "integrity": "sha512-iclAHeNqNm68zFtnZ0e+1L2yUIdvzNoauKU4WBA3VvH/
vPFieF7qfRlwUZU+DA9P9bPXIS90ulxoUoCH23sV2w==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/mime": {
      "version": "1.6.0",
      "resolved": "https://registry.npmjs.org/mime/-/mime-1.6.0.tgz",
      "integrity": "sha512-x0Vn8spI+wuJ1O6S7gnbaQg8Pxh4NNHb7KSINmEWKiPE4RKOplvijn+
NkmYmmRgP68mc70j2EbeTFRsrswaQeg==",
      "license": "MIT",
      "bin": {
        "mime": "cli.js"
      },
      "engines": {
        "node": ">=4"
      }
    },
```

```
    "node_modules/mime-db": {
      "version": "1.52.0",
      "resolved": "https://registry.npmjs.org/mime-db/-/mime-db-1.52.0.tgz",
      "integrity": "sha512-sPU4uV7dYlvtWJxwwxHD0PuihVNiE7TyAbQ5SWxDCB9mUYvOgroQOwY
QQOKPJ8CIbE+1ETVlOoK1UC2nU3gYvg==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/mime-types": {
      "version": "2.1.35",
      "resolved": "https://registry.npmjs.org/mime-types/-/mime-types-2.1.35.tgz",
      "integrity": "sha512-ZDY+bPm5zTTF+YpCrAU9nK0UgICYPT0QtT1NZWFv4s+
+TNkcgVaT0g6+4R2uI4MjQjzysHB1zxuWL50hzaeXiw==",
      "license": "MIT",
      "dependencies": {
        "mime-db": "1.52.0"
      },
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/minimatch": {
      "version": "3.1.2",
      "resolved": "https://registry.npmjs.org/minimatch/-/minimatch-3.1.2.tgz",
      "integrity": "sha512-J7p63hRiAjw1NDEww1W7i37+ByIrOWO5XQQAzZ3VOcL0PNybwpfmV/
N05zFAzwQ9USyEcX6t3UO+K5aqBQOIHw==",
      "license": "ISC",
      "dependencies": {
        "brace-expansion": "^1.1.7"
      },
      "engines": {
        "node": "*"
      }
    },
    "node_modules/minimist": {
      "version": "1.2.8",
      "resolved": "https://registry.npmjs.org/minimist/-/minimist-1.2.8.tgz",
      "integrity": "sha512-2yyAR8qBkN3YuheJanUpWC5U3bb5osDywNB8RzDVlDwDHbocAJveqqj
1u8+SVD7jkWT4yvsHCpWqqWqAxb0zCA==",
      "license": "MIT",
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/minipass": {
      "version": "5.0.0",
      "resolved": "https://registry.npmjs.org/minipass/-/minipass-5.0.0.tgz",
      "integrity": "sha512-3FnjYuehv9k6ovOEbyOswadCDPX1piCfhV8ncmYtHOjuPwylVWsghTL
o7rabjC3Rx5xD4HDx8Wm1xnMF7S5qFQ==",
      "license": "ISC",
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/minizlib": {
      "version": "2.1.2",
      "resolved": "https://registry.npmjs.org/minizlib/-/minizlib-2.1.2.tgz",
```

```
      "integrity": "sha512-
bAxsR8BVfj60DWXHE3u30oHzfl4G7khkSuPW+qvpd7jFRHm7dLxOjUk1EHACJ/
hxLY8phGJ0YhYHZo7jil7Qdg==",
      "license": "MIT",
      "dependencies": {
        "minipass": "^3.0.0",
        "yallist": "^4.0.0"
      },
      "engines": {
        "node": ">= 8"
      }
    },
    "node_modules/minizlib/node_modules/minipass": {
      "version": "3.3.6",
      "resolved": "https://registry.npmjs.org/minipass/-/minipass-3.3.6.tgz",
      "integrity": "sha512-DxiNidxSEK+tHG6zOIklvNOwm3hvCrbUrdtzY74U6HKTJxvIDfOUL5W
5P2Ghd3DTkhhKPYGqeNUIh5qcM4YBfw==",
      "license": "ISC",
      "dependencies": {
        "yallist": "^4.0.0"
      },
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/mkdirp": {
      "version": "1.0.4",
      "resolved": "https://registry.npmjs.org/mkdirp/-/mkdirp-1.0.4.tgz",
      "integrity": "sha512-
vVqVZQyf3WLx2Shd0qJ9xuvqgAyKPLAiqITEtqW0oIUjzo3PePDd6fW9iFz30ef7Ysp/
oiWqbhszeGWW2T6Gzw==",
      "license": "MIT",
      "bin": {
        "mkdirp": "bin/cmd.js"
      },
      "engines": {
        "node": ">=10"
      }
    },
    "node_modules/moment": {
      "version": "2.30.1",
      "resolved": "https://registry.npmjs.org/moment/-/moment-2.30.1.tgz",
      "integrity": "sha512-
uEmtNhbDOrWPFS+hdjFCBfy9f2YoyzRpwcl+DqpC6taX21FzsTLQVbMV/W7PzNSX6x/
bhC1zA3c2UQ5NzH6how==",
      "license": "MIT",
      "engines": {
        "node": "*"
      }
    },
    "node_modules/moment-timezone": {
      "version": "0.5.48",
      "resolved": "https://registry.npmjs.org/moment-timezone/-/moment-
timezone-0.5.48.tgz",
      "integrity": "sha512-f22b8LV1gbTO2ms2j2z13MuPogNoh5UzxL3nzNAYKGraILnbGc9NEE6
dyiiiLv46DGRb8A4kg8UKWLjPthxBHw==",
      "license": "MIT",
      "dependencies": {
        "moment": "^2.29.4"
```

```
      },
      "engines": {
        "node": "*"
      }
    },
    "node_modules/mongodb": {
      "version": "5.9.2",
      "resolved": "https://registry.npmjs.org/mongodb/-/mongodb-5.9.2.tgz",
      "integrity": "sha512-H60HecKO4Bc+7dhOv4sJlgvenK4fQNqqUIlXxZYQNbfEWSALGAwGoyJ
d/0Qwk4TttFXUOHJ2ZJQe/52ScaUwtQ==",
      "license": "Apache-2.0",
      "dependencies": {
        "bson": "^5.5.0",
        "mongodb-connection-string-url": "^2.6.0",
        "socks": "^2.7.1"
      },
      "engines": {
        "node": ">=14.20.1"
      },
      "optionalDependencies": {
        "@mongodb-js/saslprep": "^1.1.0"
      },
      "peerDependencies": {
        "@aws-sdk/credential-providers": "^3.188.0",
        "@mongodb-js/zstd": "^1.0.0",
        "kerberos": "^1.0.0 || ^2.0.0",
        "mongodb-client-encryption": ">=2.3.0 <3",
        "snappy": "^7.2.2"
      },
      "peerDependenciesMeta": {
        "@aws-sdk/credential-providers": {
          "optional": true
        },
        "@mongodb-js/zstd": {
          "optional": true
        },
        "kerberos": {
          "optional": true
        },
        "mongodb-client-encryption": {
          "optional": true
        },
        "snappy": {
          "optional": true
        }
      }
    },
    "node_modules/mongodb-connection-string-url": {
      "version": "2.6.0",
      "resolved": "https://registry.npmjs.org/mongodb-connection-string-url/-/
mongodb-connection-string-url-2.6.0.tgz",
      "integrity": "sha512-WvTZlI9ab0QYtTYnuMLgobULWhokRjtC7db9LtcVfJ+Hsnyr5eo6ZtN
At3Ly24XZScGMelOcGtm7lSn0332tPQ==",
      "license": "Apache-2.0",
      "dependencies": {
        "@types/whatwg-url": "^8.2.1",
        "whatwg-url": "^11.0.0"
      }
    },
```

```
    "node_modules/mongoose": {
      "version": "7.8.6",
      "resolved": "https://registry.npmjs.org/mongoose/-/mongoose-7.8.6.tgz",
      "integrity": "sha512-1oVPRHvcmPVwk/
zeSTEzayzQEVeYQM1D5zrkLsttfNNB7pPRUmkKeFu6gpbvyEswOuZLrWJjqB8kSTY+k2AZOA==",
      "license": "MIT",
      "dependencies": {
        "bson": "^5.5.0",
        "kareem": "2.5.1",
        "mongodb": "5.9.2",
        "mpath": "0.9.0",
        "mquery": "5.0.0",
        "ms": "2.1.3",
        "sift": "16.0.1"
      },
      "engines": {
        "node": ">=14.20.1"
      },
      "funding": {
        "type": "opencollective",
        "url": "https://opencollective.com/mongoose"
      }
    },
    "node_modules/mongoose/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/morgan": {
      "version": "1.10.0",
      "resolved": "https://registry.npmjs.org/morgan/-/morgan-1.10.0.tgz",
      "integrity": "sha512-AbegBVI4sh6El+1gNwvD5YIck7nSA36weD7xvIxG4in80j/
UoK8AEGaWnnz8v1GxonMCltmlNs5ZKbGvl9b1XQ==",
      "license": "MIT",
      "dependencies": {
        "basic-auth": "~2.0.1",
        "debug": "2.6.9",
        "depd": "~2.0.0",
        "on-finished": "~2.3.0",
        "on-headers": "~1.0.2"
      },
      "engines": {
        "node": ">= 0.8.0"
      }
    },
    "node_modules/morgan/node_modules/on-finished": {
      "version": "2.3.0",
      "resolved": "https://registry.npmjs.org/on-finished/-/on-
finished-2.3.0.tgz",
      "integrity": "sha512-ikqdkGAAyf/X/gPhXGvfgAytDZtDbr+bkNUJ0N9h5MI/
dmdgCs3l6hoHrcUv41sRKew3jIwrp4qQDXiK99Utww==",
      "license": "MIT",
      "dependencies": {
        "ee-first": "1.1.1"
      },
      "engines": {
        "node": ">= 0.8"
```

```
      }
    },
    "node_modules/mpath": {
      "version": "0.9.0",
      "resolved": "https://registry.npmjs.org/mpath/-/mpath-0.9.0.tgz",
      "integrity": "sha512-ikJRQTk8hw5DEoFVxHG1Gn9T/
xcjtdnOKIU1JTmGjZZlg9LST2mBLmcX3/ICIbgJydT2GOc15RnNy5mHmzfSew==",
      "license": "MIT",
      "engines": {
        "node": ">=4.0.0"
      }
    },
    "node_modules/mquery": {
      "version": "5.0.0",
      "resolved": "https://registry.npmjs.org/mquery/-/mquery-5.0.0.tgz",
      "integrity": "sha512-iQMncpmEK8R8ncT8HJGsGc9Dsp8xcgYMVSbs5jgnm1lFHTZqMJTUWTD
x1LBO8+mK3tPNZWFLBghQEIOULSTHZg==",
      "license": "MIT",
      "dependencies": {
        "debug": "4.x"
      },
      "engines": {
        "node": ">=14.0.0"
      }
    },
    "node_modules/mquery/node_modules/debug": {
      "version": "4.4.0",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.4.0.tgz",
      "integrity": "sha512-6WTZ/IxCY/T6BALoZHaE4ctp9xm+Z5kY/
pzYaCHRFeyVhojxlrm+46y68HA6hr0TcwEssoxNiDEUJQjfPZ/RYA==",
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
    },
    "node_modules/mquery/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/ms": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.0.0.tgz",
      "integrity": "sha512-Tpp60P6IUJDTuOq/5Z8cdskzJujfwqfOTkrwIwj7IRISpnkJnT6SyJ4
PCPnGMoFjC9ddhal5KVIYtAt97ix05A==",
      "license": "MIT"
    },
    "node_modules/multer": {
      "version": "1.4.5-lts.2",
```

```
      "resolved": "https://registry.npmjs.org/multer/-/multer-1.4.5-lts.2.tgz",
      "integrity": "sha512-VzGiVigcG9zUAoCNU+xShztrlr1auZOlurXynNvO9GiWD1/
mTBbUljOKY+qMeazBqXgRnjzeEgJI/wyjJUHg9A==",
      "deprecated": "Multer 1.x is impacted by a number of vulnerabilities, which
have been patched in 2.x. You should upgrade to the latest 2.x version.",
      "license": "MIT",
      "dependencies": {
        "append-field": "^1.0.0",
        "busboy": "^1.0.0",
        "concat-stream": "^1.5.2",
        "mkdirp": "^0.5.4",
        "object-assign": "^4.1.1",
        "type-is": "^1.6.4",
        "xtend": "^4.0.0"
      },
      "engines": {
        "node": ">= 6.0.0"
      }
    },
    "node_modules/multer/node_modules/mkdirp": {
      "version": "0.5.6",
      "resolved": "https://registry.npmjs.org/mkdirp/-/mkdirp-0.5.6.tgz",
      "integrity": "sha512-
FP+p8RB8OWpF3YZBCrP5gtADmtXApB5AMLn+vdyA+PyxCjrCs00mjyUozssO33cwDeT3wNGdLxJ5M//
YqtHAJw==",
      "license": "MIT",
      "dependencies": {
        "minimist": "^1.2.6"
      },
      "bin": {
        "mkdirp": "bin/cmd.js"
      }
    },
    "node_modules/negotiator": {
      "version": "0.6.3",
      "resolved": "https://registry.npmjs.org/negotiator/-/negotiator-0.6.3.tgz",
      "integrity": "sha512-+EUsqGPLsM+j/zdChZjsnX51g4XrHFOIXwfnCVPGlQk/
k5giakcKsuxCObBRu6DSm9opw/O6slWbJdghQM4bBg==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/node-addon-api": {
      "version": "5.1.0",
      "resolved": "https://registry.npmjs.org/node-addon-api/-/node-addon-
api-5.1.0.tgz",
      "integrity": "sha512-eh0GgfEkpnoWDq+VY8OyvYhFEzBk6jIYbRKdIlyTiAXIVJ8PyBaKb0r
p7oDtoddbdoHWhq8wwr+XZ81F1rpNdA==",
      "license": "MIT"
    },
    "node_modules/node-cron": {
      "version": "3.0.3",
      "resolved": "https://registry.npmjs.org/node-cron/-/node-cron-3.0.3.tgz",
      "integrity": "sha512-dOal67//
nohNgYWb+nWmg5dkFdIwDm8EpeGYMekPMrngV3637lqnX0lbUcCtgibHTz6SEz7DAIjKvKDFYCnO1A==",
      "license": "ISC",
      "dependencies": {
        "uuid": "8.3.2"
```

```
      },
      "engines": {
        "node": ">=6.0.0"
      }
    },
    "node_modules/node-fetch": {
      "version": "2.7.0",
      "resolved": "https://registry.npmjs.org/node-fetch/-/node-fetch-2.7.0.tgz",
      "integrity": "sha512-c4FRfUm/
dbcWZ7U+1Wq0AwCyFL+3nt2bEw05wfxSz+DWpWsitgmSgYmy2dQdWyKC1694ELPqMs/YzUSNozLt8A==",
      "license": "MIT",
      "dependencies": {
        "whatwg-url": "^5.0.0"
      },
      "engines": {
        "node": "4.x || >=6.0.0"
      },
      "peerDependencies": {
        "encoding": "^0.1.0"
      },
      "peerDependenciesMeta": {
        "encoding": {
          "optional": true
        }
      }
    },
    "node_modules/node-fetch/node_modules/tr46": {
      "version": "0.0.3",
      "resolved": "https://registry.npmjs.org/tr46/-/tr46-0.0.3.tgz",
      "integrity": "sha512-N3WMsuqV66lT30CrXNbEjx4GEwlow3v6rr4mCcv6prnfwhS01rkgyFd
jPNBYd9br7LpXV1+Emh01fHnq2Gdgrw==",
      "license": "MIT"
    },
    "node_modules/node-fetch/node_modules/webidl-conversions": {
      "version": "3.0.1",
      "resolved": "https://registry.npmjs.org/webidl-conversions/-/webidl-
conversions-3.0.1.tgz",
      "integrity": "sha512-2JAn3z8AR6rjK8Sm8orRC0h/bcl/
DqL7tRPdGZ4I1CjdF+EaMLmYxBHyXuKL849eucPFhvBoxMsflfOb8kxaeQ==",
      "license": "BSD-2-Clause"
    },
    "node_modules/node-fetch/node_modules/whatwg-url": {
      "version": "5.0.0",
      "resolved": "https://registry.npmjs.org/whatwg-url/-/whatwg-url-5.0.0.tgz",
      "integrity": "sha512-
saE57nupxk6v3HY35+jzBwYa0rKSy0XR8JSxZPwgLr7ys0IBzhGviA1/
TUGJLmSVqs8pb9AnvICXEuOHLprYTw==",
      "license": "MIT",
      "dependencies": {
        "tr46": "~0.0.3",
        "webidl-conversions": "^3.0.0"
      }
    },
    "node_modules/nodemailer": {
      "version": "6.10.1",
      "resolved": "https://registry.npmjs.org/nodemailer/-/nodemailer-6.10.1.tgz",
      "integrity": "sha512-Z+iLaBGVaSjbIzQ4pX6XV41HrooLsQ10ZWPUehGmuantvzWoDVBnmsd
UcOIDM1t+yPor5pDhVlDESgOMEGxhHA==",
      "license": "MIT-0",
```

```
      "engines": {
        "node": ">=6.0.0"
      }
    },
    "node_modules/nodemon": {
      "version": "3.1.10",
      "resolved": "https://registry.npmjs.org/nodemon/-/nodemon-3.1.10.tgz",
      "integrity": "sha512-WDjw3pJ0/0jMFmyNDp3gvY2YizjLmmOUQo6DEBY+JgdvW/
yQ9mEeSw6H5ythl5Ny2ytb7f9C2nIbjSxMNzbJXw==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "chokidar": "^3.5.2",
        "debug": "^4",
        "ignore-by-default": "^1.0.1",
        "minimatch": "^3.1.2",
        "pstree.remy": "^1.1.8",
        "semver": "^7.5.3",
        "simple-update-notifier": "^2.0.0",
        "supports-color": "^5.5.0",
        "touch": "^3.1.0",
        "undefsafe": "^2.0.5"
      },
      "bin": {
        "nodemon": "bin/nodemon.js"
      },
      "engines": {
        "node": ">=10"
      },
      "funding": {
        "type": "opencollective",
        "url": "https://opencollective.com/nodemon"
      }
    },
    "node_modules/nodemon/node_modules/debug": {
      "version": "4.4.0",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.4.0.tgz",
      "integrity": "sha512-6WTZ/IxCY/T6BALoZHaE4ctp9xm+Z5kY/
pzYaCHRFeyVhojxlrm+46y68HA6hr0TcwEssoxNiDEUJQjfPZ/RYA==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
    },
    "node_modules/nodemon/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "dev": true,
```

```
      "license": "MIT"
    },
    "node_modules/nopt": {
      "version": "5.0.0",
      "resolved": "https://registry.npmjs.org/nopt/-/nopt-5.0.0.tgz",
      "integrity": "sha512-
Tbj67rffqceeLpcRXrT7vKAN8CwfPeIBgM7E6iBkmKLV7bEMwpGgYLGv0jACUsECaa/
vuxP0IjEont6umdMgtQ==",
      "license": "ISC",
      "dependencies": {
        "abbrev": "1"
      },
      "bin": {
        "nopt": "bin/nopt.js"
      },
      "engines": {
        "node": ">=6"
      }
    },
    "node_modules/normalize-path": {
      "version": "3.0.0",
      "resolved": "https://registry.npmjs.org/normalize-path/-/normalize-
path-3.0.0.tgz",
      "integrity":
"sha512-6eZs5Ls3WtCisHWp9S2GUy8dqkpGi4BVSz3GaqiE6ezub0512ESztXUwUB6C6IKbQkY2Pnb/
mD4WYojCRwcwLA==",
      "dev": true,
      "license": "MIT",
      "engines": {
        "node": ">=0.10.0"
      }
    },
    "node_modules/npmlog": {
      "version": "5.0.1",
      "resolved": "https://registry.npmjs.org/npmlog/-/npmlog-5.0.1.tgz",
      "integrity": "sha512-AqZtDUWOMKs1G/8lwylVjrdYgqA4d9nu8hc+0gzRxlDb1I10+FHBGMX
s6aiQHFdCUUlqH99MUMuLfzWDNDtfxw==",
      "deprecated": "This package is no longer supported.",
      "license": "ISC",
      "dependencies": {
        "are-we-there-yet": "^2.0.0",
        "console-control-strings": "^1.1.0",
        "gauge": "^3.0.0",
        "set-blocking": "^2.0.0"
      }
    },
    "node_modules/numbered": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/numbered/-/numbered-1.1.0.tgz",
      "integrity": "sha512-pv/ue2Odr7IfYOO0byC1KgBI10wo5YDauLhxY6/
saNzAdAs0r1SotGCPzzCLNPL0xtrAwWRialLu23AAu9xO1g==",
      "license": "MIT"
    },
    "node_modules/object-assign": {
      "version": "4.1.1",
      "resolved": "https://registry.npmjs.org/object-assign/-/object-
assign-4.1.1.tgz",
      "integrity": "sha512-rJgTQnkUnH1sFw8yT6VSU3zD3sWmu6sZhIseY8VX+GRu3P6F7Fu+JND
oXfklElbLJSnc3FUQHVe4cU5hj+BcUg==",
```

```
      "license": "MIT",
      "engines": {
        "node": ">=0.10.0"
      }
    },
    "node_modules/object-inspect": {
      "version": "1.13.4",
      "resolved": "https://registry.npmjs.org/object-inspect/-/object-
inspect-1.13.4.tgz",
      "integrity": "sha512-W67iLl4J2EXEGTbfeHCffrjDfitvLANg0UlX3wFUUSTx92KXRFegMHU
VgSqE+wvhAbi4WqjGg9czysTV2Epbew==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/object-keys": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/object-keys/-/object-
keys-1.1.1.tgz",
      "integrity": "sha512-NuAESUOUMrlIXOfHKzD6bpPu3tYt3xvjNdRIQ+FeT0lNb4K8WR70CaD
xhuNguS2XG+GjkyMwOzsN5ZktImfhLA==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/on-finished": {
      "version": "2.4.1",
      "resolved": "https://registry.npmjs.org/on-finished/-/on-
finished-2.4.1.tgz",
      "integrity": "sha512-oVlzkg3ENAhCk2zdv7IJwd/
QUD4z2RxRwpkcGY8psCVcCYZNq4wYnVWALHM+brtuJjePWiYF/ClmuDr8Ch5+kg==",
      "license": "MIT",
      "dependencies": {
        "ee-first": "1.1.1"
      },
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/on-headers": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/on-headers/-/on-headers-1.0.2.tgz",
      "integrity": "sha512-pZAE+FJLoyITytdqK0U5s+FIpjN0JP3OzFi/u8Rx+EV5/
W+JTWGXG8xFzevE7AjBfDqHv/8vL8qQsIhHnqRkrA==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/once": {
      "version": "1.4.0",
      "resolved": "https://registry.npmjs.org/once/-/once-1.4.0.tgz",
      "integrity": "sha512-
lNaJgI+2Q5URQBkccEKHTQOPaXdUxnZZElQTZY0MFUAuaEqe1E+Nyvgdz/aIyNi6Z9MzO5dv1H8n58/
GELp3+w==",
```

```
      "license": "ISC",
      "dependencies": {
        "wrappy": "1"
      }
    },
    "node_modules/pako": {
      "version": "0.2.9",
      "resolved": "https://registry.npmjs.org/pako/-/pako-0.2.9.tgz",
      "integrity": "sha512-NUcwaKxUxWrZLpDG+z/xZaCgQITkA/Dv4V/T6bw7VON6l1Xz/
VnrBqrYjZQ12TamKHzITTfOEIYUj48y2KXImA==",
      "license": "MIT"
    },
    "node_modules/parseurl": {
      "version": "1.3.3",
      "resolved": "https://registry.npmjs.org/parseurl/-/parseurl-1.3.3.tgz",
      "integrity": "sha512-CiyeOxFT/
JZyN5m0z9PfXw4SCBJ6Sygz1Dpl0wqjlhDEGGBP1GnsUVEL0p63hoG1fcj3fHynXi9NYO4nWOL+qQ==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/path-is-absolute": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/path-is-absolute/-/path-is-
absolute-1.0.1.tgz",
      "integrity": "sha512-AVbw3UJ2e9bq64vSaS9Am0fje1Pa8pbGqTTsmXfaIiMpnr5DlDhfJOu
Lj9Sf95ZPVDAUerDfEk88MPmPe7UCQg==",
      "license": "MIT",
      "engines": {
        "node": ">=0.10.0"
      }
    },
    "node_modules/path-to-regexp": {
      "version": "0.1.12",
      "resolved": "https://registry.npmjs.org/path-to-regexp/-/path-to-
regexp-0.1.12.tgz",
      "integrity": "sha512-RA1GjUVMnvYFxuqovrEqZoxxW5NUZqbwKtYz/
Tt7nXerk0LbLblQmrsgdeOxV5SFHf0UDggjS/bSeOZwt1pmEQ==",
      "license": "MIT"
    },
    "node_modules/pdfkit": {
      "version": "0.17.1",
      "resolved": "https://registry.npmjs.org/pdfkit/-/pdfkit-0.17.1.tgz",
      "integrity": "sha512-Kkf1I9no14O/
uo593DYph5u3QwiMfby7JsBSErN1WqeyTgCBNJE3K4pXBn3TgkdKUIVu+buSl4bYUNC+8Up4xg==",
      "license": "MIT",
      "dependencies": {
        "crypto-js": "^4.2.0",
        "fontkit": "^2.0.4",
        "jpeg-exif": "^1.1.4",
        "linebreak": "^1.1.0",
        "png-js": "^1.0.0"
      }
    },
    "node_modules/picomatch": {
      "version": "2.3.1",
      "resolved": "https://registry.npmjs.org/picomatch/-/picomatch-2.3.1.tgz",
      "integrity": "sha512-JU3teHTNjmE2VCGFzuY8EXzCDVwEqB2a8fsIvwaStHhAWJEeVd1o1QD
```

```
80CU6+ZdEXXSLbSsuLwJjkCBWqRQUVA==",
      "dev": true,
      "license": "MIT",
      "engines": {
        "node": ">=8.6"
      },
      "funding": {
        "url": "https://github.com/sponsors/jonschlinkert"
      }
    },
    "node_modules/png-js": {
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/png-js/-/png-js-1.0.0.tgz",
      "integrity": "sha512-
k+YsbhpA9e+EFfKjTCH3VW6aoKlyNYI6NYdTfDL4CIvFnvsuO84ttonmZE7rc+v23SLTH8XX+5w/
Ak9v0xGY4g=="
    },
    "node_modules/process-nextick-args": {
      "version": "2.0.1",
      "resolved": "https://registry.npmjs.org/process-nextick-args/-/process-
nextick-args-2.0.1.tgz",
      "integrity": "sha512-3ouUOpQhtgrbOa17J7+uxOTpITYWaGP7/AhoR3+A+/1e9skrzelGi/
dXzEYyvbxubEF6Wn2ypscTKiKJFFn1ag==",
      "license": "MIT"
    },
    "node_modules/proxy-addr": {
      "version": "2.0.7",
      "resolved": "https://registry.npmjs.org/proxy-addr/-/proxy-addr-2.0.7.tgz",
      "integrity": "sha512-llQsMLSUDUPT44jdrU/
O37qlnifitDP+ZwrmmZcoSKyLKvtZxpyV0n2/bD/N4tBAAZ/gJEdZU7KMraoK1+XYAg==",
      "license": "MIT",
      "dependencies": {
        "forwarded": "0.2.0",
        "ipaddr.js": "1.9.1"
      },
      "engines": {
        "node": ">= 0.10"
      }
    },
    "node_modules/proxy-from-env": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/proxy-from-env/-/proxy-from-
env-1.1.0.tgz",
      "integrity": "sha512-D+zkORCbA9f1tdWRK0RaCR3GPv50cMxcrz4X8k5LTSUD1Dkw47mKJEZ
QNunItRTkWwgtaUSo1RVFRIG9ZXiFYg==",
      "license": "MIT"
    },
    "node_modules/pstree.remy": {
      "version": "1.1.8",
      "resolved": "https://registry.npmjs.org/pstree.remy/-/
pstree.remy-1.1.8.tgz",
      "integrity": "sha512-77DZwxQmxKnu3aR542U+X8FypNzbfJ+C5XQDk3uWjWxn6151aIMGthW
YRXTqT1E5oJvg+ljaa2OJi+VfvCOQ8w==",
      "dev": true,
      "license": "MIT"
    },
    "node_modules/punycode": {
      "version": "2.3.1",
      "resolved": "https://registry.npmjs.org/punycode/-/punycode-2.3.1.tgz",
```

```
      "integrity": "sha512-vYt7UD1U9Wg6138shLtLOvdAu+8DsC/
ilFtEVHcH+wydcSpNE20AfSOduf6MkRFahL5FY7X1oU7nKVZFtfq8Fg==",
      "license": "MIT",
      "engines": {
        "node": ">=6"
      }
    },
    "node_modules/qs": {
      "version": "6.13.0",
      "resolved": "https://registry.npmjs.org/qs/-/qs-6.13.0.tgz",
      "integrity": "sha512-
+38qI9SOr8tfZ4QmJNplMUxqjbe7LKvvZgWdExBOmd+egZTtjLB67Gu0HRX3u/
XOq7UU2Nx6nsjvS16Z9uwfpg==",
      "license": "BSD-3-Clause",
      "dependencies": {
        "side-channel": "^1.0.6"
      },
      "engines": {
        "node": ">=0.6"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/range-parser": {
      "version": "1.2.1",
      "resolved": "https://registry.npmjs.org/range-parser/-/range-
parser-1.2.1.tgz",
      "integrity": "sha512-
Hrgsx+orqoygnmhFbKaHE6c296J+HTAQXoxEF6gNupROmmGJRoyzfG3ccAveqCBrwr/2yxQ5BVd/
GTl5agOwSg==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/raw-body": {
      "version": "2.5.2",
      "resolved": "https://registry.npmjs.org/raw-body/-/raw-body-2.5.2.tgz",
      "integrity": "sha512-8zGqypfENjCIqGhgXToC8aB2r7YrBX+AQAfIPs/
Mlk+BtPTztOvTS01NRW/3Eh60J+a48lt8qsCzirQ6loCVfA==",
      "license": "MIT",
      "dependencies": {
        "bytes": "3.1.2",
        "http-errors": "2.0.0",
        "iconv-lite": "0.4.24",
        "unpipe": "1.0.0"
      },
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/react": {
      "version": "19.1.0",
      "resolved": "https://registry.npmjs.org/react/-/react-19.1.0.tgz",
      "integrity": "sha512-FS+XFBNvn3GTAWq26joslQgWNoFu08F4kl0J4CgdNKADkdSGXQyTCnK
teIAJy96Br6YbpEU1LSzV5dYtjMkMDg==",
      "license": "MIT",
      "peer": true,
```

```
      "engines": {
        "node": ">=0.10.0"
      }
    },
    "node_modules/react-dom": {
      "version": "19.1.0",
      "resolved": "https://registry.npmjs.org/react-dom/-/react-dom-19.1.0.tgz",
      "integrity": "sha512-
Xs1hdnE+DyKgeHJeJznQmYMIBG3TKIHJJT95Q58nHLSrElKlGQqDTR2HQ9fx5CN/Gk6Vh/kupBTDLU11/
nDk/g==",
      "license": "MIT",
      "peer": true,
      "dependencies": {
        "scheduler": "^0.26.0"
      },
      "peerDependencies": {
        "react": "^19.1.0"
      }
    },
    "node_modules/readable-stream": {
      "version": "3.6.2",
      "resolved": "https://registry.npmjs.org/readable-stream/-/readable-
stream-3.6.2.tgz",
      "integrity": "sha512-9u/sniCrY3D5WdsERHzHE4G2YCXqoG5FTHUiCC4SIbr6XcLZBY05ya9
EKjYek9O5xOAwjGq+1JdGBAS7Q9ScoA==",
      "license": "MIT",
      "dependencies": {
        "inherits": "^2.0.3",
        "string_decoder": "^1.1.1",
        "util-deprecate": "^1.0.1"
      },
      "engines": {
        "node": ">= 6"
      }
    },
    "node_modules/readdirp": {
      "version": "3.6.0",
      "resolved": "https://registry.npmjs.org/readdirp/-/readdirp-3.6.0.tgz",
      "integrity": "sha512-hOS089on8RduqdbhvQ5Z37A0ESjsqz6qnRcffsMU3495FuTdqSm+7bh
J29JvIOsBDEEnan5DPu9t3To9VRlMzA==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "picomatch": "^2.2.1"
      },
      "engines": {
        "node": ">=8.10.0"
      }
    },
    "node_modules/restructure": {
      "version": "3.0.2",
      "resolved": "https://registry.npmjs.org/restructure/-/
restructure-3.0.2.tgz",
      "integrity": "sha512-gSfoiOEA0VPE6Tukkrr7I0RBdE0s7H1eFCDBk05l1KIQT1UIKNc5JZy
6jdyW6eYH3aR3g5b3PuL77rq0hvwtAw==",
      "license": "MIT"
    },
    "node_modules/rimraf": {
      "version": "3.0.2",
```

```
      "resolved": "https://registry.npmjs.org/rimraf/-/rimraf-3.0.2.tgz",
      "integrity": "sha512-JZkJMZkAGFFPP2YqXZXPbMlMBgsxzE8ILs4lMIX/2o0L9UBw9O/
Y3o6wFw/i9YLapcUJWwqbi3kdxIPdC62TIA==",
      "deprecated": "Rimraf versions prior to v4 are no longer supported",
      "license": "ISC",
      "dependencies": {
        "glob": "^7.1.3"
      },
      "bin": {
        "rimraf": "bin.js"
      },
      "funding": {
        "url": "https://github.com/sponsors/isaacs"
      }
    },
    "node_modules/safe-buffer": {
      "version": "5.2.1",
      "resolved": "https://registry.npmjs.org/safe-buffer/-/safe-
buffer-5.2.1.tgz",
      "integrity": "sha512-rp3So07KcdmmKbGvgaNxQSJr7bGVSVk5S9Eq1F+ppbRo70+YeaDxkw5
Dd8NPN+GD6bjnYm2VuPuCXmpuYvmCXQ==",
      "funding": [
        {
          "type": "github",
          "url": "https://github.com/sponsors/feross"
        },
        {
          "type": "patreon",
          "url": "https://www.patreon.com/feross"
        },
        {
          "type": "consulting",
          "url": "https://feross.org/support"
        }
      ],
      "license": "MIT"
    },
    "node_modules/safer-buffer": {
      "version": "2.1.2",
      "resolved": "https://registry.npmjs.org/safer-buffer/-/safer-
buffer-2.1.2.tgz",
      "integrity": "sha512-YZo3K82SD7Riyi0E1EQPojLz7kpepnSQI9IyPbHHg1XXXevb5dJI7tp
yN2ADxGcQbHG7vcyRHk0cbwqcQriUtg==",
      "license": "MIT"
    },
    "node_modules/scheduler": {
      "version": "0.26.0",
      "resolved": "https://registry.npmjs.org/scheduler/-/scheduler-0.26.0.tgz",
      "integrity": "sha512-NlHwttCI/
l5gCPR3D1nNXtWABUmBwvZpEQiD4IXSbIDq8BzLIK/7Ir5gTFSGZDUu37K5cMNp0hFtzO38sC7gWA==",
      "license": "MIT",
      "peer": true
    },
    "node_modules/semver": {
      "version": "7.7.1",
      "resolved": "https://registry.npmjs.org/semver/-/semver-7.7.1.tgz",
      "integrity": "sha512-hlq8tAfn0m/61p4BVRcPzIGr6LKiMwo4VM6dGi6pt4qcRkmNzTcWq6e
CEjEh+qXjkMDvPlOFFSGwQjoEa6gyMA==",
      "license": "ISC",
```

```
      "bin": {
        "semver": "bin/semver.js"
      },
      "engines": {
        "node": ">=10"
      }
    },
    "node_modules/send": {
      "version": "0.19.0",
      "resolved": "https://registry.npmjs.org/send/-/send-0.19.0.tgz",
      "integrity": "sha512-dW41u5VfLXu8SJh5bwRmyYUbAoSB3c9uQh6L8h/
KtsFREPWpbX1lrljJo186Jc4nmci/sGUZ9a0a0J2zgfq2hw==",
      "license": "MIT",
      "dependencies": {
        "debug": "2.6.9",
        "depd": "2.0.0",
        "destroy": "1.2.0",
        "encodeurl": "~1.0.2",
        "escape-html": "~1.0.3",
        "etag": "~1.8.1",
        "fresh": "0.5.2",
        "http-errors": "2.0.0",
        "mime": "1.6.0",
        "ms": "2.1.3",
        "on-finished": "2.4.1",
        "range-parser": "~1.2.1",
        "statuses": "2.0.1"
      },
      "engines": {
        "node": ">= 0.8.0"
      }
    },
    "node_modules/send/node_modules/encodeurl": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/encodeurl/-/encodeurl-1.0.2.tgz",
      "integrity": "sha512-TPJXq8JqFaVYm2CWmPvnP2Iyo4ZSM7/QKcSmuMLDObfpH5fi7RUGmd/
rTDf+rut/saiDiQEeVTNgAmJEdAOx0w==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/send/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/serve-static": {
      "version": "1.16.2",
      "resolved": "https://registry.npmjs.org/serve-static/-/serve-
static-1.16.2.tgz",
      "integrity": "sha512-VqpjJZKadQB/PEbEwvFdO43Ax5dFBZ2UECszz8bQ7pi7wt//
PWe1P6MN7eCnjsatYtBT6EuiClbjSWP2WrIoTw==",
      "license": "MIT",
      "dependencies": {
        "encodeurl": "~2.0.0",
        "escape-html": "~1.0.3",
```

```
        "parseurl": "~1.3.3",
        "send": "0.19.0"
      },
      "engines": {
        "node": ">= 0.8.0"
      }
    },
    "node_modules/set-blocking": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/set-blocking/-/set-
blocking-2.0.0.tgz",
      "integrity": "sha512-KiKBS8AnWGEyLzofFfmvKwpdPzqiy16LvQfK3yv/fVH7Bj13/
wl3JSR1J+rfgRE9q7xUJK4qvgS8raSOeLUehw==",
      "license": "ISC"
    },
    "node_modules/set-function-length": {
      "version": "1.2.2",
      "resolved": "https://registry.npmjs.org/set-function-length/-/set-function-
length-1.2.2.tgz",
      "integrity": "sha512-pgRc4hJ4/
sNjWCSS9AmnS40x3bNMDTknHgL5UaMBTMyJnU90EgWh1Rz+MC9eFu4BuN/UwZjKQuY/1v3rM7HMfg==",
      "license": "MIT",
      "dependencies": {
        "define-data-property": "^1.1.4",
        "es-errors": "^1.3.0",
        "function-bind": "^1.1.2",
        "get-intrinsic": "^1.2.4",
        "gopd": "^1.0.1",
        "has-property-descriptors": "^1.0.2"
      },
      "engines": {
        "node": ">= 0.4"
      }
    },
    "node_modules/setprototypeof": {
      "version": "1.2.0",
      "resolved": "https://registry.npmjs.org/setprototypeof/-/
setprototypeof-1.2.0.tgz",
      "integrity": "sha512-
E5LDX7Wrp85Kil5bhZv46j8jOeboKq5JMmYM3gVGdGH8xFpPWXUMsNrlODCrkoxMEeNi/
XZIwuRvY4XNwYMJpw==",
      "license": "ISC"
    },
    "node_modules/side-channel": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/side-channel/-/side-
channel-1.1.0.tgz",
      "integrity": "sha512-
ZX99e6tRweoUXqR+VBrslhda51Nh5MTQwou5tnUDgbtyM0dBgmhEDtWGP/xbKn6hqfPRHujUNwz5fy/
wbbhnpw==",
      "license": "MIT",
      "dependencies": {
        "es-errors": "^1.3.0",
        "object-inspect": "^1.13.3",
        "side-channel-list": "^1.0.0",
        "side-channel-map": "^1.0.1",
        "side-channel-weakmap": "^1.0.2"
      },
      "engines": {
```

```
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/side-channel-list": {
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/side-channel-list/-/side-channel-
list-1.0.0.tgz",
      "integrity": "sha512-FCLHtRD/
gnpCiCHEiJLOwdmFP+wzCmDEkc9y7NsYxeF4u7Btsn1ZuwgwJGxImImHicJArLP4R0yX4c2KCrMrTA==",
      "license": "MIT",
      "dependencies": {
        "es-errors": "^1.3.0",
        "object-inspect": "^1.13.3"
      },
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/side-channel-map": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/side-channel-map/-/side-channel-
map-1.0.1.tgz",
      "integrity": "sha512-VCjCNfgMsby3tTdo02nbjtM/
ewra6jPHmpThenkTYh8pG9ucZ/1P8So4u4FGBek/BjpOVsDCMoLA/iuBKIFXRA==",
      "license": "MIT",
      "dependencies": {
        "call-bound": "^1.0.2",
        "es-errors": "^1.3.0",
        "get-intrinsic": "^1.2.5",
        "object-inspect": "^1.13.3"
      },
      "engines": {
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/side-channel-weakmap": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/side-channel-weakmap/-/side-channel-
weakmap-1.0.2.tgz",
      "integrity": "sha512-WPS/HvHQTYnHisLo9McqBHOJk2FkHO/
tlpvldyrnem4aeQp4hai3gythswg6p01oSoTl58rcpiFAjF2br2Ak2A==",
      "license": "MIT",
      "dependencies": {
        "call-bound": "^1.0.2",
        "es-errors": "^1.3.0",
        "get-intrinsic": "^1.2.5",
        "object-inspect": "^1.13.3",
        "side-channel-map": "^1.0.1"
      },
      "engines": {
```

```
        "node": ">= 0.4"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/sift": {
      "version": "16.0.1",
      "resolved": "https://registry.npmjs.org/sift/-/sift-16.0.1.tgz",
      "integrity": "sha512-Wv6BjQ5zbhW7VFefWusVP33T/EM0vYikCaQ2qR8yULbsilAT8/
wQaXvuQ3ptGLpoKx+lihJE3y2UTgKDyyNHZQ==",
      "license": "MIT"
    },
    "node_modules/signal-exit": {
      "version": "3.0.7",
      "resolved": "https://registry.npmjs.org/signal-exit/-/signal-
exit-3.0.7.tgz",
      "integrity": "sha512-wnD2ZE+l+SPC/
uoS0vXeE9L1+0wuaMqKlfz9AMUo38JsyLSBWSFcHR1Rri62LZc12vLr1gb3jl7iwQhgwpAbGQ==",
      "license": "ISC"
    },
    "node_modules/simple-update-notifier": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/simple-update-notifier/-/simple-
update-notifier-2.0.0.tgz",
      "integrity": "sha512-a2B9Y0KlNXl9u/vsW6sTIu9vGEpfKu2wRV6l1H3XEas/0gUIzGzBoP/
IouTcUQbm9JWZLH3COxyn03TYlFax6w==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "semver": "^7.5.3"
      },
      "engines": {
        "node": ">=10"
      }
    },
    "node_modules/smart-buffer": {
      "version": "4.2.0",
      "resolved": "https://registry.npmjs.org/smart-buffer/-/smart-
buffer-4.2.0.tgz",
      "integrity": "sha512-94hK0Hh8rPqQl2xXc3HsaBoOXKV20MToPkcXvwbISWLEs+64sBq5kFg
n2kJDHb1Pry9yrP0dxrCI9RRci7RXKg==",
      "license": "MIT",
      "engines": {
        "node": ">= 6.0.0",
        "npm": ">= 3.0.0"
      }
    },
    "node_modules/socket.io": {
      "version": "4.8.1",
      "resolved": "https://registry.npmjs.org/socket.io/-/socket.io-4.8.1.tgz",
      "integrity": "sha512-oZ7iUCxph8WYRHHcjBEc9unw3adt5CmSNlppj/5Q4k2RIrhl8Z5yY2X
r4j9zj0+wzVZ0bxmYoGSzKJnRl6A4yg==",
      "license": "MIT",
      "dependencies": {
        "accepts": "~1.3.4",
        "base64id": "~2.0.0",
        "cors": "~2.8.5",
        "debug": "~4.3.2",
```

```
        "engine.io": "~6.6.0",
        "socket.io-adapter": "~2.5.2",
        "socket.io-parser": "~4.2.4"
      },
      "engines": {
        "node": ">=10.2.0"
      }
    },
    "node_modules/socket.io-adapter": {
      "version": "2.5.5",
      "resolved": "https://registry.npmjs.org/socket.io-adapter/-/socket.io-
adapter-2.5.5.tgz",
      "integrity": "sha512-eLDQas5dzPgOWCk9GuuJC2lBqItuhKI4uxGgo9aIV7MYbk2h9Q6uULE
h8WBzThoI7l+qU9Ast9fVUmkqPP9wYg==",
      "license": "MIT",
      "dependencies": {
        "debug": "~4.3.4",
        "ws": "~8.17.1"
      }
    },
    "node_modules/socket.io-adapter/node_modules/debug": {
      "version": "4.3.7",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.3.7.tgz",
      "integrity": "sha512-Er2nc/
H7RrMXZBFCEim6TCmMk02Z8vLC2Rbi1KEBggpo0fS6l0S1nnapwmIi3yW/
+GOJap1Krg4w0Hg80oCqgQ==",
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
    },
    "node_modules/socket.io-adapter/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/socket.io-parser": {
      "version": "4.2.4",
      "resolved": "https://registry.npmjs.org/socket.io-parser/-/socket.io-
parser-4.2.4.tgz",
      "integrity": "sha512-/GbIKmo8ioc+NIWIhwdecY0ge+qVBSMdgxGygevmdHj24bsfgtCmcUU
cQ5ZzcylGFHsN3k4HB4Cgkl96KVnuew==",
      "license": "MIT",
      "dependencies": {
        "@socket.io/component-emitter": "~3.1.0",
        "debug": "~4.3.1"
      },
      "engines": {
        "node": ">=10.0.0"
```

```
      }
    },
    "node_modules/socket.io-parser/node_modules/debug": {
      "version": "4.3.7",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.3.7.tgz",
      "integrity": "sha512-Er2nc/
H7RrMXZBFCEim6TCmMk02Z8vLC2Rbi1KEBggpo0fS6l0S1nnapwmIi3yW/
+GOJap1Krg4w0Hg80oCqgQ==",
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
    },
    "node_modules/socket.io-parser/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/socket.io/node_modules/debug": {
      "version": "4.3.7",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.3.7.tgz",
      "integrity": "sha512-Er2nc/
H7RrMXZBFCEim6TCmMk02Z8vLC2Rbi1KEBggpo0fS6l0S1nnapwmIi3yW/
+GOJap1Krg4w0Hg80oCqgQ==",
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
    },
    "node_modules/socket.io/node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity": "sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/
s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/socks": {
      "version": "2.8.4",
      "resolved": "https://registry.npmjs.org/socks/-/socks-2.8.4.tgz",
      "integrity": "sha512-D3YaD0aRxR3mEcqnidIs7ReYJFVzWdd6fXJYUM8ixcQcJRGTka/
b3saV0KflYhyVJXKhb947GndU35SxYNResQ==",
```

```
      "license": "MIT",
      "dependencies": {
        "ip-address": "^9.0.5",
        "smart-buffer": "^4.2.0"
      },
      "engines": {
        "node": ">= 10.0.0",
        "npm": ">= 3.0.0"
      }
    },
    "node_modules/sparse-bitfield": {
      "version": "3.0.3",
      "resolved": "https://registry.npmjs.org/sparse-bitfield/-/sparse-
bitfield-3.0.3.tgz",
      "integrity": "sha512-kvzhi7vqKTfkh0PZU+2D2PIllw2ymqJKujUcyPMd9Y75Nv4nPbGJZXN
hxsgdQab2BmlDct1YnfQCguEvHr7VsQ==",
      "license": "MIT",
      "optional": true,
      "dependencies": {
        "memory-pager": "^1.0.2"
      }
    },
    "node_modules/sprintf-js": {
      "version": "1.1.3",
      "resolved": "https://registry.npmjs.org/sprintf-js/-/sprintf-js-1.1.3.tgz",
      "integrity": "sha512-Oo+0REFV59/
rz3gfJNKQiBlwfHaSESl1pcGyABQsnnIfWOFt6JNj5gCog2U6MLZ//IGYD+nA8nI+mTShREReaA==",
      "license": "BSD-3-Clause"
    },
    "node_modules/statuses": {
      "version": "2.0.1",
      "resolved": "https://registry.npmjs.org/statuses/-/statuses-2.0.1.tgz",
      "integrity": "sha512-
RwNA9Z/7PrK06rYLIzFMlaF+l73iwpzsqRIFgbMLbTcLD6cOao82TaWefPXQvB2fOC4AjuYSEndS7N/
mTCbkdQ==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/streamsearch": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/streamsearch/-/
streamsearch-1.1.0.tgz",
      "integrity": "sha512-
Mcc5wHehp9aXz1ax6bZUyY5afg9u2rv5cqQI3mRrYkGC8rW2hM02jWuwjtL+
+LS5qinSyhj2QfLyNsuc+VsExg==",
      "engines": {
        "node": ">=10.0.0"
      }
    },
    "node_modules/string_decoder": {
      "version": "1.3.0",
      "resolved": "https://registry.npmjs.org/string_decoder/-/
string_decoder-1.3.0.tgz",
      "integrity": "sha512-hkRX8U1WjJFd8LsDJ2yQ/
wWWxaopEsABU1XfkM8A+j0+85JAGppt16cr1Whg6KIbb4okU6Mql6BOj+uup/wKeA==",
      "license": "MIT",
      "dependencies": {
```

```
        "safe-buffer": "~5.2.0"
      }
    },
    "node_modules/string-width": {
      "version": "4.2.3",
      "resolved": "https://registry.npmjs.org/string-width/-/string-
width-4.2.3.tgz",
      "integrity": "sha512-
wKyQRQpjJ0sIp62ErSZdGsjMJWsap5oRNihHhu6G7JVO/9jIB6UyevL+tXuOqrng8j/
cxKTWyWUwvSTriiZz/g==",
      "license": "MIT",
      "dependencies": {
        "emoji-regex": "^8.0.0",
        "is-fullwidth-code-point": "^3.0.0",
        "strip-ansi": "^6.0.1"
      },
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/strip-ansi": {
      "version": "6.0.1",
      "resolved": "https://registry.npmjs.org/strip-ansi/-/strip-ansi-6.0.1.tgz",
      "integrity": "sha512-Y38VPSHcqkFrCpFnQ9vuSXmquuv5oXOKpGeT6aGrr3o3Gc9AlVa6JBf
USOCnbxGGZF+/0ooI7KrPuUSztUdU5A==",
      "license": "MIT",
      "dependencies": {
        "ansi-regex": "^5.0.1"
      },
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/strnum": {
      "version": "1.1.2",
      "resolved": "https://registry.npmjs.org/strnum/-/strnum-1.1.2.tgz",
      "integrity": "sha512-vrN+B7DBIoTTZjnPNewwhx6cBA/
H+IS7rfW68n7XxC1y7uoiGQBxaKzqucGUgavX15dJgiGztLJ8vxuEzwqBdA==",
      "funding": [
        {
          "type": "github",
          "url": "https://github.com/sponsors/NaturalIntelligence"
        }
      ],
      "license": "MIT",
      "optional": true
    },
    "node_modules/supports-color": {
      "version": "5.5.0",
      "resolved": "https://registry.npmjs.org/supports-color/-/supports-
color-5.5.0.tgz",
      "integrity": "sha512-QjVjwdXIt408MIiAqCX4oUKsgU2EqAGzs2Ppkm4aQYbjm+ZEWEcW4Sf
FNTr4uMNZma0ey4f5lgLrkB0aX0QMow==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "has-flag": "^3.0.0"
      },
      "engines": {
```

```
        "node": ">=4"
      }
    },
    "node_modules/tar": {
      "version": "6.2.1",
      "resolved": "https://registry.npmjs.org/tar/-/tar-6.2.1.tgz",
      "integrity": "sha512-DZ4yORTwrbTj/7MZYq2w+/ZFdI6OZ/
f9SFHR+71gIVUZhOQPHzVCLpvRnPgyaMpfWxxk/4ONva3GQSyNIKRv6A==",
      "license": "ISC",
      "dependencies": {
        "chownr": "^2.0.0",
        "fs-minipass": "^2.0.0",
        "minipass": "^5.0.0",
        "minizlib": "^2.1.1",
        "mkdirp": "^1.0.3",
        "yallist": "^4.0.0"
      },
      "engines": {
        "node": ">=10"
      }
    },
    "node_modules/tiny-inflate": {
      "version": "1.0.3",
      "resolved": "https://registry.npmjs.org/tiny-inflate/-/tiny-
inflate-1.0.3.tgz",
      "integrity": "sha512-pkY1fj1cKHb2seWDy0B16HeWyczlJA9/WW3u3c4z/
NiWDsO3DOU5D7nhTLE9CF0yXv/QZFY7sEJmj24dK+Rrqw==",
      "license": "MIT"
    },
    "node_modules/to-regex-range": {
      "version": "5.0.1",
      "resolved": "https://registry.npmjs.org/to-regex-range/-/to-regex-
range-5.0.1.tgz",
      "integrity": "sha512-65P7iz6X5yEr1cwcgvQxbbIw7Uk3gOy5dIdtZ4rDveLqhrdJP+Li/
Hx6tyK0NEb+2GCyneCMJiGqrADCSNk8sQ==",
      "dev": true,
      "license": "MIT",
      "dependencies": {
        "is-number": "^7.0.0"
      },
      "engines": {
        "node": ">=8.0"
      }
    },
    "node_modules/toidentifier": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/toidentifier/-/
toidentifier-1.0.1.tgz",
      "integrity": "sha512-o5sSPKEkg/DIQNmH43V0/
uerLrpzVedkUh8tGNvaeXpfpuwjKenlSox/2O/BTlZUtEe+JG7s5YhEz608PlAHRA==",
      "license": "MIT",
      "engines": {
        "node": ">=0.6"
      }
    },
    "node_modules/touch": {
      "version": "3.1.1",
      "resolved": "https://registry.npmjs.org/touch/-/touch-3.1.1.tgz",
      "integrity": "sha512-
```

```
r0eojU4bI8MnHr8c5bNo7lJDdI2qXlWWJk6a9EAFG7vbhTjElYhBVS3/miuE0uOuoLdb8Mc/
rVfsmm6eo5o9GA==",
      "dev": true,
      "license": "ISC",
      "bin": {
        "nodetouch": "bin/nodetouch.js"
      }
    },
    "node_modules/tr46": {
      "version": "3.0.0",
      "resolved": "https://registry.npmjs.org/tr46/-/tr46-3.0.0.tgz",
      "integrity": "sha512-
l7FvfAHlcmulp8kr+flpQZmVwtu7nfRV7NZujtN0OqES8EL4O4e0qqzL0DC5gAvx/
ZC/9lk6rhcUwYvkBnBnYA==",
      "license": "MIT",
      "dependencies": {
        "punycode": "^2.1.1"
      },
      "engines": {
        "node": ">=12"
      }
    },
    "node_modules/tslib": {
      "version": "2.8.1",
      "resolved": "https://registry.npmjs.org/tslib/-/tslib-2.8.1.tgz",
      "integrity": "sha512-oJFu94HQb+KVduSUQL7wnpmqnfmLsOA/nAh6b6EH0wCEoK0/
mPeXU6c3wKDV83MkOuHPRHtSXKKU99IBazS/2w==",
      "license": "0BSD"
    },
    "node_modules/type-is": {
      "version": "1.6.18",
      "resolved": "https://registry.npmjs.org/type-is/-/type-is-1.6.18.tgz",
      "integrity": "sha512-TkRKr9sUTxEH8MdfuCSP7VizJyzRNMjj2J2do2Jr3Kym598JVdEksuz
PQCnlFPW4ky9Q+iA+ma9BGm06XQBy8g==",
      "license": "MIT",
      "dependencies": {
        "media-typer": "0.3.0",
        "mime-types": "~2.1.24"
      },
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/typedarray": {
      "version": "0.0.6",
      "resolved": "https://registry.npmjs.org/typedarray/-/typedarray-0.0.6.tgz",
      "integrity": "sha512-/aCDEGatGvZ2BIk+HmLf4ifCJFwvKFNb9/
JeZPMulfgFracn9QFcAf5GO8B/mweUjSoblS5In0cWhqpfs/5PQA==",
      "license": "MIT"
    },
    "node_modules/undefsafe": {
      "version": "2.0.5",
      "resolved": "https://registry.npmjs.org/undefsafe/-/undefsafe-2.0.5.tgz",
      "integrity": "sha512-
WxONCrssBM8TSPRqN5EmsjVrsv4A8X12J4ArBiiayv3DyyG3ZlIg6yysuuSYdZsVz3TKcTg2fd//
Ujd4CHV1iA==",
      "dev": true,
      "license": "MIT"
    },
```

```
    "node_modules/undici-types": {
      "version": "6.21.0",
      "resolved": "https://registry.npmjs.org/undici-types/-/undici-
types-6.21.0.tgz",
      "integrity": "sha512-iwDZqg0QAGrg9Rav5H4n0M64c3mkR59cJ6wQp+7C4nI0gsmExaedaYL
NO44eT4AtBBwjbTiGPMlt2Md0T9H9JQ==",
      "license": "MIT"
    },
    "node_modules/unicode-properties": {
      "version": "1.4.1",
      "resolved": "https://registry.npmjs.org/unicode-properties/-/unicode-
properties-1.4.1.tgz",
      "integrity": "sha512-CLjCCLQ6UuMxWnbIylkisbRj31qxHPAurvena/0iwSVbQ2G1VY5/
HjV0IRabOEbDHlzZlRdCrD4NhB0JtU40Pg==",
      "license": "MIT",
      "dependencies": {
        "base64-js": "^1.3.0",
        "unicode-trie": "^2.0.0"
      }
    },
    "node_modules/unicode-trie": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/unicode-trie/-/unicode-
trie-2.0.0.tgz",
      "integrity": "sha512-x7bc76x0bm4prf1VLg79uhAzKw8DVboClSN5VxJuQ+LKDOVEW9CdH+V
Y7SP+vX7xCYQqzzgQpFqz15zeLvAtZQ==",
      "license": "MIT",
      "dependencies": {
        "pako": "^0.2.5",
        "tiny-inflate": "^1.0.0"
      }
    },
    "node_modules/unpipe": {
      "version": "1.0.0",
      "resolved": "https://registry.npmjs.org/unpipe/-/unpipe-1.0.0.tgz",
      "integrity": "sha512-pjy2bYhSsufwWlKwPc+l3cN7+wuJlK6uz0YdJEOlQDbl6jo/
YlPi4mb8agUkVC8BF7V8NuzeyPNqRksA3hztKQ==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/util-deprecate": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/util-deprecate/-/util-
deprecate-1.0.2.tgz",
      "integrity": "sha512-
EPD5q1uXyFxJpCrLnCc1nHnq3gOa6DZBocAIiI2TaSCA7VCJ1UJDMagCzIkXNsUYfD1daK//
LTEQ8xiIbrHtcw==",
      "license": "MIT"
    },
    "node_modules/utils-merge": {
      "version": "1.0.1",
      "resolved": "https://registry.npmjs.org/utils-merge/-/utils-
merge-1.0.1.tgz",
      "integrity": "sha512-
pMZTvIkT1d+TFGvDOqodOclx0QWkkgi6Tdoa8gC8ffGAAqz9pzPTZWAybbsHHoED/ztMtkv/
VoYTYyShUn81hA==",
      "license": "MIT",
```

```
      "engines": {
        "node": ">= 0.4.0"
      }
    },
    "node_modules/uuid": {
      "version": "8.3.2",
      "resolved": "https://registry.npmjs.org/uuid/-/uuid-8.3.2.tgz",
      "integrity": "sha512-
+NYs2QeMWy+GWFOEm9xnn6HCDp0l7QBD7ml8zLUmJ+93Q5NF0NocErnwkTkXVFNiX3/fpC6afS8Dhb/
gz7R7eg==",
      "license": "MIT",
      "bin": {
        "uuid": "dist/bin/uuid"
      }
    },
    "node_modules/vary": {
      "version": "1.1.2",
      "resolved": "https://registry.npmjs.org/vary/-/vary-1.1.2.tgz",
      "integrity": "sha512-BNGbWLfd0eUPabhkXUVm0j8uuvREyTh5ovRa/dyow/
BqAbZJyC+5fU+IzQOzmAKzYqYRAISoRhdQr3eIZ/PXqg==",
      "license": "MIT",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/webidl-conversions": {
      "version": "7.0.0",
      "resolved": "https://registry.npmjs.org/webidl-conversions/-/webidl-
conversions-7.0.0.tgz",
      "integrity": "sha512-
VwddBukDzu71offAQR975unBIGqfKZpM+8ZX6ySk8nYhVoo5CYaZyzt3YBvYtRtO+aoGlqxPg/B87NGVZ/
fu6g==",
      "license": "BSD-2-Clause",
      "engines": {
        "node": ">=12"
      }
    },
    "node_modules/whatwg-url": {
      "version": "11.0.0",
      "resolved": "https://registry.npmjs.org/whatwg-url/-/whatwg-url-11.0.0.tgz",
      "integrity": "sha512-RKT8HExMpoYx4igMiVMY83lN6UeITKJlBQ+vR/8ZJ8OCdSiN3RwCq+9
gH0+Xzj0+5IrM6i4j/6LuvzbZIQgEcQ==",
      "license": "MIT",
      "dependencies": {
        "tr46": "^3.0.0",
        "webidl-conversions": "^7.0.0"
      },
      "engines": {
        "node": ">=12"
      }
    },
    "node_modules/wide-align": {
      "version": "1.1.5",
      "resolved": "https://registry.npmjs.org/wide-align/-/wide-align-1.1.5.tgz",
      "integrity": "sha512-eDMORYaPNZ4sQIuuYPDHdQvf4gyCF9rEEV/
yPxGfwPkRodwEgiMUUXTx/dex+Me0wxx53S+NgUHaP7y3MGlDmg==",
      "license": "ISC",
      "dependencies": {
        "string-width": "^1.0.2 || 2 || 3 || 4"
```

```
      }
    },
    "node_modules/wrappy": {
      "version": "1.0.2",
      "resolved": "https://registry.npmjs.org/wrappy/-/wrappy-1.0.2.tgz",
      "integrity": "sha512-l4Sp/DRseor9wL6EvV2+TuQn63dMkPjZ/
sp9XkghTEbV9KlPS1xUsZ3u7/IQO4wxtcFB4bgpQPRcR3QCvezPcQ==",
      "license": "ISC"
    },
    "node_modules/ws": {
      "version": "8.17.1",
      "resolved": "https://registry.npmjs.org/ws/-/ws-8.17.1.tgz",
      "integrity": "sha512-6XQFvXTkbfUOZOKKILFG1PDK2NDQs4azKQl26T0YS5CxqWLgXajbPZ+
h4gZekJyRqFU8pvnbAbbs/3TgRPy+GQ==",
      "license": "MIT",
      "engines": {
        "node": ">=10.0.0"
      },
      "peerDependencies": {
        "bufferutil": "^4.0.1",
        "utf-8-validate": ">=5.0.2"
      },
      "peerDependenciesMeta": {
        "bufferutil": {
          "optional": true
        },
        "utf-8-validate": {
          "optional": true
        }
      }
    },
    "node_modules/xtend": {
      "version": "4.0.2",
      "resolved": "https://registry.npmjs.org/xtend/-/xtend-4.0.2.tgz",
      "integrity": "sha512-LKYU1iAXJXUgAXn9URjiu+MWhyUXHsvfp7mcuYm9dSUKK0/
CjtrUwFAxD82/mCWbtLsGjFIad0wIsod4zrTAEQ==",
      "license": "MIT",
      "engines": {
        "node": ">=0.4"
      }
    },
    "node_modules/yallist": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/yallist/-/yallist-4.0.0.tgz",
      "integrity": "sha512-3wdGidZyq5PB084XLES5TpOSRA3wjXAlIWMhum2kRcv/41Sn2emQ0dy
cQW4uZXLejwKvg6EsvbdlVL+FYEct7A==",
      "license": "ISC"
    }
  }
}
```


package.json


```
{
  "name": "crm-server",
  "version": "1.0.0",
  "description": "Backend server for CRM application",
  "main": "server.js",
```

```
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "test": "echo \"Error: no test specified\" && exit 1",
    "postinstall": "npm rebuild bcrypt --build-from-source",
    "drop-indexes": "node utils/dropUniqueIndexes.js"
  },
  "dependencies": {
    "agenda": "^5.0.0",
    "axios": "^1.9.0",
    "bcrypt": "^5.1.1",
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "express-rate-limit": "^6.7.0",
    "jsonwebtoken": "^9.0.0",
    "lottie-react": "^2.4.1",
    "mongoose": "^7.0.3",
    "morgan": "^1.10.0",
    "multer": "^1.4.5-lts.1",
    "node-cron": "^3.0.3",
    "nodemailer": "^6.9.1",
    "pdfkit": "^0.17.1",
    "socket.io": "^4.8.1"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

PRODUCTION_DEPLOYMENT_GUIDE.md

```
# Ø=Þ€ Production Deployment Guide

## Ø=ÜË Document Storage in Production

### Current Status
- **Development**: Files stored locally in `./uploads/documents/`
- **Production**: Multiple options available (see below)

## Ø<ß¯ **RECOMMENDED FOR 15-20 EMPLOYEES: VPS Storage**

For your team size (15-20 employees), **VPS/Server storage** is the optimal
choice:

- **Monthly Cost**: $20-40 (vs $50-200 for cloud)
- **Storage Capacity**: 50GB+ (handles 2,000+ documents)
- **Setup Time**: 2-4 hours with our guide
- **Maintenance**: 1 hour/month

'¡Þ  **See `VPS_SETUP_GUIDE.md` for complete setup instructions**
'¡Þ  **See `TEAM_QUICK_REFERENCE.md` for your specific requirements**

## Ø<ß×Þ  Production Storage Options

### Option 1: AWS S3 (Recommended) +P
```

Best for: Scalable applications, multiple servers, high availability

```bash
# Environment Variables
STORAGE_TYPE=s3
AWS_S3_BUCKET=your-crm-documents-prod
AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=AKIA...
AWS_SECRET_ACCESS_KEY=your-secret-key
```

Advantages:
- ' Unlimited storage
- ' 99.999999999% (11 9's) durability
- ' Global CDN integration
- ' Automatic backups
- ' Pay-as-you-use pricing

Monthly Cost Estimate:
- Storage: ~$0.023/GB
- Requests: ~$0.40/1M requests
- Data transfer: ~$0.09/GB

Option 2: VPS/Dedicated Server +P **RECOMMENDED FOR YOUR TEAM**

Best for: Small to medium applications, cost control, 15-20 employees

```bash
# Environment Variables
STORAGE_TYPE=local
UPLOAD_PATH=/var/www/crm/uploads/documents
PUBLIC_PATH=/uploads/documents
```

Server Structure:
```
/var/www/crm/
% % %  app/                    # Application code
% % %  uploads/                # Document storage
%    % % %  documents/          # Employee documents
%    % % %  profile-pictures/   # Profile images
% % %  backups/               # Automated backups
%    % % %  daily/
%    % % %  weekly/
%    % % %  monthly/
% % %  logs/                  # Application logs
```

Advantages:
- ' Lower cost
- ' Full control
- ' No external dependencies

Disadvantages:
- 'L Manual backup required
- 'L Single point of failure
- 'L Limited scalability

Option 3: Google Cloud Storage

Best for: Google Cloud ecosystem, competitive pricing

```bash
# Environment Variables
STORAGE_TYPE=gcs
GCS_BUCKET=your-crm-documents-prod
GCS_PROJECT_ID=your-project-id
GCS_KEY_FILE=/path/to/service-account-key.json
```

Option 4: Azure Blob Storage

Best for: Microsoft ecosystem, enterprise integration

```bash
# Environment Variables
STORAGE_TYPE=azure
AZURE_STORAGE_CONNECTION_STRING=DefaultEndpointsProtocol=https;AccountName=...
```

Ø=Ý' Implementation Steps

1. Choose Storage Type

Based on your needs:
- **Startup/Small team**: VPS storage
- **Growing business**: AWS S3
- **Enterprise**: AWS S3 + CloudFront CDN

2. Environment Configuration

Create `.env.production`:
```bash
# Basic Configuration
NODE_ENV=production
BASE_URL=https://yourcrm.com

# Database
DB_URI=mongodb://username:password@cluster.mongodb.net/crm_production

# Storage (choose one)
STORAGE_TYPE=s3  # or 'local', 'gcs', 'azure'

# AWS S3 (if using S3)
AWS_S3_BUCKET=your-crm-documents
AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=your-key
AWS_SECRET_ACCESS_KEY=your-secret
```

3. Update Deployment Scripts

Docker Deployment:
```dockerfile
# Production Dockerfile
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
```

```
RUN npm ci --only=production
COPY . .
EXPOSE 8080

# For local storage, create volume
VOLUME ["/app/uploads"]

CMD ["npm", "start"]
```

Docker Compose:
```yaml
version: '3.8'
services:
  crm-app:
    build: .
    ports:
      - "8080:8080"
    environment:
      - NODE_ENV=production
    volumes:
      # Only needed for local storage
      - ./uploads:/app/uploads
    env_file:
      - .env.production
```

Ø=Þáþ Security Configuration

1. S3 Bucket Policy (Private Access)
```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyDirectAccess",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::your-crm-documents/*",
      "Condition": {
        "StringNotEquals": {
          "aws:sourceIp": "your-server-ip"
        }
      }
    }
  ]
}
```

2. Server Security (Local Storage)
```bash
# Set proper permissions
sudo chown -R www-data:www-data /var/www/crm/uploads/
sudo chmod -R 755 /var/www/crm/uploads/

# Create backup script
sudo crontab -e
# Add: 0 2 * * * /var/www/crm/scripts/backup.sh
```

```

## Ø=ÜÊ Cost Comparison (Monthly)

| Storage Type | 100GB | 500GB | 1TB | Bandwidth |
|--------------|-------|-------|-----|-----------|
| **AWS S3** | $2.30 | $11.50 | $23 | $0.09/GB |
| **Google Cloud** | $2.00 | $10.00 | $20 | $0.08/GB |
| **Azure Blob** | $2.40 | $12.00 | $24 | $0.087/GB |
| **VPS (2TB)** | $20-50 | $20-50 | $20-50 | Included |

## Ø=Ý  Migration Strategy

### From Development to Production

1. **Export Current Files**:
```bash
# Create archive of current uploads
tar -czf development-uploads.tar.gz uploads/
```

2. **Upload to Production Storage**:
```bash
# For S3
aws s3 sync uploads/ s3://your-crm-documents/

# For server
scp -r uploads/ user@server:/var/www/crm/
```

3. **Update Database Paths**:
```javascript
// Migration script
const updateFilePaths = async () => {
  const employees = await Employee.find({});
  for (let employee of employees) {
    if (employee.documents) {
      Object.keys(employee.documents).forEach(key => {
        if (employee.documents[key].path) {
          // Update path for new storage
          employee.documents[key].path = employee.documents[key].path.replace(
            'uploads/documents/',
            'documents/' // S3 prefix
          );
        }
      });
      await employee.save();
    }
  }
};
```

Ø=Þ¨ Backup Strategy

For S3 Storage
- ' Built-in versioning
- ' Cross-region replication
- ' Glacier archiving for old files
```

For Local Storage
```bash
#!/bin/bash
# backup.sh
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/var/backups/crm"
SOURCE_DIR="/var/www/crm/uploads"

# Create backup
tar -czf "$BACKUP_DIR/uploads_$DATE.tar.gz" "$SOURCE_DIR"

# Upload to S3 (optional)
aws s3 cp "$BACKUP_DIR/uploads_$DATE.tar.gz" s3://your-backup-bucket/

# Clean old backups (keep 30 days)
find "$BACKUP_DIR" -name "uploads_*.tar.gz" -mtime +30 -delete
```

Ø=ÜÈ Monitoring & Alerts

Storage Monitoring
```javascript
// Monitor storage usage
const getStorageStats = async () => {
  if (storageType === 's3') {
    // AWS CloudWatch metrics
    const params = {
      MetricName: 'BucketSizeBytes',
      Namespace: 'AWS/S3',
      StartTime: new Date(Date.now() - 24*60*60*1000),
      EndTime: new Date(),
      Period: 3600,
      Statistics: ['Average'],
      Dimensions: [
        {
          Name: 'BucketName',
          Value: process.env.AWS_S3_BUCKET
        }
      ]
    };
  } else {
    // Local storage check
    const { execSync } = require('child_process');
    const usage = execSync('df -h /var/www/crm/uploads').toString();
    console.log('Storage usage:', usage);
  }
};
```

Ø<ß¯ Recommended Production Setup

For Most Applications:
1. **Primary**: AWS S3 for document storage
2. **CDN**: CloudFront for fast global access
3. **Backup**: S3 Cross-Region Replication
4. **Monitoring**: CloudWatch + alerts
5. **Security**: Private buckets + signed URLs

Monthly Cost: ~$10-50 for typical usage

This setup provides enterprise-grade reliability, security, and scalability for your CRM document storage! Ø=Þ€

routes/activity.js

```
const express = require('express');
const router = express.Router();
const { protect, authorize } = require('../middleware/auth');
const UserActivity = require('../models/UserActivity');
const User = require('../models/User');
const jwt = require('jsonwebtoken');

// Special middleware for sendBeacon requests (no custom headers)
const protectBeacon = async (req, res, next) => {
  let token;

  // Try to get token from Authorization header first
  if (req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')) {
    token = req.headers.authorization.split(' ')[1];
  }
  // If no header, try to get from body (for sendBeacon)
  else if (req.body && req.body.token) {
    token = req.body.token;
  }

  // Make sure token exists
  if (!token) {
    return res.status(401).json({
      success: false,
      message: 'Not authorized to access this route'
    });
  }

  try {
    // Verify token
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Get user from token
    const user = await User.findById(decoded.id);

    if (!user) {
      return res.status(401).json({
        success: false,
        message: 'No user found with this token'
      });
    }

    req.user = user;
    next();
  } catch (err) {
    return res.status(401).json({
      success: false,
      message: 'Not authorized to access this route'
    });
  }
};
```

```
// Start a new activity session
router.post('/start-session', protect, async (req, res) => {
  try {
    const userId = req.user._id;

    // Get or create today's activity record
    const activity = await UserActivity.getTodaysActivity(userId);

    // End any existing active session first
    await activity.endCurrentSession();

    // Start new session
    await activity.startSession();

    res.json({
      success: true,
      message: 'Activity session started',
      data: {
        sessionStarted: true,
        totalActiveTime: activity.totalActiveTime
      }
    });
  } catch (error) {
    console.error('Error starting activity session:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to start activity session',
      error: error.message
    });
  }
});

// End current activity session
router.post('/end-session', protect, async (req, res) => {
  try {
    const userId = req.user._id;
    const { duration } = req.body; // Duration in seconds from frontend

    // Get today's activity record
    const activity = await UserActivity.getTodaysActivity(userId);

    // If there's an active session, end it
    const activeSession = activity.sessions.find(session => session.isActive);
    if (activeSession) {
      activeSession.endTime = new Date();

      // Use frontend duration if provided, otherwise calculate
      if (duration && duration > 0) {
        activeSession.duration = Math.floor(duration);
      } else {
        activeSession.duration = Math.floor((activeSession.endTime -
activeSession.startTime) / 1000);
      }

      activeSession.isActive = false;
      activity.totalActiveTime += activeSession.duration;
      activity.lastActivity = new Date();
```

```
      await activity.save();
    }

    res.json({
      success: true,
      message: 'Activity session ended',
      data: {
        sessionEnded: true,
        totalActiveTime: activity.totalActiveTime,
        sessionDuration: activeSession ? activeSession.duration : 0
      }
    });
  } catch (error) {
    console.error('Error ending activity session:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to end activity session',
      error: error.message
    });
  }
});

// Special endpoint for sendBeacon requests (handles token in body)
router.post('/end-session-beacon', protectBeacon, async (req, res) => {
  try {
    const userId = req.user._id;
    const { duration } = req.body; // Duration in seconds from frontend

    // Get today's activity record
    const activity = await UserActivity.getTodaysActivity(userId);

    // If there's an active session, end it
    const activeSession = activity.sessions.find(session => session.isActive);
    if (activeSession) {
      activeSession.endTime = new Date();

      // Use frontend duration if provided, otherwise calculate
      if (duration && duration > 0) {
        activeSession.duration = Math.floor(duration);
      } else {
        activeSession.duration = Math.floor((activeSession.endTime -
activeSession.startTime) / 1000);
      }

      activeSession.isActive = false;
      activity.totalActiveTime += activeSession.duration;
      activity.lastActivity = new Date();

      await activity.save();
    }

    res.json({
      success: true,
      message: 'Activity session ended via beacon',
      data: {
        sessionEnded: true,
        totalActiveTime: activity.totalActiveTime,
        sessionDuration: activeSession ? activeSession.duration : 0
      }
```

```
    });
  } catch (error) {
    console.error('Error ending activity session via beacon:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to end activity session via beacon',
      error: error.message
    });
  }
});

// Track activity (for periodic updates)
router.post('/track', protect, async (req, res) => {
  try {
    const userId = req.user._id;
    const { duration, isActive = true } = req.body;

    if (!duration || duration <= 0) {
      return res.status(400).json({
        success: false,
        message: 'Valid duration is required'
      });
    }

    // Get today's activity record
    const activity = await UserActivity.getTodaysActivity(userId);

    // Update last activity time
    activity.lastActivity = new Date();

    // If there's an active session, update it
    const activeSession = activity.sessions.find(session => session.isActive);
    if (activeSession && isActive) {
      // Session is still active, just update last activity
      await activity.save();
    } else if (!isActive && activeSession) {
      // Session should be ended
      await activity.endCurrentSession();
    }

    res.json({
      success: true,
      message: 'Activity tracked',
      data: {
        totalActiveTime: activity.totalActiveTime,
        isActive: !!activeSession
      }
    });
  } catch (error) {
    console.error('Error tracking activity:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to track activity',
      error: error.message
    });
  }
});

// Get user's own activity data
```

```
router.get('/my-activity', protect, async (req, res) => {
  try {
    const userId = req.user._id;
    const { date, startDate, endDate } = req.query;

    let activities;

    if (date) {
      // Get activity for specific date
      activities = await UserActivity.findOne({ userId, date });
    } else if (startDate && endDate) {
      // Get activity for date range
      activities = await UserActivity.getActivityByDateRange(userId, startDate,
endDate);
    } else {
      // Get today's activity
      const today = new Date().toISOString().split('T')[0];
      activities = await UserActivity.findOne({ userId, date: today });
    }

    res.json({
      success: true,
      data: activities
    });
  } catch (error) {
    console.error('Error fetching user activity:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to fetch activity data',
      error: error.message
    });
  }
});

// Get all users' activity data (Admin/Manager only)
router.get('/all-users', protect, authorize('Admin', 'Manager'), async (req, res)
=> {
  try {
    const { date, startDate, endDate } = req.query;
    const today = new Date().toISOString().split('T')[0];
    const targetDate = date || today;

    let activities;

    if (startDate && endDate) {
      // Get activities for date range
      activities = await UserActivity.find({
        date: { $gte: startDate, $lte: endDate }
      }).populate('userId', 'fullName email role').sort({ date: -1,
totalActiveTime: -1 });
    } else {
      // Get activities for specific date (default today)
      activities = await UserActivity.find({ date: targetDate })
        .populate('userId', 'fullName email role')
        .sort({ totalActiveTime: -1 });
    }

    // Format the response
    const formattedActivities = activities.map(activity => ({
```

```
      userId: activity.userId._id,
      userName: activity.userId.fullName,
      userEmail: activity.userId.email,
      userRole: activity.userId.role,
      date: activity.date,
      totalActiveTime: activity.totalActiveTime,
      totalActiveTimeFormatted: formatDuration(activity.totalActiveTime),
      sessionsCount: activity.sessions.length,
      lastActivity: activity.lastActivity,
      isCurrentlyActive: activity.sessions.some(session => session.isActive)
    }));

    res.json({
      success: true,
      data: formattedActivities,
      summary: {
        totalUsers: formattedActivities.length,
        activeUsers: formattedActivities.filter(a => a.isCurrentlyActive).length,
        totalActiveTime: formattedActivities.reduce((sum, a) => sum +
a.totalActiveTime, 0),
        averageActiveTime: formattedActivities.length > 0
          ? Math.floor(formattedActivities.reduce((sum, a) => sum +
a.totalActiveTime, 0) / formattedActivities.length)
          : 0
      }
    });
  } catch (error) {
    console.error('Error fetching all users activity:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to fetch users activity data',
      error: error.message
    });
  }
});

// Get activity statistics (Admin/Manager only)
router.get('/statistics', protect, authorize('Admin', 'Manager'), async (req,
res) => {
  try {
    const { days = 7 } = req.query;
    const endDate = new Date().toISOString().split('T')[0];
    const startDate = new Date(Date.now() - (days - 1) * 24 * 60 * 60 *
1000).toISOString().split('T')[0];

    // Get all activities in the date range
    const activities = await UserActivity.find({
      date: { $gte: startDate, $lte: endDate }
    }).populate('userId', 'fullName email role');

    // Group by date
    const dailyStats = {};
    const userStats = {};

    activities.forEach(activity => {
      // Daily statistics
      if (!dailyStats[activity.date]) {
        dailyStats[activity.date] = {
          date: activity.date,
```

```
          totalUsers: 0,
          totalActiveTime: 0,
          averageActiveTime: 0
        };
      }
      dailyStats[activity.date].totalUsers++;
      dailyStats[activity.date].totalActiveTime += activity.totalActiveTime;

      // User statistics
      const userId = activity.userId._id.toString();
      if (!userStats[userId]) {
        userStats[userId] = {
          userId: activity.userId._id,
          userName: activity.userId.fullName,
          userEmail: activity.userId.email,
          userRole: activity.userId.role,
          totalDays: 0,
          totalActiveTime: 0,
          averageActiveTime: 0
        };
      }
      userStats[userId].totalDays++;
      userStats[userId].totalActiveTime += activity.totalActiveTime;
    });

    // Calculate averages
    Object.values(dailyStats).forEach(day => {
      day.averageActiveTime = day.totalUsers > 0 ?
Math.floor(day.totalActiveTime / day.totalUsers) : 0;
      day.totalActiveTimeFormatted = formatDuration(day.totalActiveTime);
      day.averageActiveTimeFormatted = formatDuration(day.averageActiveTime);
    });

    Object.values(userStats).forEach(user => {
      user.averageActiveTime = user.totalDays > 0 ?
Math.floor(user.totalActiveTime / user.totalDays) : 0;
      user.totalActiveTimeFormatted = formatDuration(user.totalActiveTime);
      user.averageActiveTimeFormatted = formatDuration(user.averageActiveTime);
    });

    res.json({
      success: true,
      data: {
        dailyStats: Object.values(dailyStats).sort((a, b) =>
a.date.localeCompare(b.date)),
        userStats: Object.values(userStats).sort((a, b) => b.totalActiveTime -
a.totalActiveTime),
        period: {
          startDate,
          endDate,
          days: parseInt(days)
        }
      }
    });
  } catch (error) {
    console.error('Error fetching activity statistics:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to fetch activity statistics',
```

```
      error: error.message
    });
  }
});

// Helper function to format duration
function formatDuration(seconds) {
  if (!seconds || seconds < 0) return '0m';

  const hours = Math.floor(seconds / 3600);
  const minutes = Math.floor((seconds % 3600) / 60);

  if (hours > 0) {
    return `${hours}h ${minutes}m`;
  } else {
    return `${minutes}m`;
  }
}

module.exports = router;
```

routes/attendance.js

```
const express = require('express');
const {
  checkIn,
  checkOut,
  getTodayAttendance,
  getAttendanceHistory,
  getAllAttendance,
  updateAttendance,
  getMonthlyAttendanceSummary
} = require('../controllers/attendance');

const router = express.Router();
const { protect } = require('../middleware/auth');

// Protect all routes
router.use(protect);

// Employee attendance routes
router.post('/checkin', checkIn);
router.put('/checkout', checkOut);
router.get('/today', getTodayAttendance);
router.get('/history', getAttendanceHistory);
router.get('/summary/:month/:year', getMonthlyAttendanceSummary);

// Admin/HR/Manager routes
router.get('/all', getAllAttendance);
router.put('/:id', updateAttendance);

module.exports = router;
```

routes/auth.js

```
const express = require('express');
const router = express.Router();
const { register, login, getMe, getAllUsers, updateUser, deleteUser,
```

```
updateProfilePicture, createUser, createUserWithDocuments,
updateUserWithDocuments } = require('../controllers/auth');
const path = require('path');
const fs = require('fs');
const { protect, authorize } = require('../middleware/auth');
const nodemailer = require('nodemailer');
const bcrypt = require('bcrypt');
const UserModel = require('../models/User.js');
const { upload, storageType, getFileUrl, deleteFile } = require('../config/
storage');

// Document upload middleware using centralized storage config
const uploadDocuments = upload.fields([
  { name: 'photograph', maxCount: 1 },
  { name: 'tenthMarksheet', maxCount: 1 },
  { name: 'twelfthMarksheet', maxCount: 1 },
  { name: 'bachelorDegree', maxCount: 1 },
  { name: 'postgraduateDegree', maxCount: 1 },
  { name: 'aadharCard', maxCount: 1 },
  { name: 'panCard', maxCount: 1 },
  { name: 'pcc', maxCount: 1 },
  { name: 'resume', maxCount: 1 },
  { name: 'offerLetter', maxCount: 1 }
]);

// Profile picture upload middleware using centralized storage config
const uploadProfilePicture = upload.fields([
  { name: 'profilePicture', maxCount: 1 }
]);

// Register all routes
console.log('Registering auth routes...');

router.post('/register', register);
console.log('POST /api/auth/register registered');

router.post('/login', login);
console.log('POST /api/auth/login registered');

router.get('/me', protect, getMe);
console.log('GET /api/auth/me registered');

router.get('/users', protect, getAllUsers);
console.log('GET /api/auth/users registered');

router.post('/users', protect, authorize('Admin', 'Manager'), createUser);
console.log('POST /api/auth/users registered');

router.post('/users/with-documents', protect, authorize('Admin', 'Manager'),
uploadDocuments, createUserWithDocuments);
console.log('POST /api/auth/users/with-documents registered');

router.put('/users/:id', protect, authorize('Admin', 'Manager'), updateUser);
console.log('PUT /api/auth/users/:id registered');

// Handle document uploads optionally for user updates
router.put('/users/:id/with-documents', protect, authorize('Admin', 'Manager'),
(req, res, next) => {
  // Use uploadDocuments middleware but handle errors gracefully
```

```
  uploadDocuments(req, res, (err) => {
    if (err) {
      console.log('File upload error (continuing without files):', err.message);
      // Continue without files if upload fails
      req.files = {};
    }
    next();
  });
}, updateUserWithDocuments);
console.log('PUT /api/auth/users/:id/with-documents registered');

// Serve documents with proper authentication and storage-agnostic support
router.get('/documents/:filename', protect, async (req, res) => {
  try {
    const { filename } = req.params;

    if (storageType === 's3') {
      // For S3 storage, generate signed URL for secure access
      const AWS = require('aws-sdk');
      const s3 = new AWS.S3({
        accessKeyId: process.env.AWS_ACCESS_KEY_ID,
        secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
        region: process.env.AWS_REGION
      });

      const params = {
        Bucket: process.env.AWS_S3_BUCKET,
        Key: `documents/${filename}`,
        Expires: 3600 // 1 hour expiry
      };

      try {
        const signedUrl = s3.getSignedUrl('getObject', params);
        return res.redirect(signedUrl);
      } catch (s3Error) {
        console.error('S3 signed URL error:', s3Error);
        return res.status(404).json({
          success: false,
          message: 'Document not found'
        });
      }
    } else {
      // For local storage, serve files directly
      const { currentConfig } = require('../config/storage');
      const filePath = path.join(currentConfig.destination, filename);

      // Check if file exists
      if (!fs.existsSync(filePath)) {
        return res.status(404).json({
          success: false,
          message: 'Document not found'
        });
      }

      // Get file stats
      const stats = fs.statSync(filePath);

      // Set appropriate headers based on file extension
      const ext = path.extname(filename).toLowerCase();
```

```
      let contentType = 'application/octet-stream';

      switch (ext) {
        case '.pdf':
          contentType = 'application/pdf';
          break;
        case '.jpg':
        case '.jpeg':
          contentType = 'image/jpeg';
          break;
        case '.png':
          contentType = 'image/png';
          break;
        case '.gif':
          contentType = 'image/gif';
          break;
        case '.doc':
          contentType = 'application/msword';
          break;
        case '.docx':
          contentType = 'application/vnd.openxmlformats-
officedocument.wordprocessingml.document';
          break;
        default:
          contentType = 'application/octet-stream';
      }

      res.setHeader('Content-Type', contentType);
      res.setHeader('Content-Length', stats.size);
      res.setHeader('Content-Disposition', `inline; filename="${filename}"`);
      res.setHeader('Cache-Control', 'private, no-cache, no-store, must-
revalidate');
      res.setHeader('Expires', '-1');
      res.setHeader('Pragma', 'no-cache');

      // Stream the file
      const fileStream = fs.createReadStream(filePath);

      fileStream.on('error', (error) => {
        console.error('File stream error:', error);
        if (!res.headersSent) {
          res.status(500).json({
            success: false,
            message: 'Error reading file'
          });
        }
      });

      fileStream.pipe(res);
    }

  } catch (error) {
    console.error('Error serving document:', error);
    if (!res.headersSent) {
      res.status(500).json({
        success: false,
        message: 'Error serving document'
      });
    }
```

```
  }
});
console.log('GET /api/auth/documents/:filename registered');

router.delete('/users/:id', protect, authorize('Admin', 'Manager'), deleteUser);
console.log('DELETE /api/auth/users/:id registered');

// Add profile picture update route
router.put('/profile-picture', protect, uploadProfilePicture,
updateProfilePicture);
console.log('PUT /api/auth/profile-picture registered');

router.post("/sendOTPToEmail", async (req, res) => {
  console.log("Received request to sendOTPToEmail:", req.body);

  // Log environment variables for debugging (without leaking secrets)
  console.log("Email config:", {
    emailUser: process.env.EMAIL_USER ? `${process.env.EMAIL_USER.substring(0,
5)}...` : 'undefined',
    emailPassSet: process.env.EMAIL_PASS ? 'Yes' : 'No'
  });

  // Check if email configuration is set
  if (!process.env.EMAIL_USER || !process.env.EMAIL_PASS) {
    console.error("Email configuration missing:", {
      hasEmailUser: !!process.env.EMAIL_USER,
      hasEmailPass: !!process.env.EMAIL_PASS
    });
    return res.status(500).json({
      success: false,
      message: "Email server configuration is missing. Please contact support.",
      error: "Missing email configuration"
    });
  }

  const transporter = nodemailer.createTransport({
    host: "smtp.hostinger.com",
    port: 465,
    secure: true,
    auth: {
      user: process.env.EMAIL_USER,
      pass: process.env.EMAIL_PASS
    }
  });

  // Verify transporter configuration
  try {
    await transporter.verify();
    console.log("Email transporter verified successfully");
  } catch (error) {
    console.error("Email transporter verification failed:", error);
    return res.status(500).json({
      success: false,
      message: "Email server configuration is invalid. Please contact support.",
      error: error.message
    });
  }

  const { email } = req.body;
```

```
  if (!email) {
    console.log("Email not provided in request");
    return res.status(400).json({ success: false, message: "Email is required" });
  }

  try {
    console.log(`Looking for user with email: ${email}`);
    const user = await UserModel.findOne({ email });

    if (!user) {
      console.log(`User with email ${email} not found`);
      return res.status(400).json({
        success: false,
        message: "Email ID does not exist in the database"
      });
    }

    console.log(`User found, generating OTP for ${email}`);
    const otp = String(Math.floor(100000 + Math.random() * 900000));
    user.verifyOtp = otp;
    user.verifyOtpExpireAt = Date.now() + 10 * 60 * 1000; // 10 minutes expiry

    await user.save();
    console.log(`OTP saved to user database: ${otp}`);

    const mailOptions = {
      from: process.env.EMAIL_USER,
      to: email,
      subject: "Password Reset OTP",
      html: `
      <div style="font-family: Arial, sans-serif; text-align: center; padding:
20px; background-color: #f4f4f4;">
        <div style="max-width: 600px; margin: auto; background: #fff; padding:
20px; border-radius: 10px; border: 1px solid #ddd;">
          <h2 style="color: #333;">OTP Verification</h2>
          <p style="color: #555; font-size: 16px;">Your One-Time Password (OTP)
for verification is:</p>
          <div style="font-size: 24px; font-weight: bold; color: #333; padding:
10px 20px; background: #f8f8f8; border: 1px dashed #333; display: inline-block;
margin: 10px 0;">
            ${otp}
          </div>
          <p style="color: #777; font-size: 14px;">This OTP is valid for 10
minutes. Do not share it with anyone.</p>
          <p style="color: #777; font-size: 14px;">If you did not request this,
please ignore this email.</p>
          <div style="font-size: 12px; color: #aaa; margin-top: 20px;">© 2025
TrainCape Industries</div>
        </div>
      </div>
      `
    };

    console.log("Attempting to send email now...");

    // Send email
    const info = await transporter.sendMail(mailOptions);
    console.log("Email sent successfully:", info.messageId);
```

```
    return res.json({ success: true, message: "OTP sent successfully" });

  } catch (error) {
    console.error("Error in sendOTPToEmail:", {
      name: error.name,
      message: error.message,
      stack: error.stack,
      code: error.code
    });

    // Provide more specific error messages
    if (error.code === 'EAUTH') {
      return res.status(500).json({
        success: false,
        message: "Email authentication failed. Please contact support.",
        error: "Invalid email credentials"
      });
    }

    if (error.code === 'ESOCKET') {
      return res.status(500).json({
        success: false,
        message: "Could not connect to email server. Please try again later.",
        error: "Connection error"
      });
    }

    return res.status(500).json({
      success: false,
      message: "Failed to send OTP. Please try again later.",
      error: error.message
    });
  }
});

router.post("/verifyOtp", async (req, res) => {
  const { otp, email } = req.body;
  console.log("req.body", req.body);
  console.log("otp", otp);
  console.log("email", email);
  try {
    const user = await UserModel.findOne({ email });
    if (!user) {
      return res.status(400).send({ msg: "Wrong Credentials" });
    }
    if (user.verifyOtp !== otp || user.verifyOtp === "") {
      return res.json({ success: false, message: "Invalid OTP" });
    }
    if (user.verifyOtpExpireAt < Date.now()) {
      return res.json({ success: false, message: "OTP expired" });
    }
    user.verifyOtp = "";
    user.verifyOTPExpireAt = 0;
    await user.save();
    return res.json({ success: true, message: "Email verified successfully" });
  } catch (error) {
    console.error(error);
    return res.json({ success: false, message: error.message });
  }
```

```
});

router.post("/reset_password", async (req, res) => {
  const { email, newPassword } = req.body;
  try {
    const user = await UserModel.findOne({ email });
    if (!user) {
      return res.status(400).send({ msg: "Wrong Credentials" });
    }

    // Don't hash the password here - let the pre-save hook handle it
    // This prevents double-hashing which causes login failures
    console.log("Setting new password for user:", user._id);
    user.password = newPassword;
    user.resetOtp = "";
    user.resetOtpExpireAt = 0;

    await user.save();
    console.log("Password reset successful for user:", user._id);

    return res.json({
      success: true,
      message: "Password has been changed Successfully",
    });
  } catch (error) {
    console.error("Password reset error:", error);
    return res.json({ success: false, message: error.message });
  }
});




// Debug route to check token
router.get('/debug', protect, (req, res) => {
  res.status(200).json({
    success: true,
    message: 'Token is valid',
    user: req.user
  });
});
console.log('GET /api/auth/debug registered');

// Test route for profile picture update
router.get('/profile-picture-test', (req, res) => {
  res.status(200).json({
    success: true,
    message: 'Profile picture endpoint is available'
  });
});
console.log('GET /api/auth/profile-picture-test registered');

module.exports = router;
```

routes/chat.js

```
const express = require('express');
const router = express.Router();
```

```
const {
  sendMessage,
  getChatMessages,
  getChatRooms,
  getOnlineUsers,
  getAllUsersForChat,
  updateChatStatus,
  markMessagesAsRead
} = require('../controllers/chatController');

const { protect } = require('../middleware/auth');

// All routes require authentication
router.use(protect);

// Message routes
router.post('/messages', sendMessage);
router.get('/messages/:recipientId', getChatMessages);
router.put('/messages/read/:senderId', markMessagesAsRead);

// Chat room routes
router.get('/rooms', getChatRooms);

// User routes
router.get('/users', getAllUsersForChat);
router.get('/users/online', getOnlineUsers);
router.put('/status', updateChatStatus);

module.exports = router;
```

routes/currency.js

```
const express = require('express');
const router = express.Router();
const { getRates } = require('../controllers/currency');

// @route   GET /api/currency/rates
// @desc    Get latest exchange rates
// @access  Public
router.get('/rates', getRates);

// Get specific exchange rate
router.get('/rate', async (req, res) => {
  try {
    const { from = 'USD', to = 'USD' } = req.query;

    // Get current rates from the controller
    const mockReq = {};
    const mockRes = {
      json: (data) => data
    };

    const { getRates } = require('../controllers/currency');
    const ratesData = await new Promise((resolve) => {
      const res = {
        json: (data) => resolve(data)
      };
      getRates(mockReq, res);
```

```
    });

    const rates = ratesData.rates || {};
    const fromRate = rates[from] || 1;
    const toRate = rates[to] || 1;
    const rate = toRate / fromRate;

    res.json({
      success: true,
      data: {
        from,
        to,
        rate: Math.round(rate * 10000) / 10000,
        timestamp: new Date()
      }
    });
  } catch (error) {
    console.error('Error calculating exchange rate:', error);
    res.status(500).json({
      success: false,
      message: 'Error calculating exchange rate'
    });
  }
});

module.exports = router;
```

routes/documentation.js

```
const express = require('express');
const router = express.Router();
const { generateProjectDocumentation } = require('../controllers/documentation');
const { protect, authorize } = require('../middleware/auth');

// @route   GET /api/documentation/project
// @desc    Generate project documentation PDF
// @access  Private (Admin/HR/Manager only)
router.get('/project', protect, authorize('Admin', 'HR', 'Manager'),
generateProjectDocumentation);

module.exports = router;
```

routes/employees.js

```
const express = require('express');
const router = express.Router();
const { protect } = require('../middleware/auth');
const {
  getEmployees,
  getEmployee,
  createEmployee,
  updateEmployee,
  deleteEmployee,
  getDepartments,
  getRoles,
  uploadEmployeeFiles,
  getDocument
} = require('../controllers/employees');
```

```
// Employee routes
router.get('/', protect, getEmployees);
router.get('/:id', protect, getEmployee);
router.post('/', protect, uploadEmployeeFiles, createEmployee);
router.put('/:id', protect, uploadEmployeeFiles, updateEmployee);
router.delete('/:id', protect, deleteEmployee);

// Department and Role routes
router.get('/departments', protect, getDepartments);
router.get('/roles', protect, getRoles);

// Document routes
router.get('/documents/:filename', protect, getDocument);

module.exports = router;
```

routes/incentives.js

```
const express = require('express');
const {
  createIncentive,
  getIncentives,
  getIncentive,
  updateIncentive,
  approveIncentive,
  rejectIncentive,
  addComment,
  deleteIncentive,
  getIncentiveStats,
  uploadIncentiveFiles
} = require('../controllers/incentives');

const router = express.Router();
const { protect } = require('../middleware/auth');

// Protect all routes
router.use(protect);

// Incentive management routes
router.get('/stats', getIncentiveStats);
router.post('/', uploadIncentiveFiles, createIncentive);
router.get('/', getIncentives);
router.get('/:id', getIncentive);
router.put('/:id', updateIncentive);
router.delete('/:id', deleteIncentive);

// Incentive approval routes
router.put('/:id/approve', approveIncentive);
router.put('/:id/reject', rejectIncentive);

// Comment routes
router.post('/:id/comments', addComment);

module.exports = router;
```

routes/leadPersonSales.js

```
const express = require('express');
const router = express.Router();
const {
  getLeadPersonSales,
  getLeadPersonSale,
  createLeadPersonSale,
  updateLeadPersonSale,
  deleteLeadPersonSale
} = require('../controllers/leadPersonSales');

const { protect, authorize } = require('../middleware/auth');

// All routes below this line require authentication
router.use(protect);

// Routes specific to roles
router.route('/')
  .get(authorize('Lead Person', 'Manager', 'Admin'), getLeadPersonSales)
  .post(authorize('Lead Person', 'Manager', 'Admin'), createLeadPersonSale);

router.route('/:id')
  .get(authorize('Lead Person', 'Manager', 'Admin'), getLeadPersonSale)
  .put(authorize('Lead Person', 'Manager', 'Admin'), updateLeadPersonSale)
  .delete(authorize('Lead Person', 'Manager', 'Admin'), deleteLeadPersonSale);

module.exports = router;
```

routes/leads.js

```
const express = require('express');
const router = express.Router();
const {
  getLeads,
  getLead,
  createLead,
  updateLead,
  deleteLead,
  updateFeedback,
  getAssignedLeads,
  importLeads,
  getAllCustomers,
  getRepeatCustomers
} = require('../controllers/leads');

const { protect, authorize } = require('../middleware/auth');

// All routes below this line require authentication
router.use(protect);

// Routes specific to roles
router.route('/')
  .get(authorize('Lead Person', 'Sales Person', 'Manager', 'Admin'), getLeads)
  .post(authorize('Lead Person', 'Sales Person', 'Manager', 'Admin'), createLead);

// Import route (Admin, Manager, Lead Person)
router.post('/import', authorize('Admin', 'Manager', 'Lead Person'), importLeads);

// Repeat customers route (Admin/Manager only)
```

```
router.get('/repeat-customers', authorize('Admin', 'Manager'),
getRepeatCustomers);

// The '/assigned' route must come BEFORE the '/:id' route
router.get('/assigned', authorize('Sales Person'), getAssignedLeads);
router.get('/customers', authorize('Sales Person'), getAllCustomers);

// Rajesh duplicate checking routes - must come before /:id route
router.get('/check-rajesh-duplicates', authorize('Admin', 'Manager'), async (req,
res) => {
  try {
    console.log('=== CHECKING RAJESH DUPLICATES ===');

    // Find Rajesh
    const User = require('../models/User');
    const Lead = require('../models/Lead');
    const rajesh = await User.findOne({ fullName: /rajesh/i });
    if (!rajesh) {
      return res.status(404).json({
        success: false,
        message: 'Rajesh not found'
      });
    }

    console.log('' Found Rajesh:', rajesh.fullName, rajesh._id);

    // Get all leads assigned to Rajesh
    const leads = await Lead.find({ assignedTo: rajesh._id }).sort({ createdAt:
1 });

    console.log(`Total leads assigned to Rajesh: ${leads.length}`);

    // Group by month/year
    const leadsByMonth = {};
    leads.forEach(lead => {
      const date = new Date(lead.createdAt);
      const monthKey = `${date.getFullYear()}-${String(date.getMonth() +
1).padStart(2, '0')}`;

      if (!leadsByMonth[monthKey]) {
        leadsByMonth[monthKey] = [];
      }
      leadsByMonth[monthKey].push({
        id: lead._id,
        name: lead.name,
        phone: lead.phone,
        course: lead.course,
        createdAt: lead.createdAt
      });
    });

    console.log('Ø=ÜÊ Leads by month:');
    Object.keys(leadsByMonth).sort().forEach(month => {
      console.log(`${month}: ${leadsByMonth[month].length} leads`);
    });

    // Check for September 2024 and June 2025 leads
    const sep2024 = leadsByMonth['2024-09'] || [];
    const jun2025 = leadsByMonth['2025-06'] || [];
```

```
    // Look for potential duplicates (same name or phone)
    const duplicates = [];
    if (sep2024.length > 0 && jun2025.length > 0) {
      sep2024.forEach(sepLead => {
        jun2025.forEach(junLead => {
          if (sepLead.name === junLead.name || sepLead.phone === junLead.phone) {
            duplicates.push({
              sep2024: sepLead,
              jun2025: junLead
            });
          }
        });
      });
    }

    res.status(200).json({
      success: true,
      data: {
        rajesh: {
          id: rajesh._id,
          name: rajesh.fullName
        },
        totalLeads: leads.length,
        leadsByMonth: Object.keys(leadsByMonth).sort().map(month => ({
          month,
          count: leadsByMonth[month].length
        })),
        sep2024Leads: sep2024,
        jun2025Leads: jun2025,
        duplicates: duplicates,
        duplicateCount: duplicates.length
      }
    });

  } catch (error) {
    console.error('Error checking duplicates:', error);
    res.status(500).json({
      success: false,
      message: error.message
    });
  }
});

router.delete('/remove-rajesh-duplicates', authorize('Admin', 'Manager'), async
(req, res) => {
  try {
    console.log('=== REMOVING RAJESH DUPLICATES ===');

    // Find Rajesh
    const User = require('../models/User');
    const Lead = require('../models/Lead');
    const rajesh = await User.findOne({ fullName: /rajesh/i });
    if (!rajesh) {
      return res.status(404).json({
        success: false,
        message: 'Rajesh not found'
      });
    }
```

```
    // Get September 2024 and June 2025 leads
    const sep2024Start = new Date('2024-09-01');
    const sep2024End = new Date('2024-09-30T23:59:59.999Z');
    const jun2025Start = new Date('2025-06-01');
    const jun2025End = new Date('2025-06-30T23:59:59.999Z');

    const sep2024Leads = await Lead.find({
      assignedTo: rajesh._id,
      createdAt: { $gte: sep2024Start, $lte: sep2024End }
    });

    const jun2025Leads = await Lead.find({
      assignedTo: rajesh._id,
      createdAt: { $gte: jun2025Start, $lte: jun2025End }
    });

    console.log(`Found ${sep2024Leads.length} September 2024 leads`);
    console.log(`Found ${jun2025Leads.length} June 2025 leads`);

    // Find duplicates to remove from June 2025
    const duplicatesToRemove = [];
    sep2024Leads.forEach(sepLead => {
      jun2025Leads.forEach(junLead => {
        if (sepLead.name === junLead.name || sepLead.phone === junLead.phone) {
          duplicatesToRemove.push(junLead._id);
        }
      });
    });

    if (duplicatesToRemove.length === 0) {
      return res.status(200).json({
        success: true,
        message: 'No duplicates found to remove',
        removedCount: 0
      });
    }

    console.log(`Removing ${duplicatesToRemove.length} duplicate leads from June
2025`);

    // Remove the duplicate leads from June 2025
    const result = await Lead.deleteMany({
      _id: { $in: duplicatesToRemove }
    });

    console.log(`Successfully removed ${result.deletedCount} duplicate leads`);

    res.status(200).json({
      success: true,
      message: `Successfully removed ${result.deletedCount} duplicate leads from
June 2025`,
      removedCount: result.deletedCount,
      removedIds: duplicatesToRemove
    });

  } catch (error) {
    console.error('Error removing duplicates:', error);
    res.status(500).json({
```

```
      success: false,
      message: error.message
    });
  }
});

router.route('/:id')
  .get(authorize('Lead Person','Sales Person', 'Manager', 'Admin'), getLead)
  .put(authorize('Lead Person', 'Manager', 'Admin', 'Sales Person'), updateLead)
  .delete(authorize('Sales Person', 'Manager', 'Admin'), deleteLead);

router.put('/:id/feedback', authorize('Sales Person', 'Lead Person', 'Manager',
'Admin'), updateFeedback);

module.exports = router;
```

routes/leadSalesRoute.js

```
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');
const { protect, authorize } = require('../middleware/auth');
const Sale = require('../models/Sale');

// All routes below this line require authentication
router.use(protect);

// @route   GET /api/lead-sales
// @desc    Get sales sheet data for lead persons
// @access  Private (Lead Person, Manager, Admin)
router.get('/', authorize('Lead Person', 'Manager', 'Admin'), async (req, res) =>
{
  try {
    console.log('============= LEAD SALES REQUEST =============');
    console.log('User making request:', {
      id: req.user.id,
      _id: req.user._id ? req.user._id.toString() : 'undefined',
      role: req.user.role,
      name: req.user.fullName
    });

    let salesQuery = {};

    if (req.user.role === 'Lead Person') {
      const leadPersonId = req.user._id.toString();
      console.log('Filtering for lead person ID:', leadPersonId);

      // Find both lead person sales AND regular sales where this person is
assigned as lead
      salesQuery = {
        $or: [
          { isLeadPersonSale: true, leadPerson: new
mongoose.Types.ObjectId(leadPersonId) },
          { leadPerson: new mongoose.Types.ObjectId(leadPersonId) }
        ]
      };
    } else {
      // For Manager and Admin, show all lead-related sales
```

```
      salesQuery = {
        $or: [
          { isLeadPersonSale: true },
          { leadPerson: { $exists: true, $ne: null } }
        ]
      };
    }

    console.log('Sales query:', JSON.stringify(salesQuery));

    // Get sales based on the query
    const allSales = await Sale.find(salesQuery)
      .select('date customerName country course countryCode contactNumber email
pseudoId salesPerson leadPerson source clientRemark feedback totalCost
totalCostCurrency tokenAmount tokenAmountCurrency isLeadPersonSale')
      .populate('salesPerson', 'fullName')
      .populate('leadPerson', 'fullName')
      .sort({ date: -1 });

    console.log(`Found ${allSales.length} total sales records`);

    // Process the sales data
    const processedSales = allSales.map(sale => {
      const saleObj = sale.toObject();

      // Set default currency values if not present
      if (!saleObj.totalCostCurrency) {
        saleObj.totalCostCurrency = 'USD';
      }
      if (!saleObj.tokenAmountCurrency) {
        saleObj.tokenAmountCurrency = 'USD';
      }

      // Add a type field to distinguish between lead person sales and regular
sales
      saleObj.saleType = saleObj.isLeadPersonSale ? 'Lead Person Sale' : 'Sales
Person Sale';

      return saleObj;
    });

    console.log('============= LEAD SALES RESPONSE =============');
    console.log(`Returning ${processedSales.length} sales records`);

    res.status(200).json({
      success: true,
      count: processedSales.length,
      data: processedSales
    });
  } catch (err) {
    console.error('Error in lead sales route:', err);
    console.error('Stack trace:', err.stack);

    res.status(500).json({
      success: false,
      message: 'Server error while fetching sales data',
      error: err.message
    });
  }
```

```
});

module.exports = router;
```

routes/leaves.js

```
const express = require('express');
const router = express.Router();
const {
  applyLeave,
  getAllLeaves,
  getMyLeaves,
  updateLeaveStatus,
  cancelLeave,
  getLeaveStats,
  getLeaveBalance
} = require('../controllers/leaves');
const { protect, authorize } = require('../middleware/auth');

// Apply for leave
router.post('/', protect, applyLeave);

// Get my leaves
router.get('/my-leaves', protect, getMyLeaves);

// Get leave balance
router.get('/balance', protect, getLeaveBalance);

// Get leave statistics
router.get('/stats', protect, getLeaveStats);

// Get all leaves (for managers/admins)
router.get('/', protect, authorize('Admin', 'Manager'), getAllLeaves);

// Update leave status (approve/reject)
router.put('/:id/status', protect, authorize('Admin', 'Manager'),
updateLeaveStatus);

// Cancel leave
router.put('/:id/cancel', protect, cancelLeave);

module.exports = router;
```

routes/logs.js

```
const express = require('express');
const router = express.Router();
const { protect, authorize } = require('../middleware/auth');
const {
  createLog,
  getLogs,
  getLogStats,
  getLogsByResource,
  cleanupOldLogs
} = require('../controllers/logController');

// All routes need authentication
router.use(protect);
```

```
// Create log route - accessible to all authenticated users
router.post('/', createLog);

// Admin only routes
router.use(authorize('Admin'));

// These routes are admin-only
router.get('/', getLogs);
router.get('/stats', getLogStats);
router.get('/resource/:resourceId', getLogsByResource);
router.delete('/cleanup', cleanupOldLogs);

module.exports = router;
```

routes/payroll.js

```
const express = require('express');
const {
  generatePayroll,
  getPayroll,
  updatePayroll,
  deletePayroll,
  generateSalarySlip,
  downloadSalarySlip,
  approvePayroll
} = require('../controllers/payroll');

const router = express.Router();
const { protect } = require('../middleware/auth');

// Protect all routes
router.use(protect);

// Payroll management routes
router.post('/generate', generatePayroll);
router.get('/', getPayroll);
router.put('/:id', updatePayroll);
router.delete('/:id', deletePayroll);
router.put('/:id/approve', approvePayroll);

// Salary slip routes
router.get('/:id/salary-slip', generateSalarySlip);
router.get('/:id/download', downloadSalarySlip);

module.exports = router;
```

routes/prospects.js

```
const express = require('express');
const router = express.Router();
const {
  getProspects,
  getProspectById,
  createProspect,
  updateProspect,
  deleteProspect,
  convertToLead,
```

```
  getProspectStats
} = require('../controllers/prospectController');
const { protect, authorize } = require('../middleware/auth');

// Apply authentication to all prospect routes
router.use(protect);

// Apply role-based authorization (only Sales Person, Manager, Admin)
router.use(authorize('Sales Person', 'Manager', 'Admin'));

// GET /api/prospects - Get all prospects with filtering
router.get('/', getProspects);

// GET /api/prospects/stats - Get prospect statistics
router.get('/stats', getProspectStats);

// GET /api/prospects/:id - Get single prospect
router.get('/:id', getProspectById);

// POST /api/prospects - Create new prospect
router.post('/', createProspect);

// PUT /api/prospects/:id - Update prospect
router.put('/:id', updateProspect);

// DELETE /api/prospects/:id - Delete prospect (Admin/Manager only)
router.delete('/:id', deleteProspect);

// POST /api/prospects/:id/convert - Convert prospect to lead
router.post('/:id/convert', convertToLead);

module.exports = router;
```

routes/sales.js

```
const express = require('express');
const router = express.Router();
const {
  getSales,
  getSale,
  createSale,
  updateSale,
  deleteSale,
  getSalesCount,
  importSales
} = require('../controllers/sales');

const { protect, authorize } = require('../middleware/auth');
const Sale = require('../models/Sale');

// All routes below this line require authentication
router.use(protect);

// Routes specific to roles
router.route('/')
  .get(authorize('Sales Person', 'Lead Person', 'Manager', 'Admin'), getSales)
  .post(authorize('Sales Person', 'Lead Person', 'Manager', 'Admin'), createSale);
```

```
// Import route (Admin only)
router.post('/import', authorize('Admin'), importSales);

// Count route
router.get('/count', authorize('Sales Person', 'Lead Person', 'Manager',
'Admin'), getSalesCount);

router.route('/:id')
  .get(authorize('Sales Person', 'Lead Person', 'Manager', 'Admin'), getSale)
  .put(authorize('Sales Person', 'Lead Person', 'Manager', 'Admin'), updateSale)
  .delete(authorize('Sales Person','Manager', 'Admin'), deleteSale);

// Routes for token and pending amount updates - TODO: Implement these functions
// router.route('/:id/token')
//   .put(authorize('Sales Person', 'Lead Person', 'Manager', 'Admin'),
updateToken);

// router.route('/:id/pending')
//   .put(authorize('Sales Person', 'Lead Person', 'Manager', 'Admin'),
updatePending);

// @route   GET /api/sales/lead-sheet
// @desc    Get sales sheet data for lead persons
// @access  Private (Lead Person, Manager, Admin)
router.get('/lead-sheet', authorize('Lead Person', 'Manager', 'Admin'), async
(req, res) => {
  try {

    // Get query parameters for filtering
    const { startDate, endDate, leadPerson, salesPerson } = req.query;

    // Build filter object
    const filter = {
      isLeadPersonSale: true  // Always filter for lead person sales
    };

    // Date range filter
    if (startDate || endDate) {
      filter.date = {};
      if (startDate) filter.date.$gte = new Date(startDate);
      if (endDate) filter.date.$lte = new Date(endDate);
    }

    // Lead person filter - if user is a lead person, only show their leads
    // If admin or manager, allow filtering by lead person
    if (req.user.role === 'Lead Person') {
      // Convert to string ID for comparison
      const userId = req.user._id.toString();

      // Use mongoose ObjectId for the query
      const mongoose = require('mongoose');
      const ObjectId = mongoose.Types.ObjectId;

      try {
        filter.leadPerson = new ObjectId(userId);
      } catch (err) {
        // Fallback to string ID
        filter.leadPerson = userId;
      }
```

```
    } else if (leadPerson) {
      filter.leadPerson = leadPerson;
    }

    // Sales person filter
    if (salesPerson) {
      filter.salesPerson = salesPerson;
    }

    // Get sales data with all fields
    // Populate both leadPerson and salesPerson fields
    const sales = await Sale.find(filter)
      .select('date customerName country course countryCode contactNumber email
pseudoId salesPerson leadPerson source clientRemark feedback totalCost
totalCostCurrency tokenAmount tokenAmountCurrency currency')
      .populate('salesPerson', 'fullName')
      .populate('leadPerson', 'fullName')
      .sort({ date: -1 });

    // Post-process results to ensure currency fields exist and are consistent
    const processedSales = sales.map(sale => {
      const saleObj = sale.toObject();

      // Ensure currency fields are properly set
      // Priority: specific currency fields > general currency field > default USD
      if (!saleObj.totalCostCurrency) {
        saleObj.totalCostCurrency = saleObj.currency || 'USD';
      }
      if (!saleObj.tokenAmountCurrency) {
        saleObj.tokenAmountCurrency = saleObj.currency || 'USD';
      }

      // Also ensure the general currency field is set for consistency
      if (!saleObj.currency) {
        saleObj.currency = saleObj.totalCostCurrency || 'USD';
      }

      return saleObj;
    });

    res.status(200).json({
      success: true,
      count: processedSales.length,
      data: processedSales
    });
  } catch (err) {
    res.status(500).json({
      success: false,
      message: 'Server error while fetching sales data',
      error: err.message
    });
  }
});

// Add comprehensive reports endpoints
router.get('/reports/course-analysis', protect, authorize('Admin', 'Manager'),
async (req, res) => {
  try {
    const { period = 'monthly' } = req.query; // monthly, quarterly, half-yearly,
yearly
```

```
let dateFilter = {};
const now = new Date();

switch (period) {
  case 'monthly':
    // Last 12 months
    dateFilter = {
      $gte: new Date(now.getFullYear(), now.getMonth() - 11, 1),
      $lte: now
    };
    break;
  case 'quarterly':
    // Last 4 quarters (12 months)
    dateFilter = {
      $gte: new Date(now.getFullYear(), now.getMonth() - 11, 1),
      $lte: now
    };
    break;
  case 'half-yearly':
    // Last 3 years (6 half-year periods)
    dateFilter = {
      $gte: new Date(now.getFullYear() - 2, now.getMonth(), 1),
      $lte: now
    };
    break;
  case 'yearly':
    // Last 3 years
    dateFilter = {
      $gte: new Date(now.getFullYear() - 2, 0, 1),
      $lte: now
    };
    break;
}

const pipeline = [
  {
    $match: {
      date: dateFilter,
      status: { $ne: 'Cancelled' }
    }
  },
  {
    $group: {
      _id: {
        course: '$course',
        year: { $year: '$date' },
        month: { $month: '$date' },
        quarter: { $ceil: { $divide: [{ $month: '$date' }, 3] } }
      },
      totalSales: { $sum: 1 },
      totalRevenue: { $sum: { $ifNull: ['$totalCost', 0] } },
      averagePrice: { $avg: { $ifNull: ['$totalCost', 0] } }
    }
  },
  {
    $sort: { '_id.year': -1, '_id.month': -1 }
  }
];
```

```
    const results = await Sale.aggregate(pipeline);

    // Process results based on period
    const processedResults = {};

    results.forEach(item => {
      const course = item._id.course || 'Unknown Course';
      let periodKey = '';

      switch (period) {
        case 'monthly':
          periodKey = `${item._id.year}-${String(item._id.month).padStart(2, '0')}
`;
          break;
        case 'quarterly':
          periodKey = `${item._id.year}-Q${item._id.quarter}`;
          break;
        case 'half-yearly':
          const halfYear = item._id.month <= 6 ? 'H1' : 'H2';
          periodKey = `${item._id.year}-${halfYear}`;
          break;
        case 'yearly':
          periodKey = `${item._id.year}`;
          break;
      }

      if (!processedResults[course]) {
        processedResults[course] = {};
      }

      if (!processedResults[course][periodKey]) {
        processedResults[course][periodKey] = {
          totalSales: 0,
          totalRevenue: 0,
          averagePrice: 0
        };
      }

      processedResults[course][periodKey].totalSales += item.totalSales;
      processedResults[course][periodKey].totalRevenue += item.totalRevenue || 0;
      processedResults[course][periodKey].averagePrice = item.averagePrice || 0;
    });

    res.json({
      success: true,
      data: {
        period,
        courseAnalysis: processedResults
      }
    });
  } catch (error) {
    console.error('Error in course analysis:', error);
    res.status(500).json({
      success: false,
      message: 'Error generating course analysis report',
      error: error.message
    });
  }
```

```
});

router.get('/reports/revenue-analysis', protect, authorize('Admin', 'Manager'),
async (req, res) => {
  try {
    const { period = '1month' } = req.query; // 1month, 3month, 6month, 1year

    let dateFilter = {};
    const now = new Date();

    switch (period) {
      case '1month':
        dateFilter = {
          $gte: new Date(now.getFullYear(), now.getMonth() - 1, now.getDate()),
          $lte: now
        };
        break;
      case '3month':
        dateFilter = {
          $gte: new Date(now.getFullYear(), now.getMonth() - 3, now.getDate()),
          $lte: now
        };
        break;
      case '6month':
        dateFilter = {
          $gte: new Date(now.getFullYear(), now.getMonth() - 6, now.getDate()),
          $lte: now
        };
        break;
      case '1year':
        dateFilter = {
          $gte: new Date(now.getFullYear() - 1, now.getMonth(), now.getDate()),
          $lte: now
        };
        break;
    }

    const pipeline = [
      {
        $match: {
          date: dateFilter,
          status: { $ne: 'Cancelled' }
        }
      },
      {
        $group: {
          _id: {
            currency: '$currency',
            status: '$status'
          },
          totalSales: { $sum: 1 },
          totalRevenue: { $sum: '$totalCost' },
          totalTokens: { $sum: '$tokenAmount' },
          averageOrderValue: { $avg: '$totalCost' }
        }
      }
    ];

    const results = await Sale.aggregate(pipeline);
```

```
    // Get exchange rates with fallback
    let exchangeRates = {
      'USD': 1,
      'EUR': 0.85,
      'GBP': 0.73,
      'INR': 83.12,
      'CAD': 1.36,
      'AUD': 1.52,
      'JPY': 149.50,
      'CNY': 7.24
    };

    try {
      const ExchangeRate = require('../models/ExchangeRate');
      const exchangeRateDoc = await ExchangeRate.findOne().sort({ updatedAt:
-1 });
      if (exchangeRateDoc && exchangeRateDoc.rates) {
        exchangeRates = Object.fromEntries(exchangeRateDoc.rates);
      }
    } catch (err) {
      console.log('Using default exchange rates');
    }

    let totalRevenueUSD = 0;
    let totalTokensUSD = 0;
    let totalSalesCount = 0;
    const currencyBreakdown = {};

    results.forEach(item => {
      const currency = item._id.currency || 'USD';
      const rate = exchangeRates[currency] || 1;
      const revenueInUSD = item.totalRevenue / rate;
      const tokensInUSD = item.totalTokens / rate;

      totalRevenueUSD += revenueInUSD;
      totalTokensUSD += tokensInUSD;
      totalSalesCount += item.totalSales;

      if (!currencyBreakdown[currency]) {
        currencyBreakdown[currency] = {
          totalSales: 0,
          totalRevenue: 0,
          totalTokens: 0,
          revenueUSD: 0,
          tokensUSD: 0
        };
      }

      currencyBreakdown[currency].totalSales += item.totalSales;
      currencyBreakdown[currency].totalRevenue += item.totalRevenue;
      currencyBreakdown[currency].totalTokens += item.totalTokens;
      currencyBreakdown[currency].revenueUSD += revenueInUSD;
      currencyBreakdown[currency].tokensUSD += tokensInUSD;
    });

    // Get daily breakdown for the period
    const dailyPipeline = [
      {
```

```
        $match: {
          date: dateFilter,
          status: { $ne: 'Cancelled' }
        }
      },
      {
        $group: {
          _id: {
            year: { $year: '$date' },
            month: { $month: '$date' },
            day: { $dayOfMonth: '$date' }
          },
          dailySales: { $sum: 1 },
          dailyRevenue: { $sum: '$totalCost' },
          dailyTokens: { $sum: '$tokenAmount' }
        }
      },
      {
        $sort: { '_id.year': 1, '_id.month': 1, '_id.day': 1 }
      }
    ];

    const dailyResults = await Sale.aggregate(dailyPipeline);

    const dailyBreakdown = dailyResults.map(item => ({
      date: `${item._id.year}-${String(item._id.month).padStart(2, '0')}-
${String(item._id.day).padStart(2, '0')}`,
      sales: item.dailySales,
      revenue: item.dailyRevenue,
      tokens: item.dailyTokens
    }));

    res.json({
      success: true,
      data: {
        period,
        summary: {
          totalSales: totalSalesCount,
          totalRevenueUSD: Math.round(totalRevenueUSD * 100) / 100,
          totalTokensUSD: Math.round(totalTokensUSD * 100) / 100,
          pendingAmountUSD: Math.round((totalRevenueUSD - totalTokensUSD) *
100) / 100,
          averageOrderValueUSD: totalSalesCount > 0 ?
Math.round((totalRevenueUSD / totalSalesCount) * 100) / 100 : 0
        },
        currencyBreakdown,
        dailyBreakdown,
        exchangeRatesUsed: exchangeRates
      }
    });
  } catch (error) {
    console.error('Error in revenue analysis:', error);
    res.status(500).json({
      success: false,
      message: 'Error generating revenue analysis report'
    });
  }
});
```

```
router.get('/reports/top-courses', protect, authorize('Admin', 'Manager'), async
(req, res) => {
  try {
    const { period = 'all', limit = 10 } = req.query;

    let dateFilter = {};
    const now = new Date();

    if (period !== 'all') {
      switch (period) {
        case '1month':
          dateFilter = {
            $gte: new Date(now.getFullYear(), now.getMonth() - 1, now.getDate()),
            $lte: now
          };
          break;
        case '3month':
          dateFilter = {
            $gte: new Date(now.getFullYear(), now.getMonth() - 3, now.getDate()),
            $lte: now
          };
          break;
        case '6month':
          dateFilter = {
            $gte: new Date(now.getFullYear(), now.getMonth() - 6, now.getDate()),
            $lte: now
          };
          break;
        case '1year':
          dateFilter = {
            $gte: new Date(now.getFullYear() - 1, now.getMonth(), now.getDate()),
            $lte: now
          };
          break;
      }
    }

    const matchStage = {
      status: { $ne: 'Cancelled' }
    };

    if (Object.keys(dateFilter).length > 0) {
      matchStage.date = dateFilter;
    }

    const pipeline = [
      { $match: matchStage },
      {
        $group: {
          _id: '$course',
          totalSales: { $sum: 1 },
          totalRevenue: { $sum: { $ifNull: ['$totalCost', 0] } },
          averagePrice: { $avg: { $ifNull: ['$totalCost', 0] } },
          completedSales: {
            $sum: { $cond: [{ $eq: ['$status', 'Completed'] }, 1, 0] }
          }
        }
      },
      {
```

```
        $sort: { totalSales: -1 }
      },
      {
        $limit: parseInt(limit)
      }
    ];

    const results = await Sale.aggregate(pipeline);

    res.json({
      success: true,
      data: {
        period,
        topCourses: results.map(item => ({
          course: item._id || 'Unknown Course',
          totalSales: item.totalSales,
          totalRevenue: Math.round((item.totalRevenue || 0) * 100) / 100,
          averagePrice: Math.round((item.averagePrice || 0) * 100) / 100,
          completedSales: item.completedSales,
          completionRate: Math.round((item.completedSales / item.totalSales) *
100)
        }))
      }
    });
  } catch (error) {
    console.error('Error in top courses report:', error);
    res.status(500).json({
      success: false,
      message: 'Error generating top courses report',
      error: error.message
    });
  }
});

router.get('/reports/status-analysis', protect, authorize('Admin', 'Manager'),
async (req, res) => {
  try {
    const { period = '1month', status = null } = req.query; // 1month, 3month,
6month, 1year, all

    let dateFilter = {};
    const now = new Date();

    if (period !== 'all') {
      switch (period) {
        case '1month':
          dateFilter = {
            $gte: new Date(now.getFullYear(), now.getMonth() - 1, now.getDate()),
            $lte: now
          };
          break;
        case '3month':
          dateFilter = {
            $gte: new Date(now.getFullYear(), now.getMonth() - 3, now.getDate()),
            $lte: now
          };
          break;
        case '6month':
          dateFilter = {
```

```
            $gte: new Date(now.getFullYear(), now.getMonth() - 6, now.getDate()),
            $lte: now
          };
          break;
        case '1year':
          dateFilter = {
            $gte: new Date(now.getFullYear() - 1, now.getMonth(), now.getDate()),
            $lte: now
          };
          break;
      }
    }

    // Build match stage
    const matchStage = {};
    if (Object.keys(dateFilter).length > 0) {
      matchStage.date = dateFilter;
    }
    if (status) {
      matchStage.status = status;
    }

    // Get status summary
    const statusPipeline = [
      { $match: matchStage },
      {
        $group: {
          _id: '$status',
          totalSales: { $sum: 1 },
          totalRevenue: { $sum: '$totalCost' },
          totalTokens: { $sum: '$tokenAmount' },
          averageOrderValue: { $avg: '$totalCost' }
        }
      },
      {
        $sort: { totalSales: -1 }
      }
    ];

    const statusResults = await Sale.aggregate(statusPipeline);

    // Get detailed sales for specific status if requested
    let detailedSales = [];
    if (status) {
      detailedSales = await Sale.find(matchStage)
        .populate('salesPerson', 'fullName email')
        .populate('leadPerson', 'fullName email')
        .select('date customerName country course contactNumber email totalCost
tokenAmount pendingAmount status salesPerson leadPerson')
        .sort({ date: -1 })
        .limit(100); // Limit to 100 for performance
    }

    // Get exchange rates for USD conversion
    let exchangeRates = {
      'USD': 1,
      'EUR': 0.85,
      'GBP': 0.73,
      'INR': 83.12,
```

```
      'CAD': 1.36,
      'AUD': 1.52,
      'JPY': 149.50,
      'CNY': 7.24
    };

    try {
      const ExchangeRate = require('../models/ExchangeRate');
      const exchangeRateDoc = await ExchangeRate.findOne().sort({ updatedAt:
-1 });
      if (exchangeRateDoc && exchangeRateDoc.rates) {
        exchangeRates = Object.fromEntries(exchangeRateDoc.rates);
      }
    } catch (err) {
      console.log('Using default exchange rates for status analysis');
    }

    // Process status results with USD conversion
    const processedStatusResults = statusResults.map(item => {
      const currency = 'USD'; // Assuming most sales are in USD, you can modify
this logic
      const rate = exchangeRates[currency] || 1;

      return {
        status: item._id || 'Unknown',
        totalSales: item.totalSales,
        totalRevenue: Math.round(item.totalRevenue * 100) / 100,
        totalRevenueUSD: Math.round((item.totalRevenue / rate) * 100) / 100,
        totalTokens: Math.round(item.totalTokens * 100) / 100,
        totalTokensUSD: Math.round((item.totalTokens / rate) * 100) / 100,
        pendingAmountUSD: Math.round(((item.totalRevenue - item.totalTokens) /
rate) * 100) / 100,
        averageOrderValue: Math.round(item.averageOrderValue * 100) / 100,
        averageOrderValueUSD: Math.round((item.averageOrderValue / rate) * 100) /
100
      };
    });

    res.json({
      success: true,
      data: {
        period,
        selectedStatus: status,
        statusSummary: processedStatusResults,
        detailedSales: detailedSales.map(sale => ({
          _id: sale._id,
          date: sale.date,
          customerName: sale.customerName,
          country: sale.country,
          course: sale.course,
          contactNumber: sale.contactNumber,
          email: sale.email,
          totalCost: sale.totalCost,
          tokenAmount: sale.tokenAmount,
          pendingAmount: sale.pendingAmount,
          status: sale.status,
          salesPerson: sale.salesPerson?.fullName || 'Unknown',
          leadPerson: sale.leadPerson?.fullName || 'Unknown'
        })),
```

```
        totalCount: detailedSales.length,
        exchangeRatesUsed: exchangeRates
      }
    });
  } catch (error) {
    console.error('Error in status analysis:', error);
    res.status(500).json({
      success: false,
      message: 'Error generating status analysis report'
    });
  }
});

module.exports = router;
```

routes/taskRoutes.js

```
const express = require('express');
const {
  createTask,
  getTasks,
  getTask,
  updateTask,
  deleteTask
} = require('../controllers/taskController');

const router = express.Router();

const { protect, authorize } = require('../middleware/auth');

// Apply authentication to all routes
router.use(protect);

// Routes for tasks
router.route('/')
  .get(getTasks)
  .post(authorize('Sales Person', 'Lead Person', 'Manager', 'Admin'), createTask);

router.route('/:id')
  .get(getTask)
  .put(authorize('Sales Person', 'Lead Person', 'Manager', 'Admin'), updateTask)
  .delete(authorize('Sales Person', 'Lead Person', 'Manager', 'Admin'),
deleteTask);

module.exports = router;
```

routes/testExamNotifications.js

```
const express = require('express');
const router = express.Router();
const Task = require('../models/Task');
const User = require('../models/User');
const { checkUpcomingExams } = require('../utils/examNotificationService');

// @route   POST /api/test/create-exam
// @desc    Create a test exam task for notification testing
// @access  Private (for testing only)
router.post('/create-exam', async (req, res) => {
```

```
  try {
    const {
      course = 'Test Course',
      minutesFromNow = 11, // Default to 11 minutes from now (will trigger in 1
minute)
      userEmail
    } = req.body;

    // Find a user to assign the exam to
    let assignedUser;
    if (userEmail) {
      assignedUser = await User.findOne({ email: userEmail });
    } else {
      // Find any user for testing
      assignedUser = await User.findOne();
    }

    if (!assignedUser) {
      return res.status(400).json({
        success: false,
        message: 'No user found to assign the exam to'
      });
    }

    // Create exam date/time
    const examDateTime = new Date();
    examDateTime.setMinutes(examDateTime.getMinutes() + minutesFromNow);

    // Create test exam task
    const examTask = new Task({
      title: `Test Exam - ${course}`,
      description: `This is a test exam for ${course} to test the notification
system`,
      taskType: 'Exam',
      course: course,
      location: 'Online Test Environment',
      examLink: 'https://example.com/exam-portal',
      examDate: examDateTime,
      examDateTime: examDateTime,
      assignedTo: assignedUser._id,
      salesPerson: assignedUser._id,
      reminderSent: false
    });

    await examTask.save();

    res.status(201).json({
      success: true,
      message: `Test exam created successfully! Notification will be sent at
${new Date(examDateTime.getTime() - 10 * 60 * 1000).toLocaleString()}`,
      data: {
        examId: examTask._id,
        course: examTask.course,
        examDateTime: examTask.examDateTime,
        assignedTo: {
          name: assignedUser.fullName,
          email: assignedUser.email
        },
        notificationTime: new Date(examDateTime.getTime() - 10 * 60 * 1000)
```

```
      }
    });

  } catch (error) {
    console.error('Error creating test exam:', error);
    res.status(500).json({
      success: false,
      message: 'Error creating test exam',
      error: error.message
    });
  }
});

// @route   POST /api/test/trigger-notifications
// @desc    Manually trigger notification check (for testing)
// @access  Private (for testing only)
router.post('/trigger-notifications', async (req, res) => {
  try {
    const io = req.app.get('io');

    if (!io) {
      return res.status(500).json({
        success: false,
        message: 'Socket.IO not available'
      });
    }

    // Manually trigger the notification check
    await checkUpcomingExams(io);

    res.json({
      success: true,
      message: 'Notification check triggered successfully'
    });

  } catch (error) {
    console.error('Error triggering notifications:', error);
    res.status(500).json({
      success: false,
      message: 'Error triggering notifications',
      error: error.message
    });
  }
});

// @route   GET /api/test/upcoming-exams
// @desc    Get upcoming exams for testing
// @access  Private (for testing only)
router.get('/upcoming-exams', async (req, res) => {
  try {
    const now = new Date();
    const oneHourFromNow = new Date(now.getTime() + 60 * 60 * 1000);

    const upcomingExams = await Task.find({
      taskType: 'Exam',
      examDateTime: {
        $gte: now,
        $lte: oneHourFromNow
      }
```

```
    }).populate('assignedTo', 'fullName email');

    res.json({
      success: true,
      count: upcomingExams.length,
      data: upcomingExams.map(exam => ({
        id: exam._id,
        course: exam.course,
        examDateTime: exam.examDateTime,
        assignedTo: exam.assignedTo ? {
          name: exam.assignedTo.fullName,
          email: exam.assignedTo.email
        } : null,
        reminderSent: exam.reminderSent,
        minutesUntilExam: Math.round((exam.examDateTime - now) / (1000 * 60))
      }))
    });

  } catch (error) {
    console.error('Error fetching upcoming exams:', error);
    res.status(500).json({
      success: false,
      message: 'Error fetching upcoming exams',
      error: error.message
    });
  }
});

// @route   DELETE /api/test/cleanup-test-exams
// @desc    Clean up test exam tasks
// @access  Private (for testing only)
router.delete('/cleanup-test-exams', async (req, res) => {
  try {
    const result = await Task.deleteMany({
      title: { $regex: /^Test Exam/ }
    });

    res.json({
      success: true,
      message: `Cleaned up ${result.deletedCount} test exam tasks`
    });

  } catch (error) {
    console.error('Error cleaning up test exams:', error);
    res.status(500).json({
      success: false,
      message: 'Error cleaning up test exams',
      error: error.message
    });
  }
});

module.exports = router;
```

scripts/convertUsersToEmployees.js

```
const mongoose = require('mongoose');
const User = require('../models/User');
```

```
const Employee = require('../models/Employee');
const Department = require('../models/Department');
const EmployeeRole = require('../models/EmployeeRole');

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/crm', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const convertUsersToEmployees = async () => {
  try {
    console.log('Starting conversion of existing users to employees...');

    // Get all existing users
    const users = await User.find({});
    console.log(`Found ${users.length} users to convert`);

    // Get or create departments and roles
    const departments = await Department.find({});
    const roles = await EmployeeRole.find({});

    // Map CRM roles to departments and employee roles
    const roleMapping = {
      'Sales Person': {
        department: departments.find(d => d.name === 'Sales') || departments[0],
        role: roles.find(r => r.title === 'International Sales Executive') ||
roles[0]
      },
      'Lead Person': {
        department: departments.find(d => d.name === 'Sales') || departments[0],
        role: roles.find(r => r.title === 'Lead Executive') || roles[0]
      },
      'Manager': {
        department: departments.find(d => d.name === 'Sales') || departments[0],
        role: roles.find(r => r.title === 'Manager') || roles[0]
      },
      'Admin': {
        department: departments.find(d => d.name === 'IT') || departments[0],
        role: roles.find(r => r.title === 'Manager') || roles[0]
      },
      'HR': {
        department: departments.find(d => d.name === 'HR') || departments[0],
        role: roles.find(r => r.title === 'HR Executive') || roles[0]
      }
    };

    let convertedCount = 0;
    let skippedCount = 0;

    for (const user of users) {
      // Check if employee record already exists for this user
      const existingEmployee = await Employee.findOne({
        $or: [
          { userId: user._id },
          { 'personalInfo.email': user.email }
        ]
      });
```

```
      if (existingEmployee) {
        console.log(`Employee record already exists for user: ${user.email}`);
        skippedCount++;
        continue;
      }

      // Get mapping for this user's role
      const mapping = roleMapping[user.role];
      if (!mapping) {
        console.log(`No mapping found for role: ${user.role}, skipping user:
${user.email}`);
        skippedCount++;
        continue;
      }

      // Generate employee ID
      const employeeCount = await Employee.countDocuments();
      const employeeId = `EMP${String(employeeCount + 1).padStart(4, '0')}`;

      // Split full name
      const nameParts = user.fullName.split(' ');
      const firstName = nameParts[0] || '';
      const lastName = nameParts.slice(1).join(' ') || '';

      // Create employee record
      const employeeData = {
        employeeId,
        userId: user._id,
        personalInfo: {
          firstName,
          lastName,
          email: user.email,
          phone: '', // Will be empty, can be filled later
          address: '', // Will be empty, can be filled later
          dateOfBirth: null,
          gender: '',
          emergencyContact: {
            name: '',
            relationship: '',
            phone: ''
          }
        },
        professionalInfo: {
          joiningDate: user.createdAt, // Use account creation date as joining
date
          employmentType: 'Full-time',
          workLocation: 'Office',
          salary: 0, // Will be 0, can be updated later
          probationPeriod: 3
        },
        department: mapping.department._id,
        role: mapping.role._id,
        educationalInfo: [], // Empty array, can be filled later
        documents: {}, // Empty object, can be filled later
        status: 'Active'
      };

      const newEmployee = new Employee(employeeData);
      await newEmployee.save();
```

```
      // Update user record to link to employee
      await User.findByIdAndUpdate(user._id, {
        employeeId: newEmployee._id
      });

      console.log(`'  Converted user ${user.email} to employee ${employeeId}`);
      convertedCount++;
    }

    console.log('\n=== Conversion Summary ===');
    console.log(`Total users processed: ${users.length}`);
    console.log(`Successfully converted: ${convertedCount}`);
    console.log(`Skipped (already exists): ${skippedCount}`);
    console.log('Conversion completed!');

  } catch (error) {
    console.error('Error during conversion:', error);
  } finally {
    mongoose.connection.close();
  }
};

// Run the conversion
convertUsersToEmployees();
```

scripts/generateCodePDF.js

```
const PDFDocument = require('pdfkit');
const fs = require('fs');
const path = require('path');
const { promisify } = require('util');
const readdir = promisify(fs.readdir);
const readFile = promisify(fs.readFile);
const stat = promisify(fs.stat);

// Function to get all files recursively
async function getFiles(dir) {
  const files = [];
  const items = await readdir(dir);

  for (const item of items) {
    if (item.startsWith('.') || item === 'node_modules' || item === 'dist' ||
item === 'build') continue;

    const fullPath = path.join(dir, item);
    const stats = await stat(fullPath);

    if (stats.isDirectory()) {
      const subFiles = await getFiles(fullPath);
      files.push(...subFiles);
    } else {
      // Only include code files
      const ext = path.extname(item).toLowerCase();
      if (['.js', '.jsx', '.ts', '.tsx', '.css', '.html', '.json',
'.md'].includes(ext)) {
        files.push(fullPath);
      }
```

```
    }
  }

  return files;
}

async function generatePDF() {
  try {
    // Create PDF document
    const doc = new PDFDocument({
      size: 'A4',
      margins: {
        top: 50,
        bottom: 50,
        left: 50,
        right: 50
      }
    });

    // Pipe output to file
    doc.pipe(fs.createWriteStream('code_documentation.pdf'));

    // Set font
    doc.font('Helvetica');

    // Get all files
    const rootDir = path.join(__dirname, '..');
    const files = await getFiles(rootDir);

    // Sort files by directory and name
    files.sort((a, b) => a.localeCompare(b));

    // Process each file
    for (const file of files) {
      // Get relative path for display
      const relativePath = path.relative(rootDir, file);

      // Add page break except for first page
      if (doc.page.pageNumber > 1) {
        doc.addPage();
      }

      // Add file path as header
      doc.fontSize(16)
         .fillColor('#2563eb')
         .text(relativePath, { underline: true })
         .moveDown();

      // Read and add file content
      const content = await readFile(file, 'utf8');

      // Add file content with monospace font and smaller size
      doc.font('Courier')
         .fontSize(10)
         .fillColor('#000000')
         .text(content, {
           lineGap: 2,
           align: 'left'
         })
```

```
          .moveDown();
    }

    // Finalize PDF
    doc.end();
    console.log('PDF generated successfully: code_documentation.pdf');
  } catch (error) {
    console.error('Error generating PDF:', error);
  }
}

// Run the script
generatePDF();
```

scripts/seedData.js

```
const mongoose = require('mongoose');
const Department = require('../models/Department');
const Role = require('../models/EmployeeRole');

// Load env vars
require('dotenv').config({ path: './.env' });

// Connect to database
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const seedData = async () => {
  try {
    // Clear existing data
    await Department.deleteMany({});
    await Role.deleteMany({});

    // Seed departments
    const departments = [
      {
        name: 'Sales',
        description: 'Sales and Business Development Department'
      },
      {
        name: 'HR',
        description: 'Human Resources Department'
      },
      {
        name: 'IT',
        description: 'Information Technology Department'
      },
      {
        name: 'Marketing',
        description: 'Marketing and Digital Marketing Department'
      },
      {
        name: 'Finance',
        description: 'Finance and Accounting Department'
      },
      {
```

```
    name: 'Operations',
    description: 'Operations and Support Department'
  },
  {
    name: 'General',
    description: 'General Department for all employees'
  }
];

const createdDepartments = await Department.insertMany(departments);
console.log(`'  Seeded ${createdDepartments.length} departments`);

// Seed roles
const roles = [
  {
    name: 'Sales Person',
    description: 'Sales Executive responsible for customer acquisition'
  },
  {
    name: 'Lead Person',
    description: 'Lead Generation Specialist'
  },
  {
    name: 'Manager',
    description: 'Department Manager'
  },
  {
    name: 'Admin',
    description: 'System Administrator'
  },
  {
    name: 'HR',
    description: 'Human Resources Executive'
  },
  {
    name: 'HR Executive',
    description: 'Human Resources Executive'
  },
  {
    name: 'International Sales Executive',
    description: 'International Sales Executive'
  },
  {
    name: 'Lead Executive',
    description: 'Lead Generation Executive'
  },
  {
    name: 'Marketing Executive',
    description: 'Marketing and Promotion Executive'
  },
  {
    name: 'IT Support',
    description: 'Information Technology Support'
  },
  {
    name: 'Finance Executive',
    description: 'Finance and Accounting Executive'
  },
  {
```

```
        name: 'Operations Executive',
        description: 'Operations and Support Executive'
      },
      {
        name: 'Employee',
        description: 'General Employee'
      }
    ];

    const createdRoles = await Role.insertMany(roles);
    console.log(`'  Seeded ${createdRoles.length} roles`);

    console.log(''  Database seeded successfully!');
    process.exit(0);

  } catch (error) {
    console.error(''L Error seeding database:', error);
    process.exit(1);
  }
};

seedData();
```

seedEmployeeData.js

```
const mongoose = require('mongoose');
const Department = require('./models/Department');
const Role = require('./models/EmployeeRole');
require('dotenv').config();

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI || 'mongodb://
localhost:27017/crm');
    console.log(`MongoDB Connected: ${conn.connection.host}`);
  } catch (error) {
    console.error('Database connection error:', error);
    process.exit(1);
  }
};

const seedData = async () => {
  try {
    await connectDB();

    // Clear existing data
    await Department.deleteMany({});
    await Role.deleteMany({});

    // Create departments
    const departments = [
      {
        name: 'Sales',
        description: 'Sales and business development team'
      },
      {
        name: 'IT',
        description: 'Information Technology department'
```

```
  },
  {
    name: 'Marketing',
    description: 'Marketing and promotion team'
  },
  {
    name: 'HR',
    description: 'Human Resources department'
  },
  {
    name: 'Finance',
    description: 'Finance and accounting team'
  }
];

const createdDepartments = await Department.insertMany(departments);
console.log('✅ Departments created:', createdDepartments.length);

// Create roles
const roles = [
  {
    name: 'Lead Executive',
    description: 'Lead generation and management'
  },
  {
    name: 'International Sales Executive',
    description: 'International sales and client management'
  },
  {
    name: 'IT Specialist',
    description: 'Technical support and development'
  },
  {
    name: 'Intern',
    description: 'Internship position'
  },
  {
    name: 'Manager',
    description: 'Team management and supervision'
  },
  {
    name: 'HR Executive',
    description: 'Human resources management'
  },
  {
    name: 'Marketing Executive',
    description: 'Marketing campaigns and strategies'
  }
];

const createdRoles = await Role.insertMany(roles);
console.log('✅ Roles created:', createdRoles.length);

console.log('\n🎉 Employee data seeding completed successfully!');
console.log('\nCreated Departments:');
createdDepartments.forEach(dept => {
  console.log(`- ${dept.name}: ${dept.description}`);
});
```

```
    console.log('\nCreated Roles:');
    createdRoles.forEach(role => {
      console.log(`- ${role.name}: ${role.description}`);
    });

    process.exit(0);
  } catch (error) {
    console.error(''L Error seeding data:', error);
    process.exit(1);
  }
};

seedData();
```

server.js

```
const express = require('express');
const dotenv = require('dotenv');
const morgan = require('morgan');
const cors = require('cors');
const connectDB = require('./config/db');
const { corsMiddleware, ensureCorsHeaders, handleOptions } = require('./
middleware/cors');
// const ipFilter = require('./middleware/ipFilter');
const http = require('http');
const socketIo = require('socket.io');
// Load env vars
dotenv.config();

// Set DEBUG_CORS in development for testing
if (process.env.NODE_ENV === 'development') {
  process.env.DEBUG_CORS = 'true';
}

// Connect to database
console.log('Connecting to CRM database...');
connectDB();

// Use the IP filter middleware
// app.use(ipFilter);

// Route files
const authRoutes = require('./routes/auth');
const leadRoutes = require('./routes/leads');
const salesRoutes = require('./routes/sales');
const leadSalesRoutes = require('./routes/leadSalesRoute');
const leadPersonSalesRoutes = require('./routes/leadPersonSales');
const currencyRoutes = require('./routes/currency');
const taskRoutes = require('./routes/taskRoutes');
const testExamRoutes = require('./routes/testExamNotifications');
const chatRoutes = require('./routes/chat');
const prospectRoutes = require('./routes/prospects');
const activityRoutes = require('./routes/activity');
const employeeRoutes = require('./routes/employees');
const leaveRoutes = require('./routes/leaves');
const attendanceRoutes = require('./routes/attendance');
const payrollRoutes = require('./routes/payroll');
const incentivesRoutes = require('./routes/incentives');
```

```
const documentationRoutes = require('./routes/documentation');
const logs = require('./routes/logs');
const app = express();
const server = http.createServer(app);

// Socket.IO setup with CORS
const io = socketIo(server, {
  cors: {
    origin: [
      "http://localhost:3000",
      "http://localhost:5173",
      "https://traincapecrm.traincapetech.in"
    ],
    methods: ["GET", "POST"],
    credentials: true
  }
});

// Make io available to other modules
app.set('io', io);

// Chat service for Socket.IO
const ChatService = require('./services/chatService');

// Socket.IO connection handling
io.on('connection', (socket) => {
  console.log('User connected:', socket.id);

  // Check if this is a guest connection
  const isGuest = socket.handshake.query.isGuest === 'true';
  const guestId = socket.handshake.query.guestId;

  if (isGuest) {
    console.log('Guest connected:', guestId);

    // Handle guest joining their room
    socket.on('join-guest-room', (guestId) => {
      socket.join(`guest-${guestId}`);
      console.log(`Guest ${guestId} joined their room`);
    });

    // Handle guest requesting support team
    socket.on('get-support-team', async () => {
      try {
        const User = require('./models/User');
        const supportTeam = await User.find({
          role: { $in: ['Admin', 'Manager', 'Sales Person', 'Lead Person'] },
          chatStatus: 'ONLINE'
        }).select('fullName role chatStatus');

        socket.emit('support-team-list', supportTeam);
      } catch (error) {
        console.error('Error getting support team:', error);
      }
    });

    // Handle guest messages
    socket.on('guest-message', async (data) => {
      try {
```

```
      const { guestId, guestInfo, recipientId, content, timestamp } = data;

      // Create a guest message object
      const guestMessage = {
        id: Date.now(),
        guestId,
        guestInfo,
        content,
        timestamp,
        sender: 'guest'
      };

      // Send to support team member
      if (recipientId !== 'offline') {
        io.to(`user-${recipientId}`).emit('guest-message-received', {
          ...guestMessage,
          sender: 'guest',
          senderName: guestInfo.name,
          senderEmail: guestInfo.email
        });

        // Send notification
        io.to(`user-${recipientId}`).emit('messageNotification', {
          senderId: guestId,
          senderName: `${guestInfo.name} (Guest)`,
          content: content,
          timestamp: timestamp,
          isGuest: true
        });
      }

      // Confirm message received
      socket.emit('guest-message-sent', guestMessage);

    } catch (error) {
      console.error('Error handling guest message:', error);
      socket.emit('guest-message-error', { error: error.message });
    }
  });

  // Handle support team responding to guest
  socket.on('respond-to-guest', (data) => {
    const { guestId, content, senderName, timestamp } = data;

    io.to(`guest-${guestId}`).emit('guest-message-received', {
      id: Date.now(),
      content,
      sender: 'support',
      senderName,
      timestamp: new Date(timestamp)
    });
  });

} else {
  // Regular user connection handling

  // Join user to their personal room for targeted notifications
  socket.on('join-user-room', (userId) => {
    socket.join(`user-${userId}`);
```

```
  console.log(`User ${userId} joined their room`);

  // Update user status to online
  ChatService.updateUserStatus(userId, 'ONLINE').catch(console.error);

  // Broadcast user status update
  socket.broadcast.emit('userStatusUpdate', {
    userId,
    status: 'ONLINE',
    lastSeen: new Date()
  });
});

// Handle chat message sending via Socket.IO
socket.on('sendMessage', async (data) => {
  try {
    const { senderId, recipientId, content, messageType = 'text' } = data;

    const message = await ChatService.saveMessage({
      senderId,
      recipientId,
      content,
      messageType
    });

    // Send to recipient
    io.to(`user-${recipientId}`).emit('newMessage', {
      _id: message._id,
      chatId: message.chatId,
      senderId: message.senderId,
      recipientId: message.recipientId,
      content: message.content,
      messageType: message.messageType,
      timestamp: message.timestamp,
      isRead: message.isRead
    });

    // Send confirmation to sender
    socket.emit('messageDelivered', {
      _id: message._id,
      timestamp: message.timestamp
    });

    // Send notification to recipient
    io.to(`user-${recipientId}`).emit('messageNotification', {
      senderId: message.senderId,
      senderName: message.senderId.fullName,
      content: message.content,
      timestamp: message.timestamp
    });
  } catch (error) {
    console.error('Error sending message via socket:', error);
    socket.emit('messageError', { error: error.message });
  }
});

// Handle typing indicators
socket.on('typing', (data) => {
  const { recipientId, isTyping } = data;
```

```
      io.to(`user-${recipientId}`).emit('userTyping', {
        senderId: data.senderId,
        isTyping
      });
    });

    // Handle user status updates
    socket.on('updateStatus', async (data) => {
      try {
        const { userId, status } = data;
        await ChatService.updateUserStatus(userId, status);

        // Broadcast status update to all users
        io.emit('userStatusUpdate', {
          userId,
          status,
          lastSeen: new Date()
        });
      } catch (error) {
        console.error('Error updating user status:', error);
      }
    });
  }

  socket.on('disconnect', () => {
    console.log('User disconnected:', socket.id);

    // Note: We can't easily get userId from socket on disconnect
    // This would need to be handled by storing userId in socket data
    // For now, we'll rely on the frontend to send status updates
  });
});

// Reminder service
const { processExamReminders } = require('./utils/reminderService');
const { startExamNotificationScheduler } = require('./utils/
examNotificationService');

// Body parser
app.use(express.json({ limit: '50mb' }));
app.use(express.urlencoded({ limit: '50mb', extended: true }));

// Dev logging middleware
if (process.env.NODE_ENV === 'development') {
  app.use(morgan('dev'));
}

// Enable CORS with our custom middleware
app.use(corsMiddleware);

// Add a pre-flight route handler for OPTIONS requests
app.options('*', handleOptions);

// Add second layer of CORS protection to ensure headers are set
app.use(ensureCorsHeaders);

// Add a specific route for CORS preflight that always succeeds
app.options('/api/*', handleOptions);
```

```
// Serve static files from uploads directory
app.use('/uploads', express.static('uploads'));

// Mount routers
app.use('/api/auth', authRoutes);
app.use('/api/leads', leadRoutes);
app.use('/api/sales', salesRoutes);
app.use('/api/lead-sales', leadSalesRoutes);
app.use('/api/lead-person-sales', leadPersonSalesRoutes);
app.use('/api/currency', currencyRoutes);
app.use('/api/tasks', taskRoutes);
app.use('/api/test-exam', testExamRoutes);
app.use('/api/chat', chatRoutes);
app.use('/api/prospects', prospectRoutes);
app.use('/api/activity', activityRoutes);
app.use('/api/employees', employeeRoutes);
app.use('/api/leaves', leaveRoutes);
app.use('/api/attendance', attendanceRoutes);
app.use('/api/payroll', payrollRoutes);
app.use('/api/incentives', incentivesRoutes);
app.use('/api/documentation', documentationRoutes);
app.use('/api/logs', logs);

// Basic route for testing
app.get('/', (req, res) => {
  res.json({
    message: 'Welcome to CRM API',
    environment: process.env.NODE_ENV,
    version: '1.0.0'
  });
});

// Handle 404
app.use((req, res) => {
  res.status(404).json({
    success: false,
    message: 'Route not found'
  });
});

// Error handling middleware
app.use((err, req, res, next) => {
  console.error('Error:', err.message);
  if (err.name === 'UnauthorizedError') {
    return res.status(401).json({
      success: false,
      message: 'Invalid token'
    });
  }
  res.status(500).json({
    success: false,
    message: err.message || 'Internal server error'
  });
});

const PORT = process.env.PORT || 8080;

server.listen(PORT, () => {
  console.log(`Server running in ${process.env.NODE_ENV} mode on port ${PORT}`);
```

```
  // Start the exam notification scheduler
  startExamNotificationScheduler(io);
});

// Set up the reminder scheduler - run every 10 minutes
const REMINDER_INTERVAL = 10 * 60 * 1000; // 10 minutes in milliseconds
setInterval(() => {
  console.log('Running exam reminder scheduler...');
  processExamReminders(io);
}, REMINDER_INTERVAL);

// Also run once at startup
console.log('Initial run of exam reminder scheduler...');
processExamReminders(io);

// Handle unhandled promise rejections
process.on('unhandledRejection', (err, promise) => {
  console.log(`Error: ${err.message}`);
  // Close server & exit process
  server.close(() => process.exit(1));
});
```

services/chatService.js

```
const ChatRoom = require('../models/ChatRoom');
const ChatMessage = require('../models/ChatMessage');
const User = require('../models/User');

class ChatService {
  // Create or get existing chat room between two users
  static async getOrCreateChatRoom(senderId, recipientId) {
    try {
      // Create a consistent chatId regardless of who initiates
      const chatId = [senderId, recipientId].sort().join('_');

      // Check if chat room already exists
      let chatRoom = await ChatRoom.findOne({ chatId });

      if (!chatRoom) {
        // Create new chat room
        chatRoom = new ChatRoom({
          chatId,
          senderId,
          recipientId
        });
        await chatRoom.save();
      }

      return chatRoom;
    } catch (error) {
      throw new Error(`Error creating/getting chat room: ${error.message}`);
    }
  }

  // Save a chat message
  static async saveMessage(messageData) {
    try {
```

```
      const { senderId, recipientId, content, messageType = 'text' } =
messageData;

      // Get or create chat room
      const chatRoom = await this.getOrCreateChatRoom(senderId, recipientId);

      // Create new message
      const message = new ChatMessage({
        chatId: chatRoom.chatId,
        senderId,
        recipientId,
        content,
        messageType,
        timestamp: new Date()
      });

      const savedMessage = await message.save();

      // Update chat room with last message info
      await ChatRoom.findByIdAndUpdate(chatRoom._id, {
        lastMessage: content,
        lastMessageTime: new Date(),
        $inc: {
          [`unreadCount.${recipientId === chatRoom.senderId ? 'senderId' :
'recipientId'}`]: 1
        }
      });

      // Populate sender and recipient info
      await savedMessage.populate('senderId', 'fullName email profilePicture');
      await savedMessage.populate('recipientId', 'fullName email profilePicture');

      return savedMessage;
    } catch (error) {
      throw new Error(`Error saving message: ${error.message}`);
    }
  }

  // Get chat messages between two users
  static async getChatMessages(senderId, recipientId, page = 1, limit = 50) {
    try {
      const chatId = [senderId, recipientId].sort().join('_');

      const messages = await ChatMessage.find({ chatId })
        .populate('senderId', 'fullName email profilePicture')
        .populate('recipientId', 'fullName email profilePicture')
        .sort({ timestamp: 1 })
        .limit(limit * 1)
        .skip((page - 1) * limit);

      // Mark messages as read for the recipient
      await ChatMessage.updateMany(
        {
          chatId,
          recipientId: senderId,
          isRead: false
        },
        { isRead: true }
      );
```

```
      // Reset unread count for the recipient
      await ChatRoom.updateOne(
        { chatId },
        {
          $set: {
            [`unreadCount.${senderId}`]: 0
          }
        }
      );

      return messages;
    } catch (error) {
      throw new Error(`Error getting chat messages: ${error.message}`);
    }
  }

  // Get user's chat rooms with last message info
  static async getUserChatRooms(userId) {
    try {
      const chatRooms = await ChatRoom.find({
        $or: [
          { senderId: userId },
          { recipientId: userId }
        ]
      })
      .populate('senderId', 'fullName email profilePicture chatStatus lastSeen')
      .populate('recipientId', 'fullName email profilePicture chatStatus
lastSeen')
      .sort({ lastMessageTime: -1 });

      // Format the response to include the other user's info
      const formattedRooms = chatRooms.map(room => {
        const otherUser = room.senderId._id.toString() === userId.toString()
          ? room.recipientId
          : room.senderId;

        const unreadCount = room.senderId._id.toString() === userId.toString()
          ? room.unreadCount.senderId
          : room.unreadCount.recipientId;

        return {
          chatId: room.chatId,
          otherUser: {
            _id: otherUser._id,
            fullName: otherUser.fullName,
            email: otherUser.email,
            profilePicture: otherUser.profilePicture,
            chatStatus: otherUser.chatStatus,
            lastSeen: otherUser.lastSeen
          },
          lastMessage: room.lastMessage,
          lastMessageTime: room.lastMessageTime,
          unreadCount
        };
      });

      return formattedRooms;
    } catch (error) {
```

```
      throw new Error(`Error getting user chat rooms: ${error.message}`);
    }
  }

  // Update user chat status
  static async updateUserStatus(userId, status) {
    try {
      await User.findByIdAndUpdate(userId, {
        chatStatus: status,
        lastSeen: new Date()
      });
    } catch (error) {
      throw new Error(`Error updating user status: ${error.message}`);
    }
  }

  // Get online users
  static async getOnlineUsers(excludeUserId = null) {
    try {
      const query = { chatStatus: 'ONLINE' };
      if (excludeUserId) {
        query._id = { $ne: excludeUserId };
      }

      const users = await User.find(query)
        .select('fullName email profilePicture chatStatus lastSeen role')
        .sort({ fullName: 1 });

      return users;
    } catch (error) {
      throw new Error(`Error getting online users: ${error.message}`);
    }
  }

  // Get all users for chat (excluding current user)
  static async getAllUsersForChat(excludeUserId) {
    try {
      const users = await User.find({ _id: { $ne: excludeUserId } })
        .select('fullName email profilePicture chatStatus lastSeen role')
        .sort({ fullName: 1 });

      return users;
    } catch (error) {
      throw new Error(`Error getting users for chat: ${error.message}`);
    }
  }
}

module.exports = ChatService;
```

services/emailService.js

```
const nodemailer = require('nodemailer');

// Email provider configurations
const getEmailConfig = (email) => {
  const domain = email.split('@')[1].toLowerCase();
```

```
  // Hostinger email configuration
  if (domain === 'traincapetech.in' || domain.includes('hostinger')) {
    return {
      host: 'smtp.hostinger.com',
      port: 587,
      secure: false, // true for 465, false for other ports
      auth: {
        user: email,
        pass: process.env.HOSTINGER_EMAIL_PASS || process.env.EMAIL_PASS
      }
    };
  }

  // Gmail configuration
  if (domain === 'gmail.com') {
    return {
      service: 'gmail',
      auth: {
        user: email,
        pass: process.env.GMAIL_APP_PASS || process.env.EMAIL_PASS
      }
    };
  }

  // Outlook/Hotmail configuration
  if (domain === 'outlook.com' || domain === 'hotmail.com' || domain ===
'live.com') {
    return {
      service: 'hotmail',
      auth: {
        user: email,
        pass: process.env.OUTLOOK_EMAIL_PASS || process.env.EMAIL_PASS
      }
    };
  }

  // Yahoo configuration
  if (domain === 'yahoo.com' || domain === 'yahoo.in') {
    return {
      service: 'yahoo',
      auth: {
        user: email,
        pass: process.env.YAHOO_EMAIL_PASS || process.env.EMAIL_PASS
      }
    };
  }

  // Generic SMTP configuration (fallback)
  return {
    host: process.env.SMTP_HOST || 'smtp.gmail.com',
    port: process.env.SMTP_PORT || 587,
    secure: false,
    auth: {
      user: email,
      pass: process.env.EMAIL_PASS
    }
  };
};
```

```
// Create transporter based on sender email
const createTransporter = (senderEmail) => {
  const config = getEmailConfig(senderEmail);
  return nodemailer.createTransporter(config);
};

// Payment confirmation email template
const getPaymentConfirmationTemplate = (data) => {
  const { customerName, tokenAmount, currency, course, totalCost, pendingAmount,
paymentDate } = data;

  return `
    <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto;
padding: 20px; border: 1px solid #ddd; border-radius: 8px;">
      <h2 style="color: #2563eb; text-align: center;">Payment Confirmation</h2>

      <p>Dear ${customerName},</p>

      <p><strong>Warm Greetings!</strong></p>

      <p>We earnestly acknowledge your payment of <strong>${tokenAmount}
${currency}</strong> received through UPI ahead savings account on ${paymentDate}
for <strong>${course}</strong> service delivery.</p>

      <p>Thank you for trusting <strong>Traincape Technology Pvt Ltd</strong>
ahead for your certification process.</p>

      <p>Please note, your next installment for the payment will be
<strong>${pendingAmount} ${currency}</strong> after service delivery.</p>

      <p>We look forward to offering you our best services and to continue being
in business with you in the long run.</p>

      <p>If we can be of any further assistance, please do not hesitate to
contact me.</p>

      <p><strong>We appreciate your business</strong></p>

      <hr style="margin: 20px 0;">
      <p style="font-size: 12px; color: #666;">
        This is an automated message from Traincape Technology Pvt Ltd.<br>
        Total Course Amount: ${totalCost} ${currency}<br>
        Token Amount Paid: ${tokenAmount} ${currency}<br>
        Pending Amount: ${pendingAmount} ${currency}
      </p>
    </div>
  `;
};

// Service delivery email template
const getServiceDeliveryTemplate = (data) => {
  const { customerName, totalCost, currency, course, paymentDate } = data;

  return `
    <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto;
padding: 20px; border: 1px solid #ddd; border-radius: 8px;">
      <h2 style="color: #16a34a; text-align: center;">Service Delivery
Confirmation</h2>
```

```
      <p>Dear ${customerName},</p>

      <p><strong>Warm Greetings!</strong></p>

      <p>We earnestly acknowledge your payment of <strong>${totalCost} ${currency}
</strong> received through Stripe on ${paymentDate} for <strong>${course}</
strong> service delivery.</p>

      <p>Thank you for trusting <strong>Traincape Technology Pvt Ltd</strong>
ahead for your certification process.</p>

      <p>We look forward to offering you our best services and to continue being
in business with you in the long run.</p>

      <p>If we can be of any further assistance, please do not hesitate to
contact me.</p>

      <p><strong>We appreciate your business</strong></p>

      <hr style="margin: 20px 0;">
      <p style="font-size: 12px; color: #666;">
        This is an automated message from Traincape Technology Pvt Ltd.<br>
        Course: ${course}<br>
        Total Amount: ${totalCost} ${currency}<br>
        Status: Service Delivered
      </p>
    </div>
  `;
};

// Send payment confirmation email
const sendPaymentConfirmationEmail = async (saleData, salesPersonEmail) => {
  try {
    // Validate inputs
    if (!salesPersonEmail) {
      console.log(''L Sales person email not available');
      return { success: false, message: 'Sales person email not available' };
    }

    if (!saleData.email) {
      console.log(''L Customer email unavailable for:', saleData.customerName);
      return { success: false, message: 'Customer email not available' };
    }

    const transporter = createTransporter(salesPersonEmail);

    const pendingAmount = (saleData.totalCost || 0) - (saleData.tokenAmount || 0);
    const paymentDate = new Date(saleData.date ||
Date.now()).toLocaleDateString();

    const emailData = {
      customerName: saleData.customerName,
      tokenAmount: saleData.tokenAmount || 0,
      currency: saleData.totalCostCurrency || saleData.currency || 'USD',
      course: saleData.course,
      totalCost: saleData.totalCost || 0,
      pendingAmount: pendingAmount,
      paymentDate: paymentDate
    };
```

```
    const mailOptions = {
      from: `"Traincape Technology" <${salesPersonEmail}>`,
      to: saleData.email,
      cc: salesPersonEmail, // CC the sales person
      subject: `Payment Confirmation - ${saleData.course} -
${saleData.customerName}`,
      html: getPaymentConfirmationTemplate(emailData)
    };

    const result = await transporter.sendMail(mailOptions);
    console.log(''  Payment confirmation email sent:', result.messageId);

    return { success: true, messageId: result.messageId };
  } catch (error) {
    console.error(''L Error sending payment confirmation email:', error);

    // Provide specific error messages based on error type
    let errorMessage = 'Failed to send payment confirmation email';
    if (error.code === 'EAUTH') {
      errorMessage = 'Email authentication failed - check email credentials';
    } else if (error.code === 'ECONNECTION') {
      errorMessage = 'Email server connection failed';
    } else if (error.responseCode === 535) {
      errorMessage = 'Invalid email credentials';
    }

    return { success: false, error: errorMessage };
  }
};

// Send service delivery email
const sendServiceDeliveryEmail = async (saleData, salesPersonEmail) => {
  try {
    // Validate inputs
    if (!salesPersonEmail) {
      console.log(''L Sales person email not available');
      return { success: false, message: 'Sales person email not available' };
    }

    if (!saleData.email) {
      console.log(''L Customer email unavailable for:', saleData.customerName);
      return { success: false, message: 'Customer email not available' };
    }

    const transporter = createTransporter(salesPersonEmail);

    const paymentDate = new Date(saleData.date ||
Date.now()).toLocaleDateString();

    const emailData = {
      customerName: saleData.customerName,
      totalCost: saleData.totalCost || 0,
      currency: saleData.totalCostCurrency || saleData.currency || 'USD',
      course: saleData.course,
      paymentDate: paymentDate
    };

    const mailOptions = {
```

```
      from: `"Traincape Technology" <${salesPersonEmail}>`,
      to: saleData.email,
      cc: salesPersonEmail, // CC the sales person
      subject: `Service Delivery Confirmation - ${saleData.course} -
${saleData.customerName}`,
      html: getServiceDeliveryTemplate(emailData)
    };

    const result = await transporter.sendMail(mailOptions);
    console.log(''  Service delivery email sent:', result.messageId);

    return { success: true, messageId: result.messageId };
  } catch (error) {
    console.error(''L Error sending service delivery email:', error);

    // Provide specific error messages based on error type
    let errorMessage = 'Failed to send service delivery email';
    if (error.code === 'EAUTH') {
      errorMessage = 'Email authentication failed - check email credentials';
    } else if (error.code === 'ECONNECTION') {
      errorMessage = 'Email server connection failed';
    } else if (error.responseCode === 535) {
      errorMessage = 'Invalid email credentials';
    }

    return { success: false, error: errorMessage };
  }
};

module.exports = {
  sendPaymentConfirmationEmail,
  sendServiceDeliveryEmail
};
```

test notifications.js

```
const mongoose = require('mongoose');
const Task = require('./models/Task');
const User = require('./models/User');
require('dotenv').config();

// Connect to MongoDB
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

async function createTestExam() {
  try {
    // Find a user to assign the exam to
    const user = await User.findOne();

    if (!user) {
      console.log('No users found in database');
      return;
    }

    // Create exam date/time - 2 minutes from now
```

```
    const examDateTime = new Date();
    examDateTime.setMinutes(examDateTime.getMinutes() + 2);

    // Create test exam task
    const examTask = new Task({
      title: 'Test Notification Exam',
      description: 'React Development Certification Exam - Testing notification
system',
      taskType: 'Exam',
      course: 'React Development',
      location: 'Online',
      examLink: 'https://example.com/exam-portal',
      examDate: examDateTime,
      examDateTime: examDateTime,
      assignedTo: user._id,
      salesPerson: user._id,
      reminderSent: false,
      manualCustomer: {
        name: 'Test Customer',
        email: 'test@example.com',
        contactNumber: '+1234567890',
        course: 'React Development'
      }
    });

    await examTask.save();

    console.log('' Test exam created successfully!');
    console.log(`Ø=ÜÅ Exam scheduled for: ${examDateTime.toLocaleString()}`);
    console.log(`Ø=Üd Assigned to: ${user.fullName} (${user.email})`);
    console.log(`Ø=Ý Notifications will be sent at:`);
    console.log(`   - 30 minutes before: ${new Date(examDateTime.getTime() - 30 *
60 * 1000).toLocaleString()}`);
    console.log(`   - Exam time: ${examDateTime.toLocaleString()}`);
    console.log(`   - 10 minutes after: ${new Date(examDateTime.getTime() + 10 *
60 * 1000).toLocaleString()}`);
    console.log(`\nØ<ß¯ Since this exam is in 2 minutes, you should receive an "exam-
time" notification soon!`);

    process.exit(0);
  } catch (error) {
    console.error('Error creating test exam:', error);
    process.exit(1);
  }
}

createTestExam();
```

test_rajesh_duplicates.js

```
const axios = require('axios');

const BASE_URL = 'http://localhost:8080/api';

async function testRajeshDuplicates() {
  try {
    console.log('Ø=Ý Authenticating as admin...');
```

```
    // Login with provided admin credentials
    const loginResponse = await axios.post(`${BASE_URL}/auth/login`, {
      email: 'SK@gmail.com',
      password: 'Canada@1212'
    });

    const token = loginResponse.data.token;
    console.log('' Authentication successful');

    // Set up headers with token
    const headers = {
      'Authorization': `Bearer ${token}`,
      'Content-Type': 'application/json'
    };

    console.log('\nØ=Ý Checking Rajesh duplicates...');

    // Check for duplicates
    const checkResponse = await axios.get(`${BASE_URL}/leads/check-rajesh-
duplicates`, { headers });

    const data = checkResponse.data.data;

    console.log('\nØ=ÜÊ Results:');
    console.log(`Rajesh: ${data.rajesh.name} (${data.rajesh.id})`);
    console.log(`Total leads: ${data.totalLeads}`);

    console.log('\nØ=ÜÅ Leads by month:');
    data.leadsByMonth.forEach(month => {
      console.log(`  ${month.month}: ${month.count} leads`);
    });

    console.log(`\nØ=Ý  September 2024 leads: ${data.sep2024Leads.length}`);
    if (data.sep2024Leads.length > 0) {
      data.sep2024Leads.forEach((lead, i) => {
        console.log(`  ${i+1}. ${lead.name} - ${lead.phone} - ${lead.course}`);
      });
    }

    console.log(`\nØ=Ý  June 2025 leads: ${data.jun2025Leads.length}`);
    if (data.jun2025Leads.length > 0) {
      data.jun2025Leads.forEach((lead, i) => {
        console.log(`  ${i+1}. ${lead.name} - ${lead.phone} - ${lead.course}`);
      });
    }

    console.log(`\n& þ   Duplicates found: ${data.duplicateCount}`);
    if (data.duplicates.length > 0) {
      data.duplicates.forEach((dup, i) => {
        console.log(`  ${i+1}. Sep 2024: ${dup.sep2024.name}
(${dup.sep2024.phone})`);
        console.log(`     Jun 2025: ${dup.jun2025.name} (${dup.jun2025.phone})`);
        console.log(`     Jun 2025 ID: ${dup.jun2025.id}`);
      });

      // Remove duplicates
      console.log('\nØ=ÝÑþ   Removing duplicate June 2025 leads...');

      const removeResponse = await axios.delete(`${BASE_URL}/leads/remove-rajesh-
```

```
duplicates`, { headers });

      console.log('' Removal result:', removeResponse.data.message);
      console.log(`  Removed count: ${removeResponse.data.removedCount}`);

    } else {
      console.log('' No duplicates found');
    }

  } catch (error) {
    console.error(''L Error:', error.response?.data?.message || error.message);
    if (error.response?.data) {
      console.error('Response data:', error.response.data);
    }
  }
}

testRajeshDuplicates();
```

test-email.js

```
const { sendPaymentConfirmationEmail, sendServiceDeliveryEmail } = require('./
services/emailService');
require('dotenv').config();

// Test data
const testSaleData = {
  customerName: 'Test Customer',
  email: 'test@example.com', // Change this to your test email
  course: 'PL-300 Test Course',
  totalCost: 15000,
  totalCostCurrency: 'INR',
  tokenAmount: 5000,
  tokenAmountCurrency: 'INR',
  date: new Date()
};

// Test different sales person emails
const testEmails = [
  'saurav@traincapetech.in', // Hostinger
  'test@gmail.com',          // Gmail (if you have one)
  'test@outlook.com',        // Outlook (if you have one)
  'test@yahoo.com'           // Yahoo (if you have one)
];

const testEmailConfiguration = async () => {
  console.log('Ø>Ýê Testing Email Configuration...\n');

  // Test each email provider
  for (const salesPersonEmail of testEmails) {
    console.log(`\nØ=Üç Testing with sales person: ${salesPersonEmail}`);
    console.log('=' .repeat(50));

    try {
      // Test payment confirmation email
      console.log('Testing payment confirmation email...');
      const paymentResult = await sendPaymentConfirmationEmail(testSaleData,
salesPersonEmail);
```

```
      if (paymentResult.success) {
        console.log('  Payment confirmation email test: SUCCESS');
        console.log(`   Message ID: ${paymentResult.messageId}`);
      } else {
        console.log('L Payment confirmation email test: FAILED');
        console.log(`   Error: ${paymentResult.message || paymentResult.error}`);
      }

      // Wait a bit between emails
      await new Promise(resolve => setTimeout(resolve, 2000));

      // Test service delivery email
      console.log('Testing service delivery email...');
      const deliveryResult = await sendServiceDeliveryEmail(testSaleData,
salesPersonEmail);

      if (deliveryResult.success) {
        console.log('  Service delivery email test: SUCCESS');
        console.log(`   Message ID: ${deliveryResult.messageId}`);
      } else {
        console.log('L Service delivery email test: FAILED');
        console.log(`   Error: ${deliveryResult.message || deliveryResult.error}
`);
      }

    } catch (error) {
      console.log('L Email test failed with exception:');
      console.log(`   ${error.message}`);
    }

    console.log(''); // Empty line for readability
  }

  console.log('\nØ<ßÁ Email testing completed!');
  console.log('\nNext steps:');
  console.log('1. Check your test email inbox for received emails');
  console.log('2. Verify the dynamic content is correct');
  console.log('3. Check that sales person gets CC copies');
  console.log('4. Update environment variables for any failed providers');
};

// Run the test
testEmailConfiguration().catch(console.error);

// Instructions for running this test
console.log(`
Ø=ÜË EMAIL TEST INSTRUCTIONS:

1. Update testSaleData.email to your actual test email address
2. Set up environment variables for the email providers you want to test:

   For Hostinger:
   HOSTINGER_EMAIL_PASS=your_hostinger_password

   For Gmail:
   GMAIL_APP_PASS=your_gmail_app_password

   For Outlook:
```

```
   OUTLOOK_EMAIL_PASS=your_outlook_password

   For Yahoo:
   YAHOO_EMAIL_PASS=your_yahoo_password

3. Run this test: node test-email.js

4. Check your test email inbox for the emails

Note: Only test with email addresses you actually have credentials for.
Remove or comment out email addresses you don't want to test.
`);
```

test-mongo-connection.js

```
const mongoose = require('mongoose');
const dns = require('dns');

async function testConnection() {
  try {
    // Get MongoDB URI from environment
    const mongoUri = process.env.MONGO_URI;

    if (!mongoUri) {
      console.error(''L MONGO_URI is not set in environment variables');
      return;
    }

    // Extract hostname from URI
    const match = mongoUri.match(/mongodb(?:\+srv)?:\/\/([^/:]+)/);
    if (!match) {
      console.error(''L Invalid MongoDB URI format');
      return;
    }

    const hostname = match[1];

    // Test DNS resolution
    console.log(`Ø=Ý  Testing DNS resolution for ${hostname}...`);
    try {
      const addresses = await new Promise((resolve, reject) => {
        dns.resolve(hostname, (err, addresses) => {
          if (err) reject(err);
          else resolve(addresses);
        });
      });
      console.log(''  DNS resolution successful:', addresses);
    } catch (dnsError) {
      console.error(''L DNS resolution failed:', dnsError.message);
      console.log('\nØ=ÜI Possible solutions:');
      console.log('1. Check your internet connection');
      console.log('2. Verify the cluster name in MONGO_URI is correct');
      console.log('3. Ensure the MongoDB Atlas cluster is running');
      console.log('4. Try using a different DNS server');
      return;
    }

    // Test MongoDB connection
```

```
    console.log('\nØ=Ý  Testing MongoDB connection...');
    await mongoose.connect(mongoUri, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
      serverSelectionTimeoutMS: 5000 // 5 second timeout
    });

    console.log('  Successfully connected to MongoDB!');
    console.log(`Ø=ÜÍ Connected to: ${mongoose.connection.host}`);

    // Test basic operation
    console.log('\nØ=Ý  Testing basic database operation...');
    const pingResult = await mongoose.connection.db.admin().ping();
    console.log('  Database ping successful:', pingResult);

  } catch (error) {
    console.error('\n'L Connection error:', error.message);
    console.log('\nØ=ÜI Error details:', error);
    console.log('\nØ=ÜI Possible solutions:');
    console.log('1. Verify MONGO_URI is correct');
    console.log('2. Check if IP whitelist includes your current IP');
    console.log('3. Verify database user credentials');
    console.log('4. Ensure the cluster is running and accessible');
  } finally {
    await mongoose.disconnect();
  }
}

testConnection().catch(console.error);
```

utils/dropUniqueIndexes.js

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const path = require('path');

// Load environment variables
dotenv.config({ path: path.resolve(__dirname, '../.env') });

// Set default MongoDB connection string if not in env
if (!process.env.MONGO_URI) {
  console.log('MongoDB connection string not found in .env, using default local
connection');
  process.env.MONGO_URI = 'mongodb://localhost:27017/crm';
}

// Connect to MongoDB
const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true
    });

    console.log(`MongoDB Connected: ${conn.connection.host}`);
    return conn;
  } catch (error) {
    console.error(`Error: ${error.message}`);
```

```
    process.exit(1);
  }
};

// Function to drop unique indexes from Lead collection
async function dropUniqueIndexes() {
  try {
    // Connect to the database
    await connectDB();

    console.log('Connected to MongoDB, checking for unique indexes...');

    // Get the Lead collection
    const db = mongoose.connection.db;
    const Lead = db.collection('leads');

    // Get all indexes on the Lead collection
    const indexes = await Lead.indexes();
    console.log('Current indexes:', indexes);

    // Find and drop any unique indexes on email or phone
    for (const index of indexes) {
      // Check if this is a unique index on email or phone
      if (index.unique === true) {
        if (index.key.email || index.key.phone) {
          const fieldName = index.key.email ? 'email' : 'phone';
          console.log(`Found unique index on ${fieldName} field: ${index.name}`);

          // Drop the unique index
          await Lead.dropIndex(index.name);
          console.log(`Successfully dropped unique index: ${index.name}`);
        }
      }
    }

    // Verify indexes after dropping
    const remainingIndexes = await Lead.indexes();
    console.log('Remaining indexes after cleanup:', remainingIndexes);

    console.log('Index cleanup completed successfully');
  } catch (error) {
    console.error('Error dropping unique indexes:', error);
  } finally {
    // Close the connection
    await mongoose.connection.close();
    console.log('Database connection closed');
  }
}

// Run the function
dropUniqueIndexes();
```

utils/errorResponse.js

```
class ErrorResponse extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
```

```
    Error.captureStackTrace(this, this.constructor);
  }
}

module.exports = ErrorResponse;
```

utils/examNotificationService.js

```
const cron = require('node-cron');
const nodemailer = require('nodemailer');
const Task = require('../models/Task');

// Email transporter setup
const transporter = nodemailer.createTransport({
  host: 'smtp.hostinger.com',
  port: 587,
  secure: false,
  auth: {
    user: process.env.EMAIL_USER || 'noreply@traincapetech.in',
    pass: process.env.EMAIL_PASS
  }
});

// Function to send exam reminder email
const sendExamReminderEmail = async (userEmail, userName, examDetails) => {
  try {
    const mailOptions = {
      from: process.env.EMAIL_USER || 'noreply@traincapetech.in',
      to: userEmail,
      subject: 'Ø=Þ¨ Exam Reminder - 10 Minutes to Go!',
      html: `
        <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0
auto; padding: 20px; background-color: #f8f9fa;">
          <div style="background-color: #dc3545; color: white; padding: 20px;
border-radius: 8px 8px 0 0; text-align: center;">
            <h1 style="margin: 0; font-size: 24px;">Ø=Þ¨ EXAM ALERT Ø=Þ¨</h1>
          </div>

          <div style="background-color: white; padding: 30px; border-radius: 0 0
8px 8px; box-shadow: 0 2px 10px rgba(0,0,0,0.1);">
            <h2 style="color: #dc3545; margin-top: 0;">Hi ${userName},</h2>

            <div style="background-color: #fff3cd; border: 1px solid #ffeaa7;
padding: 15px; border-radius: 5px; margin: 20px 0;">
              <p style="margin: 0; font-size: 18px; font-weight: bold; color:
#856404;">
                #ð Your exam starts in 10 minutes!
              </p>
            </div>

            <div style="background-color: #f8f9fa; padding: 20px; border-radius:
5px; margin: 20px 0;">
              <h3 style="margin-top: 0; color: #495057;">Exam Details:</h3>
              <p><strong>Ø=ÜÚ Course:</strong> ${examDetails.course}</p>
              <p><strong>Ø=ÜÅ Date:</strong> ${examDetails.date}</p>
              <p><strong>#ð Time:</strong> ${examDetails.time}</p>
              <p><strong>Ø=ÜÍ Location:</strong> ${examDetails.location || 'Online'}</
p>
```

```
            </div>

            <div style="text-align: center; margin: 30px 0;">
              <a href="${examDetails.examLink || '#'}"
                 style="background-color: #28a745; color: white; padding: 15px
30px; text-decoration: none; border-radius: 5px; font-weight: bold; display:
inline-block;">
                Ø=Þ€ Start Exam Now
              </a>
            </div>

            <div style="background-color: #d1ecf1; border: 1px solid #bee5eb;
padding: 15px; border-radius: 5px; margin: 20px 0;">
              <h4 style="margin-top: 0; color: #0c5460;">Ø=ÜË Quick Checklist:</h4>
              <ul style="margin: 10px 0; padding-left: 20px; color: #0c5460;">
                <li>'  Stable internet connection</li>
                <li>'  Quiet environment</li>
                <li>'  Required materials ready</li>
                <li>'  Browser updated and tested</li>
              </ul>
            </div>

            <p style="color: #6c757d; font-size: 14px; margin-top: 30px;">
              Good luck with your exam! Ø<ß@<br>
              <strong>TrainCape Technology Team</strong>
            </p>
          </div>
        </div>
      `
    };

    await transporter.sendMail(mailOptions);
    console.log(`Exam reminder email sent to ${userEmail}`);
    return true;
  } catch (error) {
    console.error('Error sending exam reminder email:', error);
    return false;
  }
};

// Function to send WebSocket notification
const sendWebSocketNotification = (io, userId, examDetails) => {
  try {
    const notification = {
      type: 'exam-reminder',
      userId: userId,
      title: 'Ø=Þ¨ Exam Starting Soon!',
      message: `Your ${examDetails.course} exam starts in 10 minutes!`,
      examDetails: examDetails,
      sound: true,
      priority: 'high',
      timestamp: new Date().toISOString()
    };

    // Send to user's personal room
    io.to(`user-${userId}`).emit('exam-reminder', notification);
    console.log(`WebSocket notification sent to user ${userId}`);
    return true;
  } catch (error) {
```

```
    console.error('Error sending WebSocket notification:', error);
    return false;
  }
};

// Function to check for upcoming exams and send notifications
const checkUpcomingExams = async (io) => {
  try {
    console.log('Ø=Ý  Checking for upcoming exams...');

    const now = new Date();
    const tenMinutesFromNow = new Date(now.getTime() + 10 * 60 * 1000);
    const elevenMinutesFromNow = new Date(now.getTime() + 11 * 60 * 1000);

    // Find tasks that are exams and start between 10-11 minutes from now
    const upcomingExams = await Task.find({
      taskType: 'Exam',
      examDateTime: {
        $gte: tenMinutesFromNow,
        $lt: elevenMinutesFromNow
      },
      reminderSent: { $ne: true } // Only send reminder once
    }).populate('assignedTo', 'fullName email');

    console.log(`Ø=ÜË Found ${upcomingExams.length} upcoming exams`);

    for (const exam of upcomingExams) {
      if (!exam.assignedTo) {
        console.log(`& þ  Skipping exam ${exam._id} - no assigned user`);
        continue;
      }

      const examDetails = {
        course: exam.course,
        date: exam.examDateTime.toLocaleDateString(),
        time: exam.examDateTime.toLocaleTimeString(),
        location: exam.location || 'Online',
        examLink: exam.examLink || '#'
      };

      console.log(`Ø=Üç Sending notifications for exam: ${exam.course} to
${exam.assignedTo.fullName}`);

      // Send email notification
      const emailSent = await sendExamReminderEmail(
        exam.assignedTo.email,
        exam.assignedTo.fullName,
        examDetails
      );

      // Send WebSocket notification
      const socketSent = sendWebSocketNotification(io, exam.assignedTo._id,
examDetails);

      // Mark reminder as sent
      if (emailSent || socketSent) {
        await Task.findByIdAndUpdate(exam._id, { reminderSent: true });
        console.log(`'  Reminder sent for exam ${exam._id}`);
      }
```

```
    }

  } catch (error) {
    console.error(''L Error checking upcoming exams:', error);
  }
};

// Set up cron job to check every minute for upcoming exams
const startExamNotificationScheduler = (io) => {
  console.log('Ø=Þ€ Starting exam notification scheduler...');

  // Run every minute
  cron.schedule('* * * * *', () => {
    checkUpcomingExams(io);
  });

  // Also run once at startup
  setTimeout(() => {
    checkUpcomingExams(io);
  }, 5000); // Wait 5 seconds after startup
};

module.exports = {
  startExamNotificationScheduler,
  sendExamReminderEmail,
  sendWebSocketNotification,
  checkUpcomingExams
};
```

utils/fixEmailIndex.js

```
/**
 * Utility script to drop the unique email index from the leads collection
 *
 * Usage:
 * 1. Make sure MongoDB connection details are in .env
 * 2. Run with: node server/utils/fixEmailIndex.js
 */

const mongoose = require('mongoose');
const dotenv = require('dotenv');
dotenv.config();

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true
    });

    console.log(`MongoDB Connected: ${conn.connection.host}`);
    return conn;
  } catch (error) {
    console.error(`Error connecting to MongoDB: ${error.message}`);
    process.exit(1);
  }
};
```

```
const dropEmailIndex = async () => {
  try {
    console.log('Connecting to MongoDB...');
    const conn = await connectDB();

    console.log('Listing existing indexes on leads collection...');
    const indexes = await conn.connection.db.collection('leads').indexes();
    console.log('Current indexes:');
    indexes.forEach(index => {
      console.log(`- ${index.name}: ${JSON.stringify(index.key)}`);
    });

    // Find and drop any email index
    const emailIndexes = indexes.filter(index => index.key.email);
    if (emailIndexes.length > 0) {
      console.log(`Found ${emailIndexes.length} email indexes. Dropping them...`);
      for (const index of emailIndexes) {
        console.log(`Dropping index: ${index.name}`);
        await conn.connection.db.collection('leads').dropIndex(index.name);
        console.log(`Successfully dropped index: ${index.name}`);
      }
    } else {
      console.log('No email indexes found. No action needed.');
    }

    console.log('Operation complete. Disconnecting...');
    await mongoose.disconnect();
    console.log('Disconnected from MongoDB');
  } catch (error) {
    console.error(`Error: ${error.message}`);
    await mongoose.disconnect();
    process.exit(1);
  }
};

// Run the function
dropEmailIndex();
```

utils/reminderService.js

```
const Task = require('../models/Task');
const nodemailer = require('nodemailer');

// Configure nodemailer with environment variables
const transporter = nodemailer.createTransport({
  host: "smtp.hostinger.com",
  port: 465,
  secure: true,
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS
  }
});

/**
 * Sends an email notification
 * @param {Object} task - The task object
 * @param {string} reminderType - Type of reminder (30-minute-before, exam-time,
```

```
10-minute-after)
 * @param {Object} io - Socket.IO instance for WebSocket notifications
 */
const sendEmailNotification = async (task, reminderType = 'exam-time', io = null)
=> {
  try {
    // Check if email configuration is available
    if (!process.env.EMAIL_USER || !process.env.EMAIL_PASS) {
      console.error('Email configuration missing for exam reminders');
      return;
    }

    // Verify transporter configuration
    try {
      await transporter.verify();
      console.log('Email transporter verified for exam reminders');
    } catch (error) {
      console.error('Email transporter verification failed for exam reminders:',
error);
      return;
    }

    // Get customer and sales person details
    await task.populate([
      { path: 'customer', select: 'name NAME email E-MAIL contactNumber phone
MOBILE' },
      { path: 'salesPerson', select: 'fullName email' }
    ]);

    // Check if salesPerson is properly populated
    if (!task.salesPerson) {
      console.log('Sales person not found for task:', task._id);
      return;
    }

    // Handle both customer (Lead reference) and manualCustomer (embedded object)
    let customerData = null;
    if (task.customer) {
      // Customer is a Lead reference
      customerData = task.customer;
    } else if (task.manualCustomer && task.manualCustomer.name) {
      // Customer is manually entered data
      customerData = task.manualCustomer;
    }

    if (!customerData) {
      console.log('No customer data found for task:', task._id);
      console.log('Task customer field:', task.customer);
      console.log('Task manualCustomer field:', task.manualCustomer);
      return;
    }

    console.log(`Processing reminder for task ${task._id}:`);
    console.log(`- Customer type: ${task.customer ? 'Lead reference' : 'Manual
customer'}`);
    console.log(`- Customer name: ${customerData?.name || customerData?.NAME}`);
    console.log(`- Customer email: ${customerData?.email || customerData?.["E-
MAIL"] || 'No email'}`);
    console.log(`- Sales person: ${task.salesPerson?.fullName}
(${task.salesPerson?.email})`);
```

```
    // Get emails for notification
    const salesPersonEmail = task.salesPerson?.email;
    const customerEmail = customerData?.email || customerData?.["E-MAIL"];
    const customerName = customerData?.name || customerData?.NAME || 'Customer';
    const customerPhone = customerData?.contactNumber || customerData?.phone ||
customerData?.MOBILE || 'No contact number';

    if (!salesPersonEmail) {
      console.log('Sales person email not available, cannot send notifications.
Sales person:', task.salesPerson);
      return;
    }

    const examTime = new Date(task.examDate).toLocaleTimeString([], {
      hour: '2-digit',
      minute: '2-digit'
    });

    const examDate = new Date(task.examDate).toLocaleDateString([], {
      month: 'long',
      day: 'numeric',
      year: 'numeric'
    });

    // Customize content based on reminder type
    let salesPersonSubject, salesPersonContent, customerSubject, customerContent;

    switch (reminderType) {
      case '30-minute-before':
        salesPersonSubject = `UPCOMING: ${customerName}'s exam in 30 minutes
(${examTime})`;
        salesPersonContent = `
          <h1>Exam Starting Soon - 30 Minute Notice</h1>
          <p>Dear ${task.salesPerson.fullName},</p>
          <p>This is a reminder that ${customerName}'s exam is scheduled to start
in <strong>30 minutes</strong> at <strong>${examTime}</strong> on ${examDate}.</p>
          <p>Please ensure all preparations are complete and systems are ready.</
p>
          <p><strong>Exam Details:</strong> ${task.description || 'No additional
details provided'}</p>
          <p><strong>Contact:</strong> ${customerPhone}</p>
        `;

        customerSubject = `REMINDER: Your exam begins in 30 minutes`;
        customerContent = `
          <h1>Your Exam Starts Soon</h1>
          <p>Dear ${customerName},</p>
          <p>This is a reminder that your exam is scheduled to start in
<strong>30 minutes</strong> at <strong>${examTime}</strong>.</p>
          <p>Please ensure you are prepared and ready to begin.</p>
          <p>If you have any questions, please contact your sales representative:
${task.salesPerson.fullName}</p>
          <p>Best of luck with your exam!</p>
        `;
        break;

      case 'exam-time':
        salesPersonSubject = `ALERT: ${customerName}'s exam is starting now
(${examTime})`;
```

```
        salesPersonContent = `
          <h1>Exam Starting Now</h1>
          <p>Dear ${task.salesPerson.fullName},</p>
          <p>This is a notification that ${customerName}'s exam is scheduled to
start <strong>right now</strong> at <strong>${examTime}</strong>.</p>
          <p>Please ensure the exam process is initiated and the customer is
ready.</p>
          <p><strong>Exam Details:</strong> ${task.description || 'No additional
details provided'}</p>
          <p><strong>Contact:</strong> ${customerPhone}</p>
        `;

        customerSubject = `ALERT: Your exam is starting now`;
        customerContent = `
          <h1>Your Exam Is Starting</h1>
          <p>Dear ${customerName},</p>
          <p>Your exam is scheduled to start <strong>right now</strong> at
<strong>${examTime}</strong>.</p>
          <p>Please begin the exam process as instructed.</p>
          <p>If you have any technical issues, please contact your sales
representative immediately: ${task.salesPerson.fullName}</p>
          <p>Good luck!</p>
        `;
        break;

      case '10-minute-after':
        salesPersonSubject = `FOLLOW-UP: ${customerName}'s exam started 10
minutes ago`;
        salesPersonContent = `
          <h1>Exam Follow-up Reminder</h1>
          <p>Dear ${task.salesPerson.fullName},</p>
          <p>This is a follow-up notification for ${customerName}'s exam that
started 10 minutes ago at <strong>${examTime}</strong>.</p>
          <p>Please check if:</p>
          <ul>
            <li>The exam is progressing as expected</li>
            <li>The customer has encountered any issues</li>
            <li>You need to mark the task as completed when finished</li>
          </ul>
          <p><strong>Exam Details:</strong> ${task.description || 'No additional
details provided'}</p>
          <p><strong>Contact:</strong> ${customerPhone}</p>
        `;

        // No customer notification for the 10-minute-after reminder
        customerSubject = null;
        customerContent = null;
        break;

      default:
        salesPersonSubject = `Reminder: ${customerName}'s exam at ${examTime}`;
        salesPersonContent = `
          <h1>Exam Reminder</h1>
          <p>Dear ${task.salesPerson.fullName},</p>
          <p>This is a reminder that ${customerName}'s exam is scheduled for
today at <strong>${examTime}</strong>.</p>
          <p>Please ensure all preparations are complete.</p>
          <p><strong>Exam Details:</strong> ${task.description || 'No additional
details provided'}</p>
```

```
          <p><strong>Contact:</strong> ${customerPhone}</p>
        `;

        customerSubject = `Reminder: Your exam is scheduled for today at
${examTime}`;
        customerContent = `
          <h1>Exam Reminder</h1>
          <p>Dear ${customerName},</p>
          <p>This is a reminder that your exam is scheduled for today at
<strong>${examTime}</strong>.</p>
          <p>If you have any questions, please contact your sales representative:
${task.salesPerson.fullName}</p>
          <p>Best of luck with your exam!</p>
        `;
    }

    // Email to sales person (always sent regardless of customer email)
    await transporter.sendMail({
      from: process.env.EMAIL_FROM || 'shivam@traincapetech.in',
      to: salesPersonEmail,
      subject: salesPersonSubject,
      html: salesPersonContent
    });

    console.log(`${reminderType} reminder email sent to sales person:
${salesPersonEmail}`);

    // Only send to customer if they have an email and if there's content for
this reminder type
    if (customerEmail && customerSubject && customerContent) {
      // Email to customer
      await transporter.sendMail({
        from: process.env.EMAIL_FROM || 'shivam@traincapetech.in',
        to: customerEmail,
        subject: customerSubject,
        html: customerContent
      });
      console.log(`${reminderType} reminder email sent to customer:
${customerEmail}`);
    } else if (!customerEmail) {
      console.log('Customer email not available, skipping customer notification');
    } else {
      console.log(`No customer notification configured for ${reminderType}
reminder type`);
    }

    // Update the remindersSent array with type information
    task.remindersSent.push({
      sentAt: new Date(),
      reminderType: reminderType
    });
    await task.save();

    // Send WebSocket notification
    if (io) {
      const notification = {
        type: 'exam-reminder',
        userId: task.salesPerson._id,
        title: salesPersonSubject,
```

```
        message: salesPersonContent,
        examDetails: {
          course: task.description || 'Exam',
          customerName: customerName,
          date: new Date(task.examDate).toLocaleDateString(),
          time: examTime,
          location: 'As scheduled',
          taskId: task._id
        },
        sound: reminderType === 'exam-time' || reminderType === '30-minute-
before', // Sound for urgent reminders
        priority: 'medium',
        reminderType: reminderType,
        timestamp: new Date().toISOString()
      };

      // Send to sales person's personal room
      io.to(`user-${task.salesPerson._id}`).emit('exam-reminder', notification);
      console.log(`Ø=Ý  WebSocket notification sent to sales person
${task.salesPerson._id} for ${reminderType}`);
    }
  } catch (error) {
    console.error('Error sending email notification:', error);
  }
};

/**
 * Sends a WebSocket notification to the sales person
 * @param {Object} task - The task object
 * @param {string} reminderType - Type of reminder
 * @param {Object} io - Socket.IO instance
 */
const sendWebSocketNotification = (task, reminderType, io) => {
  try {
    if (!io) {
      console.log('Socket.IO not available for WebSocket notifications');
      return;
    }

    // Get customer data
    let customerData = null;
    if (task.customer) {
      customerData = task.customer;
    } else if (task.manualCustomer && task.manualCustomer.name) {
      customerData = task.manualCustomer;
    }

    if (!customerData || !task.salesPerson) {
      console.log('Missing customer or sales person data for WebSocket
notification');
      return;
    }

    const customerName = customerData?.name || customerData?.NAME || 'Customer';
    const examTime = new Date(task.examDate).toLocaleTimeString([], {
      hour: '2-digit',
      minute: '2-digit'
    });
```

```
    // Create notification based on reminder type
    let title, message, priority;

    switch (reminderType) {
      case '30-minute-before':
        title = '#ð Exam in 30 Minutes';
        message = `${customerName}'s exam starts in 30 minutes at ${examTime}`;
        priority = 'medium';
        break;
      case 'exam-time':
        title = 'Ø=Þ¨ Exam Starting Now!';
        message = `${customerName}'s exam is starting right now at ${examTime}`;
        priority = 'high';
        break;
      case '10-minute-after':
        title = 'Ø=ÜË Exam Follow-up';
        message = `${customerName}'s exam started 10 minutes ago - please check
progress`;
        priority = 'medium';
        break;
      default:
        title = 'Ø=ÜÅ Exam Reminder';
        message = `${customerName}'s exam is scheduled for ${examTime}`;
        priority = 'low';
    }

    const notification = {
      type: 'exam-reminder',
      userId: task.salesPerson._id,
      title: title,
      message: message,
      examDetails: {
        course: task.description || 'Exam',
        customerName: customerName,
        date: new Date(task.examDate).toLocaleDateString(),
        time: examTime,
        location: 'As scheduled',
        taskId: task._id
      },
      sound: reminderType === 'exam-time' || reminderType === '30-minute-
before', // Sound for urgent reminders
      priority: priority,
      reminderType: reminderType,
      timestamp: new Date().toISOString()
    };

    // Send to sales person's personal room
    io.to(`user-${task.salesPerson._id}`).emit('exam-reminder', notification);
    console.log(`Ø=Ý  WebSocket notification sent to sales person
${task.salesPerson._id} for ${reminderType}`);

  } catch (error) {
    console.error('Error sending WebSocket notification:', error);
  }
};

/**
 * Checks for exams scheduled today and sends reminders
 * This function is called every 10 minutes
```

```
 * @param {Object} io - Socket.IO instance for WebSocket notifications
 */
exports.processExamReminders = async (io = null) => {
  try {
    const now = new Date();
    const startOfDay = new Date(now);
    startOfDay.setHours(0, 0, 0, 0);

    const endOfDay = new Date(now);
    endOfDay.setHours(23, 59, 59, 999);

    // Find all upcoming exams and recent exams that haven't been completed
    // Look for exams up to 24 hours ahead and up to 1 hour in the past
    const upcomingExams = await Task.find({
      taskType: 'Exam',
      examDate: {
        $gte: new Date(now.getTime() - 60 * 60 * 1000), // 1 hour ago
        $lte: new Date(now.getTime() + 24 * 60 * 60 * 1000) // 24 hours from now
      },
      completed: false
    });

    console.log(`Found ${upcomingExams.length} upcoming exams to check for
reminders`);

    // For each exam, check if a reminder should be sent based on specific time
triggers
    for (const task of upcomingExams) {
      console.log(`Processing task ${task._id}, salesPerson: ${task.salesPerson},
customer: ${task.customer}`);

      const examTime = new Date(task.examDate);
      const minutesDifference = Math.round((examTime - now) / (1000 * 60));

      let shouldSendReminder = false;
      let reminderType = '';

      // Send reminders at these specific times:
      // 1. 30 minutes before the exam
      // 2. At exam time (0-5 minutes window)
      // 3. 10 minutes after exam start (reminder to mark complete)

      if (minutesDifference <= 30 && minutesDifference >= 25) {
        // 30 minutes before exam (with 5 min window)
        shouldSendReminder = true;
        reminderType = '30-minute-before';
      } else if (minutesDifference <= 5 && minutesDifference >= -5) {
        // At exam time (with 5 min window before and after)
        shouldSendReminder = true;
        reminderType = 'exam-time';
      } else if (minutesDifference <= -10 && minutesDifference >= -15) {
        // 10 minutes after exam (with 5 min window)
        shouldSendReminder = true;
        reminderType = '10-minute-after';
      }

      if (shouldSendReminder) {
        // Check if this specific reminder type was already sent
        const alreadySent = task.remindersSent.some(reminder => {
```

```
          // Get the sent date from the reminder object
          const sentDate = new Date(reminder.sentAt);
          const reminderTypeSent = reminder.reminderType;

          // If this is the same type of reminder...
          if (reminderTypeSent === reminderType) {
            // For the exam-time window, check if reminder was sent within last
10 minutes
            if (reminderType === 'exam-time') {
              return now.getTime() - sentDate.getTime() < 10 * 60 * 1000;
            }

            // For 30-minute-before, check if sent in last 30 minutes
            if (reminderType === '30-minute-before') {
              return now.getTime() - sentDate.getTime() < 30 * 60 * 1000;
            }

            // For 10-minute-after, check if sent in last 15 minutes
            if (reminderType === '10-minute-after') {
              return now.getTime() - sentDate.getTime() < 15 * 60 * 1000;
            }

            // For any other type, check if sent on the same day
            const sentDay = sentDate.toDateString();
            const today = now.toDateString();
            return sentDay === today;
          }

          return false; // Not the same type of reminder
        });

        if (!alreadySent) {
          console.log(`Sending ${reminderType} reminder for task ${task._id},
exam time: ${examTime}`);
          await sendEmailNotification(task, reminderType, io);
        }
      }
    }
  } catch (error) {
    console.error('Error processing exam reminders:', error);
  }
};
```