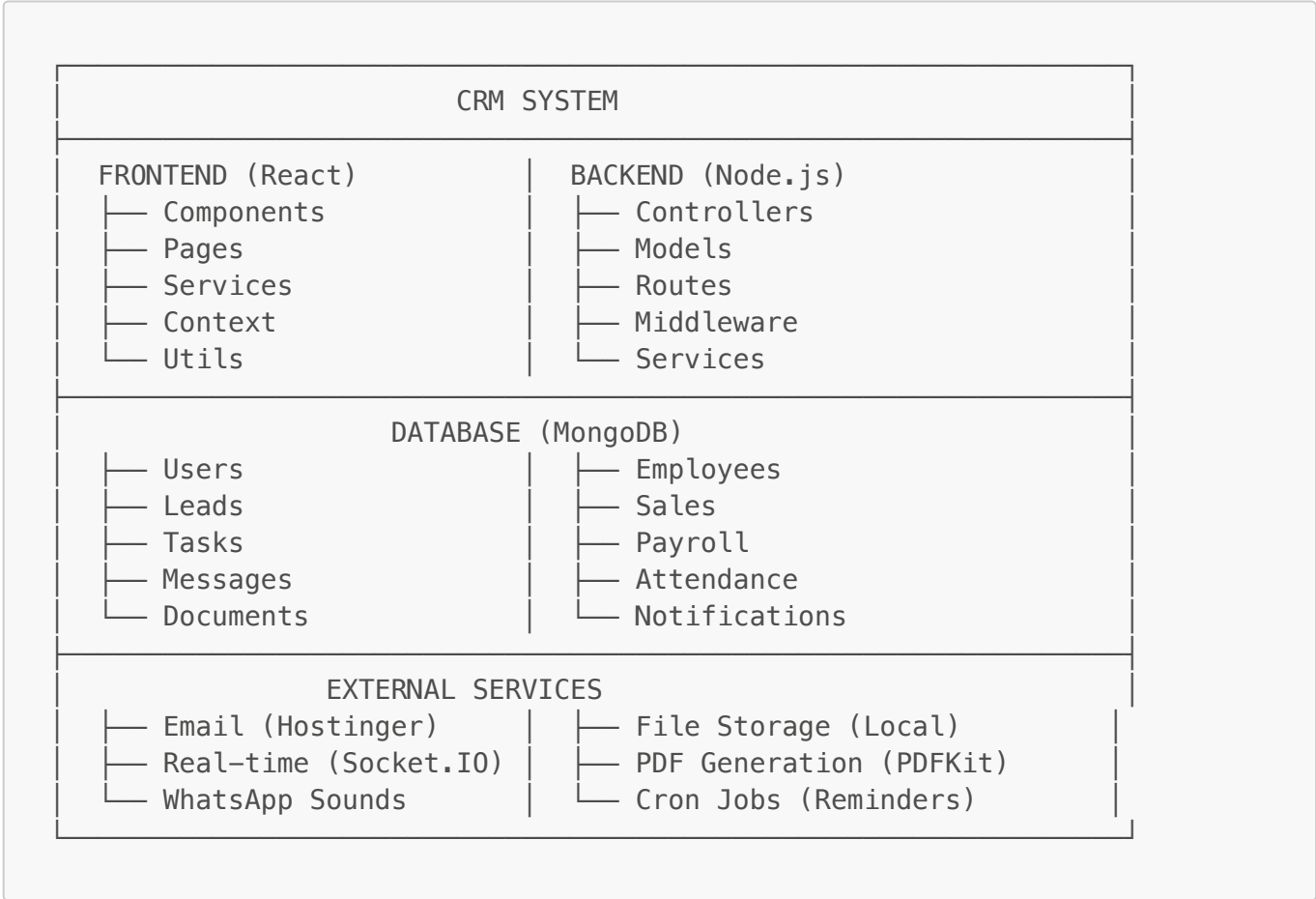
# 🏗️ COMPLETE CRM SYSTEM DOCUMENTATION

## 📋 TABLE OF CONTENTS


1. System Architecture Overview
2. Directory Structure
3. Backend Components
4. Frontend Components
5. Database Models
6. API Endpoints
7. Authentication Flow
8. Payroll System Workflow
9. Chat System Workflow
10. Email System Workflow
11. File Upload System
12. Real-time Features
13. Component Connections
14. Common Issues & Solutions


---

## 🏛️ SYSTEM ARCHITECTURE




---

## 📁 DIRECTORY STRUCTURE

### Backend Structure

```
server/
├── controllers/           # Business logic for API endpoints
│   ├── auth.js           # Authentication (login, register, OTP)
│   ├── user.js           # User management
│   ├── employee.js       # Employee CRUD operations
│   ├── lead.js           # Lead management
│   ├── task.js           # Task management
│   ├── payroll.js        # Payroll generation & management
│   ├── attendance.js     # Attendance tracking
│   ├── sales.js          # Sales management
│   ├── message.js        # Chat messages
│   └── notification.js   # Push notifications
├── models/               # MongoDB schemas
│   ├── User.js           # User data structure
│   ├── Employee.js       # Employee data structure
│   ├── Lead.js           # Lead data structure
│   ├── Task.js           # Task data structure
│   ├── Payroll.js        # Payroll data structure
│   ├── Attendance.js     # Attendance data structure
│   ├── Sales.js          # Sales data structure
│   ├── Message.js        # Chat message structure
│   └── Notification.js   # Notification structure
├── routes/               # API route definitions
│   ├── auth.js           # Authentication routes
│   ├── users.js          # User routes
│   ├── employees.js      # Employee routes
│   ├── leads.js          # Lead routes
│   ├── tasks.js          # Task routes
│   ├── payroll.js        # Payroll routes
│   ├── attendance.js     # Attendance routes
│   ├── sales.js          # Sales routes
│   ├── messages.js       # Chat routes
│   └── notifications.js  # Notification routes
├── middleware/           # Custom middleware
│   ├── auth.js           # JWT authentication
│   ├── role.js           # Role-based access
│   └── upload.js         # File upload handling
├── services/             # External service integrations
│   ├── emailService.js   # Email sending logic
│   ├── notificationService.js # Push notifications
│   └── fileService.js    # File handling
├── utils/                # Utility functions
│   ├── reminderService.js # Exam reminders
│   ├── examNotificationService.js # Exam notifications
│   └── helpers.js        # Common helper functions
├── assets/               # Static assets
│   └── images/           # Company logo, etc.
└── server.js             # Main server file
```

### Frontend Structure

```
client/
├── public/              # Static public files
├── src/
│   ├── components/      # Reusable components
│   │   ├── Employee/    # Employee-specific components
│   │   │   ├── PayrollComponent.jsx    # Payroll management
│   │   │   ├── AttendanceManagement.jsx # Attendance tracking
│   │   │   └── EmployeeList.jsx        # Employee listing
│   │   ├── Chat/        # Chat-specific components
│   │   │   ├── ChatContext.jsx         # Chat state management
│   │   │   ├── ChatSoundTest.jsx       # Sound testing
│   │   │   └── NotificationSettings.jsx # Notification settings
│   │   ├── Common/      # Common components
│   │   │   ├── Header.jsx              # Navigation header
│   │   │   ├── Sidebar.jsx             # Sidebar navigation
│   │   │   └── Layout.jsx              # Layout wrapper
│   │   └── Forms/       # Form components
│   │       ├── LoginForm.jsx           # Login form
│   │       ├── EmployeeForm.jsx        # Employee form
│   │       └── PayrollForm.jsx         # Payroll form
│   ├── pages/           # Page components
│   │   ├── LoginPage.jsx             # Login page
│   │   ├── Dashboard.jsx             # Main dashboard
│   │   ├── EmployeesPage.jsx         # Employee management
│   │   ├── PayrollPage.jsx           # Payroll management
│   │   ├── AttendancePage.jsx        # Attendance page
│   │   ├── LeadsPage.jsx             # Lead management
│   │   ├── SalesPage.jsx             # Sales management
│   │   ├── TasksPage.jsx             # Task management
│   │   └── ChatPage.jsx              # Chat interface
│   ├── context/         # React Context
│   │   ├── AuthContext.jsx           # Authentication state
│   │   ├── ChatContext.jsx           # Chat state
│   │   └── NotificationContext.jsx   # Notification state
│   ├── services/        # API services
│   │   ├── api.js                    # Main API service
│   │   ├── authService.js            # Authentication API
│   │   ├── employeeService.js        # Employee API
│   │   ├── payrollService.js         # Payroll API
│   │   └── chatService.js            # Chat API
│   ├── hooks/           # Custom React hooks
│   │   ├── useAuth.js                # Authentication hook
│   │   ├── useSocket.js              # Socket.IO hook
│   │   └── useNotifications.js       # Notifications hook
│   ├── utils/           # Utility functions
│   │   ├── notification-sounds.js    # WhatsApp-style sounds
│   │   ├── formatters.js             # Data formatting
│   │   └── validators.js             # Form validation
│   └── App.jsx          # Main app component
```

# 🔧 BACKEND COMPONENTS

## 1. Controllers (Business Logic)

### auth.js

```
// What it does: Handles user authentication
// Key functions:
- register()      // User registration
- login()         // User login
- sendOTPToEmail() // Password reset OTP
- verifyOTP()     // OTP verification
- resetPassword() // Password reset

// Connected to:
- Routes: /api/auth/*
- Models: User.js
- Services: emailService.js
```

### payroll.js

```
// What it does: Manages payroll operations
// Key functions:
- generatePayroll()     // Creates new payroll record
- getPayroll()         // Retrieves payroll records
- updatePayroll()      // Updates existing payroll
- approvePayroll()     // Approves payroll for payment
- generateSalarySlip() // Creates PDF salary slip

// Connected to:
- Routes: /api/payroll/*
- Models: Payroll.js, Employee.js, User.js
- Services: PDF generation
```

### employee.js

```
// What it does: Manages employee data
// Key functions:
- getAllEmployees()  // Get all employees
- getEmployeeById() // Get specific employee
- createEmployee()  // Create new employee
- updateEmployee()  // Update employee info
- deleteEmployee()  // Delete employee

// Connected to:
- Routes: /api/employees/*
```

```
- Models: Employee.js, User.js
- Frontend: EmployeeList.jsx
```

## 2. Models (Data Structure)

### Payroll.js

```
// What it stores:
- Employee details (employeeId, userId)
- Salary components (baseSalary, allowances, bonuses)
- Attendance data (workingDays, presentDays, absentDays)
- Calculated amounts (grossSalary, deductions, netSalary)
- Status (DRAFT, APPROVED, PAID, CANCELLED)

// Key methods:
- calculateSalary() // Calculates net salary
- Pre-save middleware // Auto-calculates on save

// Connected to:
- Controllers: payroll.js
- Frontend: PayrollComponent.jsx
```

### Employee.js

```
// What it stores:
- Personal info (fullName, email, phone, address)
- Employment details (employeeId, department, position)
- Salary info (salary, joinDate, status)
- Bank details (for payroll)

// Connected to:
- Models: User.js (reference)
- Controllers: employee.js, payroll.js
- Frontend: EmployeeList.jsx, PayrollComponent.jsx
```

## 3. Services (External Integrations)

### emailService.js

```
// What it does: Sends emails using multiple providers
// Supported providers:
- Hostinger (traincapetech.in)
- Gmail
- Outlook
- Yahoo
```

```
// Key functions:
- createTransporter()           // Creates email transporter
- sendPaymentConfirmationEmail() // Payment notifications
- sendServiceDeliveryEmail()    // Service completion emails

// Connected to:
- Controllers: All controllers that send emails
- Utils: reminderService.js
```

#### notificationService.js

```
// What it does: Manages push notifications
// Key functions:
- sendNotification()     // Sends notification
- broadcastNotification() // Broadcast to all users
- playSound()           // Plays WhatsApp-style sounds

// Connected to:
- Frontend: NotificationSettings.jsx
- Utils: notification-sounds.js
```

---

## 💻 FRONTEND COMPONENTS

### 1. Pages (Full Page Components)

#### Dashboard.jsx

```
// What it does: Main dashboard after login
// Features:
- Overview statistics
- Quick actions
- Recent activities
- Navigation to other pages

// Connected to:
- Context: AuthContext.jsx
- Services: api.js
- Components: Header.jsx, Sidebar.jsx
```

#### PayrollPage.jsx

```
// What it does: Main payroll management page
// Features:
- Lists all payroll records
- Filters by employee/month/year
```

```
- Actions: Generate, Edit, Approve, Download

// Connected to:
- Components: PayrollComponent.jsx
- Services: payrollService.js
- Context: AuthContext.jsx
```

## 2. Components (Reusable Components)

### PayrollComponent.jsx

```
// What it does: Handles all payroll operations
// Key features:
- Generate payroll form
- Edit existing payroll
- Approve payroll
- Download salary slips
- Manual allowances/deductions

// State management:
- generateForm: New payroll data
- editForm: Edit payroll data
- employees: Employee list
- payrollRecords: Payroll history

// Connected to:
- API: /api/payroll/*
- Services: payrollAPI
- Context: AuthContext.jsx
```

### ChatContext.jsx

```
// What it does: Manages chat state and real-time messaging
// Key features:
- Socket.IO connection
- Message history
- User status (online/offline)
- Sound notifications

// State management:
- messages: Chat messages
- users: Online users
- notifications: Unread counts

// Connected to:
- Services: Socket.IO
- Components: ChatPage.jsx
- Utils: notification-sounds.js
```

### 3. Context (Global State)

#### AuthContext.jsx

```
// What it provides:
- user: Current user data
- login(): Login function
- logout(): Logout function
- isAuthenticated: Login status

// Connected to:
- All protected components
- Services: authService.js
- Pages: LoginPage.jsx, Dashboard.jsx
```

---

## 🗃️ DATABASE MODELS

### User Collection

```
// Structure:
{
  _id: ObjectId,
  fullName: String,
  email: String (unique),
  password: String (hashed),
  role: ['Admin', 'HR', 'Manager', 'Employee'],
  isActive: Boolean,
  createdAt: Date,
  updatedAt: Date
}

// Relationships:
- One-to-One with Employee
- One-to-Many with Payroll
- One-to-Many with Messages
```

### Payroll Collection

```
// Structure:
{
  _id: ObjectId,
  employeeId: ObjectId (ref: Employee),
  userId: ObjectId (ref: User),
  month: Number (1-12),
  year: Number,
```

```
  baseSalary: Number,
  workingDays: Number,
  presentDays: Number,
  absentDays: Number,
  // Allowances
  hra: Number,
  da: Number,
  conveyanceAllowance: Number,
  medicalAllowance: Number,
  specialAllowance: Number,
  // Bonuses
  performanceBonus: Number,
  projectBonus: Number,
  attendanceBonus: Number,
  festivalBonus: Number,
  // Deductions
  pf: Number,
  esi: Number,
  tax: Number,
  loan: Number,
  other: Number,
  // Calculated fields
  grossSalary: Number,
  totalDeductions: Number,
  netSalary: Number,
  status: ['DRAFT', 'APPROVED', 'PAID', 'CANCELLED'],
  createdAt: Date,
  updatedAt: Date
}

// Relationships:
- Belongs to Employee
- Belongs to User
```

---

## 🔌 API ENDPOINTS

### Authentication Routes

```
POST /api/auth/register        # User registration
POST /api/auth/login           # User login
POST /api/auth/sendOTPToEmail  # Send password reset OTP
POST /api/auth/verifyOTP       # Verify OTP
POST /api/auth/resetPassword   # Reset password
```

### Payroll Routes

```
GET    /api/payroll            # Get all payroll records
POST   /api/payroll/generate   # Generate new payroll
```

```
GET    /api/payroll/:id        # Get specific payroll
PUT    /api/payroll/:id        # Update payroll
DELETE /api/payroll/:id        # Delete payroll
PUT    /api/payroll/:id/approve # Approve payroll
GET    /api/payroll/:id/salary-slip # Generate salary slip PDF
```

### Employee Routes

```
GET    /api/employees          # Get all employees
POST   /api/employees          # Create new employee
GET    /api/employees/:id      # Get specific employee
PUT    /api/employees/:id      # Update employee
DELETE /api/employees/:id      # Delete employee
```

---

## 🔐 AUTHENTICATION FLOW

```
1. User enters credentials → LoginPage.jsx
2. Form submission → authService.js → POST /api/auth/login
3. Server validates → auth.js controller → User.js model
4. JWT token generated → Sent to client
5. Client stores token → localStorage
6. AuthContext.jsx updates user state
7. Protected routes now accessible
8. JWT sent with each API request → auth.js middleware
9. Server validates token → Allows/denies access
```

---

## 💰 PAYROLL SYSTEM WORKFLOW

### Complete Payroll Flow

```
1. HR/Admin opens PayrollPage.jsx
2. Clicks "Generate Payroll" → PayrollComponent.jsx
3. Fills form with:
   - Employee selection
   - Month/Year
   - Base Salary
   - Working Days (e.g., 31 for January)
   - Present Days (e.g., 1 for mostly absent)
   - Manual allowances (HRA, DA, etc.)
   - Manual bonuses
   - Manual deductions (PF, ESI, etc.)
4. Form submits → payrollAPI.generate()
5. Backend receives → payroll.js controller
6. Data validation → Payroll.js model
```

```
7. Salary calculation:
   - calculatedSalary = (baseSalary / workingDays) * presentDays
   - grossSalary = calculatedSalary + allowances + bonuses
   - netSalary = grossSalary - deductions
8. Record saved → MongoDB
9. Frontend updates → Shows new payroll record
10. HR can approve → PUT /api/payroll/:id/approve
11. Generate salary slip → PDF with company logo
12. Download → Employee receives salary slip
```

### Salary Calculation Logic

```
// Example: Employee absent 30 days out of 31
const baseSalary = 30000;
const workingDays = 31;
const presentDays = 1;

// Step 1: Calculate prorated basic salary
const calculatedSalary = (baseSalary / workingDays) * presentDays;
// calculatedSalary = (30000 / 31) * 1 = 967.74

// Step 2: Add allowances and bonuses
const grossSalary = calculatedSalary + hra + da + bonuses;

// Step 3: Subtract deductions
const netSalary = grossSalary - pf - esi - tax - loans;

// Result: Employee gets ₹967.74 (not ₹30,000)
```

---

## 💬 CHAT SYSTEM WORKFLOW

### Real-time Chat Flow

```
1. User opens ChatPage.jsx
2. ChatContext.jsx initializes Socket.IO connection
3. Server accepts connection → server.js
4. User joins room → Socket.IO rooms
5. User sends message → ChatContext.jsx
6. Message sent to server → Socket.IO
7. Server broadcasts → All users in room
8. Other users receive → ChatContext.jsx
9. UI updates → New message appears
10. Sound notification → notification-sounds.js
11. WhatsApp-style sound plays
```

### Notification System

```
1. New message received → ChatContext.jsx
2. Check if user is active → document.hasFocus()
3. If inactive → Play sound + Browser notification
4. Update unread count → UI badge
5. Store in localStorage → Persistence
```

---

## 📧 EMAIL SYSTEM WORKFLOW

### Email Configuration Flow

```
1. Sales person updates sale → SalesPage.jsx
2. Email needed → emailService.js
3. Detect domain → getEmailConfig()
4. Select provider:
   - traincapetech.in → Hostinger SMTP
   - gmail.com → Gmail SMTP
   - outlook.com → Outlook SMTP
5. Create transporter → nodemailer
6. Send email → Customer receives
7. CC sales person → Sales person receives copy
```

### Email Template System

```
1. Payment confirmation → getPaymentConfirmationTemplate()
2. Service delivery → getServiceDeliveryTemplate()
3. Exam reminders → reminderService.js
4. OTP emails → auth.js controller
```

---

## 📁 FILE UPLOAD SYSTEM

### Document Upload Flow

```
1. User selects file → File input
2. Form submission → multer middleware
3. File validation → Size, type checks
4. File saved → server/uploads/
5. Path stored → Database
6. Frontend updates → File list
7. Download → Secure file serving
```

---

## ⚡ REAL-TIME FEATURES

### Socket.IO Implementation

```
// Server side - server.js
io.on('connection', (socket) => {
  // User joins
  socket.on('join', (userId) => {
    socket.join(`user-${userId}`);
  });

  // Message handling
  socket.on('send-message', (data) => {
    io.to(`user-${data.recipientId}`).emit('receive-message', data);
  });

  // Status updates
  socket.on('user-status', (status) => {
    socket.broadcast.emit('user-status-update', status);
  });
});

// Client side - ChatContext.jsx
const socket = io(API_BASE_URL);

socket.on('receive-message', (message) => {
  setMessages(prev => [...prev, message]);
  playNotificationSound();
});
```

---

## 🔗 COMPONENT CONNECTIONS

### Authentication Chain

```
LoginPage.jsx → AuthContext.jsx → authService.js → /api/auth/login →
auth.js → User.js → MongoDB
```

### Payroll Chain

```
PayrollPage.jsx → PayrollComponent.jsx → payrollAPI → /api/payroll/* →
payroll.js → Payroll.js → MongoDB
```

### Chat Chain

```
ChatPage.jsx → ChatContext.jsx → Socket.IO → server.js → Message.js →
MongoDB
```

### Employee Chain

```
EmployeesPage.jsx → EmployeeList.jsx → employeeService.js →
/api/employees/* → employee.js → Employee.js → MongoDB
```

---

## 🐛 COMMON ISSUES & SOLUTIONS

### 1. Salary Calculation Issues

```
// Problem: Full salary for absent employees
// Solution: Check working days calculation

// Wrong:
const salary = baseSalary; // Always full salary

// Correct:
const salary = (baseSalary / workingDays) * presentDays;
```

### 2. Email Configuration Issues

```
// Problem: Domain not found error
// Solution: Use correct domain

// Wrong:
from: 'noreply@traincapecrm.com' // Domain doesn't exist

// Correct:
from: 'noreply@traincapetech.in' // Valid domain
```

### 3. Authentication Issues

```
// Problem: Token expiry
// Solution: Implement token refresh

// Check token expiry
const isTokenExpired = (token) => {
  const payload = JSON.parse(atob(token.split('.')[1]));
  return payload.exp < Date.now() / 1000;
};
```

### 4. File Upload Issues

```
// Problem: File size limits
// Solution: Configure multer properly

const upload = multer({
  limits: { fileSize: 10 * 1024 * 1024 }, // 10MB
  fileFilter: (req, file, cb) => {
    const allowedTypes = ['image/jpeg', 'image/png', 'application/pdf'];
    if (allowedTypes.includes(file.mimetype)) {
      cb(null, true);
    } else {
      cb(new Error('Invalid file type'));
    }
  }
});
```

---

## 🚀 DEPLOYMENT WORKFLOW

### Production Deployment

```
# 1. Build frontend
cd client
npm run build

# 2. Copy build to server
cp -r build/* /var/www/crm/

# 3. Install server dependencies
cd server
npm install --production

# 4. Set environment variables
cp .env.example .env
# Edit .env with production values

# 5. Start server
pm2 start server.js --name crm-server

# 6. Setup Nginx
# Configure reverse proxy for API
# Serve static files from build
```

---

## 📊 TESTING

### Test Salary Calculation

```
# Run salary calculation test
node test-salary-calculation.js

# Expected output:
# ✅ All tests passed
# ✅ Proper proration for absent employees
```

### Test Email Configuration

```
# Run email test
node test-email.js

# Expected output:
# ✅ Email sent successfully
# ✅ Customer receives email
```

---

## 🔍 DEBUGGING TIPS

### 1. Check Console Logs

```
// Backend logs
console.log('💰 Salary calculation:', calculatedSalary);

// Frontend logs
console.log('📊 Form data:', formData);
```

### 2. Database Queries

```
// Check MongoDB directly
use crm_database;
db.payrolls.find({employeeId: ObjectId('...')});
```

### 3. Network Requests

```
// Check API calls in browser DevTools
// Network tab → Filter by XHR
// Check request/response data
```

---

## 🎯 CONCLUSION

This CRM system is a comprehensive solution with:

- **Modular Architecture**: Clear separation of concerns
- **Real-time Features**: Socket.IO for instant messaging
- **Payroll Management**: Complete salary calculation system
- **Email Integration**: Multi-provider email support
- **File Management**: Document upload and storage
- **Authentication**: JWT-based security
- **Responsive Design**: Mobile-friendly interface

Each component is designed to work independently while maintaining seamless integration with the overall system.

---

🚀 **To run tests:**

```
# Test salary calculation
node test-salary-calculation.js

# Test email configuration
node test-email.js

# Start development server
npm run dev
```

📞 **For support or questions about any component, refer to this documentation or check the respective file's comments.**