CRM System Documentation

High Level Design & Low Level Design

Generated on: 2025-09-02 10:38:17

Production Ready Documentation

Table of Contents

1. Project Overview

A production-ready mini CRM system with authentication, role-based access control, lead/customer management, and analytics dashboard.

2. High Level Design (HLD)

2.1 System Architecture

The system follows a modern 3-tier architecture:

```
[React Frontend] <-> [Node.js API Gateway] <-> [MongoDB Database]
       |                       |
[Browser Storage]       [External Services]
```

2.2 Technology Stack

```
Frontend: React 18, Vite, Tailwind CSS, Zustand, React Router
Backend: Node.js, Express.js, MongoDB, Mongoose, JWT, bcrypt
Development: ESLint, Prettier, Nodemon
Deployment: Docker, PM2, Nginx (optional)
```

2.3 Microservices Architecture

The system is structured as modular microservices:

```
- Authentication Service: JWT token management, user sessions
- User Management Service: CRUD operations with RBAC
- Lead Service: Lead lifecycle management
- Customer Service: Customer management and notes
- Task Service: Task tracking and assignments
- Analytics Service: Dashboard data aggregation
- Activity Service: Audit logging and activity tracking
```

2.4 Data Flow

Request flow through the system:

```
1. Client -> API Gateway (with JWT validation)
2. API Gateway -> Microservice (based on route)
3. Microservice -> Database (with Mongoose)
4. Database -> Microservice -> API Gateway -> Client
```

2.5 Security Architecture

```
- JWT-based Authentication (Access + Refresh tokens)
- Role-Based Access Control (Admin/Agent levels)
```

```
- HTTPS Encryption for all communications
- CORS Configuration for frontend-backend communication
- Input Validation at API boundaries
- Password hashing with bcrypt (salt rounds: 10)
```

3. Low Level Design (LLD)

3.1 Database Schema Design

User Model

```
{
  _id: ObjectId,
  name: String,        // User's full name
  emailId: String,     // Unique email
  password: String,    // Hashed password
  role: String,        // 'Admin' or 'Agent'
  createdAt: Date,
  updatedAt: Date
}

Indexes:
- { emailId: 1 } (unique)
- { role: 1 }
- { createdAt: -1 }
```

Lead Model

```
{
  _id: ObjectId,
  name: String,
  emailId: String,
  phone: String,
  status: String,      // ['New', 'In Progress', 'Closed Won', 'Closed Lost']
  source: String,
  assignedAgent: ObjectId,  // Reference to User
  isArchived: Boolean,
  createdAt: Date,
  updatedAt: Date
}

Indexes:
- { assignedAgent: 1 }
```

```
- { status: 1 }
- { createdAt: -1 }
- { emailId: 1 } (sparse)
```

Customer Model

```
{
  _id: ObjectId,
  name: String,
  company: String,
  emailId: String,
  phone: String,
  notes: [{
    text: String,
    createdBy: ObjectId,    // Reference to User
    createdAt: Date
  }],
  tags: [String],
  owner: ObjectId,          // Reference to User (Agent)
  deals: [ObjectId],        // Reference to Deals (if implemented)
  isArchived: Boolean,
  createdAt: Date,
  updatedAt: Date
}

Indexes:
- { owner: 1 }
- { tags: 1 }
- { emailId: 1 } (sparse)
- { createdAt: -1 }
```

3.2 API Specifications

Authentication Endpoints

```
POST /api/auth/login
Request: { emailId: string, password: string }
Response: { user: User, token: string }

POST /api/auth/register (Admin only)
Request: { name: string, emailId: string, password: string, role: string }
Response: { message: string }

POST /api/auth/refresh
Request: {} (with refresh token cookie)
Response: { user: User, token: string }
```

```
        POST /api/auth/logout
        Response: { message: string }
```

Lead Management Endpoints

```
        GET /api/leads
        Query Params: ?status=New&search=john&page=1&limit=10&isArchived=false
        Response: { leads: Lead[], totalLeads: number, page: number, totalPages: number
}

        POST /api/leads
        Request: { name: string, emailId: string, phone?: string, status: string,
source?: string, assignedAgent?: string }
        Response: Lead

        PATCH /api/leads/:id
        Request: { status?: string, assignedAgent?: string, ... }
        Response: Lead

        DELETE /api/leads/:id (soft delete)
        Response: { message: string }

        POST /api/leads/:id/convert
        Response: { message: string, customer: Customer }
```

3.3 Component Architecture

Frontend component structure:

```
        App
        -> Router
           -> Layout
              -> Sidebar
              -> Navbar
              -> MainContent
                 -> Dashboard
                 -> Leads
                    -> LeadList
                    -> LeadFilters
                    -> LeadModal
                    -> Pagination
                 -> Customers
                 -> Tasks
                 -> Users (Admin only)
           -> Login
        -> Providers (AuthStore, Theme, etc.)
```

State Management

```
// authStore.js
{
  user: null | { id: string, name: string, emailId: string, role: string },
  token: null | string,
  loading: boolean,
  actions: {
    setAuth: (user: User, token: string) => void,
    clearAuth: () => void,
    refreshToken: () => Promise<void>
  }
}
```

3.4 Security Implementation

JWT Strategy

```
// Token generation
const accessToken = jwt.sign(
  { userId: user._id, role: user.role },
  process.env.JWT_SECRET,
  { expiresIn: '15m' }
);

const refreshToken = jwt.sign(
  { userId: user._id },
  process.env.JWT_REFRESH_SECRET,
  { expiresIn: '7d' }
);

// Stored in HTTP-only cookie for security
res.cookie('refreshToken', refreshToken, {
  httpOnly: true,
  secure: process.env.NODE_ENV === 'production',
  sameSite: 'strict',
  maxAge: 7 * 24 * 60 * 60 * 1000 // 7 days
});
```

RBAC Middleware

```
const requireRole = (roles: string[]) => {
  return (req: Request, res: Response, next: NextFunction) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ error: 'Insufficient permissions' });
```

```
    }
    next();
  };
};

// Usage
router.get('/admin-data', requireRole(['Admin']), (req, res) => {
  // Admin only data
});
```

3.5 Performance Optimization

Database Optimization

```
// Pagination implementation
const page = parseInt(req.query.page) || 1;
const limit = parseInt(req.query.limit) || 10;
const skip = (page - 1) * limit;

// Selective field population and projection
const leads = await LeadModel.find(filters)
  .populate('assignedAgent', 'name emailId')
  .select('-__v') // Exclude unnecessary fields
  .skip(skip)
  .limit(limit)
  .lean(); // Faster read-only queries

// Aggregation for analytics
const leadsByStatus = await LeadModel.aggregate([
  { $match: filters },
  { $group: { _id: '$status', count: { $sum: 1 } } }
]);
```

Frontend Optimization

```
// React Query for data caching and synchronization
const { data: leads, isLoading, error } = useQuery({
  queryKey: ['leads', filters, page],
  queryFn: () => fetchLeads(filters, page),
  staleTime: 5 * 60 * 1000, // 5 minutes cache
  retry: 2,
});

// Debounced search inputs
const debouncedSearch = useDebounce(searchTerm, 300);
```

```
        // Virtualized lists for large datasets
        <VirtualList
          items={leads}
          itemHeight={80}
          renderItem={(lead) => <LeadCard lead={lead} />}
          overscan={5}
        />
```

3.6 Error Handling

Backend Error Handling

```
        // Unified error middleware
        const errorHandler = (err: Error, req: Request, res: Response, next:
NextFunction) => {
          console.error('Error:', err);

          // Mongoose validation error
          if (err.name === 'ValidationError') {
            return res.status(400).json({
              error: 'Validation Error',
              details: Object.values(err.errors).map(e => e.message)
            });
          }

          // JWT errors
          if (err.name === 'JsonWebTokenError') {
            return res.status(401).json({ error: 'Invalid token' });
          }

          if (err.name === 'TokenExpiredError') {
            return res.status(401).json({ error: 'Token expired' });
          }

          // MongoDB duplicate key
          if (err.code === 11000) {
            return res.status(400).json({ error: 'Duplicate field value' });
          }

          // Default error
          res.status(err.status || 500).json({
            error: err.message || 'Internal Server Error'
          });
        };
```

Frontend Error Handling

```
// Error Boundary for React components
class ErrorBoundary extends React.Component {
  state = { hasError: false, error: null };

  static getDerivedStateFromError(error) {
    return { hasError: true, error };
  }

  componentDidCatch(error, errorInfo) {
    console.error('Error caught by boundary:', error, errorInfo);
    // Log to error reporting service (Sentry, LogRocket)
  }

  render() {
    if (this.state.hasError) {
      return <ErrorFallback error={this.state.error} />;
    }
    return this.props.children;
  }
}

// API error handling
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      retry: (failureCount, error) => {
        if (error.status === 404) return false;
        if (error.status === 403) return false;
        return failureCount < 2;
      },
    },
  },
});
```

4. Deployment Architecture

Production Environment

```
[Cloud Provider (AWS/Azure/DigitalOcean)]
  |
  +-- [Frontend (CDN)]
  |      |
  |      +-- [Browser Storage]
  |
  +-- [Backend (Node.js)]
  |      |
  |      +-- [Database (MongoDB)]
```

```
  |
  +-- [Monitoring (Prometheus)]
  |
  +-- [Logging (ELK Stack)]
```

Docker Configuration

```
# Dockerfile for Backend
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
EXPOSE 3000
USER node
CMD ["npm", "start"]

# Dockerfile for Frontend
FROM node:18-alpine as build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
```

Environment Variables

```
# Backend (.env)
NODE_ENV=production
PORT=3000
MONGODB_URI=mongodb://localhost:27017/crm
JWT_SECRET=your_secure_jwt_secret_here
JWT_REFRESH_SECRET=your_secure_refresh_secret_here
FRONTEND_URL=https://yourdomain.com
CORS_ORIGIN=https://yourdomain.com

# Frontend (.env)
VITE_API_BASE_URL=https://api.yourdomain.com
VITE_APP_NAME=CRM System
```

5. Testing Strategy

Backend Testing

```
// Unit tests with Jest
describe('User Model', () => {
  it('should create user with valid data', async () => {
    const user = new UserModel(validUserData);
    await expect(user.save()).resolves.toBeDefined();
  });
});

// Integration tests with Supertest
describe('POST /api/auth/login', () => {
  it('should return token with valid credentials', async () => {
    const response = await request(app)
      .post('/api/auth/login')
      .send({ emailId: 'test@example.com', password: 'password123' });

    expect(response.status).toBe(200);
    expect(response.body.token).toBeDefined();
  });
});

// Test coverage goals:
- 80%+ line coverage
- All API endpoints tested
- All error cases covered
- Authentication flows tested
```

Frontend Testing

```
// Component tests with React Testing Library
describe('Login Component', () => {
  it('should show validation errors for empty form', async () => {
    render(<Login />);
    fireEvent.click(screen.getByText('Login'));
    expect(await screen.findByText('Email is required')).toBeInTheDocument();
  });
});

// E2E tests with Cypress
describe('User Flow', () => {
  it('should allow admin to create new user', () => {
    cy.loginAsAdmin();
    cy.visit('/users');
    cy.get('button[aria-label="Add User"]').click();
```

```
    // Fill form and assert success
  });
});

// Test coverage goals:
- All components rendered correctly
- User interactions work as expected
- Form validation working
- API calls mocked and tested
```

6. Monitoring & Maintenance

Performance Monitoring

```
// Backend monitoring with Prometheus
const prometheus = require('prom-client');
const collectDefaultMetrics = prometheus.collectDefaultMetrics;
collectDefaultMetrics({ timeout: 5000 });

// Custom metrics
const httpRequestDuration = new prometheus.Histogram({
  name: 'http_request_duration_seconds',
  help: 'Duration of HTTP requests in seconds',
  labelNames: ['method', 'route', 'status_code'],
});

// Frontend monitoring with Sentry
import * as Sentry from '@sentry/react';
Sentry.init({
  dsn: process.env.VITE_SENTRY_DSN,
  environment: process.env.NODE_ENV,
});

// Logging with Winston
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' }),
  ],
});
```

Health Checks

```
// API health endpoint
app.get('/health', (req, res) => {
  res.json({
    status: 'OK',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    memory: process.memoryUsage(),
  });
});

// Database health check
app.get('/health/db', async (req, res) => {
  try {
    await mongoose.connection.db.admin().ping();
    res.json({ database: 'connected' });
  } catch (error) {
    res.status(500).json({ database: 'disconnected', error: error.message });
  }
});
```