KEN PROJECT

BACKEND TASK:

DEVELOPER A:(Ahsan) Senior Full Stack Developer

TASK:01:

Advanced CRM Backend: Features, Integrations, and How It Works

1. Authentication & User Management

JWT-based authentication for all users (admin, agent, client).

Role-based access control via middleware (roleMiddleware), restricting endpoints by user type.

User-specific data filtering: Middleware ensures users only access their own or permitted data (e.g., agents see only their clients).

User management endpoints: Create, update, delete, and list users, with roles and permissions.

2. Client & Lead Management

Client and lead controllers: Full CRUD, assignment, scoring (hot/warm/cold), and timeline tracking.

Self-registration and public lead capture: Open endpoints for new clients/leads, with optional email verification and agent assignment.

3. Advanced Notifications

Real-time notifications: Integrated with SMS providers (e.g., Twilio), email (e.g., SendGrid), and push (e.g., Firebase).

Notification triggers: Configurable for key events (new lead, invoice paid, task due, etc.).

User preferences: Each user can set preferred notification channels.

Notification filtering: Users can filter which types of notifications they receive.

Notification model: Stores all sent notifications for audit and user review.

4. Audit Logging

Comprehensive audit logs: Every sensitive action (login, data change, permission change, etc.) is logged with user, timestamp, and action details.

Audit log endpoints: Admins can view, filter, and export logs.

Middleware-based logging: Ensures all relevant routes/actions are covered.

5. Analytics & Reporting

Advanced analytics endpoints: Aggregations for sales, leads, conversions, agent performance, etc.

Custom filters: By date, user, group, territory, and more.

Export options: CSV, PDF, and dashboard visualizations.

Scheduled reports: Automated weekly/monthly analytics sent to admins.

6. Calendar Integrations

Google Calendar integration: OAuth2 flow, token storage, event sync, and webhook handling.

Outlook Calendar integration: Microsoft OAuth2, token storage, event sync, and webhook handling.

Booking links: Integration with services like Calendly or custom booking logic, with unique URLs for each user/agent.

User model: Stores calendar tokens and booking links.

7. Other Advanced Features

Saved replies/canned responses: User-specific, full CRUD.

Advanced tagging: Color-coded, user-specific tags for clients, leads, and tasks.
Invoices/payment status: Full invoice lifecycle, status updates, and reporting.
Reminders: Recurring, multi-channel, user-specific reminders with notification triggers.
Messaging restrictions: Only allow messaging within shared groups.
Group/enterprise dashboards: Aggregated stats and member details for each group.
White-label branding/affiliate: Organization-specific branding and affiliate rates.
8. Code Quality & Fixes
Error handling: All endpoints have robust error handling and clear responses.
Validation: Input validation for all models and endpoints.
Consistent user-specific logic: All data access is filtered by user/role.
Modular structure: Controllers, models, routes, and middleware are cleanly separated.
🛠️ How It All Works
Express.js serves as the API framework.
MongoDB/Mongoose for all data storage and advanced queries.
Middleware for authentication, authorization, user-specific filtering, audit logging, and notifications.
Jobs for scheduled tasks (reminders, analytics reports).
External integrations for calendar, notifications, and booking.
All endpoints are protected, validated, and return clear, actionable responses.
🚦 Summary Results
Every feature is user-specific and secure.
Notifications and audit logs are real-time, filterable, and comprehensive.
Analytics and reporting are advanced, customizable, and exportable.
Calendar and booking integrations are ready for production use.

DEVELOPER B:(Arbab) Senior Mern Stack Developer

TASK:01:

Developer B – Properties, Transactions, Commission System
Responsibilities:
Property listing creation and management
Deal/transaction tracking with timeline and people
Commission calculation (price × rate)
File and notes handling within transactions
Endpoints to Create:
POST /properties, GET /properties, PUT /properties/:id
POST /transactions, GET /transactions/:agentId
PUT /transactions/:id (update deal stage, dates, notes)
Commission calculation logic built-in
Schemas Owned:
Property.js
Transaction.js

```
// ---------- Developer B: Properties, Transactions, Commission ----------
// Property.js
const propertySchema = new mongoose.Schema({
  title: String,
  location: String,
  price: Number,
```

```
  type: String,
  description: String,
  listedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  createdAt: { type: Date, default: Date.now },
});
module.exports = mongoose.model('Property', propertySchema);
// Transaction.js
const transactionSchema = new mongoose.Schema({
  property: { type: mongoose.Schema.Types.ObjectId, ref: 'Property' },
  buyer: { type: mongoose.Schema.Types.ObjectId, ref: 'Client' },
  seller: { type: mongoose.Schema.Types.ObjectId, ref: 'Client' },
  agent: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  price: Number,
  commissionRate: Number,
  commissionAmount: Number,
  dealStage: {
    type: String,
    enum: ['contacted', 'meeting', 'offer', 'accepted', 'closed'],
    default: 'contacted'
  },
  participants: {
    attorney: String,
    buyerAgent: String,
    sellerAgent: String,
    inspector: String
  },
  importantDates: {
    depositDue: Date,
    inspection: Date,
    appraisal: Date,
    closing: Date
  },
  notes: String,
  files: [String], // file names or URLs
  createdAt: { type: Date, default: Date.now },
});
module.exports = mongoose.model('Transaction', transactionSchema);
Syed Ahsan  [7:59 PM]
// ---------- Developer A: Auth, Client CRM, Lead Intake ----------
// User.js
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  name: String,
  email: { type: String, unique: true },
  password: String,
  role: { type: String, enum: ['admin', 'agent', 'client'], default: 'client' },
  createdAt: { type: Date, default: Date.now },
});
```

```
module.exports = mongoose.model('User', userSchema);
// Client.js
const clientSchema = new mongoose.Schema({
  name: String,
  email: String,
  phone: String,
  agent: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  status: { type: String, enum: ['new', 'contacted', 'qualified', 'closed'], default: 'new' },
  score: { type: String, enum: ['hot', 'warm', 'cold'], default: 'cold' },
  leadSource: String,
  tags: [String],
  type: { type: String, enum: ['buyer', 'seller', 'renter'], default: 'buyer' },
  notes: String,
  lastContacted: Date,
  createdAt: { type: Date, default: Date.now },
});
module.exports = mongoose.model('Client', clientSchema);
// IntakeForm.js (optional if separate)
const intakeFormSchema = new mongoose.Schema({
  name: String,
  email: String,
  phone: String,
  formType: String,
  preApproved: Boolean,
  preferences: Object,
  linkedClient: { type: mongoose.Schema.Types.ObjectId, ref: 'Client' },
  assignedTo: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
});
module.exports = mongoose.model('IntakeForm', intakeFormSchema);
```

TASK:02:

3. Role-Based Access Middleware
Protect routes like:
js
router.get('/admin/users', authorizeRoles('admin'));
4.
Admin Agent Management
POST /admin/add-agent
PATCH /admin/update-agent/:id
DELETE /admin/remove-agent/:id
GET /admin/agents → full list with filters/search
5. Dynamic Lead Assignment Rules
Auto-assign by:
Source (e.g. Facebook, Web form)
Region
Load balancing
Can use:
POST /rules/lead-assignment

GET /rules
6. Audit Log
Model:
js
```
{
  userId,
  action: "UPDATE_CLIENT",
  model: "Client",
  recordId,
  timestamp,
  details: "Updated client email"
}
```
GET /logs?user=...&date=...
7. Client Activity Timeline
GET /client/:id/timeline
Combines:
Messages
Property views
Form submissions
Stage changes
8. Bulk Actions
POST /clients/bulk-update-status
POST /messages/bulk-send
POST /assignments/bulk-assign
9. Referral Source Tracking
Add referrerId or source field to client schema
Filter reports by source
GET /leads?source=referral
10. Commission Report Generator
GET /reports/commissions?agentId=&month=
Aggregates deal amounts × agent split %
Exports to CSV/PDF (optional)
11. Smart Duplicate Detection
On POST /clients, check:
Email match
Phone number match
Fuzzy name match
12. Webhook/Event Listener (For API sync)
If calendar/email/API updates need syncing back
Example:
/webhooks/calendar
/webhooks/mailchimp

DEVELOPER C:(Ahmer) Senior Backend Developer

TASK:01:

Task Management, Messaging, Campaigns
Responsibilities:

Task creation, scheduling, and reminders
Real-time/delayed messaging system (agent :left_right_arrow: client)
Post-sale email campaign scheduling
Dashboard aggregation logic
Endpoints to Create:
POST /tasks, GET /tasks/:agentId, PUT /tasks/:id/complete
POST /messages, GET /messages/:conversationId
POST /campaigns, GET /campaigns/:clientId
GET /dashboard/:agentId (summary of tasks, deals, leads)
Schemas Owned:
Task.js
Message.js
Campaign.js

```
// ---------- Developer C: Tasks, Messaging, Campaigns ----------
// Task.js
const taskSchema = new mongoose.Schema({
  title: String,
  type: { type: String, enum: ['call', 'email', 'meeting', 'reminder'] },
  relatedTo: { type: mongoose.Schema.Types.ObjectId, ref: 'Client' },
  agent: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  dueDate: Date,
  completed: { type: Boolean, default: false },
  createdAt: { type: Date, default: Date.now },
});
module.exports = mongoose.model('Task', taskSchema);
// Message.js
const messageSchema = new mongoose.Schema({
  sender: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  receiver: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  message: String,
  attachments: [String], // optional image/file links
  createdAt: { type: Date, default: Date.now },
});
module.exports = mongoose.model('Message', messageSchema);
// Campaign.js
const campaignSchema = new mongoose.Schema({
  client: { type: mongoose.Schema.Types.ObjectId, ref: 'Client' },
  schedule: [{
    label: String,
    date: Date,
    message: String
  }],
  createdAt: { type: Date, default: Date.now },
});
module.exports = mongoose.model('Campaign', campaignSchema);
```

TASK:02:

2. Google Maps / Geo Location Integration

Depends on: Google Maps API, OpenStreetMap, or similar
What it does:
Enables radius-based searches
Displays nearby schools, hospitals, commute times
Allows agents/clients to draw custom areas on maps
Endpoints (Later):
GET /geo/radius-search?lat=...&lng=...&radius=...
Google Places API for schools, shops, hospitals nearby
3. Calendar & Scheduling Integration
Depends on: Google Calendar API, Outlook API
What it does:
Syncs agent's available time
Clients can book appointments
Auto-adds events to agent's calendar
Endpoints (Later):
GET /calendar/availability/:agentId
POST /calendar/schedule
OAuth + event webhook setup
4. Automated Email & SMS (Reminders, Campaigns)
Depends on: Mailchimp, SendGrid, Twilio (SMS), Brevo (ex-Sendinblue), or Firebase Functions
What it does:
Sends auto-emails after form submissions, deal stage changes, campaigns
Sends SMS reminders for viewings, closing dates
Endpoints (Later):
POST /notify/email
POST /notify/sms
Campaign trigger service to scan scheduled messages
5. Document E-Signing Integration
Depends on: Dotloop, Docusign, Zipforms (for real estate)
What it does:
Allows uploading and signing of contracts or disclosures
Sends signing links to clients
Tracks who signed and when
Endpoints (Later):
POST /documents/upload
POST /documents/send-for-signature
GET /documents/status/:id
6. Analytics & Tracking
Depends on: Google Analytics, custom tracking tools
What it does:
Shows lead source conversion data
Monitors time spent on properties or pages
Funnel visualization (lead → conversion)
Endpoints (Later):
GET /analytics/leads
GET /analytics/property-views

TASK:03:

Final Remaining Backend Tasks (Minor or Optional Polishing)

Since everything core is done, here are the only remaining or optional backend tasks you could consider:

1. Admin Panel Backend

If not yet built:

View & manage all agents

Manually assign clients or leads

Set commission configs

API keys manager

Endpoints:

GET /admin/agents

PATCH /admin/assign-client/:clientId

POST /admin/commission-settings

2. Audit Logging System (Optional)

Track who changed what for compliance or internal debugging.

Schema:

js

```
{
  userId,
  action,
  targetModel,
  targetId,
  timestamp,
  description
}
```

Endpoint:

GET /audit-log

3. Webhook/Event Listener Service (Optional)

If using external tools (e.g., Mailchimp or Google Calendar), webhook support to sync back data.

Example:

/webhook/mailchimp/response

/webhook/calendar/update

4. Role Permissions Middleware (Polishing)

If you want fine-grained access control for:

Admins

Agents

Clients

Example:

js

```
authorizeRoles('admin', 'agent');
```

5.

Admin Report Export (Optional Enhancement)

Export leads, commissions, tasks to PDF/CSV

Generate monthly performance

`