Web Development Assessment Project: AI-Powered Mini CRM

Project Overview

Build a simplified CRM system with an integrated AI chatbot. This project will demonstrate your ability to architect solutions, implement complex features, and work with modern web technologies.

Timeline: 10 days
Tech Stack: MERN (MongoDB, Express.js, React, Node.js) + Firebase Auth + Socket.io (for chat only) + TanStack Query + Shadcn/ui + Tailwind CSS

UI Screens & Components

1. Authentication Screens

Login Page (`/login`)

- Logo/Brand header
- Email input field
- Password input field with show/hide toggle
- "Remember me" checkbox
- Login button
- Google OAuth button
- "Forgot password?" link
- "Don't have an account? Register" link
- Error/Success alerts

Register Page (`/register`)

- Logo/Brand header
- Display name input field
- Email input field
- Password input field with strength indicator
- Confirm password field
- Terms & conditions checkbox
- Register button
- Google OAuth button
- "Already have an account? Login" link
- Error/Success alerts

2. Main Layout (After Authentication)

Sidebar Navigation

- User profile section (avatar, name, email)
- Navigation items:
  - Dashboard (with icon)
  - Contacts (with icon)
  - Activities (with icon)
  - Tags (with icon)
  - AI Assistant (with icon)
- Logout button at bottom

Top Navigation Bar

- Search bar (global search)
- Notification icon (optional)
- User dropdown menu (profile, settings, logout)

3. Dashboard Screen (`/dashboard`)

Summary Cards Row

- Total Contacts card (number + icon)
- New Contacts This Week card (number + trend indicator)
- Total Activities card (number + icon)
- Active Tags card (number + icon)

Charts Section (Grid Layout)

- Contacts by Company chart (Bar chart)
- Activities Timeline chart (Line chart)
- Tag Distribution chart (Donut chart)

4. Contacts Management Screens

Contacts List Page (`/contacts`)

- Page header with "Contacts" title
- Action bar:
  - "Add Contact" button

        - "Import CSV" button
        - Search input with icon
        - Filter dropdown (by tags)
        - View toggle (grid/list)
- Bulk action bar (appears when items selected):
    - Selected count
    - "Delete Selected" button
    - "Tag Selected" button
- Data table with columns:
    - Checkbox for selection
    - Name (clickable)
    - Email
    - Company
    - Tags (colored badges)
    - Last Interaction
    - Actions (edit, delete icons)
- Pagination controls

Add/Edit Contact Modal

- Modal header with title
- Form fields:
    - Name input (required)
    - Email input (required)
    - Phone input
    - Company input
    - Tags selector (multi-select with colors)
    - Notes textarea
- Action buttons:
    - Cancel
    - Save/Update

Contact Detail Page (`/contacts/:id`)

- Back button
- Contact header:
    - Avatar/Initial
    - Name
    - Email
    - Edit button
    - Delete button
- Information cards:
    - Basic Info (phone, company)
    - Tags section
    - Notes section
    - Activity history related to this contact

CSV Import Modal

- Drag & drop zone or file picker
- Sample CSV format display
- Import progress bar
- Results summary:
    - Successful imports
    - Failed imports with reasons
    - Download error report link

5. Activities Screen (`/activities`)

Activities Timeline Page

- Page header with "Activity Timeline" title
- Filter bar:
    - Date range picker
    - Activity type filter dropdown
    - User filter (if multiple users)
- Timeline view:
    - Activity items with:
        - User avatar
        - Action description
        - Related entity link
        - Timestamp
        - Activity icon based on type
- Load more button (infinite scroll)

6. Tags Management Screen (`/tags`)

Tags List Page

- Page header with "Tags" title
- "Add Tag" button
- Tags grid/list:
    - Tag color indicator
    - Tag name
    - Usage count
    - Edit button (inline)

    - Delete button
- Color picker popup for editing

7. AI Assistant Screen (`/chat`)

Chat Interface

- Chat header:
    - "AI Assistant" title
    - Clear conversation button
- Messages area:
    - Message bubbles (user/AI differentiated)
    - Timestamps
    - Typing indicator
    - Auto-scroll to bottom
- Input area:
    - Message input field
    - Send button
    - Character count (optional)

8. User Profile Screen (`/profile`)

Profile Page

- Profile header:
    - Avatar upload zone
    - Current avatar display
- Form fields:
    - Display name input
    - Email (read-only)
    - Update password section (optional)
- Save changes button

Component Library (Using Shadcn/ui)

Shadcn Components to Use:

- `Button` - For all action buttons
- `Input` - For all text inputs
- `Label` - For form labels
- `Card` - For dashboard cards and content sections
- `Dialog` - For modals
- `Select` - For dropdowns
- `Badge` - For tags display
- `Table` - For contacts list
- `Tabs` - For tabbed content
- `Alert` - For success/error messages
- `Avatar` - For user avatars
- `Checkbox` - For bulk selection
- `Textarea` - For notes fields
- `Toast` - For notifications
- `Skeleton` - For loading states
- `Command` - For search/command palette
- `Popover` - For color picker
- `Calendar` - For date pickers
- `Chart` - For dashboard charts

Custom Components to Build:

- `ProtectedRoute` - Route wrapper for authentication
- `PageHeader` - Reusable page header
- `DataTable` - Enhanced table with sorting/filtering
- `TagSelector` - Multi-select tag component
- `ActivityItem` - Timeline item component
- `ChatMessage` - Chat message bubble
- `FileUpload` - Drag & drop file upload

Detailed Feature Requirements & Implementation Guide

1. Authentication System (Firebase)

What to Build:

- Login/Register pages with email/password
- Google OAuth integration
- Protected route wrapper component
- User profile page with ability to update display name and profile picture

- Logout functionality with proper cleanup

Technical Requirements:

- Store user profile data in MongoDB after Firebase authentication
- Sync Firebase UID with MongoDB user document
- Handle authentication state in React Context

Implementation Approach:

```
// Suggested Context structure
const AuthContext = {
  user: null, // Contains Firebase user + MongoDB profile
  loading: true,
  login: async (email, password) => {},
  register: async (email, password, displayName) => {},
  logout: async () => {},
  updateProfile: async (updates) => {}
}
```

Hints:

- Create a `ProtectedRoute` component that checks authentication state
- On successful Firebase auth, create/update user document in MongoDB
- Use Firebase `onAuthStateChanged` listener to persist auth state
- Store additional user metadata (like preferences) in MongoDB, not Firebase

Common Pitfalls to Avoid:

- Don't expose Firebase config keys in your repository
- Remember to handle loading states while checking authentication
- Ensure MongoDB user document is created even for Google OAuth users

---

2. Contact Management Module

What to Build: A full-featured contact management system with the following capabilities:

- Add/Edit/Delete contacts
- List view with pagination
- Detailed contact view
- Search and filter functionality
- Bulk operations (import/delete)
- Tag assignment

Database Schema Design:

```
// Suggested MongoDB schema
const contactSchema = {
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  phone: String,
  company: String,
  tags: [{ type: ObjectId, ref: 'Tag' }],
  notes: String,
  createdBy: { type: ObjectId, ref: 'User' },
  createdAt: Date,
  updatedAt: Date,
  lastInteraction: Date
}

// Create indexes for search
contactSchema.index({ name: 'text', email: 'text', company: 'text' });
```

API Endpoints to Create:

```
GET    /api/contacts?page=1&limit=10&search=john&tags=tag1,tag2
POST   /api/contacts
GET    /api/contacts/:id
PUT    /api/contacts/:id
DELETE /api/contacts/:id
POST   /api/contacts/bulk-import (CSV upload)
POST   /api/contacts/bulk-delete
```

Frontend Implementation with TanStack Query:

```
// Example query hook
const useContacts = (page, filters) => {
  return useQuery({
    queryKey: ['contacts', page, filters],
    queryFn: () => fetchContacts(page, filters),
    keepPreviousData: true // For smooth pagination
  });
};

// Example mutation hook
const useCreateContact = () => {
  const queryClient = useQueryClient();
  return useMutation({
    mutationFn: createContact,
    onSuccess: () => {
      queryClient.invalidateQueries(['contacts']);
    }
  });
};
```

CSV Import Feature:

- Use `multer` for file upload handling
- Use `csv-parser` or `papaparse` for parsing CSV
- Validate each row before importing
- Return import summary (success count, errors)

Implementation Hints:

- Implement debounced search to avoid too many API calls
- Use MongoDB aggregation for complex filtering
- For pagination, return total count along with paginated results
- Handle duplicate emails during bulk import gracefully

---

3. Activity Timeline

What to Build: An activity log that tracks all actions in the CRM:

- Log contact CRUD operations
- Track user logins
- Record bulk operations
- Display in a timeline format with infinite scroll

Database Schema:

```
const activitySchema = {
  user: { type: ObjectId, ref: 'User', required: true },
  action: {
    type: String,
    enum: ['contact_created', 'contact_updated', 'contact_deleted',
           'bulk_import', 'bulk_delete', 'user_login'],
    required: true
  },
  entityType: String, // 'contact', 'user', etc.
  entityId: ObjectId,
  entityName: String, // Store name for quick display
  metadata: Object, // Additional data like import count, changed fields
  timestamp: { type: Date, default: Date.now }
}
```

Implementation Approach:

- Use Mongoose middleware (pre/post hooks) to automatically log activities
- Implement infinite scroll using TanStack Query's `useInfiniteQuery`
- Create a reusable activity logging service

Backend Middleware Example:

```
// In your contact model
contactSchema.post('save', async function(doc) {
  await ActivityLog.create({
    user: doc.createdBy,
    action: doc.isNew ? 'contact_created' : 'contact_updated',
    entityType: 'contact',
    entityId: doc._id,
    entityName: doc.name
  });
});
```

Frontend Infinite Scroll:

```
const useActivities = () => {
  return useInfiniteQuery({
    queryKey: ['activities'],
    queryFn: ({ pageParam = 0 }) => fetchActivities(pageParam),
    getNextPageParam: (lastPage) => lastPage.nextCursor
  });
};
```

---

4. AI-Powered Chatbot (Socket.io + LLM)

What to Build: A real-time chat interface where users can:

- Ask general questions to an AI assistant
- Query CRM data through natural language
- View chat history
- See typing indicators during AI processing

Backend Architecture:

```
// Socket.io setup
io.on('connection', (socket) => {
  // Join user-specific room
  socket.on('join-chat', (userId) => {
    socket.join(`chat-${userId}`);
  });

  // Handle incoming messages
  socket.on('send-message', async (data) => {
    const { userId, message } = data;

    // Save user message to DB
    const userMessage = await saveMessage(userId, message, 'user');

    // Emit typing indicator
    socket.emit('ai-typing', true);

    // Process with AI
    const aiResponse = await processWithAI(message, userId);

    // Save AI response
    const aiMessage = await saveMessage(userId, aiResponse, 'ai');

    // Send response back
    socket.emit('ai-typing', false);
    socket.emit('new-message', aiMessage);
  });
});
```

Chat Message Schema:

```
const chatMessageSchema = {
  user: { type: ObjectId, ref: 'User' },
  message: String,
  sender: { type: String, enum: ['user', 'ai'] },
  timestamp: Date,
  conversationId: String // Group messages by conversation
}
```

AI Integration Approach:

- Create a service layer for AI interactions
- Implement context awareness by fetching relevant CRM data
- Add rate limiting to prevent API abuse
- Handle API errors gracefully

Frontend Chat Component Structure:

```
// Key components to build
<ChatContainer>
  <MessageList messages={messages} loading={aiTyping} />
  <TypingIndicator visible={aiTyping} />
  <MessageInput onSend={sendMessage} />
</ChatContainer>
```

Implementation Hints:

- Store OpenAI API key in backend environment variables only
- Implement auto-scroll to bottom for new messages
- Add message persistence so chat history survives page refresh
- Consider implementing a "thinking" animation while AI processes

---

5. Dashboard & Analytics

What to Build: A dashboard showing:

- Summary cards (total contacts, new this week, total activities)

- Charts for data visualization
- Quick stats and insights

Required Charts:

1. Contacts by Company (Horizontal bar chart - top 5)
2. Activities Timeline (Line chart - last 30 days)
3. Tag Distribution (Pie/Donut chart)

Backend Aggregation Queries:

```
// Example: Get contacts by company
const getContactsByCompany = async () => {
  return await Contact.aggregate([
    { $group: {
      _id: '$company',
      count: { $sum: 1 }
    }},
    { $sort: { count: -1 }},
    { $limit: 5 }
  ]);
};

// Example: Activities over time
const getActivityTimeline = async () => {
  const thirtyDaysAgo = new Date();
  thirtyDaysAgo.setDate(thirtyDaysAgo.getDate() - 30);

  return await Activity.aggregate([
    { $match: { timestamp: { $gte: thirtyDaysAgo }}},
    { $group: {
      _id: { $dateToString: { format: "%Y-%m-%d", date: "$timestamp" }},
      count: { $sum: 1 }
    }},
    { $sort: { _id: 1 }}
  ]);
};
```

Frontend Chart Implementation:

```
// Using Recharts example
import { BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip } from 'recharts';

const ContactsByCompanyChart = ({ data }) => (
  <BarChart width={400} height={300} data={data} layout="horizontal">
    <CartesianGrid strokeDasharray="3 3" />
    <XAxis type="number" />
    <YAxis dataKey="company" type="category" />
    <Tooltip />
    <Bar dataKey="count" fill="#8884d8" />
  </BarChart>
);
```

Implementation Hints:

- Cache dashboard queries as they can be expensive
- Use TanStack Query with a reasonable `staleTime` for dashboard data
- Make charts responsive using container queries or resize observers

---

6. Tags & Organization System

What to Build: A flexible tagging system for organizing contacts:

- CRUD operations for tags
- Color picker for tag colors
- Auto-complete when assigning tags
- Tag management page

Tag Schema:

```
const tagSchema = {
  name: { type: String, required: true, unique: true },
  color: { type: String, default: '#gray' },
  createdBy: { type: ObjectId, ref: 'User' },
  usageCount: { type: Number, default: 0 } // Track popularity
}
```

Implementation Features:

1. **Tag Manager Component:**
    - List all tags with color indicators
    - Inline editing for tag names and colors
    - Usage count display
    - Delete with confirmation (check if in use)
2. **Tag Selector Component:**
    - Multi-select dropdown
    - Type-ahead search
    - Create new tag inline
    - Show tag colors in dropdown

Reusable Tag Selector Example:

```
const TagSelector = ({ selectedTags, onChange }) => {
  const { data: allTags } = useTags();

  return (
    <Autocomplete
      multiple
      options={allTags}
      value={selectedTags}
      onChange={onChange}
      renderTags={(value, getTagProps) =>
        value.map((option, index) => (
          <Chip
            style={{ backgroundColor: option.color }}
            label={option.name}
            {...getTagProps({ index })}
          />
        ))
      }
    />
  );
};
```

10-Day Development Plan

Day 1: Project Setup & Authentication Backend

- Initialize project structure (separate client and server folders)
- Set up Express server with basic middleware (cors, body-parser, etc.)
- Configure MongoDB connection and create database
- Set up Firebase Admin SDK on backend
- Create User model with Firebase UID mapping
- Implement auth endpoints (register, login, profile)
- Set up API authentication
- Create auth middleware for protected routes
- Test all endpoints with Postman

Deliverables: Working backend with authentication, User model, protected API routes

---

Day 2: Frontend Setup & Authentication UI

- Set up React app with Next.js
- Install and configure Shadcn/ui with Tailwind CSS
- Set up React Router / Next.js routing with route structure
- Configure Firebase client SDK

- Create AuthContext with Firebase integration
- Implement ProtectedRoute component
- Build Login/Register pages using Shadcn components
- Add Google OAuth integration
- Create basic layout with sidebar navigation
- Build user profile page

Deliverables: Complete authentication flow, protected routes, user profile functionality

---

Day 3: Contact Management Backend

- Create Contact model with proper schema and indexes
- Implement CRUD endpoints for contacts
- Add pagination, search, and filtering logic
- Set up multer for file uploads
- Implement CSV import functionality with validation
- Add bulk delete endpoint
- Create activity logging hooks in Contact model
- Implement proper error handling and validation

Deliverables: Complete contact API with all CRUD operations, CSV import, search functionality

---

Day 4: Contact Management Frontend

- Set up TanStack Query with proper configuration
- Create contacts list page with Shadcn Table component
- Implement search with debouncing
- Build Add/Edit contact modal with form validation
- Create contact detail page
- Implement CSV upload with drag-and-drop
- Add bulk selection and delete functionality
- Create loading skeletons and error states

Deliverables: Full contact management UI with all operations working

---

Day 5: Tags System

- Create Tag model and API endpoints
- Implement tag CRUD operations
- Update Contact model to support tags
- Modify contact endpoints to populate tags
- Build tag management page with Shadcn components
- Create reusable TagSelector component
- Implement color picker for tags
- Add tag filtering to contacts list
- Update contact forms to include tag selection

Deliverables: Complete tagging system integrated with contacts

---

Day 6: Activity Timeline

- Create Activity model with proper schema
- Set up automatic activity logging using Mongoose hooks
- Implement activity feed endpoint with pagination
- Create different activity type handlers
- Build activity timeline page with infinite scroll
- Design activity item components with proper icons
- Implement date filtering for activities
- Add activity type filtering

Deliverables: Automatic activity logging, timeline UI with infinite scroll

---

Day 7: Dashboard & Analytics

- Create dashboard API endpoints with aggregation queries
- Implement caching for expensive operations
- Set up data transformation for charts
- Build dashboard layout with responsive grid
- Implement summary cards with Shadcn Card component
- Integrate Recharts for data visualization
- Create responsive chart components
- Add loading states for dashboard data

Deliverables: Working dashboard with all charts and metrics

Day 8: AI Chatbot Backend

- Set up Socket.io server with proper configuration
- Create ChatMessage model for persistence
- Implement Socket.io event handlers
- Set up OpenAI API integration
- Build AI service layer with context awareness
- Implement rate limiting for AI requests
- Add error handling for API failures
- Create chat history endpoints

Deliverables: Working Socket.io chat server with AI integration

---

Day 9: AI Chatbot Frontend & Polish

- Build chat UI using Shadcn components
- Implement Socket.io client connection
- Create message components with proper styling
- Add typing indicators and loading states
- Implement auto-scroll and message persistence
- Polish UI across all pages
- Add toast notifications for user feedback
- Fix responsive design issues
- Implement error boundaries

Deliverables: Complete chat interface, polished UI throughout application

---

Day 10: Testing, Documentation & Deployment Prep

- Thorough testing of all features
- Fix any remaining bugs
- Create database seed script with sample data
- Optimize API queries and frontend performance
- Write comprehensive README with setup instructions
- Document all API endpoints
- Create demo video showcasing all features
- Final code cleanup and organization
- Prepare environment variables template

Deliverables: Bug-free application, complete documentation, demo video

---

Key Implementation Tips

1. **Start Simple:** Get basic functionality working before adding complexity
2. **Use Shadcn Examples:** Reference Shadcn documentation for component usage
3. **Consistent Error Handling:** Implement a global error handler early
4. **Git Commits:** Commit frequently with meaningful messages
5. **API Testing:** Use Postman to test APIs before building UI
6. **Component Reusability:** Build reusable components from the start
7. **State Management:** Use TanStack Query for server state, Context for auth
8. **Performance:** Implement pagination and lazy loading early

Remember: Focus on building a working application that demonstrates your problem-solving abilities. Clean code and proper architecture are more important than perfect styling.