As an expert AI, create a modern, scalable web application for businesses to manage client profiles and communications, with the following specifications:

\*\*Framework and Technologies\*\*:

- Use Next.js (latest stable version) with TypeScript for frontend and backend.

- Use Node.js for the server runtime.

- Style the application using Tailwind CSS for a responsive, modern UI.

- Integrate Oracle 19c database for backend data persistence.

- Integrate Google Gmail API for email notifications and fetching email data.

- Integrate VoIP.ms API for sending SMS notifications.

\*\*Application Requirements\*\*:

- Build a SaaS application for businesses to manage client profiles and communications, with the following features:

- \*\*Client Management\*\*:

- CRUD operations for client profiles (create, read, update, delete).

- Client fields: id, name, email, phone, business type (self-employed or corporation), address, created\_at, updated\_at.

- Search clients by name, email, or phone with pagination and sorting.

- \*\*Alert System\*\*:

- Create and manage alerts for tax obligations and custom reminders (e.g., GST/QST declaration due dates, source deduction deadlines, annual filings).

- Alert fields: id, client\_id, title, description, due\_date, recurrence (one-time, monthly, quarterly, yearly), notification\_method (email, SMS, both), status (pending, sent, dismissed).

- Predefined templates for common obligations based on business type:

- Self-employed: GST/QST filing (quarterly/monthly), source deductions, income tax filing.

- Corporation: GST/QST filing, source deductions, T2 filing, annual returns.

- Allow custom reminders with user-defined dates and recurrence.

- \*\*Notifications\*\*:

- Send email reminders via Google Gmail API to clients for upcoming obligations.

- Send SMS reminders via VoIP.ms API to clients or business owners.

- Log all notifications (email/SMS) in the database with status (sent, failed, pending).

- \*\*Dashboard\*\*:

- Tabbed interface for:

- Client List: Table view with search, pagination, and sorting.

- Client Profile: View/edit client details and associated alerts.

- Alerts: View upcoming/overdue alerts, create/edit alerts.

- Notifications: View notification logs (email/SMS).

- Responsive design for mobile, tablet, and desktop.

- Minimize widget repetition by reusing components (e.g., buttons, forms, modals).

- Ensure accessibility (e.g., ARIA labels, keyboard navigation).

\*\*Integrations\*\*:

1. \*\*Google Gmail API\*\*:

- Authenticate users via OAuth 2.0 to access Gmail.

- Send email notifications for alerts (e.g., “Your GST/QST filing is due on [date]”).

- Fetch recent emails related to clients (e.g., filter by client email address) for display in the client profile tab.

- Store OAuth tokens securely in the database with encryption.

2. \*\*VoIP.ms API\*\*:

- Send SMS notifications for alerts (e.g., “Reminder: Source deduction due on [date]”).

- Log SMS delivery status in the database.

- Handle VoIP.ms API errors (e.g., invalid DID, IP not whitelisted) and retry logic.

3. \*\*Oracle 19c\*\*:

- Store client profiles, alerts, notification logs, and OAuth tokens.

- Schema:

- `clients`: id, name, email, phone, business\_type, address, created\_at, updated\_at.

- `alerts`: id, client\_id, title, description, due\_date, recurrence, notification\_method, status, created\_at, updated\_at.

- `notifications`: id, alert\_id, client\_id, type (email/SMS), content, status, sent\_at, error\_message.

- `oauth\_tokens`: id, user\_id, provider (gmail), access\_token, refresh\_token, expires\_at.

\*\*Best Practices\*\*:

1. \*\*Project Structure\*\*:

- Organize code into modules/classes:

- `/components`: Reusable UI components (e.g., ClientCard, AlertForm, NotificationLog).

- `/pages`: Next.js pages (e.g., `/clients`, `/alerts`, `/notifications`).

- `/api`: API routes (e.g., `/api/clients`, `/api/alerts`, `/api/notifications`).

- `/lib`: Utilities (e.g., Gmail API client, VoIP.ms API client, database connectors).

- `/services`: Business logic (e.g., ClientService, AlertService, NotificationService).

- `/types`: TypeScript types (e.g., Client, Alert, Notification).

- `/constants`: Centralized constants (e.g., API endpoints, business types, recurrence options).

- Use barrel files for simplified imports.

2. \*\*Environment Variables\*\*:

- Create a `.env` file for:

- Oracle 19c connection details (host, port, SID, username, password).

- Google Gmail API credentials (client ID, client secret, redirect URI).

- VoIP.ms API credentials (username, password, DID).

- JWT secret for authentication.

- Include a `.env.example` file with placeholders.

- Use `dotenv` for secure loading.

3. \*\*Database Integration\*\*:

- Use `oracledb` Node.js driver with connection pooling.

- Implement a `DatabaseService` class for queries and transactions.

- Use parameterized queries to prevent SQL injection.

- Create indexes on frequently queried fields (e.g., `clients.email`, `alerts.due\_date`).

4. \*\*Error Handling\*\*:

- Implement global error handling middleware for API routes.

- Create an `AppError` class for standardized errors (e.g., `{ statusCode, message, details }`).

- Handle API-specific errors (e.g., Gmail API rate limits, VoIP.ms invalid DID).

- Display user-friendly errors in the UI (e.g., toast notifications).

- Log errors to a file in production.

5. \*\*Security\*\*:

- Hash passwords with `bcrypt`.

- Use JWT for authentication and protect API routes.

- Encrypt OAuth tokens in the database.

- Sanitize inputs to prevent XSS.

- Implement rate limiting on API endpoints.

- Use HTTPS in production.

6. \*\*State Management\*\*:

- Use Zustand for lightweight state management (e.g., user session, alert filters).

- Use SWR or React Query for data fetching and caching.

7. \*\*UI/UX\*\*:

- Use Tailwind CSS for a clean, responsive design.

- Implement tabs for navigation (Clients, Alerts, Notifications).

- Reuse components (e.g., `<SearchInput>`, `<FormSelect>`).

- Add loading states and skeleton screens.

- Use a consistent color scheme via `tailwind.config.js`.

8. \*\*Performance\*\*:

- Use Next.js SSG/SSR where appropriate.

- Optimize images with Next.js Image.

- Implement code splitting and lazy loading.

9. \*\*Alert Scheduling\*\*:

- Use a background job (e.g., `node-cron`) to check for upcoming alerts daily.

- Trigger email/SMS notifications for alerts due within 1-7 days (configurable).

- Update alert status after notification is sent.

10. \*\*Testing\*\*:

- Write unit tests for services using Jest.

- Write integration tests for API routes using Supertest.

- Mock Gmail and VoIP.ms APIs during testing.

- Use React Testing Library for components.

11. \*\*Documentation\*\*:

- Include a `README.md` with setup, environment variables, database schema, and testing instructions.

- Add inline code comments for complex logic.

- Generate API docs with Swagger.

12. \*\*Centralized Constants\*\*:

- Create `constants.ts` for:

- Business types (self-employed, corporation).

- Recurrence options (one-time, monthly, quarterly, yearly).

- Notification methods (email, SMS, both).

- Tax obligation templates (e.g., GST/QST due dates, source deductions).

\*\*Deliverables\*\*:

- Provide complete source code, including:

- `.env.example` with placeholders.

- `README.md` with setup instructions.

- SQL scripts for Oracle 19c schema.

- TypeScript code for frontend and backend.

- Tailwind CSS configuration.

- ESLint/Prettier configs.

- Include sample API responses and error formats.

- Explain folder structure and key design decisions.

- Wrap code files in `<xaiArtifact>` tags with unique `artifact\_id` values (e.g., `index.tsx`, `NotificationService.ts`, `schema.sql`).

\*\*Constraints\*\*:

- Ensure scalability and maintainability.

- Use stable libraries and avoid deprecated features.

- Target modern browsers (Chrome, Firefox, Safari, Edge).

\*\*Output Format\*\*:

- Provide source code as a ZIP archive or GitHub repository link.

- Include instructions for running locally.

- Use `contentType` values like `text/typescript`, `text/sql`, or `text/plain` for artifacts.