PRODUCT REQUIREMENTS DOCUMENT

Full Stack Developer Assessment — Task No. 002

Company	Upskill Tech Solution Pvt Ltd
Document Type	Full Stack Assessment PRD
Tech Stack	React/Next.js + Node.js/Express + MySQL
Version	1.0.0 — Final Submission Build

This document is a complete Product Requirements Document (PRD) specifying every feature, constraint, API endpoint, database schema, UI screen, and evaluation criterion required to build and pass the Full Stack Developer Assessment.

1. Project Overview

This assessment requires building a full-stack web application from scratch within a 3-day deadline. The application is a Contact & Task Management System that demonstrates proficiency in backend API design, relational database architecture, authentication, and frontend development.

1.1 Objective

Build a working web application with the following core capabilities:

- User registration and authentication with secure token-based sessions
- Management of contacts (linked to a user account) with addresses
- Task management linked to both users and contacts
- Email simulation with persistent logging to the database
- Clean, functional UI built without any third-party UI libraries

1.2 What the Evaluator Is Looking For

The company is evaluating you on the following key competencies:

Competency Area	What They Want to See
Database Design	Proper schema with all constraints (PK, FK, UNIQUE, NOT NULL), DB triggers for computed columns, and normalized structure.
Backend API	RESTful endpoints, bcrypt password hashing, JWT/token auth with 15-minute expiry, logging middleware, email log insertion.
Frontend	React or Next.js, NO UI libraries (no Bootstrap, MUI, Tailwind components), token in localStorage, auto-logout at 15 min.
Data Integrity	Foreign keys enforced, unique constraints working, proper validations on all inputs, audit fields (created_by, updated_by).
Code Quality	Clean, readable code, no dead code, proper error handling, structured project layout.
Delivery	Complete Git repository with README and a demo video walkthrough of all features.

2. Database Schema

The application uses MySQL. All tables must include proper PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, and DEFAULT constraints. Audit fields (created_by, updated_by, created_at, updated_at) are required on every table.

2.1 Table: users

Stores all registered users of the system. This is the root table of the entire application.

Column	Type	Constraints	Notes
id	INT / UUID	PK, AUTO_INCREMENT	Primary identifier
first_name	VARCHAR(100)	NOT NULL	User's first name
last_name	VARCHAR(100)	NOT NULL	User's last name
full_name	VARCHAR(200)	NULL (auto-filled by DB trigger)	CRITICAL: Must be maintained by a DB trigger, not application code. Concatenation of first_name + ' ' + last_name.
email	VARCHAR(255)	NOT NULL, UNIQUE	Must be a valid email format. Uniqueness enforced at DB level.
phone	CHAR(10)	NOT NULL, UNIQUE	Exactly 10 digits. Uniqueness enforced at DB level.
password	VARCHAR(255)	NOT NULL	Must be stored as a bcrypt hash. NEVER store plaintext.
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Auto-set on insert
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE	Auto-updated on row change
created_by	INT / VARCHAR	NOT NULL	ID or reference of the user who created the record. For self-registration, use the new user's own ID.
updated_by	INT / VARCHAR	NOT NULL	ID or reference of the user who last updated the record.

2.1.1 DB Trigger — full_name

This is an EVALUATION CHECKLIST item. The full_name column must be maintained by a MySQL TRIGGER, not in application code. Create two triggers:

- BEFORE INSERT trigger: Sets full_name = CONCAT(NEW.first_name, ' ', NEW.last_name)
- BEFORE UPDATE trigger: Re-sets full_name if first_name or last_name changes

```
SQL Example: CREATE TRIGGER trg_users_before_insert BEFORE INSERT ON users FOR EACH ROW SET NEW.full_name = CONCAT(NEW.first_name, ' ', NEW.last_name); CREATE TRIGGER trg_users_before_update BEFORE UPDATE ON users FOR EACH ROW SET NEW.full_name = CONCAT(NEW.first_name, ' ', NEW.last_name);
```

2.2 Table: users_contact

Stores multiple contact entries (phone numbers + emails) for a user. Each contact belongs to one user. The combination of user_id + contact_number must be unique (a user cannot have the same contact number twice).

Column	Type	Constraints	Notes
id	INT	PK, AUTO_INCREMENT	Primary identifier
user_id	INT	NOT NULL, FK → users.id ON DELETE CASCADE	Owning user. Must reference a valid user.
contact_number	VARCHAR(20)	NOT NULL, UNIQUE per user_id	Contact phone number. UNIQUE constraint: UNIQUE(user_id, contact_number)
contact_email	VARCHAR(255)	NULL	Optional email for this contact
note	TEXT	NULL	Free-text note about this contact
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	
updated_at	DATETIME	ON UPDATE CURRENT_TIMESTAMP	
created_by	INT	NOT NULL	Audit field — who created this record
updated_by	INT	NOT NULL	Audit field — who last updated this record

2.3 Table: contact_address

Stores one or more addresses for a contact entry. Each address belongs to one contact in users_contact.

Column	Type	Constraints	Notes
id	INT	PK, AUTO_INCREMENT	Primary identifier
contact_id	INT	NOT NULL, FK → users_contact.id ON DELETE CASCADE	Must reference a valid contact
address_line1	VARCHAR(255)	NOT NULL	Primary address line
address_line2	VARCHAR(255)	NULL	Apartment, suite, floor, etc.
city	VARCHAR(100)	NOT NULL	City name
state	VARCHAR(100)	NOT NULL	State or province

Column	Type	Constraints	Notes
pincode	VARCHAR(20)	NOT NULL	Postal / PIN code
country	VARCHAR(100)	NOT NULL, DEFAULT 'India'	Country name
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	
updated_at	DATETIME	ON UPDATE CURRENT_TIMESTAMP	
created_by	INT	NOT NULL	Audit field
updated_by	INT	NOT NULL	Audit field

2.4 Table: users_task

Stores tasks created by a user. CRITICAL constraint: the contact_id in a task MUST belong to the same user as user_id. This must be enforced via application logic or a CHECK constraint/trigger.

Column	Type	Constraints	Notes
id	INT	PK, AUTO_INCREMENT	
user_id	INT	NOT NULL, FK → users.id ON DELETE CASCADE	The user who owns this task
contact_id	INT	NOT NULL, FK → users_contact.id	MUST belong to the same user as user_id. Validate in API.
title	VARCHAR(255)	NOT NULL	Short title of the task
description	TEXT	NULL	Detailed description
status	ENUM	NOT NULL, DEFAULT 'pending'	Values: 'pending', 'in_progress', 'completed', 'cancelled'
due_date	DATE or DATETIME	NULL	Optional deadline for the task
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	
updated_at	DATETIME	ON UPDATE CURRENT_TIMESTAMP	
created_by	INT	NOT NULL	Audit field
updated_by	INT	NOT NULL	Audit field

2.5 Table: email_logs

Stores every simulated email sent by the system. No actual email service is required — all emails are written to this table instead.

Column	Type	Constraints	Notes
id	INT	PK, AUTO_INCREMENT	
to_email	VARCHAR(255)	NOT NULL	Recipient email address
subject	VARCHAR(255)	NOT NULL	Email subject line
body	TEXT	NOT NULL	Full email body content
sent_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Timestamp when the log was created

Email logs must be inserted when: (1) a user registers, (2) a task is created, (3) any other relevant system event. The exact triggers should be implemented consistently.

3. Backend Requirements (Node.js + Express + MySQL)

3.1 Project Structure

Organize the backend with a clean, professional folder structure:

Folder / File	Purpose
src/config/db.js	MySQL connection pool using mysql2
src/middleware/auth.js	JWT verification middleware — protects routes
src/middleware/logger.js	Request logging middleware (method, URL, status, timestamp)
src/middleware/validate.js	Input validation helpers (express-validator or custom)
src/routes/auth.routes.js	POST /register, POST /login
src/routes/user.routes.js	GET/PUT /users/me — profile management
src/routes/contact.routes.js	CRUD for users_contact
src/routes/address.routes.js	CRUD for contact_address
src/routes/task.routes.js	CRUD for users_task
src/controllers/	One controller per route file
src/models/	DB query functions (optional but clean)
.env	DB credentials, JWT secret, port
server.js / app.js	Express app entry point
database/schema.sql	Full DDL SQL with all tables, triggers, constraints

3.2 Authentication

3.2.1 POST /api/auth/register

Registers a new user.

Item	Detail
Request Body	{ first_name, last_name, email, phone, password }
Validation	first_name & last_name required. Email must be valid format. Phone must be exactly 10 digits. Password minimum 8 characters.
Password	Hash with bcrypt (salt rounds: 10 minimum) BEFORE storing in DB. Never store plaintext.
DB Action	INSERT into users table. DB trigger auto-fills full_name. Set created_by = new user's own ID.
Email Log	INSERT into email_logs: to_email = registered email, subject = 'Welcome to the platform', body = welcome message.
Response 201	{ message: 'User registered successfully', userId }

Item	Detail
Response 400	{ error: 'Validation error message' }
Response 409	{ error: 'Email already exists' } or { error: 'Phone already exists' }

3.2.2 POST /api/auth/login

Item	Detail
Request Body	{ email, password }
Validation	Email and password required. Email format check.
Auth Check	Fetch user by email. Use bcrypt.compare() to verify password. Return 401 if not matching — use a generic message, never say which field is wrong.
Token	Generate JWT with payload: { userId, email }. Set expiry to 15 minutes (expiresIn: '15m'). Sign with a strong secret from .env.
Response 200	{ token, user: { id, full_name, email } }
Response 401	{ error: 'Invalid credentials' }

3.3 Auth Middleware

All routes except /register and /login must be protected by this middleware. It should:

- Read the Authorization header: Bearer <token>
- Verify JWT with jwt.verify(). Return 401 if token is missing, malformed, or expired.
- Attach the decoded payload to req.user (e.g., req.user = { userId, email })
- Allow the request to proceed with next() only if token is valid
- Return 401 with { error: 'Token expired' } or { error: 'Unauthorized' } for failures

3.4 Logging Middleware

A logging middleware must be applied globally to all requests. This is an EVALUATION item. Minimum log format:

```
[2025-01-01 12:00:00] POST /api/auth/login — 200 — 45ms
```

Fields to log: Timestamp, HTTP Method, Endpoint URL, Response Status Code, Response Time. Use morgan or write a custom middleware.

3.5 Contact Endpoints

All endpoints below require a valid JWT. The contact must belong to the authenticated user (req.user.userId).

Endpoint	Method	Description & Validation
/api/contacts	GET	List all contacts belonging to the logged-in user. Filter by req.user.userId.
/api/contacts	POST	Create a new contact. Body: { contact_number, contact_email, note }. contact_number required, UNIQUE per user. Set user_id = req.user.userId.
/api/contacts/:id	GET	Get a specific contact. Verify it belongs to req.user.userId or return 403.
/api/contacts/:id	PUT	Update a contact. Verify ownership first. Update updated_by = req.user.userId.
/api/contacts/:id	DELETE	Delete a contact. Verify ownership. Cascade deletes addresses due to FK.

3.6 Address Endpoints

Endpoint	Method	Description & Validation
/api/contacts/:contactId/addresses	GET	List all addresses for a contact. Verify the contact belongs to the user.
/api/contacts/:contactId/addresses	POST	Create address. Body: { address_line1, city, state, pincode, country, address_line2 }. address_line1, city, state, pincode required.
/api/contacts/:contactId/addresses/:id	PUT	Update address. Verify chain: address → contact → user ownership.
/api/contacts/:contactId/addresses/:id	DELETE	Delete address. Verify ownership chain.

3.7 Task Endpoints

CRITICAL: When creating or updating a task, validate that the contact_id belongs to the authenticated user. Never allow a user to attach a task to someone else's contact.

Endpoint	Method	Description & Validation
/api/tasks	GET	List all tasks for the authenticated user. Support optional filter by status.
/api/tasks	POST	Create task. Body: { contact_id, title, description, status, due_date }. Validate contact_id belongs to user. Status defaults to 'pending'.
/api/tasks/:id	GET	Get specific task. Verify ownership.
/api/tasks/:id	PUT	Update task. If updating contact_id, re-validate ownership. Update updated_by.
/api/tasks/:id	DELETE	Delete task. Verify ownership.

3.8 Error Handling & Response Standards

Use consistent JSON error responses across all endpoints:

- 200 OK — Successful GET / PUT
- 201 Created — Successful POST
- 400 Bad Request — Validation error (missing fields, wrong format)
- 401 Unauthorized — Missing or invalid/expired token
- 403 Forbidden — Authenticated but not authorized (e.g., accessing another user's data)
- 404 Not Found — Resource does not exist
- 409 Conflict — Duplicate email, phone, or unique field
- 500 Internal Server Error — Catch-all with a generic message (never expose stack traces)

Add a global error handling middleware as the last middleware in Express to catch unhandled errors.

4. Frontend Requirements (React / Next.js)

Requirement	Specification
Framework	React (CRA or Vite) or Next.js (Pages Router preferred)
UI Libraries	STRICTLY NONE. No Bootstrap, Material UI, Tailwind UI components, Chakra, Ant Design, etc. Use plain CSS (CSS modules, styled-components, or inline styles).
Token Storage	Store JWT in localStorage. Key: 'token' or 'auth_token'.
Auto-Logout	After exactly 15 minutes of token issuance, clear localStorage and redirect to Login. Implement with setTimeout or by checking token expiry on each API call.
API Communication	Use fetch() or axios. Always attach Authorization: Bearer <token> header to all protected API calls.
Protected Routes	Redirect unauthenticated users to /login for all pages except /login and /register.
State Management	React Context, Redux, or Zustand. No requirement stated, but must handle auth state globally.

4.1 Pages & Components

4.1.1 Login Page — /login

Element	Specification
Form Fields	Email (type='email'), Password (type='password')
Validation	Both fields required. Email format validation. Show inline error messages.
Submit	POST /api/auth/login. On success: store token in localStorage, redirect to /dashboard.
Error State	Show API error message (e.g., 'Invalid credentials') below form.
Navigation	Link to Register page
Auto-Logout Setup	After storing token, start a 15-minute timer (setTimeout). On expiry: localStorage.clear(), redirect to /login, show 'Session expired' message.

4.1.2 Register Page — /register

Element	Specification
Form Fields	First Name, Last Name, Email (type='email'), Phone (10 digits, type='tel'), Password, Confirm Password
Validation	All fields required. Email format. Phone exactly 10 digits. Password min 8 chars. Confirm password match. Show all errors inline.

Element	Specification
Submit	POST /api/auth/register. On success: show success message, redirect to /login.
Error State	Show API errors (duplicate email/phone) clearly.
Note	full_name is NOT an input field — it is auto-generated by the DB trigger.

4.1.3 Dashboard Page — /dashboard

Element	Specification
Protected	Redirect to /login if no token in localStorage
Content	Welcome message with user's full_name. Summary cards: Total Contacts, Total Tasks, Tasks by Status (pending/in_progress/completed).
Navigation	Links/Nav to: Contacts, Tasks, Logout
Logout	Clear localStorage, redirect to /login
Data	Fetch from /api/contacts and /api/tasks to populate stats

4.1.4 Contacts Page — /contacts

Element	Specification
List View	Table or card list of all user's contacts. Show: contact_number, contact_email, note.
Add Contact	Inline form or modal: contact_number (required), contact_email (optional), note (optional). Validates uniqueness on submit.
Edit Contact	Edit button per row opens prefilled form. PUT to /api/contacts/:id on submit.
Delete Contact	Delete button with confirmation dialog. DELETE /api/contacts/:id.
View Address	Each contact row has a 'View Addresses' button/link that goes to the Address page for that contact.

4.1.5 Address Page — /contacts/:contactId/addresses

Element	Specification
List View	List of all addresses for the selected contact
Add Address	Form: address_line1 (required), address_line2, city (required), state (required), pincode (required), country (default: 'India')
Edit Address	Edit button, prefilled form, PUT to API
Delete Address	Delete with confirmation
Breadcrumb	Show: Dashboard > Contacts > Addresses for [contact_number]

4.1.6 Tasks Page — /tasks

Element	Specification
List View	Table of all tasks. Columns: title, contact (display contact_number), status, due_date, actions.
Filter	Optional: filter tasks by status using a dropdown
Add Task	Form: contact_id (dropdown of user's contacts), title, description, status, due_date (date picker or text input)
Edit Task	Edit button, prefilled form. Validate contact_id still belongs to user.
Delete Task	Delete with confirmation
Status Update	Allow quick status change inline (pending → in_progress → completed)

5. Security & Validation Requirements

5.1 Password Security

- Use bcrypt with a minimum salt round of 10
- Never log, return, or expose the hashed password in any API response
- On login, use bcrypt.compare() — never direct string comparison
- On profile update, if password is being changed, hash the new one too

5.2 JWT Token Security

- Token expiry MUST be exactly 15 minutes ('expiresIn: 15 * 60' seconds or '15m' string)
- Store JWT_SECRET in .env, minimum 32 characters long
- Return 401 with a clear error message when token is expired
- Frontend must handle 401 responses by redirecting to /login automatically

5.3 Input Validation Rules

Field	Validation Rules
email	Valid email format (RFC 5321). Must be lowercase (normalize). Max 255 chars.
phone	Exactly 10 numeric digits. No spaces, dashes, or country codes. Use regex: /^[0-9]{10}$/
password	Minimum 8 characters. Recommend requiring at least 1 number and 1 letter.
pincode	6 digits for Indian PIN codes or configurable. No letters.
contact_number	Non-empty string, reasonable max length (20 chars). Unique per user.
due_date	Valid date format (YYYY-MM-DD). Must be a present or future date on creation.
status	Enum: 'pending', 'in_progress', 'completed', 'cancelled' only

5.4 Authorization Checks

- EVERY data endpoint must verify the resource belongs to req.user.userId
- On GET /api/contacts — WHERE user_id = req.user.userId
- On GET /api/contacts/:id — Verify row's user_id = req.user.userId
- On task create — SELECT contact WHERE id = contact_id AND user_id = req.user.userId
- Return 403 Forbidden (not 404) when a resource exists but doesn't belong to the user

6. Email Simulation & Logging

There is NO requirement for a real email service (no SMTP, no SendGrid, etc.). All emails are simulated by inserting records into the email_logs table. This is an EVALUATION CHECKLIST item.

6.1 When to Insert Email Logs

Trigger Event	Email Log Content
User Registration	to: user.email | subject: 'Welcome to [App Name]!' | body: 'Hello [full_name], your account has been created successfully.'
Task Created	to: user.email | subject: 'New Task Created: [title]' | body: 'A new task titled [title] has been assigned to you with status: pending.'
Task Status Update (optional)	to: user.email | subject: 'Task Updated: [title]' | body: 'Task [title] status changed to [status].'

6.2 Implementation Pattern

Create a reusable emailService function:

```
async function sendEmail(toEmail, subject, body) {   await db.query('INSERT INTO email_logs (to_email, subject, body) VALUES (?, ?, ?)', [toEmail, subject, body]); }
```

Call this function from your controllers after successful actions. Do not block the main response — you can fire and forget or await it.

7. Evaluation Checklist — Must Pass All Items

The evaluator will test the following items specifically. Make sure each one is implemented and demonstrable in the demo video.

#	Checklist Item	How to Prove It
1	full_name maintained by DB trigger (NOT in application code)	Show the SQL trigger code. Insert a user via Postman and show full_name is auto-populated. Try updating first_name and show full_name updates too.
2	Proper validations on all inputs	Try submitting empty forms, invalid email formats, 9-digit phone numbers — all should return 400 with clear messages.
3	Unique constraints enforced at DB level	Try registering with the same email or phone twice. Try adding the same contact number twice for the same user. Show the DB constraints in the schema file.
4	Token expiry handled (15 minutes)	Login, wait for the timer (or demonstrate by reducing to 1 min for testing), and show the app automatically logs out and redirects to /login.
5	Email logs inserted into email_logs table	After registration and task creation, open the DB and show rows in the email_logs table.
6	Clean, functional UI with no UI libraries	Show package.json — no Bootstrap, MUI, Tailwind, etc. Open the app and demonstrate all CRUD operations work.
7	Complete Git repo with demo video	A GitHub/GitLab repo with all code, a database/schema.sql file, a README with setup instructions, and a Loom/YouTube video showing the full app flow.

8. Demo Video Script

The demo video must cover the following flow in order. Record a screen capture (Loom recommended). Duration: 5–10 minutes maximum.

1. Open the app and show the Register page. Fill in all fields including a valid email and 10-digit phone. Submit.
2. Open MySQL workbench or CLI — show the new row in users table. HIGHLIGHT that full_name is populated automatically. Show the email_logs row for welcome email.
3. Go to Login page. Login with the registered credentials. Show the JWT token stored in localStorage (browser DevTools > Application > Local Storage).
4. Navigate to Dashboard. Show the welcome message with full_name.
5. Go to Contacts. Add a new contact with a phone number. Show it appears in the list.
6. Click 'View Addresses' for the contact. Add a new address. Show it saved correctly.
7. Go to Tasks. Create a new task linked to the contact. Show the task appears. Show the email_logs row for the task email.
8. Edit the task status to 'in_progress'. Delete the task. Delete the contact (show address is also deleted via cascade).
9. Open the browser console/DevTools and show the logging middleware output in the terminal (or show logged requests).
10. Demonstrate auto-logout: either wait 15 minutes or temporarily change the token expiry to 10 seconds. Show that the app redirects to /login automatically.
11. Show the GitHub repository — highlight: schema.sql, .env.example, README, clean folder structure.

9. Repository & Setup Requirements

9.1 Repository Structure

- Top-level folders: /backend and /frontend (or /client and /server)
- /backend/database/schema.sql — Complete DDL with all tables, constraints, and triggers
- /backend/.env.example — Template with all required environment variables (DB_HOST, DB_USER, DB_PASS, DB_NAME, JWT_SECRET, PORT)
- README.md at root with setup instructions (see below)

9.2 README Must Include

12. Project overview and feature list
13. Prerequisites: Node.js version, MySQL version
14. Backend setup: npm install, database creation, running schema.sql, configuring .env, npm start
15. Frontend setup: npm install, configuring API base URL, npm start or npm run dev
16. Link to demo video

9.3 Environment Variables (.env)

Variable	Description
DB_HOST	MySQL host (e.g., localhost)
DB_USER	MySQL username
DB_PASS	MySQL password
DB_NAME	Database name (e.g., assessment_db)
JWT_SECRET	Min 32-character random string for token signing
PORT	Express server port (default: 5000)

10. Technology Stack Summary

Layer	Technology	Notes
Backend Runtime	Node.js (v18+)	LTS version recommended
Backend Framework	Express.js	v4.x
Database	MySQL 8.x	Required. PostgreSQL is NOT acceptable per the spec.
DB Driver	mysql2	Use with promises/async-await
Password Hashing	bcrypt or bcryptjs	Salt rounds >= 10
Auth	jsonwebtoken	JWT, 15-minute expiry, HS256 algorithm
Logging	morgan (optional) or custom middleware	Logs every HTTP request
Validation	express-validator or custom	Server-side validation required
Frontend	React (CRA/Vite) or Next.js	No UI libraries allowed
HTTP Client	fetch() or axios	For API calls from frontend
CSS	Plain CSS, CSS Modules, or styled-components	NO component libraries
Dev Tools	Postman (API testing), MySQL Workbench	For development and demo

11. Data Relationships & Business Rules

11.1 Entity Relationships

The data hierarchy is strictly: User → Contact → Address, and User + Contact → Task.

Relationship	Rule
User → Contact	One user can have MANY contacts. A contact belongs to exactly ONE user.
Contact → Address	One contact can have MANY addresses. An address belongs to exactly ONE contact.
User + Contact → Task	A task belongs to ONE user AND is linked to ONE contact. The contact MUST belong to the same user.
User DELETE	Cascades to: contacts, addresses (via contact), tasks
Contact DELETE	Cascades to: addresses, tasks referencing this contact
Address DELETE	Does not cascade (leaf node)
Task DELETE	Does not cascade (leaf node)

11.2 Critical Business Rule — Task Ownership

This is specifically called out in the assessment document. When a task is created:

17. The API must receive: user_id (from req.user.userId via JWT) and contact_id (from the request body)
18. Before inserting, run: SELECT id FROM users_contact WHERE id = contact_id AND user_id = req.user.userId
19. If no row is returned, return 403 Forbidden: 'Contact does not belong to this user'
20. Only if validation passes, insert the task with both user_id and contact_id

12. Final Submission Checklist

Before submitting, verify every item on this list is complete:

12.1 Database

- All 5 tables created with correct columns, types, and constraints
- UNIQUE constraint on users.email and users.phone
- UNIQUE constraint on users_contact (user_id, contact_number)
- Foreign keys with ON DELETE CASCADE where appropriate
- BEFORE INSERT and BEFORE UPDATE triggers for full_name on users table
- email_logs table exists and is being populated on relevant events

12.2 Backend

- POST /api/auth/register — bcrypt hash, DB insert, email log, return 201
- POST /api/auth/login — bcrypt compare, JWT signed with 15min expiry
- Auth middleware — validates JWT on all protected routes
- Logging middleware — logs all requests
- Full CRUD for contacts (ownership-checked)
- Full CRUD for addresses (ownership chain-checked)
- Full CRUD for tasks (contact ownership cross-validated)
- Consistent error responses with correct HTTP status codes

12.3 Frontend

- Login page — form, validation, token storage, 15-min auto-logout timer
- Register page — all fields, all validations
- Dashboard — protected, shows user's full_name and summary stats
- Contacts page — list, add, edit, delete, link to addresses
- Addresses page — scoped to contact, list, add, edit, delete
- Tasks page — list, add, edit, delete, status update, contact dropdown
- NO UI libraries in package.json
- Auto-logout working on 15-min expiry

12.4 Repository & Delivery

- GitHub/GitLab repository is public or shared with evaluator
- database/schema.sql contains all DDL including triggers
- .env.example with all variable names
- README.md with complete setup instructions

- Demo video (5–10 min) covering all features and the DB trigger proof

---

End of PRD — Full Stack Assessment Task No. 002
Upskill Tech Solution Pvt Ltd