✅ FINAL FULL WORKING PROMPT

NexCA Web Application – Django API + React Frontend + Email Reminders

---

SYSTEM / DEVELOPER PROMPT

You are an expert full-stack developer (Django REST API + React + PostgreSQL + Celery).
Build a complete **NexCA Web Application** as per the specifications below.
The system must be production-ready, scalable, secure, and structured with:

- **Backend:** Django REST Framework (DRF)
- **Frontend:** React (SPA)
- **Database:** PostgreSQL
- **Scheduler:** Celery + Celery Beat

1️⃣ PROJECT OVERVIEW

Build a web application for a **Chartered Accountant Office in India** to manage:

- Client master
- Work categories (GST Return, Income Tax Return, TDS Return, Tax Audit, Statutory Audit, etc.)
- Work recurrence (Monthly / Quarterly / Yearly)
- Automatic creation of due tasks (work instances)
- Status tracking of each work (Not Started → Started → In Progress → Completed / Overdue)
- Staff assignment
- Secure client credential storage (portal logins, etc.)
- Email reminder scheduling to clients (documents, filing, overdue follow-ups)
- Auto-repeat reminders if work is pending
- Auto-next due date and auto-next task generation when current work is completed
- Dashboards and reports for CA office overview

2️⃣ TECH STACK

Backend (API)

- Python 3.x
- Django
- Django REST Framework (DRF)

- PostgreSQL
- Celery + Celery Beat
- Redis (for Celery broker/result backend)
- SMTP (Gmail / domain mail / SendGrid / similar)
- Cryptography (Fernet) for encrypting client credentials

Frontend

- React (SPA)
- React Router
- Axios (or Fetch) for API calls
- State management: Redux Toolkit or React Query / Context API
- UI library (any): Material UI / Ant Design / Chakra UI (choose one and be consistent)
- Form handling: React Hook Form or Formik

DevOps

- Gunicorn + Nginx on Linux (e.g., Contabo / DigitalOcean)
- Environment variables using .env
- HTTPS enabled

---

3. DATABASE DESIGN (BACKEND MODELS)

3.1 Client

Field	Type	Description
id	PK	Auto increment
client_code	String	Unique client code
client_name	String	Legal name
PAN	String	PAN
GSTIN	String	GST number (or move to separate table later)
email	String	Primary email
mobile	String	Mobile number
category	String	Individual / Firm / Company / Trust, etc.
status	Enum	Active / Inactive

Field	Type	Description
created_at	DateTime	Created timestamp
updated_at	DateTime	Updated timestamp

3.2 Work Type (Master)

Field	Description
id	PK
work_name	e.g. GST Return, ITR, TDS Return, Audit
statutory_form	e.g. GSTR-3B, GSTR-1, ITR-3, 26Q
default_frequency	Enum: Monthly / Quarterly / Yearly / One-time
description	Optional details
is_active	Boolean

3.3 Client Work Mapping (Engagements)

Defines which recurring work applies to a given client.

Field	Description
id	PK
client_id	FK → Client
work_type_id	FK → WorkType
freq_override	Optional frequency override (Monthly/Quarterly/Yearly)
start_from_period	String: e.g. “FY 2024-25” / “Apr 2025”
active	Boolean
created_at	DateTime
updated_at	DateTime

Frequency resolution:

- Effective frequency = freq_override if set, else WorkType.default_frequency.

3.4 Work Instances (Tasks)

Each concrete due work for a specific period (e.g., “GSTR-3B – Apr 2025”).

Field	Description
id	PK
client_work_id	FK → ClientWorkMapping
period_label	String: e.g., “Apr 2025”, “Q1 2025-26”, “FY 2025-26”
period_start	Date (optional, helpful for logic)
period_end	Date (optional)
due_date	Date of statutory due date
status	Enum: Not Started / Started / In Progress / Completed / Overdue
assigned_to	FK → User (Staff / Partner)
started_on	Date (nullable)
completed_on	Date (nullable)
remarks	Text
created_at	DateTime
updated_at	DateTime

Automatic Behaviour:

- When a WorkInstance is marked **Completed**, system must:
    - Mark completed_on
    - Cancel pending reminders for this instance
    - Automatically create the **next** WorkInstance based on frequency
    - Automatically create reminder instances for that next task

---

3.5 Credential Vault (Secure Portal Credentials)

Field	Description
id	PK
client_id	FK → Client
portal_type	Enum: GST / IncomeTax / TDS / MCA / Bank / Others
login_url	String

Field	Description
username	String
password_enc	Encrypted string (Fernet AES-256)
extra_info	Text, e.g., security Q&A, notes
last_updated	DateTime

Encryption key to be stored in environment variables and never hardcoded.

3.6 Email Template (per Work Type)

Field	Description
id	PK
work_type_id	FK → WorkType
template_name	e.g., “GST Doc Reminder – T-7 Days”
subject_template	String with placeholders
body_template	Text/HTML with placeholders
is_active	Boolean
created_at	DateTime
updated_at	DateTime

Supported placeholders (must be replaceable in backend):

- {{client_name}}
- {{PAN}}
- {{GSTIN}}
- {{period_label}}
- {{due_date}}
- {{work_name}}
- {{firm_name}} (optional global setting)

3.7 Reminder Rules (Relative to Due Date)

Field	Description
id	PK
work_type_id	FK → WorkType
offset_days	Integer (e.g., -15, -7, -3, 0, +3) relative to due_date
reminder_type	Enum: “Document Reminder” / “Filing Reminder” / “Overdue Reminder”
email_template_id	FK → EmailTemplate
repeat_if_pending	Boolean
repeat_interval	Integer (days between repeats)
max_repeats	Integer (e.g., 3)
is_active	Boolean
created_at	DateTime
updated_at	DateTime

3.8 Reminder Instance (Email Queue)

Field	Description
id	PK
work_instance_id	FK → WorkInstance
reminder_rule_id	FK → ReminderRule
scheduled_at	DateTime (when email should go)
sent_at	DateTime (nullable)
send_status	Enum: Pending / Sent / Failed / Cancelled
repeat_count	Integer
last_attempt_at	DateTime (nullable)
email_to	String (usually client email)
subject_rendered	String (final subject at time of sending)
body_rendered	Text (optional cache; or generate at send time)
created_at	DateTime

3.9 Users and Roles

Use Django's auth system:

- Extend User model with role field:

Field	Description
role	Enum: Partner / Manager / Staff / Admin

Role-based access control for:

- Viewing clients
- Editing tasks
- Viewing credentials
- Configuring reminder rules
- Changing system settings

---

4. BACKEND LOGIC & AUTOMATION

4.1 Task Auto-Creation for Recurring Work

When:

- A **ClientWorkMapping** is created, OR
- A **WorkInstance** is completed,

Backend must:

1. Determine frequency (Monthly / Quarterly / Yearly).
2. Calculate next period start/end and period_label.
3. Calculate next due date (statutory logic: can be configurable or simple rules).
4. Create new WorkInstance with:
    - status = Not Started
    - assigned_to = same as previous (optional)
    - appropriate period_label and due_date.
5. Generate reminders for that new task (see 4.2).

4.2 Generating Reminder Instances for a Task

When a WorkInstance is created:

1. Fetch all active ReminderRules for its WorkType.

2. For each rule:
    - Compute scheduled_at = due_date + offset_days.
    - If scheduled_at >= current_date:
        - Create ReminderInstance:
            - send_status = Pending
            - repeat_count = 0
            - work_instance_id, reminder_rule_id, email_to = client.email.

4.3 Email Sending Job (Celery Task)

- Celery beat runs a periodic task (every 5–10 minutes).

Logic:

1. Fetch all reminders where:
    - send_status = Pending
    - scheduled_at <= now
2. For each reminder:
    - Get related WorkInstance & Client.
    - If WorkInstance.status = Completed:
        - Set send_status = Cancelled
        - Skip sending.
    - Else:
        - Render subject/body from EmailTemplate + placeholders.
        - Send via SMTP.
        - On success:
            - send_status = Sent
            - sent_at = now
            - last_attempt_at = now
        - On failure:
            - send_status = Failed
            - last_attempt_at = now
3. If ReminderRule.repeat_if_pending = True AND WorkInstance is still not Completed:
    - If repeat_count < max_repeats:

- Create new ReminderInstance with:
    - scheduled_at = now + repeat_interval days
    - repeat_count = current_repeat_count + 1
    - send_status = Pending.

4.4 Completing a Task

When API receives status update to Completed:

1. Update WorkInstance:
    - status = Completed
    - completed_on = today
2. Cancel all pending ReminderInstances:
    - send_status = Cancelled where send_status = Pending.
3. Auto-create next WorkInstance (see 4.1).
4. Auto-create reminders for next task (4.2).

4.5 Changing Due Date

When due_date of WorkInstance is updated via API:

1. Find all future ReminderInstances:
    - send_status = Pending
    - scheduled_at > now
2. Mark them Cancelled or delete them.
3. Regenerate ReminderInstances based on the new due_date using ReminderRules.

5️⃣ FRONTEND – REACT (UI / UX REQUIREMENTS)

5.1 Main Pages (SPA with Routing)

1. **Login / Logout**
    - JWT or token-based authentication against Django REST.
    - Remember-me option, token refresh.
2. **Dashboard**
    - Cards / KPIs:
        - Total active clients

        - Pending tasks count
        - Overdue tasks count
        - Today's due tasks
        - Tasks due in next 7 days
    - Table / list:
        - Upcoming tasks (client, work type, period, due date, status, assigned_to)
3. **Client Management**
    - List, filter, search by name/PAN/GSTIN.
    - Add/Edit/View client.
    - View all works associated with client.
    - Open client detail page:
        - Basic info
        - List of active work types
        - List of tasks for that client.
4. **Work Types (Master)**
    - CRUD for Work Types (GST, ITR, TDS, Audit etc.)
    - Set default frequency.
    - Toggle active/inactive.
5. **Client Work Setup**
    - For each client, assign / remove work types.
    - Set frequency override (e.g., GST Quarterly for some clients).
    - Show preview of upcoming periods.
6. **Task / Work Instances List**
    - Filter by:
        - Client
        - Work type
        - Status
        - Staff
        - Period
    - Columns:
        - Client

        - Work type
        - Period
        - Due date
        - Status
        - Assigned to
    - Actions:
        - Change status
        - Assign staff
        - Edit due date
        - View all reminders triggered/scheduled for that task (log view).

7. **Calendar View (Optional but recommended)**
    - Month view showing due tasks as events.
    - Click event → open task detail.
8. **Credential Vault UI**
    - Protected page.
    - List per client, per portal type.
    - Show only username / masked password.
    - Button “Reveal password” (optional, with confirmation).
    - Add/Edit credential.
9. **Email Templates & Reminder Rules UI**
    - Templates:
        - Create / edit email templates for each work type.
    - Reminder Rules:
        - Configure offset days, repeat settings, and link templates.
    - Show a preview: “For GST monthly due 20th April → this rule will schedule on 10th April, repeat every 3 days up to 2 times.”
10. **User & Role Management (Admin only)**
    - Create users (Partner / Manager / Staff).
    - Map role and, optionally, restrict clients per user.
11. **Reports**
    - Work completion report by period, client, work type.

- Staff productivity – tasks completed by staff and time taken.

6️⃣ API DESIGN (SUGGESTED ENDPOINTS)

You don't need to list every field in the URL, but at minimum:

- /api/auth/login/ – login, returns JWT/token.
- /api/auth/refresh/ – refresh token.
- /api/clients/ – list/create.
- /api/clients/{id}/ – retrieve/update/delete.
- /api/work-types/ – CRUD.
- /api/client-works/ – assign work types to clients.
- /api/tasks/ – list / filter / create / update (status, due date, assigned_to).
- /api/tasks/{id}/ – detail.
- /api/tasks/{id}/reminders/ – list reminders for a task.
- /api/credentials/ – CRUD (role protected).
- /api/email-templates/ – CRUD.
- /api/reminder-rules/ – CRUD.
- /api/dashboard/summary/ – counts & KPIs.
- /api/dashboard/upcoming-tasks/ – list for home dashboard.

All endpoints must be:

- Authenticated
- Role-filtered
- Paginated where needed

7️⃣ SECURITY REQUIREMENTS

- Use HTTPS in production.
- Store API keys, DB passwords, email creds, encryption keys in environment.
- Use Django's CSRF and DRF's auth for web + tokens for API.
- Encrypt client portal passwords with Fernet AES-256.
- Audit log for critical operations (optional but recommended).

8️⃣ DEPLOYMENT REQUIREMENTS

- Separate **frontend** and **backend** builds:
    - React build served via Nginx or separate static hosting.
    - Backend on Gunicorn behind Nginx reverse proxy.
- Celery worker & Celery Beat configured as systemd services.
- PostgreSQL tuned for production.
- Proper logging (Django + Celery + Nginx).

9️⃣ DELIVERABLES EXPECTED

- Fully working **Django REST API** project.
- Fully working **React** SPA project.
- Database migrations for all models.
- Sample fixtures (dummy clients, work types, etc.).
- Celery + Redis setup and sample tasks.
- Email send test implementation.
- Role-based auth.
- Clear README and step-by-step setup guide:
    - Local dev setup
    - Environment variables
    - Commands for migrations, Celery, server, frontend build
    - Production deployment steps (Linux, Gunicorn, Nginx).

🔟 FINAL FUNCTIONAL SUMMARY

The final application must:

- Let a CA office **store client details**.
- Configure **work types** like GST, ITR, TDS, Audit with frequency.
- For each client, configure which work applies and its recurrence.
- Auto-create **tasks** with due dates (monthly/quarterly/yearly).
- Track **status** and **assigned staff**.
- Store **login credentials securely** for portals.
- Send **email reminders** to clients:
    - Before due date (for documents)

    - On due date
    - After due date (overdue)
- Repeat reminders as per rules until work completed.
- When work is completed:
    - Stop reminders for that period.
    - Auto-create **next period's task** and its reminders.
- Provide **dashboard & reports** for compliance tracking.