CRM for Employee Productivity Platform — Design & Implementation Plan

> Purpose: Build a lightweight, extensible CRM tailored to integrate tightly with the Employee Productivity Monitoring Software. The CRM will manage contacts, accounts, deals/projects, tasks/tickets, and map time logs from the productivity system to customer work for billing, reporting, and visibility.

---

1. Goals & Key Use Cases

Primary goals - Store and manage customer/company and contact data. - Track deals/projects and associate tasks and time entries from the productivity app. - Provide simple sales pipeline and reporting useful to managers and finance. - Expose APIs and webhooks for two-way sync with the Productivity system. - Secure, auditable, and privacy-friendly by default.

Key use cases 1. Create & manage Contacts and Accounts (companies). 2. Create Projects/Deals and assign to employees or teams. 3. Attach time logs (from agent) to deals/projects for invoicing and reporting. 4. Sync tasks between CRM and Productivity app; status updates flow both ways. 5. Trigger alerts when high-priority customer tasks are idle/overdue. 6. Provide CRM-driven dashboards: sales pipeline, revenue by client, time-per-project.

---

2. High-level Architecture

- **Frontend (CRM UI):** React + Tailwind (single app or separate routes within main UI). Role-based views: Sales, Support, Manager, Admin.
- **Backend:** RESTful API (Django REST Framework or Node.js/Express). Auth via JWT/OAuth2.
- **Database:** PostgreSQL (relational), optional Elasticsearch for search, Redis for caching/queues.
- **Integration layer:** CRM Service that handles API clients, webhooks, and mapping logic to/from Productivity backend.
- **Async jobs:** Celery/RQ + Redis (Django) or BullMQ (Node) for scheduled syncs, retries, heavy reports.
- **Storage:** Object storage (S3) for attachments and large exports.

Diagram (conceptual):

```
Productivity Agent -> Productivity Backend -> CRM Integration Layer <-> CRM
Backend -> CRM Frontend
                                          ^
                                          | (webhooks, API)
External Tools: Email, Payment, Invoicing, Analytics
```

---

3. Data Model (Core Tables)

3.1 Accounts (companies)

- id (uuid)
- name (string)
- industry (string)
- address, phone, website
- primary_contact_id (fk -> contacts)
- metadata (jsonb)
- created_at, updated_at

3.2 Contacts

- id (uuid)
- account_id (fk -> accounts)
- first_name, last_name
- email, phone, job_title
- owner_id (fk -> users)
- tags, metadata (jsonb)
- created_at, updated_at

3.3 Users (internal employees)

- id (uuid)
- name, email, role
- crm_roles (sales/support/manager)
- linked_employee_id (optional, fk -> productivity_users)
- created_at, updated_at

3.4 Deals / Projects

- id (uuid)
- account_id (fk)
- name, description
- value (decimal), currency
- stage (enum: prospecting, negotiation, won, lost)
- owner_id (fk -> users)
- start_date, end_date
- metadata (jsonb)
- created_at, updated_at

3.5 Tasks / Tickets

- id (uuid)
- deal_id (fk)
- title, description
- assigned_to (fk -> users)
- status (enum), priority
- due_date, estimated_hours
- created_at, updated_at

3.6 Time Entries (synced from productivity platform)

- id (uuid)
- user_id (fk -> users)
- deal_id (fk -> deals) nullable
- task_id (fk -> tasks) nullable
- start_time, end_time, duration_seconds
- activity_type (enum: app, website, manual)
- productivity_score (float)
- source (enum: agent, manual)
- created_at, synced_at

3.7 Notes & Attachments

- id (uuid), parent_type (deal/contact), parent_id, author_id, text
- attachments: file metadata -> object storage path

3.8 Audit Log

- id, actor_id, action, object_type, object_id, changes (jsonb), timestamp

---

4. API Design (Representative Endpoints)

> Use RESTful conventions and version the API (e.g., `/api/v1/`). Support pagination, filtering, and sorting.

Auth - `POST /api/v1/auth/login` → returns JWT - `POST /api/v1/auth/oauth/callback` → connect external CRM accounts (if needed)

Accounts & Contacts - `GET /api/v1/accounts` - `GET /api/v1/accounts/{id}` - `POST /api/v1/accounts` - `PUT /api/v1/accounts/{id}` - `GET /api/v1/contacts?account_id=...`

Deals / Projects - `GET /api/v1/deals` - `POST /api/v1/deals` (link to account) - `PUT /api/v1/deals/{id}` - `POST /api/v1/deals/{id}/close` (mark won/lost)

Tasks / Tickets - `GET /api/v1/tasks?assigned_to=...&status=...` - `POST /api/v1/tasks` (can be created by CRM UI or via webhook) - `PATCH /api/v1/tasks/{id}` (update status/hours)

Time Entries - `POST /api/v1/time-entries` (ingest from Productivity backend) - `GET /api/v1/time-entries?deal_id=...&user_id=...&date_from=...`

Sync & Webhooks - `POST /api/v1/webhooks/crm-event` (to receive events from external systems) - `POST /api/v1/integrations/{provider}/oauth` (connect external services)

Reports - `GET /api/v1/reports/pipeline?group_by=stage&date_from=&date_to=` - `GET /api/v1/reports/time-per-deal?deal_id=`

Example JSON: POST /api/v1/time-entries

```
{
  "user_id": "uuid-user-1",
  "deal_id": "uuid-deal-1",
  "task_id": "uuid-task-1",
  "start_time": "2025-09-15T09:00:00Z",
  "end_time": "2025-09-15T10:30:00Z",
  "duration_seconds": 5400,
  "activity_type": "app",
  "productivity_score": 0.82,
  "source": "agent"
}
```

---

5. UI Pages / Components (CRM Frontend)

Main navigation (left sidebar) - Dashboard, Accounts, Contacts, Deals/Projects, Tasks, Time Entries, Reports, Integrations, Settings

Pages & Key components 1. **CRM Dashboard** — pipeline summary (kanban), revenue metrics, recent activity, top accounts, time-per-deal quick view. 2. **Accounts List / Detail** — contacts, deals, time logged, notes, attachments. 3. **Contacts List / Detail** — contact timeline, activity log, associated deals. 4. **Deals / Projects** — kanban view by stage, deal detail with time-entries and tasks. 5. **Tasks** — list, filters, bulk assign, link to deals. 6. **Time Entries** — searchable table, bulk actions, link back to agent data. 7. **Reports** — pipeline report, time-per-client, revenue vs hours. 8. **Integrations** — configure Productivity sync, webhooks, billing/payment. 9. **Settings & Roles** — user/role management, data retention, compliance settings.

UI notes: - Make collection pages filterable (date ranges, account, owner). - Show merged data from the Productivity platform: small badge near time entries like "synced" or "pending". - Inline quick-create for tasks from deal detail.

---

6. Integration Points & Workflows

6.1 Productivity → CRM (Primary)

- Agent → Productivity backend → POST /api/v1/time-entries on CRM (or an intermediary integration layer).
- CRM validates time entry, tries to associate with deal/task (based on task id or mapping rules). If mapping fails, mark as unassigned and notify.

6.2 CRM → Productivity

- Creating a deal/project in CRM can create a corresponding project in Productivity system (via API).
- Assigning tasks in CRM pushes tasks to Productivity app so employees see and log time under the CRM deal.

6.3 Webhooks & Real-time

- CRM exposes webhooks for events (task created/updated, deal stage changed) that Productivity backend can subscribe to.
- Use message queues for reliability.

6.4 Conflict Resolution

- Timestamp-based: last-writer-wins or explicit merge UI.
- Unmapped entries: put into a "to review" queue for manager assignment.

---

7. Security, Privacy & Compliance

- **Authentication:** JWT + refresh tokens; support SSO (SAML/OAuth) for enterprises.
- **Authorization:** Role-based access control (RBAC). Granular permissions for viewing time entries and attachments.
- **Encryption:** TLS in transit; AES-256 for sensitive fields at rest.
- **Data minimization:** Only sync fields needed for CRM-scenarios. Allow opt-out for personal activities.
- **Audit logs:** Every change to deals, time-entries, and user assignments should be auditable.
- **Data retention & deletion:** Configurable retention policies; export and delete features to meet GDPR/Indian rules.
- **Rate limits & API keys:** For public integrations, issue API keys per account with rate-limits and quotas.

---

8. Reporting & Billing

- Build reports that combine CRM value and productivity hours: e.g., `hours_spent_per_deal`, `revenue_per_hour`.
- Support CSV/PDF exports and scheduled email reports.
- If needed, generate invoices by aggregating billable hours per deal and pushing to billing systems.

---

9. MVP Scope & Roadmap

MVP (4–6 weeks) - Accounts, Contacts CRUD - Deals/Projects CRUD - Tasks CRUD + basic Kanban - Ingest time-entries API (manual and agent-driven) - Basic Dashboard (pipeline + time-per-deal) - Simple mapping rules and unresolved queue - Auth + RBAC + basic settings

Phase 2 (next 6–8 weeks) - Webhooks and two-way sync - Advanced reports & exports - Attachments & notes - Search (Elasticsearch) - Integrations (email, invoicing)

Phase 3 (optional) - Predictive analytics (churn, deal win probability) - Multi-currency billing & invoicing - Advanced permission models / enterprise features

---

10. Implementation Checklist & Developer Notes

- Create DB schema migrations for all core tables.
- Implement API auth and user mapping endpoints.
- Build onboarding flows: how to link a productivity user to a CRM user/account.
- Build importer (CSV) for initial account/contact load.
- Add tests: unit tests for API, integration tests for sync flows.
- Monitor: set up logging, metrics (Prometheus) and error alerts (Sentry).

---

11. Example: Quick Mapping Rules

- If time-entry payload contains `deal_id`, attach directly.
- If time-entry has `project_name` and no `deal_id`, try fuzzy-match account/deal by name.
- If user tags the entry with `client:ACME`, parse tag to map to Account ACME.
- If mapping fails, create an `unmapped_time_entries` row for manual review.

Next steps I can help with

- Generate DB migration SQL for PostgreSQL for the core tables.
- Provide starter backend scaffold (Django REST or Express) with auth and Accounts/Deals endpoints.
- Produce React/Tailwind UI mockups and component structure for CRM pages.

Document created to pair with the Employee Productivity Monitoring Software. Adjust scope/timeline based on your team size and priorities.