**CRM Platform: Hyper-Detailed Technical Specification**

**Version:** 1.0  
**Date:** July 26, 2025  
**Audience:** Backend & Frontend Developers, Database Administrators, QA Engineers, System Architects

**1. Core Architectural Principles**

1. **Architecture Style:** Microservices. The CRM will be decomposed into logical business domains (Contacts, Sales, Marketing, etc.) to promote independent development, deployment, and scaling.
2. **Communication:**
   * **Synchronous:** REST APIs for internal service-to-service requests and external client interactions.
   * **Asynchronous:** A message broker (e.g., RabbitMQ, Kafka) will be used for event-driven workflows. Events like Contact.Created, Deal.StageChanged, or Email.Opened will decouple services and enable complex automations.
3. **Database:** Database-per-service pattern. Each microservice owns its data.
   * **contacts-service:** PostgreSQL for strong relational data (Accounts, Contacts).
   * **sales-service:** PostgreSQL for transactional deal data.
   * **activity-service:** Potentially a NoSQL database like MongoDB or DynamoDB to handle high-volume, unstructured data like notes, emails, and timeline events efficiently.
4. **Security:**
   * **Authentication:** JWT (JSON Web Tokens) issued by a dedicated auth-service.
   * **Authorization:** Granular Role-Based Access Control (RBAC) and Hierarchy-Based Access. A Sales Rep can only see their own deals, while a Sales Manager can see their entire team's deals. This is a critical CRM requirement.
   * **Data Isolation:** Strict multi-tenancy at the database level to ensure one company's data is never visible to another. Each database query **must** include a tenant\_id check.
5. **API Standards:** All APIs will be versioned (/api/v1/...). Use standard HTTP verbs. JSON for data interchange. snake\_case for database fields, camelCase for JSON keys.

**Module 1: Contacts & Accounts Service (contacts-service)**

This service is the foundational repository for all people (Contacts) and companies (Accounts). It's the "who" of the CRM.

**Feature 1.1: Create/Update Contacts and Accounts with Duplicate Detection**

* **User Story:** As a Sales Rep, when I enter a new contact's email, the system should check if they or their company already exist to prevent creating duplicate records, and allow me to either merge the information or create a new record.
* **Database Schema Design:**

Generated sql

-- accounts Table (Companies)

CREATE TABLE accounts (

account\_id SERIAL PRIMARY KEY,

tenant\_id INT NOT NULL, -- For Multi-tenancy

account\_name VARCHAR(255) NOT NULL,

website VARCHAR(255),

industry VARCHAR(100),

owner\_user\_id INT NOT NULL, -- The user responsible for this account

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

-- Add an index for tenant and name for fast lookups

UNIQUE (tenant\_id, account\_name)

);

-- contacts Table (People)

CREATE TABLE contacts (

contact\_id SERIAL PRIMARY KEY,

tenant\_id INT NOT NULL,

first\_name VARCHAR(100),

last\_name VARCHAR(100) NOT NULL,

primary\_email VARCHAR(255),

phone\_number VARCHAR(50),

job\_title VARCHAR(100),

owner\_user\_id INT NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

UNIQUE (tenant\_id, primary\_email)

);

-- account\_contacts Table (Many-to-Many Join Table)

-- A contact can be associated with multiple accounts (e.g., consultant)

CREATE TABLE account\_contacts (

account\_id INT NOT NULL REFERENCES accounts(account\_id),

contact\_id INT NOT NULL REFERENCES contacts(contact\_id),

tenant\_id INT NOT NULL,

PRIMARY KEY (tenant\_id, account\_id, contact\_id)

);

-- custom\_fields\_data Table (for extensibility)

CREATE TABLE custom\_fields\_data (

data\_id SERIAL PRIMARY KEY,

tenant\_id INT NOT NULL,

entity\_id INT NOT NULL, -- e.g., contact\_id or account\_id

entity\_type ENUM('contact', 'account', 'deal') NOT NULL,

field\_name VARCHAR(100) NOT NULL,

field\_value TEXT -- Can store various data types

);

* **API Endpoint Specification:**
  + **Endpoint:** POST /api/v1/contacts
  + **Request Body (JSON):**

Generated json

{

"firstName": "John",

"lastName": "Smith",

"primaryEmail": "john.smith@acmecorp.com",

"jobTitle": "VP of Engineering",

"accountName": "Acme Corp", // System can use this to find/create an account

"customFields": {

"leadSource": "Website",

"linkedInProfile": "linkedin.com/in/johnsmith"

}

}

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Json

IGNORE\_WHEN\_COPYING\_END

* **Business Logic:**
  + Extract the domain from primaryEmail (acmecorp.com).
  + Check for existing contacts: SELECT contact\_id FROM contacts WHERE tenant\_id = ? AND primary\_email = ?. If found, return 409 Conflict with the existing contact's ID, prompting the user to merge or cancel.
  + Check for existing accounts: SELECT account\_id FROM accounts WHERE tenant\_id = ? AND (website LIKE '%acmecorp.com' OR account\_name = 'Acme Corp').
  + **Start a Database Transaction.**
  + If no matching account is found, create a new record in the accounts table.
  + Create the new record in the contacts table.
  + Create a record in the account\_contacts join table to link the new contact and the (new or existing) account.
  + Iterate through the customFields object and insert records into custom\_fields\_data.
  + **Commit the Transaction.**
  + **Publish Contact.Created and Account.Created (if applicable) events** to the message broker. The activity-service will listen for this to create a "Contact Created" event on the timeline.
* **Success Response:** 201 CREATED with the full contact and linked account objects.

**Module 2: Sales Pipeline Service (sales-service)**

This service manages deals, pipelines, and revenue forecasting. It's the core operational module for the sales team.

**Feature 2.1: Manage Deals in a Visual Pipeline**

* **User Story:** As a Sales Manager, I want to define the stages of our sales process (e.g., Lead, Qualified, Proposal, Won, Lost) and see all my team's deals organized in these stages, so I can forecast revenue and identify bottlenecks.
* **Database Schema Design:**

Generated sql

-- sales\_pipelines Table

CREATE TABLE sales\_pipelines (

pipeline\_id SERIAL PRIMARY KEY,

tenant\_id INT NOT NULL,

pipeline\_name VARCHAR(100) NOT NULL

);

-- pipeline\_stages Table

CREATE TABLE pipeline\_stages (

stage\_id SERIAL PRIMARY KEY,

tenant\_id INT NOT NULL,

pipeline\_id INT NOT NULL REFERENCES sales\_pipelines(pipeline\_id),

stage\_name VARCHAR(100) NOT NULL,

stage\_order INT NOT NULL, -- To define the order in the pipeline

stage\_type ENUM('Open', 'Won', 'Lost') DEFAULT 'Open',

-- For forecasting: probability of closing from this stage

win\_probability NUMERIC(5, 2)

);

-- deals Table

CREATE TABLE deals (

deal\_id SERIAL PRIMARY KEY,

tenant\_id INT NOT NULL,

deal\_name VARCHAR(255) NOT NULL,

amount NUMERIC(12, 2) NOT NULL,

expected\_close\_date DATE,

stage\_id INT NOT NULL REFERENCES pipeline\_stages(stage\_id),

contact\_id INT NOT NULL REFERENCES contacts(contact\_id), -- Main contact for the deal

account\_id INT NOT NULL REFERENCES accounts(account\_id),

owner\_user\_id INT NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

-- deal\_stage\_history Table (Crucial for Analytics)

CREATE TABLE deal\_stage\_history (

history\_id SERIAL PRIMARY KEY,

tenant\_id INT NOT NULL,

deal\_id INT NOT NULL REFERENCES deals(deal\_id),

from\_stage\_id INT REFERENCES pipeline\_stages(stage\_id),

to\_stage\_id INT REFERENCES pipeline\_stages(stage\_id),

changed\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

time\_in\_previous\_stage\_days INT -- Calculated on change

);

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). SQL

IGNORE\_WHEN\_COPYING\_END

* **API Endpoint Specification (To move a deal):**
  + **Endpoint:** PUT /api/v1/deals/{dealId}/stage
  + **Request Body (JSON):**

Generated json

{

"newStageId": 5

}

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Json

IGNORE\_WHEN\_COPYING\_END

* **Business Logic:**
  + Validate user permissions (Are they the deal owner or a manager?).
  + Retrieve the deal and its current stage\_id.
  + **Start a Database Transaction.**
  + Calculate time\_in\_previous\_stage\_days by comparing CURRENT\_TIMESTAMP with the changed\_at timestamp of the *last* entry in deal\_stage\_history for this deal.
  + Insert a new record into deal\_stage\_history with the old stage, new stage, and calculated duration.
  + Update the stage\_id on the deals table to newStageId.
  + **Commit the Transaction.**
  + **Publish a Deal.StageChanged event.** The message should contain dealId, oldStageId, newStageId, and ownerUserId.
    - *Listener 1 (Automation Service):* If newStageId corresponds to "Proposal Sent," automatically create a task in the activity-service for the ownerUserId: "Follow up on proposal" due in 3 days.
    - *Listener 2 (Analytics Service):* Ingest this event to update sales velocity and conversion rate metrics.

**Module 3: Activity & Timeline Service (activity-service)**

This service captures every interaction with a contact or deal.

**Feature 3.1: Log an Activity and Display on a Timeline**

* **User Story:** As a Sales Rep, after I finish a call with a contact, I want to quickly log the notes of our conversation so that I and my team have a complete history of all interactions.
* **Database Schema Design (Example using NoSQL - MongoDB):**

Generated json

// A single 'activities' collection

{

"\_id": ObjectId("..."),

"tenantId": 101,

"activityId": "act\_12345", // A unique, human-readable ID

"userId": 205, // User who performed the activity

"type": "CALL", // ENUM: CALL, EMAIL, MEETING, NOTE, TASK

"timestamp": "2025-07-26T14:30:00Z",

"content": "John is very interested in the enterprise package. Key pain point is integration with their existing BI tool. Sent follow-up email with API docs.",

"outcome": "Connected", // e.g., 'Left Voicemail', 'Scheduled Demo'

"associations": {

"contacts": [312, 315], // Array of contact\_ids

"accounts": [58],

"deals": [92]

}

}

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Json

IGNORE\_WHEN\_COPYING\_END

* **API Endpoint Specification:**
  + **Endpoint:** POST /api/v1/activities
  + **Endpoint to retrieve a timeline:** GET /api/v1/contacts/{contactId}/timeline
* **Business Logic (for POST):**
  + Validate the incoming data.
  + Insert the document into the activities collection.
  + **Publish an Activity.Created event.**
    - *Listener (Sales Service):* If the activity type is "Meeting" and is associated with a deal, update a "Last Activity Date" field on the deal record to keep it from going stale.
* **Business Logic (for GET Timeline):**
  + Query the activities collection where the associations.contacts array contains the requested {contactId}.
  + Sort the results in descending order by timestamp.
  + Return the paginated list of activities. This provides a complete, chronological history for the user.