North South University

Product Requirements Document (PRD)

Course: CSE327 – Software Engineering

Project Title: Multi-Tenant CRM System
Platform: Web & Mobile Application
Team Size: 5 Members

Submitted to: NBM
Department of Computer Science & Engineering

Date: [27-10-25]

1. Introduction

1.1 Purpose of the Document

This Product Requirements Document (PRD) has been developed to describe the detailed functional and non-functional requirements of the project titled "Multi-Tenant CRM System." The document explains how the system will function, what features it will include, and the constraints under which it will operate. The main purpose is to ensure that all team members, instructors, and evaluators have a clear understanding of what is being built and how it will solve the target problem.

This PRD acts as a formal reference throughout the project life cycle — from planning and design to development, testing, and final deployment.

1.2 Background and Motivation

In today's world, businesses rely heavily on CRM (Customer Relationship Management) systems to handle customer data, communication, and issue tracking.
However, most CRM tools in the market are built for single organizations and require separate installations for every client. This causes additional cost, maintenance overhead, and complexity. To address these limitations, our team decided to design a **multi-tenant CRM system**, where multiple independent organizations (tenants) can use the same platform securely, with isolated data and customized interfaces.

The motivation behind this project comes from the need for:

- A cost-effective CRM system for multiple businesses.
- Centralized issue tracking and communication system.
- Simple yet powerful dashboard for both web and mobile users.
- Support for modern authentication and security mechanisms (OAuth2, JWT).
- Efficient search and analytics features to access data quickly.

1.3 Objectives

The objectives of this project are as follows:

- To design and implement a **multi-tenant CRM system** that supports multiple clients using a single codebase.
- To ensure **data isolation** and **role-based access control** for each tenant.
- To develop both **web** and **mobile** applications for accessibility and ease of use.
- To include **issue tracking**, **chat-based communication**, and **voice command** functionalities.
- To maintain system security through modern authentication techniques.
- To achieve a user-friendly interface that can be easily used by both technical and non-technical users.

1.4 Scope

The system will primarily focus on providing the following features:

- Customer and user management under multiple tenants.
- Support ticket, bug report, and feature request handling.
- Real-time chat between customers and support agents.
- Secure authentication with OAuth2 and JWT tokens.
- Full-text search capability across customers, issues, and messages.
- Voice-based command system for navigation.
- Modular API architecture for scalability and maintainability.

The scope does not include high-end AI automation or third-party CRM integrations at this stage, though such features can be considered in future upgrades.

2. Problem Statement

Customer service and data management are crucial for any business. Small and medium enterprises often rely on free or limited tools that do not support multiple organizations within a single system.
Most existing CRM tools either:

- Are **too expensive** for small organizations,
- Lack **data separation** between clients, or
- Do not provide **cross-platform access** (web + mobile).

Our system addresses these issues by providing a **multi-tenant architecture**, allowing multiple companies to use the same CRM platform independently and securely.
Each company (tenant) will have its own set of users, customers, and issue records, completely isolated from others. This ensures privacy and efficiency.

3. Proposed Solution

Our proposed solution is a **Multi-Tenant CRM System** that combines modern web technologies and scalable design principles.
The system will allow multiple tenants (organizations) to log in, manage customers, handle support tickets, communicate with clients, and track product-related issues—all within one unified platform.

The backend will be built using **Java (Spring Boot)** for reliability and performance. The frontend will use **React.js** to create a dynamic, user-friendly interface.
A **mobile version** will also be developed using **React Native**, ensuring accessibility across devices.

Key Characteristics:

- **Multi-Tenant Access Control:** Each tenant's data remains isolated.
- **Secure Authentication:** OAuth2 + JWT ensures data security.
- **Full-text Search:** Enables searching across large datasets.
- **API-First Architecture:** Ensures modularity and easier integration.

- **Issue Tracking:** Handles customer support tickets and feedback efficiently.
- **Voice Command:** Allows users to interact via speech commands.

4. System Overview

4.1 Users and Roles

The system will have multiple roles to manage different functionalities:

- **Super Admin:** Controls all tenants and manages overall system.
- **Tenant Admin:** Manages users and customers within a tenant.
- **Support Agent:** Communicates with customers and handles issues.
- **Customer/User:** Creates tickets and interacts with support.

4.2 System Modules

The main modules of the system include:

1. **User Management Module** – for registration, role assignment, and authentication.
2. **Customer Management Module** – stores and manages customer data.
3. **Issue Tracking Module** – handles tickets, bug reports, and feature requests.
4. **Chat and Communication Module** – enables messaging between support and customers.
5. **Search Module** – allows users to find records using keywords.
6. **Analytics & Reporting Module** – generates usage and performance reports.
7. **Voice Assistant Module** – integrates voice command and navigation features.

5. System Architecture

5.1 Overview

The system will follow a **Modular, API-First Architecture**.
Each module (User, Customer, Ticket, etc.) will expose its functionalities via RESTful APIs.
This makes the system easily maintainable, scalable, and ready for third-party integrations.

5.2 Multi-Tenant Design

Each tenant's data will be stored separately in a database schema, ensuring complete isolation.
The backend middleware will handle tenant identification using a tenant ID derived from the user's authentication token.

Benefits:

- Data privacy between tenants.
- Independent management and scaling.
- Easier debugging and updates.

5.3 Technology Stack

Layer	**Technology**
Frontend	React.js
Backend	Java
Database	MySQL
Authentication	OAuth2 + JWT
Hosting	AWS / Render

Layer	Technology
Testing	JUnit, Postman

6. Functional Requirements

6.1 User Management

- Admin can create, update, and delete users.
- Each user belongs to a specific tenant.
- Authentication uses JWT token-based security.

6.2 Customer Management

- Tenants can register new customers.
- Customers can communicate via chat or Telegram integration.
- Each customer's history is stored for future reference.

6.3 Issue Tracking

- Support tickets can be created, assigned, updated, or closed.
- Tickets are categorized as **Support Request**, **Bug Report**, or **Feature Request**.
- Each ticket has a priority and status field.

6.4 Chat System

- Real-time text messaging using WebSocket.
- Telegram and email integration.
- Message logs are stored securely.

6.5 Search

- Full-text search across all customer names, tickets, and product details.
- Advanced filters for date range, status, and category.

6.6 Voice Command

- Voice-based navigation and command using speech recognition API.
- Commands like “Show open tickets” or “Search customer John” supported.

7. Non-Functional Requirements

7.1 Security

- Encrypted data transmission using HTTPS.
- OAuth2 and JWT for authentication.
- Role-based access control.

7.2 Performance

- API response time under 500ms.
- Support for 5000+ concurrent users.

7.3 Reliability

- Automatic backups every 24 hours.
- Graceful error handling for failed services.

7.4 Usability

- Simple and clean interface.
- Accessible both on web and mobile.
- Minimal user training required.

7.5 Maintainability

- Modular code structure.
- Automated test coverage above 80%.

8. Constraints and Assumptions

- Internet connectivity is required for all services.
- Each tenant must have unique login credentials.
- The system will initially support English only.
- Payment integration is not included in the first release.
- The project will be developed over one semester following agile methodology.

9. System Design and Data Model

9.1 Overview

The Multi-Tenant CRM System follows a layered architecture.
Each layer performs a specific set of responsibilities ensuring clean separation of concerns and scalability.

Layers include:

- **Presentation Layer:** Frontend (React for web, React Native for mobile).
- **Application Layer:** RESTful API built with Java Spring Boot.
- **Data Access Layer:** Responsible for communication with MySQL and ElasticSearch.
- **Infrastructure Layer:** Includes hosting, load balancing, and CI/CD pipeline.

9.2 Database Design

The system will use a **relational database (MySQL)** with strict tenant-level separation.
Each table will include a tenant_id column to distinguish data between different organizations.

Main Tables

Users Table

Field	Type	Description
user_id	INT	Primary Key
name	VARCHAR(100)	User's full name
email	VARCHAR(150)	Unique email address
password_hash	VARCHAR(255)	Encrypted password
role	ENUM(Admin, Agent, Customer)	User role
tenant_id	INT	Tenant reference

Tenants Table

Field	Type	Description
tenant_id	INT	Primary Key

Field	Type	Description
tenant_name	VARCHAR(150)	Organization name
contact_email	VARCHAR(150)	Admin contact
created_at	DATETIME	Registration time

Tickets Table

Field	Type	Description
ticket_id	INT	Primary Key
title	VARCHAR(255)	Ticket title
description	TEXT	Detailed issue
category	ENUM(Support, Bug, Feature)	Type of request
status	ENUM(Open, In Progress, Closed)	Ticket status
priority	ENUM(Low, Medium, High)	Ticket priority
created_by	INT	User reference
tenant_id	INT	Tenant reference

Messages Table

Field	Type	Description
message_id	INT	Primary Key
sender_id	INT	Reference to user
ticket_id	INT	Related ticket

Field	Type	Description
content	TEXT	Message body
timestamp	DATETIME	Sent time

9.3 Entity Relationship (ER) Overview

- One **Tenant** can have multiple **Users** and **Tickets**.
- One **Ticket** can have multiple **Messages**.
- Users belong to exactly one Tenant.
- Each Message is tied to one Ticket and one User.

This relational structure ensures strong referential integrity and easy reporting.

---

10. System Flow

10.1 Login and Authentication Flow

1. User enters credentials on web or mobile app.
2. Backend verifies using OAuth2 protocol.
3. Upon success, a JWT token is issued containing user_id, tenant_id, and role.
4. Client stores token securely (localStorage or keychain).
5. All subsequent API calls are authorized via the JWT header.

10.2 Ticket Lifecycle Flow

1. **Create Ticket:** Customer submits an issue or request.
2. **Assign Ticket:** Admin assigns ticket to a support agent.
3. **Work in Progress:** Agent updates ticket status and communication.

4. **Resolution:** Once resolved, ticket marked as “Closed”.
5. **Feedback:** Customer provides optional feedback or rating.

10.3 Search and Reporting Flow

- User types a keyword → Search service calls ElasticSearch API →
  Results are filtered by tenant_id →
  Matching records (tickets, messages, customers) displayed instantly.

11. Testing and Quality Assurance

11.1 Testing Strategy

To ensure high quality, different levels of automated and manual testing will be used.

Type	Tools	Purpose
Unit Testing	JUnit (Java), Jest (React)	Test individual components
Integration Testing	Postman, Newman	Verify communication between services
System Testing	Selenium	End-to-end validation
Load Testing	JMeter	Check performance under stress
UAT (User Acceptance)	Manual	Ensure user satisfaction

11.2 Test Coverage Goals

- ➢ Minimum 80% code coverage required before final deployment.
- ➢ CI/CD pipeline will automatically run unit and integration tests on each commit.

- Testing reports stored in the repository for transparency.

12. Risk Analysis

Risk	Impact	Probability	Mitigation Strategy
Server Downtime	High	Medium	Use cloud redundancy and auto-restart policies
Data Breach	Critical	Low	Strong encryption and token validation
Tenant Data Leakage	High	Low	Strict tenant-based query filtering
API Latency	Medium	Medium	Implement caching and indexing
Development Delay	Medium	Medium	Use Agile sprints and regular milestones
Team Skill Gap	Low	Medium	Conduct internal workshops and pair programming

13. Future Enhancements

The system has been designed to be modular so that new functionalities can be added easily in future versions.
Some proposed enhancements include:

1. **AI Chatbot Integration:** For automated customer replies.
2. **Payment and Billing Module:** For tenant subscription management.
3. **Multilingual Support:** To support global users.
4. **Voice-to-Text Ticket Creation:** Advanced NLP integration.
5. **Analytics Dashboard with Charts:** Using data visualization libraries.
6. **Third-party Integrations:** Slack, Jira, and Google Workspace connectivity.

7. **Mobile Offline Mode:** Allowing agents to manage tickets without continuous internet.

14. Project Management and Timeline

The project will follow the **Agile Methodology** divided into five sprints.

Sprint	Duration	Goals
Sprint 1	Week 1–2	Requirement analysis & architecture setup
Sprint 2	Week 3–4	User management and authentication
Sprint 3	Week 5–6	Ticket management and chat module
Sprint 4	Week 7–8	Search, analytics, and voice command features
Sprint 5	Week 9–10	Testing, bug fixing, and final documentation

Progress will be tracked using GitHub Projects and daily stand-up meetings among team members.

15. Tools and Resources

Category	Tools Used
IDE	IntelliJ IDEA, VS Code
Version Control	Git & GitHub
Communication	Slack / Discord
Documentation	Google Docs / Notion
API Testing	Postman
Deployment	AWS / Render
Database	MySQL Workbench
UI Design	Figma

Category	Tools Used
Task Tracking	Jira / Trello

16. Deployment Plan

- Backend and database will be deployed on AWS EC2 or Render Cloud.
- Continuous deployment handled using GitHub Actions.
- Web app hosted on Vercel or Netlify.
- Mobile app published on Google Play Store and TestFlight for testing.

18. Conclusion

The **Multi-Tenant CRM System** is designed to be a secure, scalable, and user-friendly platform for businesses that need efficient customer relationship management under a unified architecture. By supporting both web and mobile platforms, and by ensuring strict data isolation among tenants, this system addresses the major gaps present in traditional CRM software.

The project emphasizes:

- Modern development practices (API-first, modular design).
- Security through OAuth2 and JWT.
- Strong focus on usability and performance.
- Adaptability for future AI and analytics integrations.

Ultimately, this PRD outlines a complete blueprint for developing a robust, enterprise-grade CRM system that can scale with growing organizational needs.