PROJECT SCOPE DOCUMENT

Search Sensei: Multi Tenant SaaS Customer Portal

CLIENT **Search Sensei Pty Ltd**
TYPE RMIT Capstone: 5 person team, shipped to a live paying client
MY FOCUS Billing and identity architecture
STATUS v1.0.0 shipped, in production for a paying client
STACK ASP.NET Core 8, React 18 TypeScript, Keycloak 26, Stripe, BookStack, Zammad, Docker, Azure App Service
SOURCE Private client repository: available for walkthrough on request

1. Overview

Search Sensei wraps an existing enterprise search API in a multi tenant customer portal. Organisations self register, choose a subscription plan, and get an isolated workspace with authentication, billing, a knowledge base, and support ticketing, with zero cross tenant visibility.

Onboarding dropped from manual provisioning to under five minutes once the portal shipped.

The project was delivered as a 5 person RMIT capstone to a live, paying client. My responsibility was the billing and identity architecture: Stripe metered billing, Keycloak multi tenant organisations, and the backend proxy pattern that keeps all downstream credentials off the browser.

2. Problem Statement

Before this project, every new customer of Search Sensei required manual provisioning by the team. There was no self serve onboarding, no tenant isolation mechanism, no subscription billing, and no way for customers to access the company's knowledge base or support system without going through a person. The

client needed a portal that would let organisations sign up independently, pay for a plan, and immediately access their own isolated workspace.

3. In Scope

Identity and multi tenant access

- Self serve organisation registration and onboarding flow
- Multi tenant isolation using Keycloak 26 native Organisations
- OAuth 2.0 Authorization Code flow with PKCE for all browser sessions
- Role based access control across admin and member roles within each tenant
- Single source of truth for tenant identity: Stripe customer and subscription IDs stored as Keycloak organisation attributes, not duplicated into a first party database

Subscription billing

- Stripe Checkout integration for plan selection and payment at registration
- Three metered billing dimensions: search query volume, document pages indexed, and additional sites crawled
- Idempotent usage reporting endpoint with caller supplied idempotency keys, preventing double counting on retries
- Stripe webhook handling for subscription lifecycle events

Backend proxy and third party integrations

- ASP.NET Core 8 backend acting as the single trust boundary for all downstream services
- BookStack knowledge base: proxied read and admin access via Personal Access Token held server side
- Zammad support ticketing: proxied API access with On Behalf Of identity forwarding
- Search API proxy: authenticated forwarding of tenant scoped search requests
- No long lived credentials ever sent to the browser

Frontend

- React 18 TypeScript single page application
- Role based UI routing across distinct user types from a single codebase
- Embedded read only knowledge base and support ticketing views

Deployment

- Containerised services running on Docker
- Deployed to Azure App Service

4. Out of Scope

- Modifications to the underlying Search Sensei search engine or Cosmos DB layer
- Native mobile applications
- Custom email delivery infrastructure (handled by Keycloak and Stripe defaults)
- Advanced analytics dashboards beyond what Stripe and Keycloak provide out of the box
- Hardware integrations

5. Key Architecture Decisions

Metered billing with idempotent usage reporting

Three consumption dimensions are tracked through Stripe: search query volume, document pages indexed, and additional sites crawled. An internal endpoint accepts usage events from backend platform services and requires a caller supplied idempotency key before forwarding them as Stripe meter events. This is the standard pattern for preventing double counting on retries in a financial grade event pipeline.

Subscription and customer IDs are not duplicated into a first party database. They are stored as attributes on the Keycloak organisation itself, so there is a single

source of truth for which org maps to which Stripe customer, rather than two systems that can drift out of sync.

Keycloak Organisations for multi tenancy

Rather than building tenant isolation into application code, the system uses Keycloak 26 native Organisations to scope users, roles, and sessions to an org boundary. The backend resolves the current tenant from an active_tenant claim on every request. Isolation lives in a battle tested layer rather than bespoke logic.

Backend proxy pattern for third party credentials

BookStack and Zammad both require long lived Personal Access Tokens. Rather than sending those to the browser, every call passes through the ASP.NET Core backend, which holds the PAT and returns only what the authenticated user is allowed to see. No dedicated API gateway is needed. The backend itself is the trust boundary.

6. Outcomes

> Onboarding time: manual provisioning reduced to under 5 minutes, fully self service
>
> Credential exposure surface reduced: BookStack and Zammad PATs never reach the browser
>
> Shipped v1.0.0 to a live, paying client on schedule

7. Integration Footprint

Service	Role	Auth method
Keycloak 26	Identity, multi tenant org management, SSO broker	Client Secret + PKCE

Service	Role	Auth method
Stripe	Checkout, subscriptions, metered billing, webhooks	Secret Key + Webhook Signature
BookStack	Knowledge base via backend proxy	PAT via backend proxy
Zammad	Support ticketing via backend proxy	API Token via backend proxy
Search API	Enterprise search, proxied per tenant	Bearer JWT via backend
Azure App Service	Hosting for containerised backend and frontend	N/A

8. Team and My Role

Delivered as a 5 person RMIT capstone team. My specific responsibilities:

- Billing architecture: Stripe Checkout integration, metered billing pipeline, idempotent usage reporting endpoint, webhook handling
- Identity architecture: Keycloak organisation setup, multi tenant isolation strategy, PKCE auth flow, role based access
- Backend proxy pattern: design and implementation of the ASP.NET Core proxy layer for BookStack, Zammad, and the Search API
- Sprint planning and client demos within the Agile delivery process
- AI assisted code review for vulnerability and edge case detection ahead of every merge

---

Source code is in a private client repository. Akash Ramesh is available to walk through implementation in detail during any interview or technical review.