Requirement Document: Appetite Checker SaaS (React + .NET Core API)

# 1. Overview

The Appetite Checker solution enables carriers and agents to evaluate underwriting appetite based on NAICS codes, rules, and carrier-provided guidelines.  
  
It consists of:  
- Frontend (React + Tailwind): Dashboard UI, rule management, search, analytics visualization.  
- Backend (ASP.NET Core API): Exposes APIs for appetite checking, search, rule library, onboarding, analytics.  
- Analytics (Power BI Embedded): Provides dashboards via iframe integration.  
- External Dependencies: NAICS data, optional BizAPI subscription, Power BI service.

# 2. System Architecture

- Frontend (React): Provides interactive dashboards, search, upload, and rule configuration screens. Consumes backend APIs via REST (JSON). Embeds Power BI analytics via iframe.  
  
- Backend (.NET Core API): Implements REST APIs for Checker, Search, Canvas, Analytics. Manages authentication, authorization, rule storage, business classification, analytics logging. Integrates with NAICS datasets and optional BizAPI. Publishes telemetry events to analytics store.

# 3. Solution Components & APIs

# 3.1 Checker APIs (Count: 7)

Purpose: Evaluate submissions against carrier appetite rules using NAICS codes, guidelines, and geography.  
  
Dependencies:  
- Carrier rules & guidelines (internal DB)  
- NAICS taxonomy (free/paid)  
- Optional BizAPI (paid)  
  
API List & Functionality:  
1. POST /evaluate — Takes business description, NAICS code, location; returns eligibility decision (Eligible / Restricted / Declined).  
2. GET /confidenceScore — Returns AI-based confidence score on classification accuracy.  
3. POST /eligibilityCheck — Validates rules for a product/line of business; provides pass/fail and reasoning.  
4. GET /recommendations — Suggests alternate classifications or products when appetite is restricted/declined.  
5. POST /prepareSummary — Generates human-friendly summary of evaluation results (via AI service if available).  
6. GET /result/{id} — Retrieves previously evaluated submission results.  
7. POST /notifyAnalytics — Sends checker events (eligibility outcome, processing time) to analytics pipeline.  
  
Integration:  
- Agent portal (submission workflow).  
- Rating systems (eligibility pre-check).  
- Analytics API (event logging).

# 3.2 Search APIs (Count: 5)

Purpose: Enable fuzzy/keyword search and NAICS mapping for free-text business descriptions.  
  
Dependencies:  
- NAICS & SIC lists (free or paid).  
- Optional synonym dictionaries / BizAPI.  
  
API List & Functionality:  
1. GET /getRules — Retrieves all rules with pagination.  
2. GET /getRulesByKeyword — Keyword search across rules (business type, state, restrictions).  
3. GET /getRulesByNaics/{naicsCode} — Search rules by NAICS code.  
4. GET /getRulesByBusinessType/{type} — Search rules by standard business types.  
5. POST /getRulesByCustomFilter — Advanced filtering by carrier, product, geography.  
  
Integration:  
- Agent portal search bar.  
- Input for Checker APIs.  
- Rule library filters in Canvas APIs.

# 3.3 Canvas APIs (Count: 10)

Purpose: Core platform services for onboarding, rules, products, and dashboards.  
  
Dependencies:  
- Carrier-provided rule & product catalogs.  
- NAICS codes for mapping.  
  
API List & Functionality:  
1. POST /login — Authenticate user.  
2. POST /register — Register new carrier/agent.  
3. GET /user/{id} — Fetch user profile.  
4. GET /users — List all users (admin only).  
5. GET /product/{id} — Fetch product details.  
6. GET /products — List all products.  
7. GET /rule/{id} — Fetch a rule.  
8. GET /rules — List rules with pagination.  
9. POST /rules/upload — Bulk rule upload (CSV/Excel) with validation.  
10. GET /analytics — Provides latest aggregated analytics snapshot.  
  
Integration:  
- Frontend dashboards (React).  
- Search & Checker APIs.  
- Power BI embedding for advanced analytics.

# 3.4 Analytics APIs (Count: 2)

Purpose: Capture telemetry and serve aggregated analytics.  
  
Dependencies:  
- Internal telemetry events (from Checker/Search).  
- Power BI Service (paid).  
  
API List & Functionality:  
1. POST /add — Accepts raw analytics events.  
2. GET /fetch — Returns aggregated analytics (eligibility distribution, submissions over time, appetite share, rules by product).  
  
Integration:  
- Consumed by Canvas /analytics endpoint for lightweight charts.  
- Feeds Power BI reports via data pipeline.  
- Optionally enhanced by AI for natural language summaries.

# 4. Frontend Requirements (React)

- Dashboard: Active Products, Rules, NAICS Classes, Recent Activity.  
- Carrier Onboarding: Forms for carrier/product setup.  
- Product Onboarding: UI to create/manage products.  
- Rule Library: Search, filter, view, add, edit rules.  
- Upload Rules: Upload & validate CSV/Excel, show preview with errors/warnings.  
- Checker Interface: Form to input business description, state, NAICS code; run evaluation and show results.  
- Analytics: Embedded Power BI iframe + lightweight charts.  
- Profile: User profile management.  
  
Integrations:  
- React hooks to call APIs.  
- SignalR integration for live updates.  
- Secure Power BI embedding.

# 5. Non-Functional Requirements

- Authentication & Authorization: JWT tokens (ASP.NET Identity).  
- Multi-tenancy: Tenant separation for carriers.  
- Scalability: Deploy on Azure App Services + SQL DB.  
- Real-time Updates: SignalR for status push.  
- Analytics Security: Power BI Embedded with service principal.  
- APIs: REST (JSON), versioned (/api/v1/).  
- Testing: Unit + integration tests.

# 6. API Fit in Appetite Checker Solution

- Checker APIs: Core evaluation engine.  
- Search APIs: Pre-processing for NAICS mapping.  
- Canvas APIs: Platform services for onboarding and rule management.  
- Analytics APIs: Feedback loop for insights and trends.

Snapshots-

1. ) Appetite Checker Search Screen this would be integrated to our existing Agent Portal – Sai is working on this   
   A screenshot of a computer

   AI-generated content may be incorrect.
2. ) Canvas Portal (Rule Management System)

FrontEnd – React (Saketh)

BackEnd - .NetCoreAPI (Veena)

A screenshot of a login screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.