NOTE TO DEV TEAM

> **CRITICAL DEVELOPMENT STRATEGY**
>
> Please note: This plan for application development may necessitate to commence work essentially from the "end" (the desired user experience and interface) and work backward to the beginning of the build process. By defining the final, integrated UI/UX through a robust design system before the bulk of the engineering begins, we may ensure the (multiple) modular integrations' continuity across this Review Engine application. Accordingly we may prevent a business partner's/user fragmented experience and guarantee that the user-facing features of all modules align with the core strategic vision of a powerful single-source, unified, streamlined, seamless business engine. This strategy is critical for this Review Engine's streamlined function and appearance and for scaling of the application with future functions and modules.

Review Engine

> **Enterprise White-Label Review Analytics Platform**
> *Production-Grade, AI-Powered, Multi-Tenant Review Intelligence System*

Table of Contents

Overview

Review Engine is a comprehensive platform for aggregating, analyzing, and acting on customer reviews from multiple sources. The system leverages artificial intelligence to extract actionable insights and automatically triggers loyalty and marketing workflows.

Core Purpose

- Aggregate reviews from Google, Yelp, Facebook, and Synup
- Analyze sentiment and extract themes using AI (Gemini Pro + Claude Haiku)
- Cluster insights by geospatial regions using H3 hexagons
- Reward customers automatically via Yotpo loyalty program
- Trigger marketing campaigns through Klaviyo

Quality-First Approach

```
Principles:
  - Quality over speed
  - Modular development (1 sprint = 1 complete feature)
  - Production-ready at every step
  - ≥90% test coverage on all code
  - Zero shortcuts on security
```

✨ Key Features

📊 Review Management

- ☑ Multi-platform review aggregation (Google, Yelp, Facebook, Synup)
- ☑ Real-time sentiment analysis with AI
- ☑ Theme and topic extraction
- ☑ Review filtering and search
- ☑ Automated response suggestions

🌍 Community Intelligence

- ☑ Geospatial clustering (H3 hexagons)
- ☑ Neighborhood-level insights
- ☑ Community trend analysis
- ☑ Location-based performance metrics

🎁 Loyalty Integration

- ☑ Yotpo rewards integration
- ☑ Automatic point allocation
- ☑ Referral program management
- ☑ Real-time reward notifications

📧 Marketing Automation

- ☑ Klaviyo campaign triggers
- ☑ Personalized email workflows
- ☑ GDPR-compliant consent management
- ☑ Automated follow-up sequences

Native Mobile Apps

- [x] iOS app (App Store)
- [x] Android app (Google Play)
- [x] Biometric authentication
- [x] Offline-first sync
- [x] Push notifications
- [x] QR code scanning

White-Label Platform

- [x] Custom branding per tenant
- [x] Dynamic theme system
- [x] Custom domain support
- [x] Tenant isolation (Row-Level Security)

E-Commerce Platform

- [x] Multi-provider integration (Shopify, WooCommerce, custom)
- [x] Product catalog management
- [x] Order processing and fulfillment tracking
- [x] Customer purchase history and analytics
- [x] Automatic product review invitations
- [x] Customer LTV calculation (purchase + review + loyalty data)

Enterprise Security

- [x] SAML SSO authentication
- [x] Role-Based Access Control (RBAC)
- [x] Row-level security (PostgreSQL RLS)
- [x] AES-256 encryption at rest
- [x] API key management
- [x] Rate limiting & DDoS protection

Technology Stack

Backend

Component	Technology	Version
Language	Python	3.11+
Framework	FastAPI	0.104+
Database	CockroachDB	Serverless
Cache	Redis	7+
Background Jobs	Celery	5.4+

Component	Technology	Version
AI Models	Gemini Pro / Claude 3	Latest
Authentication	Auth0	SAML + OIDC
Hosting	Render.com	Production

Key Libraries:

```
httpx              # Async HTTP client
pydantic           # Data validation
sqlalchemy         # Database ORM
celery             # Task queue
redis              # Caching
h3                 # Geospatial indexing
opentelemetry      # Distributed tracing
structlog          # Structured logging
tenacity           # Retry logic
```

Frontend (Web)

Component	Technology	Version
Framework	Next.js	14 (App Router)
Language	TypeScript	5+
UI Components	shadcn/ui	Latest
Styling	Tailwind CSS	3+
State Management	TanStack Query	v5
Forms	React Hook Form + Zod	Latest
Charts	Recharts	2+
Maps	React Leaflet	4+
Hosting	Vercel	Edge Network

Mobile (iOS & Android)

Component	Technology	Version
Framework	React Native	0.73+ (Expo)
Language	TypeScript	5+
UI Library	React Native Paper	5+
Navigation	React Navigation	6+

Component	Technology	Version
State	Zustand	4+
Offline DB	WatermelonDB	0.26+
Notifications	Expo Push	Latest
Analytics	PostHog	3+

Integrations

- **Loyalty:** Yotpo Loyalty & Referrals API
- **Marketing:** Klaviyo Marketing Automation API
- **Reviews:** Synup, Google My Business, Yelp Fusion, Facebook
- **E-Commerce:** Shopify, WooCommerce, custom storefronts
- **Monitoring:** Datadog, Sentry, PagerDuty
- **Security:** Cloudflare WAF, Auth0

🚀 Quick Start

Prerequisites

- Python 3.11+
- Node.js 18+
- Docker & Docker Compose
- Git

Backend Setup (5 minutes)

```
# Clone repository
git clone https://github.com/Merriwood/Data_Pipeline.git
cd Review-Engine-Workshop-v2

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
poetry install

# Set up environment variables
cp .env.example .env
# Edit .env with your API keys

# Start services with Docker
docker-compose up -d

# Run migrations
poetry run alembic upgrade head
```

```
# Start API server
poetry run uvicorn app.main:app --reload
```

Visit: `http://localhost:8000/docs` (Interactive API docs)

Frontend Setup (5 minutes)

```
# Navigate to web directory
cd review-engine-web

# Install dependencies
npm install

# Set up environment
cp .env.example .env.local

# Start development server
npm run dev
```

Visit: `http://localhost:3000`

Mobile Setup (5 minutes)

```
# Navigate to mobile directory
cd review-engine-mobile

# Install dependencies
npm install

# Start Expo development server
npx expo start

# Scan QR code with Expo Go app
```

Project Structure

```
Review-Engine-Workshop-v2/
├── review-engine-ui/                    # All UI/UX and frontend code
│   ├── app/                            # Next.js app directory
│   ├── components/                     # React components
│   ├── package.json                    # Frontend dependencies
│   ├── tsconfig.json                   # TypeScript config
│   ├── tailwind.config.js              # Tailwind CSS config
│   └── [other frontend files]
│
```

```
├── app/                                  # FastAPI application (backend)
│   ├── api/                             # API endpoints
│   ├── models/                          # SQLAlchemy ORM models
│   ├── services/                        # Business logic
│   ├── integrations/                    # External API clients
│   └── main.py                          # FastAPI app factory
│
├── tests/                               # Test suite
├── alembic/                             # Database migrations
├── docker-compose.yml                   # Local development
├── pyproject.toml                       # Poetry dependencies
├── README.md                            # This file
└── README.pdf                           # Project summary (PDF)
```

Frontend Organization

All UI/UX files are exclusively within `review-engine-ui/`:

- Module-based dashboard and analytics
- Component library (shared + module-specific)
- Configuration files
- Frontend dependencies

Recommended Frontend Structure:

```
review-engine-ui/
├── app/
│   ├── [tenant]/
│   │   ├── dashboard/          # Qrvey aggregator (unified view)
│   │   ├── modules/
│   │   │   ├── reviews/        # Review analytics module
│   │   │   ├── ecommerce/      # E-commerce module (NEW)
│   │   │   ├── loyalty/        # Yotpo integration
│   │   │   ├── marketing/      # Klaviyo integration
│   │   │   └── analytics/      # Cross-module insights
│   │   └── [layout files]
│   └── [other app files]
├── components/
│   ├── shared/                 # Reusable across all modules
│   │   ├── Navigation.tsx
│   │   ├── TenantHeader.tsx
│   │   ├── ModuleCard.tsx
│   │   ├── LoadingState.tsx
│   │   └── ErrorBoundary.tsx
│   └── modules/                # Module-specific components
│       ├── reviews/
│       ├── ecommerce/
│       ├── loyalty/
│       └── marketing/
├── hooks/                      # Shared custom React hooks
├── lib/
```

```
│   ├── module-registry.ts     # Module configuration & feature flags
│   └── [other utilities]
├── constants/                  # Shared constants and enums
└── [other frontend files]
```

Module Architecture & Integration

Module Registry System

Each module is self-contained with feature flag support for per-tenant enablement:

```
// lib/module-registry.ts
export const MODULE_REGISTRY = {
  reviews: {
    name: 'Review Analytics',
    icon: 'Star',
    path: '/modules/reviews',
    enabled: true,
    requiredPermissions: ['view_reviews', 'manage_reviews']
  },
  ecommerce: {
    name: 'E-Commerce',
    icon: 'ShoppingCart',
    path: '/modules/ecommerce',
    enabled: true,  // Per-tenant feature flag
    requiredPermissions: ['view_ecommerce', 'manage_products']
  },
  loyalty: {
    name: 'Loyalty & Rewards',
    icon: 'Gift',
    path: '/modules/loyalty',
    enabled: true,
    requiredPermissions: ['view_loyalty', 'manage_rewards']
  },
  marketing: {
    name: 'Marketing Automation',
    icon: 'Mail',
    path: '/modules/marketing',
    enabled: true,
    requiredPermissions: ['view_campaigns', 'create_campaigns']
  },
  analytics: {
    name: 'Cross-Module Analytics',
    icon: 'TrendingUp',
    path: '/modules/analytics',
    enabled: true,
    requiredPermissions: ['view_analytics']
  }
};
```

Qrvey Dashboard Integration

The Qrvey dashboard acts as a unified aggregation layer, combining insights from all modules:

```
Qrvey Dashboard
├── Review Sentiment Trends (reviews module)
├── Product Performance (ecommerce module)
├── Customer Lifetime Value (loyalty + ecommerce)
├── Campaign ROI (marketing module)
├── Community Insights (analytics module)
└── Real-time Alerts (cross-module)
```

E-Commerce Integration Pattern

Tight Coupling Scenario (Recommended):

- Customer purchase events trigger loyalty point awards
- Product reviews influence featured products on storefront
- Purchase history influences review recommendations
- Ecommerce metrics feed into community intelligence

Backend Services:

```
# app/services/ecommerce_service.py
- Product catalog management
- Order processing and fulfillment
- Customer purchase tracking
- Inventory management

# app/integrations/ecommerce.py
- Shopify API client
- WooCommerce API client
- Custom storefront provider

# app/models/ecommerce.py
- EcommerceProduct entity
- CustomerPurchase entity
- ProductReview cross-reference
```

Data Flow:

```
Customer Purchase (Ecommerce)
        ↓
Celery Task Hook: on_product_purchased
        ↓
├─ Loyalty Module: Award points
├─ Marketing Module: Send thank-you email
```

```
  ├─ Reviews Module: Request product review
  └─ Analytics Module: Update customer LTV
```

🚀 Implementation Roadmap: Next Phases

Phase 1: Foundation (Week 1-2) - CRITICAL

Objective: Establish modular architecture foundation for all modules

Priority 1a: Shared Components Library

Create `review-engine-ui/components/shared/` with reusable UI components:

- Navigation component (module-aware)
- TenantHeader with logo/branding
- ModuleCard for dashboard
- LoadingState and ErrorBoundary
- Common form components

Why First: Prevents code duplication, ensures UI consistency across modules, enables faster module development.

Priority 1b: Module Registry System

Implement `review-engine-ui/lib/module-registry.ts`:

- Define MODULE_REGISTRY with all module metadata
- Feature flag system per tenant
- Permission matrix per module
- Dynamic module loading

Why First: Enables Qrvey dashboard to automatically discover and display available modules.

Deliverable: Dashboard automatically shows/hides modules based on tenant feature flags.

Phase 2: Core Modules (Week 3-6)

Objective: Implement ecommerce module and RBAC system

Priority 2a: E-Commerce Module

- `app/services/ecommerce_service.py` - Business logic
- `app/integrations/ecommerce.py` - Shopify/WooCommerce clients
- `app/models/ecommerce.py` - Product, Order, Purchase entities
- `review-engine-ui/components/modules/ecommerce/` - UI components
- `app/api/v1/endpoints/ecommerce.py` - REST API endpoints

Priority 2b: RBAC Middleware & Permissions

- `app/core/permissions.py` - Permission matrix definition
- `app/middleware/auth.py` - RBAC middleware enforcement
- Permission checks on all module endpoints
- Per-tenant role assignments

RBAC Matrix Example:

```
RBAC_MATRIX = {
    'admin': ['all_permissions'],
    'manager': ['view_all_modules', 'manage_reviews', 'manage_campaigns'],
    'viewer': ['view_all_modules'],
    'reviewer': ['view_reviews', 'respond_to_reviews']
}
```

Priority 2c: Celery Task Hooks

- `app/tasks.py` - Background job definitions
- `on_product_purchased` - Triggers loyalty + marketing + analytics updates
- `on_review_submitted` - Updates product ratings, notifies team
- `on_campaign_created` - Logs to analytics module

Phase 3: Optimization & Scaling (Week 7-14)

Objective: Performance, monitoring, and production readiness

Priority 3a: Performance Optimization

- TTL caching strategy per module (5min-1hr based on update frequency)
- Database connection pooling
- API response time optimization
- Frontend lazy-loading of modules

Priority 3b: Feature Flags System

- Integrate with LaunchDarkly or Unleash
- Per-tenant feature flag support
- A/B testing infrastructure
- Rollout safety mechanisms

Priority 3c: Module-Specific Logging & Metrics

- Structured logging per module
- Module-specific Datadog dashboards
- Performance metrics and SLAs
- Usage analytics per module

Priority 3d: Database Schema Enhancements

- Add `ecommerce_enabled`, `ecommerce_provider`, `ecommerce_config` to Tenant model
- Create EcommerceProduct, CustomerPurchase, ProductReview entities
- Add indexes for module-specific queries
- Implement archival strategy for old data

🏛 Architecture Decisions

Decision 1: Modular Module Design

☑ **Chosen:** Module-first architecture (reviews, ecommerce, loyalty, marketing, analytics)

- **Rationale:** Enables independent scaling, feature flags per tenant, clear API boundaries
- **Alternative:** Monolithic feature-first (rejected - creates tight coupling)

Decision 2: E-Commerce Integration Depth

☑ **Recommended:** Tight Coupling - Customer purchase events trigger loyalty/marketing/reviews workflows

- **Rationale:** Purchase history influences review likelihood, loyalty rewards drive repeat purchases
- **Tight Coupling Benefits:**
  - Customer LTV calculation includes purchase + review + loyalty data
  - Personalized product recommendations based on behavior
  - Seamless workflow automation
- **Loose Coupling Alternative:** Standalone storefront (if ecommerce is separate from platform)

Decision 3: Dashboard Aggregation

☑ **Chosen:** Qrvey as unified dashboard hub

- **Rationale:** Single source of truth, cross-module insights, executive visibility
- **Alternative:** Separate dashboards per module (rejected - fragmented UX)

Decision 4: Feature Flags

☑ **Recommended:** Implement module-level feature flags

- **Rationale:** Support progressive rollout, A/B testing, per-tenant module enablement
- **Tools:** LaunchDarkly, Unleash, or Flags as a Service

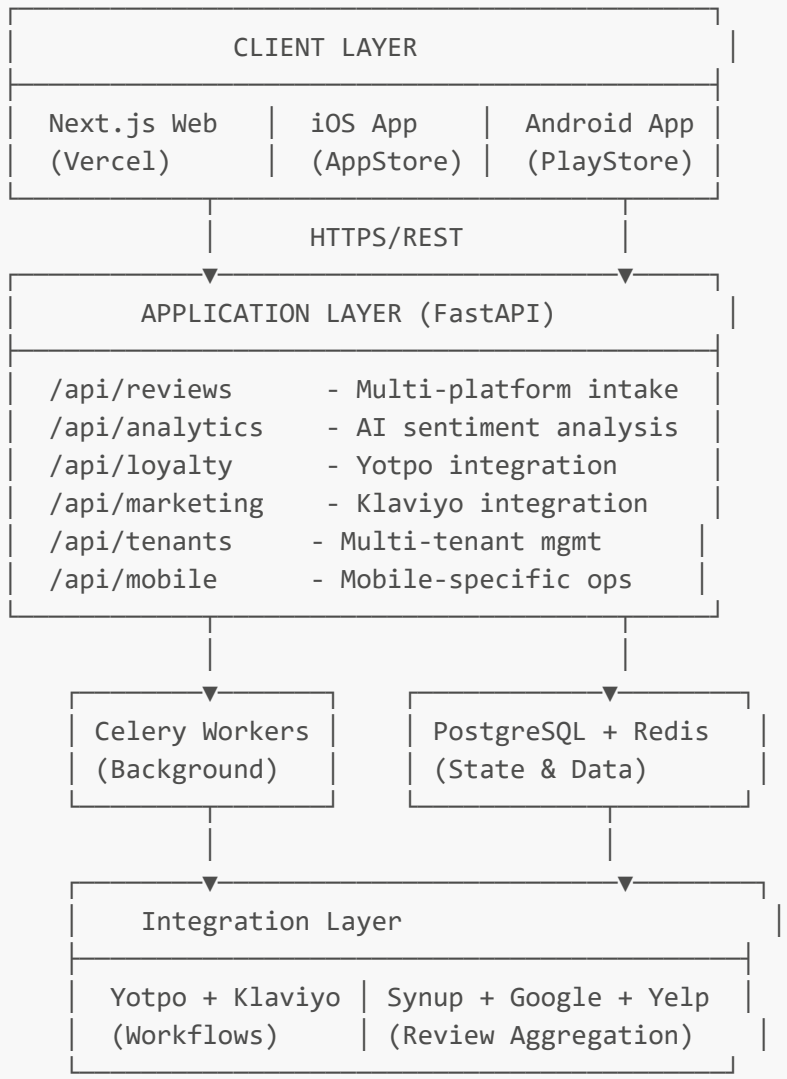
Decision 5: RBAC Implementation

☑ **Recommended:** Permission matrix + middleware enforcement

- **Rationale:** Secure multi-tenant access, clear permission definitions, audit trail
- **Pattern:** Attribute-Based Access Control (ABAC) optional for advanced scenarios

---

System Diagram



Data Flow: Review → Loyalty Workflow

```
1. Customer leaves 5⭐ review on Google
        ↓
2. Synup webhook triggers your API
        ↓
3. FastAPI validates with Pydantic
        ↓
4. AI (Gemini) analyzes sentiment & themes
        ↓
5. Community engine assigns H3 hexagon
        ↓
6. Celery task: Yotpo.award_points(100)
        ↓
7. Yotpo webhook confirms points awarded
        ↓
8. Klaviyo triggers "Thank you" email
        ↓
9. Mobile push: "You earned 100 points!"
```

```
        ↓
10. Dashboard analytics update in real-time
```

📅 Development Roadmap

14-Week Sprint Plan

Sprint	Weeks	Focus Area	Key Deliverables
0	1	Foundation	Greenfield setup, walking skeleton
1	1	Review Ingestion	Synup client, pagination, validation
2	1	AI Analytics	Gemini integration, sentiment analysis
3	1	Community Sharding	H3 geospatial indexing, clustering
4	1	Multi-Tenancy	RLS, tenant isolation, migrations
5	1	Authentication	Auth0 JWT, RBAC, API keys
6	1	Integration Framework	OAuth 2.0, webhooks, credentials
7	1	Yotpo Loyalty	API client, auto-awards, triggers
8	1	Klaviyo Marketing	API client, campaigns, GDPR
9	1	Background Jobs	Celery setup, Redis, scheduling
10	2	Next.js Web App	Dashboard, analytics, white-label
11	2	React Native Mobile	Native apps, biometric auth, offline
12	1	Performance	Optimization, caching, load testing
13	1	Security	OWASP, penetration testing, audit
14	1	Production Deploy	Infrastructure, monitoring, launch

Timeline

- **Duration:** 14 weeks (98 days)
- **Approach:** Quality-first, not speed-first
- **Exit Criteria:** All 14 sprints must meet quality gates
- **Flexibility:** Can extend sprints for quality

🔧 Getting Started

1. Environment Setup

```
# Clone the repository
git clone https://github.com/Merriwood/Data_Pipeline.git
```

```
cd Review-Engine-Workshop-v2

# Create .env file with your credentials
cp .env.example .env

# Required API Keys:
# - GOOGLE_GEMINI_API_KEY
# - AUTH0_DOMAIN, AUTH0_CLIENT_ID, AUTH0_CLIENT_SECRET
# - SYNUP_API_KEY
# - YOTPO_API_KEY
# - KLAVIYO_API_KEY
# - DATABASE_URL (PostgreSQL or CockroachDB)
# - REDIS_URL
```

2. Start Local Services

```
# Start PostgreSQL, Redis, and other services
docker-compose up -d

# Verify services are running
docker-compose ps
```

3. Run Database Migrations

```
# Apply all pending migrations
poetry run alembic upgrade head

# Create initial data (optional)
poetry run python scripts/seed_database.py
```

4. Start Development Servers

Backend:

```
poetry run uvicorn app.main:app --reload
# API available at http://localhost:8000
# Docs at http://localhost:8000/docs
```

Frontend:

```
cd review-engine-ui
npm run dev
# Web app available at http://localhost:3000
```

Mobile:

```
cd review-engine-mobile
npx expo start
# Scan QR code with Expo Go app
```

5. Run Tests

```
# All tests
poetry run pytest --cov

# Specific test file
poetry run pytest tests/unit/test_reviews.py -v

# With coverage report
poetry run pytest --cov --cov-report=html
```

API Documentation

Interactive API Docs

- **Swagger UI:** http://localhost:8000/docs
- **ReDoc:** http://localhost:8000/redoc
- **OpenAPI JSON:** http://localhost:8000/openapi.json

Key Endpoints

Reviews

```
GET    /api/reviews              - List all reviews
GET    /api/reviews/{id}         - Get review details
POST   /api/reviews              - Create review
PATCH  /api/reviews/{id}         - Update review
DELETE /api/reviews/{id}         - Delete review
```

Analytics

```
GET    /api/analytics/dashboard  - Dashboard metrics
GET    /api/analytics/sentiment  - Sentiment analysis
GET    /api/analytics/communities - Community insights
GET    /api/analytics/trends     - Trend analysis
```

Loyalty

```
POST   /api/loyalty/setup        - Initialize Yotpo integration
POST   /api/loyalty/awards       - Create reward
GET    /api/loyalty/balance      - Get customer balance
```

Marketing

```
POST   /api/marketing/setup      - Initialize Klaviyo integration
POST   /api/marketing/campaigns  - Create campaign
GET    /api/marketing/analytics  - Campaign performance
```

Mobile Apps

Features

- ☑ **Biometric Authentication** (Face ID, Fingerprint)
- ☑ **Offline-First** - Work without internet, sync when connected
- ☑ **Push Notifications** - Real-time updates
- ☑ **QR Code Scanning** - Quick review access
- ☑ **Deep Linking** - Direct access to reviews
- ☑ **App Launch:** <2 seconds

Deployment

iOS:

```
cd review-engine-mobile
eas build --platform ios
eas submit --platform ios
```

Android:

```
cd review-engine-mobile
eas build --platform android
eas submit --platform android
```

Testing

- Internal testing via TestFlight (iOS)
- Internal testing via Google Play (Android)
- Public release after QA

White-Label System

Custom Branding Per Tenant

```
// Automatic theme loading based on hostname
GET /api/tenants/{tenant}/theme
→ {
    "primaryColor": "#007AFF",
    "secondaryColor": "#34C759",
    "logoUrl": "https://...",
    "companyName": "Client Name",
    "customDomain": "reviews.clientname.com"
}
```

Usage

```
// app/[tenant]/layout.tsx
export default async function Layout({ params }) {
  const theme = await getTenantTheme(params.tenant);

  return (
    <html>
      <head>
        <style>{`
          :root {
            --primary: ${theme.primaryColor};
            --secondary: ${theme.secondaryColor};
          }
        `}</style>
      </head>
      <body>
        <Header logo={theme.logoUrl} />
        {children}
      </body>
    </html>
  );
}
```

Custom Domains

- `reviews.acmecorp.com` → Dedicated tenant instance
- Full domain ownership & SSL
- White-label UI with custom branding
- Tenant isolation at database level

Security

Authentication & Authorization

- **SSO:** Auth0 with SAML 2.0
- **API Keys:** Tenant-specific keys for integrations
- **JWT Tokens:** 1-hour expiration, refresh tokens
- **RBAC:** Admin, Manager, Viewer roles

Data Protection

- **Encryption:** AES-256 at rest
- **TLS 1.3:** All data in transit
- **Row-Level Security:** PostgreSQL RLS policies
- **Secrets Management:** Environment variables only

Compliance

- ☑ **GDPR:** Data deletion, portability, consent
- ☑ **CCPA:** Privacy notices, opt-out mechanisms
- ☑ **OWASP Top 10:** All mitigations implemented
- ☑ **Penetration Testing:** Annual audits

Quality Gates (Every Sprint)

```
Code Quality:
  ✓ pytest --cov ≥90%
  ✓ mypy --strict = 0 errors
  ✓ ruff check = 0 violations
  ✓ bandit = 0 critical issues

Security:
  ✓ No hardcoded secrets
  ✓ Pydantic validation on all inputs
  ✓ Dependencies scanned (safety)

Performance:
  ✓ API P95 <200ms
  ✓ Mobile <2s page load
  ✓ DB queries <50ms
```

📊 Monitoring & Operations

Logging & Tracing

- **Structured Logs:** JSON format via structlog
- **Distributed Tracing:** OpenTelemetry + Datadog
- **Error Tracking:** Sentry integration
- **Uptime Monitoring:** Pingdom

Alerting

```
Critical:
  - Error rate >5%
  - API response >500ms
  - Database connection pool exhausted
  - Out of memory (>90%)

High:
  - Slow query logs detected
  - Redis eviction rate high
  - Celery task backlog growing

Medium:
  - Long GC pauses
  - Cache hit ratio <80%
  - External API latency spike
```

Dashboards

- **Main Dashboard:** Overview of all metrics
- **API Performance:** Request latency, throughput
- **Database:** Query performance, connections
- **Celery:** Task queue, worker health
- **Errors:** Error rates by endpoint

📖 Documentation

- **Development Instructions** - Complete 14-week plan
- **API Docs** - Endpoint reference
- **Architecture Guide** - System design
- **Deployment Guide** - Production setup
- **Security Guide** - Safety practices

🤝 Contributing

Code Quality Requirements

- All tests pass: `poetry run pytest --cov`
- Type checking: `poetry run mypy app/ --strict`
- Linting: `poetry run ruff check app/`
- Security: `poetry run bandit -r app/`

Pull Request Process

1. Create feature branch: `git checkout -b feature/your-feature`
2. Write tests and code

3. Run all checks: `poetry run pre-commit run --all-files`
4. Commit: `git commit -m "feat: description"`
5. Push and open PR
6. Address review comments
7. Merge when approved

Commit Convention

```
feat:   New feature
fix:    Bug fix
docs:   Documentation
style:  Code style
refactor: Refactoring
perf:   Performance
test:   Tests
chore:  Dependencies, setup
```

📞 Support

Getting Help

- **Issues:** GitHub Issues
- **Discussions:** GitHub Discussions
- **Email:** support@reviewengine.com
- **Slack:** #review-engine channel

Reporting Bugs

Include:

- Python/Node version
- Error message & stack trace
- Steps to reproduce
- Expected vs. actual behavior
- Environment (local/staging/prod)

On-Call Support

- **Critical:** <15 min response (24/7)
- **High:** <1 hour response
- **Medium:** <4 hour response
- **Low:** Business hours

📋 Success Metrics

Technical KPIs

Metric	Target	Current
API Uptime	>99.9%	—
Page Load P95	<150ms	—
Mobile Launch	<2s	—
Error Rate	<0.1%	—
Test Coverage	>90%	—

Business KPIs

Metric	Target	Current
Onboarding Time	<30 min	—
Time to First Insight	<5 min	—
Mobile App Rating	>4.5 ⭐	—
Customer Satisfaction	>90%	—

📝 Revision History

Version	Date	Changes
2.0	Dec 2024	Complete rewrite with mobile apps, white-label, greenfield approach
1.0	Dec 2024	Initial brownfield refactoring plan

📜 License

👥 Team

- **Lead Architect:** AI Development Team
- **Backend Lead:** [Name]
- **Frontend Lead:** [Name]
- **DevOps Lead:** [Name]
- **Product Manager:** [Name]

🎯 Ready to Build

This is a **complete, production-ready development plan** for an enterprise-grade review analytics platform. All components, integrations, security practices, and deployment strategies are documented and ready for implementation.

Status: ☑ Ready for Execution
Last Updated: December 2024
Questions? Review with senior architect before Sprint 0

Ready to build world-class software! 🚀