Data Flow Diagram

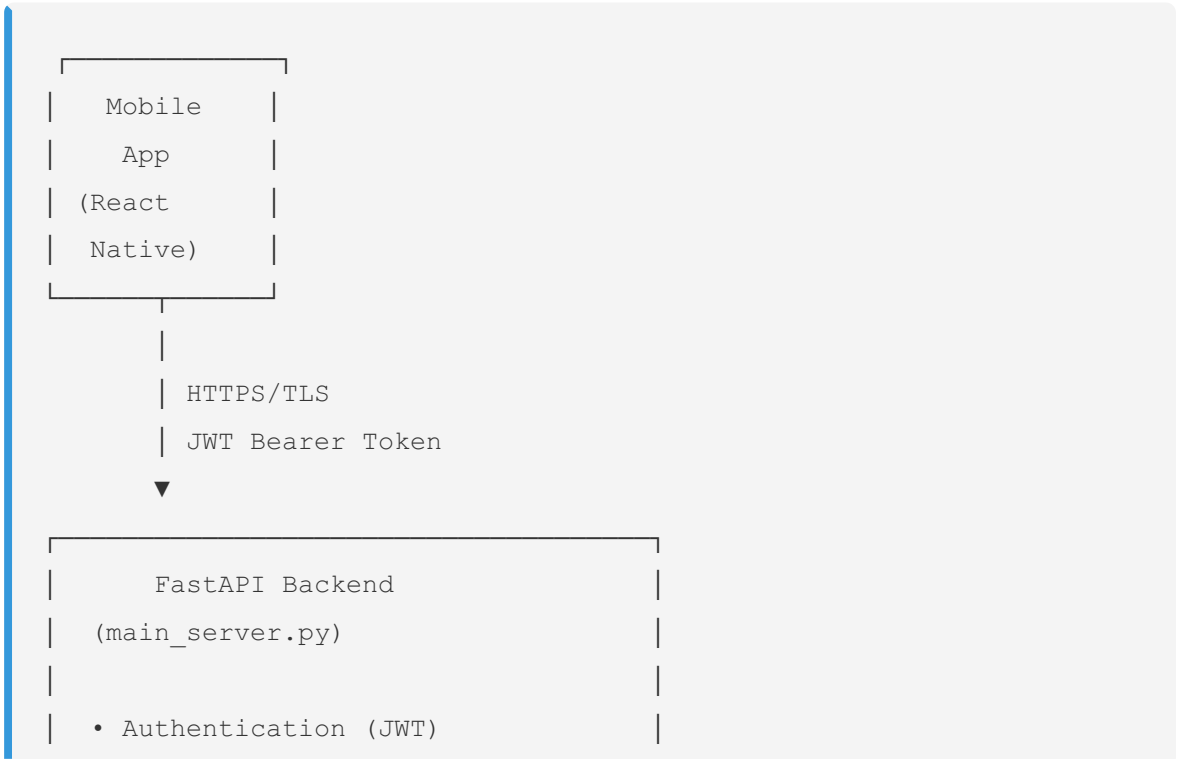
RichesReach System Architecture

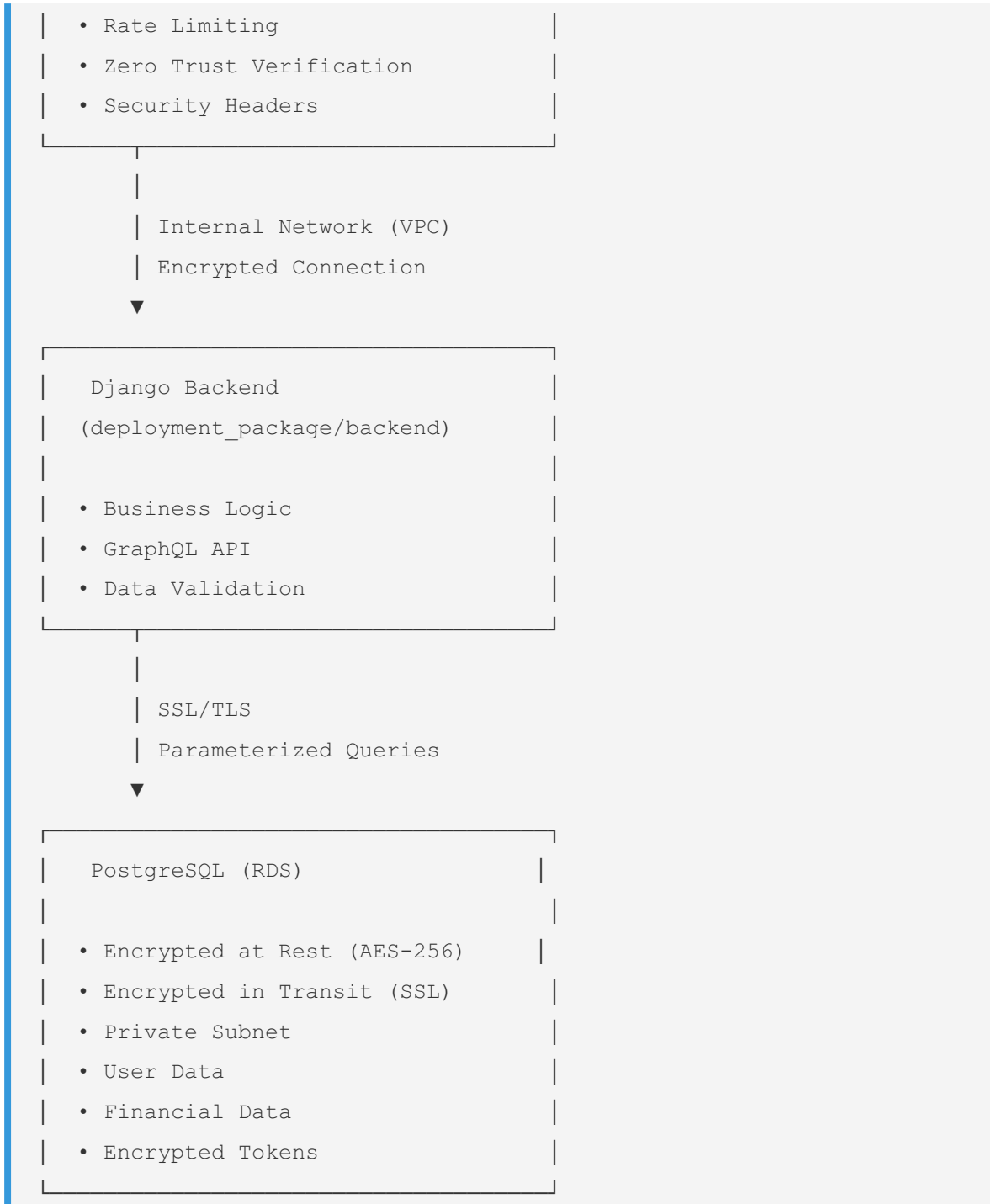
Version: 1.0
Last Updated: 2026
Purpose: Security questionnaire compliance, architecture documentation

Overview

This document describes the flow of sensitive data through the RichesReach system, including user authentication, financial data, and third-party integrations.

High-Level Data Flow




• Rate Limiting
• Zero Trust Verification
• Security Headers
Internal Network (VPC)
Encrypted Connection
Django Backend
(deployment_package/backend)
• Business Logic
• GraphQL API
• Data Validation
SSL/TLS
Parameterized Queries
PostgreSQL (RDS)
• Encrypted at Rest (AES-256)
• Encrypted in Transit (SSL)
• Private Subnet
• User Data
• Financial Data
• Encrypted Tokens

Detailed Data Flows

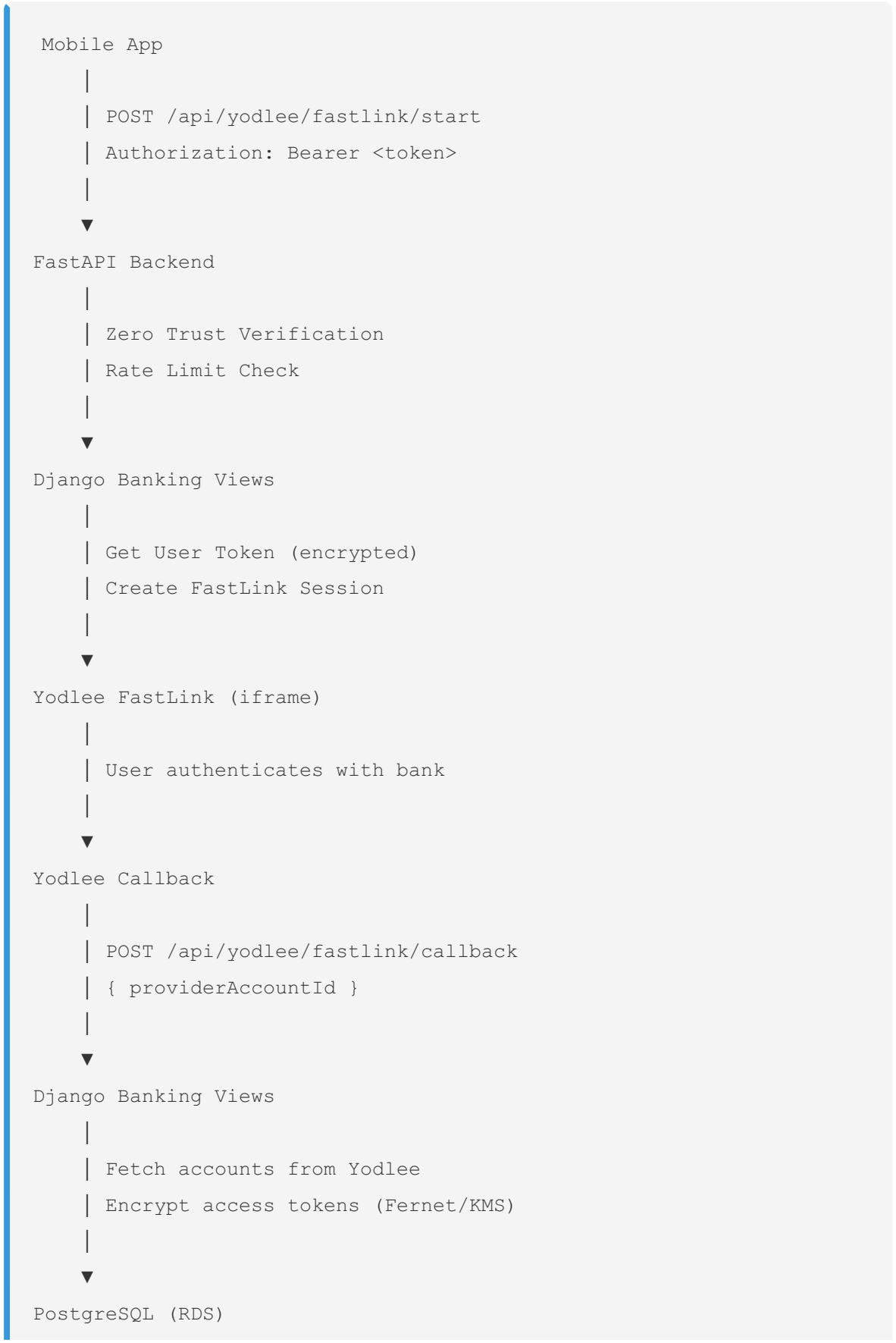
1. User Authentication Flow

```
 Mobile App
     │
     │ POST /api/auth/login
     │ { email, password }
     │
     ▼
FastAPI Backend
     │
     │ Rate Limit Check (IP + username)
     │
     ▼
Django Authentication
     │
     │ Password Hash Verification (PBKDF2)
     │
     ▼
JWT Token Generation
     │
     │ Token: { user_id, email, exp, iat }
     │
     ▼
Mobile App
     │
     │ Store token securely (Keychain/Keystore)
     │
     │ All subsequent requests:
     │ Authorization: Bearer <token>
```

Security Controls: - ✅ Password hashed with PBKDF2 - ✅ Rate limiting (5 attempts/minute) - ✅ Account lockout after failed attempts - ✅ JWT tokens with expiration - ✅ HTTPS/TLS for all communication

---

2. Banking Data Flow (Yodlee Integration)



```
    |
    | Store: BankProviderAccount
    |   - access_token_enc (encrypted)
    |   - refresh_token_enc (encrypted)
    |
    | Store: BankAccount
    |   - Account details (normalized)
    |   - Balance information
```

Security Controls: - ✅ Tokens encrypted at rest (Fernet/KMS) - ✅ HTTPS for all Yodlee communication - ✅ Rate limiting on banking endpoints - ✅ Zero Trust verification - ✅ Database encryption (AES-256)

---

3. Financial Data Access Flow

```
 Mobile App
    |
    | GraphQL Query
    | Authorization: Bearer <token>
    |
    ▼
FastAPI Backend
    |
    | JWT Token Validation
    | Zero Trust Verification
    |
    ▼
Django GraphQL
    |
    | User Authorization Check
    | Query Execution
    |
    ▼
PostgreSQL (RDS)
    |
    | Parameterized Queries Only
```

```
    | User-scoped Data Access
    |
    ▼
Django GraphQL
    |
    | Data Transformation
    | Response Formatting
    |
    ▼
Mobile App
    |
    | Display Financial Data
```

Security Controls: - ✅ JWT token validation - ✅ User-scoped queries (no cross-user access) - ✅ Parameterized SQL queries - ✅ HTTPS/TLS encryption - ✅ Zero Trust verification

---

4. Market Data Flow

```
 Mobile App
    |
    | GET /api/market/quotes?symbol=AAPL
    | Authorization: Bearer <token>
    |
    ▼
FastAPI Backend
    |
    | Rate Limit Check
    |
    ▼
Market Data Service
    |
    | Check Cache (Redis)
    | If miss: Fetch from external API
    |
    ▼
```

```
External APIs (Polygon, Finnhub, etc.)
    │
    │ API Key Authentication
    │ Rate Limiting
    │
    ▼
Market Data Service
    │
    │ Cache Response (Redis)
    │
    ▼
Mobile App
    │
    │ Display Market Data
```

Security Controls: - ✅ API keys stored in Secrets Manager - ✅ Rate limiting on external APIs - ✅ Caching to reduce API calls - ✅ HTTPS for all external communication

---

Data Storage Locations

PostgreSQL (RDS)

- **Location:** AWS RDS (Private Subnet)
- **Encryption:** AES-256 at rest, SSL in transit
- **Data Stored:**
- User accounts (hashed passwords)
- Financial data (bank accounts, transactions)
- Encrypted tokens (Yodlee, Alpaca)
- Portfolio data
- Security events

Redis (ElastiCache)

- **Location:** AWS ElastiCache (Private Subnet)

- **Encryption:** At rest and in transit
- **Data Stored:**
- Session tokens (temporary)
- Rate limit counters
- Cache data (market data, etc.)
- No PII stored

AWS Secrets Manager

- **Location:** AWS (encrypted)
- **Data Stored:**
- API keys (Yodlee, market data providers)
- Database passwords
- Encryption keys (KMS)

Third-Party Integrations

Yodlee (Banking)

- **Purpose:** Bank account linking, transaction sync
- **Data Flow:** Mobile → Backend → Yodlee API
- **Security:**
- OAuth 2.0 authentication
- HTTPS/TLS
- Tokens encrypted before storage
- Webhook signature verification

Alpaca (Trading)

- **Purpose:** Brokerage account management
- **Data Flow:** Mobile → Backend → Alpaca API
- **Security:**
- OAuth 2.0 authentication

- HTTPS/TLS
- API keys in Secrets Manager

Market Data Providers

- **Purpose:** Real-time market data
- **Data Flow:** Backend → External APIs
- **Security:**
- API key authentication
- Rate limiting
- HTTPS/TLS

Security Boundaries

Network Security

- ✅ VPC with private subnets
- ✅ Security groups (least privilege)
- ✅ No public database access
- ✅ WAF (Web Application Firewall) - planned

Application Security

- ✅ Zero Trust architecture
- ✅ JWT token authentication
- ✅ Rate limiting
- ✅ Input validation
- ✅ SQL injection protection

Data Security

- ✅ Encryption at rest (AES-256)
- ✅ Encryption in transit (TLS 1.2+)

- ✅ Token encryption (Fernet/KMS)
- ✅ Password hashing (PBKDF2)

Compliance Mapping

SOC 2

- ✅ Access controls documented
- ✅ Encryption documented
- ✅ Data flow documented
- ✅ Monitoring documented

GDPR

- ✅ Data flow documented
- ✅ Third-party processors identified
- ✅ Data retention policies

PCI-DSS (if applicable)

- ✅ Encryption at rest and in transit
- ✅ Access controls
- ✅ Network segmentation

Diagram Legend

```
┌──────────┐   = System/Component
    │          = Data Flow
    ▼          = Direction
[Encrypted]    = Encryption Applied
HTTPS/TLS    = Secure Communication
```

Last Updated: 2026-01-XX
Version: 1.0
Owner: Security Team